UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

JAVA PARA APLICACIONES DE INGENIERÍA

Darwin Omar Alulema Flores

CIENCIAS DE LA COMPUTACIÓN

JAVA PARA APLICACIONES DE INGENIERÍA

DARWIN OMAR ALULEMA FLORES

Java para aplicaciones de ingenieria
Ing. Darwin Omar Alulema Flores, Mgs.
Primera edición electrónica. Diciembre de 2014
ISBN: 978-9978-301-46-3
Par revisor: Luis Alberto Orozco Brito, Mgs.

Universidad de las Fuerzas Armadas - ESPE
Grab. Roque Moreira Cedeño
Rector
Crnl. Francisco Armendáriz Saénz
Vicerrector Académico General
Crnl. Ricardo Urbina
Vicerrector de Investigación

Publicación autorizada por:
Comisión Editorial de la Universidad de las Fuerzas Armadas - ESPE

Edición y producción
David Andrade Aguirre

Diseño
Pablo Zavala A.

Universidad de las Fuerzas Armadas - ESPE
Av. General Rumiñahui s/n, Sangolquí, Ecuador.
htpp//www.espe.edu.ec

DEDICATORIA

A Dios, por permitirme llegar donde estoy.

A mis padres, Carlos y Marcia, por ser el apoyo incondicional en mi vida.

INDICE

INTRODUCCION

Este libro, está enfocado hacia el aprendizaje del lenguaje Java y su aplicación principal en la ingeniería electrónica, ilustra un amplio espectro de posibilidades que tiene el lenguaje Java para el desarrollo de proyectos que incorporan Hardware y software. Dentro de su estructura, se revisa el origen de Java, sentencias de control, tipos de datos, programación orienta a objetos, diseño de interfaces graficas, I/O en ficheros, hilos, manejo de puertos, código nativo Java, modelos de comunicaciones (TCP y UDP), bases de datos, aplicaciones web y programación de Dispositivos Móviles en J2ME, además se presentan una serie de proyectos en diversas áreas y con distintos grados de dificultad, como objetivo final de adquirir destreza en el desarrollo de aplicaciones que incorporen la gestión de dispositvos electrónicos por medio de puertos del PC, se incluyen tambien anexos, en los que se da una breve descripción de UML, BlueJ y Netbeans.

CAPITULO I

QUÉ ES JAVA?

Java fue desarrollado en Sun Microsystems, en 1991, por Patrick Naughton y James Gosling, como un lenguaje para propósitos especiales. Este lenguaje se utilizaría para programar dispositivos electrodomésticos, por lo que el lenguaje tenía que ser portátil; es decir, debía poder ejecutarse en muchos procesadores diferentes. Este lenguaje nace con la idea de utilizarse para programar dispositivos electrodomésticos tales como decodificadores de televisión, por lo que debía cumplir ciertas características, tales como:

- Ser portátil.
- Poder ejecutarse en muchos procesadores diferentes.
- Crear programas pequeños y eficientes a causa de los limitados recursos de un dispositivo electrodoméstico.
- Fácil de programar.

Aunque el lenguaje de programación Java para electrodomésticos no se convirtió en un gran éxito comercial, hoy se utiliza Java para programar muchos dispositivos de mano, como teléfonos celulares y PDAs.

Con el incremento del uso de Internet, fue claro que los beneficios que podía brindar Java eran idóneos para la programación de páginas web. Programas Java pequeños y eficientes llamados applets podían descargarse rápidamente de Internet. Estos applets otorgan al programador web mucha más flexibilidad y muchos más elementos gráficos que el HTML. La portabilidad del lenguaje permitió el desarrollo de soluciones mediante el uso de Java para diferentes tipos de computadoras conectadas a Internet. La primera etapa en el uso del lenguaje se originó con el lanzamiento del explorador Netscape en 1996. Esta versión del explorador Netscape y todas las versiones subsecuentes, así como las del explorador Microsoft Internet Explorer, están habilitadas para Java.

A medida que los programadores creaban más grandes y complejos applets Java, el tiempo requerido para descargar estos applets se incrementaba sustancialmente. Los applets Java fueron rápidamente reemplazados por otros lenguajes de programación como JavaScript y Flash. Éstos eran más fáciles de utilizar por personas no programadoras tales como los artistas gráficos.

Aunque los applets Java han sido reemplazados y los dispositivos electrodomésticos utilizan otros lenguajes, Java no se extinguió como lenguaje de programación. De hecho, se ha convertido en el lenguaje más popular para la creación de aplicaciones distribuidas que incorporan acceso a datos de muchas computadoras en redes diferentes.

Java es un lenguaje dinámico y en evolución. La cantidad de programas pre-escritos que vienen con el API de Java se duplican con cada nueva versión. Los autores de Java, James Gosling y Patrick Naughton, describieron de la mejor manera estas características únicas en su definición del lenguaje Java como "un lenguaje simple, orientado al objeto, conocedor de la red, interpretado, robusto, seguro, neutral en cuanto a la arquitectura, portátil, de alto desempeño, de subprocesos múltiples y dinámico".

CARACTERISTICAS DE JAVA

Java desde sus orígenes se caracterizó por ser:

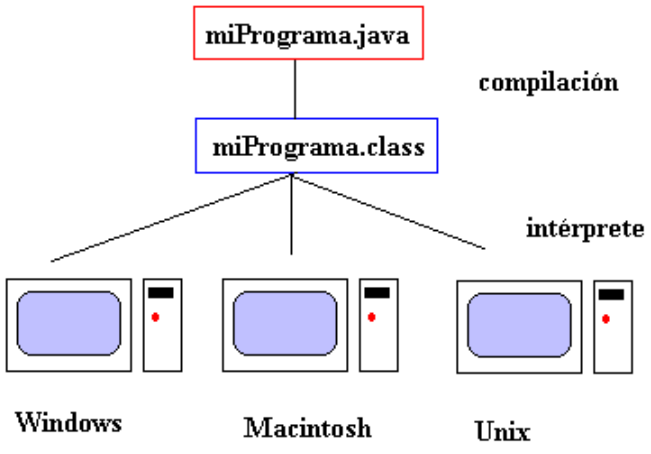
SIMPLE: Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos.

ORIENTADO A OBJETOS: Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje

DISTRIBUIDO: Java se diseño con extensas capacidades de interconexión TCP/IP.

ROBUSTO: Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria.

ARQUITECTURA NEUTRAL: Cualquier máquina que tenga el sistema de ejecución (*run-time*) (jre) puede ejecutar un programa en Java, sin importar la máquina en que ha sido generado.

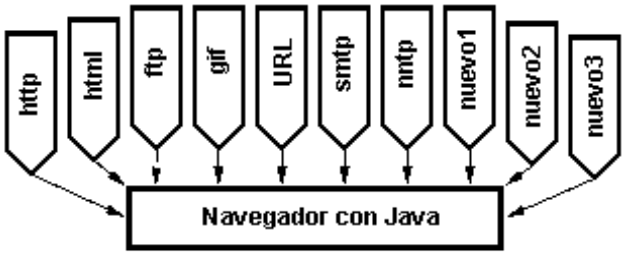


INTERPRETADO: Java es tanto un lenguaje compilado (código fuente compilado a código de bytes) como un lenguaje interpretado (código de bytes interpretado por la JVM a código nativo). El intérprete de Java (run-time) puede ejecutar directamente el código. Java es más lento que otros lenguajes de programación, como C++, ya que debe ser interpretado y no ejecutado como sucede en cualquier programa tradicional

MULTITHREADED: Al ser multithreaded (multihilo), Java permite muchas actividades simultáneas en un programa. El beneficio consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. Aunque el comportamiento en tiempo real está limitado a las capacidades del sistema operativo subyacente (Unix, Windows, etc.), aún supera a los entornos de flujo único de programa (single-threaded) tanto en facilidad de desarrollo como en rendimiento.

DINAMICO: Java no intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución.

Monolito: cada pieza de código se compacta dentro del código del navegador



Sistema Federado: el navegador es un coordinador de piezas, y cada pieza es responsable de una función. Las piezas se pueden añadir dinámicamente a través de la red

Java es utilizado para crear programas autónomos que se ejecutan en computadoras de cualquier tipo. Un ejemplo puede apreciarse en el uso de las aplicaciones de comercio electrónico para permitir la compra y venta de bienes y servicios por Internet. Estos programas utilizan librerías Enterprise Java Beans, Java Beans, y Remote Method Invocation (RMI) para soportar estas aplicaciones de negocios.

¿Por qué se eligió Java como nombre para el lenguaje? El nombre original era *7 (estrella siete). Cuando los desarrolladores no aceptaron este nombre fácilmente, se eligió el nombre OAK. Una vez que se descubrió que existía otro lenguaje llamado OAK, se eligió el nombre Java.

LA MAQUINA VIRTUAL DE JAVA

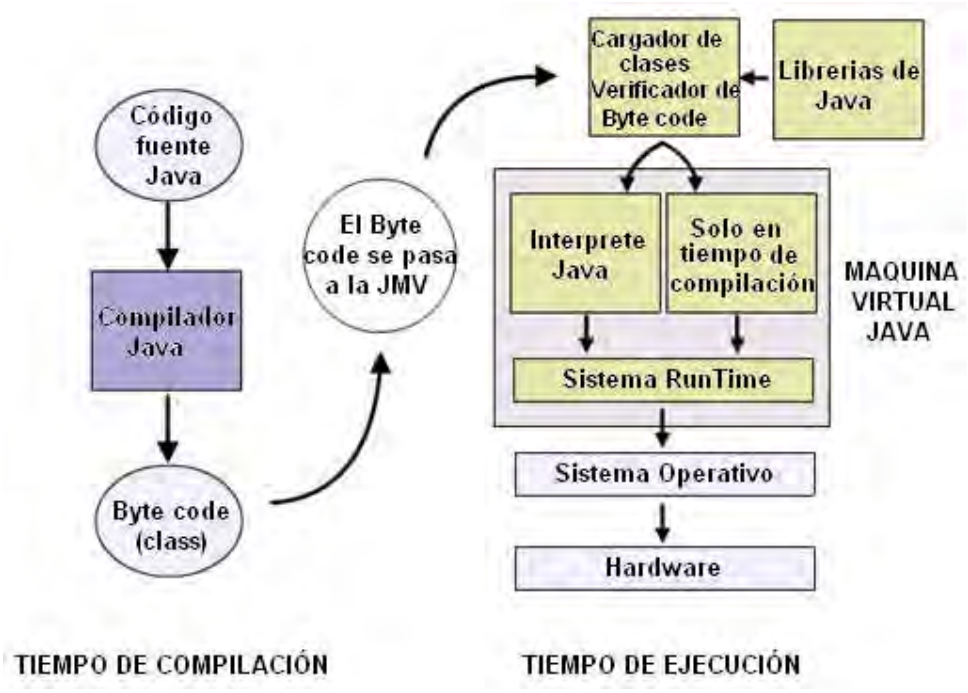
Cuando se creaban programas para controlar pequeños dispositivos de mano, estos tenían en cuenta las limitaciones de potencia y memoria de tales dispositivos. Además, el lenguaje no se podía limitar a un único tipo de unidad de procesamiento central (CPU) puesto que los fabricantes podían elegir diferentes CPUs. Esto condujo al diseño de un lenguaje portátil que podía ejecutarse en cualquier plataforma. El lenguaje podía hacer esto generando un código intermedio para una computadora hipotética llamado máquina virtual.

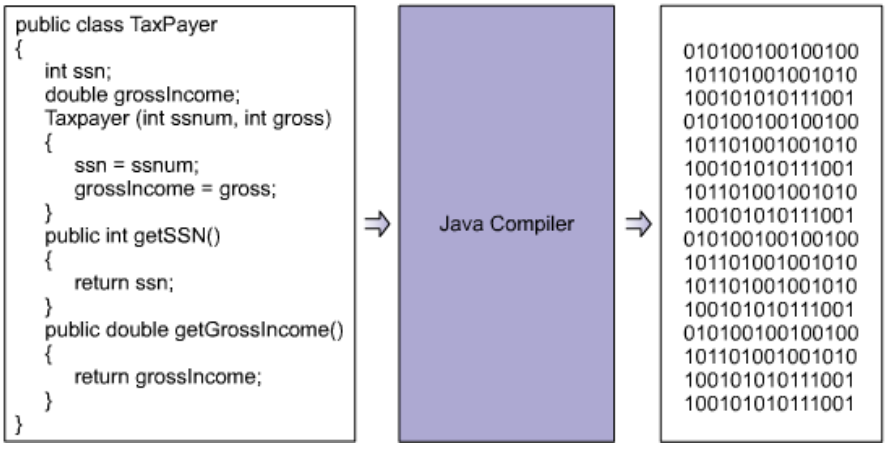
Las máquinas virtuales son programas de software que manejan la comunicación entre las aplicaciones, el hardware y el sistema operativo subyacente. Los programas generan un código que es traducido luego por la máquina virtual a código comprendido por el hardware y el sistema operativo específico.

Antes de la introducción de las máquinas virtuales, un programa tenía que ser re-escrito para cada tipo específico de CPU. Por ejemplo, un programa escrito en Basic o C++ tenía que re-escribirse y re-compilarse para poder ser ejecutado en una Macintosh. No sólo tenía que re-escribirse el código, sino también hacer un mantenimiento del código en computadoras diferentes y para diferentes versiones del hardware de computadoras y de los sistemas operativos. La máquina virtual cambió todo eso. Los programas ahora pueden ser escritos en el lenguaje de la máquina virtual y ejecutarlo en cualquier computadora que tenga una máquina virtual. El programa es leído e interpretado por esa máquina virtual, que traduce el código al lenguaje del hardware de la computadora.

La Máquina Virtual Java [Java Virtual Machine] (JVM) es un programa que se ejecuta en todas las computadoras. La JVM crea una simulación del software de una CPU y de la memoria y maneja toda la comunicación entre el programa y el sistema operativo y hardware subyacentes. En otras palabras, el programa piensa que la computadora en la cual se está ejecutando es la JVM. La carga de comprender y escribir código para plataformas de hardware específicas pasa del programador de la aplicación a la JVM.

Un programador escribe el código fuente Java y compila el código utilizando un compilador Java. El compilador Java traduce el programa a código de bytes [bytecode], que será comprendido por la JVM, no a código nativo que es comprendido por una computadora específica. La JVM toma el código de bytes y lo traduce a código binario (código nativo o código de procesador) para la CPU específica que se utiliza para ejecutar el programa. La JVM es un intérprete para el código de bytes, no un compilador para el código de bytes. El intérprete Java específico ejecuta las instrucciones que se guardan en los archivos cuya extensión es **.class**.





EDICIONES JAVA

Java es un conjunto de tecnologías que abarca a todos los ámbitos de la computación con dos elementos en común:

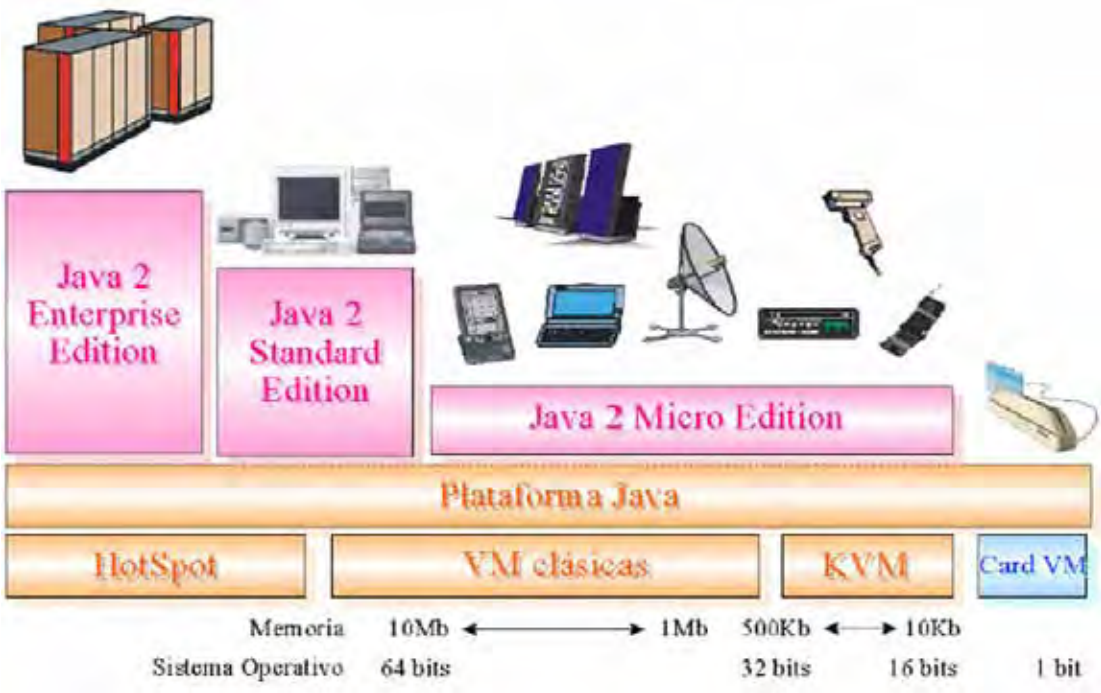
- El código fuente en lenguaje Java es compilado a código intermedio interpretado por una Java Virtual Machine (JVM), por lo que el código ya compilado es independiente de la plataforma.
- Todas las tecnologías comparten un conjunto más o menos amplio de APIs básicas del lenguaje, agrupadas principalmente en los paquetes java.lang y java.io.

Java se divide en 3 ediciones distintas:

JAVA STANDARD EDITION (J2SE): Esta edición de Java es la que recoge la iniciativa original del lenguaje Java. Inspirado inicialmente en C++, pero con componentes de alto nivel, como soporte nativo y recolector de basura. Código independiente de la plataforma, precompilado intermedio y ejecutado en el cliente por una JVM (Java Virtual Machine).

JAVA ENTERPRISE EDITION (J2EE): Esta versión está orientada al entorno empresarial. Ha sido orientada especialmente al desarrollo de servicios web, autenticación, etc. Está pensado para no ser ejecutado en un equipo, sino para ejecutarse sobre una red de ordenadores de manera distribuida y remota.

JAVA MICRO EDITION (J2ME): Esta versión de Java está enfocada a la aplicación en dispositivos electrónicos con capacidades computacionales y gráficas muy reducidas, tales como teléfonos móviles, PDAs. Esta edición tiene unos componentes básicos que la diferencian de las otras versiones, como el uso de una máquina virtual denominada KVM (Kilo Virtual Machine).



EL JDK (JAVA DEVELOPMENT KIT)

El JDK, es el conjunto de herramientas de desarrollo que proporciona Sun Microsystems a la comunidad de programadores gratuitamente, desde la página: http://java.sun.com/javase/downloads/index.jsp

El JDK consta de una serie de aplicaciones y componentes, para realizar cada una de las tareas de las que es capaz de encargarse. El entorno básico del JDK de Java que proporciona Sun está formado por herramientas en modo texto, entre las cuales se tiene:

- *java*, intérprete que ejecuta programas en byte-code.
- *javac*, compilador de Java que convierte el código fuente en byte-code.
- *javah*, crea ficheros de cabecera para implementar métodos para cualquier clase.
- *javap*, es un descompilador de byte-code a código fuente Java.
- *javadoc*, es un generador automático de documentos HTML a partir del código fuente Java.

Javac.exe: Se utiliza para compilar archivos de código fuente Java (*.java), en archivos de clases Java ejecutables (*.class). Se crea un archivo de clase para cada clase definida en un archivo fuente.

```
javac ArchivoACompilar.java
```

Java.exe: El comando java es un interfaz simple a base de línea de comandos para acceder a la Máquina Virtual Java. Es el intérprete de Java que ejecuta byte-codes creados por javac(*.class), el compilador de Java. Este comando ejecuta el método main() contenido dentro del programa .

```
java ClaseAEjecutar
```

ELEMENTOS DE UN PROGRAMA JAVA SIMPLE

El programa de muestra describe un objeto llamado “Hola". Este objeto contiene algunos datos (un nombre, un número y un mensaje) que se imprimirán en la línea de comandos (no en la impresora).

Paso 1: cree un código fuente utilizando un editor. El código fuente se almacena en un archivo .java y se debe guardar en la carpeta Bin del jdk que se encuentre instalado en la maquina.

```
public class Hola
{
    public static void main (String [] args)
    {
        String nombre = "Alumno";
        int numero    = 1;

        System.out.print("Hola ");
        System.out.println(nombre);
        System.out.println("Tu numero es: "+numero);
    }
}
```

Cada archivo fuente Java comienza definiendo una clase (o más). Cada clase tiene un nombre único, y el código fuente para una clase particular debe almacenarse en un archivo fuente que tiene el mismo nombre que la clase. Este ejemplo ha definido una clase llamada Hola. Se la guarda en un archivo llamado Hola.java. Por convención, el nombre de una clase comienza con una letra mayúscula.

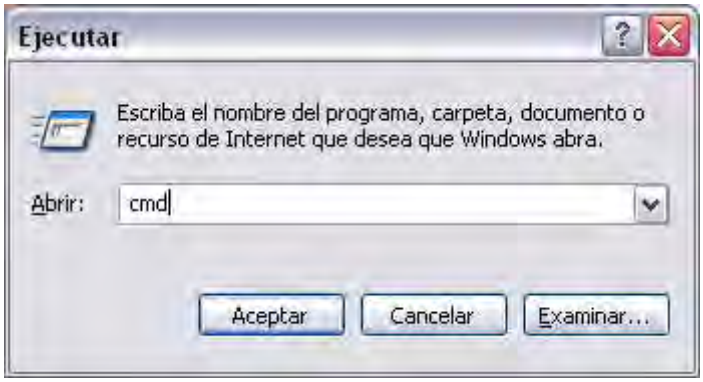
Hay tres nombres de clase en el programa Hola, los cuales son String, System y Hola. La palabra clave public, ubicada en frente de la definición de clase, significa que el acceso a los objetos de esta clase es ilimitado. La llave de apertura "{" marca el comienzo de la definición de clase y la llave de cierre "}" marca el final de la definición de clase. El identificador "nombre" se utiliza para almacenar el nombre del alumno, en el programa. El método System.out.printIn() concatena e imprime la cadena literal del mensaje y agrega un salto de línea. El símbolo punto (.) tiene un significado muy especial en el lenguaje Java; se denomina operador punto. Le permite al programador acceder a métodos u objetos de otros objetos. En el programa Hola, los dos puntos de la expresión System.out.println le indican al compilador Java que es necesario utilizar el método println en el objeto out que se encontró en la clase System.

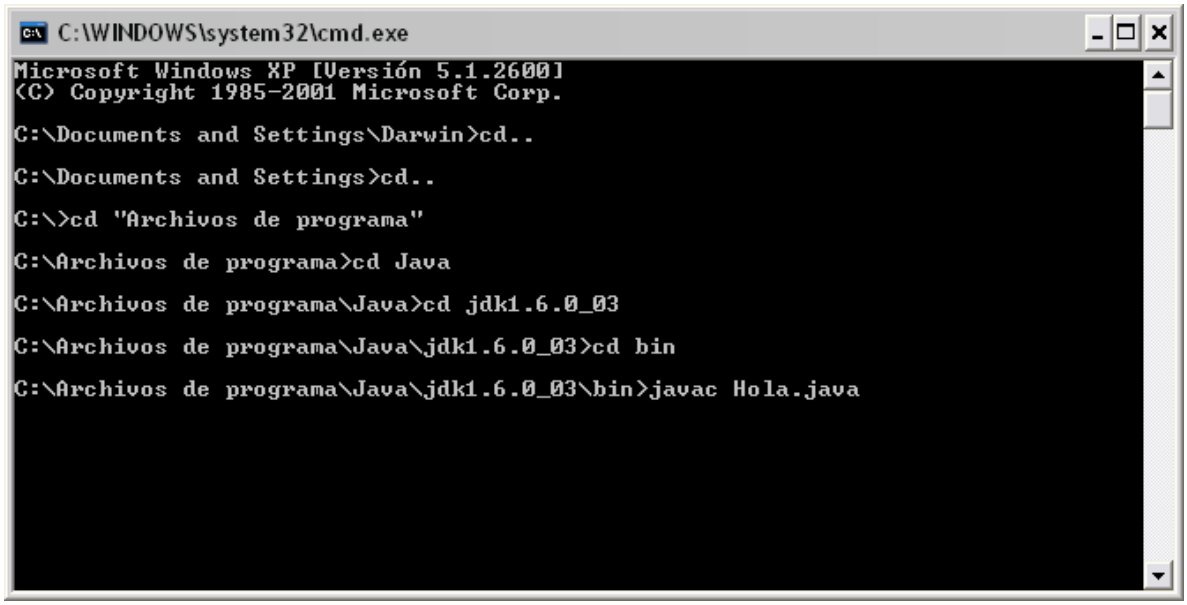
Las clases contienen tanto datos como métodos. Los datos pueden ser números, partes de texto u otros objetos. Los métodos son partes de código que le indican al objeto qué hacer con los datos del objeto. La clase Hola tiene un método llamado main. El método principal [main] de un programa Java tiene una única definición. Es el punto donde la JVM comienza a leer y ejecutar el programa. Las propiedades únicas de cada método incluyen su nombre y los datos que se proporcionan como entrada al método. Sólo hay dos formas de escribir el método principal, cualquiera de las formas es correcta.

```
public static void main (String[] args)

public static void main (String args[])
```

Paso 2: compile el código fuente para crear el código de bytes utilizando el comando javac. El código de bytes se almacena en un archivo .class. Introduzca javac Hola.java. Este código creará un archivo con la extensión. Class que contiene el código de bytes con el mismo nombre que el archivo de código fuente.

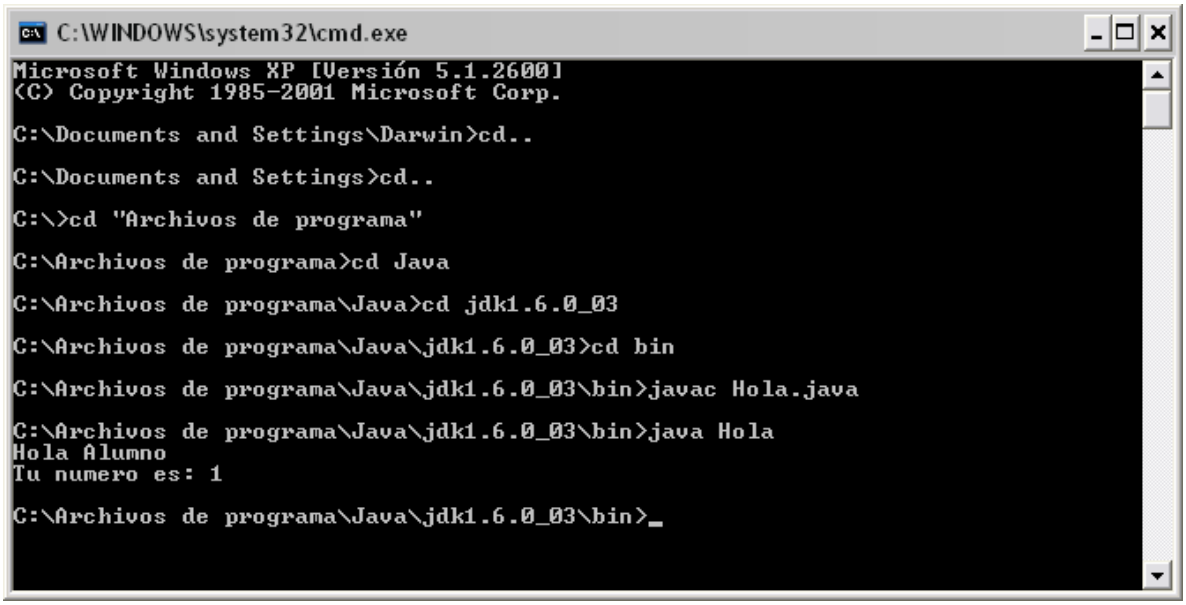
Para poder compilar el programa, se debe ejecutar previamente el *cmd* (*Windows Command Prompt*) desde donde se debe acceder a la ruta donde se encuentra la carpeta bin del jdk que se encuentre instalado en la máquina. En esa ruta se debe ejecutar el comnado javac para poder compilar el programa.





NOTA: Recuerde que Java es un programa sensible al uso de minúsculas y mayúsculas. Verifique que el archivo .class haya sido creado dentro de la carpeta bin del jdk.

Paso 3: ejecute el programa. Para ejecutar el programa Hola, escriba el comando java Hola, con lo que se mostrarán los resultados del programa.



ERRORES EN LA PROGRAMACIÓN

Hay tres tipos de errores de programación:

- Errores del compilador. Los errores del compilador son detectados por el compilador e impiden que éste cree el archivo .class a partir del código fuente.
- Errores de tiempo de ejecución. Los errores de tiempo de ejecución tienen lugar cuando se ejecuta el programa, después de que todos los errores del compilador han sido corregidos.
- Errores lógicos. Los errores lógicos no son detectados por el compilador. El programa funciona, pero no produce el resultado deseado. Por ejemplo, si un programa está diseñado para sumar dos números, pero el programador sin intención utiliza un símbolo '-' en lugar del símbolo '+', se crea un error lógico.

Todos los programadores encuentran algunos errores comunes en el código que han creado:

- Palabras clave o nombres de clase deletreados incorrectamente Identificadores con referencias inconsistentes
- Olvido de cerrar llaves {}, corchetes [], o paréntesis ( )
- Falta del operador punto (.)
- Falta del punto y coma (;)

Las palabras reservadas se pueden clasificar en las siguientes categorías:

- Tipos de datos: **boolean, float, double, int, char**
- Sentencias condicionales: **if, else, switch**
- Sentencias iterativas: **for, do, while, continue**
- Tratamiento de las excepciones: **try, catch, finally, throw**
- Estructura de datos: **class, interface, implements, extends**
- Modificadores y control de acceso: **public, private, protected, transient**
- Otras: **super**, **null, this**.

La mayoría de estos errores son ocasionados por no seguir las reglas de sintaxis del lenguaje. El compilador javac identifica estos errores y muestra mensajes de diagnóstico para ayudar a los programadores a encontrarlos y corregirlos. La mejor manera de reducir la cantidad y frecuencia de estos errores es aprender cuidadosamente los cinco elementos de un lenguaje de programación:

- Vocabulario: es el conjunto de palabras clave utilizadas dentro del lenguaje.
- Puntuación: son los símbolos utilizados por el lenguaje.
- Identificadores: son los nombres utilizados para hacer referencia a los datos almacenados en la memoria de la computadora.
- Operadores: son los comandos o símbolos utilizados para procesar datos. Ejemplos de símbolos para una actividad aritmética o para probar datos son +, *, %, and, or, not, !, >, <.
- Sintaxis: es la gramática o reglas de uso para los elementos anteriores.

Existen otras técnicas que ayudan a reducir la cantidad de errores de compilador en el programa:

- Utilizar un Entorno de Desarrollo Integrado (IDE) para escribir el código Java.
- Verificar la ortografía de todas las palabras clave.
- Colocar una sangría en el código fuente de unos pocos espacios (de dos a cuatro) antes de cada llave.
- Colocar sangría al código hará que las llaves sean más fáciles de encontrar y hará al código más legible.

ACTIVIDADES DE REELABORACION

ACTIVIDADES PRÁCTICAS

1.- Basándose en la clase Hola modifique su código de tal manera que se imprima en pantalla su nombre y apellido en una línea y en la segunda línea su número de cédula. Considerando que tanto su nombre, apellido y número de cédula deben guardarse en una variable cada una.

ACTIVIDADES TEÓRICAS

1.- ¿La Java Virtual Machine traduce y ejecuta código que se encuentra en archivos de que tipo?

2.- ¿Cuál es la extensión usada en un archivo de código fuente de Java?

3.- ¿Qué comando del JDK se usa para compilar el código fuente de Java a Códigos de Byte?

4.- ¿Qué comando del JDK se usa para ejecutar un programa en Java?

5.- ¿Cuál es el beneficio de la utilización de una maquina Virtual para la ejecución de programas Java?

6.- ¿Cuál método es el punto de inicio cuando se ejecuta un programa en Java?

7.- ¿Cual es la palabra clave que define a una clase en Java?

8.- ¿Cual es el tipo de archivos que ejecuta la Máquina Virtual de Java?

9.- ¿Cuál es la extensión que tienen los archivos que guardan el código en Java?

CAPITULO 2

FUNDAMENTOS DEL LENGUAJE

Cada lenguaje de programación posee elementos propios que lo componen. Casi todos los lenguajes tienen palabras clave, símbolos especiales, nombres para los datos o las operaciones, y reglas de sintaxis para su uso. Algunos lenguajes utilizan elementos similares.

USO DE LLAVES, PUNTO Y COMA, COMAS Y ESPACIOS EN BLANCO

A continuación siguen los símbolos más comúnmente utilizados en Java:

- **Llaves** – Un bloque es un conjunto de enunciados vinculados por medio de llaves { }. Éstos incluyen llaves que sirven para vincular definiciones de clase, definiciones de método y otros enunciados que deberán ejecutarse en grupo. Las llaves que se utilizan para definir una clase o un método deberán ubicarse generalmente debajo del enunciado de definición. La ubicación de las llaves alineadas como llaves de apertura y cierre hace más fácil el encontrar errores.
- **Punto y coma** – Un enunciado consiste en una o más líneas de código terminado, por un punto y coma. Omitir el punto y coma es un error común del compilador.
- **Comas** – Las comas sirven como separadores o delimitadores de datos. Los argumentos de método (datos proporcionados a un método para su uso en el mismo) deben ir separados por comas.
- **Espacio en blanco** – Los espacios en blanco separan palabras clave e identificadores.

IDENTIFICADORES

Los identificadores son etiquetas que los programadores asignan a los datos o direcciones de almacenamiento. Puesto que el compilador y la JVM manejan todos los detalles de la adjudicación de la memoria, el programador sólo necesita proporcionar una etiqueta para acceder a los datos almacenados. Los identificadores también son etiquetas que un programador asigna a nombres de clase y a nombres de método.

Java impone algunas reglas acerca de la creación de identificadores:

- Puede utilizarse cualquier del código ASCII.

- El primer carácter debe ser una letra. Los caracteres subsecuentes pueden ser cualquier carácter.
- Los identificadores no pueden contener los símbolos % (porcentaje) o # (símbolo numeral). Pueden contener $ (signo dólar).
- Generalmente se recomienda no utilizar símbolos especiales tales como $, &, etc.
- Los identificadores no pueden contener espacios.
- Los identificadores son sensibles al uso de mayúsculas y minúsculas.
- Los identificadores no pueden utilizar determinadas palabras clave, conocidas como palabras reservadas.

TIPOS DE VARIABLES

Java tiene tres tipos de variables (según su función):

- VARIABLES DE INSTANCIA, se usan para guardar los atributos de un objeto particular
- VARIABLES DE CLASE son similares a las variables de instancia, con la excepción de que los valores que guardan son los mismos para todos los objetos de una determinada clase.
- VARIABLES LOCALES se utilizan dentro de los métodos.

EJEMPLO No.1

En el código siguiente, ***PI*** es una variable de clase y *radio* es una variable de instancia. *PI* guarda el mismo valor para todos los objetos de la clase ***Circulo***, pero el radio de cada círculo puede ser diferente, ***area*** es una variable local al método ***calcularArea*** en la que se guarda el valor del área de un objeto de la clase ***Circulo***.

```
public class Circulo
{
    static final double PI = 3.1416;
    static double radio = 5;
    static double calcularArea()
    {
        double area = PI*radio*radio;
        return area;
    }
}
```

Las variables pueden ser (según su naturaleza):

- TIPO DE DATO PRIMITIVO, son tipos de datos que se utilizan para almacenar tipos simples de datos. Hay ocho tipos de primitivos (boolean, char, byte, short, int, long, float, y doublé).

TIPOS DE DATOS PRIMITIVOS	
TIPO	**DESCRIPCIÓN**
boolean	Tiene dos valores **true** o **false**.
Char	Los caracteres alfa-numéricos son los mismos que los ASCII. El intervalo de valores va desde 0 hasta 65535 (valores de 16-bits sin signo).
Byte	Tamaño 8 bits. El intervalo de valores va desde -2^7 hasta $2^7 -1$ (-128 a 127)
short	Tamaño 16 bits. El intervalo de valores va desde -2^{15} hasta $2^{15}-1$ (-32768 a 32767)
int	Tamaño 32 bits. El intervalo de valores va desde -2^{31} hasta $2^{31}-1$ (-2147483648 a 2147483647)
long	Tamaño 64 bits. El intervalo de valores va desde -2^{63} hasta $2^{63}-1$ (-9223372036854775808 a 9223372036854775807)
float	Tamaño 32 bits. Números en coma flotante de simple precisión de 1.40239846e–45f a 3.40282347e+38f
oublé	Tamaño 64 bits. Números en coma flotante de doble precisión de 4.94065645841246544e–324d a 1.7976931348623157e+308d

- REFERENCIAS JAVA, son variables que almacenan la direccione de objetos de clases propias de Java. La clase String, System, StringBuffer, Math, y Wrapper,

son ejemplos de clase de núcleo de Java. Algunos programadores consideran a los objetos String como en uno de los tipos Java básicos debido a que gran parte de los datos manipulados son texto.

```
String capitalUSA = “Washington D.C.”;
String nombre = “Andrea”;
```

- ARREGLOS, Un arreglo es una construcción que proporciona almacenaje a una lista de elementos del mismo tipo, ya sea simple o compuesto. En Java los arreglos se declaran utilizando corchetes (*[y]*), tras la declaración del tipo de datos que contendrá el vector.

Por ejemplo, esta sería la declaración de un vector de números enteros (*int*):

```
int vectorNumeros[ ];
```

La asignación de memoria al vector se realiza de forma explícita en algún momento del programa. Para ello o se utiliza el operador *new*:

```
int vectorNumeros [ ]= new int[ 5 ];
```

OPERADORES

Operadores aritméticos binarios de Java

Operador	**Uso**	**Descripción**
+	*op1 + op2*	Suma op1 y op2
-	*op1 – op2*	Resta op2 de op1
***	*op1 * op2*	Multiplica op1 por op2

/	*op1 / op2*	Divide op1 por op2
%	*op1 % op2*	Calcula el resto de dividir op1 entre op2

Los operadores + y – tienen versiones unarias que realizan las siguientes operaciones:

Operador	**Uso**	**Descripción**
+	+op	Convierte op a entero si es un byte, short o char
-	-op	Niega aritméticamente op

Existen dos operadores aritméticos que funcionan como atajo de la combinación de otros: ++ que incrementa su operando en 1, y – que decrementa su operando en 1. Ambos operadores tienen una versión prefija, y otra posfija.

Operador	**Uso**	**Descripción**
++	*op++*	Incrementa op en 1; se evalúa al valor anterior al incremento
++	*++op*	Incrementa op en 1; se evalúa al valor posterior al incremento
--	*op--*	Decrementa op en 1; se evalúa al valor anterior al incremento
--	*--op*	Decrementa op en 1; se evalúa al valor posterior al incremento

Un operador de comparación compara dos valores y determina la relación existente entre ambos.

Operador	**Uso**	**Devuelve verdadero si**
>	op1 > op2	op1 es mayor que op2
>=	op1 >= op2	op1 es mayor o igual que op2
<	op1 < op2	op1 es menor que op2
<=	op1 <= op2	op1 es menor o igual que op2
==	op1 == op2	op1 y op2 son iguales
¡=	op1 ¡= op2	op1 y op2 son distintos

Java soporta cinco operadores condicionales, mostrados en la siguiente tabla

Operador	**Uso**	**Devuelve verdadero si…**
&&	*op1 && op2*	op1 y op2 son ambos verdaderos, condicionalmente evalúa op2
&	*op1 & op2*	op1 y op2 son ambos verdaderos, siempre evalúa op1 y op2

||	*op1 || op2*	op1 o op2 son verdaderos, condicionalmente evalúa op2
|	*op1 | op2*	op1 o op2 son verdaderos, siempre evalúa op1 y op2
¡	*¡ op*	op es falso

El operador de asignación básico es el =, que se utiliza para asignar un valor a otro. Suponga que necesita sumar un número a una variable y almacenar el resultado en la misma variable, como a continuación:

```
i = i + 2;
```

Se puede abreviar esta sentencia con el operador de atajo *+=*, de la siguiente manera*:*

```
i += 2;
```

La siguiente tabla muestra los operadores de atajo de asignación y sus equivalentes largos:

Operador	**Uso**	**Equivalente a**
+=	*op1 += op2*	op1 = op1 + op2
-=	*op1 -= op2*	op1 = op1 – op2
**=*	*op1 *= op2*	op1 = op1 * op2
/=	*op1 /= op2*	op1 = op1 / op2

%=	*op1 %= op2*	op1 = op1 % op2
&=	*op1 &= op2*	op1 = op1 & op2

Cuando en una sentencia aparecen varios operadores el compilador deberá elegir en qué orden aplica los operadores. A esto se le llama *precedencia*.

Los operadores con mayor precedencia son evaluados antes que los operadores con una precedencia relativa menor.

INSTRUCCIONES CONDICIONALES Y DE REPETICION

Las estructuras de control, facilitan que determinadas acciones se realicen varias veces, mientras que una condición se cumpla, y en definitiva, tomar decisiones de qué hacer en función de las condiciones que se den en el programa en un momento dado de su ejecución.

El lenguaje Java soporta las siguientes estructuras de control:

Sentencia	**Clave**
Toma de decisión	if-else, switch-case
Bucle	for, while, do-while

Misceláneo	break, continue, label:, return

INSTRUCCIONES CONDICIONALES Y DE REPETICION

La sentencia if – else

Los enunciados condicionales permiten la ejecución selectiva de porciones del programa de acuerdo al valor de algunas expresiones.

La forma general es la siguiente.

```
int a = 1;
int b = 2;

if(a < b)
{
  System.out.println("a is less than b");
}
else
{
  System.out.println("a is not less than b");
}
```

La sentencia switch

Mediante la sentencia switch se puede seleccionar entre varias sentencias según el valor de cierta expresión.

- **switch**. Comienza la selección. La llave de apertura sigue a esta expresión.
- **case**. La condición es igual a la primera constante. Se requieren los dos puntos (☺ al final de este enunciado.
- **break**. Finaliza la secuencia de acciones y sale de la estructura de control switch.
- **default**. El conjunto de acciones que se ejecutarán si no se halla una coincidencia entre la expresión y cada constante.

La forma general es la siguiente:

```
int meses = 1;
switch ( meses ){
    case 1: System.out.println( "Enero" ); break;
    case 2: System.out.println( "Febrero" ); break;
    case 3: System.out.println( "Marzo" ); break;
    default: System.out.println( "Mes no valido" ); break;
}
```

Bucle while

El bucle *while* es el bucle básico de iteración. Sirve para realizar una acción sucesivamente mientras se cumpla una determinada condición.

La forma general es la siguiente:

```
int I = 0;
while(I < 10)
{
  System.out.println("Are you finished yet?");
  I++;
}
System.out.println("Done");
```

Bucle do-while

El bucle do-while es similar al bucle while, pero en el bucle while la expresión se evalúa al principio del bucle y en el bucle do-while la evaluación se realiza al final.

```
int i = 0;
do{
  System.out.println("Are you finished yet?");
  i++;
  }while(i < 10)
  System.out.println("Done");
```

Bucle for

El bucle for permite un número de ejecuciones especifico, determinado por un punto de inicio y fin.

```
for(int i = 0; i < 10; i ++)
{
  System.out.println("Are you finished yet?");
}
System.out.println("Done");
```

LOS ARRAYS

Un array es un medio para guardar un conjunto de objetos de la misma clase. Se accede a cada elemento individual del array mediante un número entero denominado índice. 0 es el índice del primer elemento y *n-1* es el índice del último elemento, siendo *n*, la dimensión del array.

Para declarar un array de enteros se escribe:

```
int [ ] numeros;
int numeros [ ];
```

Para crear un array de 4 números enteros se escribe:

```
numeros=new int[4];
```

La declaración y la creación del array se pueden hacer en una misma línea:

```
int [ ] numeros =new int[4];
```

Para inicializar el array de 4 enteros se escribe:

```
numeros[0]=2;
numeros[1]=-4;
numeros[2]=15;
numeros[3]=-25;
```

Se pueden inicializar en un bucle **for** como resultado de alguna operación:

```
for(int i=0; i<4; i++)
{
        numeros[i]=i*i+4;
}
```

No es necesario recordar el número de elementos del array, *length* proporciona la dimensión del array.

```
for(int i=0; i<numeros.length; i++)
{
        numeros[i]=i*i+4;
}
```

Los arrays se pueden declarar, crear e inicializar en una misma línea, del siguiente modo:

```
int[ ] números = {2, -4, 15, -25};
String[ ] nombres = {“Juan”, “José”, “Miguel”, “Antonio”};
```

Para imprimir los elementos de array *nombres* se escribe

```
for(int i=0; i<nombres.length; i++)
{
        System.out.println(nombres[i]);
}
```

Una matriz bidimensional puede tener varias filas, y en cada fila no tiene por qué haber el mismo número de elementos o columnas.

```
double [][] matriz={{1,2,3,4},{5,6},{7,8,9,10,11,12},{13}};
```

Para mostrar los elementos del array bidimensional escribimos el siguiente código

```
 for (int i=0; i < matriz.length; i++)
{
        for (int j=0; j < matriz[i].length; j++)
        {
System.out.print(matriz[i][j]+"\t");
        }
        System.out.println("");
}
```

CLASE MATH

Math es una clase que proporciona operaciones matemáticas más complejas

Math.abs(*num*)	**Math.max(*a*,*b*)**	**Math.min(*a*,*b*)**
Math.asin(num)	**Math.acos(*num*)**	**Math.atan(*num*)**
Math.sin(num)	**Math.cos(*num*)**	**Math.tan(*num*)**
Math.exp(*num*)	**Math.log(*num*)**	**Math.pow(*a*,*b*)**
Math.sqrt(*num*)	**Math.random()**	**Math.round(*num*)**
Math.toDegrees(*num*)	**Math.toRadians(*num*)**	

CLASE STRING

Java posee gran capacidad para el manejo de cadenas dentro de su clase **String**. Un objeto *String* representa una cadena alfanumérica de un valor constante. Los Strings son objetos constantes y por lo tanto muy baratos para el sistema. La mayoría de las

funciones relacionadas con cadenas esperan valores String como argumentos y devuelven valores String.

Funciones Básicas:

- int length();
- char charAt(int oublé);
- String substring(int beginindex);
- String substring(int beginindex,int endindex);
- String concat(String str);
- String replace(char oldchar,char newchar);
- String toLowerCase();
- String toUpperCase();
- int compareTo(String str2);

La clase **String** posee numerosas funciones para transformar valores de otros tipos de datos a su representación como cadena. Todas estas funciones tienen el nombre de *valueOf*, estando el método sobrecargado para todos los tipos de datos básicos.

- String valueOf(boolean b);
- String valueOf(int i);
- String valueOf(long l);
- String valueOf(float f);
- String valueOf(oublé d);
- String valueOf(Object obj);
- String valueOf(char data[]);
- String valueOf(char data[],int offset,int count);

A continuación un ejemplo:

```
String Uno = new String( “Hola Mundo” );
float f = 3.141592;
String PI = Uno.valueOf( f );
String PI = String.valueOf( f ); // Mucho más correcto
```

LECTURA DESDE EL TECLADO

Para facilitar la lectura de teclado se puede conseguir que se lea una línea entera con una sola orden si se utiliza un objeto BufferdReader. El método String readLine() pertenece a BufferReader lee todos los caracteres hasta encontrar un ‘\n’.

Ejemplo No. 2:

```
import java.io.*;

public class Main {

    public static void main(String[] args) {

        System.out.println("Nombre: ");
        String aux="";
        DataInputStream dato1=new DataInputStream(System.in);
        try{
            aux=dato1.readLine();
        }
        catch(Exception e)
        { }
        System.out.println("CI: ");
        int aux2=0;
        try{
            aux2=Integer.parseInt(dato1.readLine());
        }
        catch(Exception e)
        { }
        System.out.println("Ingreso:"+aux+" "+aux2);
    }

}
```

Ejemplo No. 3:

```
import java.io.*;
public class a
{
    public static void main(String [] args)
    {
        BufferedReader dato = new BufferedReader (new InputStreamReader(System.in));
        String aux="";
        try
        {
            System.out.println("Ingrese dato");
            aux=dato.readLine();
        }
        catch(Exception e)
        {}
        System.out.println("Dato:"+aux);

    }
}
```

ACTIVIDADES DE REELABORACION

ACTIVIDADES PRÁCTICAS

1**.-** Basándose en el Ejemplo No. 2 modifique el programa de tal manera que se pueda sumar dos números de cómo flotante (double) y que su resultado se imprima en pantalla?

ACTIVIDADES TEÓRICAS

1.- Indique en la tabla de qué tipo de dato pueden ser las variables necesarias para guardar cada dato.

	boolean	char	byte	short	int	long	float	double
4								
2.5								
-3.6111156511876								
-40000								
'a'								
0								

2.- Escriba el valor de verdad que se produciría al ejecutarse los siguientes casos de operadores de comparación.

<	3 < 8		8 > 3	
>	2 > 4		4 > 2	
==	7 == 8		7 == 7	
<=	5 <= 5		8 <= 6	
>=	7 >= 3		1 >= 2	
!=	5 != 5		4 != 3	

3.- Escriba el valor de verdad que se produciría al ejecutarse los siguientes casos.

4 > 1	
43 >=43	
2 == 3	
2 + 5 == 7	
3 + 8 <= 10	
4 > 1	
3 != 9	
13 != 13	
-4 != 4	
2 +5 *3 == 21	

4.- Escriba el valor de verdad que se produciría al ejecutarse los siguientes casos.

Asuma que la variable z = 12

z > 3 && z > 6	
z > 3 && z < 20	
z > 3 && z < 0	

Asuma que la variable z = 2

z > 3 && z >6	
z > 0 && z >1	
z < 7 && z == 2	

Asuma que la variable z = 5

z > 6 || z == 5	
z < 3 || z > 4	
z >= 0 || z < 2	

5.- Indique los valores que contienen las variables x e y en cada momento durante la ejecución del programa.

int x = 4;
int y = 0;
y = ++x;
 	
int x = 4;
int y = 0;
y = x++;	

x = 45;	
x =45; x *= 2;	
x = 70; x /= 5;	
x = 90; x %= 8;	

6.- Indique los valores que contienen las variables x e y en cada momento durante la ejecución del programa.

public static void main(String[] args) {
int x;
int a = 6;
int b = 7;
x = a++;
x = ++b;
b += ++a;
x = a++ + b++;
int y =0;
y += x;
--y;
a = --x;
}	

public static void main(String[] args)
{
int a = 4567, b =1237, c,d;
c = a * b / b;
d = a / b * b;
}	

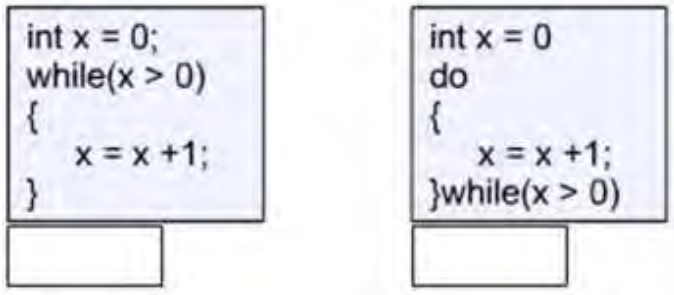
7.- Analice el siguiente código e indique que imprime.

```
int x =5, y =4, z =6, temp;
for(int a = 0; a <2; a++)
{
    if  ( x > y )
    {
        temp = x;
        x = y;
        y = temp;
    }
    if (y > z )
    {
        temp = y;
        y = z;
        z = temp;
    }
}
System.out.println (  x + y + z);
```

8.- Analice el siguiente código e indique que contiene el arreglo de 3 dimensiones.

```
public class Array3D_Example
{
    public static void main (String args [ ])
    {
        int array3D[][][] = new int[5][4][3];
        for (int x = 0; x < array3D.length; x ++)
            for(int y = 0; y < array3D[x].length; y++)
                for (int z = 0; z < array3D[x][y].length; z++)
                {
                        array3D[x][y][z] = 0;
                }
    }
}
```

9.- Indique cuantas veces se ejecutan los siguientes bucles de repetición



10.- Indique el valor que se imprimiría al ejecutarse las siguientes líneas de código

```
int a = 12;
int b = 2;
int c = a * b / b++;
a++;
int d = ++a / b;
System.out.println(c);
System.out.println(d);
```

11.- Verifique si el siguiente código es correcto, si no lo es corrija de tal manera que se cumpla el número de ejecuciones indicado.

```
int y = -1;
for(int x = 0; x > y; x++)
{
    x = x++;
    if(x == 4)
    y = 10;
}
```

6

```
int y = -1;
for(int x = 0; x > y; x++)
{
    x = ++x;
    if(x == 4)
    y = 10;
}
```

3

```
int y = -1;
for(int x = 0; x > y; x++)
{

    if(x == 4)
    y = 10;
}
```

6

12.- Cuál es el elemento en el índice 3 del siguiente array.

```
char theArray[] = {'H','0','A','5','3'};
```

13.- A partir del siguiente código, que valores tiene A, B, C y x, al final de su ejecución.

```
for(int x = 0; x < 10; x++){          //1
        A = A + B;                    //2

        if(A > x && A % 2 == 0){      //3
        x = x + 2;                    //4
            do{
                C = C - A;            //5
                B = A + x;            //6
                x = x + 1;            //7
            }while(x % 3 != 1);       //8
        }
    }
```

A=

B=

C=

x=

14.- Qué imprime el método main?

```
class Prueba{
  static int x = 5;
  public static int f(){
    x = 0;
    return x;
  }
  public static void main(String [] args){
    System.out.println("Resultado = " + f() + ", " + (++x));
  }
}
```

15.- Escriba un programa en el que, se solicite al usuario el ingreso por teclado de tres enteros: a, b y c, que serán los coeficientes de la ecuación de segundo grado: $y = ax^2 + bx + c$, y muestre por pantalla los valores de x que resuelven la ecuación.

16.- Escriba un programa en el que se permita al usuario introducir por teclado una matriz A_{nxm}, de tal manera que permita: Mostrar las diagonales de la matriz A si y sólo si es cuadrada. Crear una nueva matriz At que sea la transpuesta de A y mostrarla por pantalla de forma bidimensional (utilice el carácter \t como tabulador).

$n = 3$
$m = 4$

$$A = \begin{pmatrix} \mathbf{1} & 2 & \mathbf{3} \\ 4 & \mathbf{5} & 6 \\ \mathbf{7} & 8 & 9 \end{pmatrix} \qquad A^t = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

17.- Escriba un programa en el que: Se solicite al usuario que introduzca un valor *double* por teclado. Este valor representará un ángulo medido en grados sexagesimales:α. Transforme este ángulo a radianes: ang_rad = (α * PI)/180. Calcule el seno, coseno y tangente del ángulo transformado y muestre los resultados por pantalla.

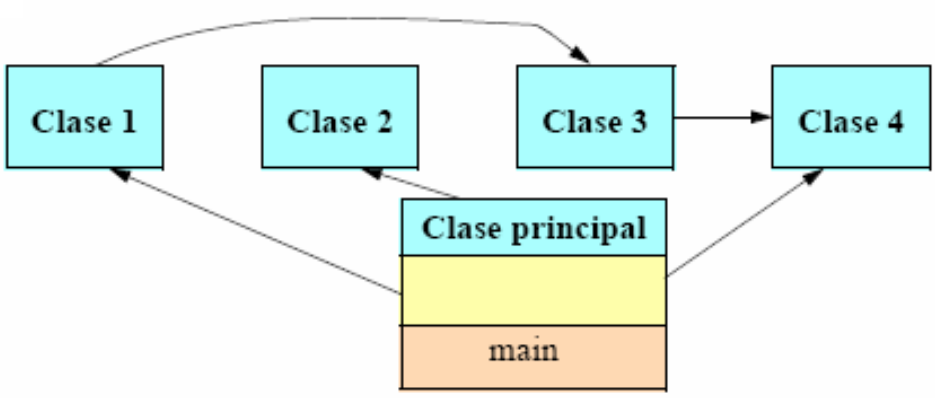
CAPITULO 3

OBJETOS Y CLASES

ESTRUCTURA DE UN PROGRAMA

Un programa java es un conjunto de clases, donde una de ellas es especial en dos aspectos:

- Contiene una operación llamada main
- Habitualmente no se crean objetos de esta clase
- El sistema invoca el método main al ejecutar el programa



Ejemplo de la estructura de un programa:

```
class Mensaje
{
        public static void main (String [] args)
        {
                System.out.println(“MENSAJE”);
        }
}
```

Explicación:

- `public`: el método se puede usar desde fuera
- `static`: el método pertenece a la clase (no a los objetos de la clase)
- `void`: no retorna nada
- `String[] args`: es el argumento, datos que se pasan a la operación
- `System`: es una clase predefinida que representa al computador
- `out`: es un objeto de la clase `System`, predefinido: representa la pantalla
- `println`: método para poner un texto en la pantalla

El elemento básico en Java no es la función, sino un ente denominado **objeto**.

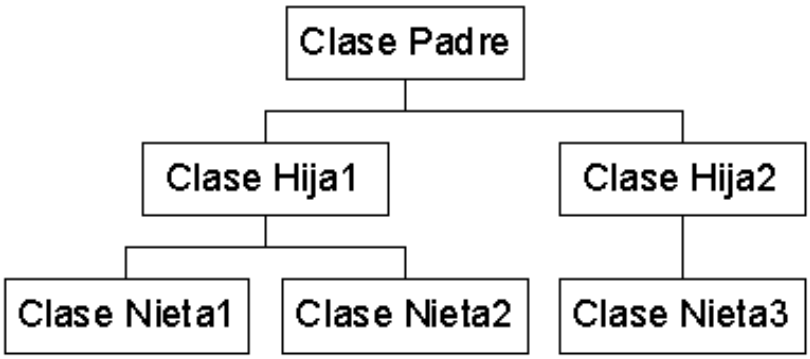
"Un objeto es la representación de un concepto para un programa, y contiene toda la información necesaria para abstraer dicho concepto: los datos que describen su estado y las operaciones que pueden modificar dicho estado, y determinan las capacidades del objeto."

Un objeto es un elemento de programa que se caracteriza por:

- **ATRIBUTOS:** son datos contenidos en el objeto, y que determinan su estado
- **TIEMPO DE VIDA:** La mayoría de los objetos sólo existen durante una parte de la ejecución del programa. Los objetos son creados mediante un mecanismo denominado **instanciación**
- **METODOS:** son acciones con las que se puede solicitar información del objeto, o modificarla.

Existen una serie de principios fundamentales para comprender cómo se modeliza la realidad al crear un programa bajo el paradigma de la orientación a objetos. Estos principios son:

a.) Principio de Abstracción.- Mediante la abstracción la mente humana modeliza la realidad en forma de objetos. Para ello busca parecidos entre la realidad y la posible implementación de objetos del programa que simulen el funcionamiento de los objetos reales.
b.) Principio de Modularidad.- Mediante la modularidad, se propone al programador dividir su aplicación en varios módulos diferentes (ya sea en forma de clases, paquetes o bibliotecas), cada uno de ellos con un sentido propio.
c.) Principio de Jerarquía.- Las clases de un programa se organizan mediante la *jerarquía*. La representación de dicha organización da lugar a los denominados *árboles de herencia*.



Las clases son abstracciones que representan a un conjunto de objetos con un comportamiento e interfaz común. Se puede considerar a una clase como *"un conjunto de cosas que tienen el mismo comportamiento y características".* Una clase define la forma y comportamiento de un objeto. Una clase es una plantilla para un objeto.

Los métodos son las funciones, mediante las que, las clases representan el comportamiento de los objetos.

Cuando se ejecuta un programa en Java, el sistema utiliza definiciones de clase para crear instancias de las clases, que son los objetos reales.

NOTA: Los términos instancia y objeto se utilizan de manera indistinta.
La forma general de una definición de clase es:

```
class Nombre_De_Clase
{
  tipo_de_variable nombre_de_atributo1;
  tipo_de_variable nombre_de_atributo2;

  tipo_devuelto nombre_de_método1( lista_de_argumentos )
        {
        cuerpo_del_método1;
        }
  tipo_devuelto nombre_de_método2( lista_de_argumentos)
        {
        cuerpo_del_método2;
        }
}
```

LOS ATRIBUTOS

Los datos se encapsulan dentro de una clase declarando variables dentro de las llaves de apertura y cierre de la declaración de la clase. Los atributos que se pueden declarar son de tres tipos: tipo primitivo, arreglo o de clase.

LOS MÉTODOS

Los métodos son subrutinas que definen la interfaz de una clase, sus capacidades y comportamiento. Un método ha de tener por nombre cualquier identificador legal distinto de los ya utilizados por los nombres de la clase en que está definido. La forma general de una declaración de método es:

```
tipo_devuelto nombre_de_método( lista-formal-de-argumentos )
{
  cuerpo_del_método;
}
```

Cada vez que se crea una clase se añade otro tipo de dato que se puede utilizar igual que uno de los tipos. Por ello al declarar una nueva variable, se puede utilizar un nombre de clase como tipo. A estas variables se las conoce como *referencias a objeto*.

class MiClase
{ }

Referencia a una instancia de este tipo de clase

MiClase C;

Ejemplo No. 1:

Implemente el siguiente código Java.

```
public class DemoClass
{
    String name =  "Student";
    public String getName() {return name;}
}

class Pay
{
    public double computeNetPay(double hrs, double pr, double wr)
    {
        return (hrs * pr) - (hrs * pr * wr);
    }
}
```

CONSTRUCTORES

Un constructor es un método que inicia un objeto inmediatamente después de su creación. De esta forma se evita el tener que iniciar las variables explícitamente para su iniciación. El constructor tiene exactamente el mismo nombre de la clase que lo implementa; no puede haber ningún otro método que comparta su nombre con el de su clase. Una vez definido, se llamará automáticamente al constructor al crear un objeto de esa clase (al utilizar el operador *new*).

EL OPERADOR NEW

El operador new crea una instancia de una clase (objetos) y devuelve una referencia a ese objeto. Cuando ya no haya ninguna variable que haga referencia a un objeto, Java reclama automáticamente la memoria utilizada por ese objeto, a lo que se denomina *recogida de basura*. Cuando se realiza una instancia de una clase (mediante *new*) se reserva en la memoria un espacio para un conjunto de datos como el que definen los atributos de la clase que se indica en la instanciación.

EL OPERADOR PUNTO (.)

El operador punto (.) se utiliza para acceder a las variables de instancia y los métodos contenidos en un objeto, mediante su referencia a objeto
referencia_a_objeto.nombre_de_método(lista-de-argumentos);

Ejemplo No. 2:

Identifique que líneas de código son necesarias para poder ejecutar el siguiente programa.

```
public Student(String name, String grd)
{
     studentName = name;
     grade = grd;
     studentID = 99999;
}
public Student( )
{
     studentName = "";
     grade = "";
     studentID = 99999;
}
public Student(String name, String grd, int id)
{
    studentName = name;
     grade = grd;
     studentID = id;
 }
```

```
public void printData()
{
    System.out.println("Student name is: " + studentName + "

Grade is: " + grade);
}
 public void setGrade(String newGrade)
 {
     grade = newGrade;
 }
```

LA REFERENCIA THIS

Java incluye un valor de referencia especial llamado this, que se utiliza dentro de cualquier método para referirse al objeto actual. Se puede utilizar this siempre que se requiera una referencia a un objeto del tipo de una clase actual. Si hay dos objetos que utilicen el mismo código, seleccionados a través de otras instancias, cada uno tiene su propio valor único de this.

CONTROL DE ACCESO

Cuando se crea una nueva clase en Java, se puede especificar el nivel de acceso que se quiere para las variables de instancia y los métodos definidos en la clase:

Public:

public void CualquieraPuedeAcceder(){}

Cualquier clase desde cualquier lugar puede acceder a las variables y métodos de instancia públicos.

Protected:

protected void SoloSubClases(){}

Sólo las subclases de la clase y nadie más puede acceder a las variables y métodos de instancia protegidos.

Prívate:

private String NumeroDelCarnetDeIdentidad;

Las variables y métodos de instancia privados sólo pueden ser accedidos desde dentro de la clase. No son accesibles desde las subclases.

PAQUETES

Los paquetes son el mecanismo por el que Java permite agrupar clases, interfaces, excepciones y constantes. De esta forma, se agrupan conjuntos de estructuras de datos y de clases con algún tipo de relación en común. Para declarar un paquete se utiliza la sentencia package seguida del nombre del paquete que se este creando:

```
package NombrePaquete;
```

BIBLIOTECAS DE LA API DE JAVA

Estas clases se pueden incluir en los programas Java, sin temor a fallos de portabilidad, y organizadas en paquetes y en un árbol de herencia. A este conjunto de paquetes (o bibliotecas) se le conoce como la API de Java (Application Programming Interface).

Paquetes de utilidades:

- ***java.lang*:** Fundamental para el lenguaje. Incluye clases como *String* o *StringBuffer*, se importa implícitamente sin necesidad de una sentencia **import**.
- ***java.io*:** Para la entrada y salida a través de flujos de datos, y ficheros del sistema.
- ***java.util*:** Contiene colecciones de datos y clases, el modelo de eventos, información del sistema, y otras clases de utilidad.
- ***java.math*:** Clases para realizar aritmética con la precisión que se desee.

Paquetes para el desarrollo gráfico:

- ***java.applet*:** Para crear applets y clases que los applets utilizan para comunicarse con su contexto.
- ***java.awt*:** Para crear interfaces con el usuario, y para dibujar imágenes y gráficos.
- ***javax.swing*:** Conjunto de componentes gráficos que funcionan igual en todas las plataformas que Java soporta.
- ***java.net:*** Se usa para leer y escribir datos en la red.

Para importar clases de un paquete se usa el comando **import**. Se puede importar una clase individual

```
import java.awt.Font;
```

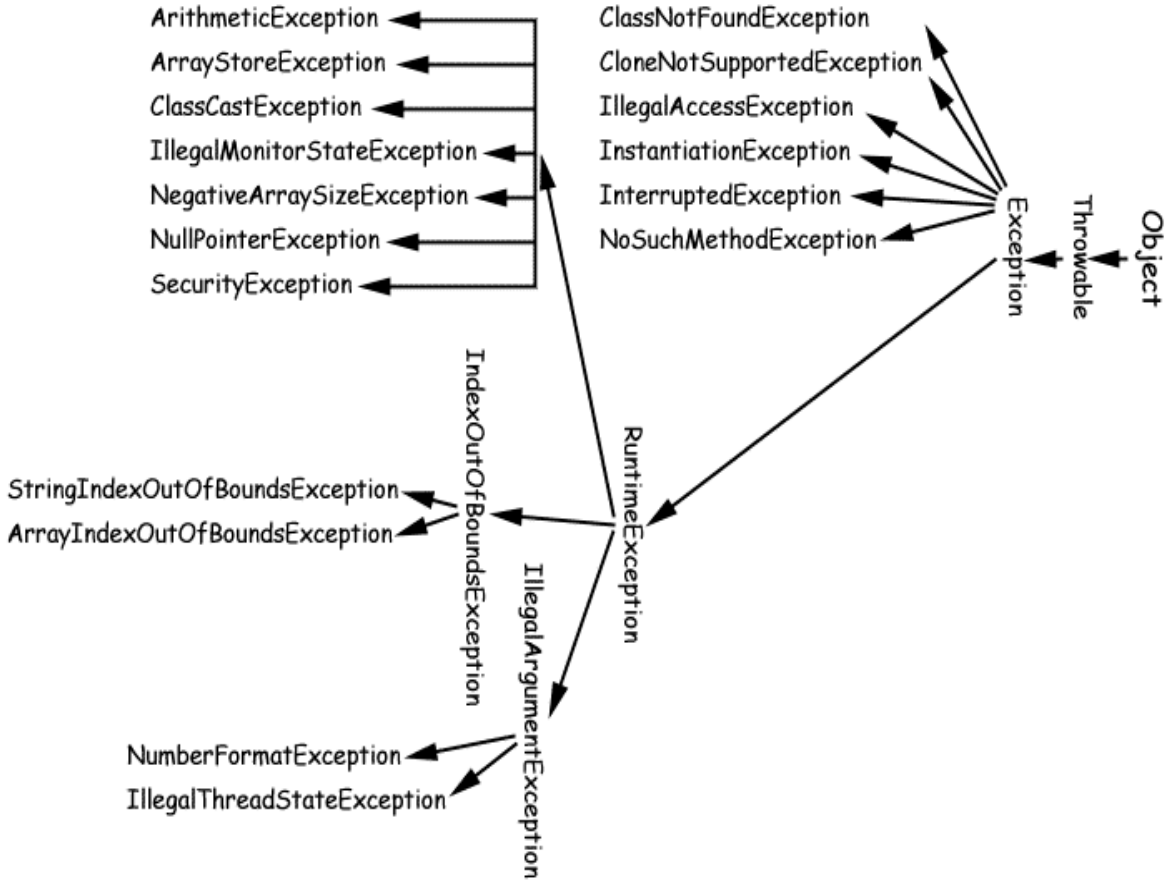
o bien, se puede importar las clases declaradas públicas de un paquete completo, utilizando un arterisco (*) para reemplazar los nombres de clase individuales.

```
import java.awt.*;
```

EXCEPCIONES

Las excepciones son una manera de controlar los errores en tiempo de ejecución. Por ejemplo si se trabaja con arreglos es importante controlar que los índices no se salgan de los límites.

Las excepciones funcionan de la siguiente manera: Cuando se sucede un error, se lanza (throw) una excepción la cual debe se recogida (catch) para tener control del error.



ArithmeticException

Las excepciones aritméticas son típicamente el resultado de división por 0:

int i = 12 / 0;

NullPointerException

Se produce cuando se intenta acceder a una variable o método antes de ser definido

IncompatibleClassChangeException

El intento de cambiar una clase afectada por referencias en otros objetos, específicamente cuando esos objetos todavía no han sido recompilados.

ClassCastException

El intento de convertir un objeto a otra clase que no es válida.

NegativeArraySizeException

Puede ocurrir si hay un error aritmético al cambiar el tamaño de un array.

OutOfMemoryException

¡No debería producirse nunca! El intento de crear un objeto con el operador new ha fallado por falta de memoria. Y siempre tendría que haber memoria suficiente porque el *garbage collector* se encarga de proporcionarla al ir liberando objetos que no se usan y devolviendo memoria al sistema.

NoClassDefFoundException

Se referenció una clase que el sistema es incapaz de encontrar.

ArrayIndexOutOfBoundsException

Es la excepción que más frecuentemente se produce. Se genera al intentar acceder a un elemento de un array más allá de los límites definidos inicialmente para ese array.

UnsatisfiedLinkException

Se hizo el intento de acceder a un método nativo que no existe. Aquí no existe un método *a.kk()*

```
class A
{
        native void kk();
}
```

y se llama a *a.kk()*, cuando debería llamar a *A.kk()*.

InternalException

Este error se reserva para eventos que no deberían ocurrir. Por definición, el usuario nunca debería ver este error y esta excepción no debería lanzarse.

Ejemplo No. 3:

En el siguiente ejemplo se pretende ilustrar el funcionamiento de las Excepciones cuando el índice de un arreglo está incorrecto.

```
int datos[] = new int[5];
try{
    datos[6]=7;
}
catch(Exception e)
{
    System.out.println("Indice fuera de límites");
}
    System.out.println("Continua");
```

Ejemplo No. 4:

En el siguiente ejemplo se pretende ilustrar el funcionamiento de las Excepciones cuando el índice de un arreglo está incorrecto.

```
int [] array = new int[20];
try
{
    array[-3] = 24;
}
catch(ArrayIndexOutOfBoundsException excepcion)
{
    System.out.println(" Error de índice en un array");
}
```

Ejemplo No. 5:

En el siguiente ejemplo se pretende ilustrar el funcionamiento de las Excepciones cuando ocurre una división por cero (0) y cuando una variable no está inicializada.

```
int valor=10;
try {
    for( x=0; x < 10; x ++ )    valor /= x;
}
catch( Exception e )
{
    System.out.println( "Se ha producido un error" );
}
```

Ejemplo No. 6:

En el siguiente ejemplo se pretende ilustrar el funcionamiento de las Excepciones cuando se pretende convertir un carácter simbólico a número.

```
String str="a";
int numero;
try{
    numero=Integer.parseInt(str);
    System.out.println(numero);
}
catch(Exception e)
{
    System.out.println("No es un número");
}
```

ACTIVIDADES DE REELABORACION

ACTIVIDADES PRÁCTICAS

1.- Desarrollar un programa que permita ingresar dos arreglos numéricos, uno de enteros y otro de decimales, por teclado. Además que presente en pantalla el resultado de sumar cada uno de los elementos de los arreglos. Considere que se debe hacer validación de los datos para que correspondan a cada arreglo.

2.- Implemente una clase Main que permita dar funcionalidad a la siguiente clase programada en Java.

```
public class Person
{
   private String name;
   private String dateOfBirth;
   private int id;

   public Person (String  aName, String aDate, int aID)
   {
      name = aName;
      dateOfBirth = aDate;
      id = aID;
   }
     public void setName(String aName)
     {
          name = aName;
     }
     public void setDateOfBirth(String adate)
     {
          dateOfBirth = adate;
     }
     public String getName()
     {
          return name;
     }
     public String getDateOfBirth()
     {
          return dateOfBirth;
     }

}
```

3.- Implemente una clase Main que permita dar funcionalidad a la siguiente clase programada en Java.

```
public class MyDate
{
    private int day;
    private int month;
    private int year;

    public MyDate(int day, int month, int year)
    {
        this.day = day;
        this.month = month;
        this.year = year;
    }

   public void setDay(int day)
   {
       this.day = day;
   }
    public void setYear(int year)
    {
        this.year = year;
    }
    public void displayDate()
    {
        System.out.println("The date is: " + month
            + "/" + day + "/" + year);
    }

} // end of MyDate class
```

4.- Implemente una clase Main que permita dar funcionalidad a la siguiente clase programada en Java.

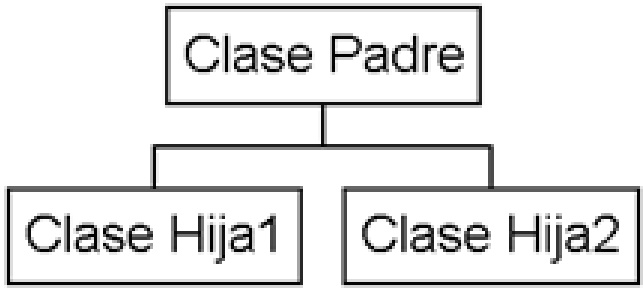
```
public class LabelText
{
    private static final int MAX_TEXT = 20;
    private String label;
   public LabelText(String inputText)
   {
       System.out.println( "Creating a LabelText object");
       if ( inputText.length() <= MAX_TEXT )
       {
           label = inputText;
       }
       else
       {
           System.err.println("Input label text is too long!");
           label = inputText.substring( 0, MAX_TEXT );
       }
   }
} // end of LabelText class
```

CAPITULO 4

HERENCIA

HERENCIA

La herencia es el mecanismo fundamental de relación entre clases en la orientación a objetos. Relaciona las clases de manera jerárquica; una clase *padre* o *superclase* sobre otras clases *hijas* o *subclases*.



Los descendientes de una clase heredan todas las variables y métodos que sus ascendientes hayan especificado como *heredables*, además de crear los suyos propios.

La característica de la herencia, es que permite definir nuevas clases derivadas de otras ya existentes, que la especializan de alguna manera. Así se logra definir una jerarquía de clases, que se puede mostrar mediante un árbol de herencia.

La clase *Object* es la clase raíz de la cual derivan todas las clases. Esta derivación es implícita. La clase *Object* define una serie de funciones miembro que heredan todas las clases.

HERENCIA MÚLTIPLE

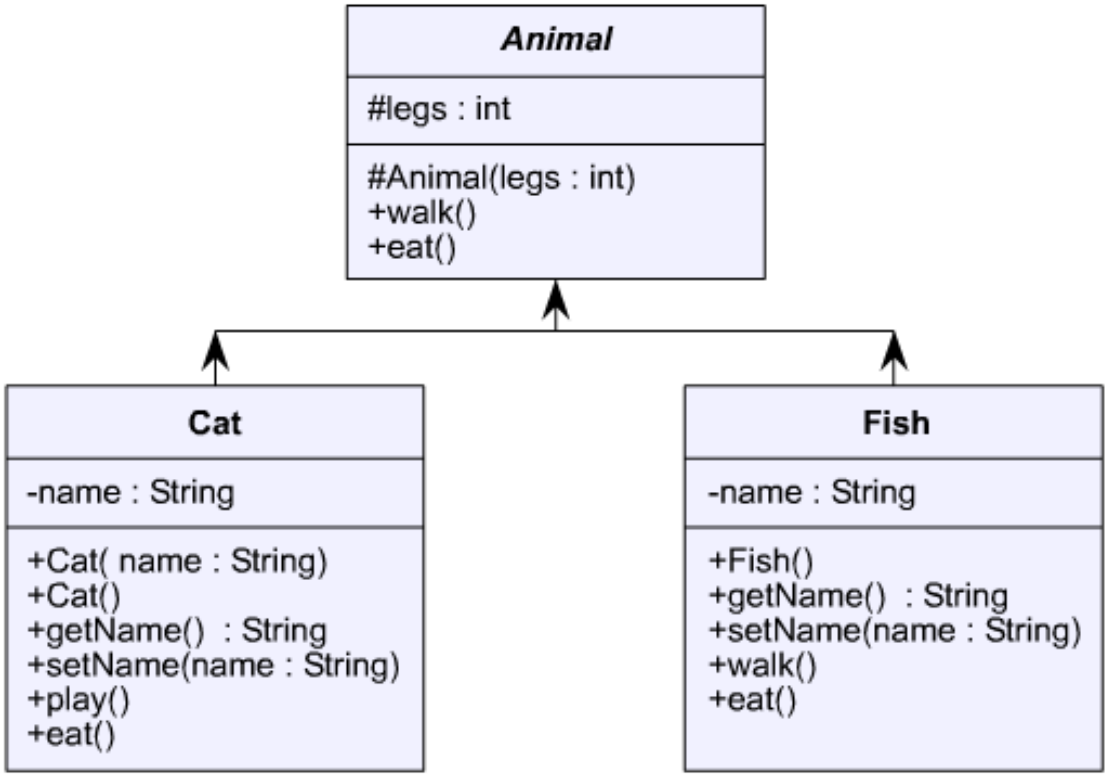
En la orientación a objetos, se consideran dos tipos de herencia, simple y múltiple. En el caso de la primera, una clase sólo puede derivar de una única superclase. Para el segundo tipo, una clase puede descender de varias superclases.

"En Java sólo se dispone de herencia simple, para una mayor sencillez del lenguaje."

La herencia ofrece una ventaja importante, permite la reutilización del código.
Una vez que una clase ha sido depurada y probada, el código fuente de dicha clase no necesita modificarse. Su funcionalidad se puede cambiar derivando una nueva clase que herede la funcionalidad de la clase base y le añada otros comportamientos.

Las razones para el uso de la herencia son:

- Reutilización de clases predefinidas y bien probadas
- Estandarización de comportamientos a través de un grupo de clases
- Capacidad para utilizar miembros de una familia de clases de manera intercambiable en los métodos



Para indicar que una clase deriva de otra, heredando sus propiedades (métodos y atributos), se usa el término ***extends***, como en el siguiente ejemplo:

```
public class SubClase extends SuperClase
{
  // Contenido de la clase
}
```

La palabra clave *extends* se utiliza para decir que deseamos crear una subclase de la clase que es nombrada a continuación,

ACCESO A LA PROPIA CLASE: THIS

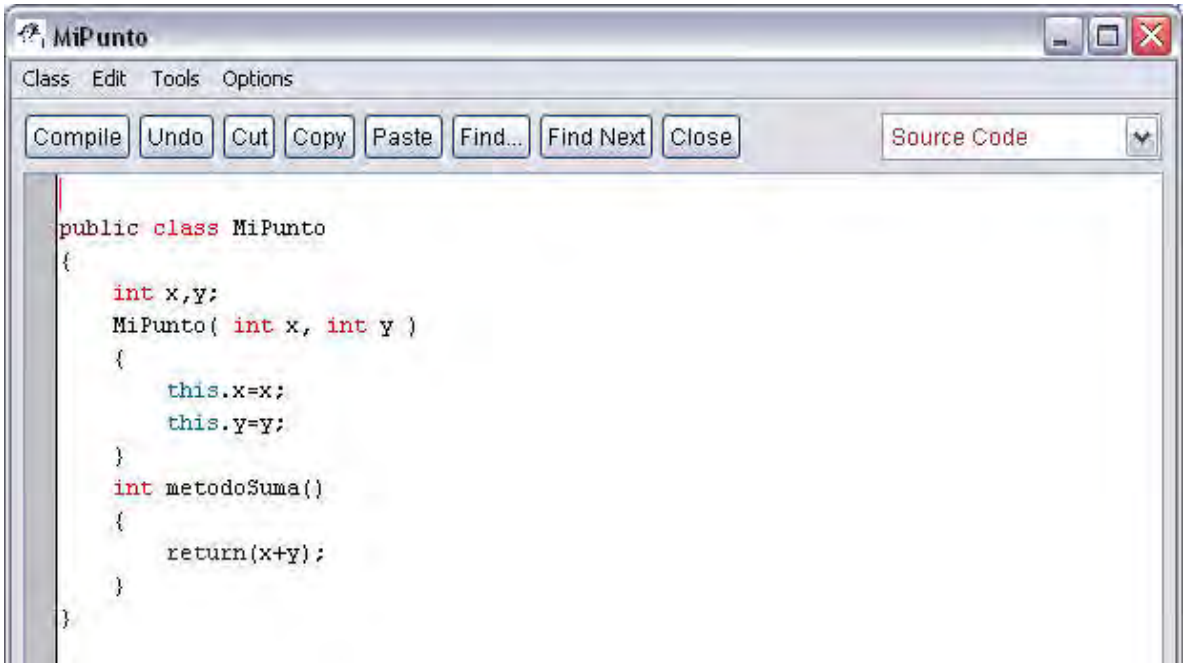
El valor this se refiere al objeto sobre el que ha sido llamado el método actual. Se puede utilizar this siempre que se requiera una referencia a un objeto del tipo de una clase actual. Si hay dos objetos que utilicen el mismo código, seleccionados a través de otras instancias, cada uno tiene su propio valor único de this.

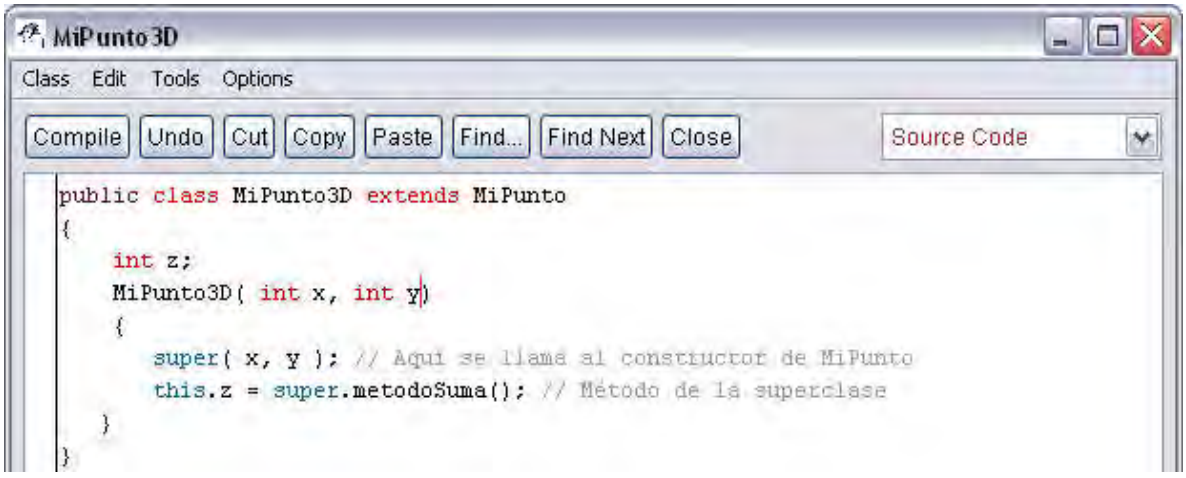
Acceso a la superclase: super

La referencia *super* usa para acceder a métodos o atributos de la superclase

Ejemplo No.1:

Implemente el siguiente código en el que se ilustra la utilización de la referencia this y super.





Ingreso

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Find Next Close Source Code

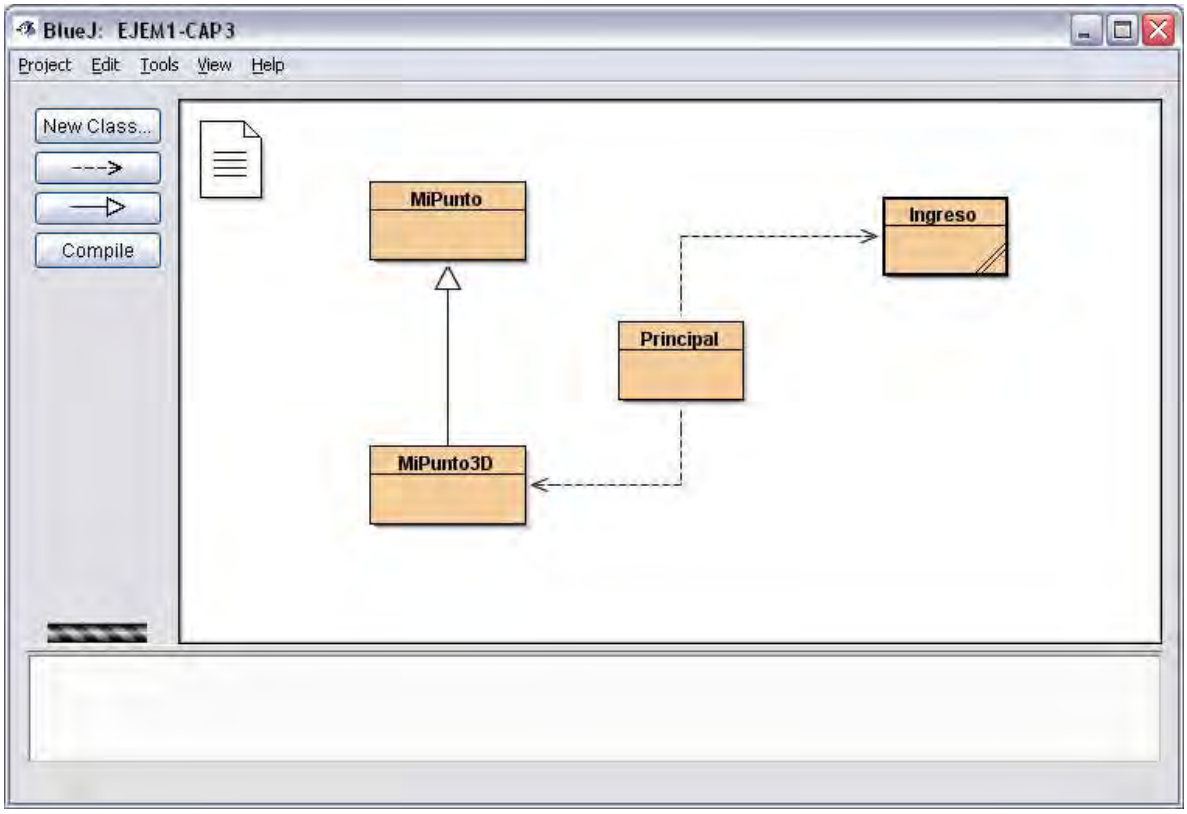
```
import java.io.*;
public class Ingreso
{
    DataInputStream ob = new DataInputStream(System.in);
    int Teclado(String msn)
    {
        System.out.println(msn);
        int numero=0;
        try
        {
            numero=Integer.parseInt(ob.readLine());
        }
        catch(IOException e) {}
        return(numero);
    }
}
```

Principal

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Find Next Close Source Code

```
public class Principal
{
    static Ingreso   obj1;
    static MiPunto3D obj2;
    public static void main(String [] args)
    {
        obj1 = new Ingreso();
        obj2 = new MiPunto3D(obj1.Teclado("Ingrese X:"),obj1.Teclado("Ingrese Y:"));
        System.out.println("X:"+obj2.x);
        System.out.println("Y:"+obj2.y);
        System.out.println("z:"+obj2.z);
    }
}
```



SOBRECARGA DE MÉTODOS

La sobrecarga de métodos es la creación de varios métodos con el mismo nombre pero con diferentes definiciones. Java diferencia los métodos sobrecargados con base en el número y tipo de argumentos que tiene el método y no por el tipo que devuelve. También existe la sobrecarga de constructores: Cuando en una clase existen constructores múltiples, se dice que hay sobrecarga de constructores.

Métodos sobrecargados:

```
int calculaSuma(int x, int y, int z)
{ ... }
int calculaSuma(double x, double y, double z)
{ ... }
```

Estos métodos no están sobrecargados:

```
int calculaSuma(int x, int y, int z)
{ ... }
double calculaSuma(int x, int y, int z)
{ ... }
```

Ejemplo No.2:

En el siguiente ejemplo se ilustra la forma de sobrecargar métodos.

```
public class OverloadDemo {

    void test() {
    System.out.println("Sin parametros");
  }

    void test(int a) {
    System.out.println("a: " + a);
  }

  void test(int a, int b) {
    System.out.println("a y b: " + a + " " + b);
  }

  double test(double a) {
    System.out.println("double a: " + a);
    return a*a;
  }
}
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        OverloadDemo ob = new OverloadDemo();
        double result;
        // llama a todas las versiones de test()
        ob.test();
        ob.test(10);
        ob.test(10, 20);
        result = ob.test(123.2);
        System.out.println("Resultado de ob.test(123.2): " + result);
    }
}
```

POLIMORFISMO

El polimorfismo es una característica por la cual se puede definir un nuevo código para un método definido anteriormente.

Ejemplo No.3:

En el siguiente ejemplo se ilustra el polimorfismo de métodos. Además implemente la clase main en la que se declare dos libros uno prestado y otro disponible.

```
class Libro{
        String Titulo;
        String Autor;
        String Estado;

        public void identificar(String t,String a){
                Titulo=t;
                Autor=a;
                Estado="Disponible";
        }

        public void prestar(){
                Estado="Prestado";
        }

        public void devolver(){
                Estado="Disponible";
        }
}

class MiLibro extends Libro{
        String usuario;

        public void prestar(String quien){
                Estado="Prestado";
                usuario=quien;
        }

        public void devolver(){
                Estado="Disponible";
                usuario=null;
        }
}
```

LAS CLASES Y MÉTODOS ABSTRACTOS: ABSTRACT

Hay situaciones en las que se necesita definir una clase que represente un concepto abstracto, y por lo tanto no se pueda proporcionar una implementación completa de algunos de sus métodos. Se puede declarar que ciertos métodos han de ser sobrescritos en las subclases, utilizando el modificador de tipo *abstract.*

Cualquier subclase de una clase *abstract* debe implementar todos los métodos *abstract* de la superclase o bien ser declarada también como *abstract.*

Ejemplo No.4:

En el siguiente ejemplo se ilustra las clases y métodos abstractos.

```
abstract class claseA
{
        abstract void metodoAbstracto();
        void metodoConcreto()
        {
                System.out.println("En el metodo concreto de claseA");
        }
}

class claseB extends claseA
 {
        void metodoAbstracto()
        {
                System.out.println("En el metodo abstracto de claseB");
        }
}

public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        claseA referenciaA = new claseB();
        referenciaA.metodoAbstracto();
        referenciaA.metodoConcreto();

    }

}
```

INTERFACES

Las interfaces Java son expresiones puras de diseño. Se trata de auténticas conceptualizaciones no implementadas que sirven de guía para definir un determinado concepto (clase) y lo que debe hacer, pero sin desarrollar un mecanismo de solución. Se trata de declarar métodos abstractos y constantes que posteriormente puedan ser implementados de diferentes maneras según las necesidades de un programa.

Para declarar una interfaz se utiliza la sentencia *interface*, de la misma manera que se usa la sentencia *class*:

```
interface MiInterfaz
{
        final int CONSTANTE = 100;
        int metodoAbstracto( int parametro );
}
```

Se observa en la declaración que las variables adoptan la declaración en mayúsculas, ya que actuarán como constantes. Los métodos tras su declaración presentan un punto y coma, en lugar de su cuerpo entre llaves. Son métodos abstractos, por tanto, métodos sin implementación.

La palabra reservada *implements* utilizada en la declaración de una clase indica que la clase implementa la interfaz, es decir, que asume las constantes de la interfaz, y codifica sus métodos:

```
class ImplementaInterfaz implements MiInterfaz
{
        int multiplicando=CONSTANTE;
        int metodoAbstracto( int parametro )
{
                return ( parametro * multiplicando );
        }
}
```

Una interfaz no puede implementar otra interfaz, aunque sí extenderla (*extends*) ampliándola.

Ejemplo No.5:

En el siguiente ejemplo se ilustra la declaración de interfaces.

```
interface Figura
{
    int area();
}

public class Cuadrado implements Figura {

    int lado;

    public Cuadrado (int ladoParametro) {
        lado = ladoParametro;
    }

    public int area(){
        return lado*lado;
    }
}
```

```
public static void main(String[] args) {
    // TODO code application logic here
    Figura figura=new Cuadrado (5);
    System.out.println("El area es: "+figura.area());
}
```

LOS ELEMENTOS GLOBALES: STATIC

Para crear un método o una variable que se utiliza fuera del contexto de cualquier instancia, es decir, de una manera global a un programa. Todo lo que se tiene que hacer es declarar estos elementos como *static.*

```
static int a = 3;
static void metodoGlobal()
{
        // implementación del método
}
```

Otro aspecto en el que es útil *static* es en la creación de métodos a los que se puede llamar directamente diciendo el nombre de la clase en la que están declarados. Se puede llamar a cualquier método *static*, o referirse a cualquier variable *static*

ACTIVIDADES DE REELABORACION

ACTIVIDADES PRÁCTICAS

1.- Desarrolle un programa que disponga de las siguientes clases:

(A)Una clase que tenga dos atributos de clase ***x*** e ***y***, privados, de tipo entero.

Un constructor que inicialice los atributos de clase con argumentos enviados y otro que los inicialice con cero. Un método protegido que imprima en pantalla los valores de sus atributos de clase. Un método protegido para incrementar en una unidad x. Un método protegido para incrementar en una unidad.

(B) Una clase que sea hija de la primera y que disponga de dos atributos de clase privados *w* y *z*, de tipo entero. Un constructor que inicialice sus atributos de clase con cero, otro que los inicialice con argumentos enviados y otro que inicialice tanto sus argumentos como los argumentos de la clase padre. Un método protegido que incremente los atributos de clase y los tributos de la clase padre. Un método protegido que imprima en pantalla los valores de sus atributos de clase.

(C) Una clase que sea hija de la segunda y que disponga de dos atributos de clase privados *t* y *v*, de tipo entero. Un constructor que inicialice los atributos de clase con argumentos enviados, tanto para sus argumentos como para los argumentos de la clase padre. Un método protegido que incremente los atributos de clase y los atributos de la clase padre. Un método protegido que imprima en pantalla los valores de sus atributos de clase.

Además considere el siguiente código para las declaraciones de las clases anteriores.

```
public static void main(String[] args)
{
    C c1= new C(10,11,12,13,14,15);
    c1.imprimirA();
    c1.imprimirB();
    c1.imprimirC();
    c1.incT();
    c1.incV();
    c1.incW();
    c1.incZ();
    c1.incX();
    c1.incY();
    c1.imprimirA();
    c1.imprimirB();
    c1.imprimirC();
}
```

2.- Desarrolle un programa que permita almacenar el nombre y teléfono de usuario de una biblioteca en una clase y que disponga de: Un constructor que inicialice sus atributos con “Nadie” y “000”. Un constructor que inicialice sus atributos a partir de argumentos enviados. Un método que permita presentar en pantalla la información. Métodos de consulta y modificadores.

Un asegunda clase que permita almacenar el nombre de un libro en una clase y que disponga de: Un constructor que inicialice su atributo con “Vacio”. Así como los atributos de la clase padre con “Nadie” y “000”. Un constructor que inicialice su atributo y los de la clase padre a partir de argumentos enviados. Un método que permita presentar en pantalla la información.

Un método de consulta. Un método que permita asignar el libro a un nuevo usuario. Un método que permita saber a partir de un grupo de libros si uno de ellos se encuentra o no asignado a un usuario.

En la clase principal el usuario definirá con cuántos libros empieza el programa así como si los libros nombres de los libros y si se encuentran o no asignados. Además deberá permitir al usuario seleccionar un libro y que se presente en pantalla el nombre del libro y los datos del usuario si existiese.

3.- Implemente una clase main en la que se declare un objeto con cada tipo de constructor.

```
public class Usuario4 {

    private String   nombre;
    private int      edad;
    private String   direccion;

     Usuario4( )
     {
        nombre     = null;
        edad       = 0;
        direccion  = null;
     }

     Usuario4(String nombre, int edad, String direccion)
     {
        this.nombre     = nombre;
        this.edad       = edad;
        this.direccion  = direccion;
     }

     Usuario4(Usuario4 persona)
     {
        nombre     = persona.getNombre();
        edad       = persona.getEdad();
        direccion  = persona.getDireccion();
     }
```

```
    void setNombre(String nombre)
    {
        this.nombre = nombre;
    }

    String getNombre()
    {
        return nombre;
    }

    void setEdad(int edad)
    {
        this.edad = edad;
    }

    int getEdad()
    {
        return edad;
    }

    void setDireccion(String d)
    {
        direccion = d;
    }

    String getDireccion()
    {
        return direccion;
    }
}
```

4.- Diseñar un clase **Funcion** con un método abstracto f(double x) que representa una función. Además implementar un método llamado evaluar que llama al método f, retornando su valor. Cree la clase **FuncionExp** que implemente la función exponencial y=e^x. Cree la clase **FuncionLineal** que representa la ecuacion de una recta y=ax+b.

Además considere el siguiente código para las declaraciones de las clases anteriores.

```
public static void main(String[] args) {
    // TODO code application logic here
    Funcion fexp = new FuncionExp( );
    System.out.println("e elevado a cero : " + fexp.evaluar(0) );
    System.out.println("e elevado a uno : " + fexp.evaluar(1.0) );

    Funcion flineal = new FuncionLineal(0.5,1.0);
    System.out.println("El valor de la funcion : " +flineal.evaluar(2.0));

}
```

5.- Defina una clase abstracta **aviones**, con dos métodos despegar y aterrizar.

Defina una interface **vuelos**, con dos constantes, una para máxima velocidad 1500 y otra para mínima velocidad 1000. Además el método abstracto cinturones.

Defina una clase Boing747 que sea hija de aviones e implemente la interfase vuelos. Que tenga como atributos el peso y el estado del aeropuerto. Además defina los métodos abstractos heredados. Para poder despegar el peso debe ser menor a 180 y el aeropuerto debe estar libre. Para poder aterrizar el aeropuerto debe estar libre y la velocidad menor a 1000. En caso de ser la velocidad mayor a 1500 se debe indicar que los cinturones deben asegurarse.

Defina una clase Hércules que sea hija de aviones e implemente la interfase vuelos. Que tenga como atributos el peso y el estado del aeropuerto. Además defina los métodos abstractos heredados. Para poder despegar el peso debe ser menor a 160 y el aeropuerto debe estar libre. Para poder aterrizar el aeropuerto debe estar libre y la velocidad menor a 1500. En caso de ser la velocidad mayor a 1500 se debe indicar que los cinturones deben asegurarse.

Para realizar las pruebas defina una clase principal en la que se cree dos aviones, uno Hércules y otro Boing; y se pida al usuario que ingrese el peso de cada avión y el estado del aeropuerto. El usuario determinara si se encuentra volando o por despegar. Si el avión está volando se deberá indicar la velocidad. A partir de los datos ingresados hacer el análisis si pude o no despegar o aterrizar y si se deben o no asegurar los cinturones

ACTIVIDADES TEÓRICAS

1.- Analice el siguiente código y explique la accesibilidad de la clase hija a los atributos de la clase padre.

```
public class Ventana
{
        private int a;
        public int b;
        int c;
```

```
        protected int d;
}

public class VentanaTitulo extends Ventana
{
        public void desplazar(int dx, int dy, int dz)
        {
                d+=dx;
                c+=dy;
                d+=dz;
        }
}
```

CAPITULO 5

INTERFACES GRÁFICAS

COMPONENTES DE UNA INTERFAZ GRAFICA CON EL USUARIO

JFC (Java Foundation Classes) es parte de la API de Java compuesto por clases que sirven para crear interfaces gráficas visuales para las aplicaciones de Java. Las JFC contienen dos paquetes gráficos: AWT y Swing.

AWT presenta componentes pesados, que en cada plataforma sólo pueden tener una representación determinada. Está disponible en el JDK como *java.awt*.

Swing presenta componentes ligeros, que pueden tomar diferente aspecto y comportamiento pues lo toman de una biblioteca de clases. Está disponible en el JDK como *javax.swing.*

MODELO DE EVENTOS

Para cada objeto que represente una interfaz gráfica, se pueden definir objetos "oyentes" (*Listener*), que esperan a que suceda un determinado evento sobre la interfaz. Por ejemplo se puede crear un objeto oyente que esté a la espera de que el usuario pulse sobre un botón de la interfaz, y si esto sucede, él es avisado, ejecutando determinada acción.

El modelo de eventos depende del paquete:
java.awt.event
javax.swing.event.

Subpaquetes de AWT: (*java.awt)* Contiene todas las clases básicas de AWT para crear interfaces e imprimir gráficos e imágenes, así como la clase base para los eventos en componente.

java.awt.event: Modelo de eventos de AWT. Contiene eventos y oyentes.
java.awt.color: Utilización de colores.
java.awt.font: Todo lo referente a las fuentes de texto.

Subpaquetes de Swing: (*javax.swing)* Tiene los componentes básicos para crear componentes ligeros Swing.

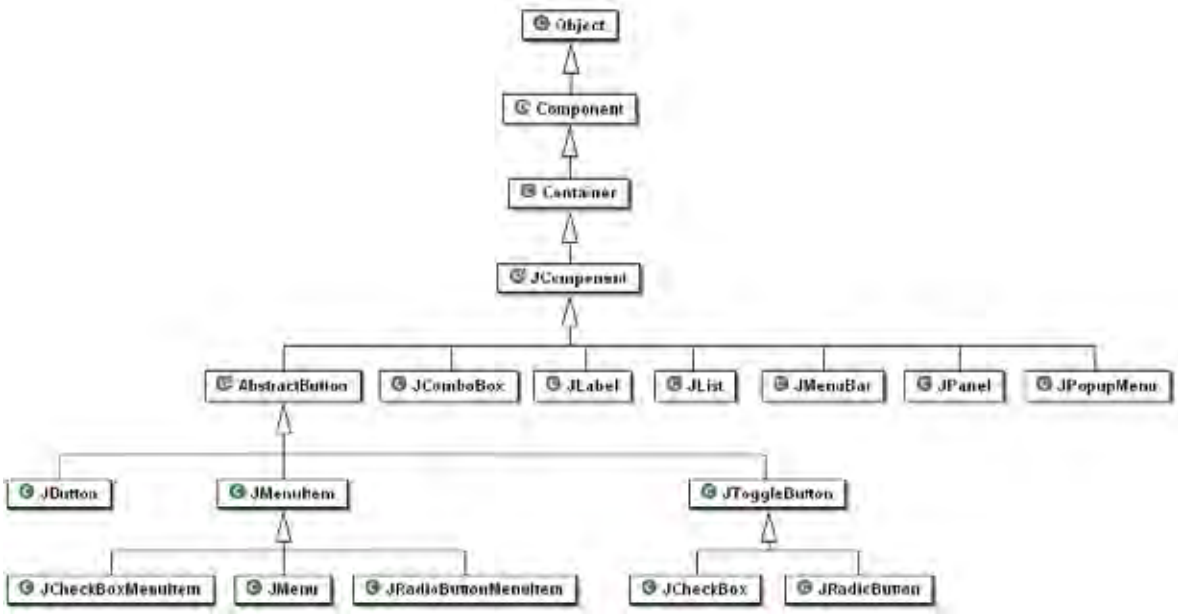
javax.swing.event: Eventos lanzados por componentes Swing, así como oyentes para dichos eventos

javax.swing.text: Para manejar componentes de texto. Soporta sintaxis resaltada, edición, estilos...

Las clases de Swing se parecen mucho a las de AWT. En general las clases que comiencen por "J" son componentes que se pueden añadir a la aplicación. Por ejemplo: *JButton*. Esto se cumple para todas las clases menos para *Choice, Canvas* y *ScrollPane*. Todas las clases componentes de Swing (clases hijas de *JComponent*), son hijas de la clase *Component* de AWT.

Una interface gráfica está construida en base a elementos gráficos básicos, llamados *Componentes*. Ejemplos de estos Componentes son los botones, barras de desplazamiento, etiquetas, listas, cajas de selección o campos de texto. Los Componentes permiten al usuario interactuar con la aplicación y proporcionar información desde el programa al usuario sobre el estado del programa. En el AWT, todos los Componentes de la interface de usuario son instancias de la clase ***Component*** o uno de sus subtipos.

Los Componentes no se encuentran aislados, sino agrupados dentro de *Contenedores*. Los Contenedores contienen y organizan los Componentes; además, los Contenedores son en sí mismos Componentes y como tales pueden ser situados dentro de otros Contenedores.



CONTAINER

La clase *Container* sabe cómo mostrar componentes embebidos. Algunos de los métodos de la clase *Container* son:

- *Component* ***add(Component c);*** Añade un componente al contenedor.

- *void* ***setLayout(LayoutManager);*** Establece un gestor de impresión para este componente.

COMPONENT

Clases componentes (hijas directas de Component):

- *Button*: Un botón gráfico para el que se puede definir una acción que sucederá cuando se presione el botón.
- *Canvas*: Permite crear gráficos.
- *Checkbox*: Soporta dos estados: *on* y *off*. Se pueden asociar acciones que se ejecuten cuando el estado cambie.
- *Choice*: Menú desplegable de opciones.
- *Label*: Cadena de etiqueta en una localización dada.
- *List*: Una lista desplegable de cadenas.
- *Scrollbar*: Desplegable de objetos *Canvas*.
- *TextComponent*: Cualquier componente que permita editar cadenas de texto.Tiene dos clases hijas:
- *TextField*: Componente de texto consistente en una línea que puede ser usada para construir formularios.
- *TextArea*: Componente para edición de texto de tamaño variable.

Acciones sobre el componente:

- *boolean* ***isEnabled();*** Comprueba si el componente está o no activo.
- *void* ***setEnable(boolean);*** Establece el componente a activo o inactivo.
- *boolean* ***isVisible();*** Comprueba si el componente está o no visible.
- *void* ***setVisible(boolean);*** Establece si el componente está visible o invisible.

EVENTOS

Eventos físicos:

- *InputEvent*: Se ha producido una entrada del usuario. Tiene como eventos hijos *KeyEvent* (pulsación de una tecla) y *MouseEvent* (acción sobre el ratón).

Eventos semánticos:

- *ActionEvent*: Avisa al programa de acciones específicas de componentes como las pulsaciones de botones.
- *ItemEvent*: Avisa al programa cuando el usuario interacciona con una elección, una lista o una casilla de verificación.

Component	Eventos generados	Significado
Button	ActionEvent	Clicar en el botón
Checkbox	ItemEvent	Seleccionar o deseleccionar un item
CheckboxMenuItem	ItemEvent	Seleccionar o deseleccionar un item
Choice	ItemEvent	Seleccionar o deseleccionar un item
Component	ComponentEvent	Mover, cambiar tamaño, mostrar u ocultar un componente
	FocusEvent	Obtener o perder el focus
	KeyEvent	Pulsar o soltar una tecla
	MouseEvent	Pulsar o soltar un botón del ratón; entrar o salir de un componente; mover o arrastrar el ratón (tener en cuenta que este evento tiene dos Listener)
Container	ContainerEvent	Añadir o eliminar un componente de un container
List	ActionEvent	Hacer doble click sobre un item de la lista
	ItemEvent	Seleccionar o deseleccionar un item de la lista
MunuItem	ActionEvent	Seleccionar un item de un menú
Scrollbar	AdjustementEvent	Cambiar el valor de la scrollbar
TextComponent	TextEvent	Cambiar el texto
TextField	ActionEvent	Terminar de editar un texto pulsando Intro
Window	WindowEvent	Acciones sobre una ventana: abrir, cerrar, iconizar, restablecer e iniciar el cierre

AWT Components	Eventos que se pueden generar									
	ActionEvent	AdjustementEvent	ComponentEvent	ContainerEvent	FocusEvent	ItemEvent	KeyEvent	MouseEvent	TextEvent	WindowEvent
Button	✓		✓		✓		✓	✓		
Canvas			✓		✓		✓	✓		
Checkbox			✓		✓	✓	✓	✓		
Checkbox-MenuItem						✓				
Choice			✓		✓	✓	✓	✓		
Component			✓		✓		✓	✓		
Container			✓	✓	✓		✓	✓		
Dialog			✓	✓	✓		✓	✓		✓
Frame			✓	✓	✓		✓	✓		✓
Label			✓		✓		✓	✓		
List	✓		✓		✓	✓	✓	✓		
MenuItem	✓									
Panel			✓	✓	✓		✓	✓		
Scrollbar		✓	✓		✓		✓	✓		
TextArea			✓		✓		✓	✓	✓	
TextField	✓		✓		✓		✓	✓	✓	
Window			✓	✓	✓		✓	✓		✓

FUENTES

Para establecer tipos de letras necesitamos el objeto Font el cual permite establecer un tipo de letra y su tamaño.

Font f = new Font(“Tipo de letra”,estilo,tamaño);

Los tipos de letras pueden ser:

- TimesRoman
- Helvitica
- Courier
- Arial

Los estilos pueden ser:

- Font.BOLD
- Font.PLAIN
- Font.ITALIC

Para poner el tipo de letra se usa el método:

void setFont(f);

CLASE COLOR

La clase java.awt.Color encapsula colores utilizando el formato RGB(Red,Green,Blue). Las componentes de cada color primario en el color resultante se expresan con números enteros entre 0 y 255, siendo 0 la intensidad mínima de ese color, y 255 la máxima. En la clase Color existen constantes para colores predeterminados de uso frecuente: black, white, green, blue, red, yellow, magemta, cyan, orange, pink, gray, darkGray, lightGray.

Ejemplo No. 1:

En este ejemplo se ilustra la forma de crear una interface gráfica por líneas de código, en la cual se están insertando Etiquetas.

```
import java.awt.*;
import javax.swing.*;

public class Etiqueta extends JPanel {
    public Etiqueta()
    {
        setLayout(new GridLayout(2,1));

        JLabel A = new JLabel();
        A.setText("Etiqueta 1");
        add(A);

        JLabel B = new JLabel("Etiqueta 2");
        B.setFont(new Font ("Arial",Font.BOLD,18));
        add(B);
    }
}
```

```
import javax.swing.*;
import java.awt.event.*;

public class Main {

    public static void main(String[] args) {
        // TODO code application logic here
        JFrame ventana = new JFrame("ESPE");
        Etiqueta obj    = new Etiqueta();
        ventana.getContentPane().add(obj);
        ventana.setSize(300, 150);
        ventana.setVisible(true);
    }
}
```

BOTONES

La clase **Button** es una clase que produce un componente de tipo botón con un título. El constructor más utilizado es el que permite pasarle como argumento una cadena, que será la que aparezca como título e identificador del botón en el interfaz de usuario. Además dispone de todos los métodos heredados de las clases **Component** y **Object**.

Métodos de la clase Button	Función que realiza
Button(String label) y Button()	Constructores
setLabel(String str), String getLabel()	Permite establecer u obtener la etiqueta del Button
addActionListener(ActionListener al), removeActionListener(ActionListener al)	Permite registrar el objeto que gestionará los eventos, que deberá implementar ActionListener
setActionCommand(String cmd), String getActionCommand()	Establece y recupera un nombre para el objeto Button independiente del label y del idioma

CHECKBOX

Los objetos de la clase Checkbox son botones de opción o de selección con dos posibles valores: on y off. Al cambiar la selección de un Checcbox se produce un ItemEvent.

Métodos de Checkbox	Función que realizan
Checkbox(), Checkbox(String), Checkbox(String, boolean), Checkbox(String, boolean, CheckboxGroup), Checkbox(String, CheckboxGroup, boolean)	Constructores de Checkbox. Algunos permiten establecer la etiqueta, si está o no seleccionado y si pertenece a un grupo
addItemListener(ItemListener), removeItemListener(ItemListener)	Registra o elimina los objetos que gestionarán los eventos ItemEvent
setLabel(String), String getLabel()	Establece u obtiene la etiqueta del componente
setState(boolean), boolean getState()	Establece u obtiene el estado (true o false, según esté seleccionado o no)
setCheckboxGroup(CheckboxGroup), CheckboxGroup getCheckboxGroup()	Establece u obtiene el grupo al que pertenece el Checkbox

Ejemplo No. 2:

En este ejemplo se ilustra la forma de crear una interface gráfica por líneas de código, en la cual se están insertando Buttons, RadioButtons y ChecBox..

```
import javax.swing.*;

public class Boton extends JPanel {
    public Boton()
    {
        add(new JButton("BOTON 1"));
        add(new JToggleButton("BOTON 1"));
        add(new JCheckBox("CHECKBOX 1"));
        add(new JRadioButton("RADIOBOTON 1"));
    }
}
import javax.swing.*;
import java.awt.event.*;

public class Main {

    public static void main(String[] args) {
        JFrame ventana  = new JFrame("ESPE");
        JFrame ventana2 = new JFrame("ESPE");
        Etiqueta obj    = new Etiqueta();
        Boton    obj2   = new Boton();
        ventana.getContentPane().add(obj);
        ventana2.getContentPane().add(obj2);
        ventana.setVisible(true);
        ventana.setBounds(100, 100, 300, 150);
        ventana2.setVisible(true);
        ventana2.setBounds(600, 100, 300, 150);
        ventana2.addWindowListener(new WindowAdapter()
            {
                public void windowClosing(WindowEvent evt)
                {
                        System.exit(0);
                }
            });
    }
}
```

AREAS DE TEXTO

TextArea y TextField heredan de la clase TextComponent y muestran texto seleccionable y editable. La diferencia principal es que TextField solo puede tener una línea, mientras que TextArea puede tener varias líneas

Métodos heredados de TextComponent	**Función que realizan**
String getText() y setText(String str)	Permiten establecer u obtener el texto del componente
setEditable(boolean b), boolean isEditable()	Hace que el texto sea editable o pregunta por ello
setCaretPosition(int n), int getCaretPosition()	Fija la posición del punto de inserción o la obtiene
String getSelectedText(), int getSelectionStart() y int getSelectionEnd()	Obtiene el texto seleccionado y el comienzo y el final de la selección
selectAll(), select(int start, int end)	Selecciona todo o parte del texto
Métodos de TextField	**Función que realizan**
TextField(), TextField(int ncol), TextField(String s), TextField(String s, int ncol)	Constructores de TextField
int getColumns(), setColoumns(int)	Obtiene o establece el número de columnas del TextField
setEchoChar(char c), char getEchoChar(), boolean echoCharIsSet()	Establece, obtiene o pregunta por el carácter utilizado para passwords, de forma que no se pueda leer lo tecleado por el usuario
Métodos de TextArea	**Función que realizan**
TextArea(), TextArea(int nfil, int ncol), TextArea(String text), TextArea(String text, int nfil, int ncol)	Constructores de TextArea
setRows(int), setColumns(int), int getRows(), int getColumns()	Establecer y/u obtener los números de filas y de columnas
append(String str), insert(String str, int pos), replaceRange(String s, int i, int f)	Añadir texto al final, insertarlo en una posición determinada y reemplazar un texto determinado

SCROLLBAR

Un Scrollbar es una barra de desplazamiento con un cursor que permite introducir y modificar valores, entre unos valores mínimo y máximo, con pequeños y grandes desplazamientos.

Métodos de Scrollbar	**Función que realizan**
Scrollbar(), Scrollbar(int pos), Scrollbar(int pos, int val, int vis, int min, int max)	Constructores de Scrollbar
addAdjustmentListener(AdjustmentListener)	egistra el objeto que gestionará los eventos
int getValue(), setValue(int)	Permiten obtener y fijar el valor
setMaximum(int), setMinimum(int)	Establecen los valores máximo y mínimo
setVisibleAmount(int), int getVisibleAmount()	Establecen y obtienen el tamaño del área visble
setUnitIncrement(int), int getUnitIncrement()	Establecen y obtienen el incremento pequeño
setBlockIncrement(int), int getBlockIncrement()	Establecen y obtienen el incremento grande
setOrientation(int), int getOrientation()	Establecen y obtienen la orientación
setValues(int value, int vis, int min, int max)	Establecen los parámetros de la barra

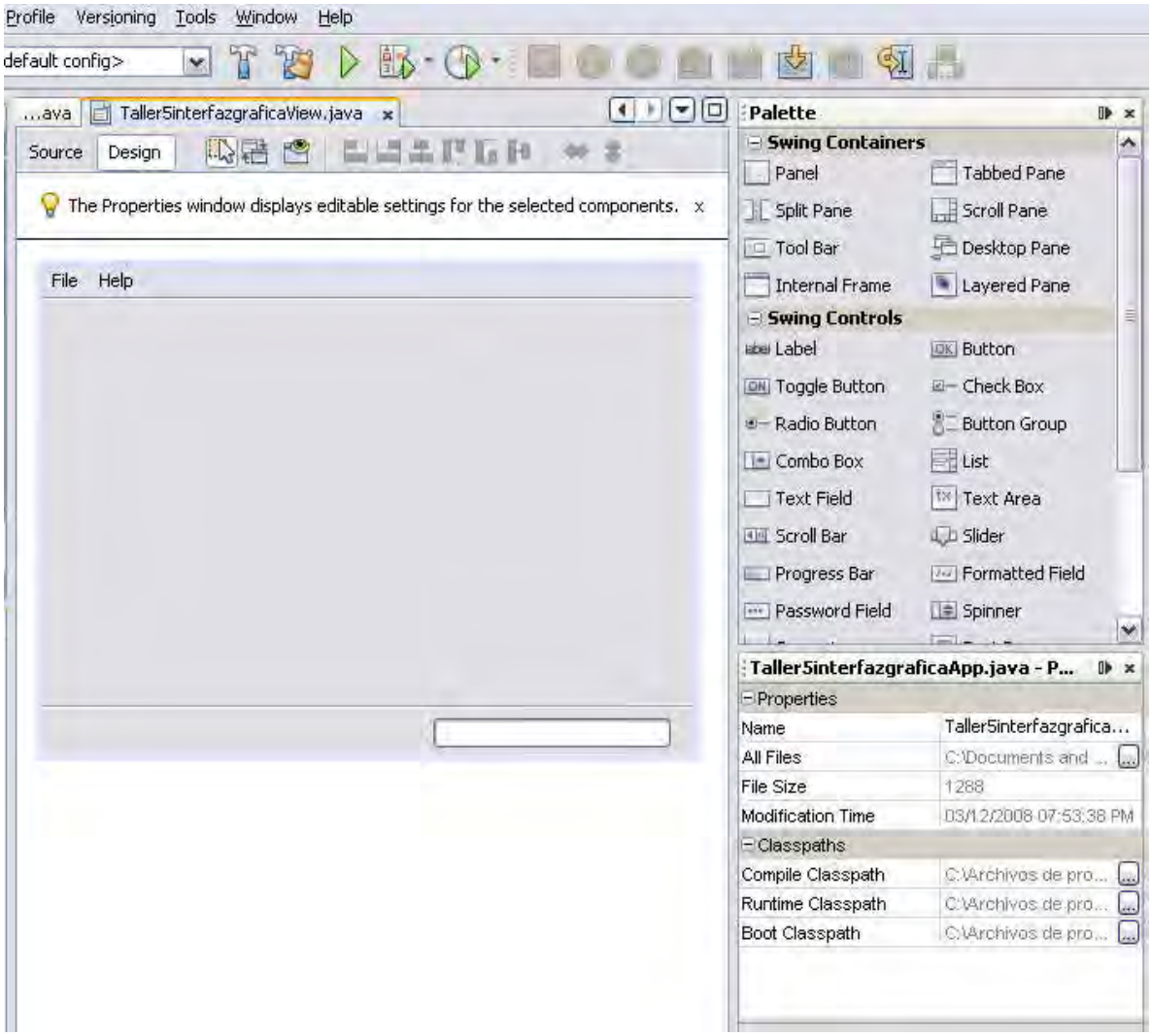
Ejemplo No. 3:

En este ejemplo se ilustra la forma de crear una interface gráfica por líneas de código, en la cual se están insertando un Slider y un Progressbar.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;

public class Barras extends JPanel{
    JProgressBar barraProg = new JProgressBar();
    JSlider      barraSlid = new JSlider(JSlider.HORIZONTAL);
    Barras()
    {
        setLayout(new GridLayout(2,1));
        add(barraProg);
        add(barraSlid);
        barraSlid.setValue(0);
        barraSlid.setPaintTicks(true);
        barraSlid.addChangeListener(new ChangeListener()
            {
                public void stateChanged(ChangeEvent evt)
                {
                        barraProg.setValue(barraSlid.getValue());
                }
            });
    }
}
import javax.swing.*;
import java.awt.event.*;

public class NewMain {

    public static void main(String[] args) {
        JFrame ventana  = new JFrame("ESPE");
        Barras obj    = new Barras();
        ventana.getContentPane().add(obj);
        ventana.setVisible(true);
        ventana.setBounds(100, 100, 300, 150);
        ventana.addWindowListener(new WindowAdapter()
            {
                public void windowClosing(WindowEvent evt)
                {
                        System.exit(0);
                }
            });
    }
}
```

INTERFAZ GRAFICA (GUI)

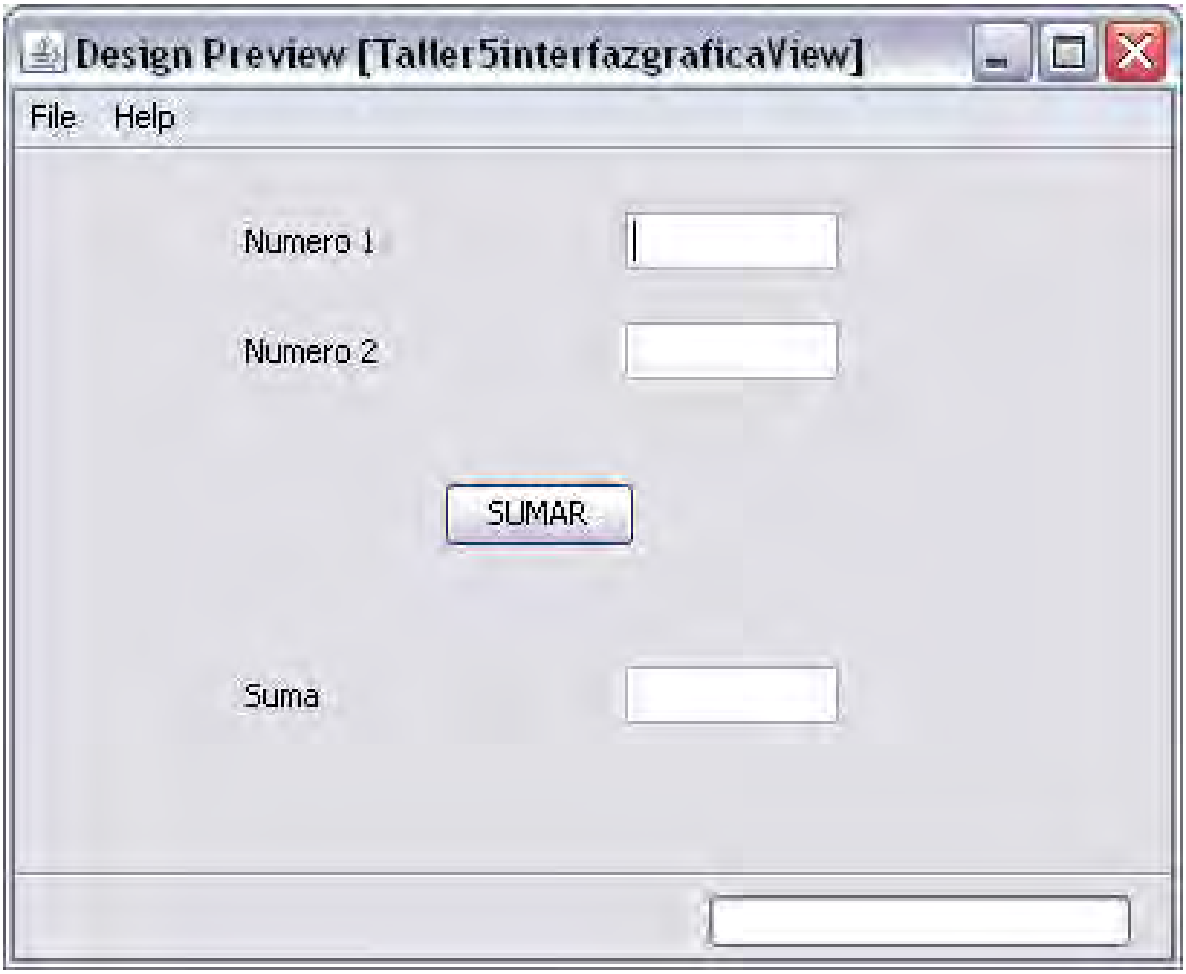
La creación del interfaz gráfico de usuario (GUI) con Netbeans, comienza creando un *contenedor Jframe* mediante una determinada plantilla, y luego se arrastran a él, los componentes visuales, ajustándoles las propiedades que se necesiten. Existen otros posibles contenedores como: JDialog, JInternalFrame, JFrame o JPanel de Swing

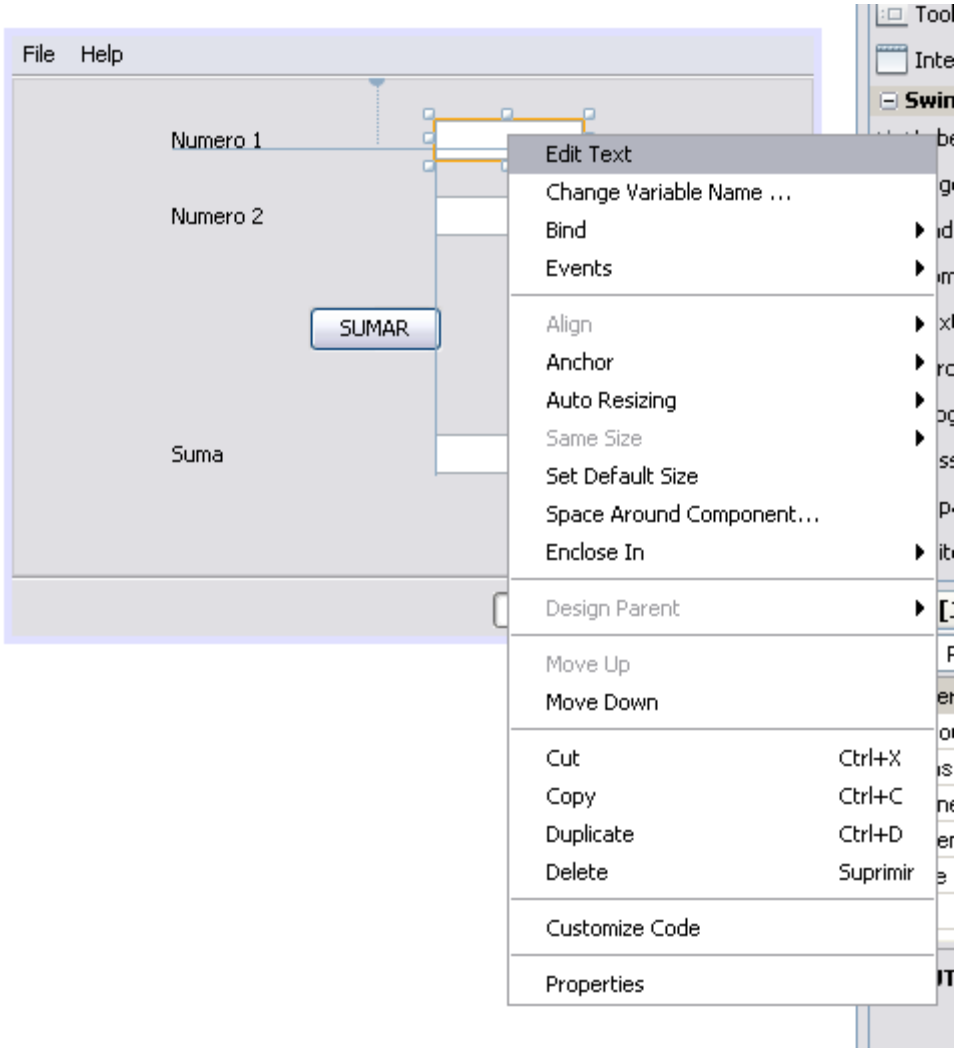


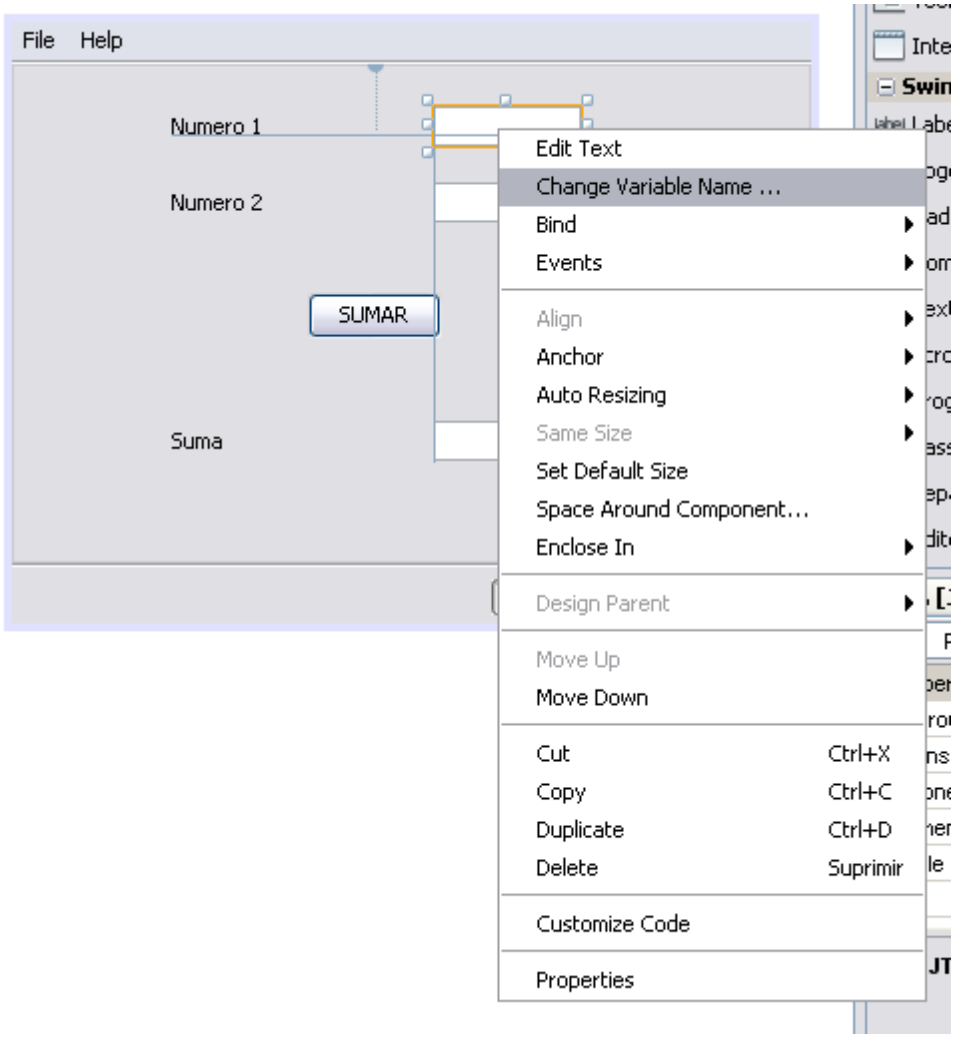
El IDE crea el form, la clase dentro del fichero .java y un form para el about.
El form se abre en el icono (*Design*) que muestra una vista gráfica de los componentes GUI. Además se abre una ventana para la *paleta de componentes*, el *inspector de componentes* aparece en la parte izquierda debajo de la ventana de proyectos, y la *ventana de propiedades* aparece en la parte derecha debajo de la paleta de componentes.

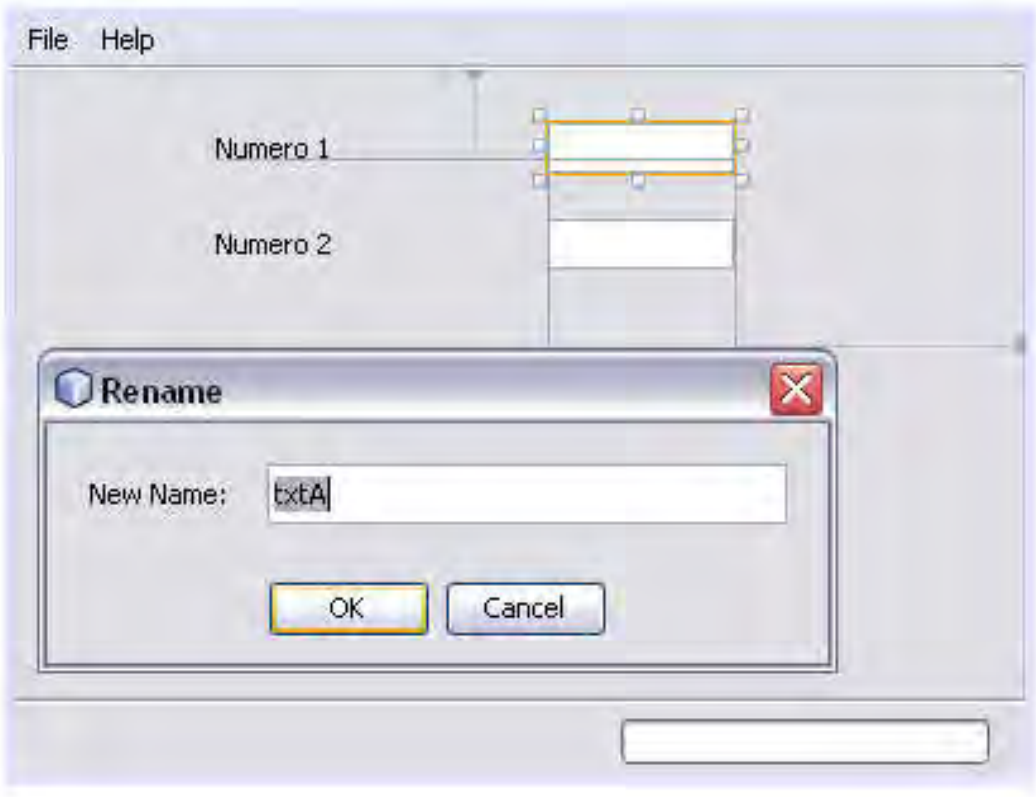
Ejemplo No. 4:

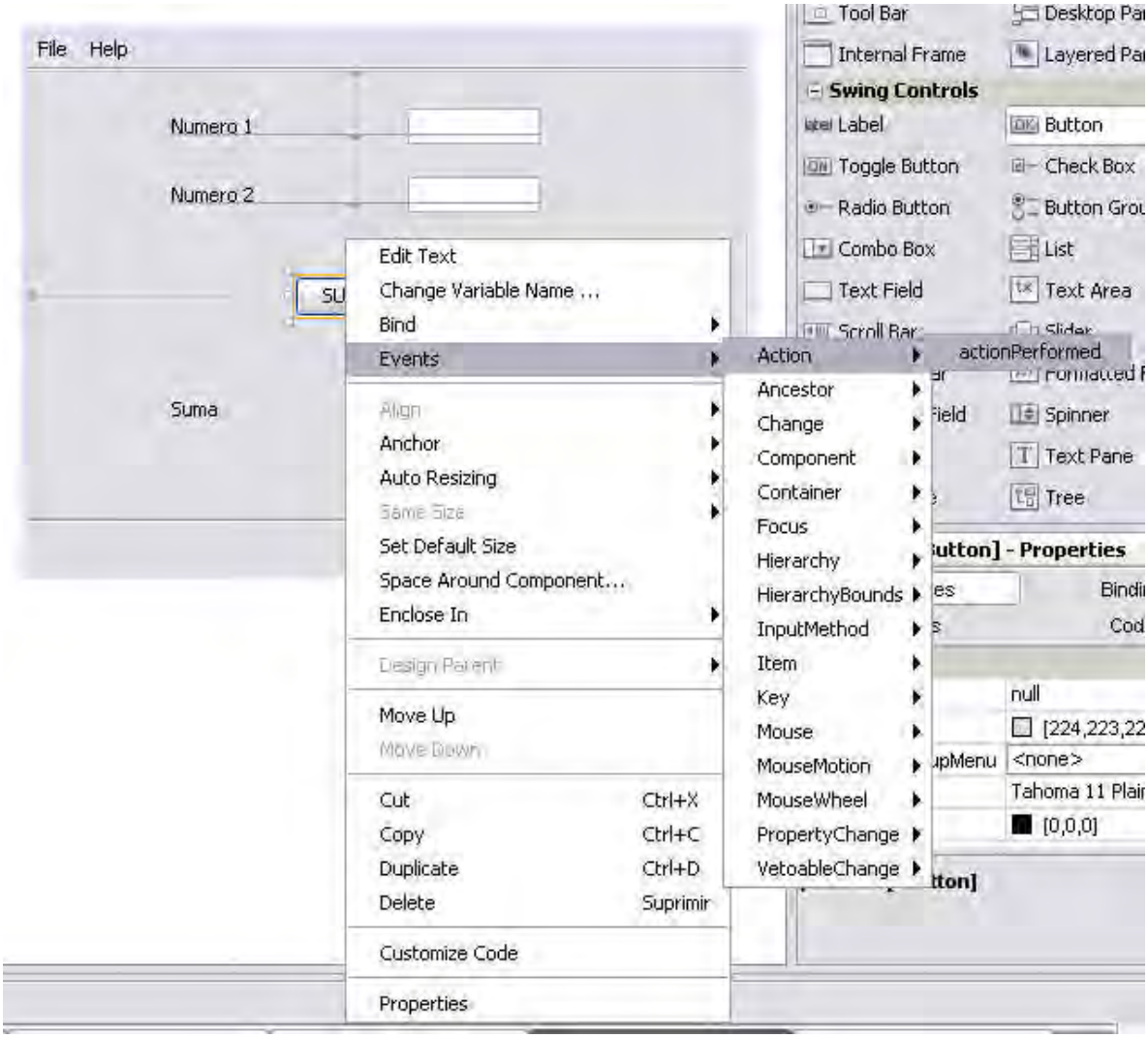
En este ejemplo se ilustra la forma de crear una interface gráfica utilizando Netbeans IDE, para poder implementar una aplicación en la que se sumen dos números enteros.


Design Preview [Taller5interfazgraficaView]
File
Help
Numero 1
Numero 2
SUMAR
Suma


File
Help
Numero 1
Numero 2
SUMAR
Suma
Edit Text
Change Variable Name ...
Bind
Events
Align
Anchor
Auto Resizing
Same Size
Set Default Size
Space Around Component...
Enclose In
Design Parent
Move Up
Move Down
Cut Ctrl+X
Copy Ctrl+C
Duplicate Ctrl+D
Delete Suprimir
Customize Code
Properties


File
Help
Numero 1
Numero 2
SUMAR
Suma
Edit Text
Change Variable Name ...
Bind
Events
Align
Anchor
Auto Resizing
Same Size
Set Default Size
Space Around Component...
Enclose In
Design Parent
Move Up
Move Down
Cut Ctrl+X
Copy Ctrl+C
Duplicate Ctrl+D
Delete Suprimir
Customize Code
Properties


File
Help
Numero 1
Numero 2
Rename
New Name:
txtA
OK
Cancel




```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
        int n1, n2, resultado;
        n1=Integer.parseInt(Numero1.getText());
        n2=Integer.parseInt(Numero2.getText());
        resultado=n1+n2;
        Resultado.setText(Integer.toString(resultado));
```

Ejemplo No. 5:

En este ejemplo se ilustra la forma de crear una interface gráfica utilizando Netbeans IDE, en la cual se implementa el Ejemplo No. 3.


```
public static void main(String[] args) {
    // TODO code application logic here
    Ventana obj = new Ventana();
    obj.setTitle("ESPE");
    obj.setVisible(true);
    obj.setBounds(100, 100, 300, 300);
}

 public class Ventana extends javax.swing.JFrame {

     /**...*/
     public Ventana() {
         initComponents();
         slider.setValue(0);
     }

     /**...*/
     @SuppressWarnings("unchecked")
     Generated Code

     private void sliderStateChanged(javax.swing.event.ChangeEvent evt) {
         // TODO add your handling code here:
         barra.setValue(slider.getValue());

     }
```

CAPITULO 6

E/S EN JAVA

ARCHIVOS

La introducción de datos a un programa o la salida de datos de un programa, se logra mediante el uso de clases del paquete java.io. Las actividades de entrada representan la lectura de los datos desde una fuente al programa. Los métodos del programa almacenan los datos en objetos o manipulan los datos para su salida inmediata. Las actividades de salida representan la escritura de los datos a partir del programa. Estos datos pueden mostrarse en una pantalla, enviarse como impresión a la impresora, o enviarse al archivo de otro programa.

Java soporta la entrada y salida de datos hacia y desde archivos y otros dispositivos tales como conexiones de red a través de flujos [streams]. Los flujos son estructuras de datos que transfieren bytes de datos. Los flujos se construyen convirtiendo objetos de una clase tipo entrada a otra clase de entrada o desde una clase tipo salida a otra. Los dos flujos principales son InputStream y OutputStream.

LA CLASE FILE

Los datos procesados por los programas se encuentran en la memoria. Estos datos no son accesibles después de ejecutado el programa. Para hacer que los datos persistan más allá del programa, los programadores almacenan los datos en archivos. Almacenar datos en archivos puede ocurrir secuencialmente o a través de técnicas de acceso aleatorio.

Para almacenar y leer datos de un archivo en orden aleatorio, se utilizan la clase RandomAccess (AccesoAleatorio) y sus métodos. Para almacenar datos en un archivo secuencialmente, el lenguaje Java utiliza objetos de software conocidos como flujos [streams].

Java proporciona la clase File, en la cual se almacena información acerca de un archivo o directorio. La clase File proporciona funcionalidad para navegar por el sistema de archivos local. El objeto File describe los directorios, archivos y estado de acceso de estos archivos. La clase File no crea un archivo ni agrega datos a un archivo.

Las clases que implementan flujos o los objetos RandomAccessFile (ArchivoAccesoAleatorio), proporcionarán la facilidad para leer y escribir en archivos.

La información para un archivo incluye un nombre de ruta y un nombre de archivo. El nombre de ruta es el directorio en el cual está localizado el archivo. La información de directorio para un archivo puede consistir en el nombre de ruta absoluto o relativo.

- La línea 1 describe el directorio del archivo como directorio relativo y el nombre del archivo como fileIO.java. El constructor File que acepta dos cadenas se utiliza en este ejemplo.
- La línea 2 describe el nombre de archivo como una cadena.
- La línea 3 describe un directorio como ruta absoluta incluyendo la unidad y la referencia al directorio raíz.
- La línea 4 utiliza el constructor que acepta un objeto File y un objeto String.

```
1. File f1 = new File("Chapter13","FileIO.java");
2. File f2 = new File("FileListing.java");
3. File f3 = new File ("c:\\Cisco");
4. File f4 = new File(f3,"FileNames.java");
```

La clase File tiene varios métodos útiles para localizar y borrar archivos y crear directorios.

public String **getName**()	Returns the name of file or directory
public String **getParent**()	Returns the name of the directory that contains the file
public String **getAbsolutePath**()	Returns the absolute path (not relative) to the file
public String **getCanonicalPath**() throws IOException	This is similar to the absolute path, but the symbols of . and .. are resolved. These symbols . represent the current directory and .. the parent directory.
public boolean **canRead**()	Is true if the file or directory may be read
public boolean **canWrite**()	Is true of the file can be written to
public boolean **exists**()	True if the file or directory exists
public boolean **isDirectory**()	True if the argument references a directory and false otherwise

public boolean **isFile**()	True if the argument references a file
public long **length**()	Length of file
public boolean **delete**()	deletes the file and returns true of operation is successful
public String[] **list**()	Returns an array of files and directories within the File
public String[] **list**(FilenameFilter filter)	Returns an array of files and directories within the File, that match the filter defined by the FileNameFilter object. An example of a filter is *.java, the filter is the string *.java which represents a pattern of all files ending with the string .java

Se debe crear un objeto File para contener la información acerca de un archivo, y luego verificar la existencia del archivo antes de leer o escribir en el mismo.

Un objeto File es inmutable, esto significa que una vez que se crea el nombre de ruta abstracto representado por un objeto File, éste nunca cambiará. El objeto File sólo puede señalar hacia el archivo descrito en el constructor.

```
File f = new File("a:\Sample.txt");
File d = new File("a:Student.java"):
f = d;     //not permissible.  The object reference are final
```

Ejemplo No. 1:

En el siguiente ejemplo se ilustra la creación de objetos File y la utilización de algunos de sus métodos.

```
import java.io.*;

public class Main {

    public static void main(String[] args) {
        File f1 = new File("build.XML");
        File f2 = new File("dos.pdf");

        System.out.println("Nombre del archivo: "+f1.getName());
        if(f1.exists())
            System.out.println("El archivo existe");
        else
            System.out.println("El archivo no existe");

        System.out.println("Nombre del archivo: "+f2.getName());
        if(f2.exists())
            System.out.println("El archivo existe");
        else
            System.out.println("El archivo no existe");
    }
}
```

Ejemplo No. 2:

En el siguiente ejemplo se ilustra la creación de objetos File y la utilización de algunos de sus métodos.

```
import java.io.*;

public class NewMain {
    public static void main(String[] args) {
        //File f = new File("c:\\WINDOWS");
        File f = new File("c:\\bar.EMF");
        File [] archivos;

        if(f.exists() & f.isDirectory() )
        {
            archivos = f.listFiles();
            System.out.println("El direcctorio contiene");
            for(int x=0;x<archivos.length;x++)
            {
                System.out.println(archivos[x]);
            }
        }
        else
        {
            System.out.println("El nombre del archivo: "+f.getName());
            System.out.println("La ruta es: "+f.getAbsolutePath());
        }
    }
}
```

STREAM (FLUJO)

La mayor parte de la entrada y salida de datos hacia y desde un programa se manipula secuencialmente. Excepto en el caso de RandomAccessFiles, Java crea objetos de software conocidos como streams. Estos streams manipulan entradas y salidas de datos.

Dos objetos Stream que hasta ahora se han utilizado son el objeto System.out y el objeto System.in. El objeto System.out imprime datos al monitor, y el objeto System.in se utiliza para leer datos de entrada desde el teclado.

Un stream puede considerarse como un flujo de bytes de datos desde una fuente a un sumidero. La fuente es el programa y el sumidero podría ser una cañería hacia otro programa, un archivo o la impresora. El término sumidero se refiere al concepto de que los datos que han alcanzado su destino no pueden procesarse por conexión a otro stream. El proceso de datos se lleva a cabo en una única dirección.

Un stream que inicia un flujo de datos se denomina stream de entrada. Éste es el stream fuente.

Un stream que termina el flujo de datos se denomina stream de salida. Éste es el stream sumidero.

Normalmente, el programa de un usuario se encontrará en un extremo u otro del stream. En un stream de salida, el programa será la fuente de los datos mientras que un archivo o impresora, u otro programa será el sumidero.
En un stream de entrada, la fuente de los datos será un archivo o entrada desde el teclado, y el programa del usuario será el destino para estos datos.

Los streams de entrada sirven para leer datos, y los streams de salida para escribirlos. El usuario puede leer desde un stream de entrada pero no escribir en él. El usuario puede escribir en un stream de salida pero no puede leer a partir de él.

Las clases de streams visualizan la entrada y la salida como una secuencia de bytes.

Java utiliza dos niveles de una estructura de clase para la lectura y escritura de streams. Las clases de stream de bajo nivel leen y escriben datos como bytes (streams de bytes) o exclusivamente caracteres Unicode (streams de caracteres).

Los streams anteriores manipulan los datos como una secuencia de bytes. Pueden leer y escribir directamente en los dispositivos. El más simple stream es el FileInputStream que lee desde un archivo y el FileOutputStream que escribe en un archivo. El constructor para ambas clases acepta una String como nombre de archivo o la referencia a un objeto archivo.

Los lectores y escritores son como los streams de entrada y salida. Estas variedades de bajo nivel se comunican con dispositivos de I/O. Estos streams están exclusivamente orientados a los Caracteres Unicode. El StringReader y el StringWriter leen y escriben strings.

SERIALIZACION

Java proporciona clases que pueden leer y escribir un objeto y sus datos en forma de unidad cohesiva. Un objeto puede almacenar datos primitivos o referencias a otros objetos. Una clase puede almacenar datos estáticos (primitivos o referencias).

La lectura y escritura de objetos utilizando streams requiere la comprensión del concepto de serialización. La serialización es el proceso de dividir un objeto y escribirlo. La serialización sólo envía los datos.

Los datos estáticos no se serializan, la definición de la clase no se serializa, y cualquier campo marcado con la palabra clave transient (transitorio) no se serializa. Si valores de datos específicos de un objeto no debieran guardarse en un archivo, entonces el atributo debe marcarse como transient.

El objeto serializado almacena un árbol de los datos que forman el objeto. El objeto es utilizado por la JVM para leer los datos serializados y construir el objeto y todos los objetos a los cuales hace referencia.

Para que una clase pueda utilizar la serialización, debe implementar la interface “Serializable”. Serializable no define campos ni métodos, tan solo es un identificador de

que el objeto se puede serializar. Si una clase es serializable, entonces todas sus subclases lo son también.

```
public class Miclase implements Serializable
{ … }
```

Sólo los objetos etiquetados como Serializable pueden serializarse. Por lo tanto, si la clase hace referencia a objetos de otra clase que no sea Serializable, entonces enviar estos objetos no resultará automáticamente en el almacenamiento de datos de los objetos a los cuales se hace referencia.

La clase responsable de escribir los objetos a streams es ObjectOutputStream. El constructor de esta clase tiene la siguiente sintaxis:

```
ObjectOutputStream(OutputStream SALIDA);
```

El argumento es el flujo de salida sobre el cual el objeto serializado será escrito.
La clase responsable de leer los objetos desde un stream es ObjectInputStream. El constructor de esta clase tiene la siguiente sintaxis:

```
ObjectInputStream(InputStream ENTRADA);
```

El argumento es el flujo de entrada desde el cual el objeto serializado se lee.

Ejemplo No. 3:

En el siguiente ejemplo se ilustra la socialización de un objeto, además la forma de leer objetos serializados.

```
import java.io.*;

public class Nombre implements Serializable{
    String nombre;
    Nombre (String nombre)
    {
        this.nombre=nombre;
    }
    Nombre ()
    {
        this.nombre="";
    }
    String ver()
    {
        return nombre;
    }
}


import java.io.*;

public class Main {

    public static void main(String[] args) throws IOException{
        Nombre obj = new Nombre("ESPE");
        FileOutputStream archivo = new FileOutputStream("datos.txt");
        ObjectOutputStream dato = new ObjectOutputStream(archivo);
        dato.writeObject(obj);
        dato.close();
}
}
```

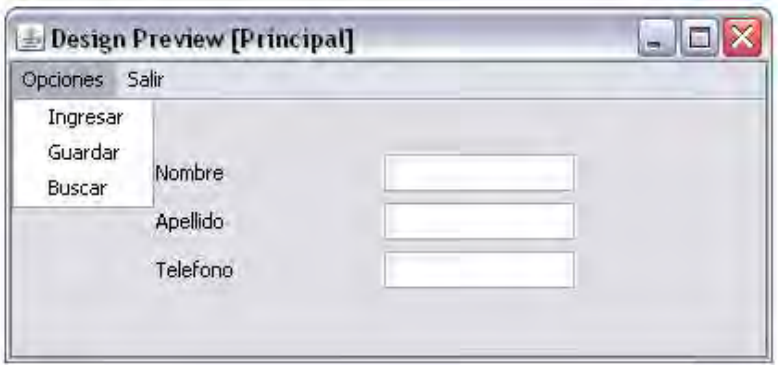
```
import java.io.*;

public class NewMain {

    public static void main(String[] args){

        try{
            FileInputStream archivo2 = new FileInputStream("datos.txt");
            ObjectInputStream dat = new ObjectInputStream(archivo2);
            Nombre aux = (Nombre) dat.readObject();
            System.out.println(aux.nombre);
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }

}
```

ACTIVIDADES DE REELABORACION

ACTIVIDADES PRÁCTICAS

1.- Implemente una interface gráfica que permita serializar un objeto en el que se guarden los valores de Rojo, Azul y Verde que definen un color que toma la interface. Además debe existir la posibilidad de extraer los valores de Rojo, Verde y Azul desde un objeto serializado para realizar el cambio de color.

2.- Implemente una interface gráfica que permita ingresar un número indeterminado de usuarios, y a partir de estos generar un arreglo de objetos los cuales serán serializados. Además existe la posibilidad de un arreglo de objetos desde un archivo y buscar si se encuentra un determinado usuario.



CAPITULO 7

THREADS

THREADS

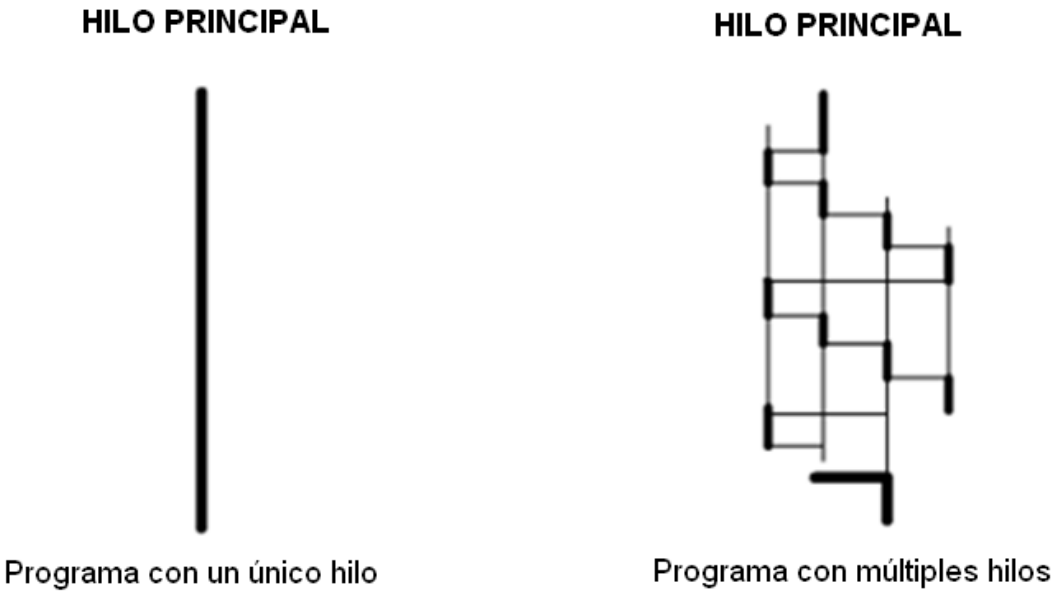
Es común hoy en día que las computadoras personales de escritorio estén compilando un programa, imprimiendo un archivo, y recibiendo mensajes de correo electrónico a través de la red, todo de manera concurrente.

Los lenguajes de programación generalmente proporcionan sólo un simple conjunto de estructuras de control. Estas estructuras de control permiten a los programadores llevar a cabo una acción a la vez y después proceder a la siguiente acción una vez que la anterior se ha finalizado.

Una visión simplista de una computadora es que ésta tiene una CPU que lleva a cabo los cálculos, memoria de sólo lectura (ROM) que contiene al programa que ejecuta la CPU, y memoria de acceso aleatorio (RAM) que contiene los datos a partir de los cuales opera el programa. En esta visión, hay un único trabajo llevándose a cabo; la CPU maneja los datos y el código de un programa secuencialmente. Sólo una tarea es llevada a cabo en la CPU. Los programas de la CPU se ejecutan como un único conjunto de instrucciones secuenciales, comenzando con el método principal.

La JVM crea una única "CPU virtual" para este programa. Este programa se ejecuta como una serie única de instrucciones. Java permite especificar que los programas contengan series de ejecución. Cada serie describe el código y los datos que se utilizarán para crear una CPU virtual. El programador designa una porción del programa para que se ejecute concurrentemente con otras series. Esta capacidad se denomina multi-threading.

Si se traza una línea a través del código para rastrear cómo se desplaza el control de expresión a expresión a medida que se ejecuta el programa, el rastreo sigue una serie de ejecución. Todos los programas cuentan con al menos una serie. Un programa multi-thread permite que más de una serie se ejecute a lo largo del código al mismo tiempo.



La Máquina Virtual Java (JVM) es un sistema multi-thread. Es decir, es capaz de ejecutar varios programas simultáneamente. La JVM gestiona todos los detalles, asignación de tiempos de ejecución, prioridades, etc, de forma similar a como gestiona un Sistema Operativo múltiples procesos. La diferencia básica entre un proceso de Sistema Operativo y un Thread Java es que los Threads corren dentro de la JVM, que es un proceso del Sistema Operativo y por tanto comparten todos los recursos, incluida la memoria y las variables y objetos allí definidos.

Java soporta el concepto de Thread desde el mismo lenguaje, con algunas clases e interfaces definidas.

Los threads son útiles porque permiten que el flujo del programa sea divido en dos o más partes, cada una ocupándose de alguna tarea. Por ejemplo un Thread puede encargarse de la comunicación con el usuario, mientras otros actúan en segundo plano, realizando la transmisión de un fichero, accediendo a recursos del sistema (cargar sonidos, leer ficheros ...), etc.

Existen dos formas de definir una clase para que pueda ejecutarse concurrentemente con otras clases.

Extendiendo la clase **Thread**

```
public class Ejemplo extends Thread
```

Implementando la interfaz **Runnable**

```
public class Ejemplo2 implements Runnable
```

La interfaz Runnable únicamente contiene el método **run().** Cuando un objeto que implementa esta interfaz se usa para crear un thread, al arrancar se ejecuta automáticamente el método run.

La clase Thread implementa la interfaz Runnable y contiene un variado grupo de constructores y métodos que permiten definir el comportamiento y evolución de los programas concurrentes.

Empleando la clase Thread

```
public class ThreadEjemplo extends Thread
{
        public void run()
        {  }
}
```

Creación y ejecución de un Thread

```
....
public static void main (String [] args)
{
        ThreadEjemplo  A =new ThreadEjemplo();
        A.start();
}
```

Empleando la interfaz Runnable

```
public class RunnableEjemplo implements Runnable
{
        public void run()
        {   }
}
```

Creación y ejecución de un Thread con Runnable

```
...
public static void main (String [] args)
{
        RunnableEjemplo  A =new RunnableEjemplo();
        new Thread(A).start();
}
```

Para ejecutar un Thread se debe utilizar el método start, que a su vez ejecuta al método run.

Ejemplo No. 1:

En el siguiente ejemplo se ilustra la creación de un Hilo imprentando la interfaz Runnable

```
public class Ejemplo1 implements  Runnable{
    String Frase;
    Random Aleatorio;
    public Ejemplo1(String Frase)
    {
        this.Frase = Frase;
        Aleatorio  = new Random();
    }
    public void run()
    {
        try
        {
            do
            {
                int aux=Math.abs(Aleatorio.nextInt())%1000;
                System.out.println(Frase);
                Thread.sleep(aux);
            }while(true);
        }
        catch(Exception e)
        { }
    }
}


 public class Main {

     public static void main(String[] args) {
         // TODO code application logic here
         Ejemplo1 A = new Ejemplo1("HOLA");
         Thread   B = new Thread(A);
         B.start();
     }
 }
```

Ejemplo No. 2:

En el siguiente ejemplo se ilustra la creación de un Hilo imprentando la interfaz Runnable

```
public class Ejemplo2 implements  Runnable{
    String Frase;
    public Ejemplo2(String Frase)
    {
        this.Frase = Frase;
    }
    public void run()
    {
        try
        {
            System.out.println(Frase);
        }
        catch(Exception e)
        { }
    }
}
public class Main {

    public static void main(String[] args) {
        String Frase = "HOLA";
        Thread [] Hilo = new Thread [Frase.length()];
        for(int x=0;x<Frase.length();x++)
        {
            Hilo[x] = new Thread(new Ejemplo2(Frase.substring(x,x+1)));
            Hilo[x].start();
        }
    }
}
```

Ejemplo No. 3:

En el siguiente ejemplo se ilustra la creación de un Hilo extendiendo la clase Thread.

```
public class Hilo1 extends Thread{

          public Hilo1()
          {
              this.start();
              System.out.println("Hilo 1");
          }
          public void run()
          {
               Hilo2 H2 = new Hilo2();
          }

}
public class Hilo2 extends Thread{

         public Hilo2()
         {
             this.start();
         }
         public void run()
         {
             try{
                 this.sleep(2000);
                 System.out.println("Hilo 2");
             }
             catch(Exception e){}

         }
}

public class Main {

     /**
      * @param args the command line arguments
      */
     public static void main(String[] args) {
         // TODO code application logic here
         new Hilo1();
     }

}
```

Ejemplo No. 4:

En el siguiente ejemplo se ilustra la creación de un Hilo extendiendo la clase Thread.

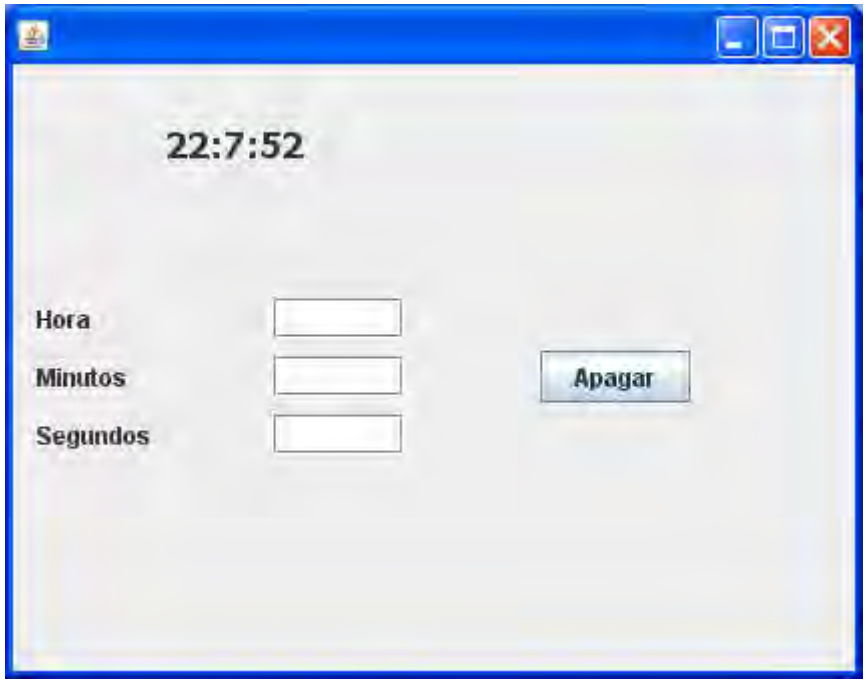
```
public class Hilo1 extends Thread{

         public Hilo1()
         {
              this.start();
              System.out.println("Hilo 1");
         }
         public void run()
         {
               Hilo2 H2 = new Hilo2();
         }

}
public class Hilo2 extends Thread{
        int i=0;
        public Hilo2()
        {
            this.start();
        }
        public void fin()
        {
            System.out.println("Fin Hilo 2");
            this.stop();
        }
        public void run()
        {
            System.out.println("Hilo 2");
            while(true)
            {
                try{
                    this.sleep(500);
                    i++;
                    Ventana.txtArea.setText(Integer.toString(i));
                }
                catch(Exception e){}
                if(i==5) fin();
            }
        }
}
```


Design Preview [Ventana]
Contar

ACTIVIDADES DE REELABORACION

ACTIVIDADES PRÁCTICAS

1.- Desarrolle una aplicación en Java que permita visualizar la hora del sistema y que permita establecer la hora a la cual la aplicación debe cerrarse.



CAPITULO 8

COMUNICACIONES CON LA PC

API JAVA COMMUNICATIONS

El API Java Communications (COMM) es un paquete opcional para la plataforma Java 2. Proporciona soporte para comunicación con dispositivos periféricos a través de los puertos serie y paralelo.

Es un API especial en el sentido de que aunque está bien definido multi-plataforma, debe descargarse una versión específica de las librerías COMM para utilizarlo. Implementa los estándares RS-232 para puerto serie y IEEE-1284 para el puerto paralelo.

El API COMM no incluye soporte para comunicación sobre el puerto Universal Serial Bus (USB) .

La instalación del API de Comunicaciones de java para Windows, se la puede realizar de la siguiente manera:

comm.jar	…\jdk1.6\jre\lib\ext
win32com.dll	…\jdk1.6\bin
javax.comm.properties	…\jdk1.6\jre\lib

Además si existiera algún error, se debe copiar:

win32com.dll	…\WINDOWS\system32

El paquete proporciona soporte para dispositivos serie y paralelo utilizando streams y eventos. El API de Comunicaciones Java, permite transmitir y recibir datos a través de dispositivos conectados al puerto serie o paralelo; proporcionando además un conjunto de opciones que permiten la configuración de todos los parámetros asociados a estos puertos.

En el paquete de comunicaciones javax.comm existe una serie de clases que permiten tratar varios niveles de programación:

- Nivel alto: En este nivel tiene las clases CommPortIdentifier y CommPort que permiten el acceso a los puertos de comunicación
- Nivel medio: Con las clases SerialPort y ParallelPort cubre las interfaces físicas RS-232 para el puerto serie y IEEE 1284 para el puerto paralelo.
- Nivel bajo: Este nivel se enlaza con el sistemaoperativo y en el se encuentra el desarrollo de drivers.

Los servicios que proporciona este paquete son:

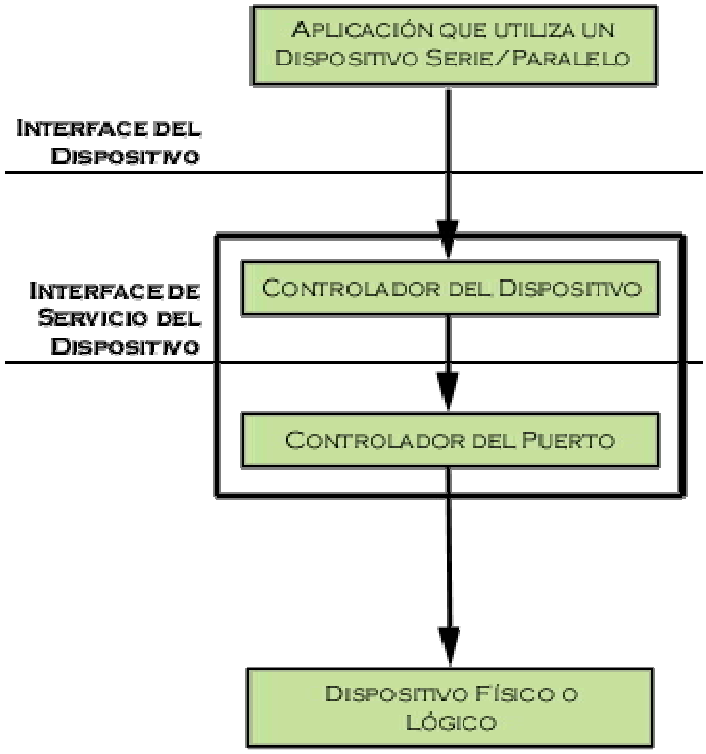
- Poder obtener los puertos disponibles así como sus características.
- Abrir y mantener una comunicación en los puertos. Esto permite tener varias aplicaciones Java funcionando a la vez.
- Resolver colisiones entre aplicaciones.

Las clases disponibles para el manejo de puertos son:

- *CommPort*
- *CommPortIdentifier*
- *ParallelPort*
- *SerialPort*

COMMPORT

Esta es una clase abstracta que describe los métodos comunes de comunicación y serán las clases que heredan de ellas(SerialPort y ParallelPort) la que añaden métodos y variables propias del tipo del puerto.



getPortIdentifiers().- Este método entrega un enumerado con tantos objetos CommPortIdentifier como puertos se disponga.

getPortType().- Devuelve un entero que informa el tipo del puerto (serie o paralelo), se dispone de las constantes PORT_PARALLEL Y PORT_SERIAL.

isCurrentOwned().- Informa si esta libre o no el puerto, en el caso de que esté ocupado se puede saber quien lo está utilizando mediante el método getCurrentOwner().

open(String, int).- Abre y por lo tanto reserva un puerto. Los parámetros son un String con el nombre de la aplicación que reserva el puerto y un int que indica el tiempo de espera para abrir el puerto.
En el caso de que se intente abrir un puerto que este siendo utilizado saltará la excepción PortInUseException.

close().- Permite liberar el puerto que se reservó con open, este notificará el cambio de dueño a las clases que se hubiesen registrado con el método addPortOwnershipListener.

getOutputStream().- Permite enlazar la salida del puerto al OutputStream que devuelve para poder escribir en el puerto de la misma forma que si escribiera en un fichero.

getInputStream() .- permite enlazar la entrada del puerto al InputStream para leer del puerto.

COMMPORTIDENTIFIER

Administra la comunicación con los puertos. Controla el acceso a los puertos y determina los puertos disponibles y los ocupados por que dispositivos y maneja eventos de cambios sobre el estado.

CLASE PARALLELPORT

En esta clase se tiene la interfaz de ajo nivel del puerto paralelo que cumple la norma IEEE 1284. Este estándar permite trabajar con 5 modos de funcionamiento. La clase ParallelPort es una clase que hereda de CommPort, y cuenta con una serie de métodos que facilitan el uso del puerto.

CLASE SERIALPORT

Corresponde a la interfase de bajo nivel del puerto serie que cumple con el estándar RS-232. La clase SerialPort hereda de la clase abstracta CommPort y por lo tanto cuenta con sus métodos pero además de estos dispone de otros métodos y variables específicas para el tratamiento de los puertos serie.

setSerialPortParam(int, in, int, int).- Permite configurar los parámetros del puerto serie.
Este método utiliza la excepción *UnsupportedCommOperationException* en el caso de que los valores no sean soportados.
Los parámetros son:

- La velocidad
- Bits de datos, para indicar el valor se utiliza las constantes de la clase (DATA_5, DATA_6, DATA_7, DATA_8)

- Bit o bits de stop, que puede ser 1, 2 o 1,5. Las constantes que definen estas configuraciones son: STOPBITS_1, STOBITS_2 y STOPBIT_1_5.
- Paridad, que puede ser PARITY_NOME en el caso de no utilizar paridad, PARITY_ODD para la paridad impar, PARITY_EVEN paridad par, PERITY_MARK paridad por marca y PARITY_SPACE paridad por espacio.

INTERFACES

Las interfaces disponibles para el manejo de puertos son:

CommDriver .- Es una interfaz para el reconocimiento de los dispositivos que se encarga de asegurarse que el hardware se encuentre, de cargar las librerías base, registrar el nombre de los puertos y agregar o retornar la instancia que extiende de los puertos serial o paralelo.

CommPortOwnershipListener .- Extiende de la clase EventListener. Es la interfaz encargada de comunicar los eventos de la actividad del puerto, que se encuentre abierto o cerrado y la recepción de mensajes.

ParallelPortEventListener .- Extiende de la clase EventListener. Es la interfaz encargada de recibir los eventos del puerto paralelo y propagar el evento generado.

public void **parallelEvent(ParallelPortEvent ev)**

SerialPortEventListener .- Extiende de la clase EventListener. Es la interfaz encargada de recibir los eventos del puerto serial y propagar el evento generado.

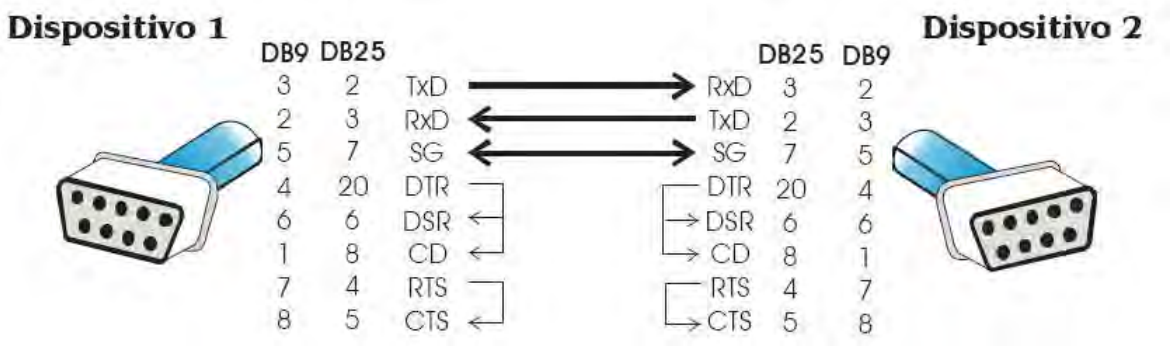
public void **serialEvent(ParallelPortEvent ev)**

EXCEPCIONES

Las excepciones disponibles para el manejo de puertos son:

- ***javax.comm.NoSuchPortException.-*** Esta excepción se presenta cuando el driver no puede encontrar un puerto especificado.
- ***javax.comm.PortInUseException.-*** Esta excepción se presenta cuando el puerto especificado esta en uso.
- ***javax.comm.UnsupportedCommOperationException.-*** Esta excepción se presenta cuando el driver no permite la operación especificada.

NOTA:
Para la realización de los siguientes ejemplos se debe poseer un cable con la siguiente configuración.

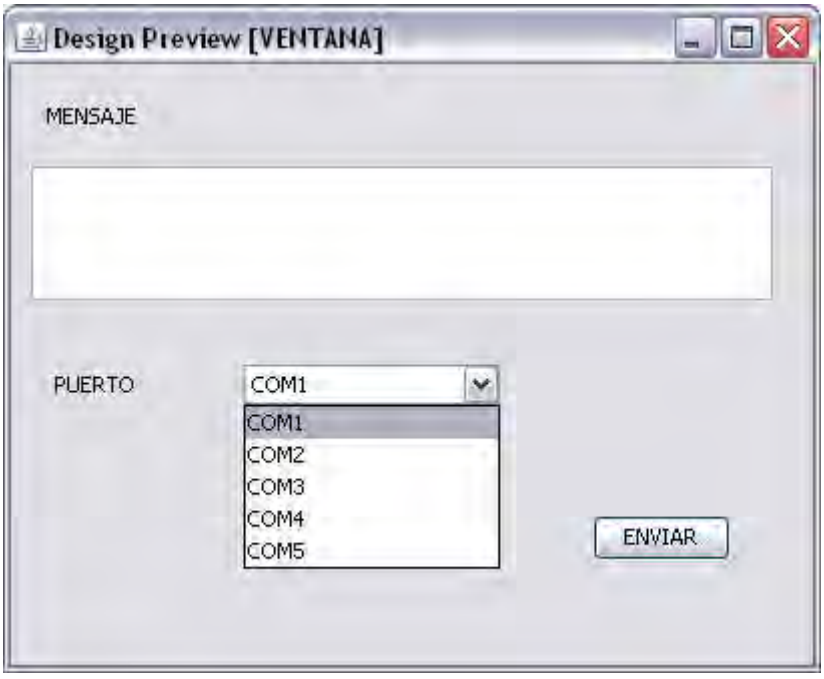


Ejemplo No. 1:

En este ejemplo se ilustra cómo se puede enviar un mensaje por el puerto serie.

```
import java.io.*;
import javax.comm.*;

public class PUERTOSERIE {
    CommPortIdentifier idPuerto;
    SerialPort         puertoSerie;
    OutputStream       salida;
```

```
    void txSerial(String mensaje, String puerto)
    {
        try{
            idPuerto=CommPortIdentifier.getPortIdentifier(puerto);
            System.out.println("Puerto "+puerto+" identificado");
            if(idPuerto.isCurrentlyOwned())
            {
                System.out.println("Puerto en uso");
            }
            else
            {
                puertoSerie=(SerialPort) idPuerto.open("ESCRITURA",2000);
                System.out.println("Puerto abierto");
                salida=puertoSerie.getOutputStream();
                puertoSerie.setSerialPortParams(9600,SerialPort.DATABITS_8,
                        SerialPort.STOPBITS_1,SerialPort.PARITY_NONE);
                System.out.println("Puerto configurado");
                salida.write(mensaje.getBytes());
                System.out.println("Mensaje enviado");
                puertoSerie.close();
                System.out.println("Puerto liberado");
            }
        }
        catch(Exception e)
        {
        System.out.println("ERROR EN LA COMUNICACION:"+e);
        }
     }
}
```



```
public class VENTANA extends javax.swing.JFrame {
    PUERTOSERIE ob= new PUERTOSERIE();
    /** Creates new form VENTANA */
    public VENTANA() {
        initComponents();
    }
```

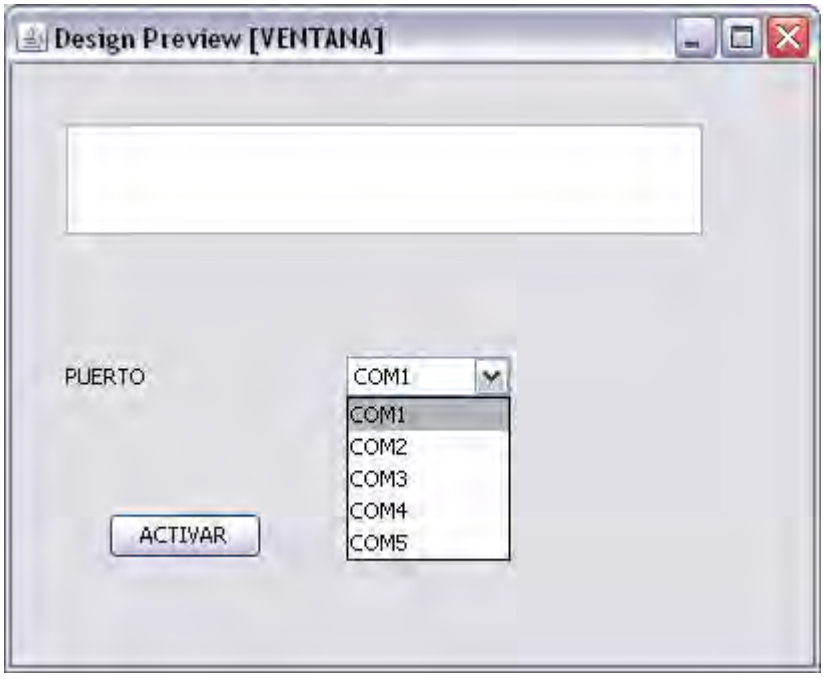
```
private void btnENVIARActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    ob.txSerial("\r"+txtMENSAJE.getText()+"\n", String.valueOf(Puertos.getSelectedItem()));
}
```

Ejemplo No. 2:

En este ejemplo se ilustra cómo se puede recibir un mensaje por el puerto serie.

```
import java.io.*;
import javax.comm.*;

public class PUERTOSERIE extends Thread{
    CommPortIdentifier  idPuerto;
    InputStream         entrada;
    SerialPort          puertoSerie;
    int                 flag    = 0;
    byte[]              buffer = new byte [1024];
    int                 len     = -1;
    String              texto1 = "";
    String              aux     = "";
```

```
void rxSerial(String puerto)
{
    try{
        idPuerto=CommPortIdentifier.getPortIdentifier(puerto);
        System.out.println("Puerto "+puerto+" identificado");
        if(idPuerto.isCurrentlyOwned())
        {
            System.out.println("Puerto en uso");
        }
        else
        {
            puertoSerie=(SerialPort) idPuerto.open("LECTURA",2000);
            System.out.println("Puerto abierto");
            puertoSerie.setSerialPortParams(9600,SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1,SerialPort.PARITY_NONE);
            System.out.println("Puerto configurado");
            entrada = puertoSerie.getInputStream();
            this.start();
        }
    }
    catch(Exception e)
    {
        System.out.println("ERROR EN LA COMUNICACION:"+e);
    }
}
  public void run()
  {
      try{
          while(true)
          {
              if((len = entrada.read(buffer))>-1)
              {
                  texto1 = new String(buffer,0,len);
                  aux=aux+texto1;
                  VENTANA.txtMENSAJE.setText(aux);
              }
          }
      }
      catch(Exception e)
      {
          System.out.println("ERROR EN LA COMUNICACION:"+e);
      }
  }
}
```

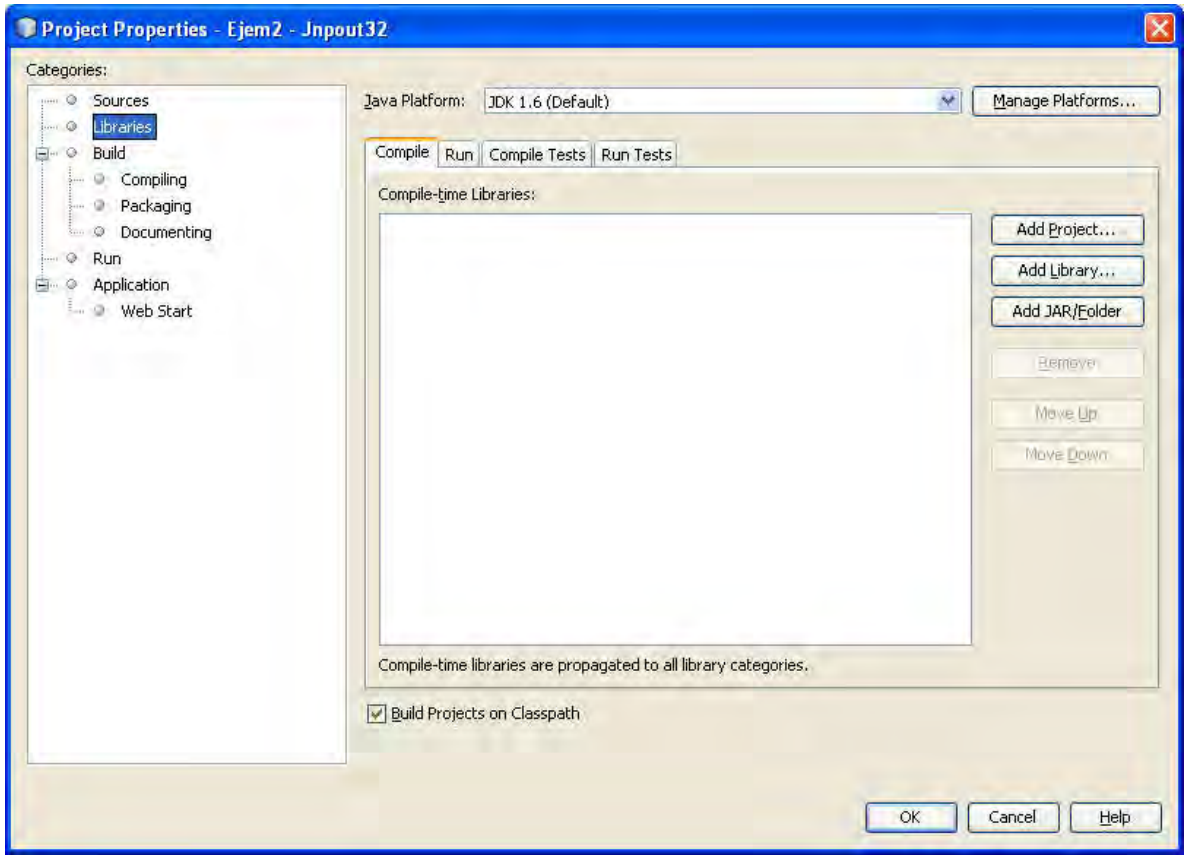


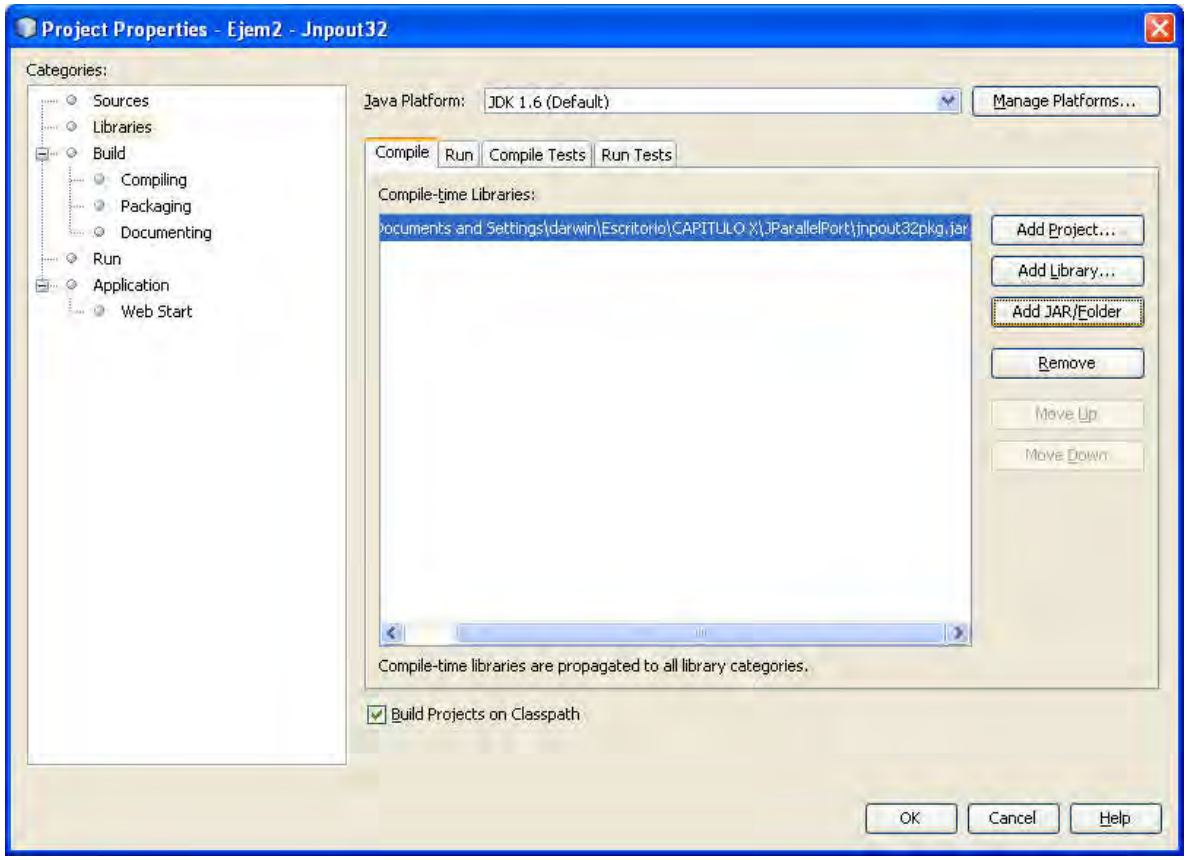

```
public class VENTANA extends javax.swing.JFrame {
    PUERTOSERIE ob = new PUERTOSERIE();
    /** Creates new form VENTANA */
    public VENTANA() {
        initComponents();

    }
```

```
private void btnACTIVARActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    ob.rxSerial(String.valueOf(Puertos.getSelectedItem()));
}
```

OTRAS LIBRERIAS PARA MANEJO DE PUERTOS

Dentro de las propiedades del proyecto seleccionamos Libraries para poder agregar la librería jnpout32pkg.jar


Project Properties - Ejem2 - Jnpout32
Categories:
Sources
Libraries
Build
Compiling
Packaging
Documenting
Run
Application
Web Start
Java Platform:
JDK 1.6 (Default)
Manage Platforms...
Compile
Run
Compile Tests
Run Tests
Compile-time Libraries:
Add Project...
Add Library...
Add JAR/Folder
Remove
Move Up
Move Down
Compile-time libraries are propagated to all library categories.
Build Projects on Classpath
OK
Cancel
Help


Project Properties - Ejem2 - Jnpout32
Categories:
Sources
Libraries
Build
Compiling
Packaging
Documenting
Run
Application
Web Start
Java Platform:
JDK 1.6 (Default)
Manage Platforms...
Compile
Run
Compile Tests
Run Tests
Compile-time Libraries:
Documents and Settings\darwin\Escritorio\CAPITULO X\JParallelPort\jnpout32pkg.jar
Add Project...
Add Library...
Add JAR/Folder
Remove
Move Up
Move Down
Compile-time libraries are propagated to all library categories.
Build Projects on Classpath
OK
Cancel
Help

Se debe copiar el archivo jnpout32pkg.dll dentro de la carpeta de Windows:

jnpout32pkg.dll WINDOWS\system32

Para poder utilizar esta librería es indispensable crear un objeto de la clase pPort, ademas se debe importar la librería

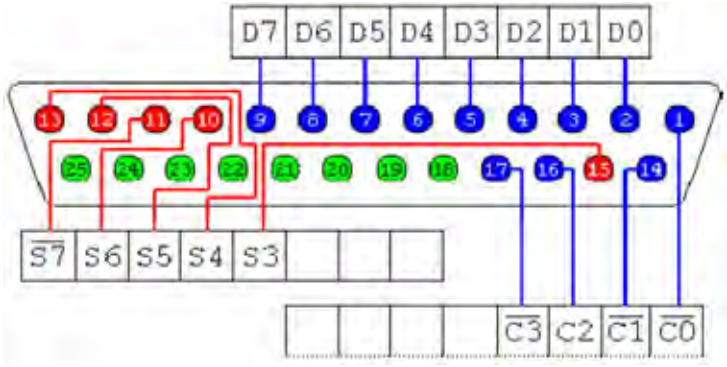
```
import jnpout32.*;
pPort puerto= new pPort();
```

El direccionamiento permite seleccionar que puerto se quiere utilizar. El puerto paralelo vá desde 0x378 hasta 0x37F. Estas direcciones establecen el uso que va a tener el puerto.

VARIABLE	DESCRIPCIÓN
Direccion 378	Corresponde al bus de datos del puerto paralelo.
Direccion 379	Corresponde al bus de estado del puerto paralelo.
Direccion 37A	Corresponde al bus de control del puerto paralelo.

El puerto paralelo está compuesto por:

- 8 pines de Datos/Salida (D0 - D7)
- 5 pines de Estado/Entrada (S3 – S7)
- 4 pines de Control/Salida (C0 – C3)
- 8 pines de Tierra (18 - 25)



El puerto paralelo es compatible con la norma SPP. Se pueden disponer de hasta tres puertos que el sistema operativo denomina LPT. Se los numera desde 1 hasta 3 quedando LPT1, LPT2 y LPT3. Como todo dispositivo, el puerto paralelo dispone de una dirección de memoria base que puede ser *378*, 278 o 3BC. Con *esta dirección el usuario se comunica.*

El puerto paralelo dispone de 8 líneas digitales de salida, (D0 a D7), cuyos niveles posibles son 0v (para bajo) y 5v (para alto), cumpliendo con la lógica TTL. Estas líneas están conectadas a los pines 2 al 9 para los bits 0 al 7 respectivamente. Estas ocho líneas conforman el **bus de datos** del puerto.

El **bus de estado** puede ser empleado para ingresar datos hacia la computadora. Este puerto, a diferencia del de datos, dispone de sólo 5 líneas digitales de entrada partiendo desde el bit 3 hasta el bit 7. Por lo general se las denomina S3 a S7. La dirección de memoria asignada para comunicarse con este bus es igual a la dirección base (la del bus de datos) más 1. O sea, si el puerto base es 378, este puerto es 379.

NOTA: El bit 7 de este puerto (conectado al terminal 11 del conector) presenta un estado lógico inverso. Esto significa que, si en el pin 11 se colocan 5v (nivel lógico alto) el bit al ser leído presentará un cero. Y, por consiguiente, si el pin 11 es puesto a tierra (nivel lógico bajo) al leer el bit se verá un uno.

El **bus de control**, al igual que el de datos, dispone de líneas de salida desde la computadora hacia el exterior. Las líneas también son digitales pero en este caso son sólo cuatro, las menos significativas que van del bit 0 al 3. Estas cuatro líneas son generalmente llamadas como C0 a C4 y se pueden controlar por medio de la dirección de memoria base (la del bus de datos) más 2. De esta forma si la dirección base es 378, este bus es 37A. Si la dirección base es 278, la dirección de este bus será 27A.

NOTA: En este bus hay tres bits cuyos niveles lógicos se encuentran invertidos (bit0, bit1 y bit 3) y sólo uno que presenta un estado lógico normal.

Cada terminal tiene una corriente entre 6mA a 20 mA, dependiendo del fabricante . Superado este miliamperio no se garantiza el nivel lógico TTL de la señal presente e incluso de sobrecargarlo demasiado se puede dañar físicamente el puerto.

El siguiente bloque de código es el que permite el envío de datos hacia la dirección 378 del puerto paralelo.

```
try
{
        short valor = (short)Integer.parseInt("3";);
        puertoParalelo.output(0x378, valor);
}
catch(Exception e)
{
        System.out.println("Error en la escritura");
}
```

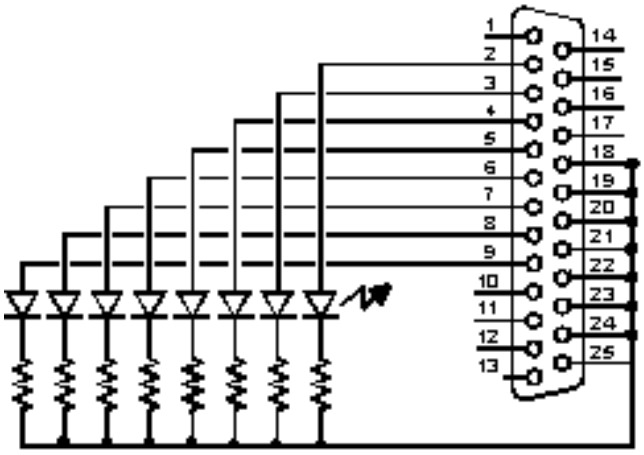
El siguiente bloque de código es el que permite la recepción de datos desde la dirección 379 del puerto paralelo.

```
try
{
        short valor = puerto.input(0x379);
        System.out.println(String.valueOf(valor));
}
catch(Exception e)
{
        System.out.println("Error en la lectura");
}
```

Ejemplo No. 3:

En este ejemplo se ilustra cómo se puede escribir un decimal por el puerto paralelo.

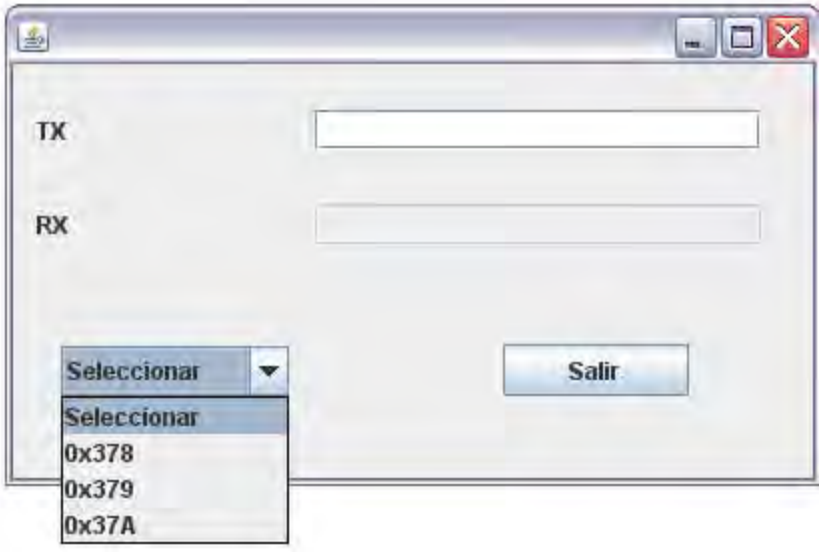
```
import jnpout32.*;

public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
            String c="3";;
            short direccion1=0x378;
            pPort puertoParalelo = new pPort();
            short valor = (short)Integer.parseInt(c);
            puertoParalelo.output(direccion1, valor);
            System.out.println(valor);
    }

}
```



Ejemplo No. 4:

Implemente una aplicación que permita escribir en el registro 0x378,0x37A y leer el registro 0x379.

```
import jnpout32.*;

public class txrxParalelo {
    short valor = 0;
    short direccion1 = 0x378;
    short direccion2 = 0x379;
    short direccion3 = 0x37A;
    pPort puerto     = new pPort();
    void txParaleloDatos(int dato)
    {
        valor = (short) dato;
        puerto.output(direccion1, valor);
    }
    int rxParaleloEstados()
    {
        valor = puerto.input(direccion2);
        return(valor);
    }
    void txParaleloControl(int dato)
    {
        valor = (short) dato;
        puerto.output(direccion3, valor);
    }
}
```



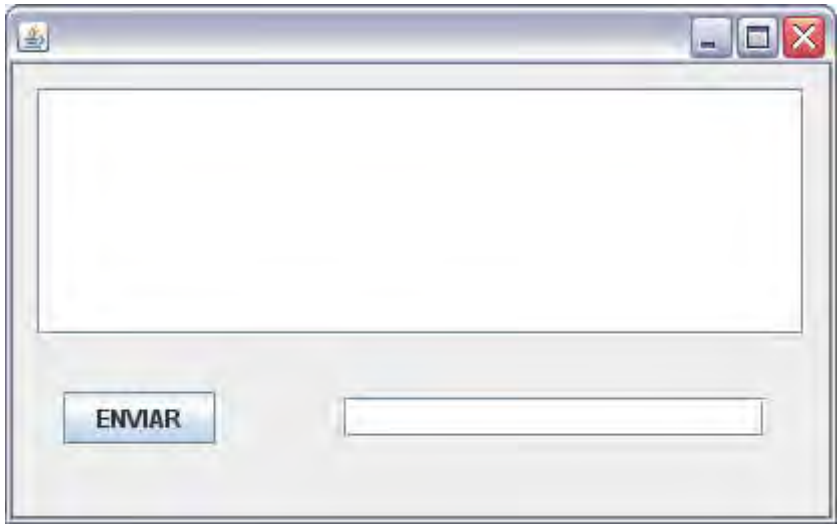
```
 */
public class VENTANA extends javax.swing.JFrame {
    txrxParalelo obj = new txrxParalelo();
    /** Creates new form VENTANA */
    public VENTANA() {
        initComponents();
    }
    private void DIRECCIONItemStateChanged(java.awt.event.ItemEvent evt) {
        // TODO add your handling code here:
        switch(DIRECCION.getSelectedIndex())
        {
            case 1:
                btnActuar.setLabel("Enviar");
                break;
            case 2:
                btnActuar.setLabel("Recibir");
                break;
            case 3:
                btnActuar.setLabel("Enviar");
                break;
            default:
                btnActuar.setLabel("Salir");
                break;
        }
    }

    private void btnActuarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        switch(DIRECCION.getSelectedIndex())
        {
            case 1:
                obj.txParaleloDatos(Integer.parseInt(txtTX.getText()));
                System.out.println("Dato enviado");
                break;
            case 2:
                txtRX.setText(Integer.toString(obj.rxParaleloEstados()));
                System.out.println("Dato recibido");
                break;
            case 3:
                obj.txParaleloDatos(Integer.parseInt(txtTX.getText()));
                System.out.println("Dato enviado");
                break;
            default:
                System.exit(0);
                break;
        }
    }
```

ACTIVIDADES DE REELABORACION

ACTIVIDADES PRÁCTICAS

1.- Implemente una aplicación en Java que permita establecer una comunicación bidireccional utilizando el puerto serie. Además considere la siguiente estructura de la clase para la definición de los métodos para transmitir y para escribir.

```
import javax.comm.*;
import java.io.*;

public class txrxSerial extends Thread{
    ...

    txrxSerial(String puerto)
    {
        ...
    }

    void txSerial(String mensaje)
    {
        ...
    }
    void rxSerial()
    {
        ...
    }

    public void run()
    {
        ...
    }
    void cerrar()
    {
        ...
    }
}
```



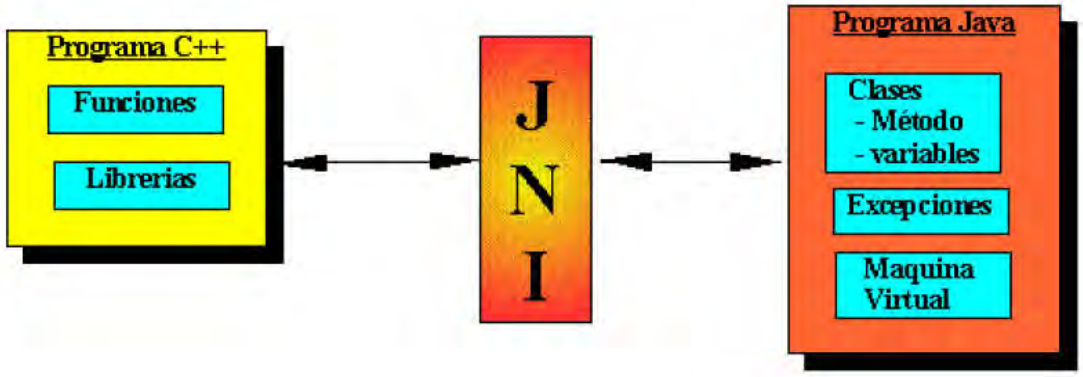
2.- Modifique la actividad anterior, de tal manera que permita transmitir un archivo JPG, de una de las máquinas a la otra.

3.- Diseñe una aplicación que permita controlar 8 leds con el puerto paralelo, de tal manera que se prendan uno a la vez.

CAPITULO 9

CÓDIGO NATIVO JAVA (JNI)

JNI es una forma de programación que permite que un programa escrito en Java ejecutado en la máquina virtual java (JVM) pueda interactuar con programas escritos en otros lenguajes como C/C++, ensamblador, etc. JNI se usa para escribir métodos nativos que permitan resolver situaciones en las que una aplicación no puede ser enteramente escrita en Java. JNI permite a un método nativo utilizar los objetos Java de la misma forma en que el propio código de Java lo hace. JNI también es utilizado para operaciones y cálculos de alta complejidad temporal, porque el código nativo es por lo general más rápido que el que se ejecuta en una máquina virtual.



LIBRERIAS DE ENLACE DINÁMICO Y ESTÁTICO

Las Librerías de Enlace Estático, son ficheros destinados a almacenar funciones, clases y variables globales y se crean a partir de varios ficheros de código objeto ".obj", estos ficheros tienen la extensión ".lib". Las funciones de la librería se incluyen dentro del ejecutable durante la fase de enlazado, con lo que una vez generado el ejecutable ya no es necesario disponer de las librerías de enlace estático.

Las Librerías de Enlace Dinámico, son ficheros cuyas funciones no se incrustan en el ejecutable durante el enlazado, sino que, en tiempo de ejecución el programa busca el fichero, carga su contenido en memoria y enlaza su contenido según sea necesario, es decir llama a las funciones. La ventaja está en que varios programas pueden compartir las mismas librerías, lo cual reduce el consumo de disco duro, estos ficheros tiene la extensión ".dll" (para el caso de Windows). Su extensión depende del sistema operativo

Sistema Operativo	Extensión
Mac OS X	`.dylib`
UNIX	`.so`
Windows	`.dll`

TIPOS DE DATOS

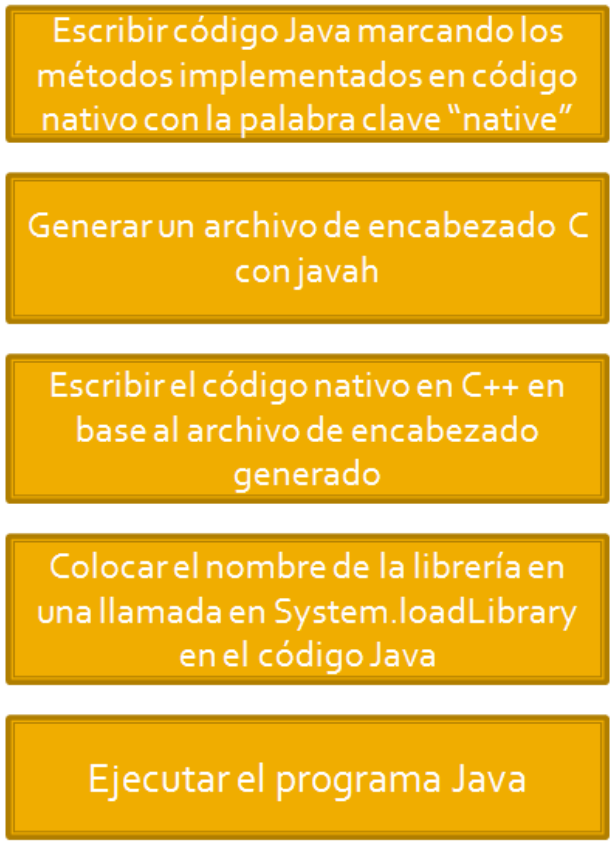
Tipos Primitivos.- Como int, float o double. Su correspondencia con tipos C es directa, ya que en el fichero jni.h se encuentran definiciones de tipos C equivalentes. La siguiente tabla muestra los tipos fundamentales Java y sus correspondientes tipos C.

Tipo Java	Tipo C	Descripción
`boolean`	`jboolean`	8 bits sin signo
`byte`	`jbyte`	8 bits con signo
`char`	`jchar`	16 bits sin signo
`short`	`jshort`	16 bits con signo
`int`	`jint`	32 bits con signo
`long`	`jlong`	64 bits con signo
`float`	`jfloat`	32 bits formato IEEE
`double`	`jdouble`	64 bits formato IEEE

Arreglos.- JNI pasa los objetos a los métodos nativos como referencia a punteros C que apuntan a estructuras internas sólo conocidas por la máquina virtual que se esté usando.

PROCESO DE CREACIÓN

El proceso de creación de un proyecto con JNI, de manera general se describe a través de los pasos especificados a continuación.



GENERACIÓN DEL ARCHIVO DE ENCABEZADO C

Se copia el archivo donde se encuentran programados lo métodos nativos dentro del **bin** del **jdk** de Java, y se procede a compilarlo con **javac** Nombre_Clase.java y a generar el archivo de cabecera con **javah** Nombre_Clase

GENERACIÓN DEL CÓDIGO NATIVO EN C++ EN BASE AL ARCHIVO DE ENCABEZADO GENERADO

Para la generación de librerías dinámicas en Windows, se ha seleccionado como compilador Microsoft Visual C++ 6.0, por lo que primeramente se debe agregar las librerías: Jni.h y Jni_md.h, ya las mismas son necesarias para la creación de código nativo en C++.

Jni.h se encuentra dentro de la carpeta de Java en la siguiente dirección:

…\Java\jdk1.6.0_03\include

Jni_md.h se encuentra dentro de la carpeta de Java en la siguiente dirección:

…\Java\jdk1.6.0_03\include\win32

Estas dos librerías deben ser copiadas dentro de la carpeta de Visual C para que se pueda compilar la dll.

...\Microsoft Visual Studio\VC98\Include

Ejemplo No. 1:

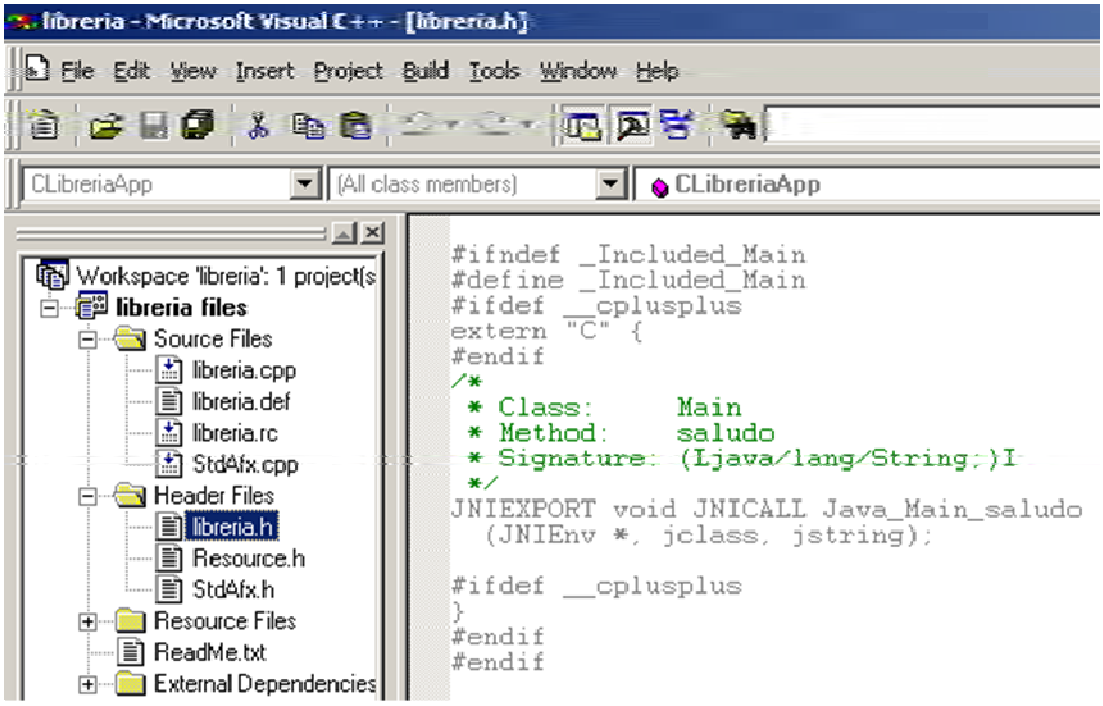
Escriba un programa en Java que utilice un método nativo el cual reciba como argumento su nombre para que sea desplegado en pantalla, utilizando la función MessageBox, contenida dentro de Visual C++ 6.0

A continuación se muestra el código que se implementa en Java

```
public class Main {

    static native void saludo(String nombre);
    static{
    System.loadLibrary("libreria");
    }
    public static void main(String[] args) {
        saludo("Darwin Alulema");
    }
}
```

*El método saludo no tiene cuerpo porque será añadido mediante una biblioteca nativa denominada **librería**. La biblioteca nativa **librería** es cargada mediante la sentencia loadLibrary(), sentencia que ha sido incluida como static para que sea ejecutada cada vez que se cree una instancia de esta clase.*

A continuación se ubica el archivo de cabecera del proyecto creado en Visual C++, llamado ***libreria.h*** *para proveer los prototipos de las funciones.*



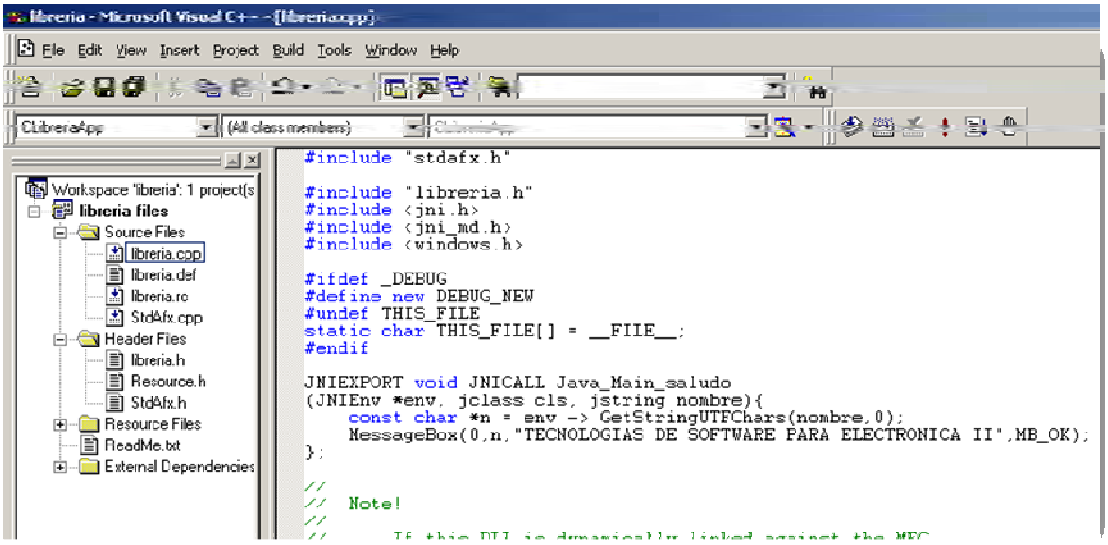
Los nombres de las funciones deberán empezar con Java, luego el nombre del paquete, seguido del nombre de la clase que llamará a la dll y el nombre de la función

*JNIEXPORT void JNICALL Java_Main_saludo(JNIEnv *, jclass, jstring);*

Cuando la máquina virtual invoca a la función, le pasa un puntero a JNIEnv, que contiene la interfaz hacia la máquina virtual, lo que permite al método interactuar con la JVM y trabajar con objetos java. El segundo argumento varía dependiendo de si es un método de instancia o un método de clase (estático).*

- *Si es un método de instancia se trata de un jobject que actúa como un puntero this al objeto Java.*
- *Si es un método de clase, se trata de una referencia jclass a un objeto que representa la clase en la cual están definidos los métodos estáticos.*

*A continuación se ubica el archivo principal del proyecto, llamado **libreria.cpp** y luego de las instrucciones dirigidas al preprocesador (que comienzan con #) se incorpora el código que corresponde a las funciones*



*Además se debe incluir en los archivos **libreria.cpp** y **libreria.h** las librerías:*

- *#include "jni.h"*
- *#include "jni_md.h"*

Una vez creada la dll, esta se debe copiar dentro de la carpeta de WINDOWS en el System32, o también puede ser copiada directamente a la carpeta del proyecto escrito en Java.

Ejemplo No. 2:

Escriba un programa en Java que utilice un método nativo el cual reciba como argumento la frecuencia, duración y pausa, para poder controlar el parlante interno de la PC, y crear así sonidos.

A continuación se muestra el código que se implementa en Java

```
static native void sonido(int frecuencia, int duracción, int pausa);
static
{
    System.loadLibrary("Ejem1");
}
public static void main(String[] args) {
    // TODO code application logic here
    for(int i=1;i<10;i++)
    sonido(100*i,100*i,0);
}
```

A continuación se muestra el código que se implementa en C++

```
#include <windows.h>
#include <iostream.h>


JNIEXPORT void JNICALL Java_Main_sonido
  (JNIEnv *, jclass,jint frecuencia,jint duracion,jint pausa)
  {

    cout<<"Sonido activado\n";
    fflush(stdout);
    Beep(frecuencia, duracion);
    cout<<"Pausa de "<<pausa<<" milisegundos\n";
    fflush(stdout);
    Sleep(pausa);

  };
```

CREACION DE DLL´s EN BORLAND C 5.5

Se debe instalar en la partición C: (*sin que este dentro de otros directorios*), para que la ruta del ejecutable que realiza la compilación (*el bcc32.exe*) quede de esta forma C:\bcc55\Bin.

Luego se debe ir a la carpeta C:\bcc55\Lib\ y copiar todos los archivos (*excepto la carpeta PSDK*) a la carpeta Bin, donde está bcc32.exe. Esto es porque a la hora de compilar, el compilador busca los archivos obj de la carpeta Lib, si no se los copia a la misma carpeta donde está el bcc32.exe generara error.

En la misma carpeta Bin, se debe crear el archivo fuente de la Dll, con la extensión cpp.

Ejemplo No. 3:

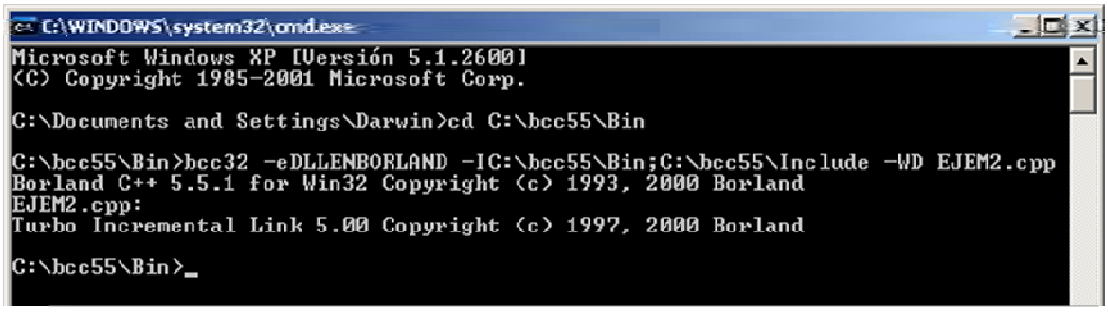
Escriba un programa en Java que utilice un método nativo el cual reciba como argumento su nombre para que sea desplegado en consola (CMD), utilizando funciones contenidas dentro de la librería iostream.h, contenida dentro de Borland C 5.5

```
#include <Main.h>
#include <jni.h>
#include <jni_md.h>
#include <iostream.h>

JNIEXPORT void JNICALL Java_Main_saludo
  (JNIEnv *, jclass){
    cout << "TECNOLOGIAS DE SOFTWARE";
  }
```

La creación del archivo de cabecera se lo realiza utilizando el ejecutable javah y se debe guardar el archivo en la carpeta include junto con los archivos jni.h y jni_md.h, tal como se lo hico con Visual C++.

```
#include <jni.h>
#include <jni_md.h>
/* Header for class Main */

#ifndef _Included_Main
#define _Included_Main
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     Main
 * Method:    saludo
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_Main_saludo
  (JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
```

Luego se va a Inicio->Ejecutar->cmd y se sitúa la carpeta del compilador.



El compilador necesita varios parámetros para crear la DLL:

- *"-eDLLENBORLAND": todo lo que va después de "-e" y sin espacios, será el nombre de la dll.*
- *-I: este comando, seguido de una ruta o varias rutas separadas por punto y coma sin espacios, le indica al compilador dónde buscar las cabeceras que se están usando.*
- *-WD: es el comando para generar una dll y no un exe.*
- *Después de todo lo anterior, se indica qué archivo se quiere compilar.*

A continuación se prueba la DDL con el siguiente código

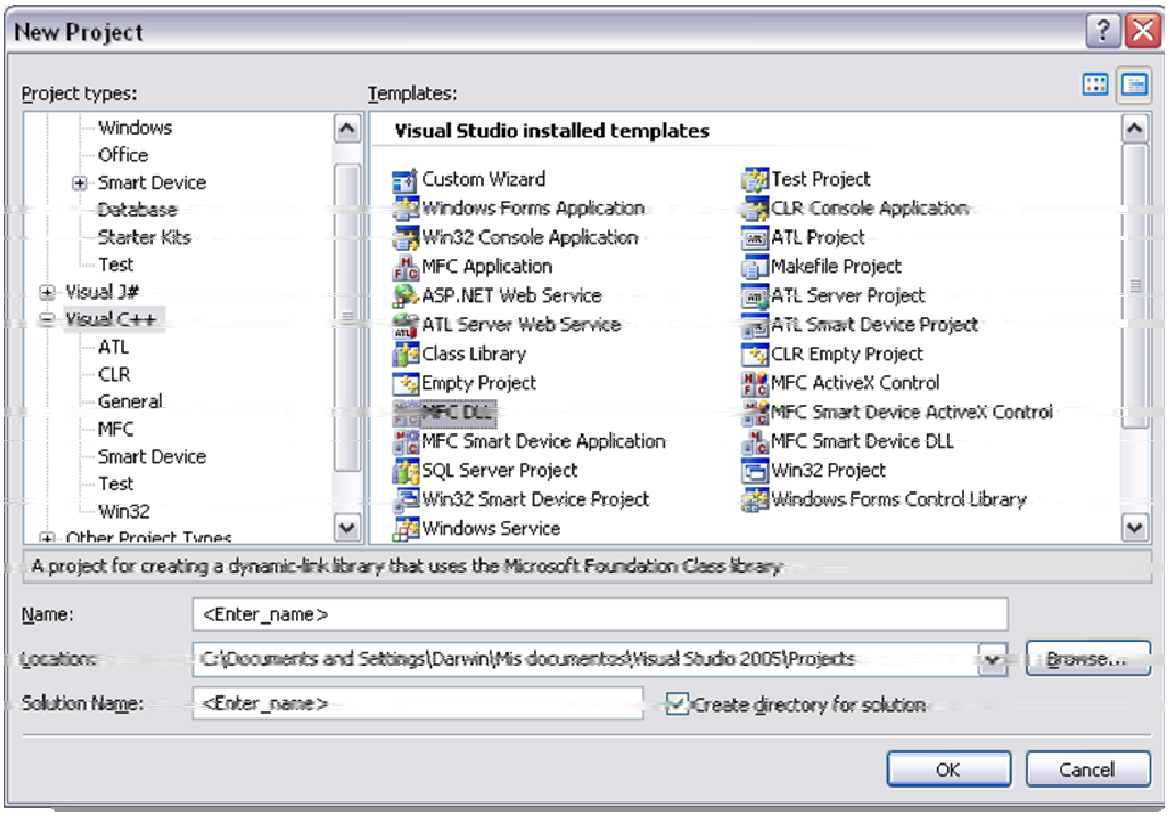
```
public class Main {

static native void saludo();
    static{
        System.loadLibrary("DLLENBORLAND");
    }
    public static void main(String[] args) {
        saludo();
    }

}
```

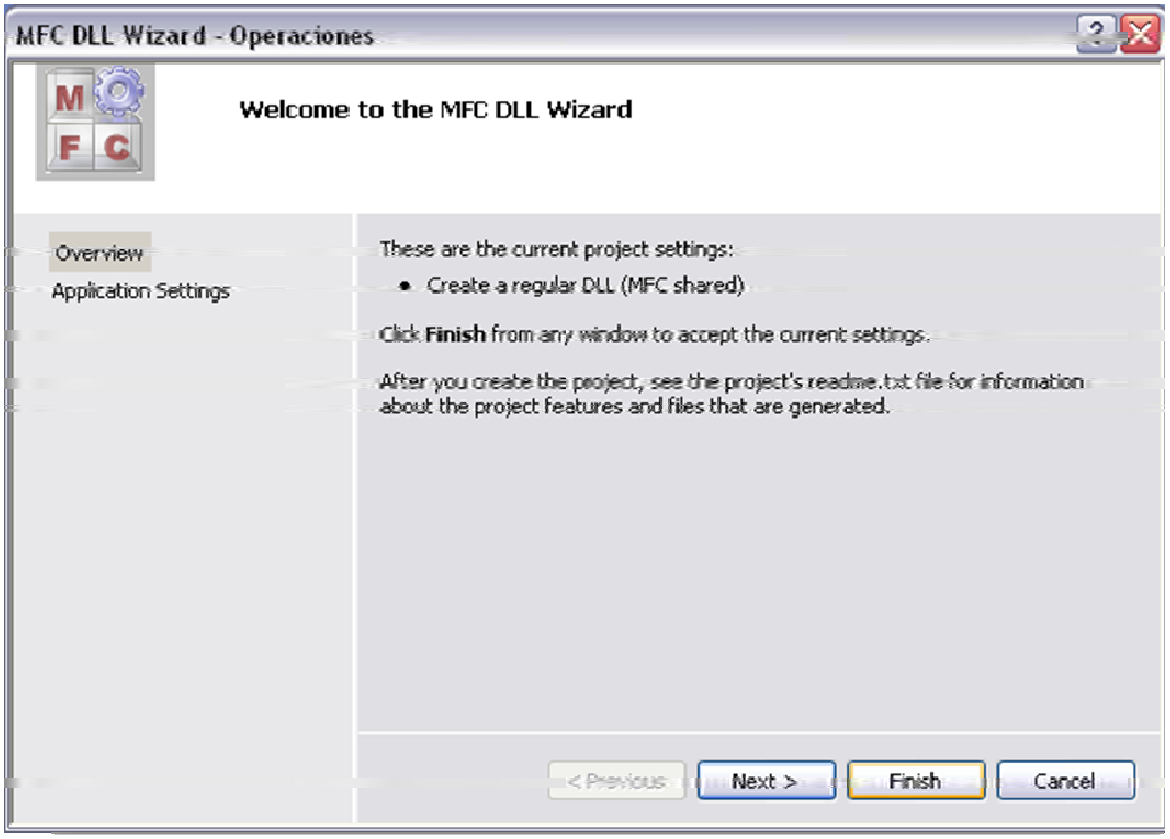
CREACIÓN DE DLL´s CON VISUAL STUDIO 2005

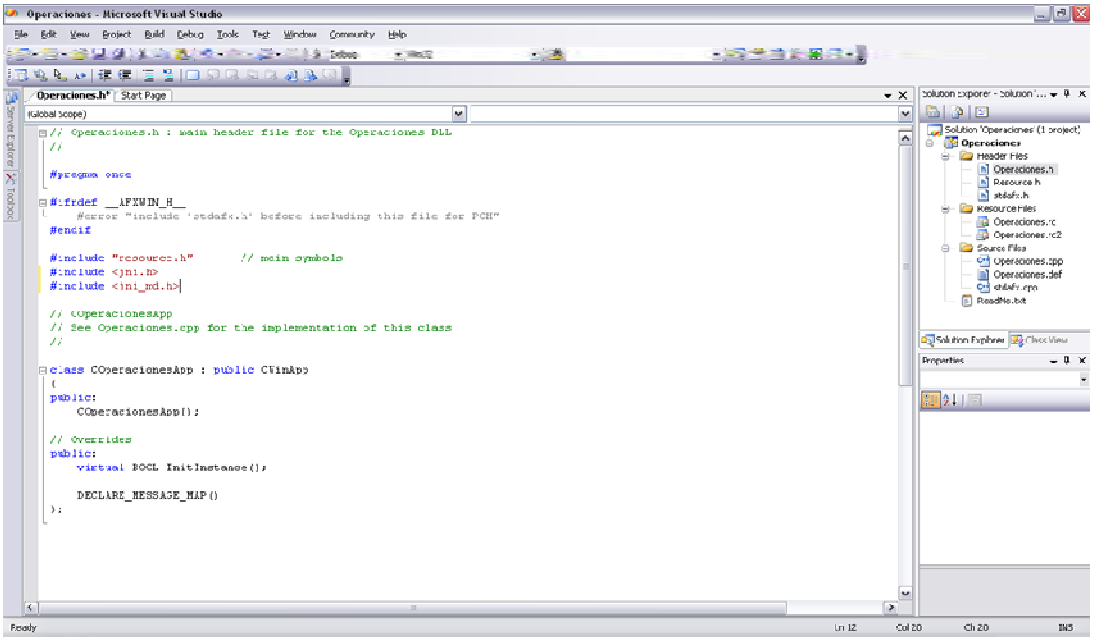
El procedimiento es muy similar a lo que se realiza con Visual C++ 6.0. Para su configuración se deben copiar los archivos jni.h y jni_md.h en la ruta:

…\Microsoft Visual Studio 8\VC\include

Luego se debe crear un proyecto Visual C++, en la opción MFC DLL


New Project
Project types:
Templates:
Windows
Office
Smart Device
Database
Starter Kits
Test
Visual J#
Visual C++
ATL
CLR
General
MFC
Smart Device
Test
Win32
Other Project Types
Visual Studio installed templates
Custom Wizard
Windows Forms Application
Win32 Console Application
MFC Application
ASP.NET Web Service
ATL Server Web Service
Class Library
Empty Project
MFC DLL
MFC Smart Device Application
SQL Server Project
Win32 Smart Device Project
Windows Service
Test Project
CLR Console Application
ATL Project
Makefile Project
ATL Server Project
ATL Smart Device Project
CLR Empty Project
MFC ActiveX Control
MFC Smart Device ActiveX Control
MFC Smart Device DLL
Win32 Project
Windows Forms Control Library
A project for creating a dynamic-link library that uses the Microsoft Foundation Class library
Name:
<Enter_name>
Location:
C:\Documents and Settings\Darwin\Mis documentos\Visual Studio 2005\Projects
Browse...
Solution Name:
<Enter_name>
Create directory for solution
OK
Cancel


MFC DLL Wizard - Operaciones
Welcome to the MFC DLL Wizard
Overview
Application Settings
These are the current project settings:
Create a regular DLL (MFC shared)
Click Finish from any window to accept the current settings.
After you create the project, see the project's readme.txt file for information about the project features and files that are generated.
< Previous
Next >
Finish
Cancel

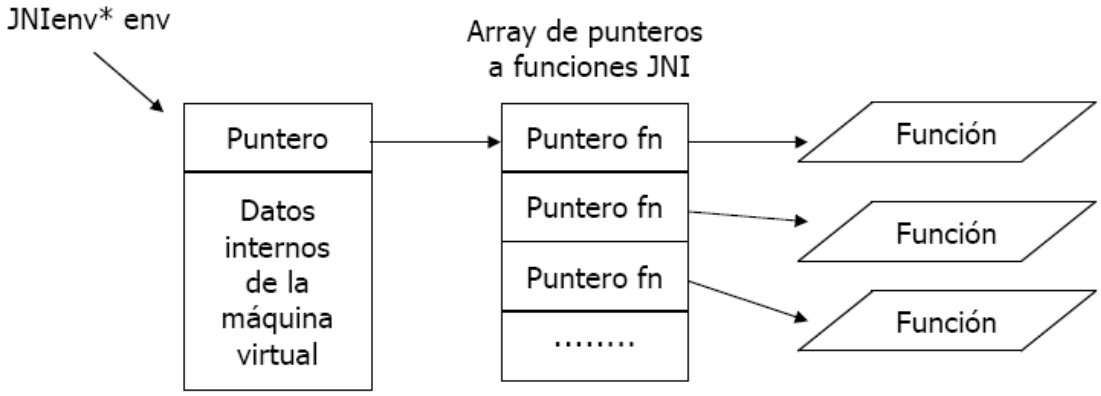


REFERENCIAS

Un método nativo siempre tiene al menos dos parámetros.

```
JNIEXPORT void JNICALL Java_HolaMundo_saluda(JNIEnv* env, jobject
obj);
```

El parámetro env apunta a una tabla de punteros a funciones, que son las funciones que se usa para acceder a los datos Java de JNI.



El segundo argumento varía dependiendo de si es un método de instancia o un método de clase (estático):

- Si es un método de instancia se trata de un jobject que actúa como un puntero this al objeto Java.
- Si es un método de clase, se trata de una referencia jclass a un objeto que representa la clase en la cual están definidos los métodos estáticos.

JNI pasa los objetos y los arreglos, a los métodos nativos como referencias, es decir como punteros C, que apuntan a estructuras internas solo conocidas por la JVM. Los campos de esta estructura no se dan a conocer al programador, por lo que para acceder a ellos se lo hace a través de funciones JNI. Todas las referencias en JNI son de tipo jobject, aunque por mejorar el control de tipos se han creado tipos derivados jstring o jobjectArray, que tienen la siguiente jerarquía.

```
jobject
      jclass
      jstring
      jarray
            jobjectArray
            jbooleanArray
            jbyteArray
            jcharArray
            jshortArray
            jintArray
            jlongArray
            jfloatArray
            jdoubleArray
```

ACCESO A OBJETOS STRING

String es una clase que está representado por el tipo C jstring. Para acceder al contenido de este objeto existen funciones que convierten un String Java en cadenas C (Unicode o UTF-8):

- env-> GetStringUTFChars(jstring,0);
  Convierte la cadena texto de java en un arreglo de caracteres de C.
- env-> NewStringUTF(Cadena en C));
  Crea un objeto String de Java a partir de una cadena en C

NOTA:

Para cerciorarse de la forma de los métodos, abrir el archivo jni.h

Ejemplo No. 4:

Cree un programa que permita ingresar dos datos desde el teclado y que realice la suma de los mismos.

```
JNIEXPORT jint JNICALL Java_Ejemplo1_leemos
(JNIEnv *env, jobject object, jstring texto)
{
    int a;
    const char *p = env -> GetStringUTFChars(texto,0);
    cout<<p;
    cin>>a;
    return(a);
}
```

```
JNIEXPORT jint JNICALL Java_Ejemplo1_leemos
(JNIEnv *env, jobject object, jstring texto)
{
    int resultado;
    resultado = a + b;
    return resultado;
}
```

Ejemplo No. 5:

Crear un programa con JNI que permite comparar entre los números ingresados como enteros y su correspondiente representación en letras, utilizando para lo cual la sentencia switch.

```
JNIEXPORT jstring JNICALL Java_Main_Letras
  (JNIEnv *env, jclass c, jint a)
{
    const char * b="";
    switch (a)
    {
        case 1: b="uno";
        break;
        case 2: b="dos";
        break;
        case 3: b="tres";
        break;
        case 4: b="cuatro";
        break;
        default:
        b="El numero sobrepasa el rango";
    }
    return(env-> NewStringUTF(b));

};
```

Ejemplo No. 6:

Escriba un método que reciba como argumento un String que se imprimirá en consola y que permita leer texto escrito por el usuario en forma de String.

```
JNIEXPORT jstring JNICALL Java_ejem10cap1tec2v4_Main_leeTexto
(JNIEnv *env, jclass cls, jstring msn)
{
    const char *mensaje = env -> GetStringUTFChars(msn,0);
    cout<<mensaje;

    char nuevo [128];
    cin>>nuevo;
    return(env->NewStringUTF(nuevo));

};
```

ARREGLOS

JNI permite trabajar con dos tipos de arreglos:

- Arreglos de tipos primitivos. Son arreglos cuyos elementos son tipos primitivos como boolean o int.
- Arreglos de referencias. Son arreglos cuyos elementos son objetos u otros arreglos. (Arreglo de arreglos

```
int[] iarr; // Array de tipos fundamentales
float[] farr; // Array de tipos fundamentales
Object[] oarr; // Array de referencias
int[][] iarr2; // Array de referencias
```

Para acceder a los arreglos de tipos primitivos existen funciones JNI que retornan el arreglo en una variable tipo **jarray** o derivada.

```
jarray
      jobjectArray
      jbooleanArray
      jbyteArray
      jcharArray
      jshortArray
      jintArray
      jlongArray
      jfloatArray
      jdoubleArray
```

Los tipos de array básicos se pueden ver en la siguiente tabla:

Tipo Nativo	TipoArray
jboolean	jbooleanArray
jbyte	jbyteArray
jchar	jcharArray
jshort	jshortArray
jint	jintArray
jlong	jlongArray
jfloat	jfloatArray
jdouble	jdoubleArray

A continuación se muestran algunas funciones empleadas para el tratamiento de arreglos:

- env -> GetIntArrayElements(arreglo,0);
  Extrae el contenido del arreglo.
- env -> GetArrayLength(arrayA);
  Obtiene la longitud del arreglo
- env -> NewIntArray(longitud);
  Crea un nuevo arreglo de la longitud especificada

A continuación se muestran todas funciones empleadas para el tratamiento de arreglos, contenidas dentro del archivo de cabecera jni.h:

```
jboolean * GetBooleanArrayElements(jbooleanArray array, jboolean *isCopy) {
    return functions->GetBooleanArrayElements(this,array,isCopy);
}
jbyte * GetByteArrayElements(jbyteArray array, jboolean *isCopy) {
    return functions->GetByteArrayElements(this,array,isCopy);
}
jchar * GetCharArrayElements(jcharArray array, jboolean *isCopy) {
    return functions->GetCharArrayElements(this,array,isCopy);
}
jshort * GetShortArrayElements(jshortArray array, jboolean *isCopy) {
    return functions->GetShortArrayElements(this,array,isCopy);
}
jint * GetIntArrayElements(jintArray array, jboolean *isCopy) {
    return functions->GetIntArrayElements(this,array,isCopy);
}
jlong * GetLongArrayElements(jlongArray array, jboolean *isCopy) {
    return functions->GetLongArrayElements(this,array,isCopy);
}
jfloat * GetFloatArrayElements(jfloatArray array, jboolean *isCopy) {
    return functions->GetFloatArrayElements(this,array,isCopy);
}
jdouble * GetDoubleArrayElements(jdoubleArray array, jboolean *isCopy) {
    return functions->GetDoubleArrayElements(this,array,isCopy);
}
```

```
jbooleanArray NewBooleanArray(jsize len) {
    return functions->NewBooleanArray(this,len);
}
jbyteArray NewByteArray(jsize len) {
    return functions->NewByteArray(this,len);
}
jcharArray NewCharArray(jsize len) {
    return functions->NewCharArray(this,len);
}
jshortArray NewShortArray(jsize len) {
    return functions->NewShortArray(this,len);
}
jintArray NewIntArray(jsize len) {
    return functions->NewIntArray(this,len);
}
jlongArray NewLongArray(jsize len) {
    return functions->NewLongArray(this,len);
}
jfloatArray NewFloatArray(jsize len) {
    return functions->NewFloatArray(this,len);
}
jdoubleArray NewDoubleArray(jsize len) {
    return functions->NewDoubleArray(this,len);
}
```

A continuación se muestran algunas funciones empleadas para el tratamiento de arreglos, de manera general:

- (jint*) env -> GetPrimitiveArrayCritical(arrayA,0);
- env -> ReleasePrimitiveArrayCritical(arrayC,C,0);

Las funciones trabajan con el arreglo original con lo que no hay necesidad de llamar a otras funciones JNI

```
void * GetPrimitiveArrayCritical(jarray array, jboolean *isCopy) {
    return functions->GetPrimitiveArrayCritical(this,array,isCopy);
}
void ReleasePrimitiveArrayCritical(jarray array, void *carray, jint mode) {
    functions->ReleasePrimitiveArrayCritical(this,array,carray,mode);
}
```

Ejemplo No. 6:

Realice un programa con JNI que sume todos los elementos contenidos dentro de un arreglo de enteros, el cual será enviado como argumento de la función.

```
JNIEXPORT jint JNICALL Java_Main_Arreglo
(JNIEnv *env, jobject obj, jintArray arreglo)
{
        jsize       len   =  env -> GetArrayLength(arreglo);
        jint        *b    =  env -> GetIntArrayElements(arreglo,0);

        int acumulador=0;
        for(int i=0;i<len;i++)
        {
            acumulador=acumulador+b[i];
        }
        return(acumulador);

};
```

Ejemplo No. 7:

Escriba un programa que utilice métodos nativos que permita ingresar dos arreglos y que devuelva un nuevo arreglo resultado de sumar los anteriores.

```
JNIEXPORT jintArray JNICALL Java_Main_SumaVector
  (JNIEnv *env, jclass cls, jintArray arrayA, jintArray arrayB)
{
    jsize longitud    = env -> GetArrayLength(arrayA);
    jintArray arrayC  = env -> NewIntArray(longitud);
    jint *A           = (jint*) env -> GetPrimitiveArrayCritical(arrayA,0);
    jint *B           = (jint*) env -> GetPrimitiveArrayCritical(arrayB,0);
    jint *C           = (jint*) env -> GetPrimitiveArrayCritical(arrayC,0);

    for(int i=0;i<longitud;i++)
    {
        C[i]=A[i]+B[i];
    }

    env -> ReleasePrimitiveArrayCritical(arrayC,C,0);
    return(arrayC);
};
```

MATRICES

JNI tiene dos funciones para el acceso a matrices:

```
jobject GetObjectArrayElement(jobjectArray array, jsize index) {
    return functions->GetObjectArrayElement(this,array,index);
}
void SetObjectArrayElement(jobjectArray array, jsize index,
                           jobject val) {
    functions->SetObjectArrayElement(this,array,index,val);
}
```

A diferencia de los arreglos de tipos de datos primitivos, no se puede acceder a todos los elementos a la vez, sino que se debe utilizar elemento a elemento.

- jobjectArray NewObjectArray(jsize, jclass,0)
  Crea un arreglo.
- jclass FindClass(const char *)
  Jclass, indica el tipo de elementos del arreglo, mediante descriptores, que hacen refencia al tipo de dato.

Un descriptor de campo consiste en uno o más caracteres que describen completamente un tipo de campo. El descriptor de tipo arreglo se precede con el carácter "[" para cada dimensión del arreglo. Por ejemplo el tipo numérico int [] se describe mediante [I

Tipo primitivo	Descriptor de campo
boolean	Z
byte	B
char	C
Short	S
Int	I
Long	J
Float	F
Double	D

Ejemplo No. 8:

Escriba un programa que permita ingresar dos matrices de números enteros y que devuelva una nueva matriz producto de sumar los anteriores.

```
JNIEXPORT jobjectArray JNICALL Java_NewMain_sumaMatrices
(JNIEnv *env, jclass cls, jobjectArray arrayA, jobjectArray arrayB){

    jobjectArray       arrayC;
    jsize              i;
    jsize n_vectores   = env ->  GetArrayLength(arrayA);
    cls                = env ->  FindClass("[I");
    arrayC             = env ->  NewObjectArray(n_vectores,cls,0);

    for(i=0;i<n_vectores;i++)
    {
        jsize                 j;
        jint                  *bufferA, *bufferB, *bufferC;
        jintArray vectorA     = (jintArray) env  ->  GetObjectArrayElement(arrayA,i);
        jintArray vectorB     = (jintArray) env  ->  GetObjectArrayElement(arrayB,i);
        jsize longitud_vector = env  ->  GetArrayLength(vectorA);
        jintArray vectorC     = env  ->  NewIntArray(longitud_vector);
        bufferA               = env  ->  GetIntArrayElements(vectorA,0);
        bufferB               = env  ->  GetIntArrayElements(vectorB,0);
        bufferC               = env  ->  GetIntArrayElements(vectorC,0);

        for(j=0;j<longitud_vector;j++)
        {
            bufferC[j] = bufferA[j] + bufferB[j];
        }

        env  ->  ReleaseIntArrayElements(vectorA,bufferA,0);
        env  ->  ReleaseIntArrayElements(vectorB,bufferB,0);
        env  ->  ReleaseIntArrayElements(vectorC,bufferC,0);
        env  ->  SetObjectArrayElement(arrayC,i,vectorC);
        env  ->  DeleteLocalRef(vectorA);
        env  ->  DeleteLocalRef(vectorB);
    }
    return arrayC;
};
```

MANEJO DE OBJETOS

Para acceder a los atributos de instancia, los cuales existen uno por cada objeto, se debe extraer el field ID que es de tipo jfieldID.

- jfieldID.- es un tipo de variable usada por JNI para referirse a un atributo de un objeto.
- GetFieldID.- permite extrear el jfieldID
- GetDoubleField.- permite leer o modificar el atributo a partir de su jobject y su jfieldID. Existe una función para cada tipo de dato.
    - GetObjectField.- permite extraer atributos que sean objetos o arreglos.
- GetObjectClass.- Esta función permite obtener una referencia a la clase del objeto.

A continuación se muestran todas funciones empleadas para el tratamiento de objetos, contenidas dentro del archivo de cabecera jni.h:

```
jobject GetObjectField(jobject obj, jfieldID fieldID) {
    return functions->GetObjectField(this,obj,fieldID);
}
jboolean GetBooleanField(jobject obj, jfieldID fieldID) {
    return functions->GetBooleanField(this,obj,fieldID);
}
jbyte GetByteField(jobject obj, jfieldID fieldID) {
    return functions->GetByteField(this,obj,fieldID);
}
jchar GetCharField(jobject obj, jfieldID fieldID) {
    return functions->GetCharField(this,obj,fieldID);
}
jshort GetShortField(jobject obj, jfieldID fieldID) {
    return functions->GetShortField(this,obj,fieldID);
}
jint GetIntField(jobject obj, jfieldID fieldID) {
    return functions->GetIntField(this,obj,fieldID);
}
jlong GetLongField(jobject obj, jfieldID fieldID) {
    return functions->GetLongField(this,obj,fieldID);
}
jfloat GetFloatField(jobject obj, jfieldID fieldID) {
    return functions->GetFloatField(this,obj,fieldID);
}
jdouble GetDoubleField(jobject obj, jfieldID fieldID) {
    return functions->GetDoubleField(this,obj,fieldID);
}
```

Ejemplo No. 9:

Cree un programa que permita determinar la distancia entre dos puntos utilizando métodos nativos.

A continuación se muestra el código implementado en java.

```
public class Objetos {
    public native double Distancia(Punto a, Punto b);
    static
    {
        System.loadLibrary("Objetos");
    }
}
```

```
public class Punto {
    public double x,y;
    Punto(double x, double y)
    {
        this.x=x;
        this.y=y;
    }
}

public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Punto a = new Punto(1,1);
        Punto b = new Punto(5,5);
        Objetos ob = new Objetos();
        double aux = ob.Distancia(a, b);
        System.out.println("Distancia:"+aux);
    }

}
```

A continuación se muestra el código de la función en C.

```
JNIEXPORT jdouble JNICALL Java_Objetos_Distancia
  (JNIEnv *env, jobject obj, jobject a, jobject b)
{
    jfieldID x1,y1;
    jfieldID x2,y2;
    jdouble ax1,ay1,bx2,by2;

    jclass cls1 = env -> GetObjectClass(a);
    jclass cls2 = env -> GetObjectClass(b);

    x1 = env -> GetFieldID(cls1,"x","D");
    y1 = env -> GetFieldID(cls1,"y","D");

    x2 = env -> GetFieldID(cls2,"x","D");
    y2 = env -> GetFieldID(cls2,"y","D");

    ax1 = env -> GetDoubleField(a,x1);
    ay1 = env -> GetDoubleField(a,y1);

    bx2 = env -> GetDoubleField(b,x2);
    by2 = env -> GetDoubleField(b,y2);

    double aux1=pow(ax1-bx2,2);
    double aux2=pow(ay1-by2,2);
    jdouble distancia= sqrt(aux1+aux2);

    return(distancia);

};
```

CÓDIGO ENSAMBLADOR EN JNI

Existen tres razones por las que puede querer usar una rutina escrita en ensamblador:

- Aumentar la velocidad y la eficiencia.
- Realizar una función específica de la máquina que no está disponible en Java o en C.
- Utilizar una rutina en lenguaje ensamblador de otro origen.

La escritura de una rutina usando ensamblador varía de compilador a compilador. Cada procesador tiene un lenguaje ensamblador diferente y cada sistema operativo tiene una estructura de interfaz diferente. Además cada compilador de C tiene un convenio de llamada diferente, que define como se pasa y se obtiene información a/y de la función.

La sintaxis de asm se puede entender fácilmente con un ejemplo:

```
__asm
{
    mov eax, a
    add eax, b
    mov resultado, eax
}
```

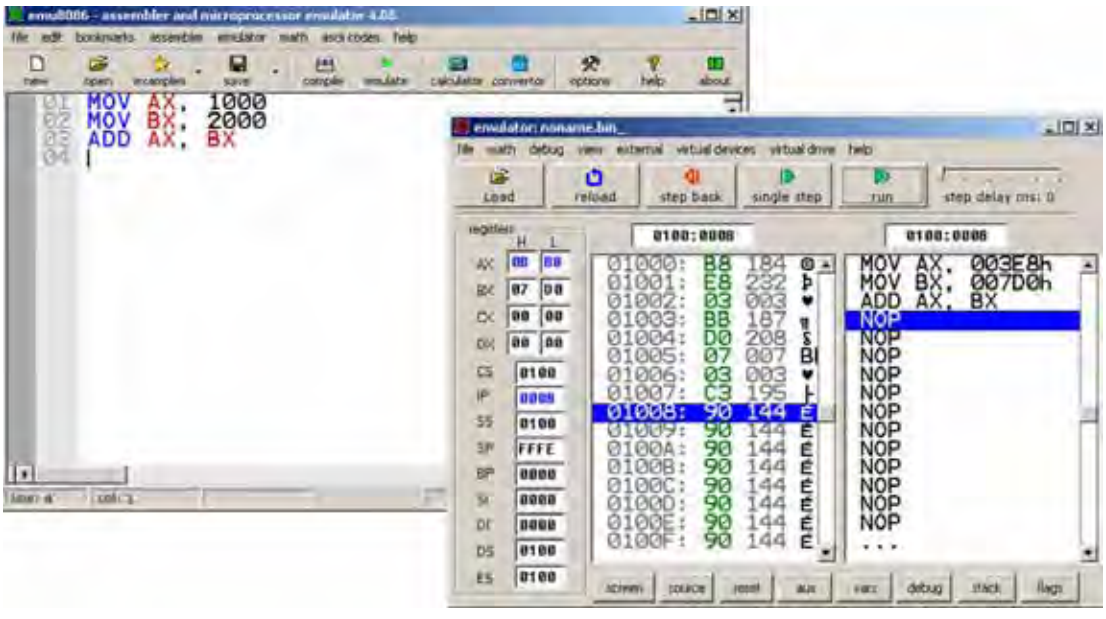
```
__asm mov eax, a
__asm add eax, b
__asm mov resultado, eax
```

Aunque __asm no es soportado por algunos compiladores, esta sentencia permite que el código en ensamblador sea parte de un programa en C sin usar módulos separados. Tenga en cuenta que se crea dependencia con la máquina lo que hará difícil de portar el programa a otras máquinas.

Como se ha visto el código en ensamblador se debe incluir entre la claúsula __asm{ }.

Los registros como AX, BX, CX, DX son de 16 bits, sin embargo las variables de tipo entero son de 32 bits, por lo que se utiliza una extensión de esos registros, de tal forma que se tiene los registros: EAX, EBX, ECX, EDX. La CPU x86 tiene 14 registros internos y básicos. Estos registros son de 16 bits nombrados de la siguiente manera, a excepción del registro de banderas. Registros de uso general

- AX: Acumulador (AL:AH)
- BX: Registro base (BL:BH)
- CX: Registro contador (CL:CH)
- DX: Registro de datos (DL:DH)



A continuación se muestran alguas instrucciones empleadas en asm.

MOV

Algoritmo:
operando1 = operando2

Ejemplo:

```
MOV DS, AX ; copia el  valor de AX a DS.
```

ADD

Algoritmo:
operando1 = operando1 + operando2
Ejemplo:

```
MOV AL, 5 ; AL = 5
ADD AL, -3 ; AL = 2
```

SUB

Algoritmo:
operando1 = operando1 - operando2
Ejemplo:

```
MOV AL, 5
SUB AL, 1; AL = 4
```

CMP

Compara los valores de los registros, los resultados se almacenan en las banderas (OF, SF, ZF, AF, PF, CF).

Ejemplo:

```
MOV AL, 5
MOV BL, 5
CMP AL, BL; AL = 5, ZF = 1
```

JMP

Salto incondicional

JNP

Salto si zF = 1

Ejemplo No. 10:

Implemente un programa que permita realizar la suma y la resta de dos números enteros empleando código nativo y lenguaje ensamblador.

```
package ejem4cap1tecv3;

public class Main {
    static native int sumar(int a, int b);
    static native int restar(int a, int b);

    static{
    System.loadLibrary("ASM");
    }
    public static void main(String[] args) {
        System.out.println("La suma es:"+sumar(2,4));
        System.out.println("La suma es:"+restar(2,4));
    }
}
```

```
#ifndef _Included_ejem4cap1tecv3_Main
#define _Included_ejem4cap1tecv3_Main
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     ejem4cap1tecv3_Main
 * Method:    sumar
 * Signature: (II)I
 */
JNIEXPORT jint JNICALL Java_ejem4cap1tecv3_Main_sumar
  (JNIEnv *, jclass, jint, jint);

JNIEXPORT jint JNICALL Java_ejem4cap1tecv3_Main_restar
  (JNIEnv *, jclass, jint, jint);


#ifdef __cplusplus
}
#endif
#endif
```

```
JNIEXPORT jint JNICALL Java_ejem4cap1tecv3_Main_sumar
(JNIEnv *, jclass, jint a, jint b){
    int resultado=0;


    __asm
    {
        mov eax, a
        add eax, b
        mov resultado, eax
    }
    return resultado;
};

JNIEXPORT jint JNICALL Java_ejem4cap1tecv3_Main_restar
(JNIEnv *, jclass, jint a, jint b){
    int resultado=0;


    __asm
    {
        mov eax, a
        sub eax, b
        mov resultado, eax
    }
    return resultado;
};
```

Ejemplo No. 11:

Cree un programa que implemente el algoritmo de la sumatoria.

$$\sum_{i=m}^{n} x_i = x_m + x_{m+1} + x_{m+2} + \cdots + x_n$$

```
JNIEXPORT jint JNICALL Java_ejem4cap1tecv3_Main_sumatoria
(JNIEnv *, jclass, jint a, jint b){
    int resultado=0;

    __asm
    {

        mov eax,a
        mov ebx,b
        mov edx,eax
        jz suma_1
    suma_1:
        add dx,1
        add ax,dx
        cmp bx,dx
        jz fin_1
        jmp suma_1
    fin_1:
        mov resultado, eax
    }
    return resultado;
};
```

PUERTOS

Un puerto es un dispositivo que conecta a un procesador con el mundo exterior. Por medio de un puerto, el procesador recibe una señal desde un dispositivo de entrada y envía una señal a un dispositivo de salida, Los puertos son identificados por sus direcciones en el intervalo OH-3FFH, o 1024 puertos en total.

Se puede utilizar las instrucciones IN y OUT para manejar E/S directamente a nivel de puerto:

- IN transfiere información desde un puerto de entrada al AL si es byte al AX si es una palabra.

  IN registro,puerto

- OUT transfiere información desde un puerto de salida *al AL si es un byte y al AX si es una palabra.*

  OUT puerto,registro

Se puede leer un dato del puerto mediante la instrucción IN ó escribir un dato en el puerto con la instrución OUT . En ambos casos el registro AL debe participar activamente en la instrucción, bien sea como fuente o destino del dato.

out DX, AL ; lleva al puerto DX el contenido del registro AL
in AL,DX ; lleva al registro AL, el contenido del puerto DX

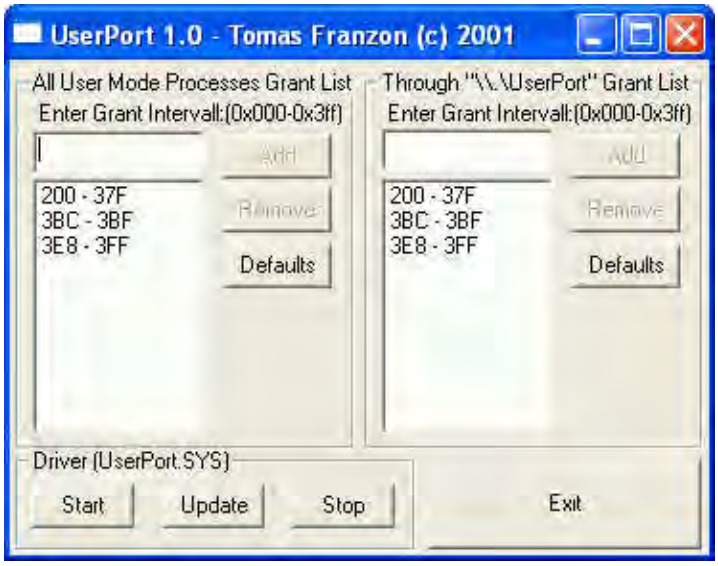
El número del puerto sobre el que se va a realizar la transferencia de datos debe estar señalado por le registro DX, a excepción de los casos en los cuales el número del puerto es inferior a 255 (FFh), en cuyo caso la instrucción que lee o escribe puede señalar directamente el puerto.

DRIVER PARA ACCEDER AL PUERTO PARALELO EN WINDOWS

Windows NT, 2000 y XP no permiten acceder al hardware de forma directa como lo hacen las versiones 95, 98 y ME. Para poder leer y escribir en el puerto paralelo con lenguaje C o ensamblador es necesario un driver que comunique el software con dispositivo externo.

A continuación se muestra el proceso a seguir para instalar el driver que permita acceder al puerto paralelo.

- Copiar el archivo "userport.sys" en la carpeta de sistema C:\WINDOWS\SYSTEM32\DRIVERS
- Ejecutar el archivo UserPort.exe



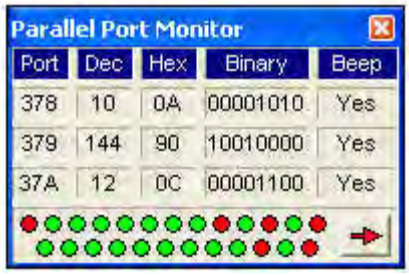
- Para iniciar el driver, acciona el botón START
- Si se desea detener la ejecución, se puede desactivar el driver con el botón STOP.

NOTA: El direccionamiento de los puertos, está indicado en el cuadro de diálogo de USERPORT. En él hace referencia a los rangos habituales que se asigna a los puertos en un PC. Si el direccionamiento del puerto no coincide con los que allí aparecen, se puede añadir un nuevo rango de direcciones escribiéndolas en notación hexadecimal.

Ejemplo No. 12:

Desarrollar una aplicación que permita escribir un dato por el puerto paralelo.

```
package ejem2cap2tec2v4;

public class Main {
    public static native void salida(byte dato);
    static
    {
        System.loadLibrary("puertos2");
    }

    public static void main(String[] args) {
        // TODO code application logic here
        byte a=10;
        salida(a);
    }

}
```

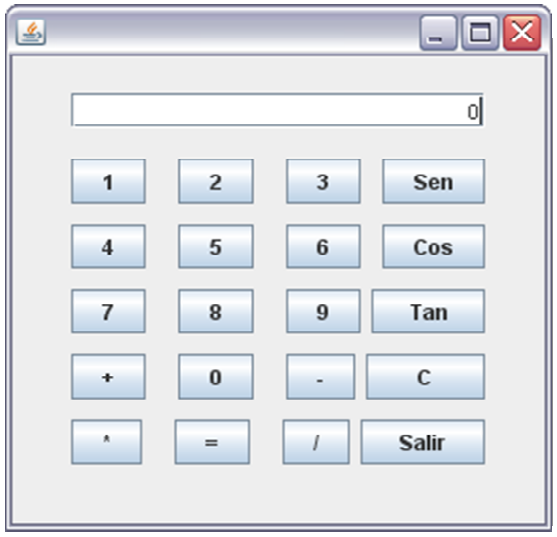



```
JNIEXPORT void JNICALL Java_ejem2cap2tec2v4_Main_salida
  (JNIEnv *, jclass, jbyte dato)
{
    __asm
    {
        mov dx,378H //Asigno a Dx la dirección del puerto que es 378H
        mov al,dato //Asigno el dato que quiero sacar a AL
        out dx,al //Saco el dato.
    }

};
```

ACTIVIDADES DE REELABORACION

ACTIVIDADES PRÁCTICAS

1.- Cree un programa que permita trabajar con la interface que se ilustra y que emplee métodos nativos para su funcionamiento. Además se debe crear una clase exclusiva para los métodos nativos, los cuales requerirán únicamente dos argumentos de tipo real.



2.- A partir del Ejemplo No.2, cree una aplicación con interface gráfica que se asemeje a un piano, para lo cual investigue las frecuencia para las notas, Do, Re, Mi, Fa, Sol, La y Si.

3.- Cree un programa en consola que permita ingresar un número y calcule su factorial.

4.- Escriba un programa el cual lee desde el teclado dos nombres, y los compara en función del número de caracteres imprimiendo en pantalla si es: igual, menor que o mayor que.

NOTA: Consulte la función pertinente en C que permite realizar esta comparación.

5.- Cree un programa que utilice métodos nativos que permita ingresar un arreglo de números decimales, por teclado y que presente en pantalla el elemento de mayor valor, así como todos sus elementos.

- Se debe crear un método nativo en C que permita realizar el ingreso de los números.
- Se debe crear un método nativo en C que permita imprimir en pantalla los elementos de un arreglo que se envíe como argumento.

6.- Cree un programa que utilice métodos nativos que permita ingresar un arreglo de números enteros y que retorne un nuevo arreglo en el cual los elementos del arreglo original estén ordenados de mayor a menor.

7.- Amplíe el ejemplo anterior de tal manera que permita ingresar dos matrices de números enteros empleando una interface gráfica en la que se utilice tablas para representar las posiciones de los elementos de la matriz y que imprima una nueva matriz producto de sumar los anteriores en la interface gráfica creada para el ingreso.

NOTA: Las tablas deben ser creadas de manera dinámica según las matrices que se ingresan.

8.- Cree una aplicación con interface gráfica que permita ingresar una matriz y que devuelva la matriz triangular inferior.

NOTA: Las tablas deben ser creadas de manera dinámica según las matrices que se ingresan.

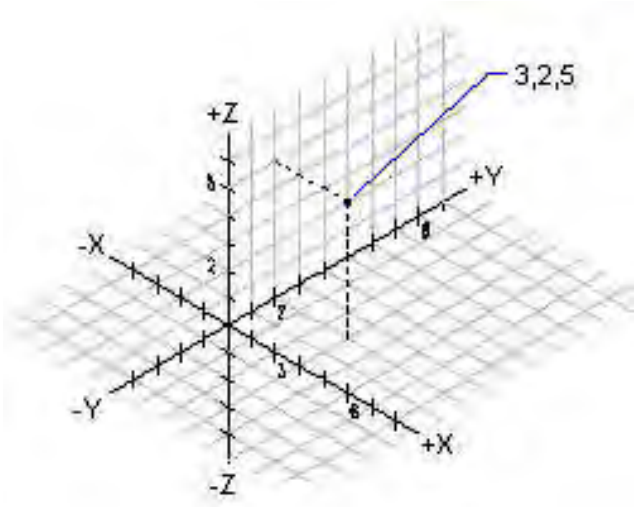
$$\begin{pmatrix} u_{11} & u_{12} & u_{13} & . & . & . & u_{1n} \\ 0 & u_{22} & u_{23} & . & . & . & u_{2n} \\ 0 & 0 & u_{33} & . & . & . & u_{3n} \\ . & . & .. & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & . & u_{nn} \end{pmatrix}$$

9.- Implemente un programa que utilice métodos nativos, que permita determinar entre dos personas cual es la mayor.

- El programa debe imprentar una clase ESTUDIANTE, que dispone de dos atributos de clase uno para edad y otro para nombre.
- El programa debe devolver un objeto.

10.- Cree un programa utilizando código nativo que permita determinar la distancia desde el origen a un punto en el espacio, además el programa debe ser capas de presentar una representación gráfica.

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} = d$$



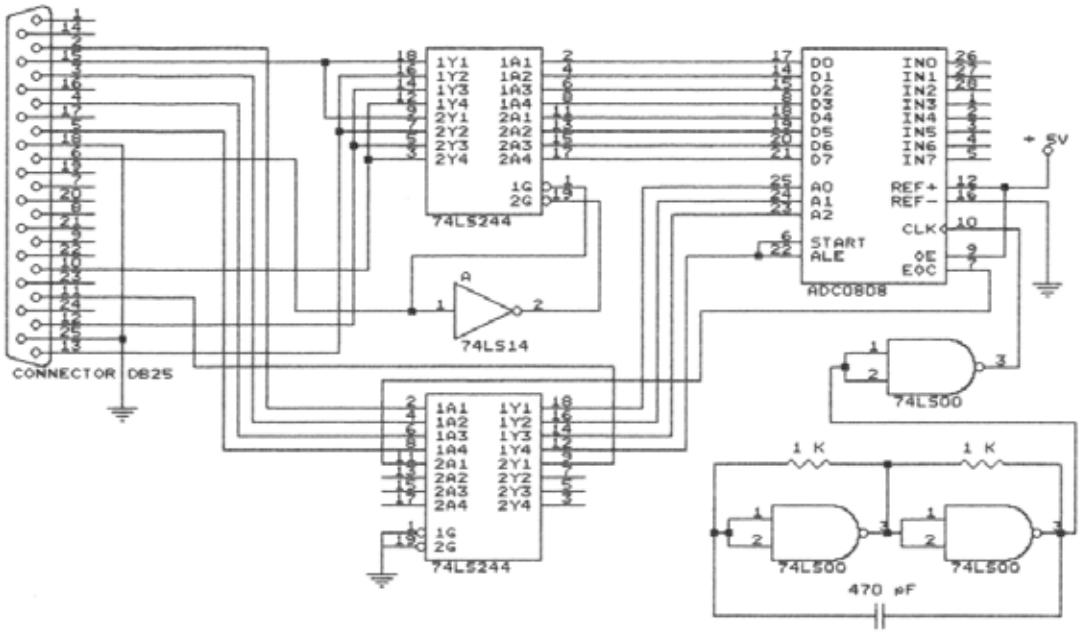
11.- Amplíe el ejemplo No. 10 anterior de tal manera que permita realizar la multiplicación y división de números enteros.

12.- Cree un programa que permita calcular el factorial de un número.

$$n! = 1 \times 2 \times 3 \times 4 \times ... \times (n-1) \times n$$

13.- Desarrollar una aplicación que utilice métodos nativos con código ensamblador para controlar la bocina integrada de imán permanente de la PC.

14.- Implementar una tarjeta de adquisición de datos análogos. Esta tarjeta, permitirá medir ocho canales análogos y, una vez convertidos a señal digital, llevarlos al computador a través del puerto paralelo.


CONNECTOR DB25
74LS244
74LS14
74LS244
ADC0808
+ 5V
74LS00
1 K
1 K
74LS00
74LS00
470 pF

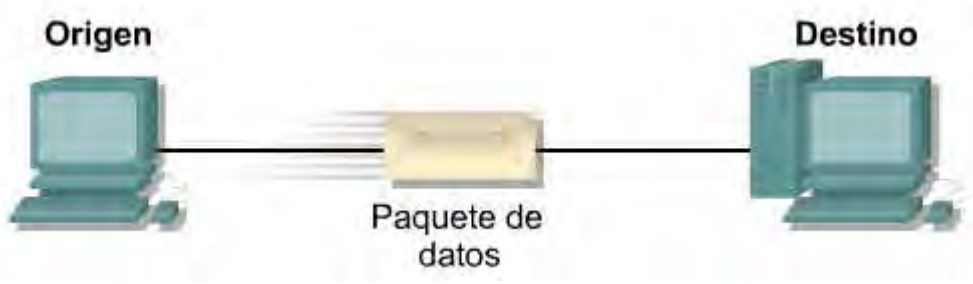
CAPITULO 10

MODELOS DE COMUNICACIONES

El concepto de capas se utiliza para describir la comunicación entre dos computadores. La figura muestra un conjunto de preguntas relacionadas con el flujo, que se define como el movimiento de objetos físicos o lógicos, a través de un sistema.

La conversación entre dos personas es un buen ejemplo para aplicar un enfoque en capas para analizar el flujo de información. Se puede desglosar este proceso en distintas capas aplicables a todas las conversaciones. La capa superior es la idea que se comunicará. La capa intermedia es la decisión respecto de cómo se comunicará la idea. La capa inferior es la creación del sonido que transmitirá la comunicación.

El mismo método de división en capas explica cómo una red informática distribuye la información desde el origen al destino. Cuando los computadores envían información a través de una red, todas las comunicaciones se generan en un origen y luego viajan a un destino.



Fuente: Cisco-CCNA1

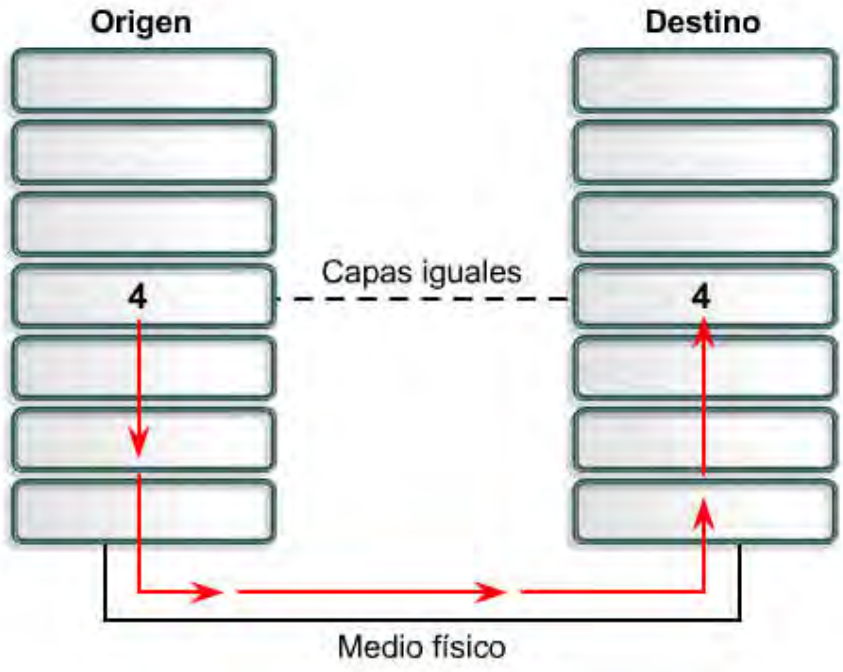
Generalmente, la información que se desplaza por una red recibe el nombre de datos o paquete. Un paquete es una unidad de información, lógicamente agrupada, que se desplaza entre los sistemas de computación. A medida que los datos atraviesan las

capas, cada capa agrega información que posibilita una comunicación eficaz con su correspondiente capa en el otro computador.

Para que los paquetes de datos puedan viajar desde el origen hasta su destino a través de una red, es importante que todos los dispositivos de la red hablen el mismo lenguaje o protocolo. Un protocolo es un conjunto de reglas que hacen que la comunicación en una red sea más eficiente. Por ejemplo, al pilotar un avión, los pilotos obedecen reglas muy específicas para poder comunicarse con otros aviones y con el control de tráfico aéreo.

Un protocolo de comunicaciones de datos es un conjunto de normas, o un acuerdo, que determina el formato y la transmisión de datos.

El protocolo en una capa realiza un conjunto determinado de operaciones sobre los datos al prepararlos para ser enviados a través de la red. Los datos luego pasan a la siguiente capa, donde otro protocolo realiza otro conjunto diferente de operaciones. Una vez que el paquete llega a su destino, los protocolos deshacen la construcción del paquete que se armó en el extremo de origen. Esto se hace en orden inverso. Los protocolos para cada capa en el destino devuelven la información a su forma original, para que la aplicación pueda leer los datos correctamente.



Fuente: Cisco-CCNA1

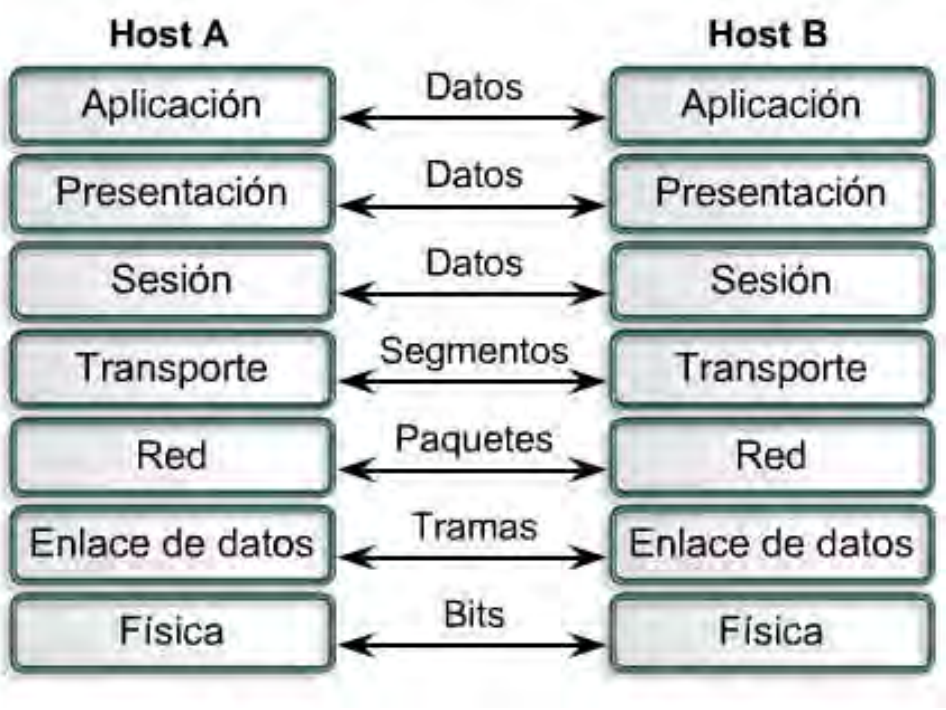
El modelo de referencia OSI es un marco que se puede utilizar para comprender cómo viaja la información a través de una red. El modelo de referencia OSI explica de qué manera los paquetes de datos viajan a través de varias capas a otro dispositivo de una red, aun cuando el remitente y el destinatario poseen diferentes tipos de medios de red. En el modelo de referencia OSI, hay siete capas numeradas, cada una de las cuales ilustra una función de red específica.

Fuente: Cisco-CCNA1

COMUNICACIONES DE PAR A PAR

Para que los datos puedan viajar desde el origen hasta su destino, cada capa del modelo OSI en el origen debe comunicarse con su capa par en el lugar destino. Esta forma de comunicación se conoce como de par-a-par. Durante este proceso, los protocolos de cada capa intercambian información, denominada unidades de datos de protocolo (PDU). Cada capa de comunicación en el computador origen se comunica con un PDU específico de capa, y con su capa par en el computador destino, como lo ilustra la figura

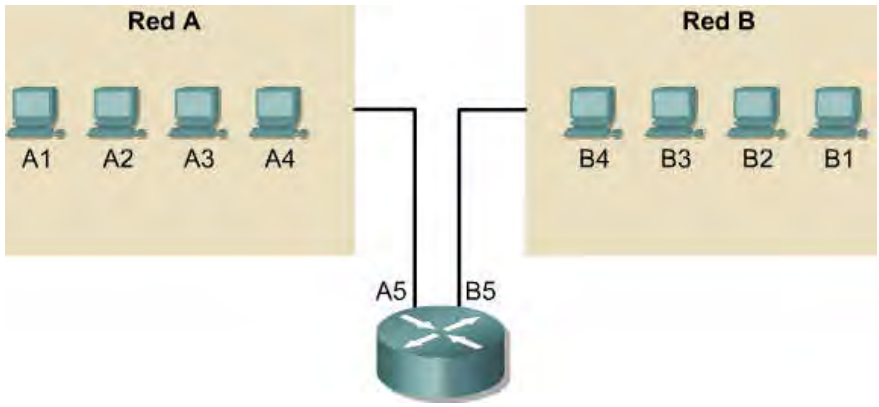
Los paquetes de datos de una red parten de un origen y se envían a un destino. Cada capa depende de la función de servicio de la capa OSI que se encuentra debajo de ella. Para brindar este servicio, la capa inferior utiliza el encapsulamiento para colocar la PDU de la capa superior en su campo de datos, luego le puede agregar cualquier encabezado e información final que la capa necesite para ejecutar su función. Posteriormente, a medida que los datos se desplazan hacia abajo a través de las capas del modelo OSI, se agregan encabezados e información final adicionales.



Fuente: Cisco-CCNA1

DIRECCIONAMIENTO IP

Para que dos sistemas se comuniquen, se deben poder identificar y localizar entre sí. Aunque las direcciones de la Figura no son direcciones de red reales, representan el concepto de agrupamiento de las direcciones. Este utiliza A o B para identificar la red y la secuencia de números para identificar el host individual.



Fuente: Cisco-CCNA1

Un computador puede estar conectado a más de una red. En este caso, se le debe asignar al sistema más de una dirección. Cada dirección identificará la conexión del computador a una red diferente. No se suele decir que un dispositivo tiene una dirección sino que cada uno de las interfaces de dicho dispositivo tienen una dirección en una red.

Cada computador conectado a una red TCP/IP debe recibir un identificador exclusivo o una dirección IP. Esta dirección, que opera en la Capa 3, permite que un computador localice otro computador en la red.

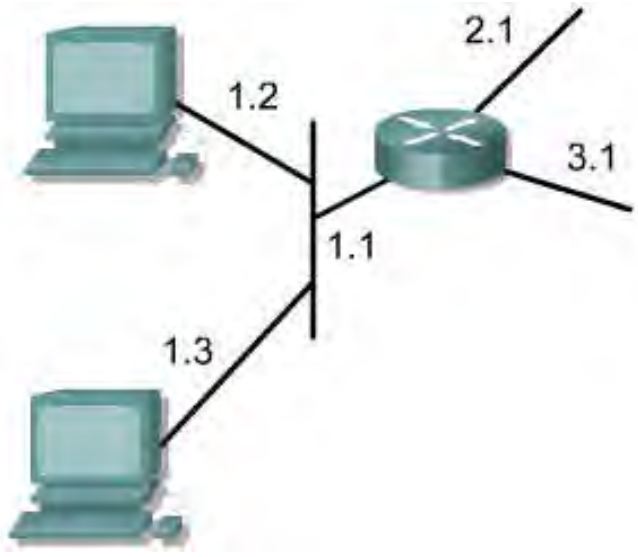
Todos los computadores también cuentan con una dirección física exclusiva, conocida como dirección MAC. Estas son asignadas por el fabricante de la tarjeta de intefaz de la red. Las direcciones MAC operan en la Capa 2 del modelo OSI.

Una dirección IP es una secuencia de unos y ceros de 32 bits. Para que el uso de la dirección IP sea más sencillo, en general, la dirección aparece escrita en forma de cuatro números decimales separados por puntos.

Binario: 11000000.10101000.000000001.00001000 y 11000000.10101000.00000001.00001001
Decimal: 192.168.1.8 y 192.168.1.9

DIRECCIONAMIENTO IPV4

Cada dirección IP consta de dos partes. Una parte identifica la red donde se conecta el sistema y la segunda identifica el sistema en particular de esa red.



Fuente: Cisco-CCNA1

Las direcciones IP se dividen en clases para definir las redes de tamaño pequeño, mediano y grande. Las direcciones Clase A se asignan a las redes de mayor tamaño. Las direcciones Clase B se utilizan para las redes de tamaño medio. Las de Clase C para redes pequeñas.

Clase de dirección	Cantidad de redes	Cantidad de hosts por red
A	126 *	16,777,216
B	16, 384	65,535
C	2,097,152	254
D (Multicast)	No es aplicable	No es aplicable

* El intervalo de direcciones 127.x.x.x está reservado como dirección de loopback, con propósitos de prueba y diagnóstico.

Fuente: Cisco-CCNA1

Clase A	Red	Host		
Octet	1	2	3	4

Clase B	Red		Host	
Octet	1	2	3	4

Clase C	Red			Host
Octet	1	2	3	4

Fuente: Cisco-CCNA1

Clase de dirección IP	Intervalo de dirección IP (Valor decimal d
Clase A	1-126 (00000001-01111110) *
Clase B	128-191 (10000000-10111111)
Clase C	192-223 (11000000-11011111)
Clase D	224-239 (11100000-11101111)
Clase E	240-255 (11110000-11111111)

Determine la clase basándose en el valor decimal del primer octeto
* 127 (01111111) es una dirección clase A reservada para pruebas loopback y no puede ser asignada a una red

Fuente: Cisco-CCNA1

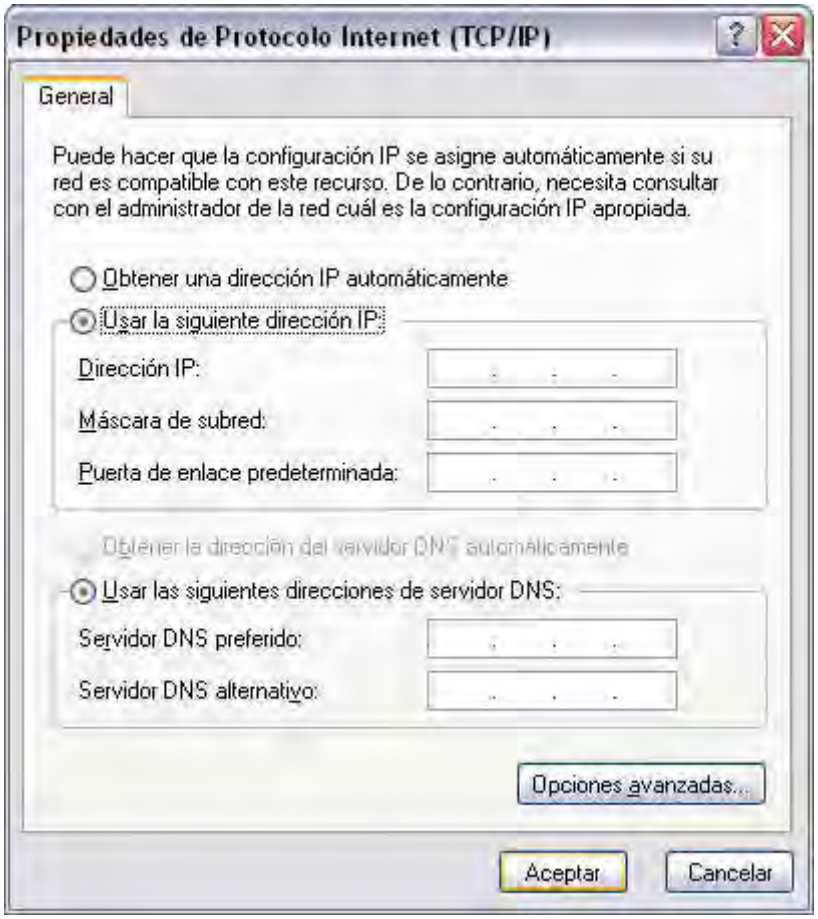
DIRECCIONES IP PÚBLICAS Y PRIVADAS

Clase	intervalo de direcciones internas RFC 1918
A	10.0.0.0 to 10.255.255.255
B	172.16.0.0 to 172.31.255.255
C	192.168.0.0 to 192.168.255.255

Fuente: Cisco-CCNA1

ASIGNACIÓN ESTÁTICA DE UNA DIRECCIÓN IP

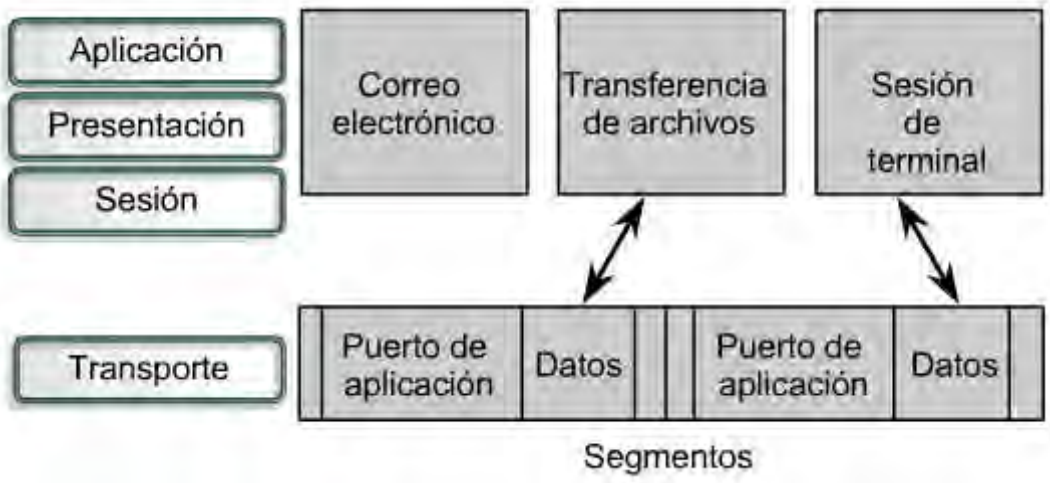
La asignación estática funciona mejor en las redes pequeñas con poca frecuencia de cambios. De forma manual, el administrador del sistema asigna y rastrea las direcciones IP para cada computador, impresora o servidor de una red interna. Es fundamental llevar un buen registro para evitar que se produzcan problemas con las direcciones IP repetidas. Esto es posible sólo cuando hay una pequeña cantidad de dispositivos que rastrear.



CAPA DE TRANSPORTE

El flujo de datos de la capa de transporte es una conexión lógica entre los puntos de terminación de una red. Sus tareas principales son las de transportar y regular el flujo de información desde el origen hasta el destino de forma confiable y precisa. La tarea principal de la Capa 4 es suministrar control de extremo a extremo usando ventanas deslizantes y brindar confiabilidad para los números de secuencia y los acuses de recibo.

La capa de transporte define la conectividad de extremo a extremo entre las aplicaciones del host. Los protocolos de la capa de transporte segmentan y reensamblan los datos mandados por las aplicaciones de capas superiores en el mismo flujo de datos de capa de transporte.



Fuente: Cisco-CCNA1

Los servicios de transporte incluyen los siguientes servicios básicos:

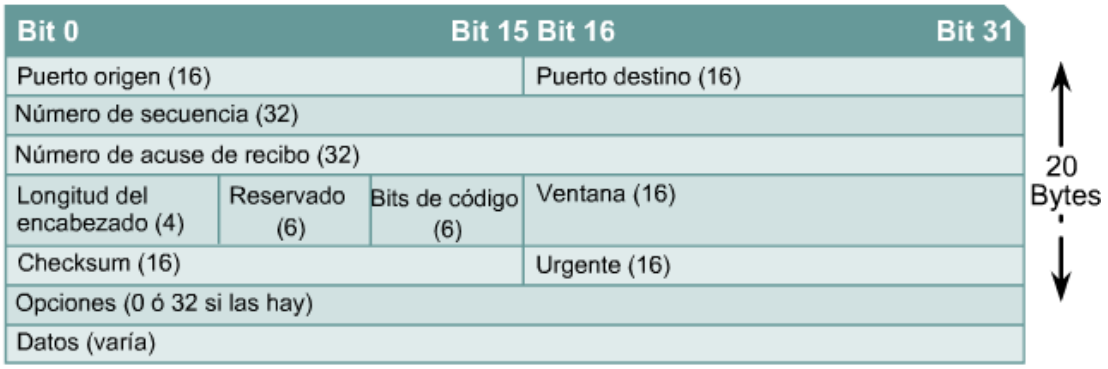
- Segmentación de los datos de las aplicaciones de capa superior
- Establecimiento de las operaciones de extremo a extremo
- Transporte de segmentos desde un host final a otro host final
- Control de flujo, suministrado por las ventanas deslizantes
- Confiabilidad, suministrada por los números de secuencia y los acuses de recibo

PROTOCOLO PARA EL CONTROL DE LA TRANSMISIÓN (TCP)

El Protocolo para el control de la transmisión (TCP) es un protocolo de Capa 4 orientado a conexión que suministra una transmisión de datos confiable. TCP forma parte de la pila del protocolo TCP/IP. En un entorno orientado a conexión, se establece una conexión entre ambos extremos antes de que se pueda iniciar la transferencia de información. TCP es responsable por la división de los mensajes en segmentos, reensamblándolos en la estación destino, reenviando cualquier mensaje que no se haya recibido y reensamblando mensajes a partir de los segmentos. TCP suministra un circuito virtual entre las aplicaciones del usuario final.

Los protocolos que usan TCP incluyen:

- FTP (Protocolo de transferencia de archivos)
- HTTP (Protocolo de transferencia de hipertexto)
- SMTP (Protocolo simple de transferencia de correo)
- Telnet



Fuente: Cisco-CCNA1

Las siguientes son las definiciones de los campos de un segmento TCP:

- **Puerto origen:** El número del puerto que realiza la llamada.
- **Puerto destino:** El número del puerto al que se realiza la llamada.
- **Número de secuencia:** El número que se usa para asegurar el secuenciamiento correcto de los datos entrantes.
- **Número de acuse de recibo:** Siguiente octeto TCP esperado.
- **HLEN:** La cantidad de palabras de 32 bits del encabezado.
- **Reservado:** Establecido en cero.
- **Bits de código:** Funciones de control, como configuración y terminación de una sesión.
- **Ventana:** La cantidad de octetos que el emisor está dispuesto a aceptar.
- **Checksum (suma de comprobación):** Suma de comprobación calculada a partir de los campos del encabezado y de los datos.
- **Indicador de mensaje urgente:** Indica el final de la transmisión de datos urgentes.
- **Opción:** Una opción definida actualmente, tamaño máximo del segmento TCP.
- **Datos:** Datos de protocolo de capa superior.

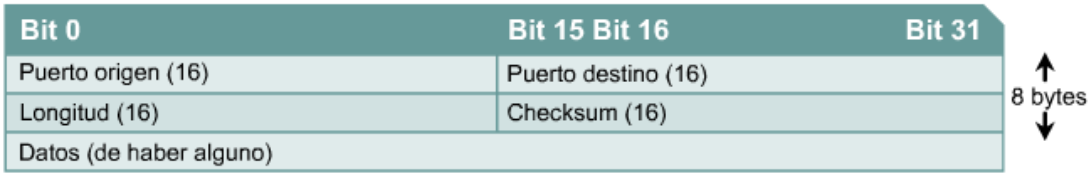
PROTOCOLO DE DATAGRAMA DE USUARIO (UDP)

El Protocolo de datagrama de usuario (UDP: User Datagram Protocol) es el protocolo de transporte no orientado a conexión de la pila de protocolo TCP/IP. El UDP es un protocolo simple que intercambia datagramas sin acuse de recibo ni garantía de entrega. El procesamiento de errores y la retransmisión deben ser manejados por protocolos de capa superior.

El UDP no usa ventanas ni acuses de recibo de modo que la confiabilidad, de ser necesario, se suministra a través de protocolos de la capa de aplicación. El UDP está diseñado para aplicaciones que no necesitan ensamblar secuencias de segmentos.

Los protocolos que usan UDP incluyen:

- TFTP (Protocolo trivial de transferencia de archivos)
- (SNMP) Protocolo simple de administración de red
- DHCP (Protocolo de configuración dinámica del host)
- DNS (Sistema de denominación de dominios)



Fuente: Cisco-CCNA1

Las siguientes son las definiciones de los campos de un segmento UDP:

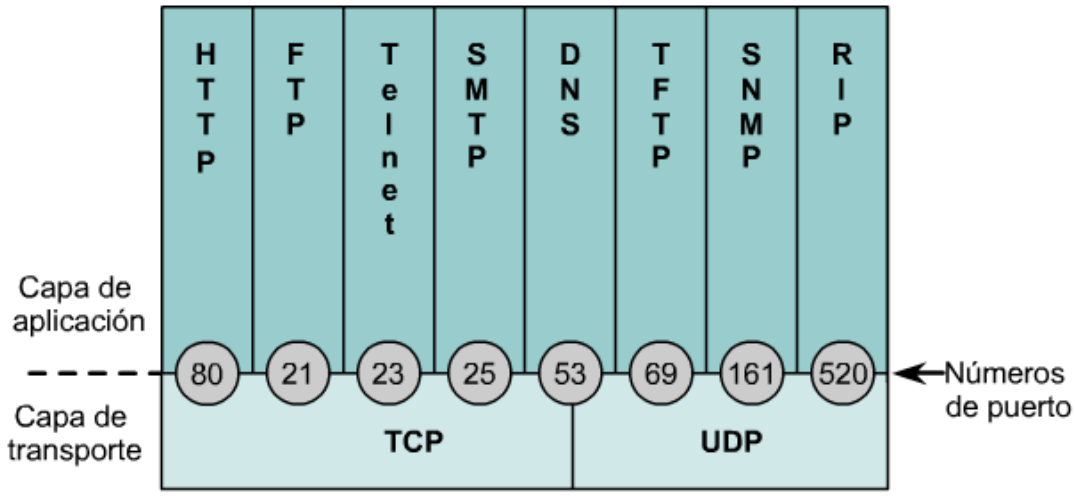
- **Puerto origen:** Número del puerto que realiza la llamada
- **Puerto destino:** Número del puerto al que se realiza la llamada
- **Longitud:** Número de bytes que se incluyen en el encabezado y los datos
- **Checksum (suma de comprobación):** Suma de comprobación calculada a partir de los campos del encabezado y de los datos.
- **Datos:** Datos de protocolo de capa superior.

NÚMEROS DE PUERTO TCP Y UDP

Tanto TCP como UDP utilizan números de puerto (socket) para enviar información a las capas superiores. Los números de puerto se utilizan para mantener un registro de las distintas conversaciones que atraviesan la red al mismo tiempo.

Los programadores del software de aplicación han aceptado usar los números de puerto conocidos que emite la Agencia de Asignación de Números de Internet (IANA: Internet Assigned Numbers Authority). Cualquier conversación dirigida a la aplicación FTP usa los números de puerto estándar 20 y 21.

A las conversaciones que no involucran ninguna aplicación que tenga un número de puerto bien conocido, se les asignan números de puerto que se seleccionan de forma aleatoria dentro de un rango específico por encima de 1023. Algunos puertos son reservados, tanto en TCP como en UDP, aunque es posible que algunas aplicaciones no estén diseñadas para admitirlos.



Fuente: Cisco-CCNA1

Los números de puerto tienen los siguientes rangos asignados:

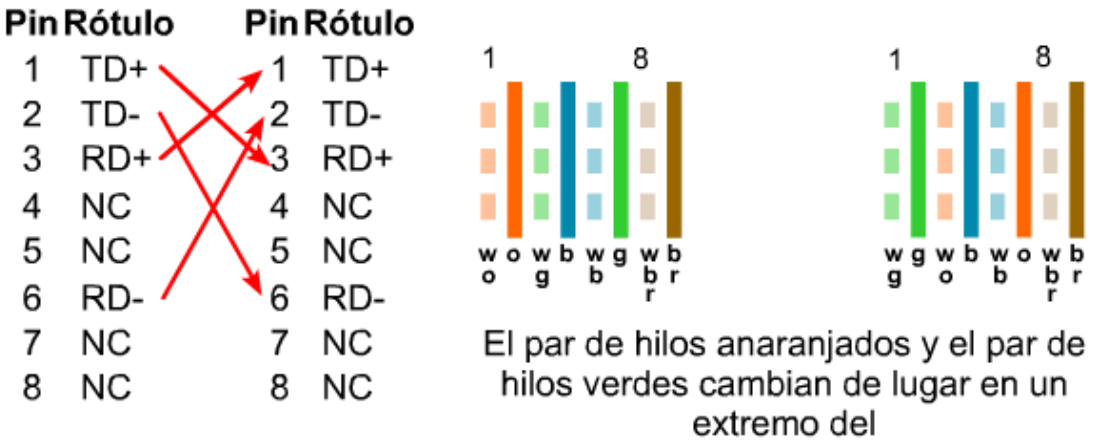
- Los números inferiores a 1024 corresponden a números de puerto bien conocidos.
- Los números superiores a 1024 son números de puerto asignados de forma dinámica.

CABLE UTP

EIA/TIA especifica el uso de un conector RJ-45 para cables UTP. Las letras RJ significan "registered jack" (jack registrado), y el número 45 se refiere a una secuencia específica de cableado. El conector transparente RJ-45 muestra ocho hilos de distintos colores. Cuatro de estos hilos conducen el voltaje. Los otros cuatro hilos están conectados a tierra.

Fuente: Cisco-CCNA1

En un cable de conexión cruzada, los conectores RJ-45 de ambos extremos muestran que algunos hilos de un extremo del cable están cruzados a un pin diferente en el otro extremo del cable. La Figura muestra que los pins 1 y 2 de un conector se conectan respectivamente a los pins 3 y 6 de otro.



Fuente: Cisco-CCNA1

SISTEMAS DISTRIBUIDOS

Un Sistema Distribuido es aquel en el que los componentes (hardware y software) localizados en computadores, conectados en red, comunican y coordinan sus acciones mediante el paso de mensajes.

Razones para Distribuir:

- Distribución Funcional.- las computadoras tienen diferencias funcionales.
  - Cliente/Servidor
  - Recaudación de datos/procesamiento de datos
- Distribución inherente al dominio de la aplicación.-
  - Cajas registradoras
  - Sistemas de inventarios
- Distribución / balanceo.- asignar tareas a procesadores tal que todo el desempeño del sistema sea optimizado.
- Replicación del poder de procesamiento.- procesadores independientes trabajan con la misma tarea.
- Separación física.- sistemas que confían en el echo de que las computadoras estén físicamente separadas
- Económicos.- colecciones de microprocesadores ofrecen una mejor cuota precio/desempeño que grandes mainframes.

Los retos a los cuales se enfrentan los Sistemas Distribuidos son:

- Heterogeneidad
- Extensibilidad
- Seguridad
- Escalabilidad
- Tratamiento de fallos
- Concurrencia

Aplicaciones:

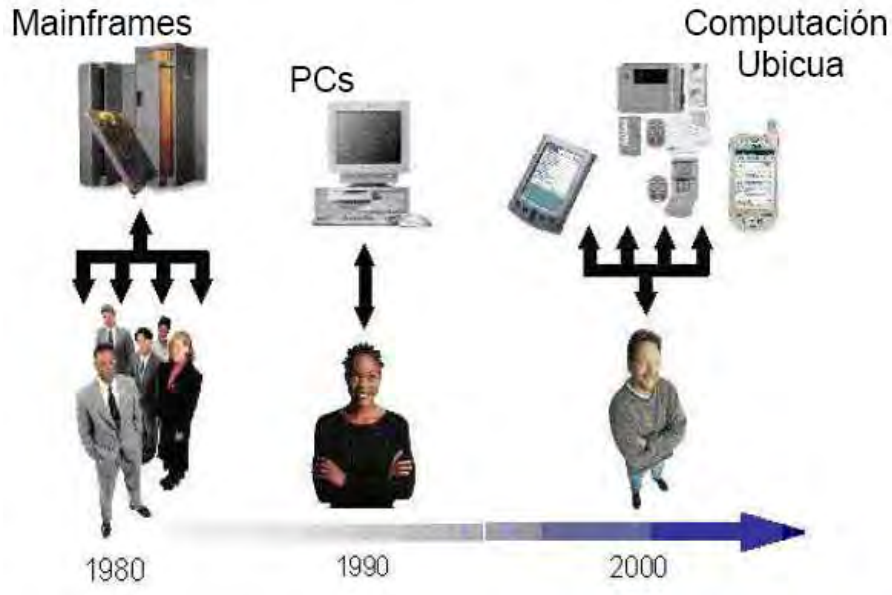
- Entorno de empresa: redes corporativas e *intranets.*
  - Sustituyen a los clásicos *mainframes*
- Entornos que requieren procesamiento paralelo.
  - Sustituyen a costosos *supercomputadores*
- Servicios con alta disponibilidad y rendimiento.
- Sistemas distribuidos de gestión de bases de datos.
- Aplicaciones multimedia.
- Sistemas industriales distribuidos y aplicaciones de control.

MIDDLEWARE

Es un término que abarca a todo el software necesario para el soporte de interacciones entre Clientes y Servidores principalmente en aplicaciones distribuidas. Se puede considerar como el enlace que permite que un cliente obtenga un servicio de un servidor.

Se encuentra representado por procesos u objetos que actúan en un conjunto de computadoras y que se comunican con el fin de proporcionar soporte para compartición de recursos en un sistema distribuido.

A continuación se ilustra la evolución de los middleware y sus enfoques de aplicación.



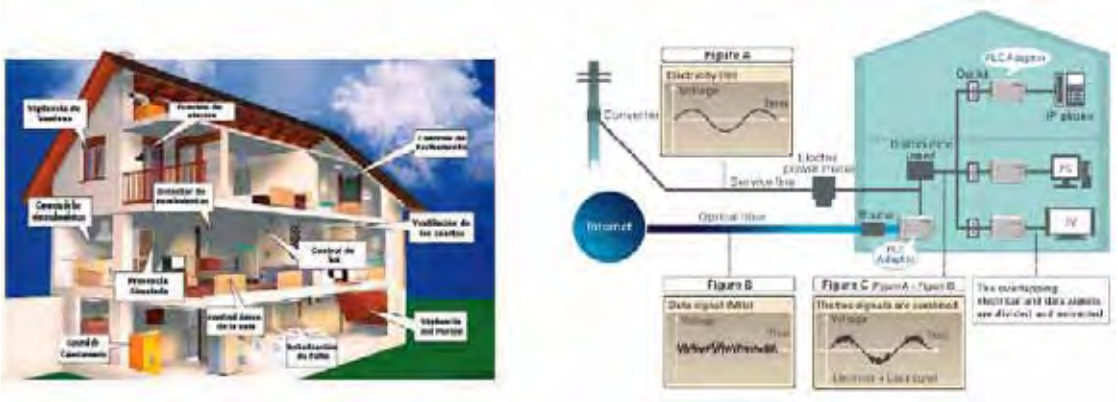
Computación Ubicua: (Automovil)

- Dispositivos de múltiples proósitos.
- Interfaces
- Operaciones concurrentes
- Computadoras (bus local e internet)
- Wireless corto alcance (llaves) y área amplia (celular)
- Sistemas de seguridad
- Funciones personalizadas (no PC)

Computación Ubicua: (Hogar)

- Teléfonos móviles
- Computadoras
- Aparatos de entretenimiento
- Control de puertas y ventanas
- Sistemas de aire acondicionado y calefacción
- Dispositivos domésticos
- Sistemas de seguridad

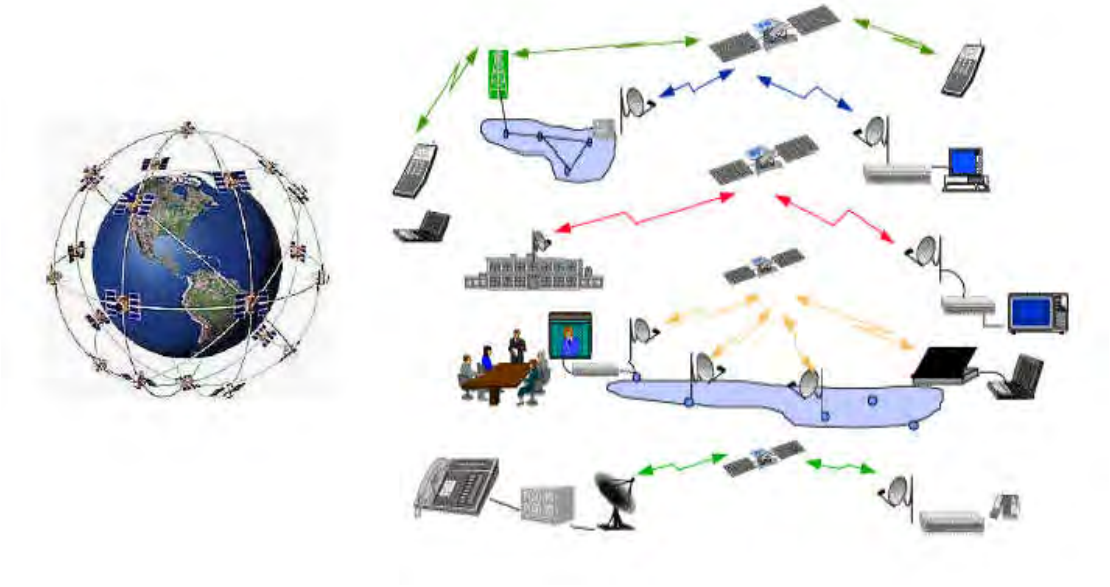
Transmisión de información vía líneas de potencia e inalámbricas.



Computación Ubicua: (Personal)

- Teléfonos móviles y tradicionales
- Localizadores
- Fax
- Computadoras
- Video grabadoras
- Consolas de juegos
- Cámaras digitales
- Grabadoras de música y video

Tecnologías GPRS, UMTS y Sistemas de localización GPS.



COMUNICACIONES BASADAS EN EL PROTOCOLO TCP

El protocolo TCP (Transmission Control Protocol) funciona en el capa de transporte, basándose en el protocolo de red IP (Internet Protocol).

TCP está orientado a conexión y proporciona comunicaciones fiables basadas en mecanismos de red que gestionan el control de flujo de paquetes. El control de flujo evita que los nodos que envían información puedan saturar a los que la reciben; para lograr este objetivo, el protocolo TCP utiliza de manera interna un mecanismo de sincronización entre cliente y servidor.

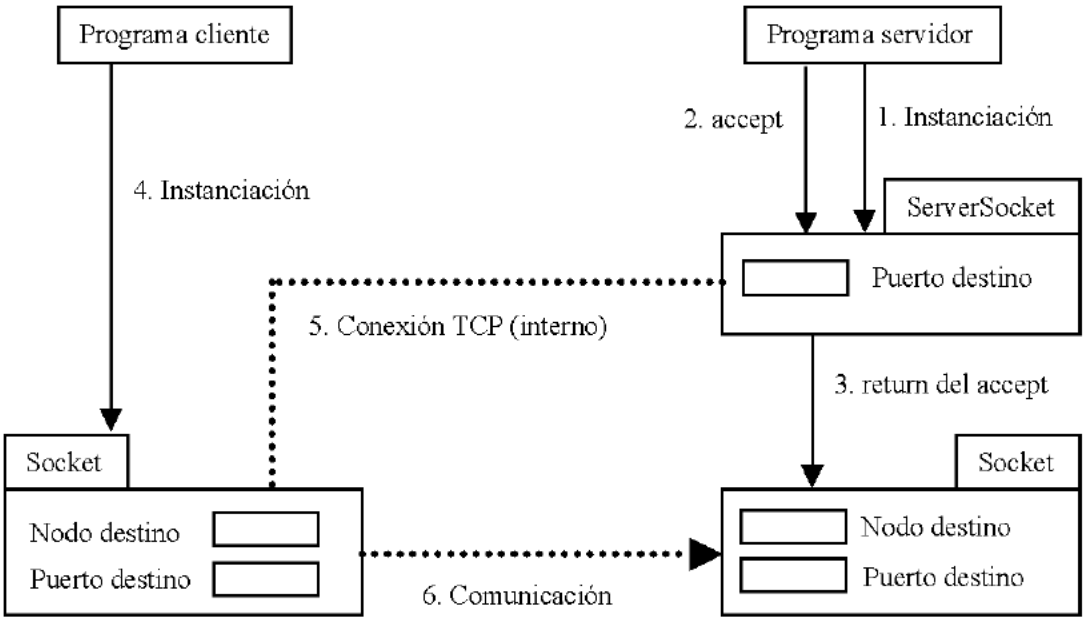
En el caso de transmitir una gran cantidad de información (por ejemplo, un fichero de video), y si no es necesario tiempo real, TCP es un protocolo adecuado, puesto que el tiempo para establecer la comunicación es despreciable respecto al utilizado para transmitir los datos. Por el contrario si es necesario una gran cantidad de comunicaciones cortas en las que la fiabilidad no es muy importante, TCP no es un protocolo adecuado; para este caso es mucho mejor UDP.

En Java, las comunicaciones TCP se realizan utilizando la abstracción de *socket.* Los sockets permiten establecer y programar comunicaciones sin tener que conocer los niveles inferiores sobre los que se asientan.

Para identificar el destino de los paquetes de datos, los sockets utilizan los conceptos de dirección y puerto.

Dado que un mismo nodo puede hacerse cargo de manejar varias comunicaciones diferentes de datos (ligadas a distintos servicios), existe la necesidad de proporcionar un mecanismo que permita distinguir los paquetes que llegan relacionados con los distintos servicios ofrecidos; este se logra con los puertos. Los puertos se representan con valores enteros, que no deben coincidir para las diferentes aplicaciones. Por ejemplo: los datos que se envían a un nodo con dirección IP 138.100.57.45 y puerto 6000 son tratados por aplicaciones diferentes que los enviados al mismo nodo 138.100.57.45 y puerto 7000.

ESTABLECIMIENTO DE COMUNICACIONES TCP



1.- El programa que proporciona el servicio (programa servidor) crea una instancia de la clase *ServerSocket,* indicando el puerto asociado al servicio:

ServerSocket SocketServidor = new ServerSocket (Puerto);

2.- El programa que proporciona el servicio invoca el método *accept* sobre el socket de tipo *ServerSocket.* Este método bloquea el programa hasta que se produce una conexión por parte de un cliente:

SocketServidor.accept();

3.- El método *accept* devuelve un socket de tipo *Socket,* con el que se realiza la comunicación de datos del cliente al servidor:

Socket ComunicaConCliente = SocketServidor. accept();

4.- El programa cliente crea una instancia de tipo *Socket,* a la que proporciona la dirección del nodo destino y el puerto del servicio:

Socket SocketCliente = new Socket(Destino, Puerto);

5.- Internamente, el socket del cliente trata de establecer comunicación con el socket de tipo *ServerSocket* existente en el servidor.

6.- Con los pasos anteriores completados se puede empezar a comunicar datos entre el cliente (o clientes) y el servidor.

TRANSMISION DE DATOS

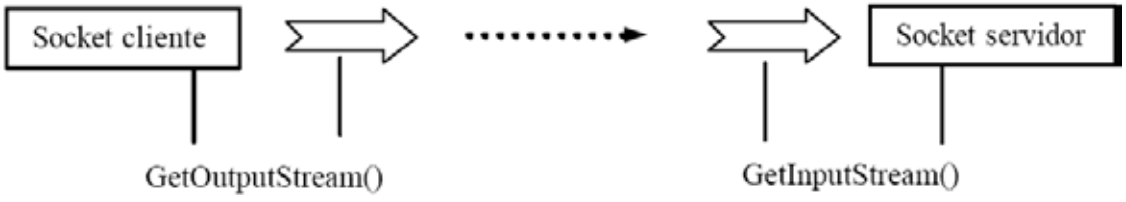
TCP es un protocolo especialmente útil cuando se desea transmitir un flujo de gran tamaño. Los sockets de Java están diseñados para transmitir y recibir datos a través de los Streams definidos en el paquete java.io.

La clase *Socket* contiene dos métodos importantes que se emplean en el proceso de transmisión de flujos de datos:

InputStream getInputStream()
OutputStream getOutputStream()

Empleando el Stream de salida del socket del cliente y el Stream de entrada del socket del servidor es posible establecer un flujo de datos continuo a través de la conexión TCP establecida:

OutputStream SALIDA = SocketCliente.getOutputStream();
InputStream ENTRADA = SocketServidor.getInputStream();



Las clases *OutputStream* e *InputStream* son abstractas, por lo que no se pueden emplear directamente todos sus métodos. En general se utiliza otras clases más especializadas que permiten trabajar con flujos de datos:

- *DataOutputStream*
- *DataInputStream*
- *FileOutputStream,*
- *FileInputStream*

DataInputStream Flujo = *new DataInputStream(*FlujoDeEntrada*);*

DataOutputStream Flujo = *new DataOutputStream(*FlujoDeSalida*);*

ServerSocket	
Métodos principales	**Acción**
Socket accept()	Espera a que se realice una conexión y devuelve un socket para comunicarse con el cliente
void bind(SocketAddress a)	Asigna la dirección establecida al socket creado con *accept*, si no se utiliza este método se asigna automáticamente una dirección temporal
void close()	Cierra el socket
InetAddress getInetAddress()	Devuelve la dirección a la que está conectada el socket
int getLocalPort()	Devuelve el número de puerto asociado al socket
int getSoTimeout()	Devuelve el valor en milisegundos que el socket espera al establecimiento de comunicación tras la ejecución de *accept*
void setSoTimeout(int ms)	Asigna el número de milisegundos que el socket espera al establecimiento de comunicación tras la ejecución de *accept*

Socket	
Métodos principales	**Acción**
void bind(SocketAddress a)	Asigna la dirección establecida al socket creado con *accept*, si no se utiliza este método se asigna automáticamente una dirección temporal
void close()	Cierra el socket
void connect(SocketAddress a)	Conecta el socket a la dirección de servidor establecida
void connect(SocketAddress a, int ms)	Conecta el socket a la dirección de servidor establecida, esperando un máximo de *ms* milisegundos
InetAddress getInetAddress()	Devuelve la dirección a la que está conectada el socket
InputStream getInputStream()	Devuelve el stream de entrada asociado al socket
int getLocalPort()	Devuelve el número de puerto asociado al socket
OutputStream getOutputStream()	Devuelve el stream de salida asociado al socket
int getPort()	Devuelve el valor del Puerto remoto al que está conectado
int getSoLinger()	Devuelve el número de milisegundos que se espera a los datos después de cerrar (*close*) el socket
int getSoTimeout()	Devuelve el valor en milisegundos que el socket espera al establecimiento de comunicación tras la ejecución de *accept*

Ejemplo No. 1:

Desarrollar una aplicación cliente/servidor básica en la que se pueda escribir un dato por una red punto a punto clase C privada.

Máquina cliente:

```
import java.net.*;
import java.io.*;

public class Main {
    public static void main(String[] args) {
        OutputStream      FlujoDeSalida;
        DataOutputStream  Flujo;
        try
        {
            Socket Cliente       = new Socket("192.168.0.101",8000);
            FlujoDeSalida        = Cliente.getOutputStream();
            Flujo                = new DataOutputStream(FlujoDeSalida);
            //-------ESCRITURA DE STRINGS------------
            Flujo.writeBytes("ELECTRONICA"); //(1)
            //Flujo.writeUTF("ELECTRONICA"); //(2)
            //-------ESCRITURA DE ENTEROS------------
            //Flujo.writeInt(12);
            //-------ESCRITURA DE DECIMALES----------
            //Flujo.writeDouble(12.33);
            //---------------------------------------
            System.out.println("Dato enviado...");
            Cliente.close();
            System.out.println("Comunicacion terminada...");
        }
        catch(Exception e)
        {
            System.out.println("ERROR: "+e);
        }

    }
}
```

Máquina servidor

```
import java.net.*;
import java.io.*;
public class Main {
    public static void main(String[] args) {
        InputStream     FlujoDeEntrada;
        DataInputStream Flujo;
        try
        {
            ServerSocket SocketServidor = new ServerSocket(8000);
            System.out.println("En espera...");
            Socket ComunicaConCliente   = SocketServidor.accept();
            System.out.println("Comunicación establecida...");
            FlujoDeEntrada              = ComunicaConCliente.getInputStream();
            Flujo                       = new DataInputStream(FlujoDeEntrada);
            System.out.println("Procende de: "+ComunicaConCliente.getInetAddress());
            //----------LECTURA DE STRINGS------------------
            System.out.println(Flujo.readLine()); //(1)
            //System.out.println(Flujo.readUTF());//(2)
            //----------LECTURA DE ENTEROS------------------
            //System.out.println(Flujo.readInt());
            //----------LECTURA DE DECIMALES----------------
            //System.out.println(Flujo.readDouble());
            //----------------------------------------------
            System.out.println("Dato recibido...");
            ComunicaConCliente.close();
            SocketServidor.close();
            System.out.println("Comunicación terminada...");
        }
        catch(Exception e)
        {
            System.out.println("ERROR:"+e);
        }
    }
}
```

Ejemplo No. 2:

Modificar el programa anterior de tal manera que se creen dos clases abstractas: “TCPServidor” y “TCPCliente”, las cuales detallaran los sockets necesarios. Además crear dos métodos abstractos “Comunicación” que implementara cada aplicación.

```
import java.net.*;
import java.io.*;
public abstract class TCPCliente {
    OutputStream      FlujoDeSalida;
    DataOutputStream  Flujo;
    Socket Cliente;
    String Host;
    int Puerto;
    TCPCliente(String Host, int Puerto)
    {
        this.Host  = Host;
        this.Puerto= Puerto;
    }
    void Configuracion()
    {
        try
        {
            Cliente               = new Socket(Host,Puerto);
            FlujoDeSalida         = Cliente.getOutputStream();
            Flujo                 = new DataOutputStream(FlujoDeSalida);

            Comunicacion(Flujo);
            System.out.println("Dato enviado...");

            Cliente.close();
            System.out.println("Comunicacion terminada...");
        }
        catch(Exception e)
        {
            System.out.println("ERROR: "+e);
        }
    }
    abstract void Comunicacion(DataOutputStream  Flujo);
}
```

```
import java.net.*;
import java.io.*;
public abstract class TCPServidor {
    InputStream     FlujoDeEntrada;
    DataInputStream Flujo;
    int Puerto;
    TCPServidor(int Puerto)
    {
        this.Puerto=Puerto;
    }
    void Configuracion()
    {
        try
        {
            ServerSocket SocketServidor = new ServerSocket(Puerto);
            Socket ComunicaConCliente   = SocketServidor.accept();
            System.out.println("En espera...");
            FlujoDeEntrada              = ComunicaConCliente.getInputStream();
            Flujo                       = new DataInputStream(FlujoDeEntrada);

            Comunicacion(Flujo);
            System.out.println("Dato recibido...");

            ComunicaConCliente.close();
            SocketServidor.close();
            System.out.println("Comunicacion terminada...");
        }
        catch(Exception e)
        {
            System.out.println("ERROR:"+e);
        }
    }
    abstract void Comunicacion(DataInputStream Flujo);
}
```

```
import java.io.*;
public class Cliente extends TCPCliente{
    String Mensaje;
    Cliente(String Host, int Puerto)
    {
        super(Host,Puerto);
    }
    void setMensaje(String Mensaje)
    {
        this.Mensaje=Mensaje;
    }
    void Comunicacion(DataOutputStream  Flujo)
    {
        try
        {
            //-------ESCRITURA DE STRINGS------------
            Flujo.writeBytes(Mensaje); //(1)
            //Flujo.writeUTF("ELECTRONICA"); //(2)
            //-------ESCRITURA DE ENTEROS------------
            //Flujo.writeInt(12);
            //-------ESCRITURA DE DECIMALES----------
            //Flujo.writeDouble(12.33);
            //---------------------------------------
        }
        catch(Exception e)
        {
            System.out.println("ERROR: "+e);
        }
    }
}
```

```
import java.io.*;
public class Servidor extends TCPServidor{
    Servidor(int Puerto)
    {
        super(Puerto);
    }
    void Comunicacion(DataInputStream Flujo)
    {
        try
        {
            //----------LECTURA DE STRINGS-----------------
            System.out.println(Flujo.readLine()); //(1)
            //System.out.println(Flujo.readUTF());//(2)
            //----------LECTURA DE ENTEROS-----------------
            //System.out.println(Flujo.readInt());
            //----------LECTURA DE DECIMALES---------------
            //System.out.println(Flujo.readDouble());
        }
        catch(Exception e)
        {
            System.out.println("ERROR:"+e);
        }
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Cliente Enlace = new Cliente("192.168.0.101",8000);
        Enlace.setMensaje("ESPE");
        Enlace.Configuracion();
    }
}
```

```
public class Main{
    public static void main(String[] args) {
        Servidor Enlace = new Servidor(8000);
        Enlace.Configuracion();
    }
}
```

Ejemplo No. 3:

Desarrollar un programa que permita transmitir un archivo utilizando el protocolo TCP.

```
import java.io.*;
public class ClienteFichero extends TCPCliente{
    FileInputStream Origen;
    int tamaño = 256;
    String archivo;
    ClienteFichero(String Host, int Puerto)
    {
        super(Host,Puerto);
    }
    void setArchivo(String archivo)
    {
        this.archivo=archivo;
    }
    void Comunicacion(DataOutputStream  Flujo)
    {
        byte memoria[]  = new byte[tamaño];
        int BytesLeidos = 0;
        try
        {
            Origen = new FileInputStream(archivo);
            do
            {
                BytesLeidos = Origen.read(memoria);
                Flujo.write(memoria,0,BytesLeidos);
            }
            while(BytesLeidos==tamaño);
        }
        catch(Exception e)
        {
            System.out.println("ERROR: "+e);
        }
    }
}
```

```
import java.io.*;
public class ServidorFichero extends TCPServidor{
    FileOutputStream Destino;
    int tamaño = 256;
    String archivo;
    ServidorFichero(int Puerto)
    {
        super(Puerto);
    }
    void setArchivo(String archivo)
    {
        this.archivo=archivo;
    }
    void Comunicacion(DataInputStream Flujo)
    {
        byte memoria[]  = new byte[tamaño];
        int BytesLeidos = 0;
        try
        {
            Destino = new  FileOutputStream(archivo);
            do
            {
                BytesLeidos = Flujo.read(memoria);
                Destino.write(memoria,0,BytesLeidos);
            }
            while(BytesLeidos==tamaño);
        }
        catch(Exception e)
        {
            System.out.println("ERROR: "+e);
        }
    }
}
```

```
public static void main(String[] args) {
    // TODO code application logic here
    ServidorFichero Enlace = new ServidorFichero(8000);
    Enlace.setArchivo("Llegada.txt");
    Enlace.Configuracion();
}
```

```
public static void main(String[] args) {
    // TODO code application logic here
    ClienteFichero Enlace = new ClienteFichero("192.168.0.101",8000);
    Enlace.setArchivo("Hola.txt");
    Enlace.Configuracion();
}
```

COMUNICACIONES BASADAS EN EL PROTOCOLO UDP

Al igual que TCP, el protocolo UDP funciona en la capa de transporte, basándose en el protocolo de red IP (Internet Protocol). IP proporciona comunicaciones no fiables y no orientadas a conexión, muy dependientes de saturaciones en la red, caídas de nodos, etc.

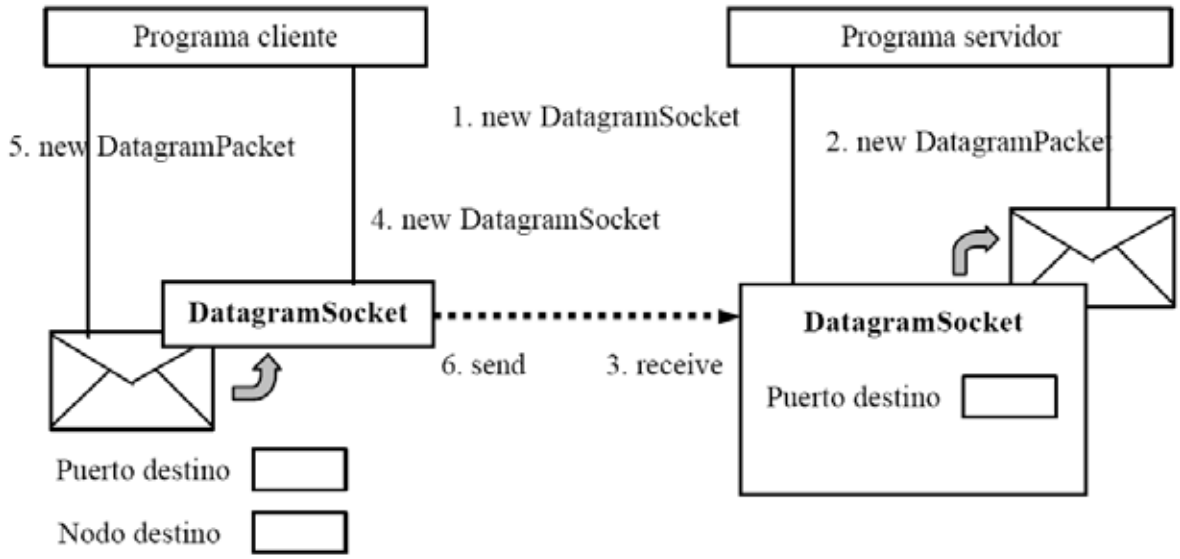
UDP no es un protocolo orientado a conexión ni a comunicaciones basadas en flujos de datos, por lo que Java no asocia los Streams a los datagramas.

En Java, las comunicaciones UDP se realizan utilizando la abstracción de *socket. Los sockets permiten establecer y programar comunicaciones sin tener que conocer los niveles inferiores sobre los que se asientan.*

ESTABLECIMIENTO DE COMUNICACIONES UDP

Java proporciona dos clases de especial importancia en la implementación de aplicaciones que hacen uso del protocolo UDP:

- Para realizar las comunicaciones *(DatagramSocket)*
- Para albergar los datos y dirección de destino *(DatagramPacket),* ambas en el paquete *java.net.*



1.- El programa que proporciona el servicio (programa servidor) crea una instancia de la clase *DatagramSocket,* indicando el Puerto asociado al servicio:

Datagramsocket MiSocket=new DatagrarrzSocket (4000);

2.- El programa servidor crea una instancia de la clase *DatagramPacket,* donde se guardarán los datos recibidos:

DatagramPacket Paquete = new DatagramPacket(buffer,buffer,length);

3.- El programa servidor invoca el método *receive* sobre el socket de tipo *DatagramSocket.* Este método, por defecto, bloquea el programa hasta que llegan los datos:

MiSocket.receive (Paquete);

4.- *El programa cliente crea una instancia de tipo DatagramSocket:*

DatagramSocket MiSocket = new DatagramSocket();

5.- El programa cliente crea una instancia de tipo *DatagramPacket,* proporcionándole los datos, además de la dirección y puerto de destino.

DatagramPacket Paquete = new DatagramPacket (buffer,Mensaje.length() ,InetAddress.getByName ("dirección IP"),4000)

6.- El programa que utiliza el servicio (programa cliente) invoca el método *send* sobre el socket de tipo *DatagramSocket:*

MiSocket.send (Paquete);

DatagramPacket	
Métodos principales	**Acción**
InetAddress getAddress()	Dirección del nodo remoto en la comunicación
byte[] getData()	Devuelve el mensaje que contiene el datagrama
int getLength()	Devuelve la longitud del mensaje del datagama
int getOffset()	Devuelve el desplazamiento que indica el comienzo del mensaje (dentro del array de bytes)
int getPort()	Devuelve el valor del puerto remoto
void setAddress(InetAddress d)	Establece el nodo remoto en la comunicación
void setData(byte[] Mensaje)	Establece el mensaje que contiene el datagrama
void setData(byte[] Mensaje, int Desplazamiento, int Longitud)	Establece el mensaje que contiene el datagrama, indicando su desplazamiento en el array de bytes y su longitud

DatagramSocket	
Métodos principales	**Acción**
void bind(SocketAddress a)	Asigna la dirección establecida al socket
void close()	Cierra el socket
void connect(SocketAddress a)	Conecta el socket a la dirección remota establecida
void connect(InetAddress a, int puerto)	Conecta el socket a la dirección establecida y el puerto especificado
void disconnect()	Desconecta el socket
InetAddress getInetAddress()	Devuelve la dirección a la que está conectada el socket
InetAddress getLocalAddress()	Devuelve la dirección del socket
int getLocalPort()	Devuelve el número de puerto asociado al socket
OutputStream getOutputStream()	Devuelve el stream de salida asociado al socket
int getPort()	Devuelve el valor del Puerto remoto al que está conectado
int getSoTimeout()	Devuelve el valor en milisegundos que el socket espera al establecimiento de comunicación
boolean isBound()	Indica si el socket está vinculado
boolean isClosed()	Indica si el socket está cerrado
boolean isConnected()	Indica si el socket está conectado
void setSoTimeout(int ms)	Indica el valor en milisegundos que el socket espera al establecimiento de comu nicación

Ejemplo No. 4:

Desarrollar una aplicación cliente/servidor básica en la que se pueda escribir un dato por una red punto a punto clase C privada.

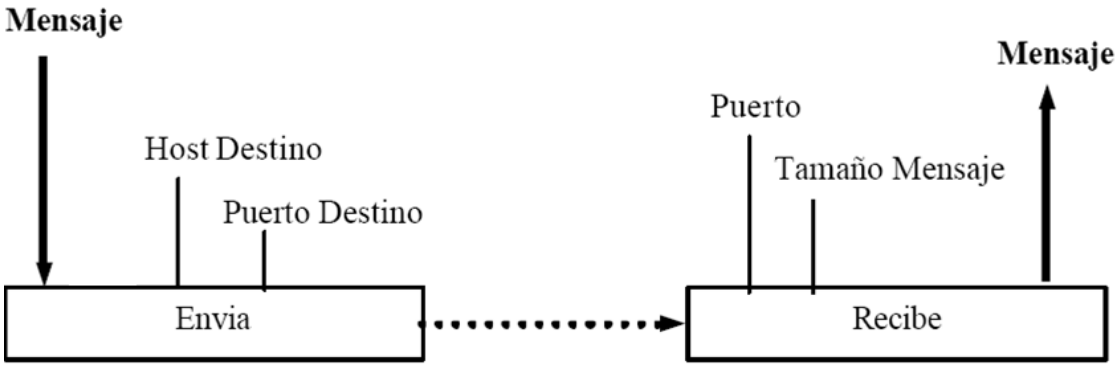
Máquina cliente:

```
import java.net.*;

public class Main {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        try
        {
            DatagramSocket MiSocket = new DatagramSocket();
            String Mensaje          = "ESPE";
            byte [] buffer          = new byte [Mensaje.length()];
            buffer                  = Mensaje.getBytes();
            DatagramPacket Paquete  = new DatagramPacket(buffer,buffer.length,
                    InetAddress.getByName("192.168.0.101"),8050);
            MiSocket.send(Paquete);
            MiSocket.close();
        }
        catch(Exception e)
        {
            System.out.println("ERROR:"+ e);
        }
    }
}
```

Ejemplo No. 5:

```
import java.net.*;
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        try
        {
            DatagramSocket MiSocket = new DatagramSocket(8050);
            byte [] buffer          = new byte [1024];
            DatagramPacket Paquete  = new DatagramPacket(buffer,buffer.length);
            MiSocket.receive(Paquete);
            System.out.println(new String(Paquete.getData()).substring(0,Paquete.getLength()));
            System.out.println("Procende de:"+Paquete.getAddress());
            MiSocket.close();
        }
        catch(Exception e)
        {
            System.out.println("ERROR:"+ e);
        }
    }
}
```

Utilizando programación orientada a objetos crear clases que proporcionen métodos que encapsulen una comunicación UDP: estableciendo, envío o recepción de Strings y cierre de la comunicación.




```
import java.net.*;

public class RecibeUDP {
    DatagramSocket MiSocket;
    DatagramPacket Paquete;
    byte [] buffer;
    public String Recibe (int Puerto, int tamaño)
    {
        try
        {
            MiSocket = new DatagramSocket(8050);
            buffer   = new byte [tamaño];
            Paquete  = new DatagramPacket(buffer,buffer.length);
            MiSocket.receive(Paquete);
            MiSocket.close();
        }
        catch(Exception e)
        {
            System.out.println("ERROR:"+ e);
        }
        return(new String(Paquete.getData()).substring(0,Paquete.getLength()));
    }
}
```

```
import java.net.*;

public class EnviaUDP {
    DatagramSocket MiSocket;
    byte [] buffer;
    public void Envia(String Mensaje, String HostDestino, int Puerto)
    {
        try
        {
            MiSocket = new DatagramSocket();
            buffer   = new byte [Mensaje.length()];
            buffer   = Mensaje.getBytes();
            DatagramPacket Paquete  = new DatagramPacket(buffer,buffer.length,
                    InetAddress.getByName(HostDestino),Puerto);
            MiSocket.send(Paquete);
            MiSocket.close();
        }
        catch(Exception e)
        {
            System.out.println("ERROR:"+ e);
        }
    }
}
```

```
public class Main {

    public static void main(String[] args) {
        EnviaUDP obj = new EnviaUDP();
        obj.Envia("ESPE","192.168.0.101", 8050);
    }
}
```

```
public class Main {

    public static void main(String[] args) {
        RecibeUDP obj = new RecibeUDP();
        System.out.println(obj.Recibe(8050, 1024));
    }
}
```

ACTIVIDADES DE REELABORACION

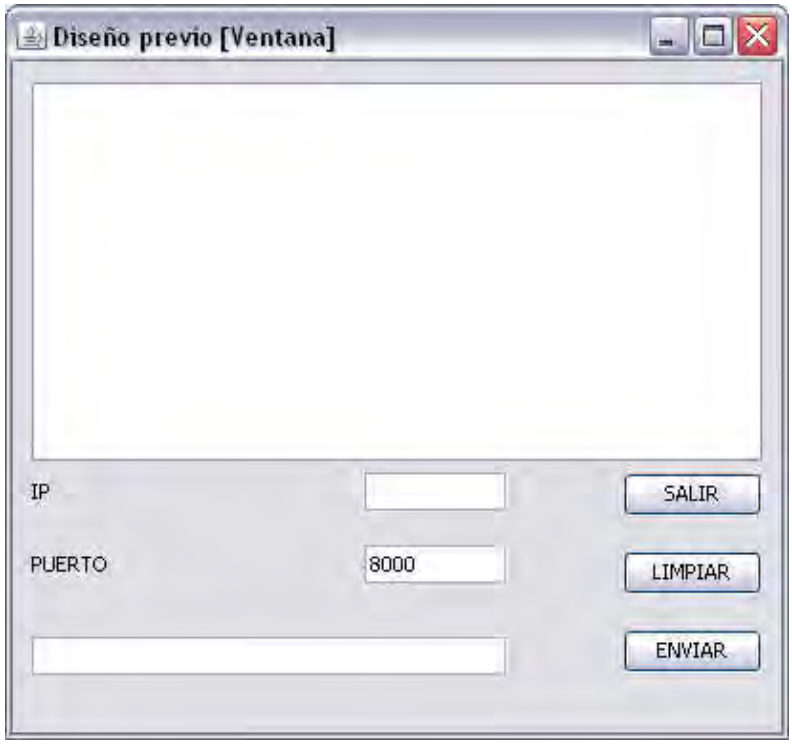
ACTIVIDADES PRÁCTICAS

1.- Establecer una red punto a punto, constituida por dos PC´s, en la cual la PC1 funciona como gateway. La red deberá tener direcciones privadas clase C.

Requerimientos: Cable de conexión cruzada UTP

2.- Repita el taller 1, pero modifique la interfase física cableada, a inalámbrica; considerando una topología Ad-Hoc.

3.- Realizar un programa que permita la comunicación bidireccional entre dos PC, mediante el protocolo TCP.



4.- Realizar un programa que permita la comunicación bidireccional entre dos PC, mediante el protocolo UDP.

ACTIVIDADES TEÓRICAS

1.- Que es el modelo TCP/IP? Cuál es su origen? Realice una comparación con el Modelo OSI.

2.- Existen sistemas operativos distribuidos?

3.- Que es computing cloud?

4.- Que son los Web Services y cuál es su función?

CAPITULO II

APLICACIONES WEB

APPLETS

Es un tipo especializado de panel, que se ejecuta dentro de otro programa (por ejemplo un navegador web), que le suministra la funcionalidad. El programa dentro del cual se ejecuta el applet le comunica los sucesos relevantes, es decir, cuando se tiene que crear, cuando se tiene que ejecutar, cuando se tiene que detener y cuando se tienen que destruir. Estos sucesos conforman el ciclo de vida de un applet

Por lo general los applets no incluyen ni método principal ni constructores. Su ejecución está determinada por unos métodos fijos del applet que ejecuta directamente el navegador en respuesta a cada uno de los sucesos del ciclo de vida. Por ejemplo cuando se crea un applet se ejecuta el método init() y cuando se destruye el método destroy().

CREACION DE UN APPLET

1. La clase principal del applet debe heredar de la clase JApplet/Applet que le proporciona la comunicación con el entorno y la funcionalidad básica de ejecución.
2. Se debe definir el método init() para inicializar todos los elementos del applet.
3. Se pueden definir los métodos start(), stop() y destroy() para obtener el comportamiento deseado.
4. SE debe crear una página web en HTML que contenga el applet.

HTML (HyperText Markup Language)

Las páginas web se construyen utilizando un lenguaje de marcado denominado lenguaje de marcdo de hipertexto (*HTML, HyperText Markup Language*).

Una página we es un archivo de texto creado con HTML en el que se puede diferenciar dos tipos de contenidos:

- Contenidos de información
- Marcas que identifican, delimitan y organizan los contenidos

HTML es un lenguaje de programación que se utiliza para la creación de páginas en la WWW. HTML se compone de una serie de comandos, que son interpretados por el visualizador, o programa que se utiliza para navegar por Internet. Los comandos HTML tienen una estructura muy básica. Son órdenes, contenidas entre los signos < y >.

ESTRUCTURA BASE DEL DOCUMENTO HTML

Es recomendable que todo fichero HTML siga la siguiente estructura:

```
<HTML>
<TITLE>Título de la ventana</TITLE>
<BODY>
.....comandos y texto.....
</body>
</HTML>
```

Estos comandos tienen una orden de inicio y otra de fin, que no es más que el mismo comando con el signo / antecediéndolo. Los comandos pueden figurar en letras mayúsculas o en minúsculas, indistintamente.

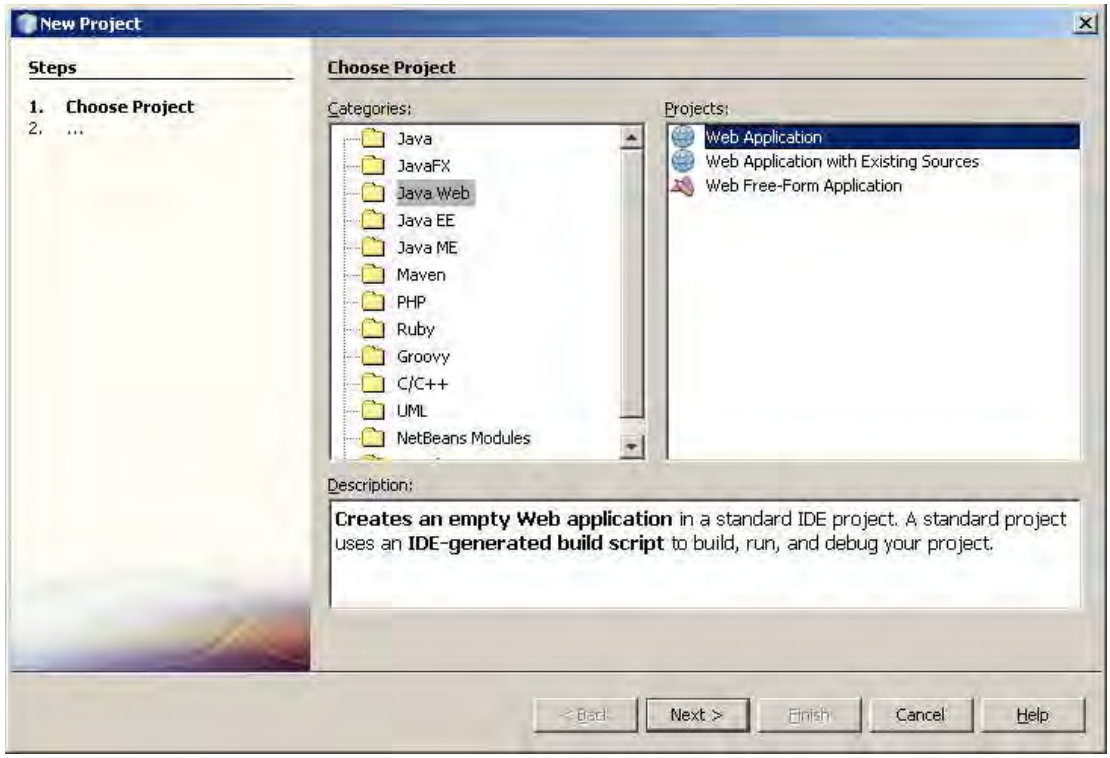
La secuencia lógica de lo anterior es lo siguiente:

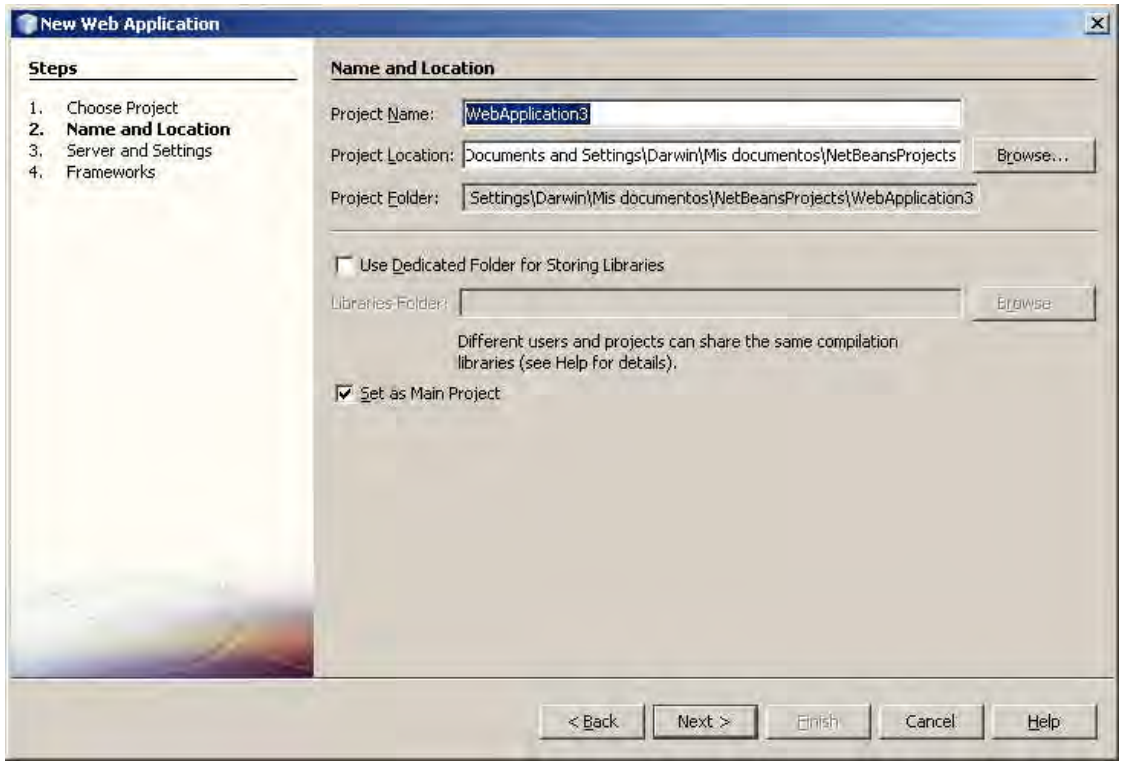
- Inicio de un documento HTML
- Inicio del título.
- Final del título.
- Inicio del cuerpo de la página.
- Fin del cuerpo de la página.
- Fin del documento HTML.

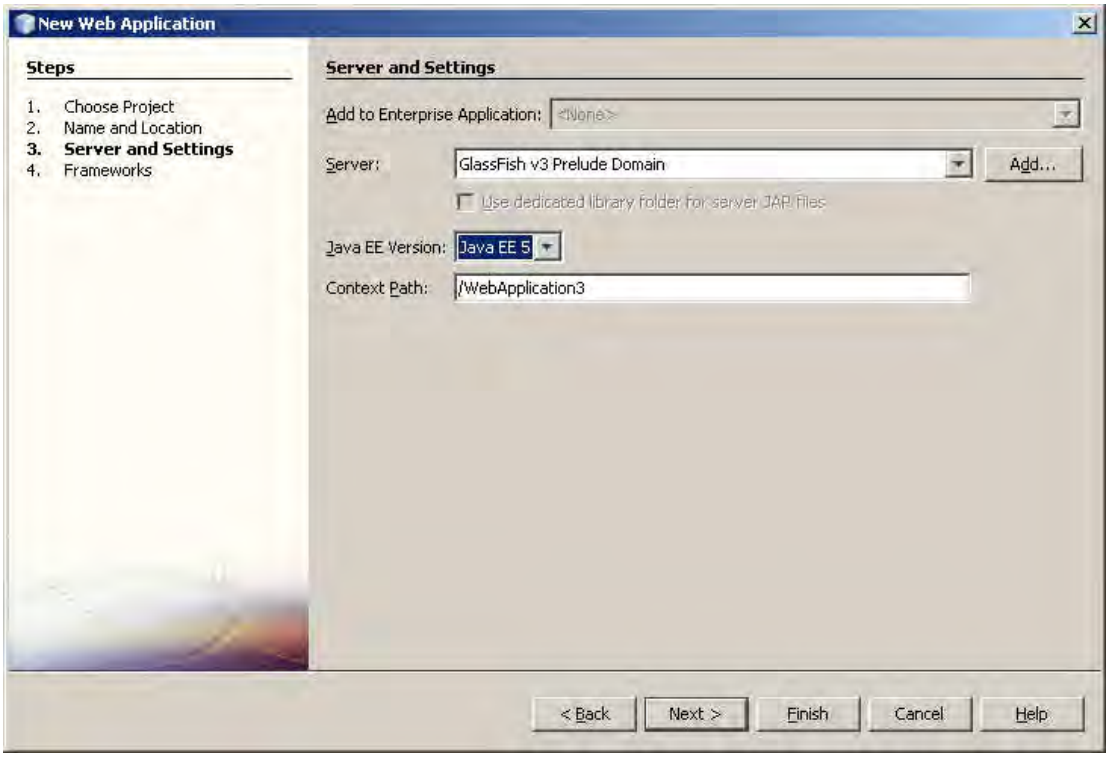
Además de las marcas de presentación, existe una marca especial <applet> que indica que hay un applet asociado a la página y especifica cuál es la clase que se debe ejecutar cuando se visualiza la página.

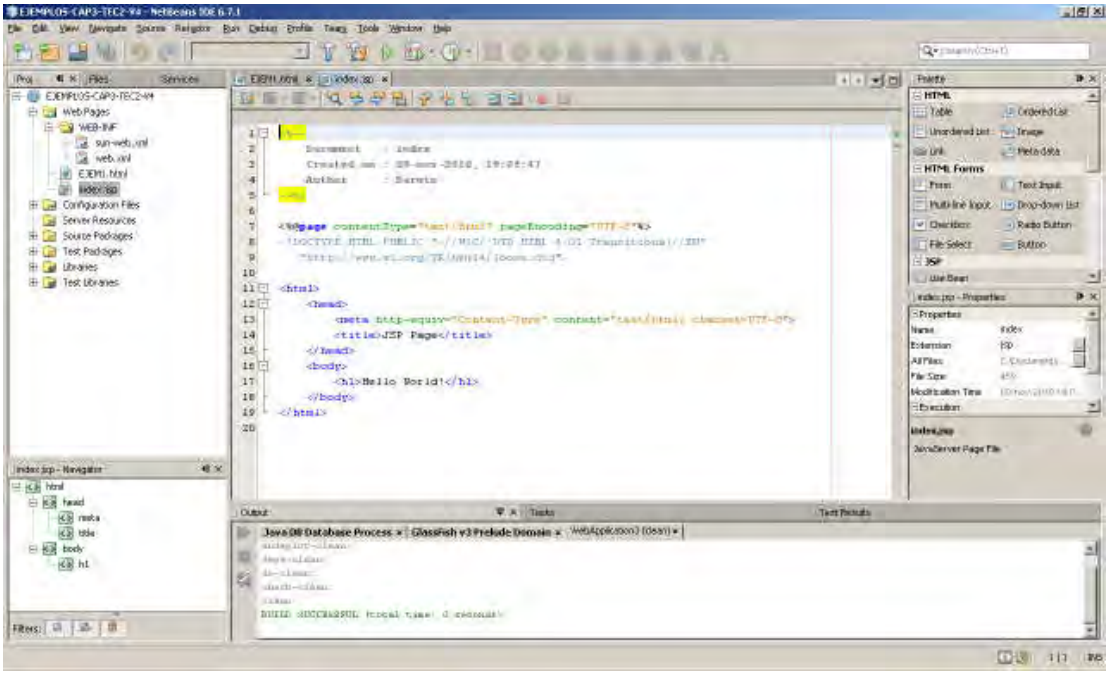
Ejemplo No. 1:

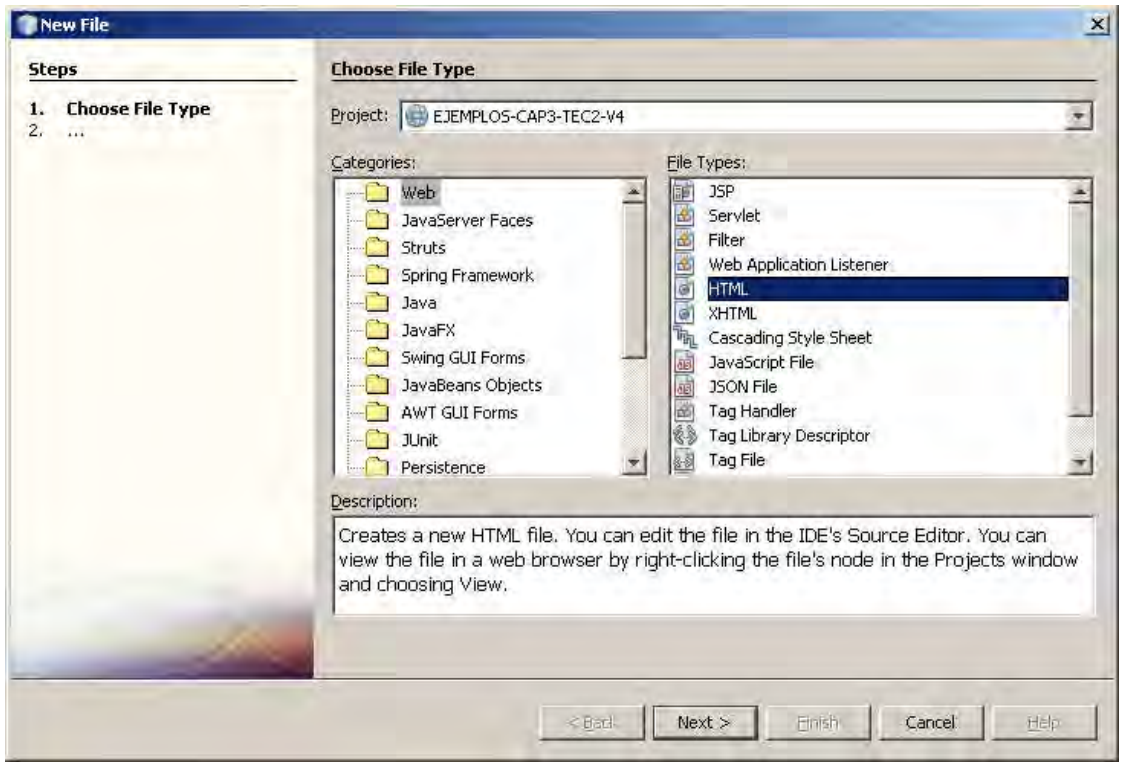
Desarrollar una aplicación Web en java básica que incorpore un applet.

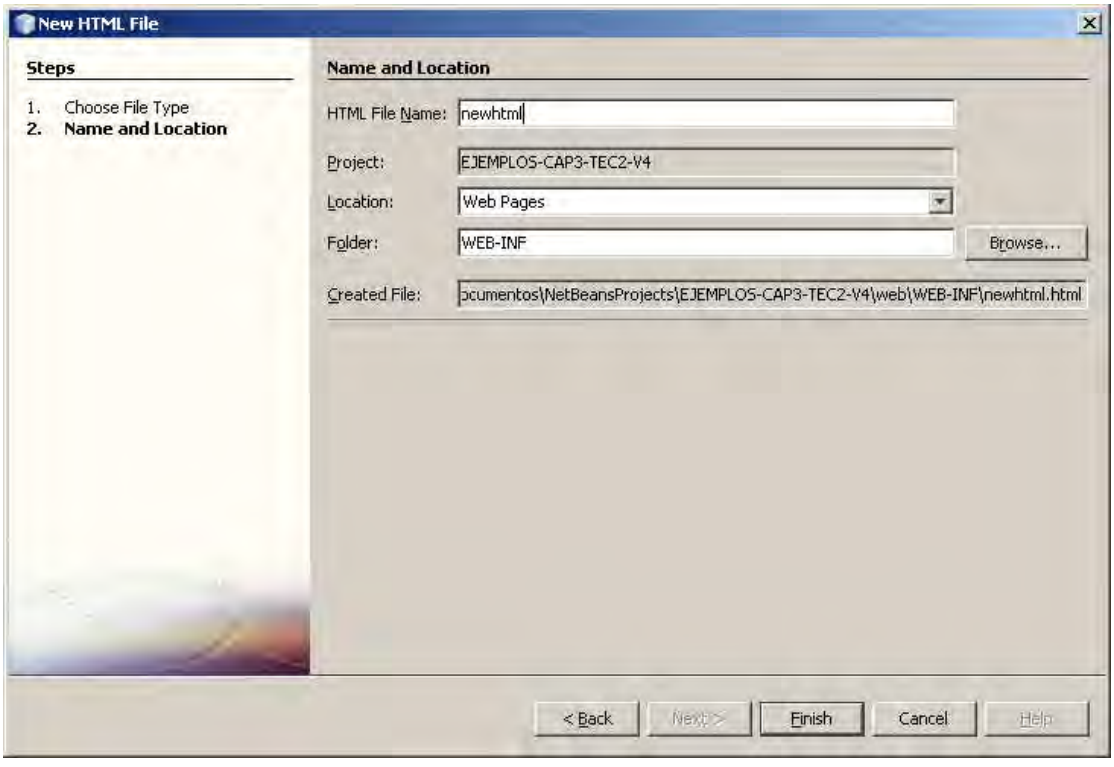

New Project
Steps
1. Choose Project
2. ...
Choose Project
Categories:
Java
JavaFX
Java Web
Java EE
Java ME
Maven
PHP
Ruby
Groovy
C/C++
UML
NetBeans Modules
Projects:
Web Application
Web Application with Existing Sources
Web Free-Form Application
Description:
Creates an empty Web application in a standard IDE project. A standard project uses an IDE-generated build script to build, run, and debug your project.
< Back
Next >
Finish
Cancel
Help

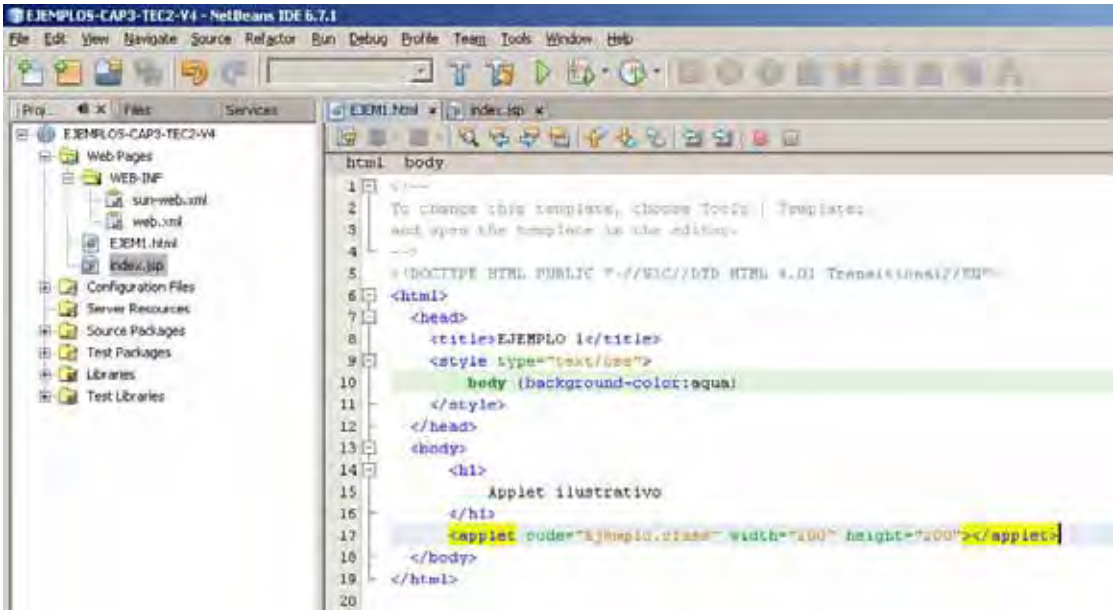

New Web Application
Steps
1. Choose Project
2. Name and Location
3. Server and Settings
4. Frameworks
Name and Location
Project Name: WebApplication3
Project Location: Documents and Settings\Darwin\Mis documentos\NetBeansProjects
Browse...
Project Folder: Settings\Darwin\Mis documentos\NetBeansProjects\WebApplication3
Use Dedicated Folder for Storing Libraries
Libraries Folder:
Browse...
Different users and projects can share the same compilation libraries (see Help for details).
Set as Main Project
< Back
Next >
Finish
Cancel
Help

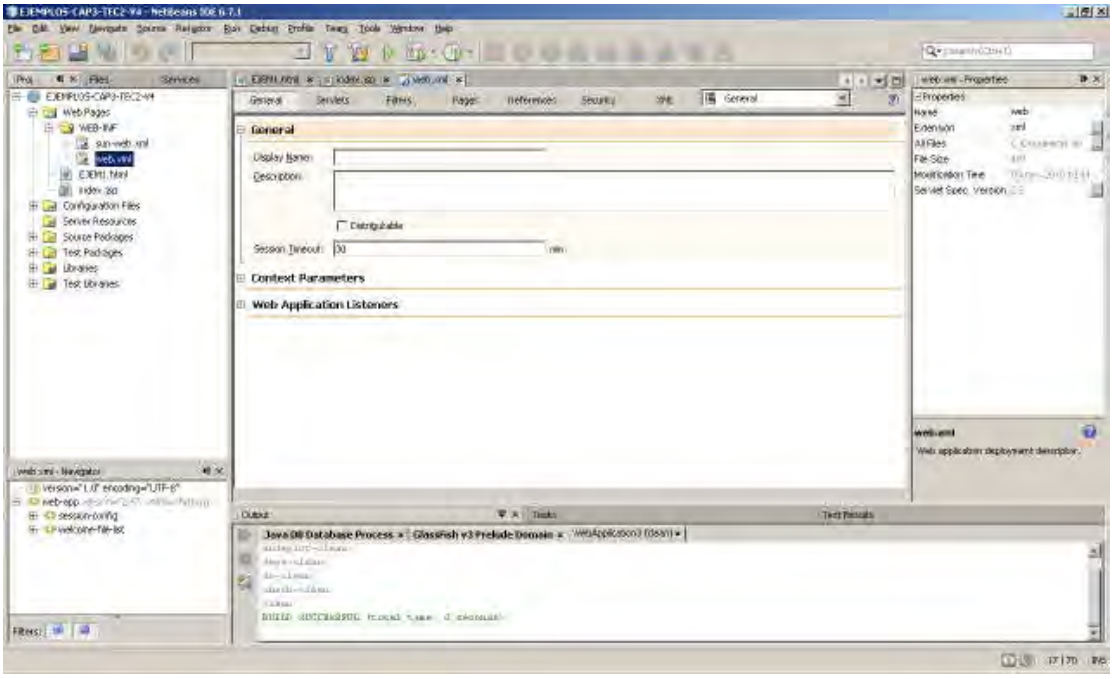

New Web Application
Steps
1. Choose Project
2. Name and Location
3. Server and Settings
4. Frameworks
Server and Settings
Add to Enterprise Application:
<None>
Server:
GlassFish v3 Prelude Domain
Add...
Use dedicated library folder for server JAR files
Java EE Version:
Java EE 5
Context Path:
/WebApplication3
< Back
Next >
Finish
Cancel
Help

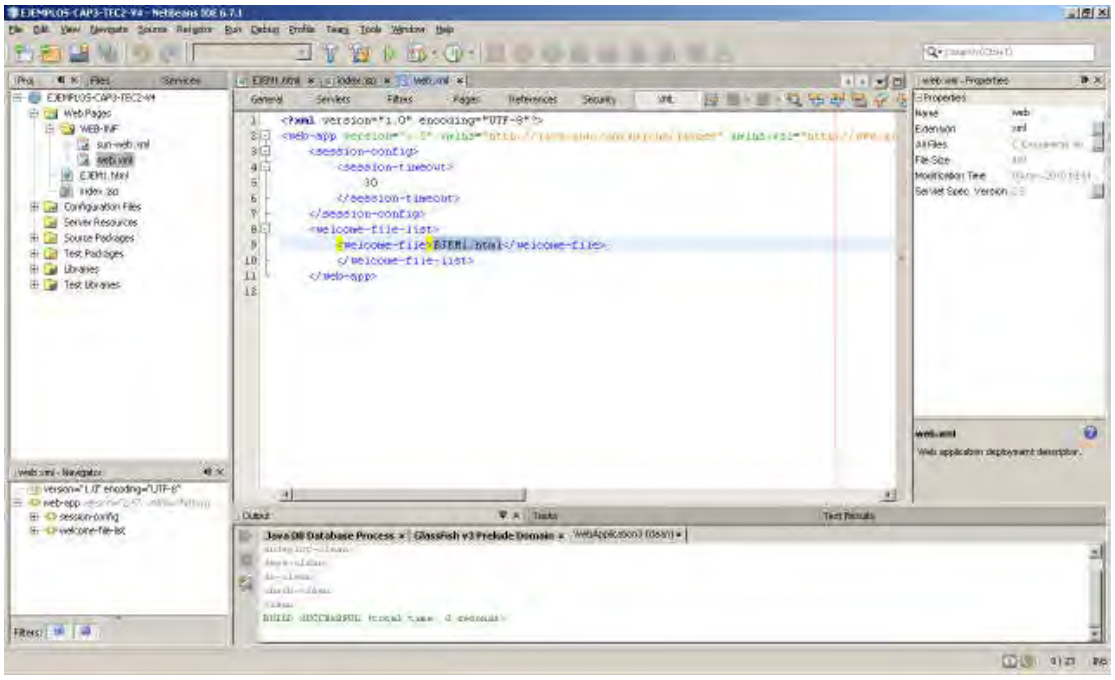



New File
Steps
1. Choose File Type
2. ...
Choose File Type
Project: EJEMPLOS-CAP3-TEC2-V4
Categories:
Web
JavaServer Faces
Struts
Spring Framework
Java
JavaFX
Swing GUI Forms
JavaBeans Objects
AWT GUI Forms
JUnit
Persistence
File Types:
JSP
Servlet
Filter
Web Application Listener
HTML
XHTML
Cascading Style Sheet
JavaScript File
JSON File
Tag Handler
Tag Library Descriptor
Tag File
Description:
Creates a new HTML file. You can edit the file in the IDE's Source Editor. You can view the file in a web browser by right-clicking the file's node in the Projects window and choosing View.
< Back
Next >
Finish
Cancel
Help


New HTML File
Steps
1. Choose File Type
2. Name and Location
Name and Location
HTML File Name: newhtml
Project: EJEMPLOS-CAP3-TEC2-V4
Location: Web Pages
Folder: WEB-INF
Browse...
Created File: ocumentos\NetBeansProjects\EJEMPLOS-CAP3-TEC2-V4\web\WEB-INF\newhtml.html
< Back
Next >
Finish
Cancel
Help


EJEMPLOS-CAP3-TEC2-V4 - NetBeans IDE 6.7.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
EJEMPLOS-CAP3-TEC2-V4
Web Pages
WEB-INF
sun-web.xml
web.xml
EJEM1.html
index.jsp
Configuration Files
Server Resources
Source Packages
Test Packages
Libraries
Test Libraries
<html>
<head>
<title>EJEMPLO 1</title>
<style type="text/css">
body {background-color:aqua}
</style>
</head>
<body>
<h1>
Applet ilustrativo
</h1>
</body>
</html>


General
Display Name:
Description:
Session Timeout:
Context Parameters
Web Application Listeners



Ejemplo No. 2:

HTML no reconoce los finales de línea. Por esa razón, aunque utilicemos distintas líneas en nuestro fichero, serán visualizadas de forma continua. Existen dos formas dos comandos básicos para saltar de línea.

CONSIDERACION:

El primero produce un salto de línea,
 pasando el texto a la línea siguiente.

El segundo, define un final de párrafo <P> dejando una línea en blanco de separación con el texto siguiente.

```
<html>
    <head>
        <title>EJEMPLO 1</title>
        <style type="text/css">
            body {background-color:gray}
        </style>
    </head>
    <body>
        <h1>
            Applet ilustrativo
        </h1>
        <applet code="Ejemplo.class" width="200" height="200"></applet>
        <BR>
        El primero produce un salto de linea, <BR>
        pasando el texto a la linea siguiente.<BR>
        El segundo, define un final de parrafo <P>
            dejando una linea en blanco de separacion <P>
            con el texto siguiente. <P>
    </body>
</html>
```

Ejemplo No. 3:

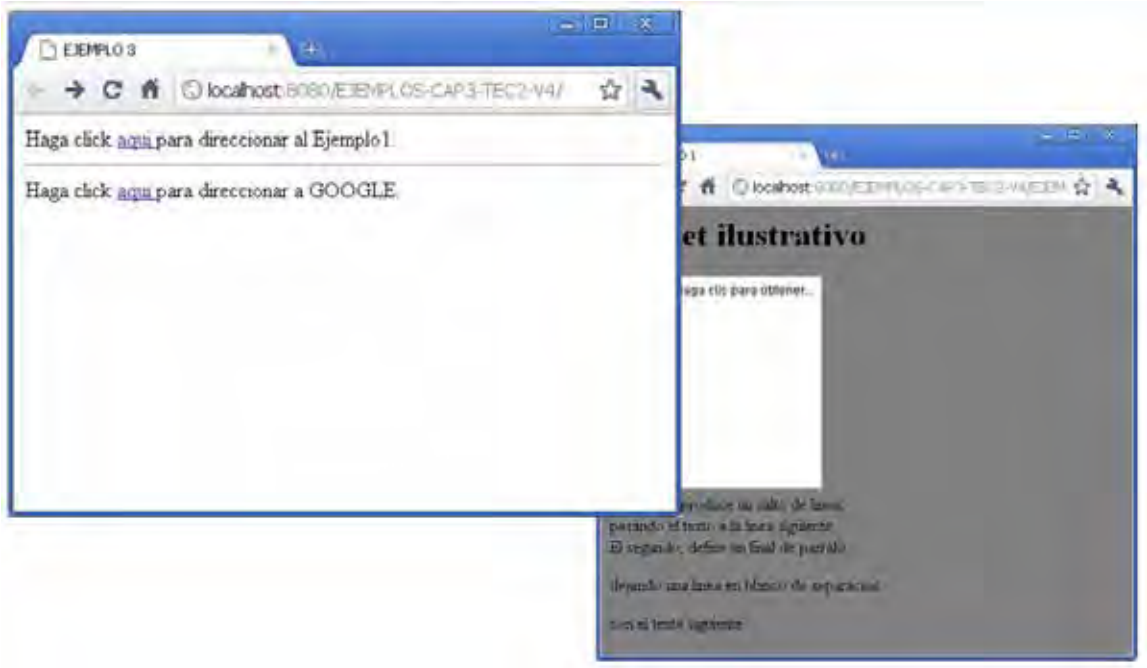
Las páginas HTML puden tener cabeceras, que se insertan con la etiqueta ***hn***, donde n es un valor de 1 a 6. El tamaño del texto es mayor cuanto mayor es el nivel, el nivel más alto es el 1. Para insertar un separador se emplea la etiqueta ***hr.***

```
<html>
    <head>
        <title>EJEMPLO 1</title>
        <style type="text/css">
            body {background-color:silver}
        </style>
    </head>
    <BODY>
        <h1>Ejemplo de un documento HTML</h1>
        <hr>
        <h3> El cuerpo del documento puede contener:</h3>
        <h4> Texto, imagenes, sonido y Ordenes HTML </h4>
        <p> HTML es un lenguaje utilizado para desarrollar pAginas Web.<p>
</html>
```

Ejemplo No. 4:

Para definir un enlace se utiliza la etiqueta "**a**". Esta etiqueta delimita el texto que se quiere utilizar para enlazar con otra. Para indicar al explorador la página que tiene que recuperar cuando el usuario haga click en un enlace, se incluye **href** y a continuación la URL de la página.

```
<html>
    <head>
        <title>EJEMPLO 3</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <body>
        Haga click <a href = "EJEM1.html"> aqui </a>
        para direccionar al Ejemplo1. <hr>
        Haga click <a href = "http://www.google.com/"> aqui </a>
        para direccionar a GOOGLE.
    </body>
</html>
```



Ejemplo No. 5:

Existe una amplia variedad de controles de entrada de datos. Para crearlos, se utiliza la etiqueta **input** y los atributos **type** y **name**.

<input
type="text“, “password“, “checkbox“, “radio“, “submit“
name=“Variable donde se guarda el valor“>

Una caja de texto es un control de tipo **text.**

<input type=“text” name=“nombre” size=“35”>

El tamaño de la caja es 35 y el valor se almacena en la variable nombre.

Se puede utilizar **value** para especificar un valor inicial.

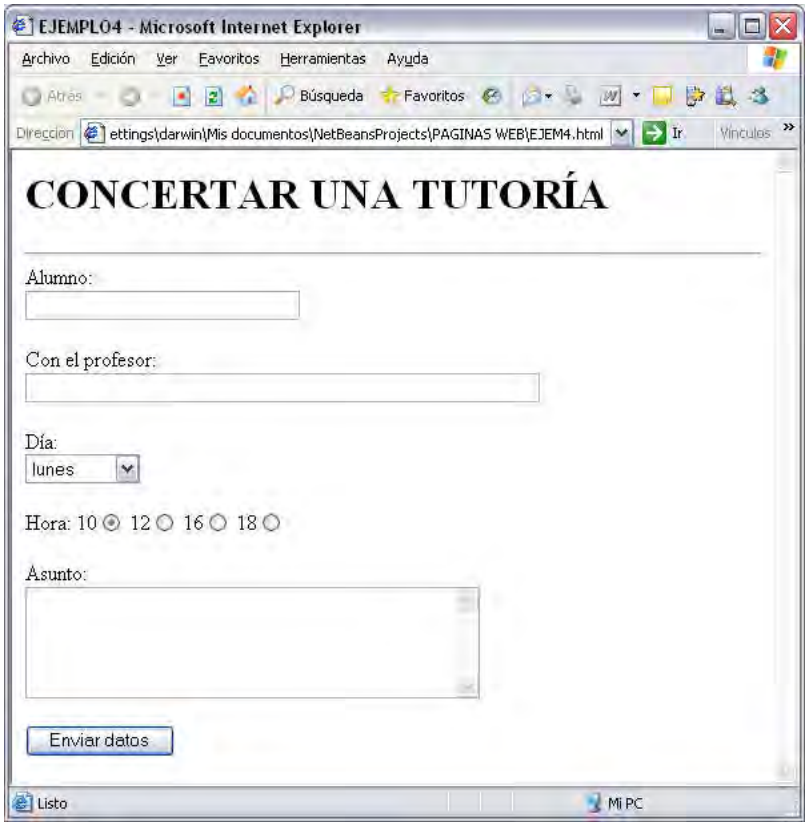
Checkbox es un botón que presenta dos estados

<input type=“checkbox” name=“cv1” value=“1” checked> Opción 1
<input type=“checkbox” name=“cv1” value=“2”> Opción 1

Checked permite iniciar el estado de la casilla como seleccionado.

El control tipo radio similar al checkbox, con la diferencia que solo uno puede estar seleccionado a la vez.

<input type=“radio” name=“opcion” value=“1” checked> Opción 1
<input type=“radio” name=“opcion” value=“2”> Opción 1


EJEMPLO4 - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Favoritos
Dirección ettings\darwin\Mis documentos\NetBeansProjects\PAGINAS WEB\EJEM4.html
Ir
Vínculos
CONCERTAR UNA TUTORÍA
Alumno:
Con el profesor:
Día:
lunes
Hora: 10 12 16 18
Asunto:
Enviar datos
Listo
Mi PC

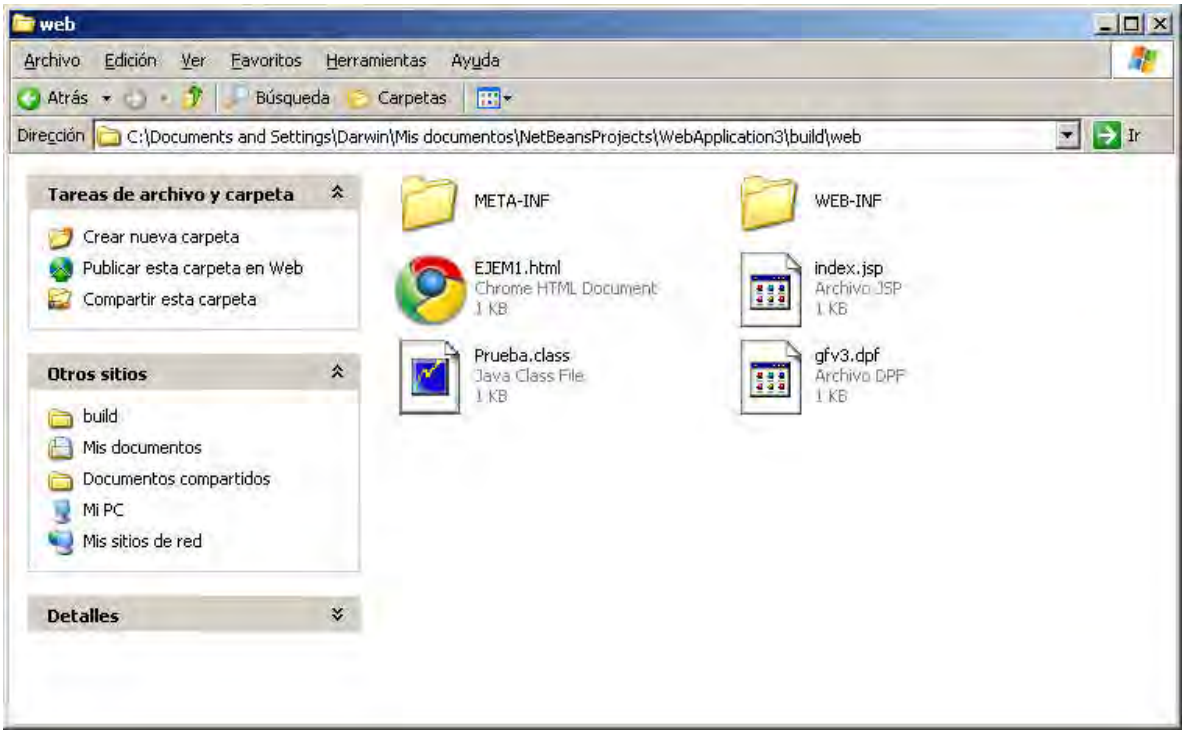
```
<html>
  <head>
    <title>EJEMPLO4</title>
  </head>
  <body>
    <h1>CONCERTAR UNA TUTORÍA</h1>
    <HR>
      Alumno:<br>
      <input type="text" name="alumno" size="60">
      <br><br>
      Con el profesor:<br>
      <input type="text" name="profesor" size="60">
      <br><br>
      Día:<br>
      <select name="día">
        <option>lunes <option>miércoles <option>jueves
      </select>
      <br><br>
      Hora:
      10<input type="radio" name="hora" value="10" checked>
      12<input type="radio" name="hora" value="12">
      16<input type="radio" name="hora" value="16">
      18<input type="radio" name="hora" value="18">
      <br><br>
      Asunto:<br>
      <textarea name="asunto" rows="5" cols="40" ></textarea>
      <br><br>
      <input type="submit" value="Enviar datos">
  </body>
</html>
```

Ejemplo No. 6:

Crear una página web en la que se permita desplegar un applet básico, en la diga ELECTRONICA.

```
import java.applet.Applet;
import java.awt.*;

/**
 *
 * @author Darwin
 */
public class Prueba extends Applet {

    /**
     * Initialization method that will be called after the applet is loaded
     * into the browser.
     */
    public void init() {
        // TODO start asynchronous download of heavy resources
    }
    public void paint(Graphics g) {
        g.drawString("ELECTRONICA",50,50);
    }
    // TODO overwrite start(), stop() and destroy() methods

}
```

```
<html>
    <head>
        <title>EJEMPLO 1</title>
        <style type="text/css">
            body {background-color:gray}
        </style>
    </head>
    <body>
        <h1>
            Applet ilustrativo
        </h1>
        <applet code="Prueba.class" width="200" height="200"></applet>

    </body>
</html>
```


web
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Carpetas
Dirección C:\Documents and Settings\Darwin\Mis documentos\NetBeansProjects\WebApplication3\build\web
Ir
Tareas de archivo y carpeta
Crear nueva carpeta
Publicar esta carpeta en Web
Compartir esta carpeta
Otros sitios
build
Mis documentos
Documentos compartidos
Mi PC
Mis sitios de red
Detalles
META-INF
WEB-INF
EJEM1.html
Chrome HTML Document
1 KB
index.jsp
Archivo JSP
1 KB
Prueba.class
Java Class File
1 KB
gfv3.dpf
Archivo DPF
1 KB


EJEMPLO 1
localhost:8080/WebApplication3/
Applet ilustrativo
ELECTRONICA

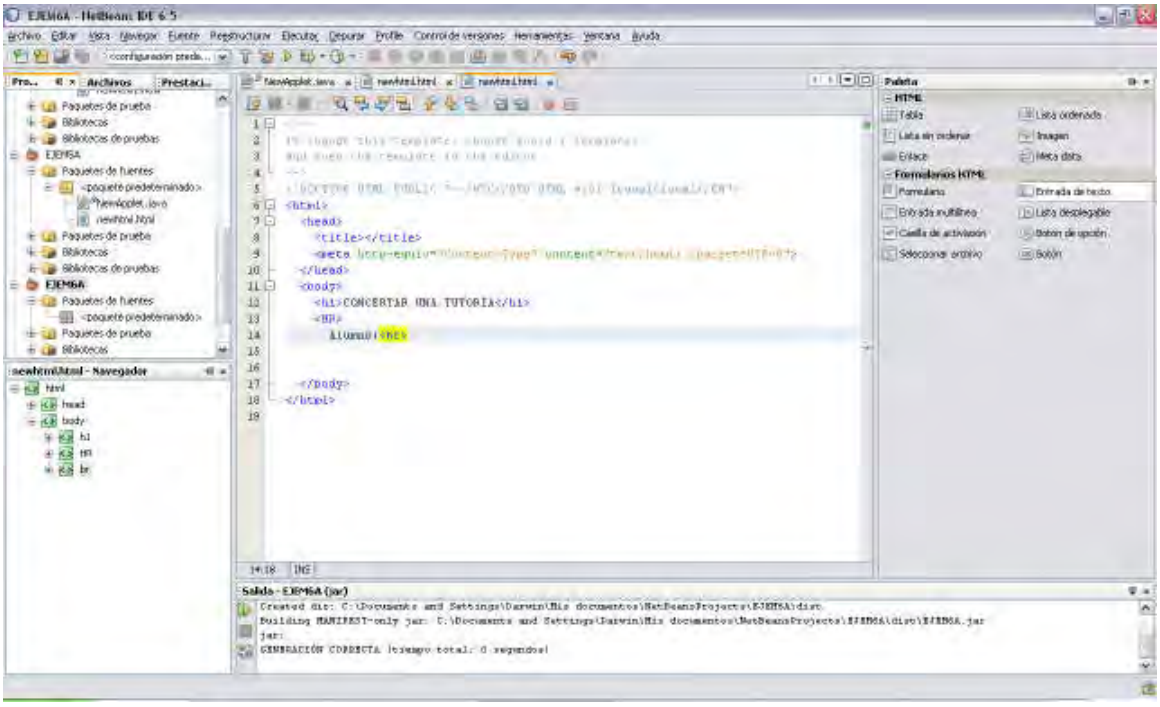
Ejemplo No. 7:

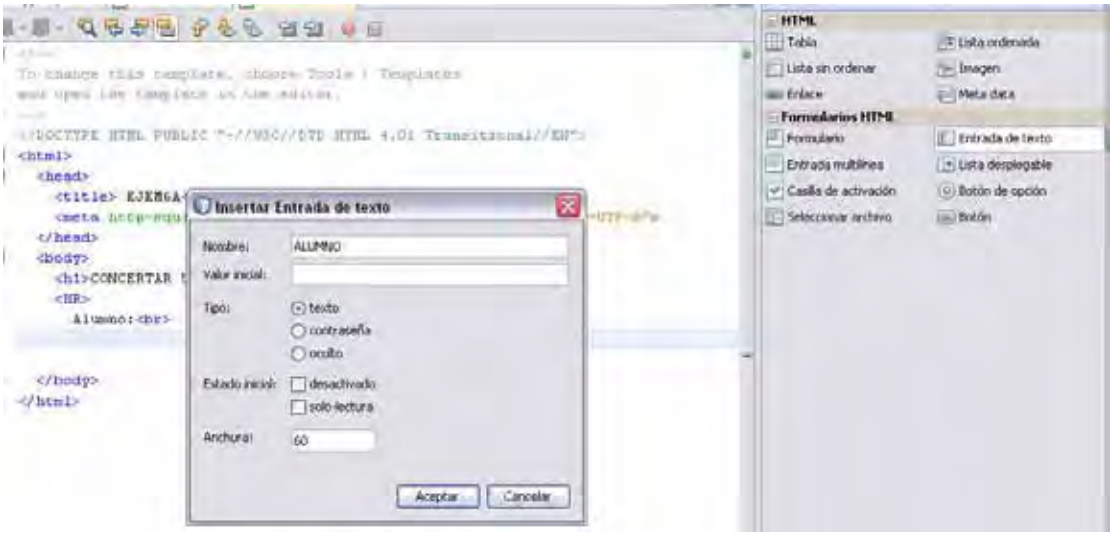
Incorporación de colores y estilos de letra en una página web

```
<HTML>
  <HEAD>
   <TITLE>
     ESPE
   </TITLE>
  </HEAD>
  <BODY>
   <H1>TECNOLOGIAS DE SOFTWARE II</H1>
   <HR>
   <BR>
   <BR>
   <BR>
   <H6><B><I><U>HOLA AMIGOS</U></I></B></H6>
   <FONT SIZE=3 COLOR="GREEN" FACE="Arial">
   FUENTE DE LETRA
   </FONT>
   <UL> <LI>Naranja <LI>Pera <LI>Durazno </UL>
   <P>
   Para saltar a otra página fuera de la máquina
   <A HREF="http://www.microsoft.com"> hacer clic aquí</A>
   , siempre y cuando esté
   conectado a internet.
   </P>
  </BODY>
</HTML>
```

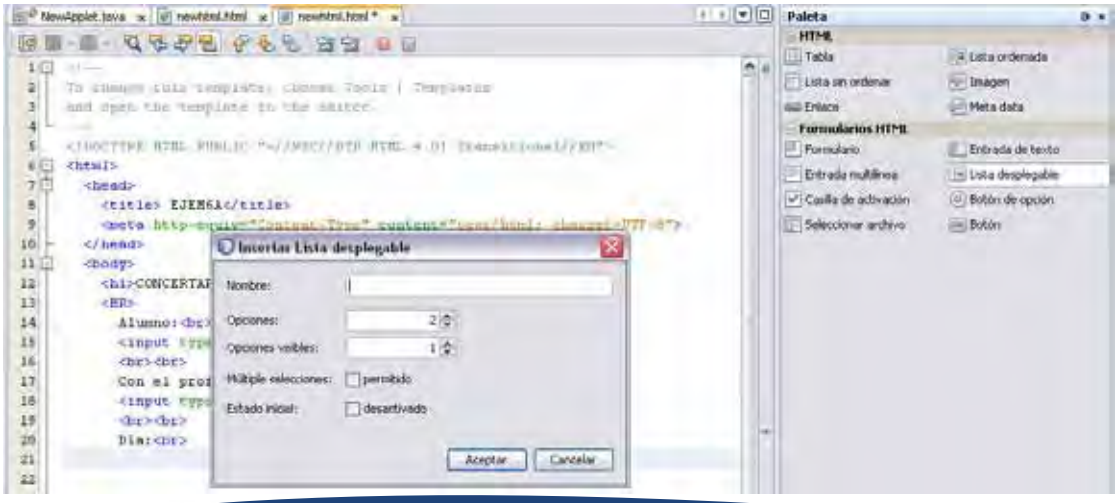
Ejemplo No. 8:

Vuelva a desarrollar el Ejemplo 5 pero utilizando Netbeans.

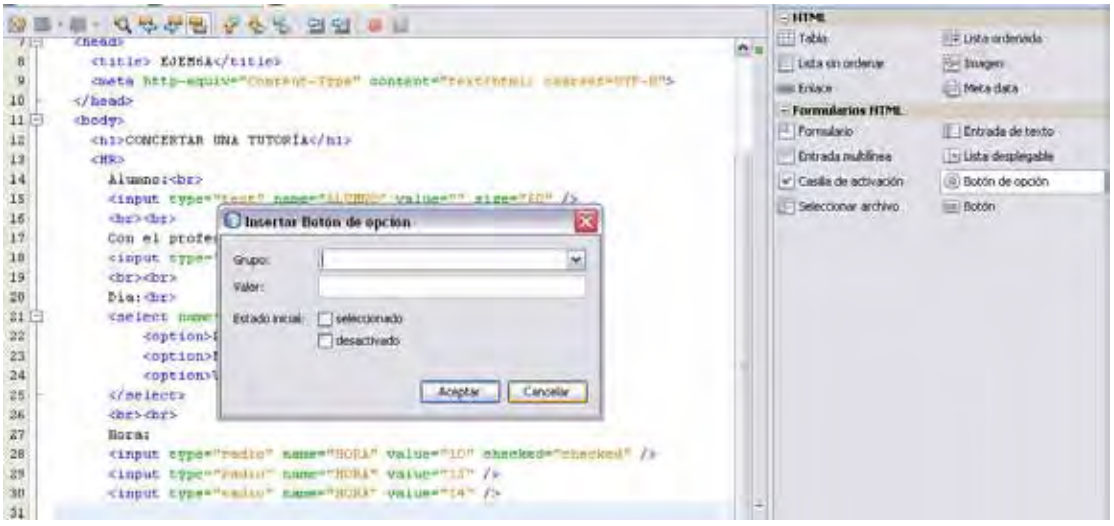




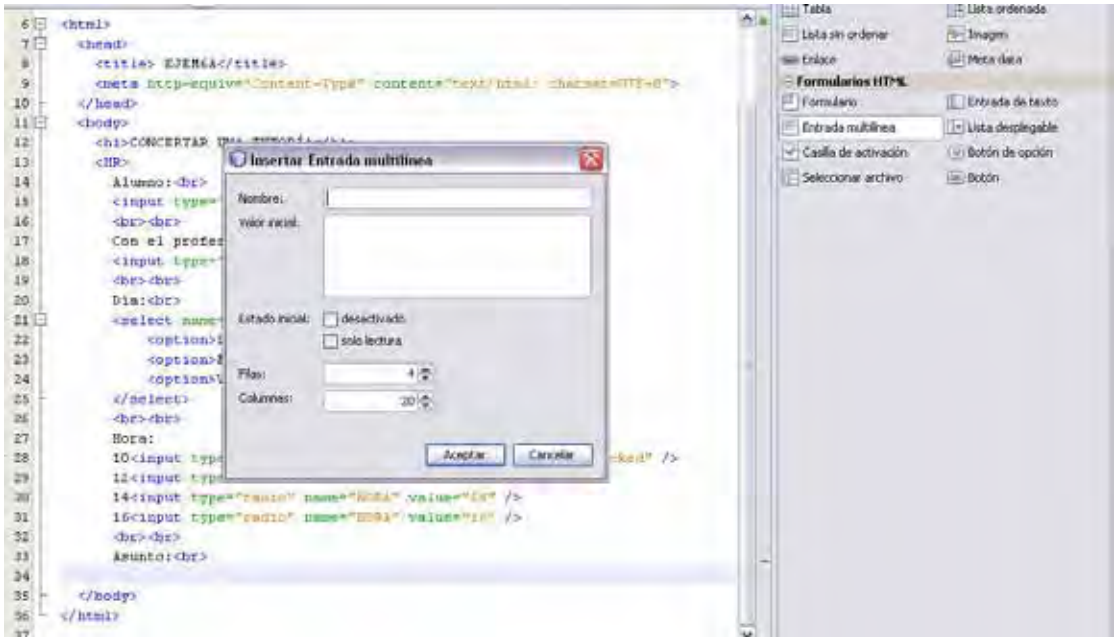

```
<html>
  <head>
    <title> EJEM6A</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <h1>CONCERTAR UNA TUTORÍA</h1>
    <HR>
      Alumno:<br>
      <input type="text" name="ALUMNO" value="" size="60" />
      <br><br>
      Con el profesor:<br>
      <input type="text" name="PROFESOR" value="" size="60" />

  </body>
</html>
```

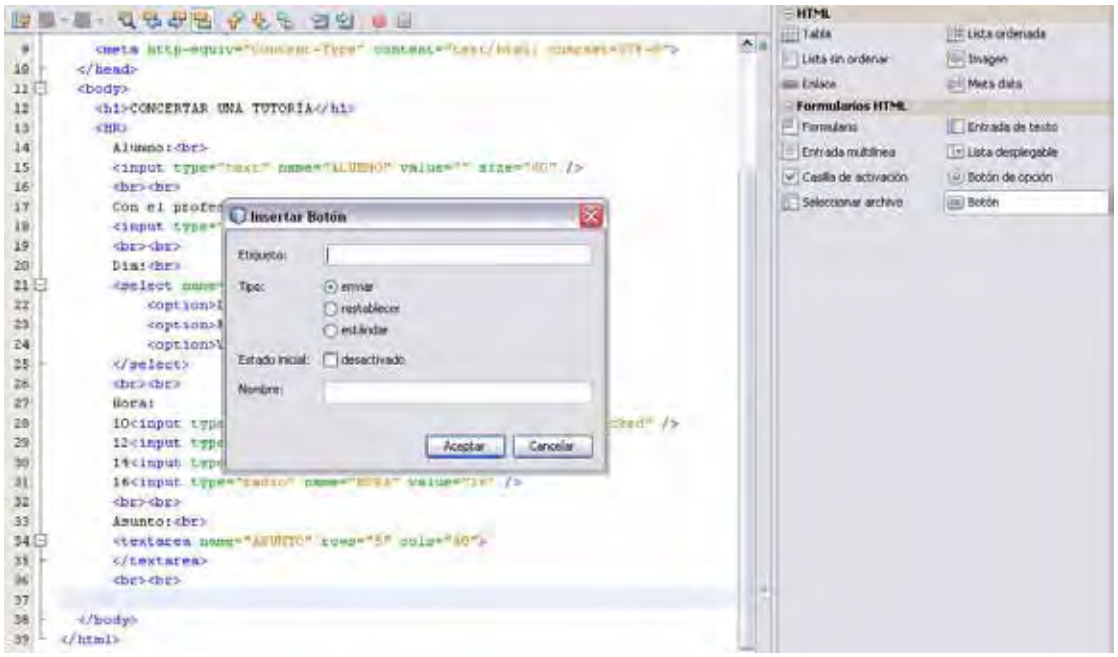


```
<html>
  <head>
    <title> EJEM6A</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <h1>CONCERTAR UNA TUTORÍA</h1>
    <HR>
      Alumno:<br>
      <input type="text" name="ALUMNO" value="" size="60" />
      <br><br>
      Con el profesor:<br>
      <input type="text" name="PROFESOR" value="" size="60" />
      <br><br>
      Día:<br>
      <select name="DIA">
          <option>LUNES</option>
          <option>MIERCOLES</option>
          <option>VIERNES</option>
      </select>

  </body>
</html>
```



```
<html>
  <head>
    <title> EJEM6A</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <h1>CONCERTAR UNA TUTORÍA</h1>
    <HR>
      Alumno:<br>
      <input type="text" name="ALUMNO" value="" size="60" />
      <br><br>
      Con el profesor:<br>
      <input type="text" name="PROFESOR" value="" size="60" />
      <br><br>
      Día:<br>
      <select name="DIA">
          <option>LUNES</option>
          <option>MIERCOLES</option>
          <option>VIERNES</option>
      </select>
      <br><br>
      Hora:
      10<input type="radio" name="HORA" value="10" checked="checked" />
      12<input type="radio" name="HORA" value="12" />
      14<input type="radio" name="HORA" value="14" />
      16<input type="radio" name="HORA" value="16" />
      <br><br>
      Asunto:<br>

  </body>
</html>
```

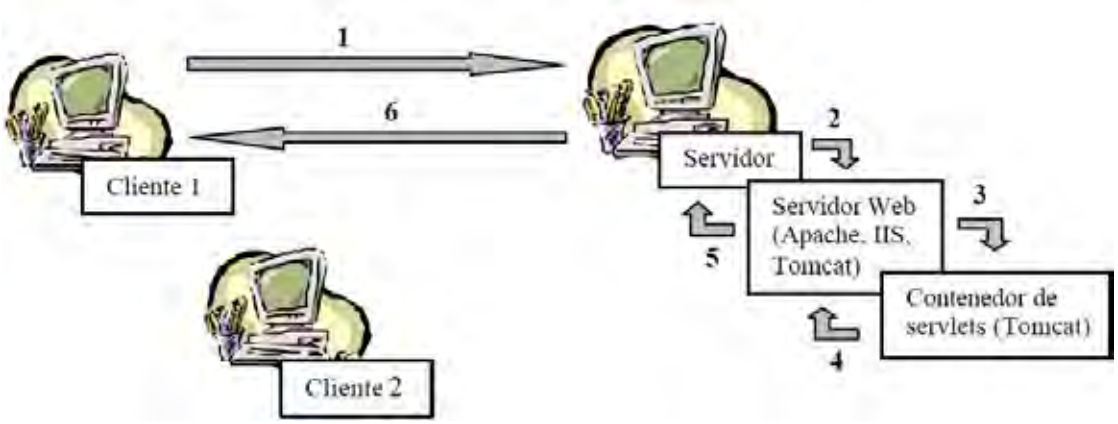



Insertar Botón
Etiqueta:
Tipo:
enviar
restablecer
enlbridar
Estado inicial:
desactivado
Nombre:
Aceptar
Cancelar
HTML
Tabla
Lista ordenada
Lista sin ordenar
Imagen
Enlace
Meta data
Formularios HTML
Formulario
Entrada de texto
Entrada multilínea
Lista desplegable
Casilla de activación
Botón de opción
Seleccionar archivo
Botón

```
<html>
  <head>
    <title> EJEM6A</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <h1>CONCERTAR UNA TUTORÍA</h1>
    <HR>
      Alumno:<br>
      <input type="text" name="ALUMNO" value="" size="60" />
      <br><br>
      Con el profesor:<br>
      <input type="text" name="PROFESOR" value="" size="60" />
      <br><br>
      Día:<br>
      <select name="DIA">
          <option>LUNES</option>
          <option>MIERCOLES</option>
          <option>VIERNES</option>
      </select>
      <br><br>
      Hora:
      10<input type="radio" name="HORA" value="10" checked="checked" />
      12<input type="radio" name="HORA" value="12" />
      14<input type="radio" name="HORA" value="14" />
      16<input type="radio" name="HORA" value="16" />
      <br><br>
      Asunto:<br>
      <textarea name="ASUNTO" rows="5" cols="40">
      </textarea>
      <br><br>
      <input type="submit" value="ENVIAR DATOS" />
  </body>
</html>
```

SERVLETS y JSP

Los servlets de Java proporcionan un mecanismo para ejecutar programas en servidores, en función de las peticiones que los clientes realicen haciendo uso de los navegadores web. Los servlets permiten crear páginas web activas, en el sentido de que la respuesta que ofrecen puede variar en función de los datos que proporcione el cliente.



1. El cliente realiza una petición http (puerto 80) haciendo uso de un navegador. En esta petición pueden existir parámetros con sus valores.
2. La petición y sus datos asociados, que le llegan al equipo servidor a través de la red, los recoge el programa servidor de web utilizado.
3. El servidor web detecta que hay que ejecutar un servlet y delega esta acción en el programa CONTENEDOR DE SERVLETS, que se encargará de llevarla a cabo. El Contenedor de Servlets también puede hacer la función de servidor web.
4. Una vez ejecutado el servlet, se traspasan los resultados al servidor web.
5. El servidor web envía la página web de resultados a través del servidor.
6. La página web de respuesta le llega al cliente, que la visualiza haciendo uso del navegador.

La tecnología JSP, es un "*Servlet maquillado*". En realidad es una extensión de la tecnología de los Servlets. JSP permite a los desasrrolladores de sitios web crear páginas que utilicen la funcionalidad de Java en sus páginas. Generalmente, JSP se utiliza cuando la mayoría del contenido que se envía al cliente es estático, y solo una pequeña porción del código se genera dinámicamente. Los Servlets se utilizan cuando una pequeña porción del cóntenido que se envía al cliente es estática.

Los paquetes de J2EE que intervienen directamente en la creación de servlets son:

- *javax.servlet*
- *javax.servlet.http*

Contienen las interfaces y clases más genéricas que soportan el funcionamiento de los servlets.

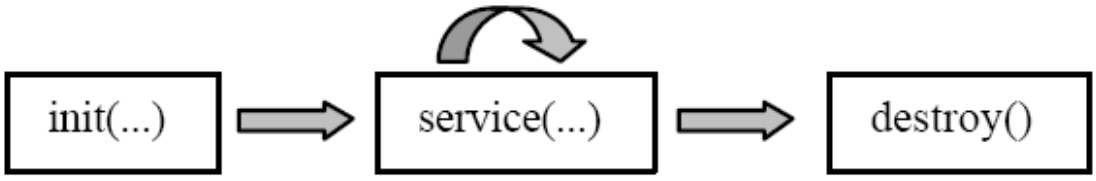
Los servlets de Java tienen un ciclo de vida marcado por tres métodos de la interfaz *javax.servlet.Servlet:*

- *init(...)*
- *service(…)*
- *desrroy(…)*

El método ***init*** lo invoca el contenedor de servlets, una sola vez; resulta útil para inicializar recursos que serán necesarios en la ejecución del servlet (abrir ficheros, conectar bases de datos, establecer comunicaciones, etc.).

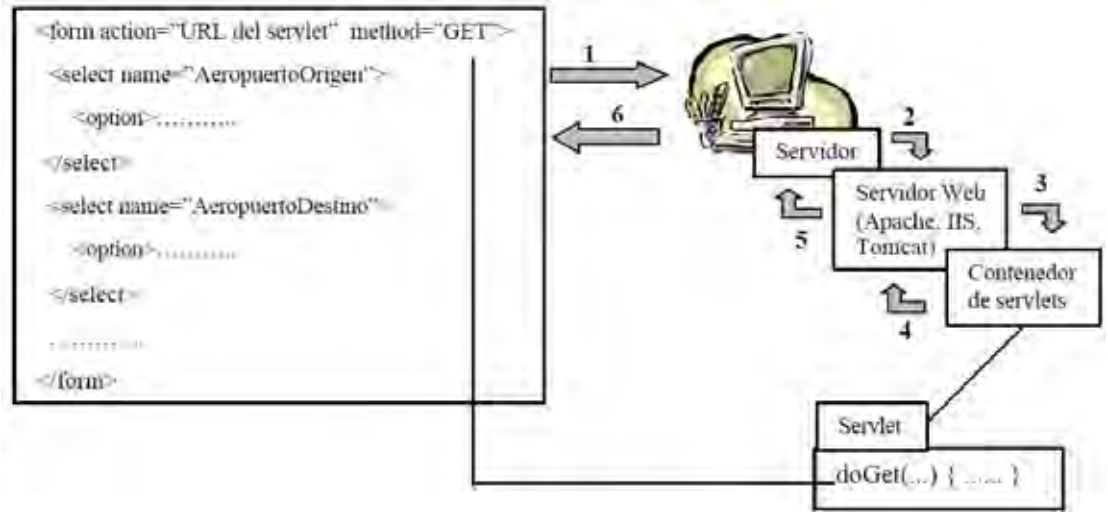
El método ***destroy*** se invoca cuando el servlet va a ser descargado del servidor; en su interior se debería programar la liberación de recursos utilizados en el método ***init***.

El método ***service*** realiza el trabajo "cotidiano" del servlet: dar respuesta a las distintas peticiones de los clientes. Cada vez que un cliente realiza una petición (GET, POST) el servidor ejecuta el método ***service***.



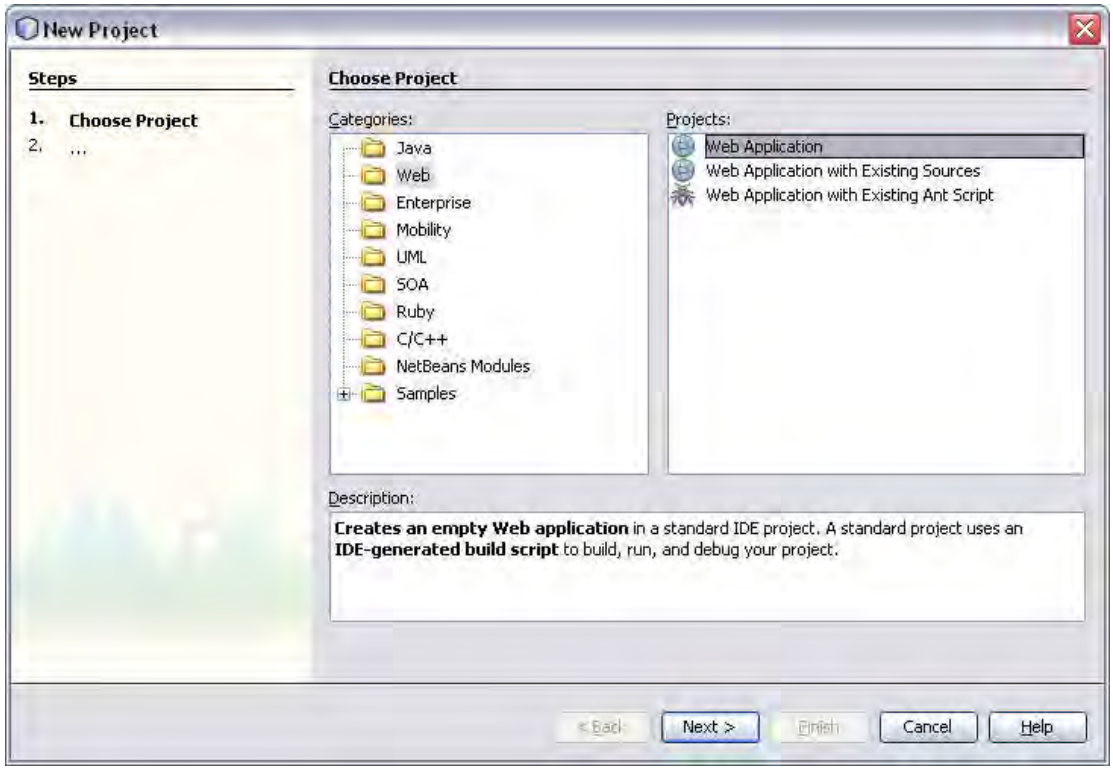
HttpServlet conserva el método *service(. ..)*, pero además proporciona los métodos *doGet, doPost,* y los de menor utilización *doPut, doDelete, doTrace, doOptions y doHead.* Cuando al servlet le llega una petición de tipo *GET,* se ejecuta automáticamente el método *doGet(...),* lo mismo ocurre con las peticiones *POST, PUT, DELETE, TRACE, OPTIONS* y *HEAD.*

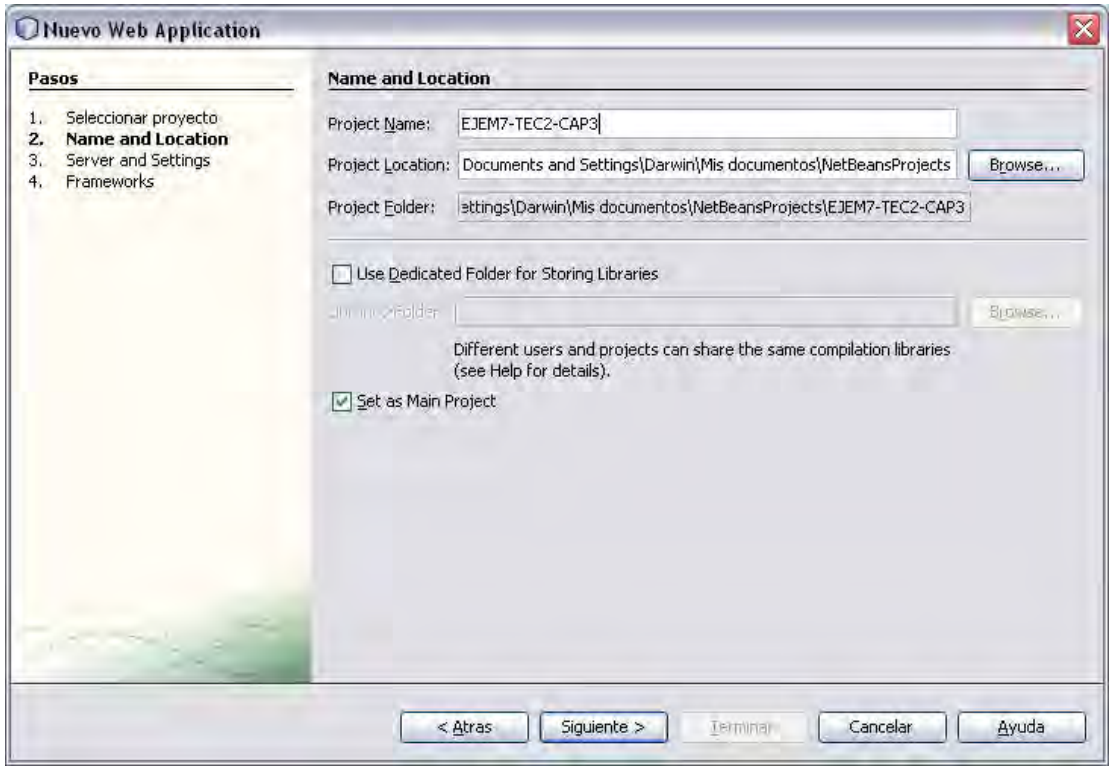
El método service tiene como parámetros un objetos de tipo ServletRequest y ServletResponse. ServletRequest, proporciona la información que llega del servidor web. ServletResponse, envía al cliente la información que genera el servlet.

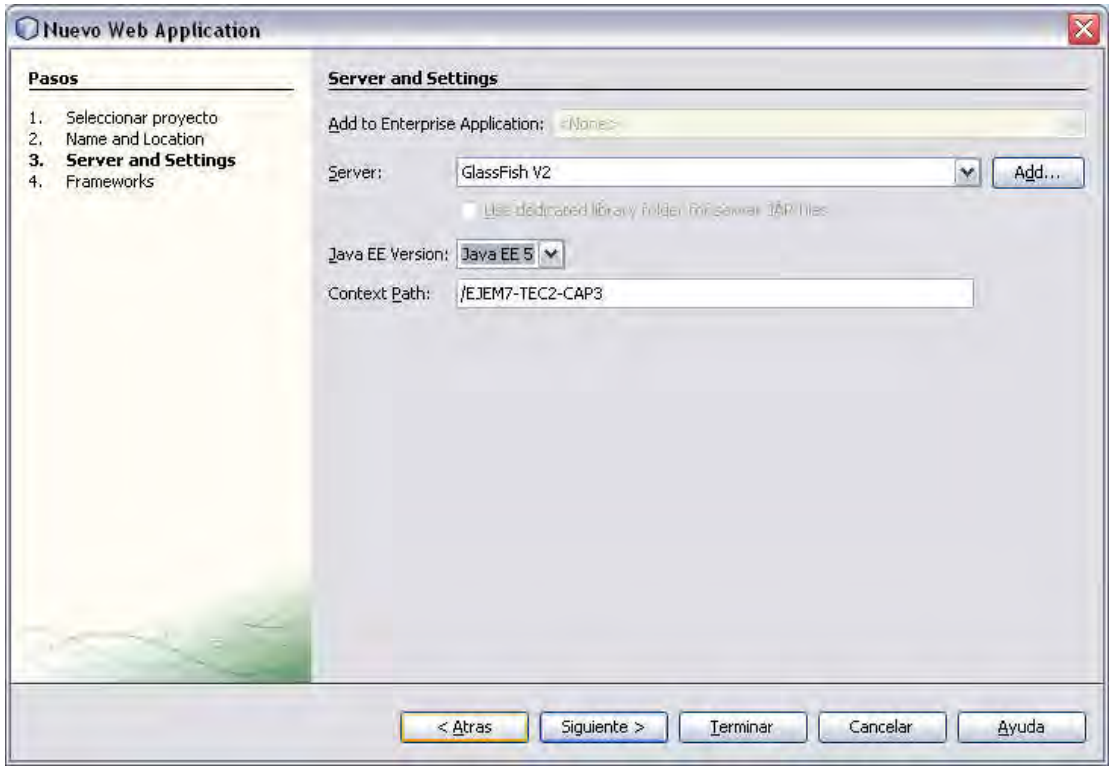


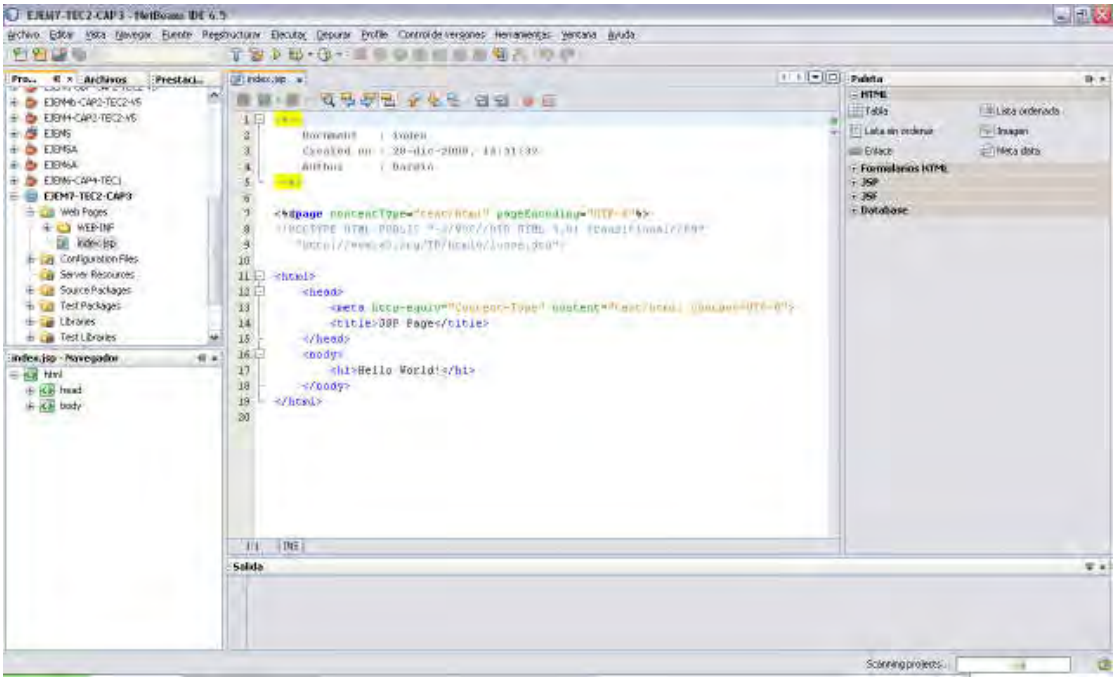
Ejemplo No. 9:

Crear una aplicación web que solicite al usuario ingresar su nombre, y que presente en otra un saludo.



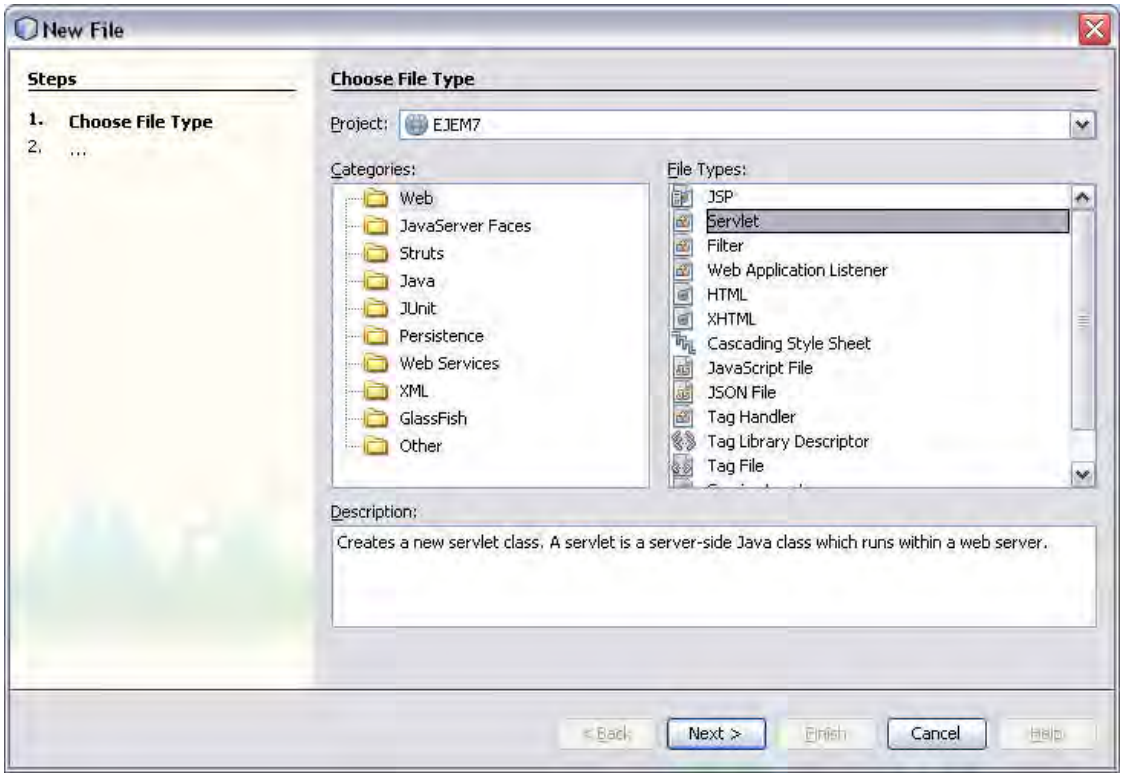

Nuevo Web Application
Pasos
1. Seleccionar proyecto
2. Name and Location
3. Server and Settings
4. Frameworks
Name and Location
Project Name: EJEM7-TEC2-CAP3
Project Location: Documents and Settings\Darwin\Mis documentos\NetBeansProjects
Browse...
Project Folder: ettings\Darwin\Mis documentos\NetBeansProjects\EJEM7-TEC2-CAP3
Use Dedicated Folder for Storing Libraries
Browse...
Different users and projects can share the same compilation libraries
(see Help for details).
Set as Main Project
< Atras
Siguiente >
Terminar
Cancelar
Ayuda

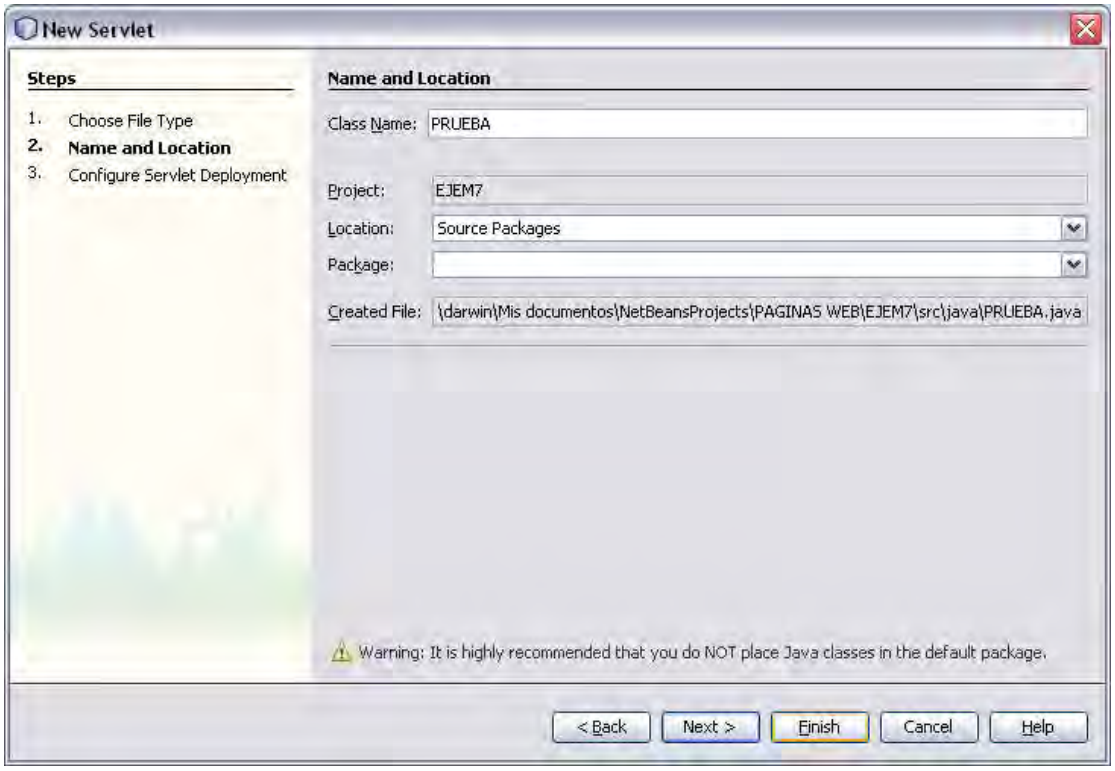

Nuevo Web Application
Pasos
1. Seleccionar proyecto
2. Name and Location
3. Server and Settings
4. Frameworks
Server and Settings
Add to Enterprise Application:
Server: GlassFish V2
Add...
Java EE Version: Java EE 5
Context Path: /EJEM7-TEC2-CAP3
< Atras
Siguiente >
Terminar
Cancelar
Ayuda



```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>EJEMPLO 7</title>
    </head>
    <body>
        <form action="PRUEBA" method="POST">
            INGRESE SU NOMBRE:
            <input type="text" name="nombre" value="" />
            <input type="submit" value="Enviar" name="enviar" />
        </form>
    </body>
</html>
```


EJEMPLO 7 - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas
INGRESE SU NOMBRE:
Enviar
Terminado


New File
Steps
1. Choose File Type
2. ...
Choose File Type
Project: EJEM7
Categories:
Web
JavaServer Faces
Struts
Java
JUnit
Persistence
Web Services
XML
GlassFish
Other
File Types:
JSP
Servlet
Filter
Web Application Listener
HTML
XHTML
Cascading Style Sheet
JavaScript File
JSON File
Tag Handler
Tag Library Descriptor
Tag File
Description:
Creates a new servlet class. A servlet is a server-side Java class which runs within a web server.
< Back
Next >
Finish
Cancel
Help



Finalizado esto, automáticamente crea una clase con el nombre de servlet dado, que hereda de **HttpServlet**. Crea un método **processRequest** (invocado desde los métodos doGet y doPost) para procesar los formularios que llegan por los métodos GET y POST.

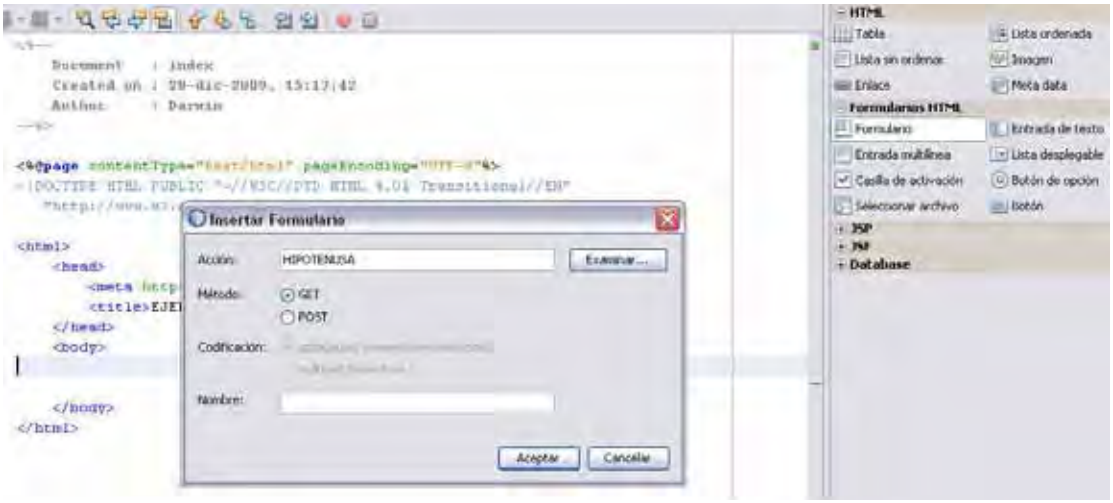
```
public class PRUEBA extends HttpServlet {
    /**...*/
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet PRUEBA</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<p>");
            out.println("HOLA " + request.getParameter("nombre").toString() + "</h1>");
            out.println("</p>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }
    HttpServlet methods. Click on the + sign on the left to edit the code.
}
```

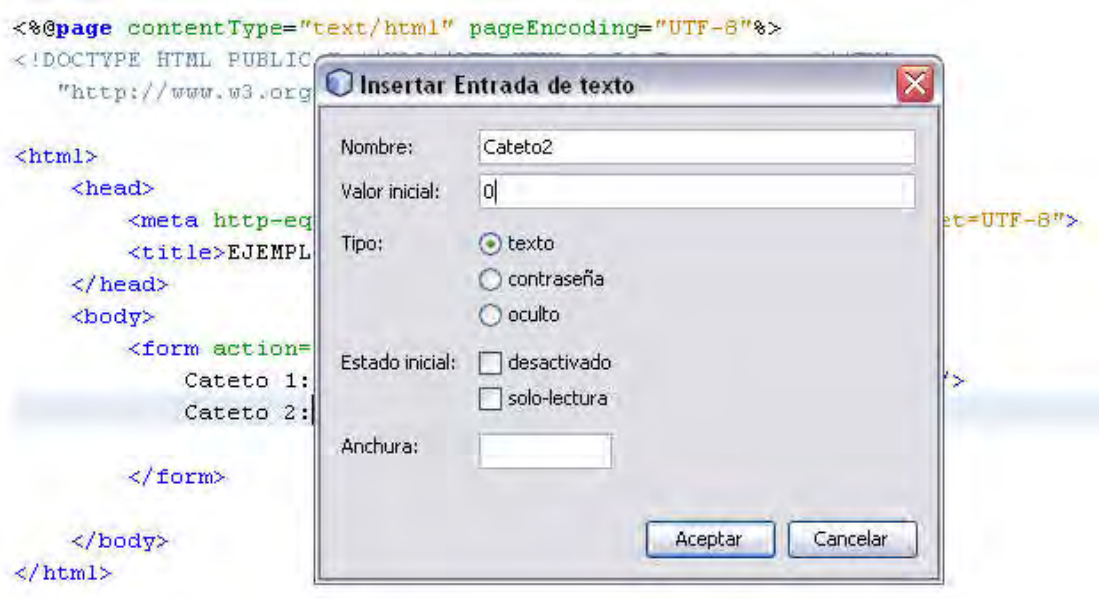
Al ejecutar una aplicación web con NetBeans, lo primero que hace el mismo es un **Deploy**, que consiste en distribuir la aplicación en el servidor. Por más que el servidor sea

local y que NetBeans lo haga transparente, se debe recordar que Glassfish se ejecuta cuando se ejecuta la aplicación y que además posee una estructura de directorios donde almacena las aplicaciones web que corre, archivos de configuración, paquetes de clases, etc.

Ejemplo No. 10:

Crear una aplicación web que permita realizar el cálculo de la hipotenusa de un triángulo definido por sus catetos.






```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>EJEMPLO 8</title>
    </head>
    <body>
        <h1>CALCULO DE LA HIPOTENUSA</h1>
        <form action="HIPOTENUSA">
            Cateto 1:<input type="text" name="Cateto1" value="0" /><br>
            Cateto 2:<input type="text" name="Cateto2" value="0" /><br><br>
            <input type="submit" value="ENVIAR" /><br>
        </form>

    </body>
</html>
```

```
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
    String Cat1= request.getParameter("Cateto1");
    String Cat2= request.getParameter("Cateto2");
    double c1= Double.parseDouble(Cat1);
    double c2= Double.parseDouble(Cat2);
    double resultado=Math.sqrt(Math.pow(c1,2)+Math.pow(c2,2));
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet HIPOTENUSA</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>HIPOTENUSA = " + resultado + "</h1>");
    out.println("</body>");
    out.println("</html>");
} finally {
    out.close();
}
```

Ejemplo No. 11:

A partir del código generado en el Ejemplo 8 desarrolle el servlet necesario para registrar la tutoría. Agregue un botón para borrar la información insertada antes de ser enviada.

```
String alumno = request.getParameter("ALUMNO");
String dias = request.getParameter("DIA");
String horas = request.getParameter("HORA");
String profesor = request.getParameter("PROFESOR");

out.println("<html>");
out.println("<head>");
out.println("<title> TUTORIAS DDE A DISTANCIA </title>");
out.println("</head>");
out.println("<body>");
out.println("<font size=5>");
out.println("Sr. " + alumno);
out.println("<br>");
out.println("Su petición ha sido aceptada y registrada");
out.println("<br>");
out.println("<font size=5 color=blue>");
out.println("El profesor: " +profesor);
out.println("<br>");
out.println("Le espera el día " + dias + " a las " + horas);
out.println("</font>");
out.println("<br>");
out.println("<br>");
out.println("¡NO FALTE!");
out.println("<br>");
out.println("</body>");
out.println("</html>");
```

ACTIVIDADES DE REELABORACION

ACTIVIDADES PRÁCTICAS

1.- Cree una aplicación web que pregunte por la fecha actual a una aplicación servlet. Además pruebe la aplicación ejecutándola en un una segunda máquina que se encuentre en la red.

http://localhost:8080/EJEM9-TEC2-CAP3/

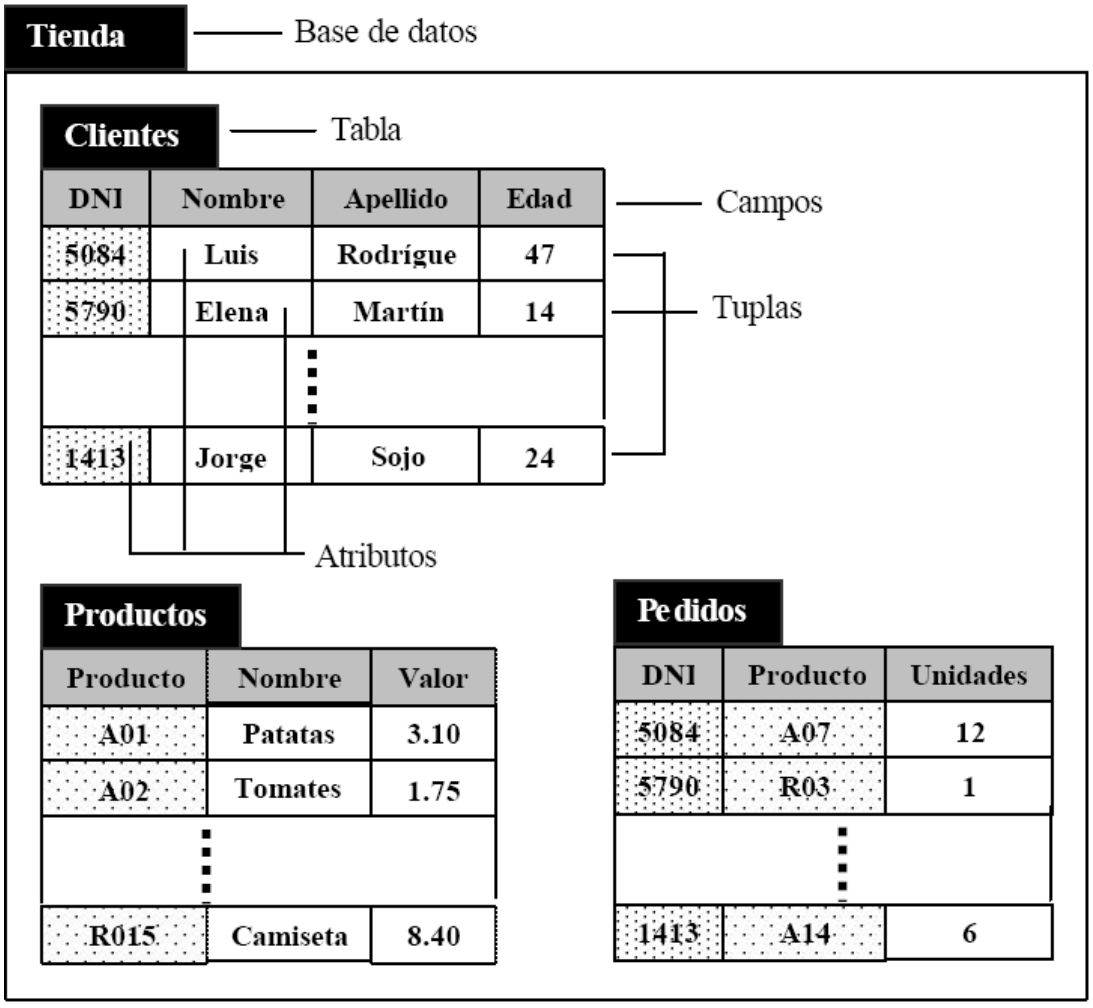
Esta línea insertada en el navegador ejecuta la aplicación en la misma máquina.Si se quiere ejecutar la aplicación desde otra máquina se debe reemplazar localhost por la IP del servidor donde se encuentra el servlet.

CAPITULO 12

BASES DE DATOS

Una base de datos puede contener un conjunto de tablas relacionadas entre sí; cada tabla está definida por una serie de campos y conformada por una lista de tuplas.

Los campos forman las columnas de las tablas; definen el tipo y variedad de sus datos. Las filas se denominan tuplas o registros.Cada valor de un campo definido en una tupla se lo denomina atributo.



Además de los datos de una base de datos, existe una información importante; los metadatos. Los metadatos se encuentran recopilados en un conjunto de tablas del sistema, denominado catálogo. Los catálogos almacenan información acerca de las bases de datos, las tablas y la composición de dichas tablas.

La manipulación de la información se realiza mediante SQL (Structured Query Language), la cual permite consultar, modificar, insertar y borrar datos en las tablas.

SQL (STRUCTURED QUERY LANGUAGE)

Las sentencias de SQL se dividen en dos grupos: Data Definition Language (DDL) y Data Manipulation (DML). Los DLL permiten crear y definir nuevas bases de datos, campos e índices. Los DML permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DLL

Comando	Descripción
CREATE	Utilizado para crear nuevas tablas, campos e índices
DROP	Empleado para eliminar tablas e índices

Comandos DML

Comando	Descripción
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados
DELETE	Utilizado para eliminar registros de una tabla de una base de datos

Las clausulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Cláusula	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar

A continuación se listan algunos de los tipos de datos disponibles:

NUMÉRICOS

- int
- float
- double

CADENAS

- char
- varchar

FECHAS

- date
- datetime

VALORES

- money

La sentencia **CRATE** pertenece al tipo DDL y permite crear tablas en una base de datos.

CREATE TABLE Clientes (CI varchar(10) primary key, Nombre varchar(10), Apellido varchar (10), Edad int not null);

primary key.- crea un índice y asegura que no podrán existir dos atributos iguales en el campo.

not null.- no permite la inserción de un atributo vacío.

Para añadir una tupla en la tabla Clientes, se utiliza la sentencia **INSERT**:

INSERT INTO Clientes (CI, Nombre, Apellido, Edad) VALUES (' 1234567890‘, ‘Luis ‘, ‘Almeida ‘, 20);

Se puede insertar un subconjunto de atributos si no se encuentra configurado como not null

INSERT INTO Clientes (CI, Nombre) VALUES (' 1234567890‘, ‘Luis);

SELECT selecciona todos los atributos que se le especifiquen.

SELECT Nombre, Apellido FROM Clientes;

Si se desea seleccionar todos los campos se usa el *., con lo que devuelve todos los campos de la tabla

SELECT * FROM Clientes

Si se desea restringir las tuplas.

SELECT CI FROM Clientes WHERE Edad < 30;

SELECT CI FROM Clientes WHERE Nombre= ‘Luis ‘;

SELECT CI FROM Clientes WHERE Edad < 30 & Nombre= ‘Luis ‘;

SQL permite incluir en sus condicones los operadores <,>, <=,>=, AND, OR.

En SQL se puede realizar consultas sobre varias tablas a la vez, para unificar el contenido de las mismas, esto permite no repetir datos en varias tablas. Para diferenciar atributos con el mismo nombre en distintas tablas, se debe utilizar el nombre de la tabla antes de referenciar el campo.

CLIENTES

Código	Nombre	Dirección
1	Luis Miguel	Guzmán el Bueno, 90
2	Beatriz	Zurriaga, 25
3	Jonás	Federico Puertas, 3
4	Joaquín	Vinateros, 121
5	Pedro	Virgen del Cerro, 154
6	Sandra	Pablo Neruda, 15
7	Miguel	Armadores, 1
8	Sergio	Avenida del Ejercito
9	Alejandro	Leonor de Cortinas,
10	Álvaro	Fuencarral, 33
11	Rocío	Cervantes, 22
12	Joaquín	Buenos Aires, 31
13	Jesús	Gaztambique, 32

PRODUCTOS

Referencia	Nombre	Precio	Concepto
1	Patatas	200 Pta	Kilo
2	Melones	500 Pta	Kilo
3	Sandías	120 Pta	Kilo
4	Zapatos	5.000 Pta	Par
5	Chandal	12.000 Pta	Unidad
6	Pantalones	2.000 Pta	Unidad
7	Camisa	2.500 Pta	Unidad
8	Corbata	950 Pta	Unidad
9	Aceite	695 Pta	Litro

PEDIDOS

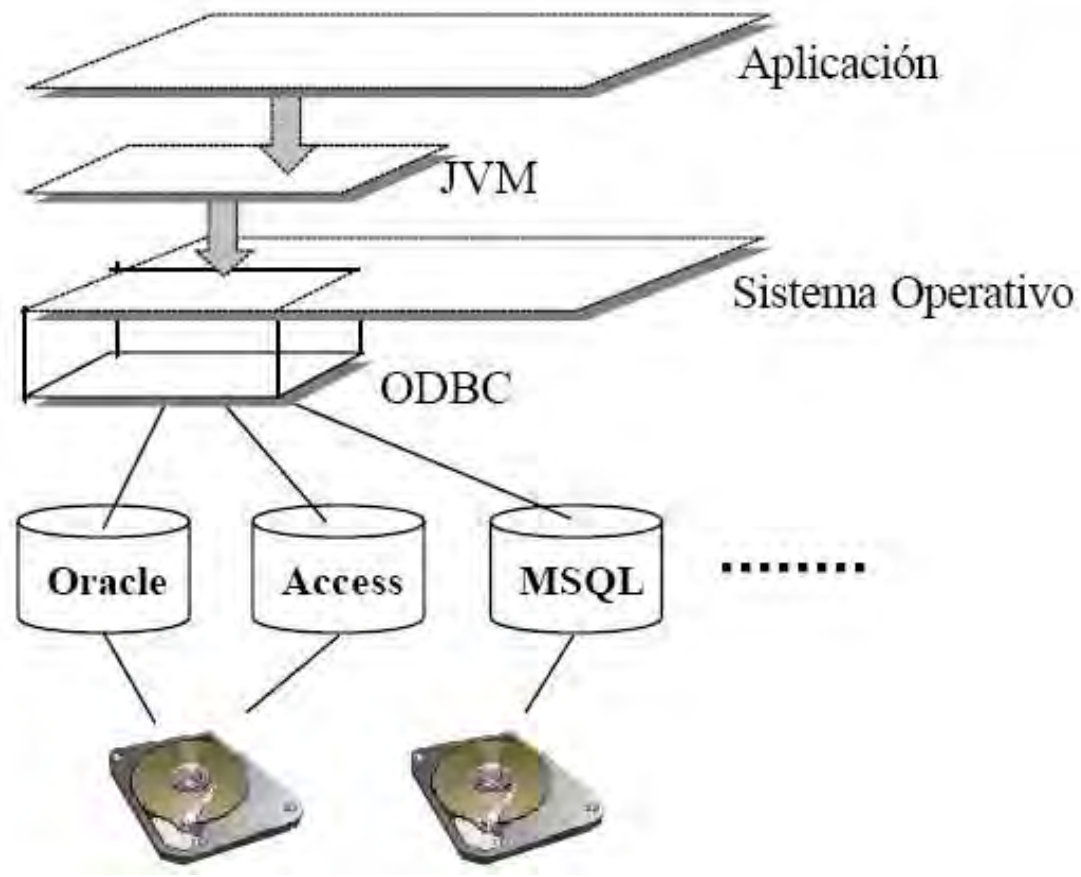
Código	Referencia	Cantidad
1	1	5
9	2	10
4	5	6
12	8	1
7	2	9
8	3	7
7	9	3
2	7	1
6	4	3
3	1	4

SELECT Nombre, Direccion, Referencia, Cantidad FROM Clientes, Pedidos WHERE Clientes.Codigo = Pedidos.Codigo

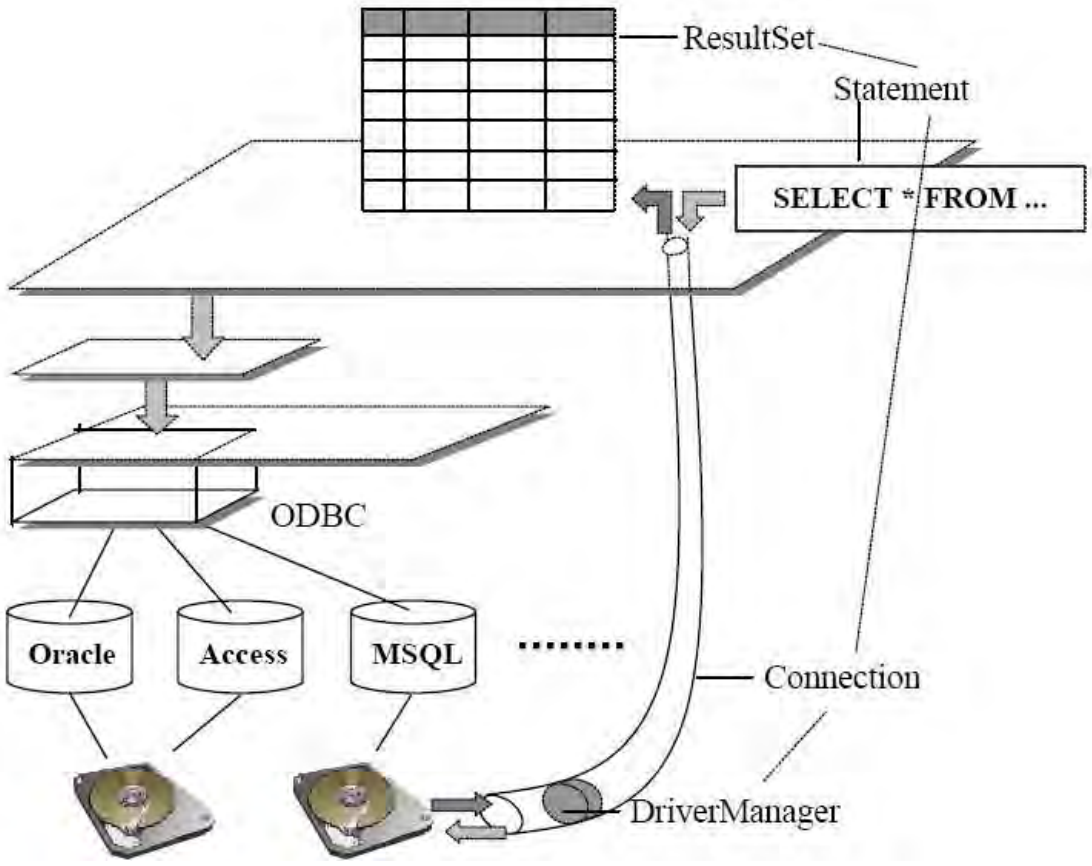
Clientes y Pedidos, no son necesarios cuando se hace referencia a Direccion, Nombre o Cantidad, ya que esos campos no se repiten en las dos tablas.

ARQUITECTURA

Una aplicación Java que realiza accesos a bases de datos funciona según una arquitectura que permite escribir los programas abstrayéndose de los detalles de los niveles inferiores (discos, drivers, sistema operativo, etc.)



En el nivel superior se encuentran las aplicaciones, estas aplicaciones son interpretadas por la máquina virtual Java (JVM). El sistema Operativo proporciona el nivel de entra/salida, que interactúa con los dispositivos físicos donde se encuentran las bases de datos. El sistema operativo también gestiona el nivel de ODBC (Open Data Base Conectivity); ODBC permite utilizar una interfaz única para los distintos tipos de bases de datos.



El tubo que se encuentra dibujado representa la clase **Connection**, que proporciona el medio para comunicarse con las bases de datos; según el tipo de la base de datos, los drivers serán diferentes. Una vez establecida la conexión, se crea una instancia de la clase **Statement**, utilizada para definir las sentencias SQL. Las sentencias SQL proporcionan una serie de datos provenientes de la base de datos que se almacenan en una instancia **ResultSet**.

LA INTERFAZ RESULTSET

Los objetos ResultSet permiten recoger los resultados de la ejecución de sentencias SQL; estos resultados proporcionan un número variable de columnas y de filas. La figura indica los métodos que permiten mover el cursor a lo largo de la tabla.

DNI	Nombre	Apellido	Edad
53023767A	Pedro	Rubio	18
84604568K	Ana	Moreno	35
82607936G	Carmen	Delgado	12
77459822P	Luis	Tejedor	47
2376856S	Jorge	Gil	21

Todos los métodos devuelven un valor de tipo boolean, que indica si el movimiento del cursor ha sido posible. Los objetos ResultSet únicamente pueden ser recorridos incrementalmente.

MYSQL SERVER

Para la instalación de MySQL Server, que es el motor de bases de datos que se utiliza es necesario disponer del motor de base de datos, un driver y un entorno de desarrollo gráfico, los cuales se pueden descargar gratuitamente de la página del fabricante.

http://dev.mysql.com/downloads/connector/j/

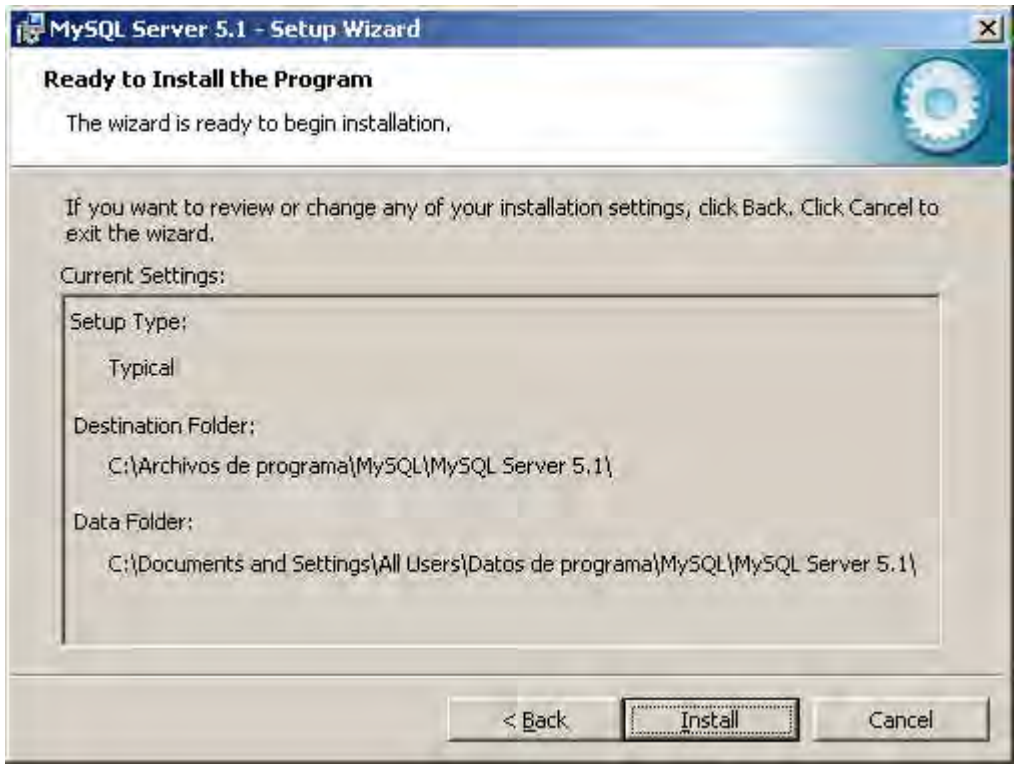
http://dev.mysql.com/downloads/workbench/5.2.html

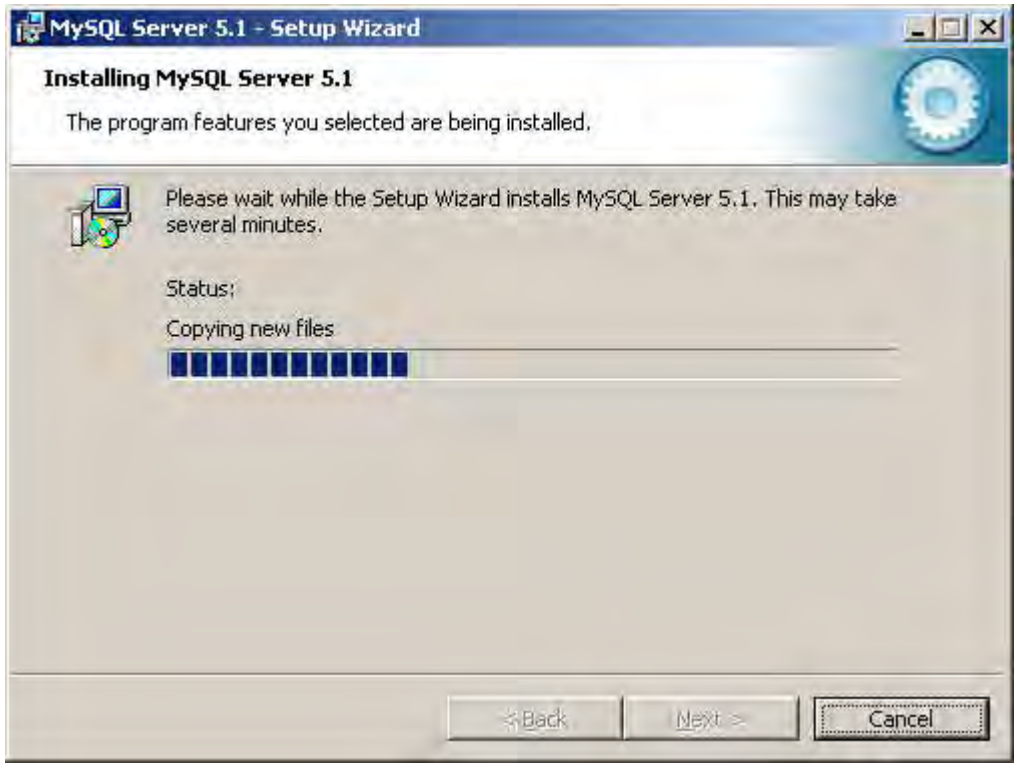
INSTALACIÓN

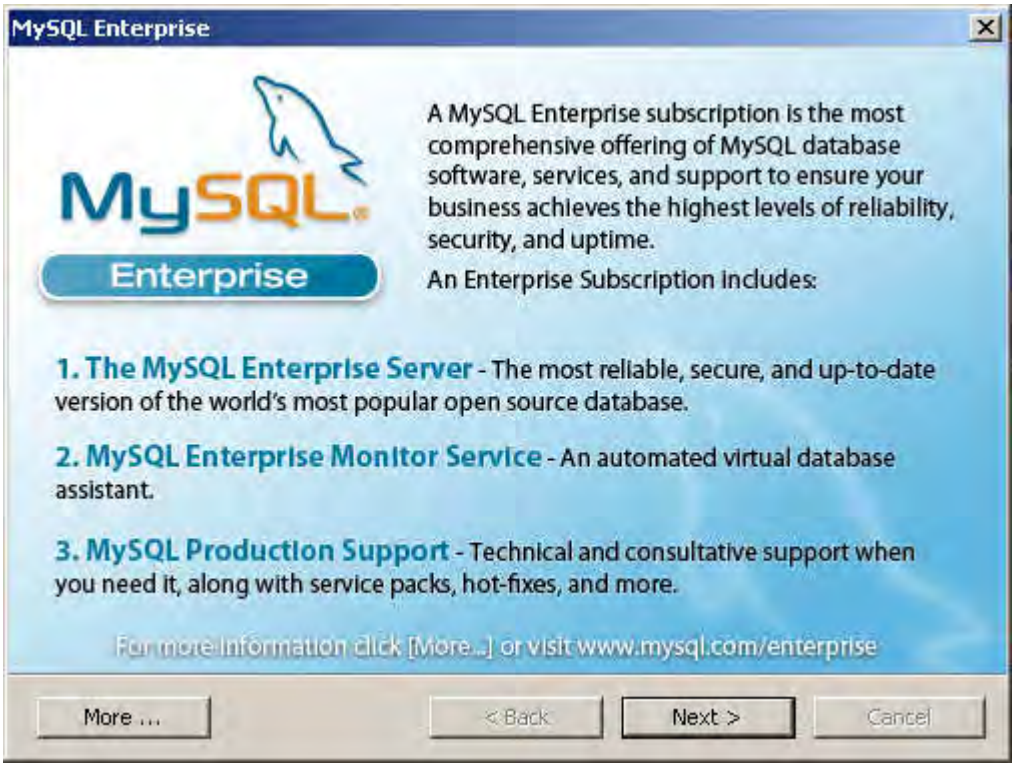
La instalación es muy sencilla, y prácticamente lo único que requiere es aceptar todas las opciones que presente el instalador, del motor de base de datos.


MySQL Server 5.1 - Setup Wizard
Welcome to the Setup Wizard for MySQL Server 5.1
The Setup Wizard will install MySQL Server 5.1 release 5.1.48 on your computer. To continue, click Next.
WARNING: This program is protected by copyright law.
MySQL
< Back
Next >
Cancel


MySQL Server 5.1 - Setup Wizard
Setup Type
Choose the setup type that best suits your needs.
Please select a setup type.
Typical
Common program features will be installed. Recommended for general use.
Complete
All program features will be installed. (Requires the most disk space.)
Custom
Choose which program features you want installed and where they will be installed. Recommended for advanced users.
< Back
Next >
Cancel


MySQL Server 5.1 - Setup Wizard
Ready to Install the Program
The wizard is ready to begin installation.
If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.
Current Settings:
Setup Type:
Typical
Destination Folder:
C:\Archivos de programa\MySQL\MySQL Server 5.1\
Data Folder:
C:\Documents and Settings\All Users\Datos de programa\MySQL\MySQL Server 5.1\
< Back
Install
Cancel


MySQL Server 5.1 - Setup Wizard
Installing MySQL Server 5.1
The program features you selected are being installed.
Please wait while the Setup Wizard installs MySQL Server 5.1. This may take several minutes.
Status:
Copying new files
< Back
Next >
Cancel


MySQL Enterprise
MySQL
Enterprise
A MySQL Enterprise subscription is the most comprehensive offering of MySQL database software, services, and support to ensure your business achieves the highest levels of reliability, security, and uptime.
An Enterprise Subscription includes:
1. The MySQL Enterprise Server - The most reliable, secure, and up-to-date version of the world's most popular open source database.
2. MySQL Enterprise Monitor Service - An automated virtual database assistant.
3. MySQL Production Support - Technical and consultative support when you need it, along with service packs, hot-fixes, and more.
For more information click [More...] or visit www.mysql.com/enterprise
More ...
< Back
Next >
Cancel


MySQL Server 5.1 - Setup Wizard
Wizard Completed
Setup has finished installing MySQL Server 5.1. Click Finish to exit the wizard.
Configure the MySQL Server now
Use this option to generate an optimized MySQL config file, setup a Windows service running on a dedicated port and to set the password for the root account.
MySQL
< Back
Finish
Cancel

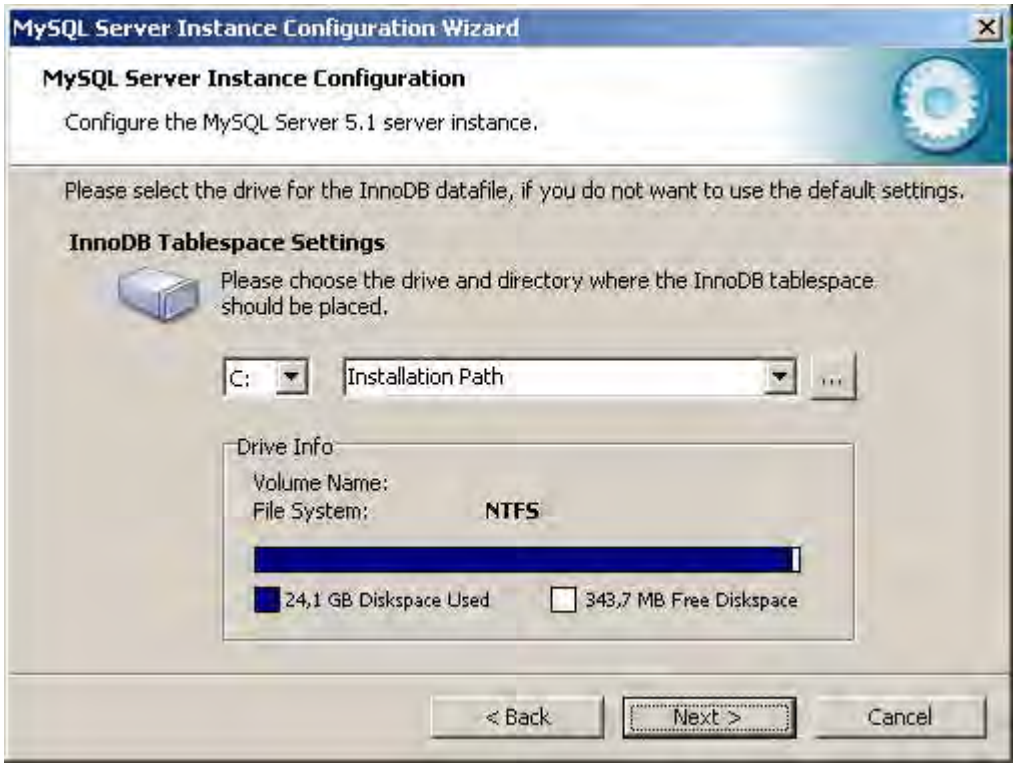
Marcar la opción "Detailed Configuration" , de esta forma se podra configurar más opciones de MySQL utilizando el asistente. Si se marca "Standard Configuration" el asistente pedirá menos información pero habría que configurar algunas opciones manualmente.

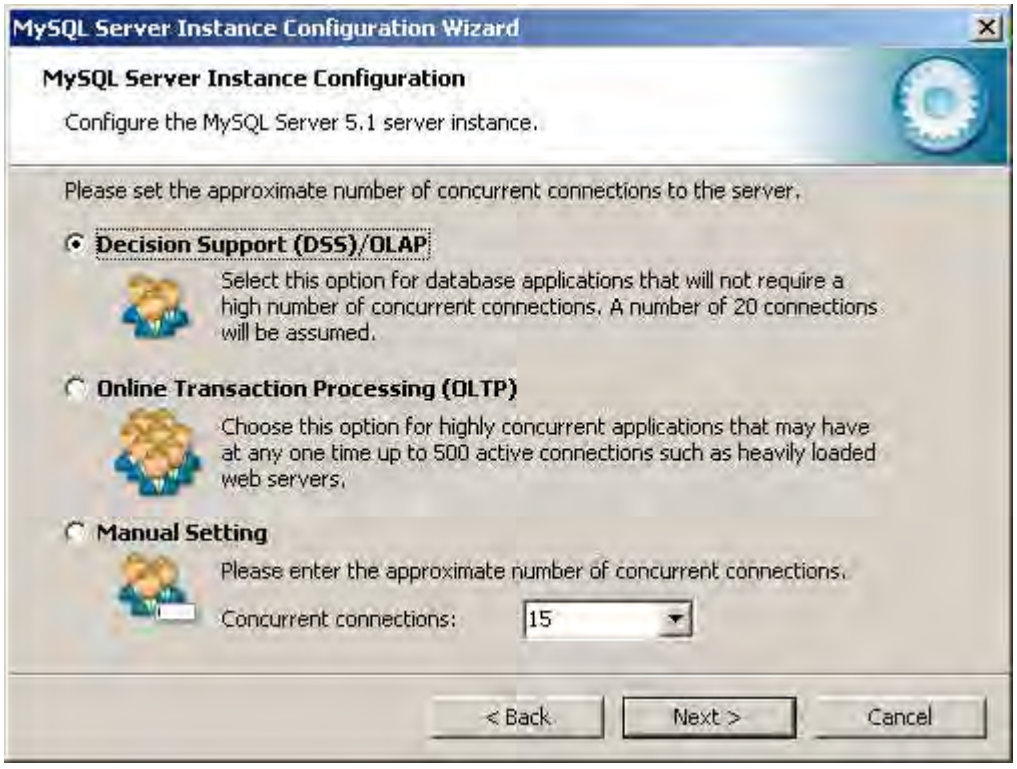
Developer Machine: se marcará esta opción si en el equipo donde se ha instalado MySQL Server se utiliza también para otras aplicaciones. MySQL Server utilizará la memoria mínima necesaria.

Server Machine: se marcará esta opción si se va a utilizar el equipo para algunas aplicaciones (no demasiadas). Con esta opción MySQL Server utilizará un nivel medio de memoria.

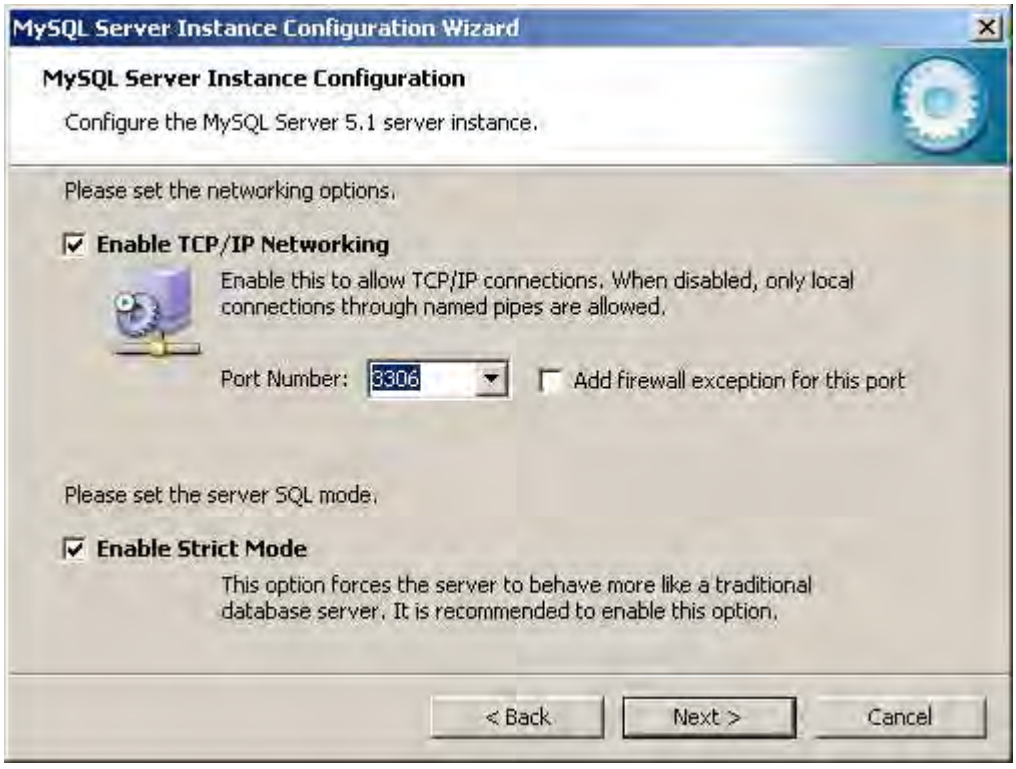
Dedicated MySQL Server Machine: se marcá esta opción sólo si se queire utilizar el equipo como un servidor dedicado exclusivamente a MySQL. Con esta opción MySQL Server utilizará el máximo de memoria disponble. Se obtendrá un rendimiento elevado pero el equipo sólo servirá para MySQL

Dependiendo del uso que se quiera dar a la Base de Datos se marcá una de las tres opciones siguientes, normalmente se marcará "Multifunctional Database" salvo que se quiera utilizar MySQL como base de datos para transacciones de otra Base de Datos MySQL.





Se selecciona el número aproximado de conexiones concurrentes (varios clientes conectados a la vez) que tendrá el servidor de MySQL). La primera opción asume unas 20, la segunda unas 500 y la tercera permite especificarlas manualmente. Este parámetro es aproximado.



Se dejará marcada la opción "Enable TCP/IP Networking" si se quiere que los clientes se puedan conectar mediante TCP/IP al equipo servidor de MySQL. Se puede cambiar el puerto por el que se establecerá la conexión, por defecto se suele dejar 3306 (si tenemos instalado algún firewall deben ser desactivados).

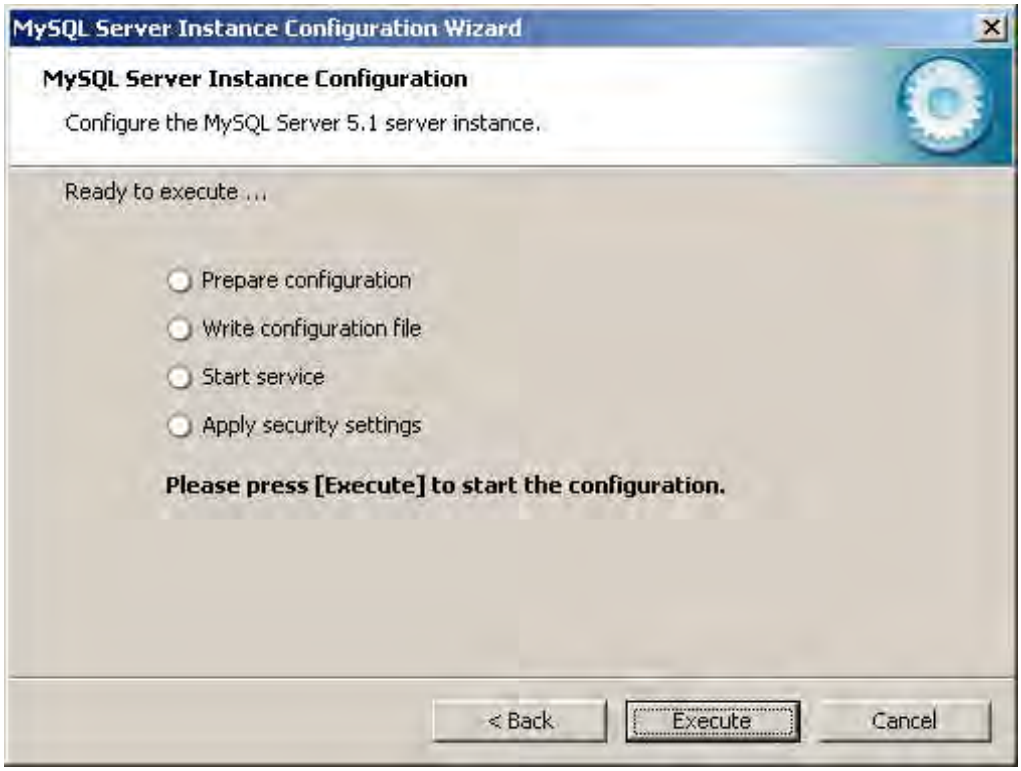
El siguiente paso es importante ya que pide que se especifique el tipo de arranque de MySQL Server.

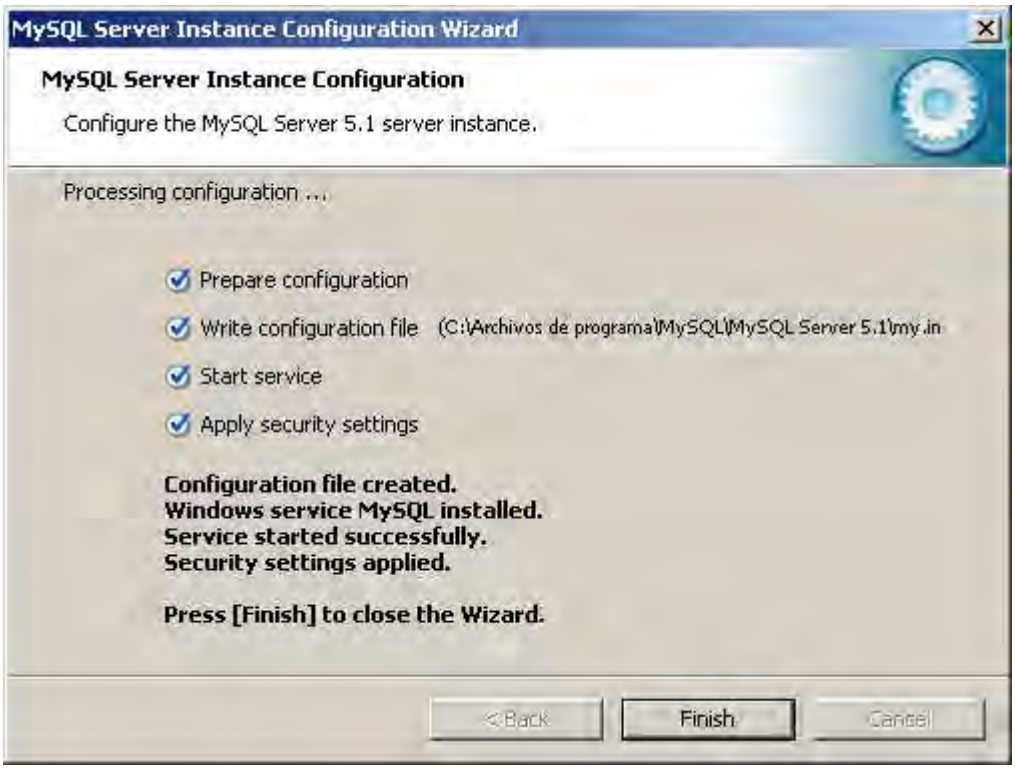
- La primera opción ("Install As Windows Service") el programa de instalación creará un Servicio que será el encargado de ejecutar MySQL Server, también permite especificar el nombre del servicio y si se quiere que arranque automáticamente al iniciar el sistema ("Launch the MySQL Server automatically").
- La segunda opción "Include Bin Directory in Windows PATH añadirá las variables de entorno necesarias para la ejecución de los ficheros necesarios para iniciar MySQL.

La opción recomendada es "Install As Windows Service".

A continuación se establece la contraseña para el usuario administrador (root).

Por último se pulsa en "Execute" para finalizar la configuración de MySQL.

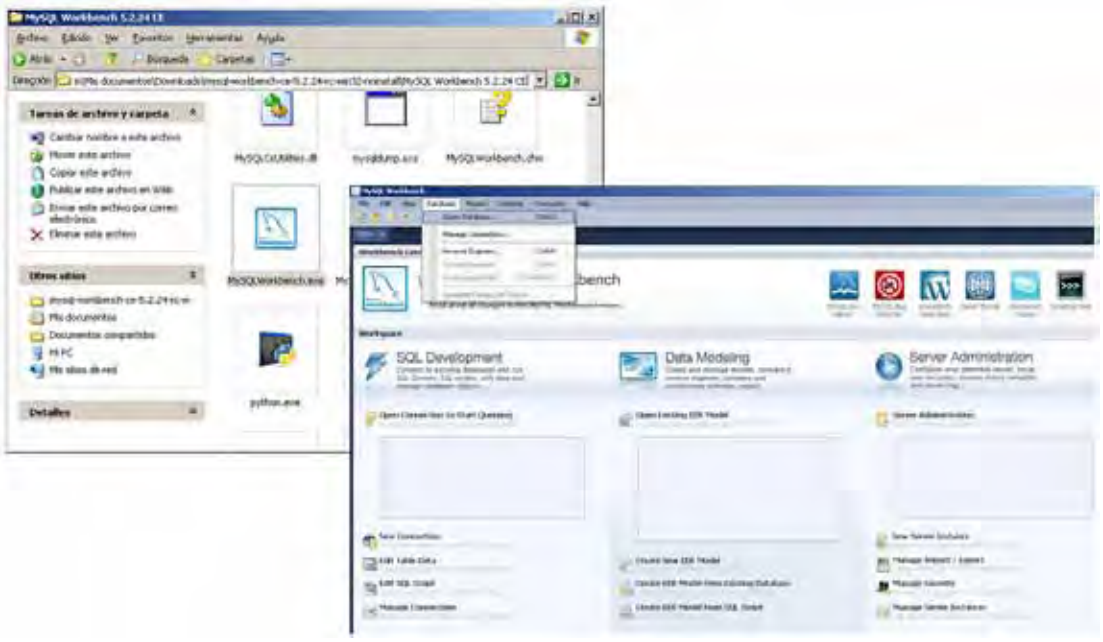




Si no hay problemas mostrará la ventana indicando que el proceso de instalación y configuración de MySQL Server ha terminado y se ha instalado e iniciado el Servicio que ejecutará MySQL.

Para comprobar que el servicio se está ejecutando se accede al administrador de tareas

El motor de bases de datos de MySQL no dispone de una interfaz gráfica, y corre directamente en consola, pero existen herramientas que permiten un trabajo más gráfico como MySQL Workbench, que puede instalarse o ejecutarse directamente desde el dirección sin necesidad de instalación.



Ejemplo No. 1:

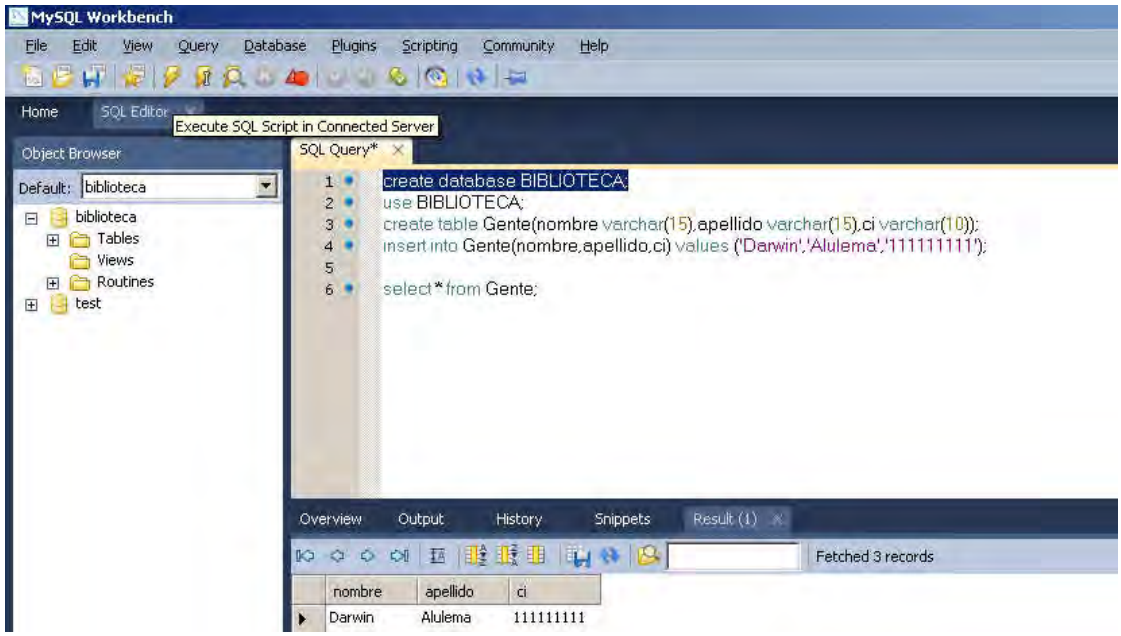
Utilizando MySQL Workbench, ejecute el siguiente script SQL:

SQL Query*

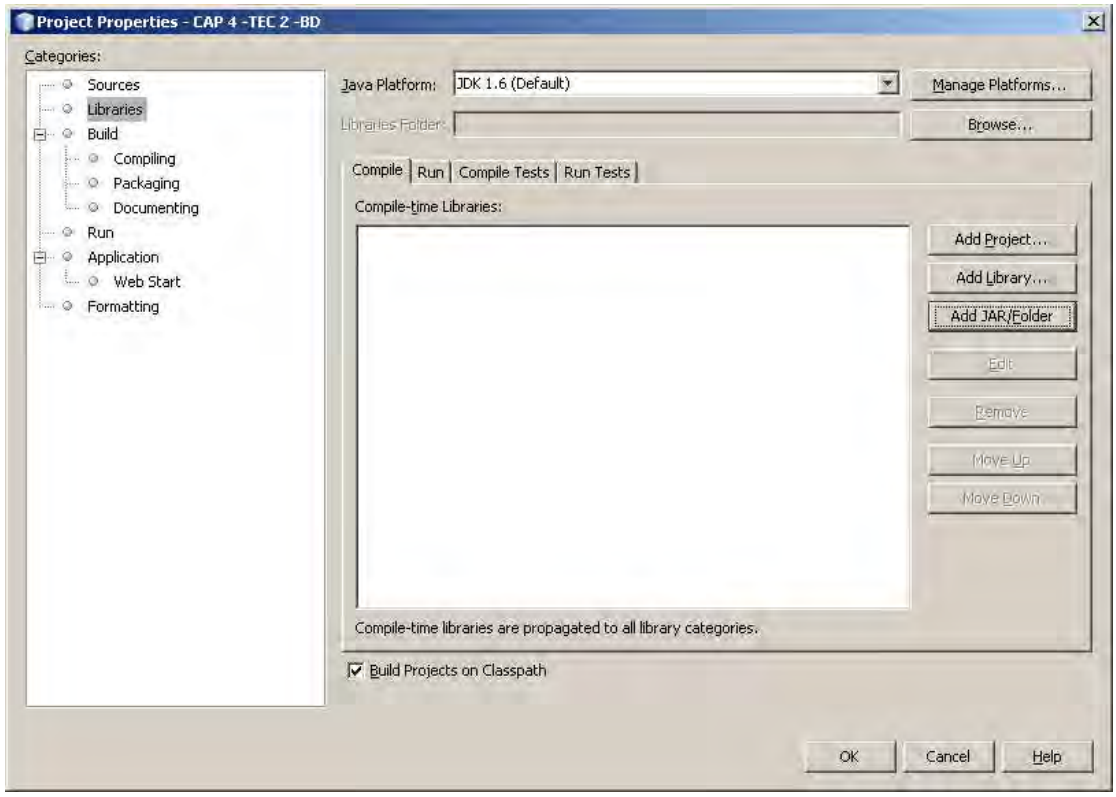
```
1  create database EJEMBASEDATOS;
2  use EJEMBASEDATOS;
3  create table DATOS(nombre varchar(10)primary key,edad int);
4
5  insert into DATOS values ('LUIS',24);
6  insert into DATOS values ('ANDREA',25);
7  insert into DATOS values ('CARLOS',20);
8  insert into DATOS values ('LIZ',24);
9  insert into DATOS values ('LUIS',24);
10 delete from DATOS where nombre='LUIS';
11 drop table DATOS;
12
13 select *from DATOS where nombre='andrea';
14
15 update datos set edad=10 where nombre='andrea';
16
17 select *from DATOS;
```

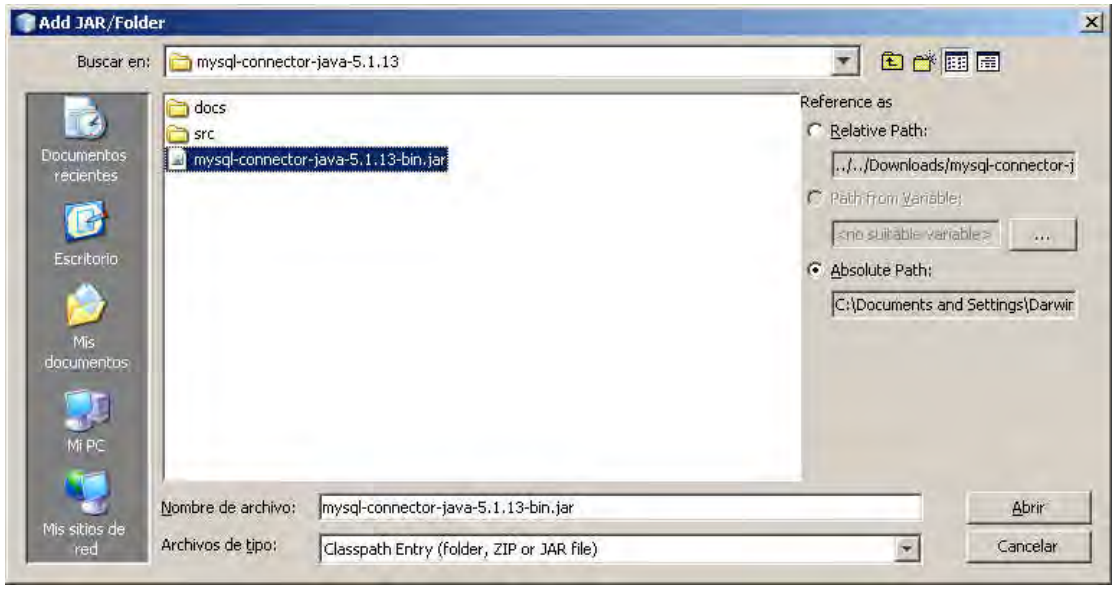
Ejemplo No. 2:

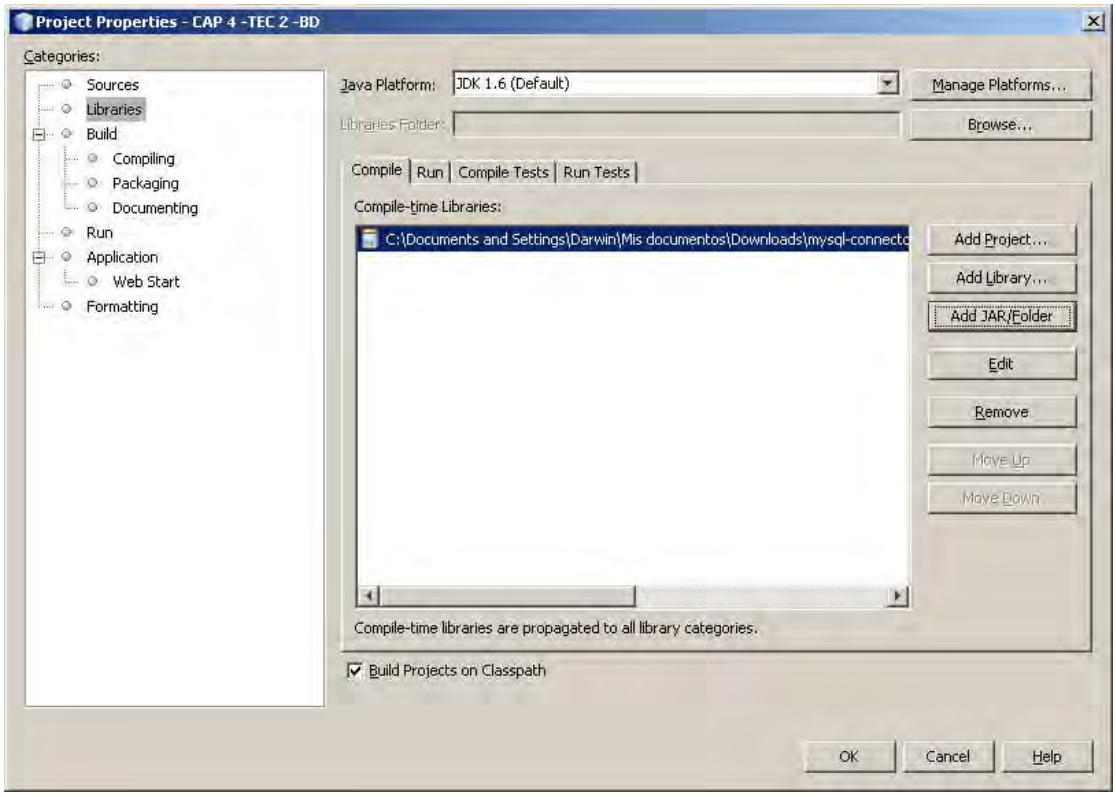
Cree una aplicación en Java que permita consultar en una base de datos, que contiene el nombre, apellido y el número de cédula de una persona. La base de datos es creada y llenada desde MySQL Server.



Para poder controlar la base de datos des una aplicación Java es necesario cargar el driver en la aplicación.


Project Properties - CAP 4 -TEC 2 -BD
Categories:
Sources
Libraries
Build
Compiling
Packaging
Documenting
Run
Application
Web Start
Formatting
Java Platform:
JDK 1.6 (Default)
Manage Platforms...
Libraries Folder:
Browse...
Compile
Run
Compile Tests
Run Tests
Compile-time Libraries:
Add Project...
Add Library...
Add JAR/Folder
Edit
Remove
Move Up
Move Down
Compile-time libraries are propagated to all library categories.
Build Projects on Classpath
OK
Cancel
Help


Add JAR/Folder
Buscar en:
mysql-connector-java-5.1.13
Documentos recientes
Escritorio
Mis documentos
Mi PC
Mis sitios de red
docs
src
mysql-connector-java-5.1.13-bin.jar
Reference as
Relative Path:
../../Downloads/mysql-connector-j
Path from Variable:
<no suitable variable>
...
Absolute Path:
C:\Documents and Settings\Darwir
Nombre de archivo:
mysql-connector-java-5.1.13-bin.jar
Archivos de tipo:
Classpath Entry (folder, ZIP or JAR file)
Abrir
Cancelar



NOTA: Debe estar el servidor de **MySQL** iniciado.

El servidor de **MySQL** abre por defecto el puerto 3306 para aceptar conexiones de posibles clientes, de programas que quieran conectarse y acceder a la base de datos.

Todas la clases necesarias para manejar la base de datos están en java.sql.*. Casi todos los métodos relativos a base de datos pueden lanzar la excepción **SQLException**, se trabaja con un **try**-**catch**. Además se debe incluir el jar que contiene el driver **MySQL.**

Connection *conexion* = DriverManager.getConnection
("jdbc:mysql://localhost/prueba","root", "clave");

- El primer parámetro del método **getConnection()** es un **String** que contiene la url de la base de datos:
- **jdb:mysql** porque se esta utilizando un driver jdbc para **MySQL.**
- **localhost** porque el servidor de base de datos, en este caso, está en el mismo ordenador en el que se va a correr el prorama java. Aquí puede ponerse una IP o un nombre de máquina que esté en la red.
- **prueba** es el nombre de la base de datos que se ha creado dentro de mysql. Se debe poner la base de datos dentro del servidor de **MySQL** a la que se quiere uno conectar.

- Los otros dos parámetros son dos **String**. Corresponden al nombre de usuario y password para acceder a la base de datos. Al instalar **MySQL** se crea el usuario root y se pide la password para él.

Esta conexión es en realidad un socket entre java y la base de datos, aunque para nosotros es transparente. Lo que sí es importante, es saber que si varios hilos comparten esta conexión, deben usarla sincronizadamente. Otra opción es que cada hilo cree su propia conexión.

Para realizar cualquier acción sobre la base de datos (consulta, insertar nuevos registros, modificar los existentes o borrar), necesitamos una clase **Statement**. Para obtenerla, se le pide dicha clase a la conexión.

```
Statement s = conexion.createStatement();
ResultSet rs = s.executeQuery ("select * from persona");
```

El **Statement** obtenido tiene un método **executeQuery()**. Este método sirve para realizar una consulta a base de datos. El parámetro que se pasa es un String en el que está la consulta en lenguaje SQL. No hace falta terminarlo con punto y coma.

El resultado lo devuelve el método como un **ResultSet**. ResultSet no es más que una clase java en la que está el resultado de la consulta. ResulSet no contiene todos los datos, sino que los va consiguiendo de la base de datos según se van pidiendo.

El **ResultSet** contiene dentro los registros leidos de la base de datos, tiene internamente un "puntero" apuntando justo delante del primer registro.

- El método **next()** del **ResultSet** hace que dicho puntero avance al siguiente registro, en este caso, al primero. Si lo consigue, el método **next()** devuelve **true**. Si no lo consigue devuelve **false**.
- Una forma de ir leyendo los registros en meternos en un **while**.

```
while (rs.next())
{
   System.out.println (rs.getInt (1) + " " + rs.getString (2)+ " " +rs.getDate(3));
}
```

Una vez que el "puntero" está apuntando a un registro, los métodos **getInt()**, **getString()**, **getDate()**, etc van devolviendo los valores de los campos de dicho registro. Se puede pasar a estos métodos un índice (que comienza en 1) para indicar qué columna de la tabla de base de datos se desea. También se puede usar un **String** con el nombre de la columna (tal cual está en la tabla de base de datos).

RecordSet es capaz de hacer la conversión de los distintos tipos de datos a String. Por ejemplo, en cualquiera de los campos anteriores se puede pedir un **getString()** y se devolveran los números como **String** y la fecha como **String**.

Una vez que se termina de usar la conexión, se debería cerrar, o bien terminar el programa, con lo que se cierra automáticamente.

conexion.close();

```
import java.sql.*;
public class Main {
    public static void main(String[] args) {
        String url="jdbc:mysql://localhost/BIBLIOTECA";
        String usuario="root";
        String clave="dalulema";
       try
        {
            DriverManager.registerDriver(new org.gjt.mm.mysql.Driver());
            System.out.println("Driver MySQL cargado...");
            Connection conexion = DriverManager.getConnection (url,usuario,clave);
            System.out.println("Conexion abierta con "+url+"...");
            Statement s = conexion.createStatement();
            System.out.println("Sentencia creada...\n");
            // Se realiza la consulta. Los resultados se guardan en el
            // ResultSet rs
            ResultSet rs = s.executeQuery ("select * from Gente");
            // Se recorre el ResultSet, mostrando por pantalla los resultados.
            while (rs.next())
            {
                System.out.println (rs.getString("nombre") + " " + rs.getString (2)+
                    " " + rs.getString(3));
            }
            conexion.close();
            System.out.println("\nConexión cerrada...");
        }
       catch (Exception e)
        {
            System.err.println("ERROR:...."+e.getMessage());
        }
    }
}
```

Ejemplo No. 2:

Implemente un programa en Java que permita insertar y buscar datos en una base de datos creada en MySQL Server.

```
public class Estudiante {
    private String CI="";
    private String NOMBRE="";
    private String APELLIDO="";

    public String getCI() {
        return CI;
    }
    public void setCI(String CI) {
        this.CI = CI;
    }
    public String getNOMBRE() {
        return NOMBRE;
    }
    public void setNOMBRE(String NOMBRE) {
        this.NOMBRE = NOMBRE;
    }

    public String getAPELLIDO() {
        return APELLIDO;
    }

    public void setAPELLIDO(String APELLIDO) {
        this.APELLIDO = APELLIDO;
    }
}
```

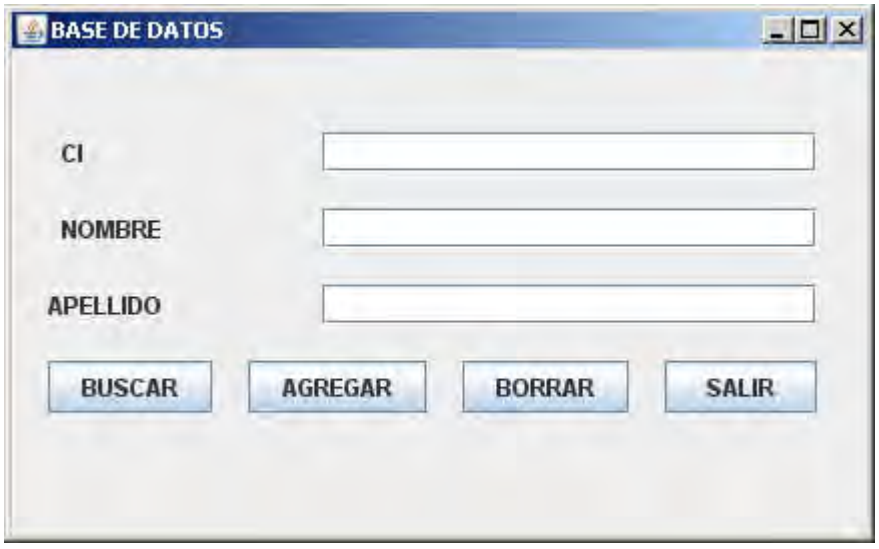
Statement permite ejecutar consultas básicas SQL y recibir los resultados mediante la clase ResultSet.

Para crear una instancia de Statement, debe llamar al método createStatement() en el objeto Connection que ha obtenido via uno de los métodos DriverManager.getConnection().

```
import java.sql.*;

public class Biblioteca extends Estudiante{
    Statement  stmt = null;
    Connection con  = null;
    ResultSet  rs   = null;
    public Biblioteca() {
        String url="jdbc:mysql://localhost/BIBLIOTECA";
        String usuario="root";
        String clave="dalulema";
       try
        {
            DriverManager.registerDriver(new org.gjt.mm.mysql.Driver());
            System.out.println("Driver MySQL cargado...");
            con = DriverManager.getConnection (url,usuario,clave);
            System.out.println("Conexion abierta con "+url+"...");
            stmt = con.createStatement();
            System.out.println("Sentencia creada...\n");
        }
        catch(Exception e){
            System.out.println("Error: "+e);
        }
    }



 public boolean buscar(String aux)
 {
     try{
         stmt.executeUpdate("use BIBLIOTECA;");
         rs=stmt.executeQuery("select *from Gente where ci ='"+aux+"';");
         while(rs.next())
         {
             setNombre(rs.getString(1));
             setApellido(rs.getString(2));
             return true;
         }
     }
     catch(Exception e){
         System.out.println("Error: "+e);
     }
     return false;
 }
```

```
public void agregar(String a,String b,String c)
{
     try{
        stmt.executeUpdate("use BIBLIOTECA;");
        stmt.executeUpdate("insert into Gente (nombre,apellido,ci) values('"
                +a+"','"+b+"','"+c+"')");
    }
    catch(Exception e){
        System.out.println("Error: "+e);
    }
}
public void borrar(String aux)
{
    try{
        stmt.executeUpdate("use BIBLIOTECA;");
        stmt.executeUpdate("delete from Gente where ci='"+aux+"';");
    }
    catch(Exception e){
        System.out.println("Error: "+e);
    }
}
```

```
     public void cerrar()
     {
          try{
               con.close();
               System.out.println("Conexión cerrada");
          }
          catch(Exception e){
               System.out.println("Error: "+e);
          }
     }
 }
```



```
public class Ventana extends javax.swing.JFrame {
    Biblioteca obj;
    String     ci;
    String     nombre;
    String     apellido;

    public Ventana() {
        initComponents();
        ci       = "";
        nombre   = "";
        apellido = "";
        obj      = new Biblioteca();
    }
```

```
private void btnAGREGARActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    ci=txtCI.getText();
    nombre=txtNOMBRE.getText();
    apellido=txtAPELLIDO.getText();
    obj.agregar(nombre,apellido,ci);
}

private void btnSALIRActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    obj.cerrar();
    System.exit(0);
}
```

```
private void btnBUSCARActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    ci=txtCI.getText();
    if(obj.buscar(ci))
    {
        txtNOMBRE.setText(obj.getNombre());
        txtAPELLIDO.setText(obj.getApellido());
    }
}

private void btnBORRARActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    ci=txtCI.getText();
    obj.borrar(ci);
}
```

ACTIVIDADES DE REELABORACION

ACTIVIDADES PRÁCTICAS

1.- Modifique el Ejemplo 2 para agregar una opción con la que se pueda leer todos los datos guardados en la base de datos y que se desplieguen en una tabla.

2.- Reemplaza la interfaz gráfica del Ejemplo 2 para que trabaje con servlets.

3.- Modifique la actividad anterior para agregar una opción con la que se pueda leer todos los datos guardados en la base de datos y que se desplieguen en una tabla.

CAPITULO 13

PROGRAMACIÓN DE DISPOSITIVOS MÓVILES CON J2ME

En el proceso de creación de contenidos para dispositivos móviles hay variables tecnológicas, de contenido y procedimentales a considerar.

La primera de ellas son los dispositivos móviles en sí, cuyas características determinan el modo de consumir la información. Entre ellas, el tamaño de la pantalla, el tipo de teclado y el soporte de ciertas tecnologías multimedia. Un segundo factor inherente a la elección de un equipo telefónico es el operador de telefonía móvil y las tecnologías de acceso que éste ofrezca para conectarse a Internet.

DISPOSITIVOS MÓVILES

Un dispositivo móvil es un término que no se refiere solamente a los teléfonos móviles. En él se incluyen algunas consolas de juegos -como la PlayStation Portable (PSP)-, el iPod Touch y los Asistentes Digitales Personales (PDA, en inglés), como la Palm. Todos éstos son equipos electrónicos portátiles que permiten acceder a los servicios de Internet, especialmente la Web, además de prestar otros servicios, como llamadas de voz, mensajería de texto, agenda y reproducción de multimedia.

REDES Y TECNOLOGÍAS DE TELEFONÍA MÓVIL

Cada una de las siguientes tecnologías implica redes de telecomunicaciones, velocidades de transmisión y equipos móviles diferentes.

1G: esta primera generación de redes para móviles tuvo su inicio en la década de los 80. Se trataba de un servicio analógico que sólo permitía llamadas de voz.

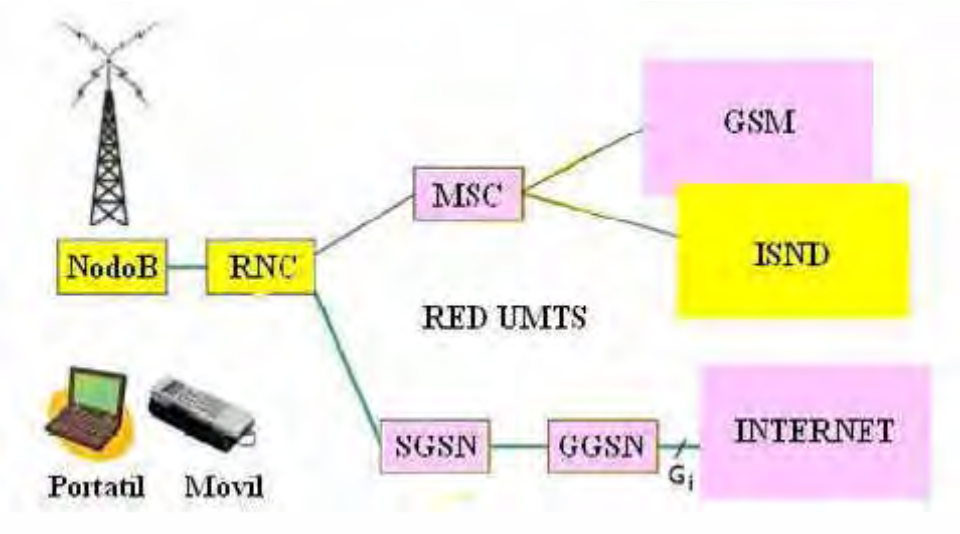
2G: redes de segunda generación, originadas en 1990. Se diferencia de las anteriores por su carácter digital y el uso de protocolos como GSM. Esta primera característica permitió la aparición de teléfonos más pequeños, gracias al ahorro de energía.

2.5G: es un término eminentemente comercial, ya que son las redes de segunda generación a las que se les aplica una tecnología conocida como GPRS (General Packet Radio System), que permite velocidades de acceso a la red que van desde los 56 Kbps a los 114 Kbps. Con este servicio se popularizaron los sitios WAP (Wireless Application Protocol), el envío de SMS (Short Message Service) y MMS (Multimedia Messaging System).

3G: comenzaron en 2001 en Japón, llegando varios años después a Chile. Permite un uso simultáneo de voz y datos, a velocidades que pueden llegar a 14 Mbps.

4G: es una red cuyo lanzamiento se proyecta para 2010 en Japón y unos años después en el resto del mundo. Se basa en la utilización del protocolo IP en teléfonos móviles y entrega velocidades de acceso de hasta 1 Gbps. Wimax y LTE (Long Term Evolution) son dos tecnologías competidoras en esta fase de desarrollo.

Por ejemplo a continuación se ilustra la arquitectura 3G.

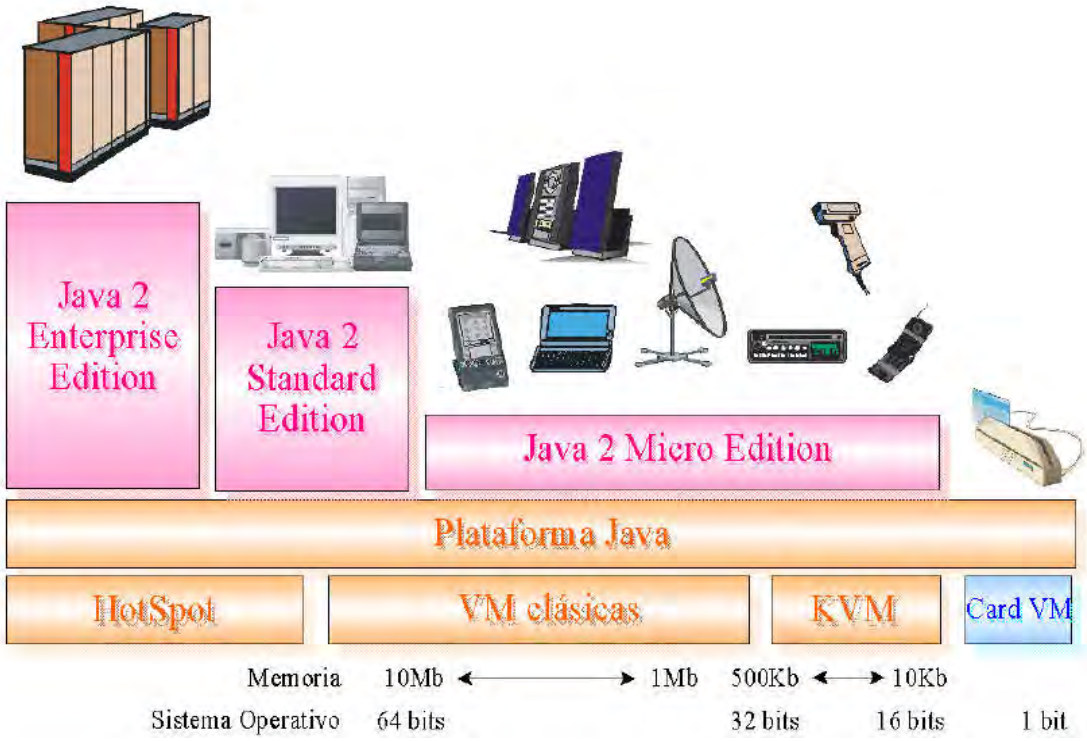


JAVA MICRO EDITION

La edición Java 2 Micro Edition fue presentada en 1999 por Sun Microsystems con el propósito de habilitar aplicaciones Java para pequeños dispositivos. En esta presentación, lo que realmente se enseñó fue una primera versión de una nueva Java Virtual Machine (JVM) que podía ejecutarse en dispositivos Palm.

Java Micro Edition es la versión del lenguaje Java que está orientada al desarrollo de aplicaciones para dispositivos pequeños con capacidades restringidas tanto en pantalla gráfica, como de procesamiento y memoria (teléfonos móviles, PDA`s, Handhelds, Pagers, etc). La tardía aparición de esta tecnología, puede ser debido a que las necesidades de los usuarios de telefonía móvil ha cambiado mucho en estos últimos años y cada vez demandan más servicios y prestaciones por parte tanto de los terminales como de las compañías.

El uso de esta tecnología depende del asentamiento en el mercado de otras, como GPRS, íntimamente asociada a J2ME y que no ha estado a nuestro alcance hasta hace poco. Actualmente las compañías telefónicas y los fabricantes de móviles están implantando los protocolos y dispositivos necesarios para soportar J2ME.



A continuación se puede ver los componentes que forman parte de esta tecnología.

- Máquinas virtuales Java con diferentes requisitos, para diferentes tipos de dispositivos.
- Configuraciones, que son un conjunto de clases básicas orientadas a conformar el corazón de las implementaciones para dispositivos de características específicas.
    - Connected Limited Device Configuration (CLDC) enfocada a dispositivos con restricciones de procesamiento y memoria
    - Connected Device Configuration (CDC) enfocada a dispositivos con más recursos.
- Perfiles, que son bibliotecas Java de clases específicas orientadas a implementar funcionalidades de más alto nivel para familias específicas de dispositivos.

KVM

KVM proviene de Kilobyte (haciendo referencia a la baja ocupación de memoria, entre 40Kb y 80Kb). Se trata de una implementación de Máquina Virtual reducida y especialmente orientada a dispositivos con bajas capacidades computacionales y de memoria.

La KVM está escrita en lenguaje C, aproximadamente unas 24000 líneas de código, y fue diseñada para ser:

- Pequeña, con una carga de memoria entre los 40Kb y los 80 Kb, dependiendo de la plataforma y las opciones de compilación.
- Alta portabilidad.

- Modulable.
- Lo más completa y rápida posible y sin sacrificar características para las que fue diseñada.

La baja ocupación de memoria hace que posea algunas limitaciones con respecto a la clásica *Java Virtual Machine* (JVM):

1. No hay soporte para tipos en coma flotante. No existen por tanto los tipos double ni float. Esta limitación está presente porque los dispositivos carecen del hardware necesario para estas operaciones.
2. No existe soporte para JNI (*Java Native Interface*) debido a los recursos limitados de memoria.
3. No existe la finalización de instancias de clases. No existe el método Object.finalize().
4. Limitada capacidad para el manejo de excepciones debido a que el manejo de éstas depende en gran parte de las APIs de cada dispositivo por lo que son éstos los que controlan la mayoría de las excepciones.

CVM

CVM (Compact Virtual Machine) ha sido tomada como Máquina Virtual Java de referencia para la configuración CDC y soporta las mismas características que la Máquina Virtual de J2SE. Está orientada a dispositivos electrónicos con procesadores de 32 bits de gama alta y en torno a 2Mb o más de memoria RAM. Las características que presenta esta Máquina Virtual son:

1. Sistema de memoria avanzado.
2. Tiempo de espera bajo para el recolector de basura.
3. Separación completa de la VM del sistema de memoria.
4. Recolector de basura modularizado.
5. Portabilidad.
6. Rápida sincronización.
7. Ejecución de las clases Java fuera de la memoria de sólo lectura (ROM).
8. Soporte nativo de hilos.
9. Baja ocupación en memoria de las clases.
10. Proporciona soporte e interfaces para servicios en Sistemas Operativos de Tiempo Real.
11. Conversión de hilos Java a hilos nativos.
12. Soporte para todas las características de Java2 v1.3 y librerías de seguridad, referencias débiles, Interfaz Nativa de Java (JNI), invocación remota de métodos (RMI), Interfaz de depuración de la Máquina Virtual (JVMDI).

CONFIGURACIONES

Existen dos configuraciones en J2ME:

- CLDC, orientada a dispositivos con limitaciones computacionales y de memoria.
- CDC, orientada a dispositivos con no tantas limitaciones.

CONFIGURACIÓN DE DISPOSITIVOS CON CONEXIÓN (CDC)

CDC está orientada a dispositivos con cierta capacidad computacional y de memoria. Por ejemplo, decodificadores de televisión digital, televisores con internet, algunos electrodomésticos y sistemas de navegación en automóviles. CDC usa una Máquina Virtual Java similar en sus características a una de J2SE, pero con limitaciones en el apartado gráfico y de memoria del dispositivo.

CDC está enfocada a dispositivos con las siguientes capacidades:

1. Procesador de 32 bits.
2. Disponer de 2 Mb o más de memoria total, incluyendo memoria RAM y ROM.
3. Poseer la funcionalidad completa de la Máquina Virtual Java.
4. Conectividad a algún tipo de red.

Ésta Máquina Virtual es la que es la CVM (Compact Virtual Machine). Las peculiaridades de la CDC están contenidas principalmente en el paquete javax.microedition.io, que incluye soporte para comunicaciones http y basadas en datagramas.

A continuación las librerías incluidas en CDC.

Nombre de Paquete CDC	Descripción
java.io	Clases e interfaces estándar de E/S.
java.lang	Clases básicas del lenguaje.
java.lang.ref	Clases de referencia.
java.lang.reflect	Clases e interfaces de reflection.
java.math	Paquete de matemáticas.
java.net	Clases e interfaces de red.
java.security	Clases e interfaces de seguridad
java.security.cert	Clases de certificados de seguridad.
java.text	Paquete de texto.
java.util	Clases de utilidades estándar.
java.util.jar	Clases y utilidades para archivos JAR.
java.util.zip	Clases y utilidades para archivos ZIP y comprimidos.
javax.microedition.io	Clases e interfaces para conexión genérica CDC.

CONFIGURACIÓN DE DISPOSITIVOS LIMITADOS CON CONEXIÓN (CLDC)

CLDC está orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, cómputo y memoria. Un ejemplo de éstos dispositivos son: teléfonos móviles, buscapersonas (pagers), PDAs, organizadores personales, etc. Algunas de las restricciones vienen dadas por el uso de la KVM, necesaria al trabajar con la CLDC debido a su pequeño tamaño.

Los dispositivos que usan CLDC deben cumplir los siguientes requisitos:

1. Disponer entre 160 Kb y 512 Kb de memoria total disponible. Como mínimo se debe disponer de 128 Kb de memoria no volátil para la Máquina Virtual Java y las bibliotecas CLDC, y 32 Kb de memoria volátil para la Máquina Virtual en tiempo de ejecución.
2. Procesador de 16 o 32 bits con al menos 25 Mhz de velocidad.
3. Ofrecer bajo consumo, debido a que éstos dispositivos trabajan con suministro de energía limitado, normalmente baterías.
4. Tener conexión a algún tipo de red, normalmente sin cable, con conexión intermitente y ancho de banda limitado (unos 9600 bps).

CLDC aporta las siguientes funcionalidades a los dispositivos:

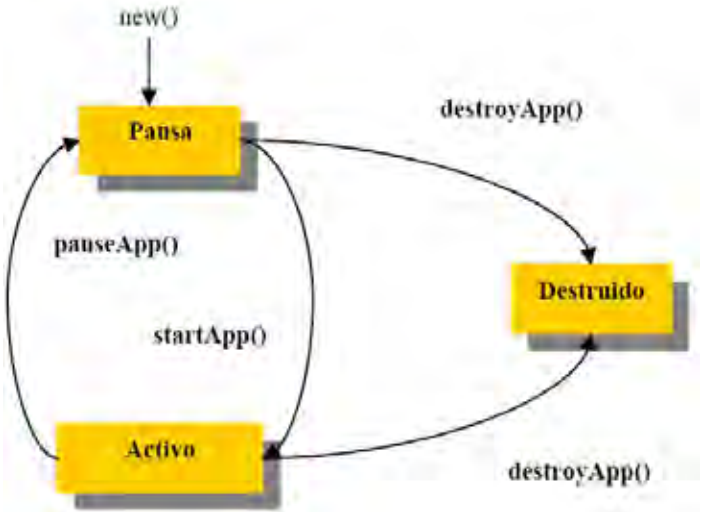
1. Un subconjunto del lenguaje Java y todas las restricciones de su Máquina Virtual (KVM).
2. Un subconjunto de las bibliotecas Java del núcleo.
3. Soporte para E/S básica.
4. Soporte para acceso a redes.
5. Seguridad.

A continuación las librerías incluidas en CLDC.

Nombre de paquete CLDC	Descripción
java.io	Clases y paquetes estándar de E/S. Subconjunto de J2SE.
java.lang	Clases e interfaces de la Máquina Virtual. Subconj. de J2SE.
java.util	Clases, interfaces y utilidades estándar. Subconj. de J2SE.
javax.microedition.io	Clases e interfaces de conexión genérica CLDC

ESTADOS DE UN MIDLET EN FASE DE EJECUCIÓN

Durante la ejecución el *MIDlet* es cargado en la memoria del dispositivo y es aquí donde puede transitar entre 3 estados diferentes: Activo, en pausa y destruido.



Cuándo un *MIDlet* comienza su ejecución, está en el estado “Activo” pero, ¿qué ocurre si durante su ejecución se recibe una llamada o un mensaje? El gestor de aplicaciones debe ser capaz de cambiar el estado de la aplicación en función de los eventos externos al ámbito de ejecución de la aplicación. En este caso, el gestor de aplicaciones interrumpiría la ejecución del *MIDlet* sin que se viese afectada la ejecución de éste y lo pasaría al estado de “Pausa”. Una vez que se termine de trabajar con el *MIDlet*, éste pasa al estado de “Destruido” dónde es eliminado de la memoria volatil del dispositivo. Una vez finalizada la ejecución del *MIDlet* se pude volver a invocarlo las veces que sea ya que éste permanece en la zona de memoria persistente hasta el momento de desinstalarlo.

ESTRUCTURA DE LOS MIDLETS

Los *MIDlets*, al igual que los *applets* carecen de la método main() y aunque existiese, el gestor de aplicaciones la ignoraría por completo. Un *MIDlet* tampoco puede realizar una llamada a System.exit(). Una llamada a este método lanzaría la excepción SecurityException.

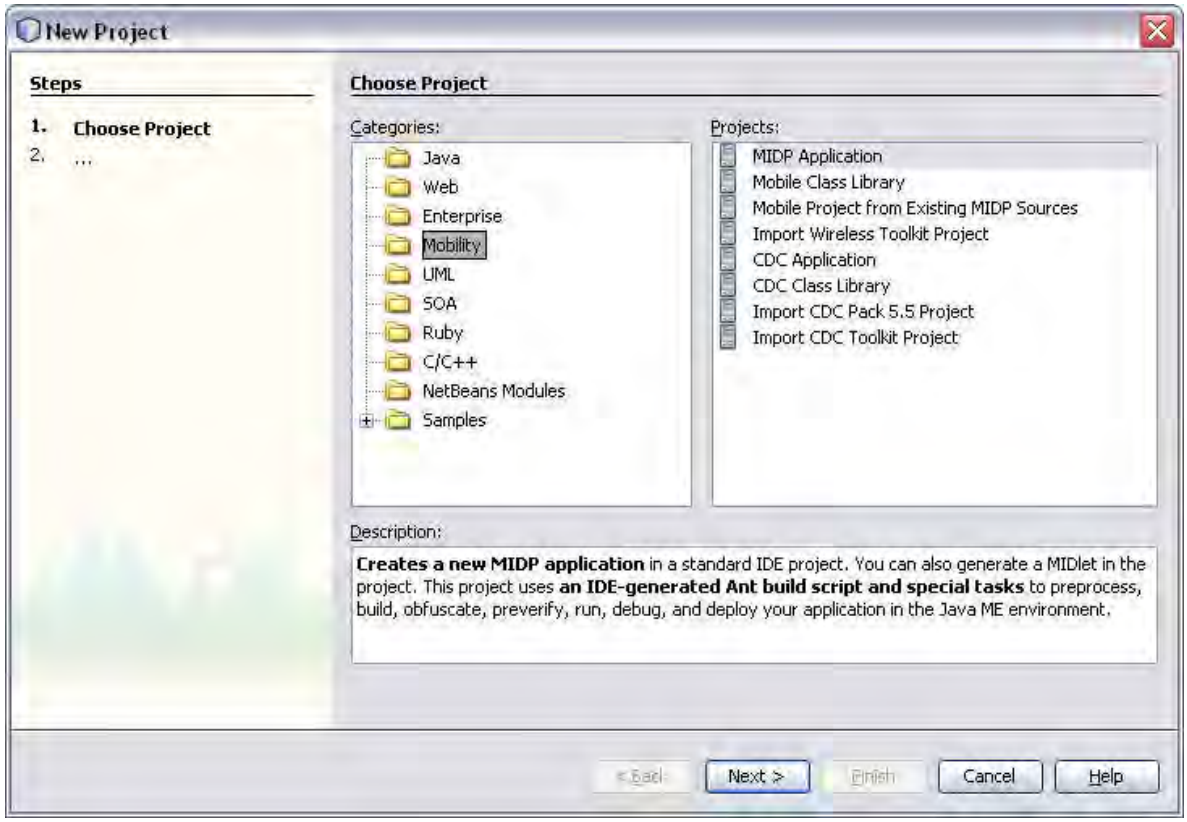
```
import javax.microedition.midlet.*
public class MiMidlet extends MIDlet{
    public MiMidlet() {
        /* Éste es el constructor de clase. Aquí debemos
        inicializar nuestras variables.*/
    }
    public startApp(){
        /* Aquí incluiremos el código que queremos que el
        MIDlet ejecute cuándo se active.*/
    }
    public pauseApp(){
        /* Aquí incluiremos el código que queremos que el
        MIDlet ejecute cuándo entre en el estado de pausa
        (Opcional)*/
    }
    public destroyApp(){
        /* Aquí incluiremos el código que queremos que el
        MIDlet ejecute cuándo sea destruido. Normalmente
        aquí se liberaran los recursos ocupados por el
        MIDlet como memoria, etc. (Opcional)*/
    }
}
```

A continuación se detalla algunos de los métodos disponibles:

Métodos	Descripción
void addComand(Command cmd)	Añade el Command *cmd*.
int getHeight()	Devuelve el alto de la pantalla.
Ticker getTicker()	Devuelve el *Ticker* (cadena de texto que se desplaza) asignado a la pantalla.
String getTitle()	Devuelve el título de la pantalla.
int getWidth()	Devuelve el ancho de la pantalla.
bolean isShown()	Devuelve *true* si la pantalla está activa.
void removeCommand(Command cmd)	Elimina el Command *cmd*.
void setCommandListener(CommandListener l)	Establece un *listener* para la captura de eventos.
void setTicker(Ticker ticker)	Establece un *Ticker* a la pantalla.
void setTitle(String s)	Establece un título a la pantalla.
protected void sizeChanged(int w, int h)	El AMS llama a este método cuándo el área disponible para el objeto Displayable es modificada.

Ejemplo No. 1:

Desarrollar una aplicación en Java que presente en la pantalla de un celular un mensaje.




```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class EJEMPLO extends MIDlet {
    Form    formulario;
    Display pantalla;
    public void startApp() {
        formulario      = new Form("ESPE");
        String mensaje = "TECNOLOGIAS DE SOFTWARRE";
        formulario.append(mensaje);
        pantalla = Display.getDisplay(this);
        pantalla.setCurrent(formulario);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

Ejemplo No. 2:

Desarrollar una aplicación que permita al usuario ingresar su nombre y su apellido, en campos de texto distintos y que el resultado de unir estos dos se presente en un tercer campo.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ejemplo extends MIDlet implements ItemStateListener,CommandListener{
    Display   pantalla;
    Form      formulario;
    TextField txt;
    TextField txt1;
    TextField txt2;
    Command   salir;
    Command   accion;
    Command   limpiar;
    public void startApp() {
        pantalla = Display.getDisplay(this);
        formulario = new Form("ESPE");
        txt = new TextField("Nombre:","",70,TextField.ANY);
        formulario.append(txt);
        txt1 = new TextField("Apellido:","",70,TextField.ANY);
        formulario.append(txt1);
        txt2 = new TextField("Resultado","",70,TextField.ANY);
        formulario.append(txt2);
        accion = new Command("UNIR", Command.SCREEN, 1);
        formulario.addCommand(accion);
        limpiar = new Command("limpiar", Command.SCREEN, 2);
        formulario.addCommand(limpiar);
        salir = new Command("Salir",Command.EXIT,1);
        formulario.addCommand(salir);
        formulario.setItemStateListener(this);
        formulario.setCommandListener(this);
        pantalla.setCurrent(formulario);
    }

    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
    public void commandAction(Command c, Displayable d){
        if (c == salir){
            destroyApp(false);
            notifyDestroyed();
        }
        if (c == accion){
            txt2.setString(txt.getString()+" "+txt1.getString());
          }
        if (c == limpiar){
            txt.setString("");
            txt1.setString("");
            txt2.setString("");
        }
    }
    public void itemStateChanged(Item i){
        if (i == txt){
            System.out.println(txt.getString());
        }
        if (i == txt1){
            System.out.println(txt1.getString());
        }
    }
}
```

Ejemplo No. 3:

Desarrollar una aplicación que permita al usuario ingresar dos números , con los cuales se realizará las cuatro operaciones básicas y que sus resultados se presenten en pantalla.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class EJEMPLO2 extends MIDlet implements ItemStateListener,CommandListener{
    Display   pantalla;
    Form      formulario;
    TextField txt;
    TextField txt1;
    TextField txt2;
    Command   salir;
    Command   accion;
    double a=0;
    double b=0;
    public void startApp() {
        pantalla = Display.getDisplay(this);
        formulario = new Form("ESPE");
        txt = new TextField("Dato1","",70,TextField.ANY);
        formulario.append(txt);
        txt1 = new TextField("Dato2","",70,TextField.ANY);
        formulario.append(txt1);
        accion = new Command("CALCULAR", Command.SCREEN, 1);
        formulario.addCommand(accion);
        salir = new Command("Salir",Command.EXIT,1);
        formulario.addCommand(salir);
        formulario.setItemStateListener(this);
        formulario.setCommandListener(this);
        pantalla.setCurrent(formulario);
    }
    public void pauseApp() {
    }
```

```
    public void destroyApp(boolean unconditional) {
    }
    public void commandAction(Command c, Displayable d){
        if (c == salir){
            destroyApp(false);
            notifyDestroyed();
        }
        if (c == accion){
            double x1=a+b;
            String mensaje = "Suma="+String.valueOf(x1)+"\n";
            formulario.append(mensaje);
            double x2=a-b;
            mensaje = "Resta="+String.valueOf(x2)+"\n";
            formulario.append(mensaje);
            double x3=a*b;
            mensaje = "Multiplicacion="+String.valueOf(x3)+"\n";
            formulario.append(mensaje);
            double x4=a/b;
            mensaje = "Division="+String.valueOf(x4)+"\n";
            formulario.append(mensaje);
        }
    }
    public void itemStateChanged(Item i){
        if (i == txt){
            System.out.println(txt.getString());
            a=Integer.parseInt(txt.getString());
        }
        if (i == txt1){
            System.out.println(txt1.getString());
            b=Integer.parseInt(txt1.getString());
        }
    }
}
```

Ejemplo No. 4:

Desarrollar una aplicación que despliege una pantalla con una lista, un grupo de radio buttons y varios check box.

Tipo	Descripción
EXCLUSIVE	Lista en la que un sólo elemento puede ser seleccionado a la vez.
IMPLICIT	Lista en la que la selección de un elemento provoca un evento.
MULTIPLE	Lista en la que cualquier número de elementos pueden ser seleccionados al mismo tiempo.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Midlet extends MIDlet implements CommandListener{
    Command atras, salir; //Declaracion de Variables
    Display pantalla;
    List menu;
    Form formu1, formu2, formu3;
    public void startApp() {
        pantalla = Display.getDisplay(this); //Creacion de pantallas
        menu = new List("Menú",List.IMPLICIT);
        menu.insert(0,"Opcion3",null);
        menu.insert(0,"Opcion2",null);
        menu.insert(0,"Opcion1",null);
        salir = new Command("Salir",Command.EXIT,1);
        menu.addCommand(salir);
        formu1 = new Form("Formulario 1");
        formu2 = new Form("Formulario 2");
        formu3 = new Form("Formulario 3");
        menu.setCommandListener(this);
        pantalla.setCurrent(menu); // Pongo el menu en pantalla
    }


    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
    public void commandAction(Command c, Displayable d){
        if (c == menu.SELECT_COMMAND){ // Si selecciono
            switch(menu.getSelectedIndex()){ //opcion del menu
                case 0:{ pantalla.setCurrent(formu1);}
                case 1:{ pantalla.setCurrent(formu2);}
            case 2:{ pantalla.setCurrent(formu3);}
            }
        }
        if (c == salir){ // Selecciono salir de la aplicacion
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

CAPITULO 14

INGENIERIA INVERSA EN JAVA

La Ingeniería Inversa es el proceso de construir especificaciones de un mayor nivel de abstracción partiendo de un código fuente compilado. Existen cuatro tipos de ingeniería inversa:

- De datos.- Se aplica sobre algún Código de Base de datos.
- Lógica o proceso.- Se aplica sobre el código de un programa para averiguar su lógica.
- Interfaces de usuario.- Se aplica para obtener los modelos y especificaciones que sirvieron de base para la especificación de la interfaz.
- Protocolo de comunicación.- Se aplica para analizar el comportamiento de la comunicación, independiente del medio físico.

DECOMPILACION

Es la acción que realiza un descompilador o decompilador el cual opera a manera inversa a un compilador; es decir traducir código o información de bajo nivel de abstracción (diseñado para ser leído por un ordenador) a un lenguaje o medio de mayor nivel de abstracción (diseñado para ser leído por un humano).

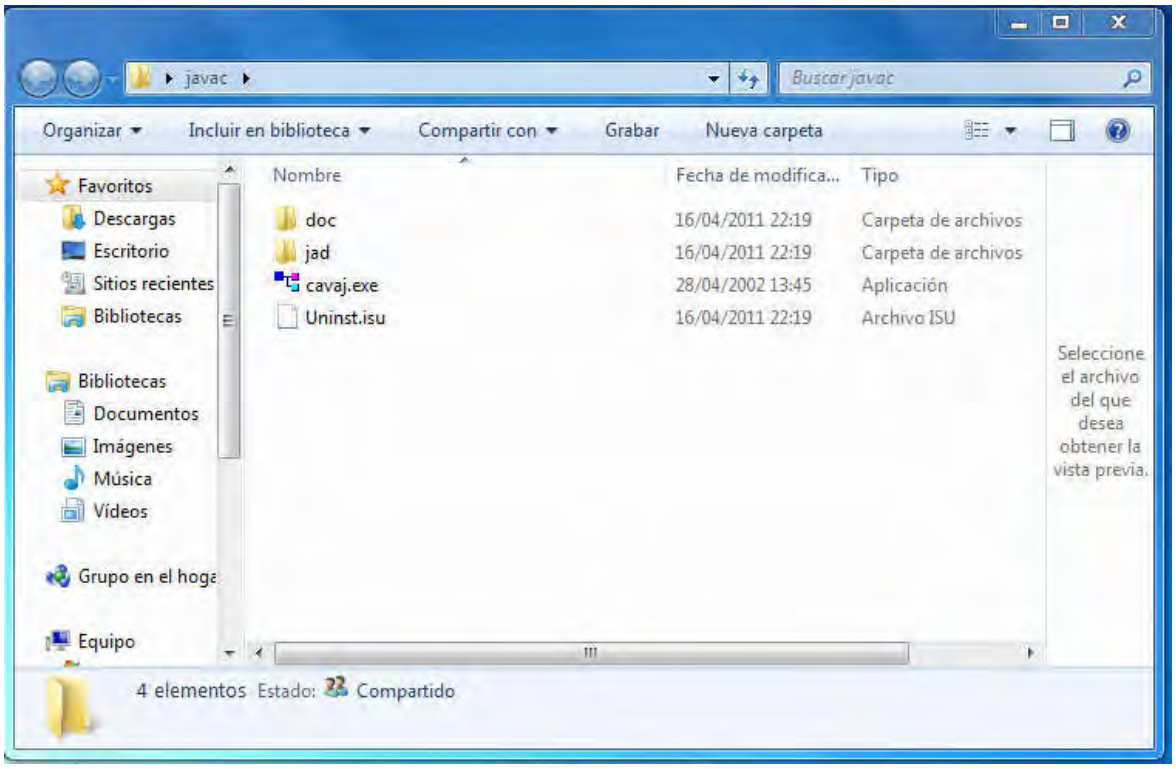
A continuación se ilustra los objetivos para descompilar:

Existen diferentes tipos de descompiladores para distintos lenguajes, cada uno de ellos especializado en los binarios o bytecodes que son generados.

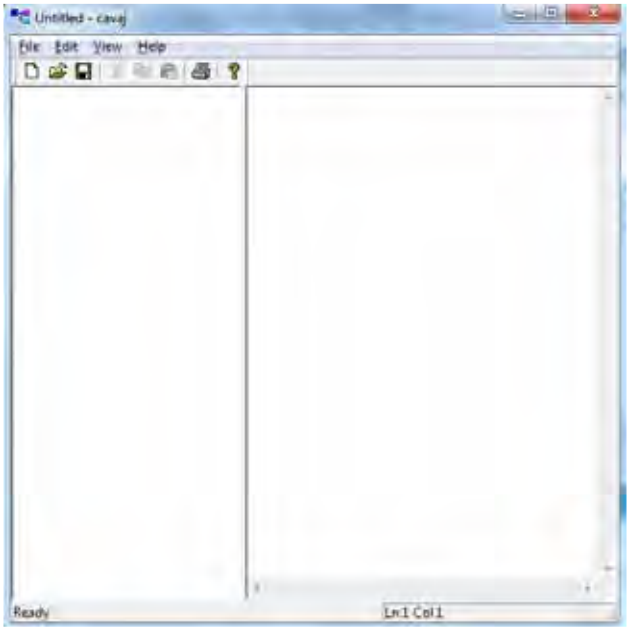
Como se ha dicho cada lenguaje tiene su descompilador, pero lo que para el caso interesa analizar son los descompiladores de Java, para sorpresa son muchos, unos gratuitos y otros pagados.

En Internet existen diversos programas descompiladores, inclusive *java* trae un comando para descompilar, que es el *javap***.** Para ilustrar este proceso se utilizará la aplicación *Cavaj* con la cual se obtendrá el código deseado en leguaje de alto nivel.

Una vez instalada dicha aplicación se procede a abrir la carpeta *javac*, creada después de la instalación, y se ejecuta el ícono *cavaj.exe*.

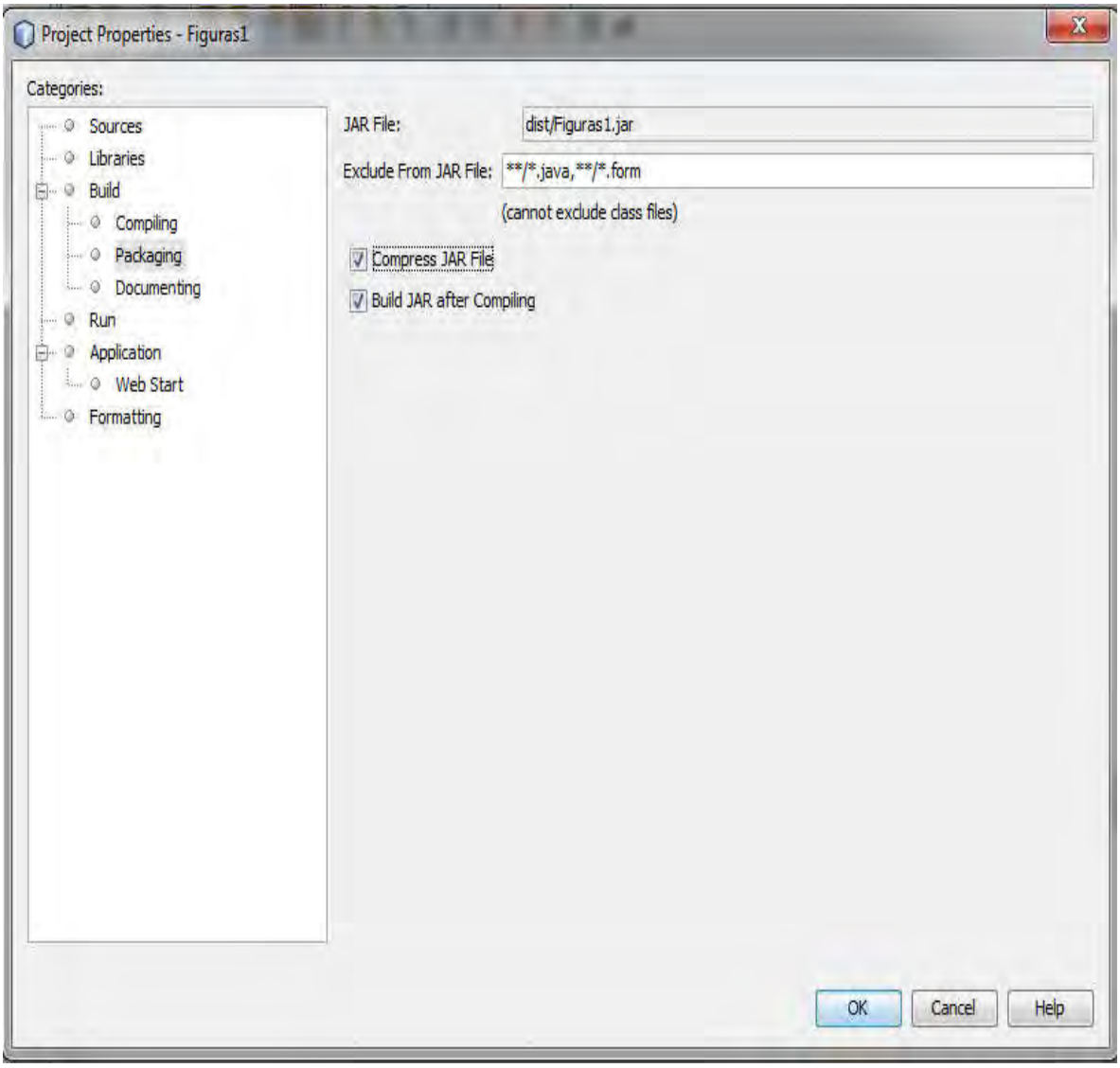


Se despliega la ventana *Untitled – cavaj*:

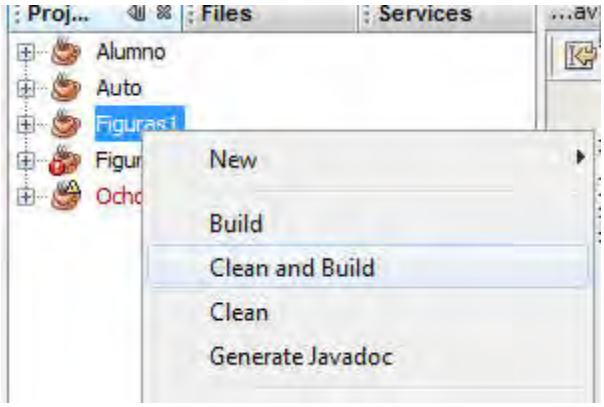


Para empezar se necesita crear el archivo *.jar* de algún proyecto realizado, por ejemplo de Figuras1

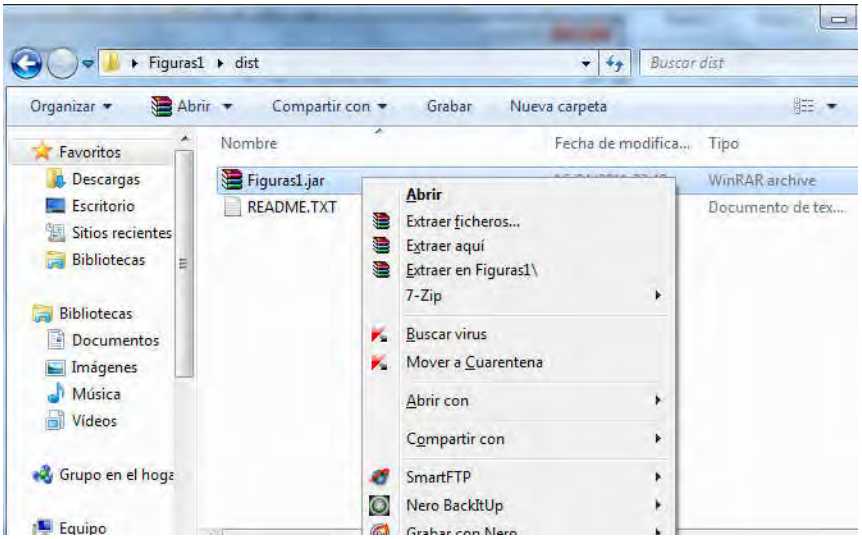
Click derecho y se selecciona PROPIEDADES
Una vez ahí seleccione PACKAGING
Y dar un *click* en COMPRESS *JAR* FILE



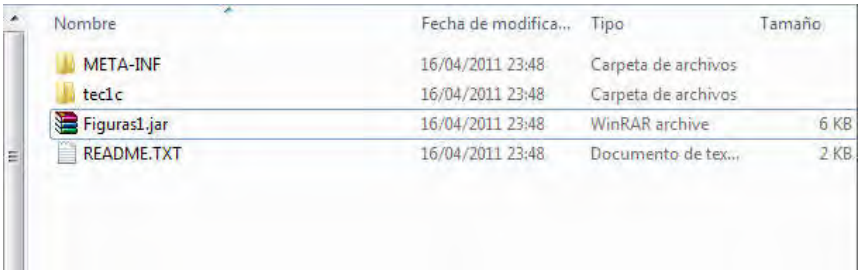
Una vez realizado esto, se procede a dar *click* derecho y *CLEAN AND BUILD*



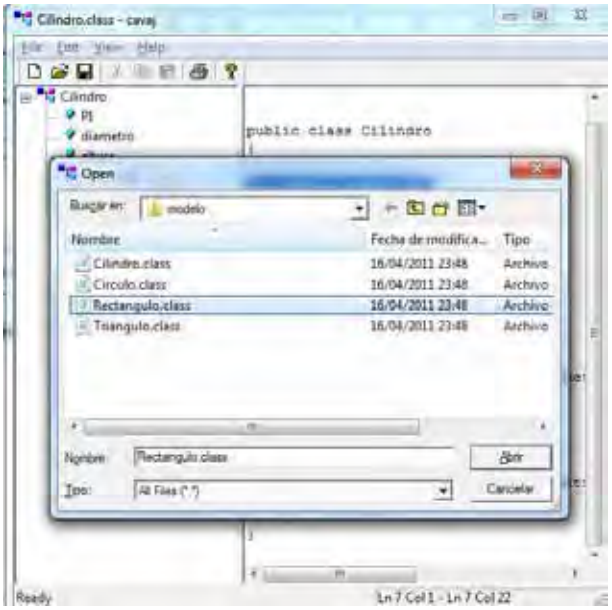
Creado el archivo.*jar*, ubicarlo y click derecho para proceder a descomprimirlo para su descompilación:



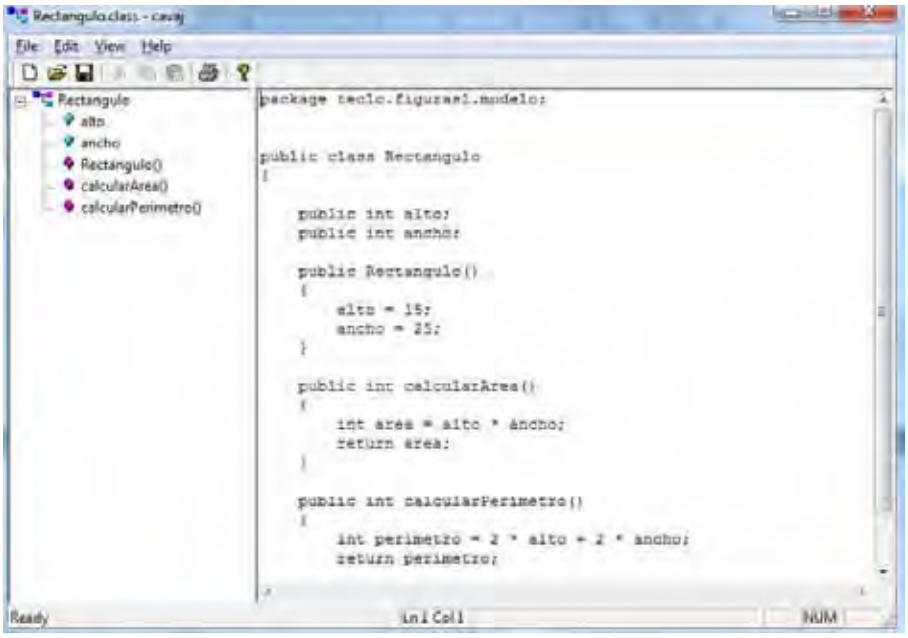
Una vez extraídos se obtiene los archivos .*jar* a descompilar:



Ahora en *Untitled – cavaj*, Clicken *File* y se ubican las clases a descompilar:



Clicken Abrir y se obtiene el código:



OFUSCACIÓN DE CÓDIGO

Es el acto deliberado de realizar un cambio no destructivo en el código fuente de un programa o código máquina cuando el programa está en forma compilada o binaria; en el cual se convierte el programa en otro equivalente de forma que no se consiga código útil al descompilar; con el fin de que no sea fácil de entender o leer, evitando la violación de la propiedad intelectual.

Se debe tener presente que ninguna tecnología de ofuscación es segura al cien por ciento.

Las formas de ofuscar el código son variadas pero se pueden resumir en las siguientes.

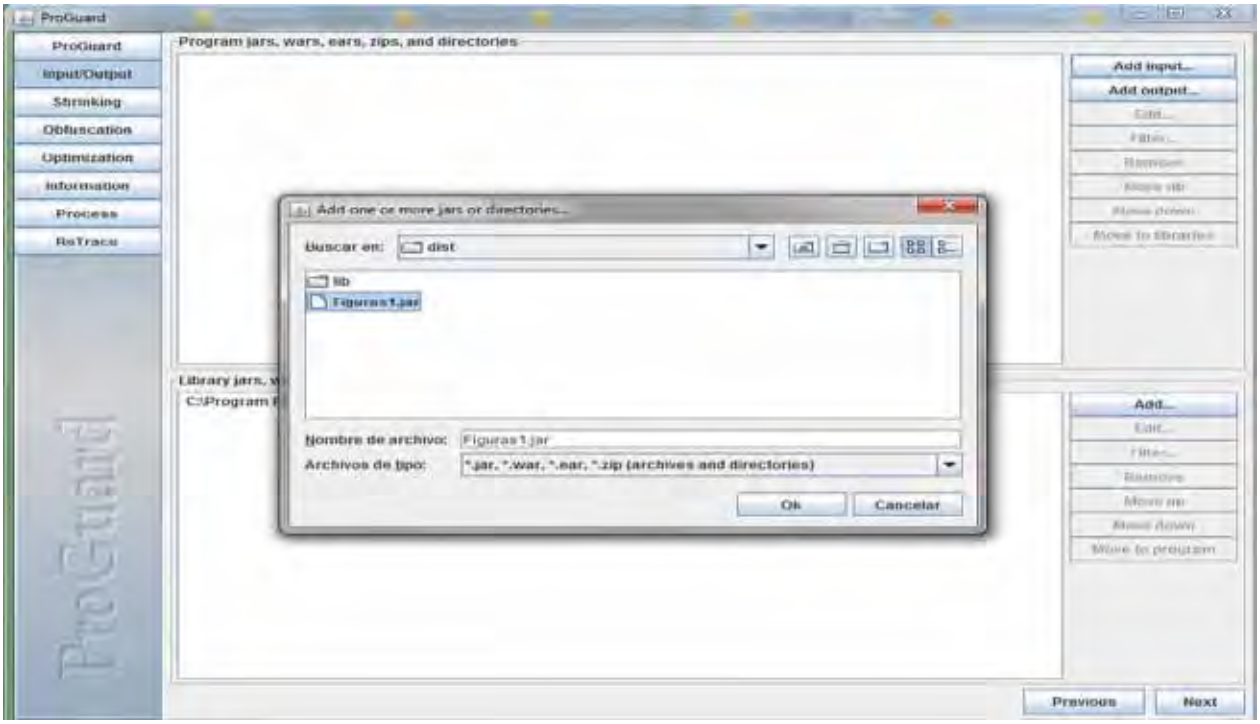
En Internet se puede encontrar diversos programas para ofuscar código, para ilustra el procedimiento se utiliza el programa *ProGuard 4.6*; el cual es un comprimidor, optimizador y ofuscador de archivos de clases de *Java* gratuito y de código abierto.

Para empezar se necesita crear el archivo *.jar* de algún proyecto realizado, por ejemplo de Figuras1, de la misma manera como se realizó antes.

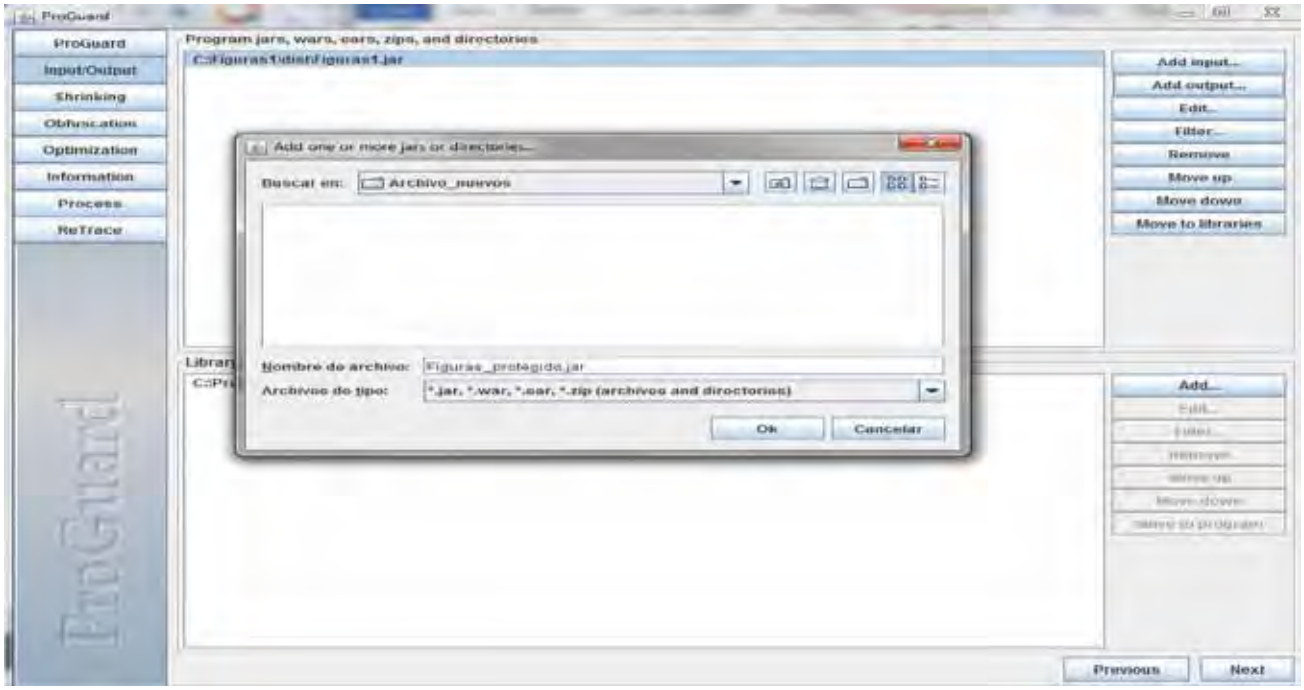
A continuación ejecute el *ProGuard* de la siguiente manera:

En la carpeta *bin* dentro de *ProGuard* dé doble clicken *proguardgui*, se desplegará la siguiente ventana:

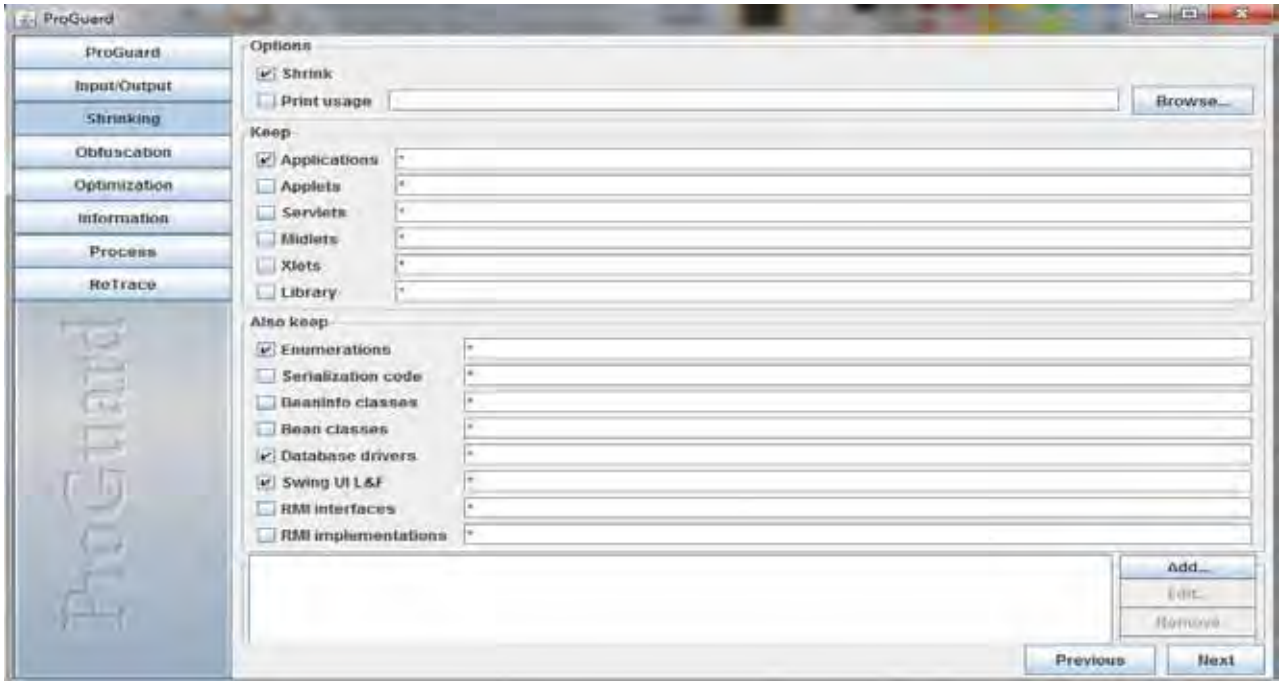
En la opción *Input/Output* se tiene la opción de seleccionar el archivo *.jar* a ofuscar en la pestaña *Add input*, se ubica el archivo *.jar* y presionar *OK*:



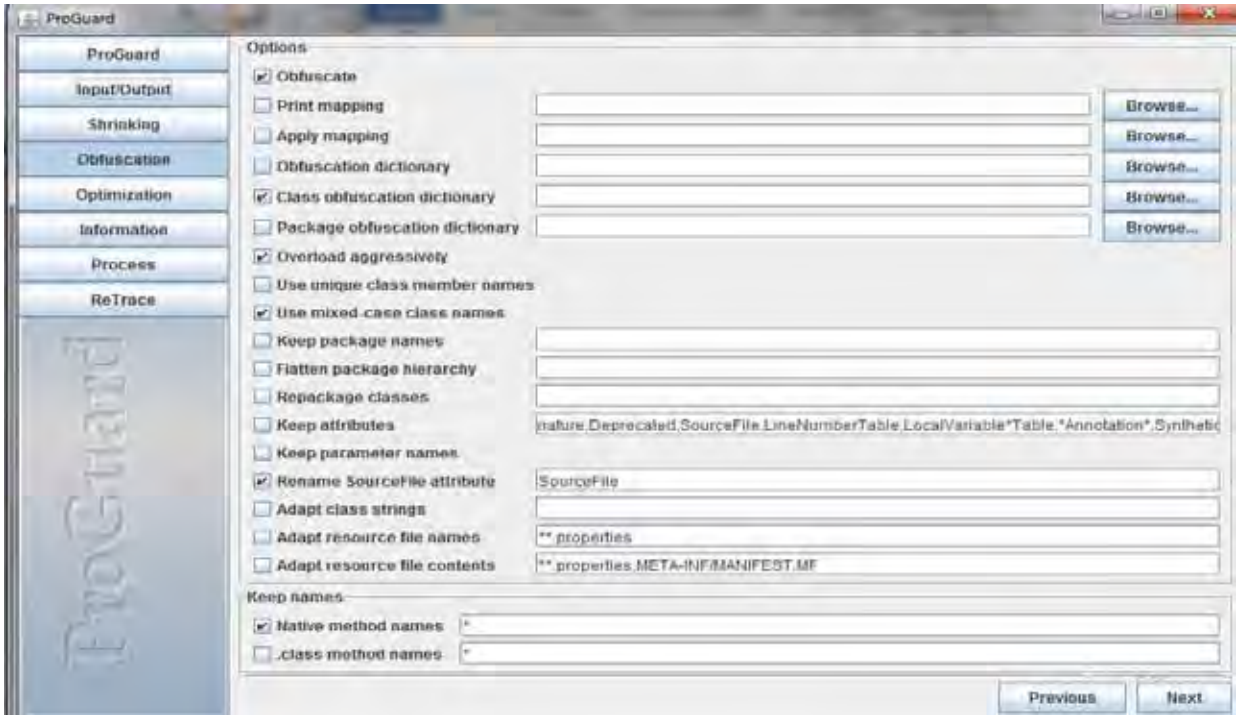
En la opción *Output* se escoge la dirección y nombre del nuevo archivo ofuscado con la extensión *.jar,* y dar clicken *OK*:



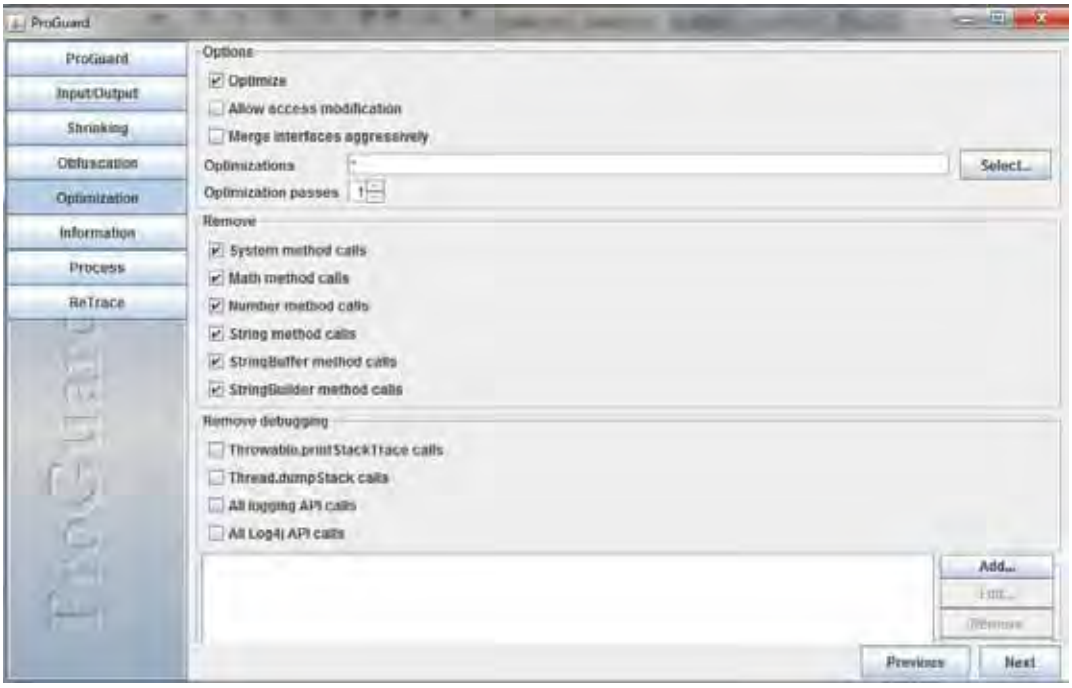
Clicken *Next* ya aparecerá la ventana *Shriking* en la cual se disminuirá el tamaño de nuestra aplicación:



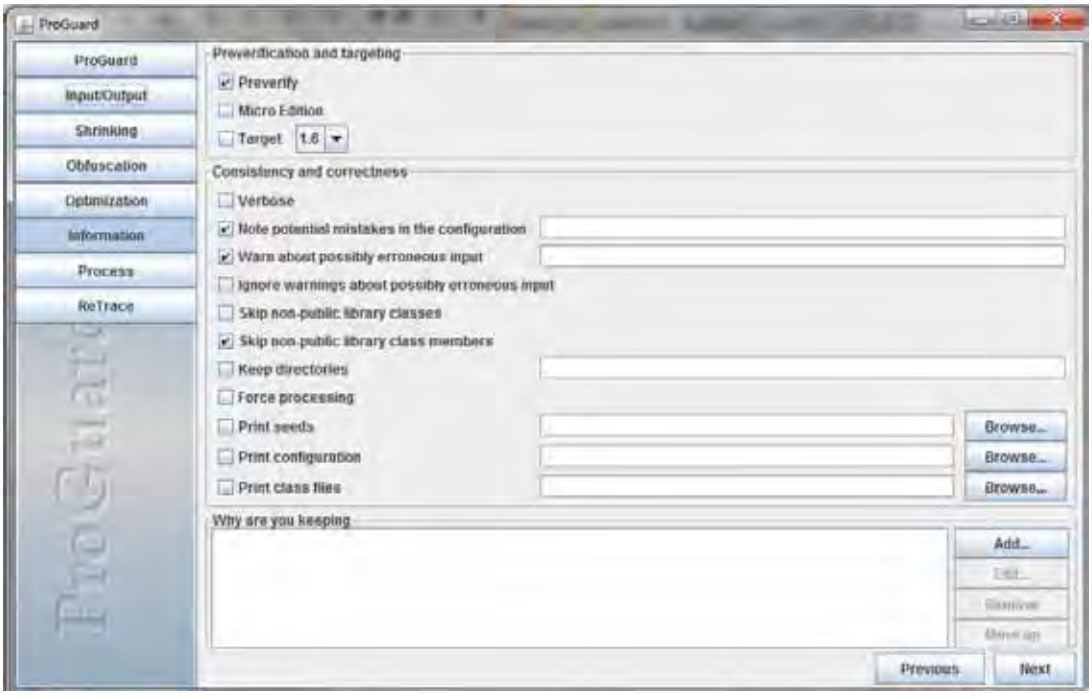
Clicken *Next* y aparecerá la ventana de *Obfuscation* (Ofuscar), la cual despliega diversas opciones para una ofuscación apropiada. Las opciones presentadas en esta ventana se encuentran detalladas en el manual de usuario presentadas en anexo.



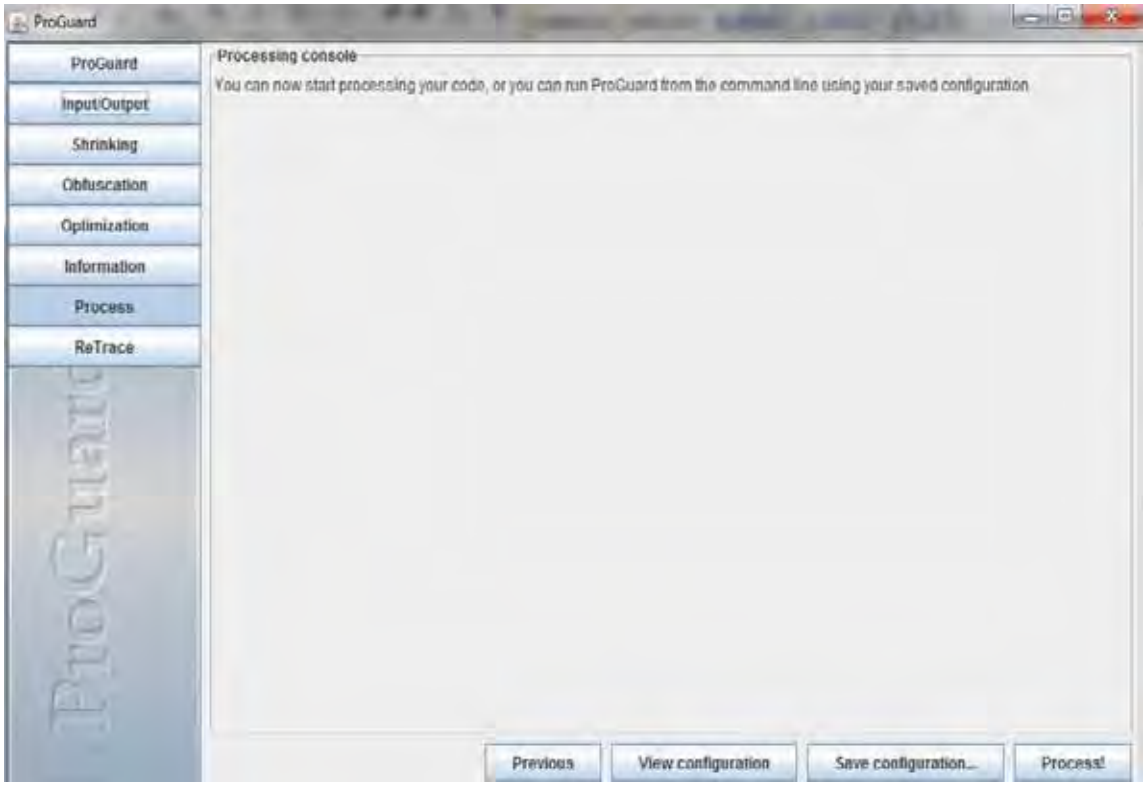
A continuación la ventana *Optimization* (optimizar), permite reducir el tamaño de la aplicación:



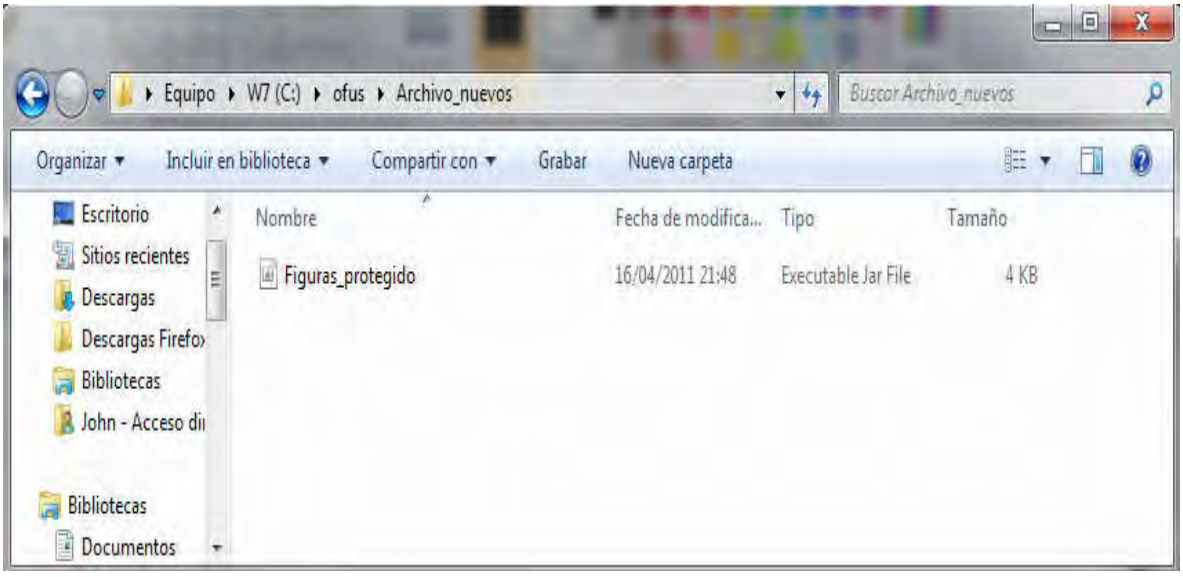
En *information* desplegará todos los datos que se desee en el proceso:



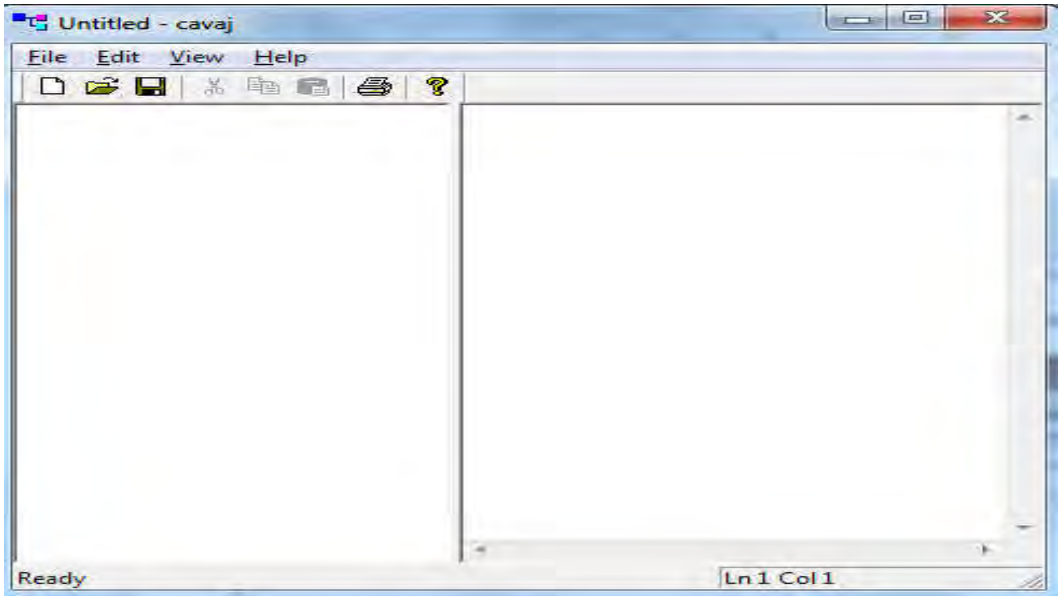
Clicken *Next* y comenzará los procesos seleccionados:



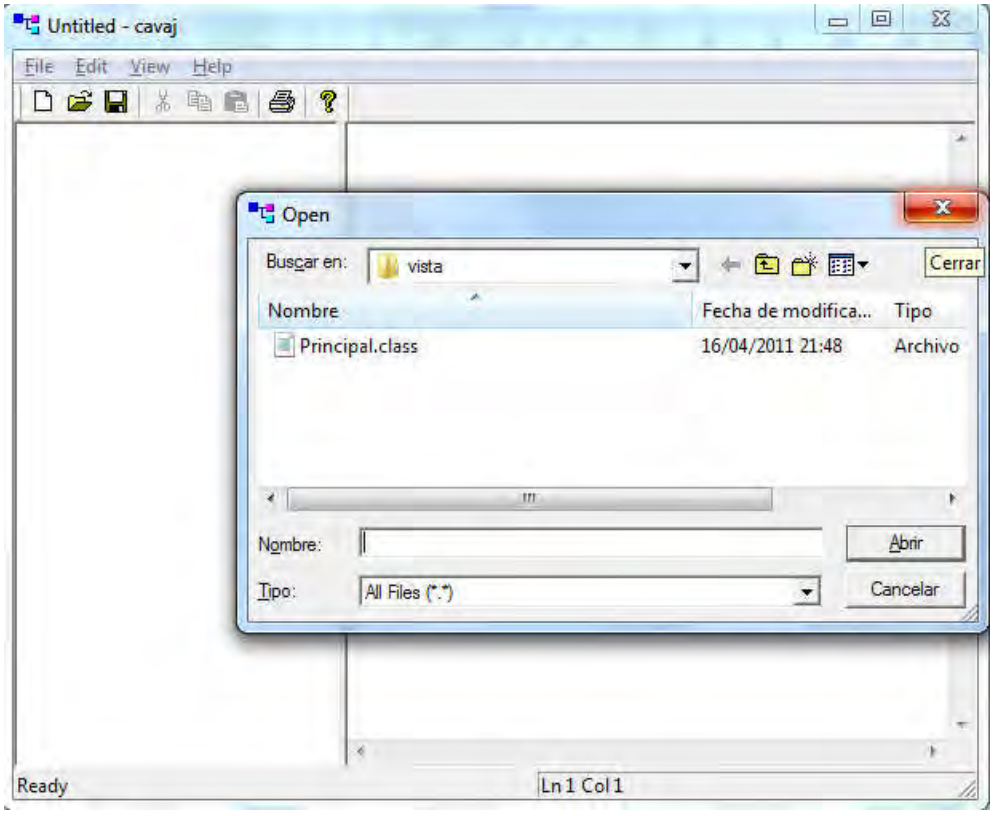
Presionar *Process!* y se crea el archivo *.jar* protegido en directorio especificado:



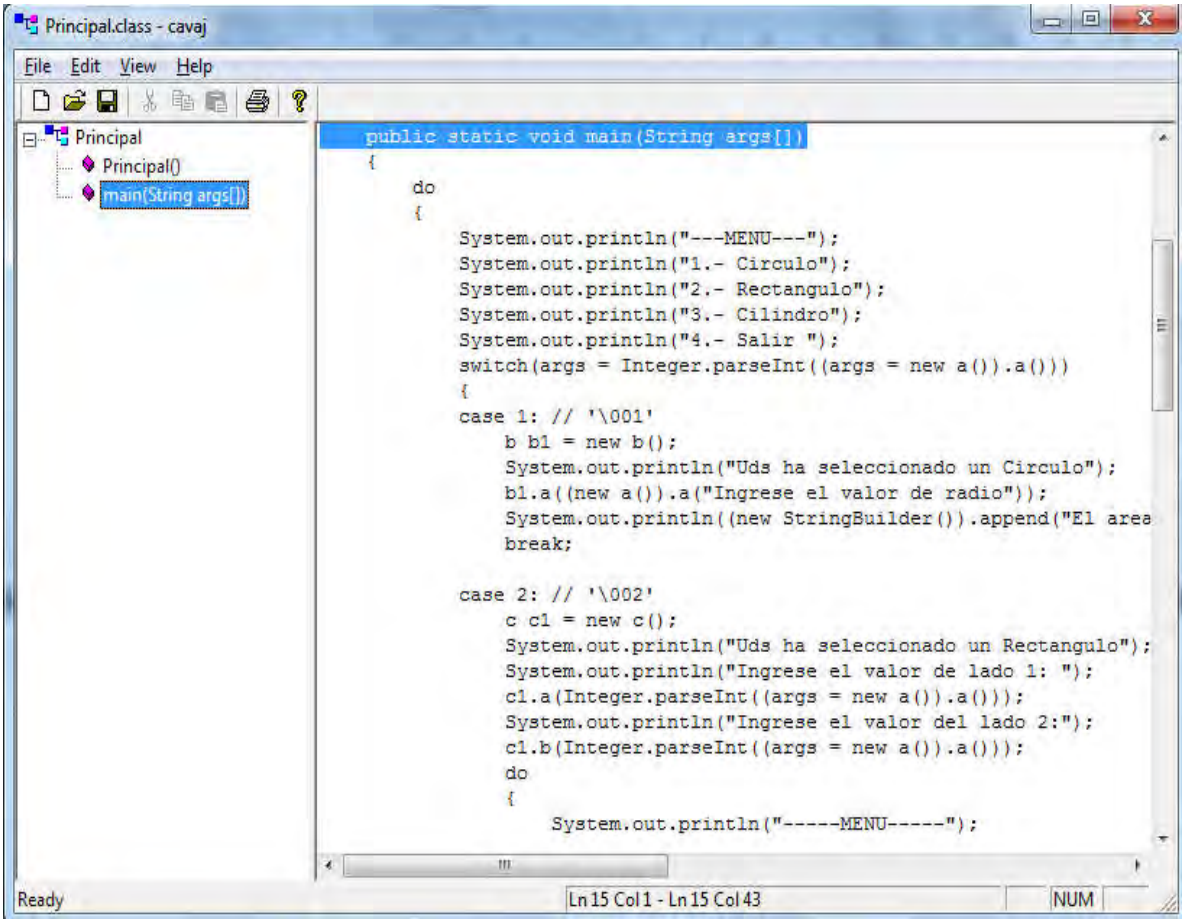
Bien, ahora tratar de decompilarlo mediante el programa *Cavaj:*



Abrir la ubicación del archivo protegido, en este caso *Figuras_protegido.jar*, descomprimir y abrir el principal:



Y se obtiene el código del archivo con cambios pero el funcionamiento es el mismo:


Principal.class - cavaj
File Edit View Help
Principal
Principal()
main(String args[])
public static void main(String args[])
{
do
{
System.out.println("---MENU---");
System.out.println("1.- Circulo");
System.out.println("2.- Rectangulo");
System.out.println("3.- Cilindro");
System.out.println("4.- Salir ");
switch(args = Integer.parseInt((args = new a()).a()))
{
case 1: // '\001'
b b1 = new b();
System.out.println("Uds ha seleccionado un Circulo");
b1.a((new a()).a("Ingrese el valor de radio"));
System.out.println((new StringBuilder()).append("El area
break;
case 2: // '\002'
c c1 = new c();
System.out.println("Uds ha seleccionado un Rectangulo");
System.out.println("Ingrese el valor de lado 1: ");
c1.a(Integer.parseInt((args = new a()).a()));
System.out.println("Ingrese el valor del lado 2:");
c1.b(Integer.parseInt((args = new a()).a()));
do
{
System.out.println("-----MENU-----");
Ready
Ln 15 Col 1 - Ln 15 Col 43
NUM

CAPITULO 15

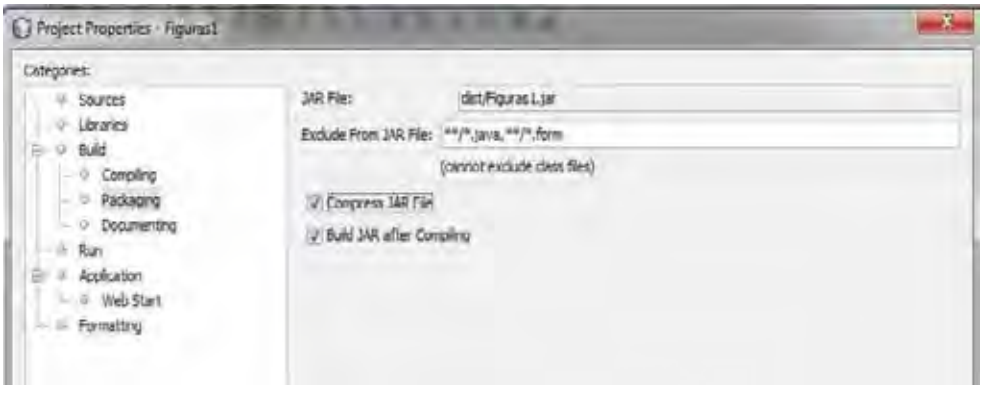
CREACIÓN DE EJECUTABLES E INSTALADORES PARA APLICACIONES JAVA EN WINDOWS

Es sabido que las aplicaciones Java son interpretadas por una JVM, a partir de los byte-codes, lo que permite que éstos puedan ejecutarse en cualquier sistema operativo que disponga de una JVM, claro está que lo se necesita es el .jar o el .class, y dependiendo de cada sistema operativo, lo podremos ejecutar, aunque esto para personas con un cierto nivel de conocimiento es bastante fácil, para la mayoría de personas que es para quienes están destinadas las aplicaciones no tiene una formación técnica y por lo general utilizan Windows; o para vender las aplicaciones desarrollas sin entregar el código fuente o el . jar, desde el cual se pueden emplear fácilmente técnicas de ingeniería inversa; la solución que se plantea es crear un .exe o directamente un instalador, aunque para algunos esto sea una herejía.

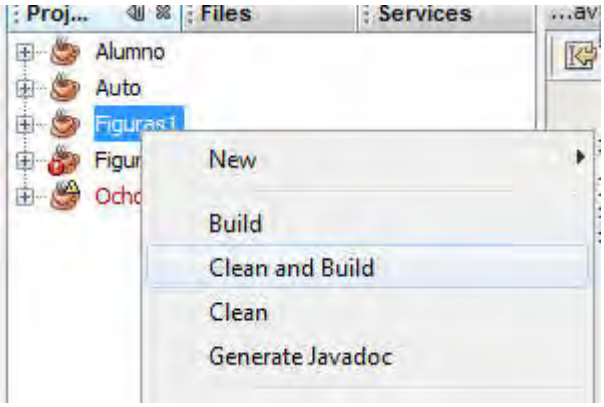
CREACION DE EJECUTABLES .EXE

Para empezar se necesita crear el archivo *.jar* de algún proyecto realizado, por ejemplo de Figuras1

Click derecho y se selecciona PROPIEDADES
Una vez ahí seleccione *PACKAGING*
Y dar un *click*en *COMPRESS JAR FILE*



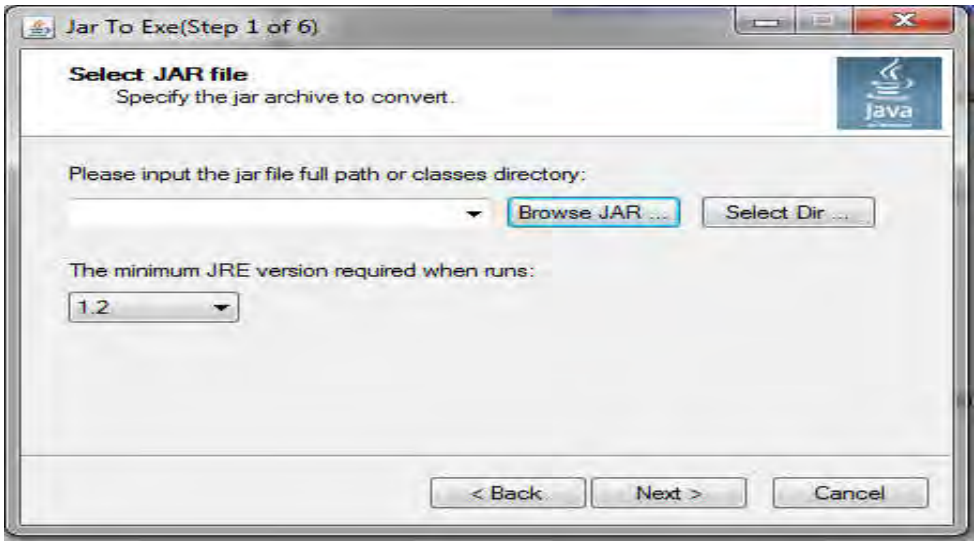
Una vez realizado esto, se procede a dar *click* derecho y *CLEAN AND BUILD*



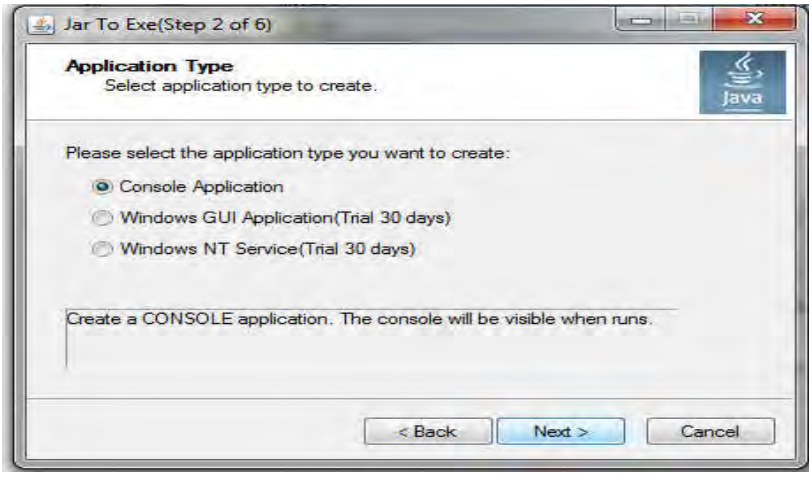
Una vez que se obtiene el .jar que para el caso se utilizó Netbeans, es necesario un programa para la creación de un punto *exe, y u*no muy fácil de encontrar en Internet es el *Jar2ExeWizard 1.8*

Una vez ejecutado, seleccione siguiente ya que es un programa pagado, pero tiene una licencia gratuita por 30 días

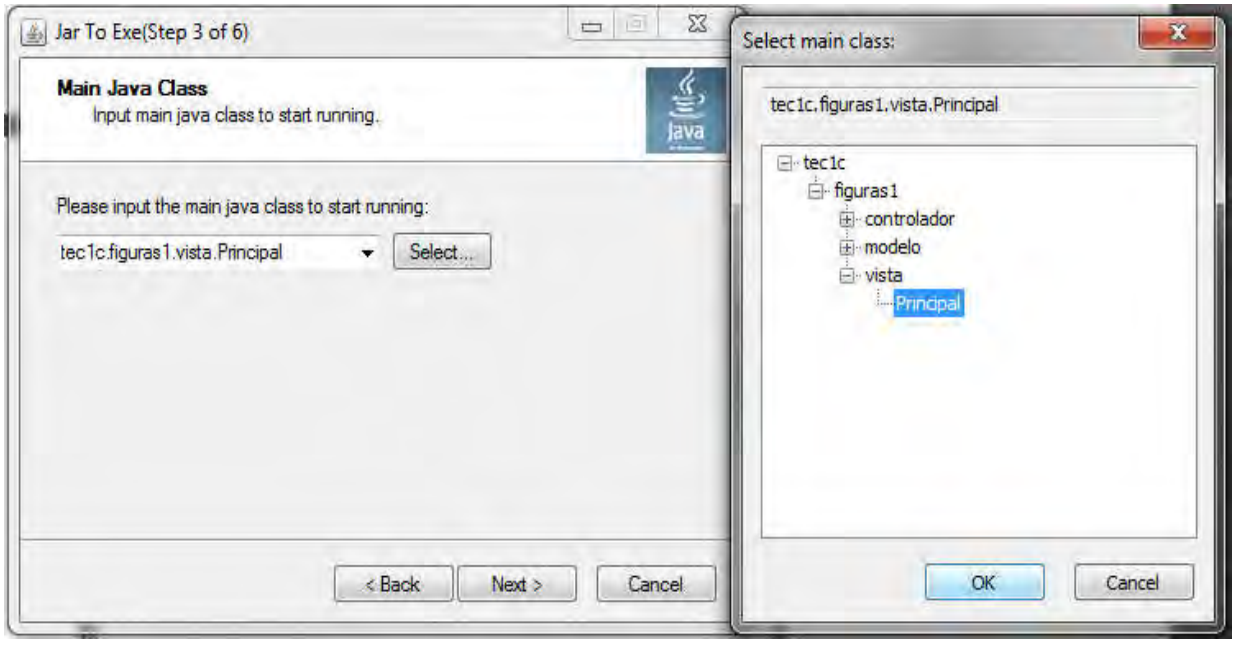
Ahora se busca la dirección donde se tiene el *.jar*



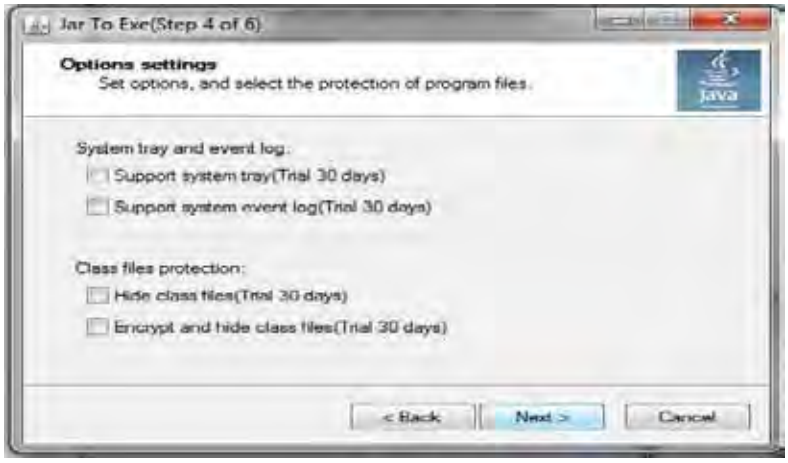
En la siguiente ventana se elige el tipo de aplicación que en este caso es *CONSOLE APLICATION*



Ahora se escoge la clase principal para correr, en este caso sólo existe una clase Principal, pero para otros programas se elige la que corresponda

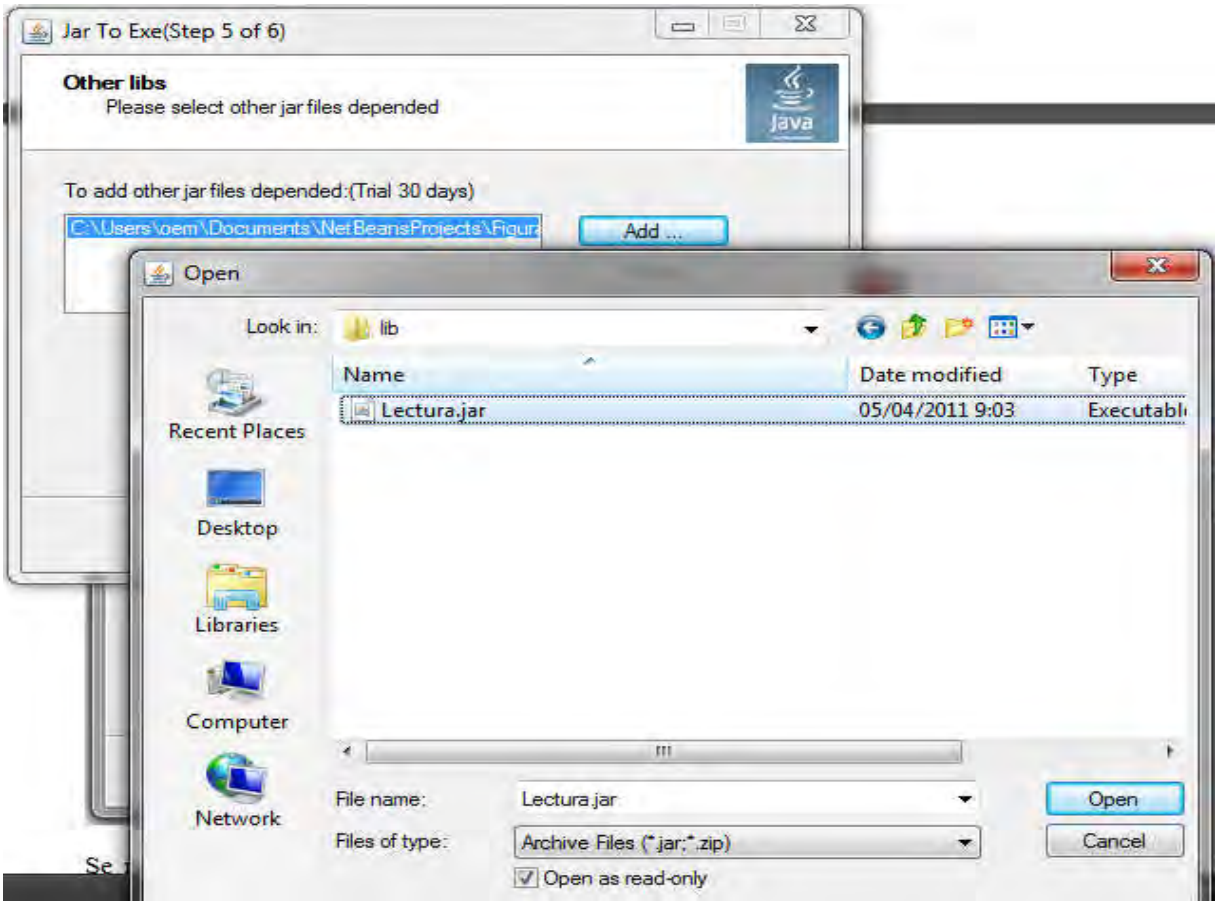


En esta ventana sólo se continúa sin seleccionar opciones

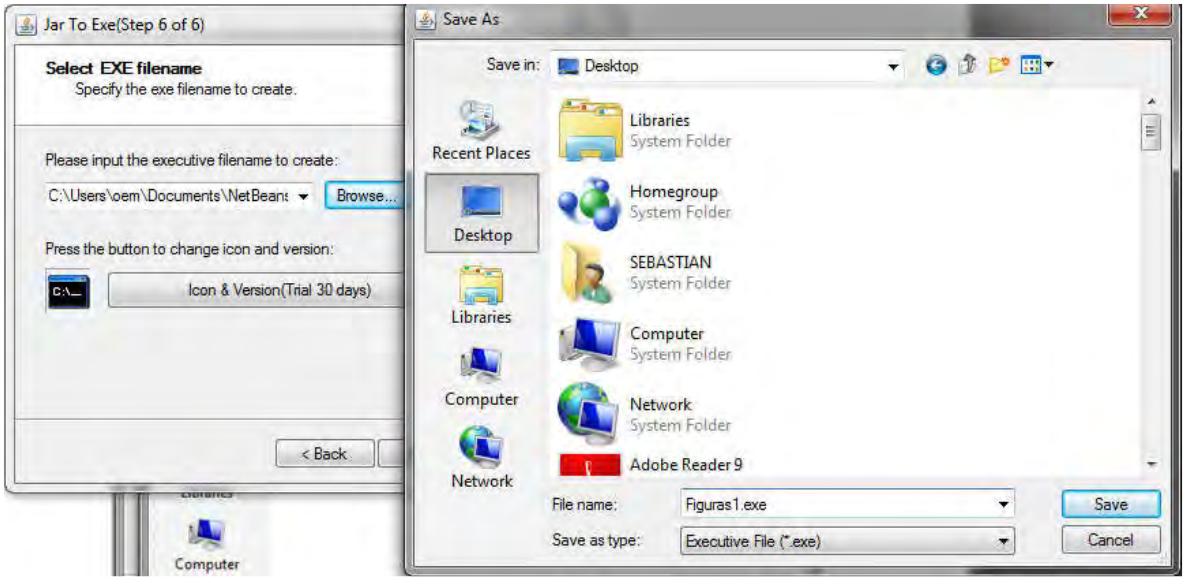


Ahora, si en caso de tener un *.jar* del que dependa el programa, se selecciona de la carpeta correspondiente.

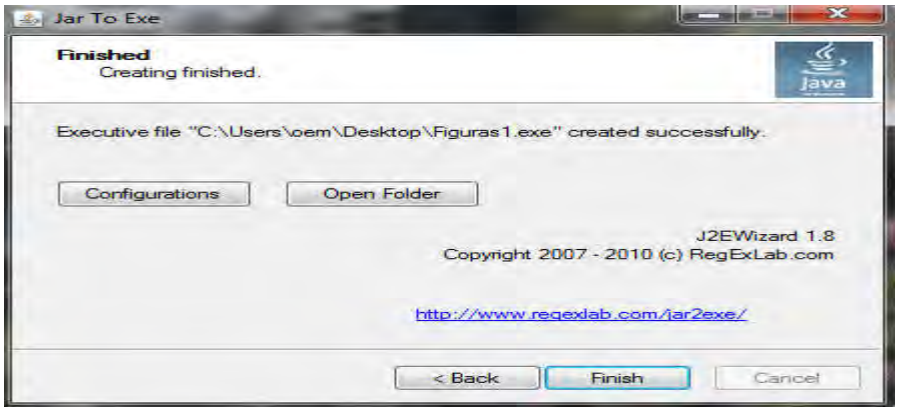
Este programa depende del *.jar* Lectura para el ingreso de datos, así que se lo selecciona.



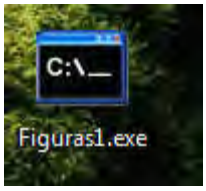
Aquí se escribe el nombre del ejecutable y además se selecciona la carpeta de destino donde queremos que se cree el *.exe*



Se despliega una ventana en la que se comprueba que ha sido creado exitosamente el archivo *.exe*



Aquí se tiene el ícono ya en el escritorio



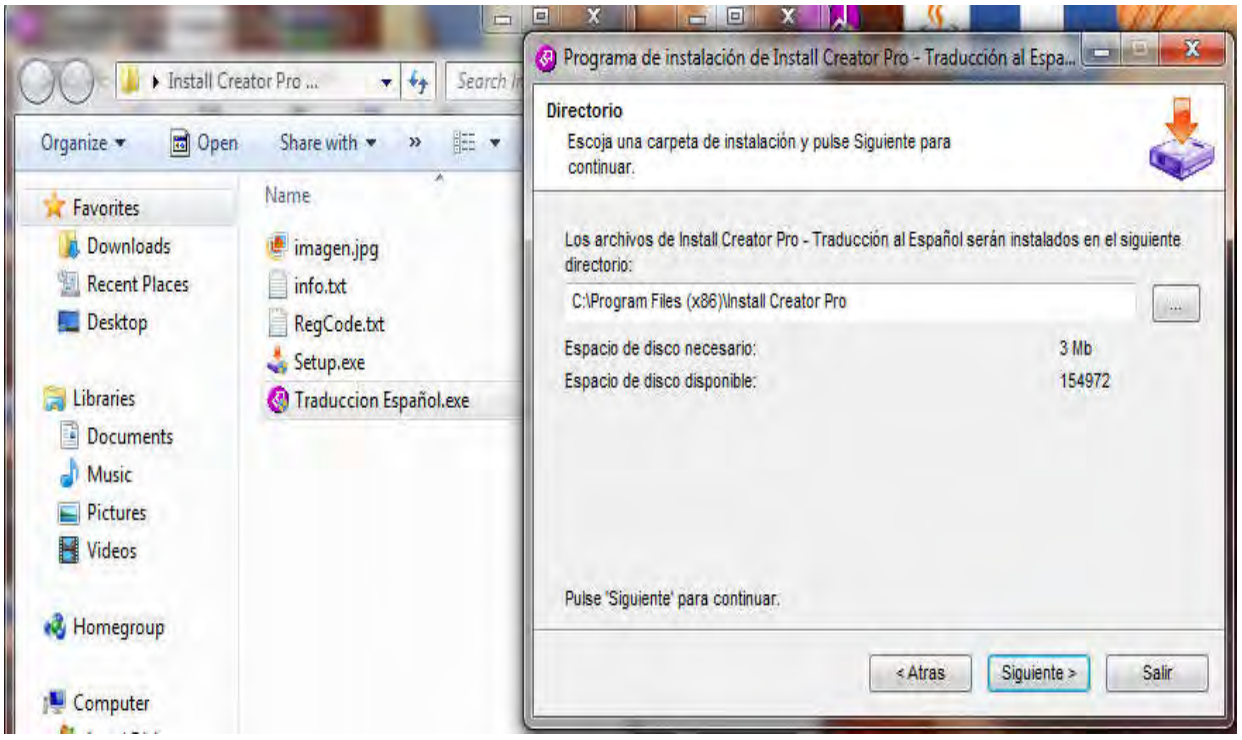
Y se comprueba que se ejecuta correctamente en consola

CREACION DE INSTALADORES

NOTA: Previamente es necesaria la creación de un archivo *.exe*

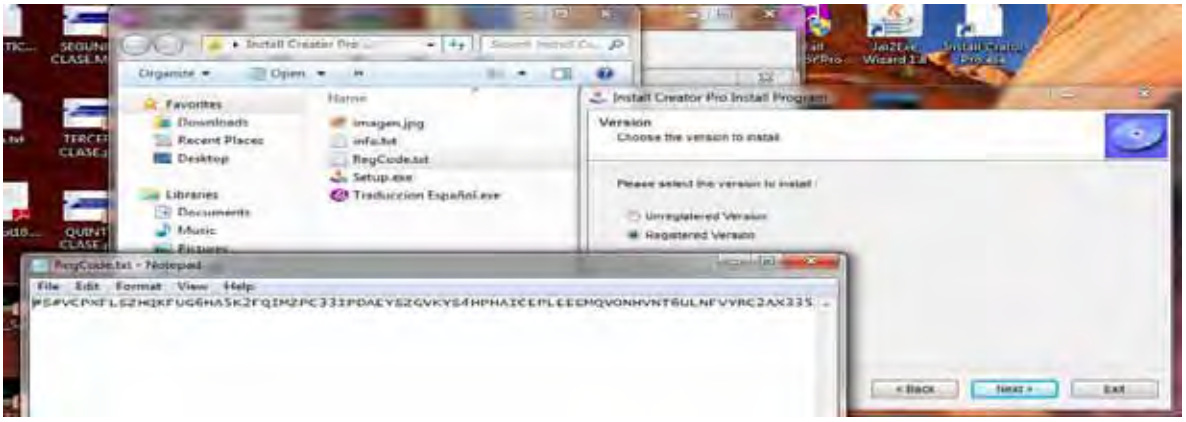
Para crear el instalador, se utilizará el programa *Install Creator Pro*

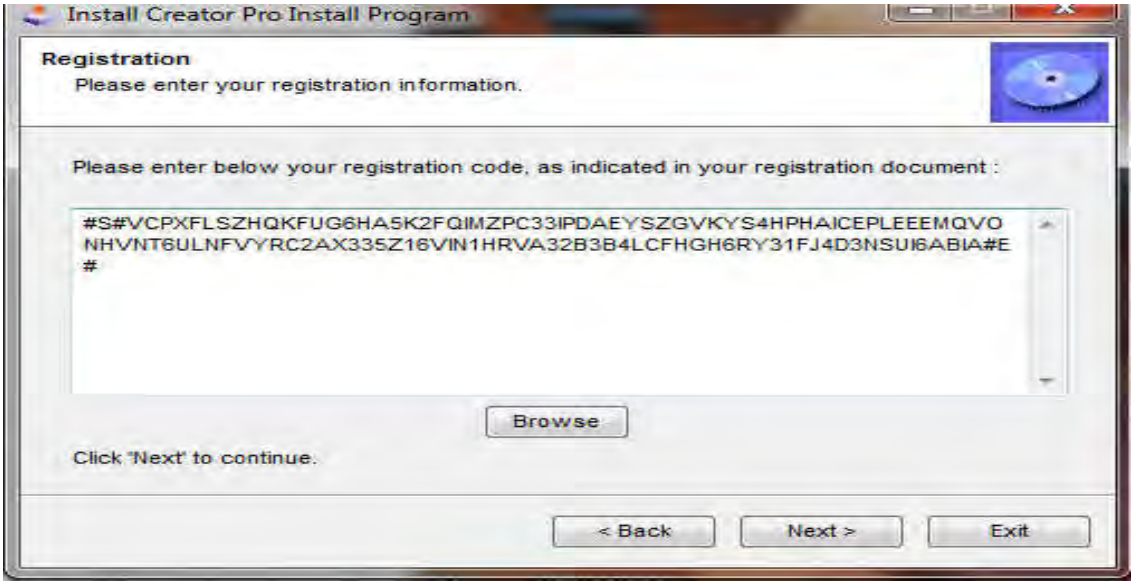
Se examina la carpeta, y se instala Traducción Español.*exe* para tener el programa en español. Este instalador es muy sencillo, sólo se sigue indicaciones de SIGUIENTE.



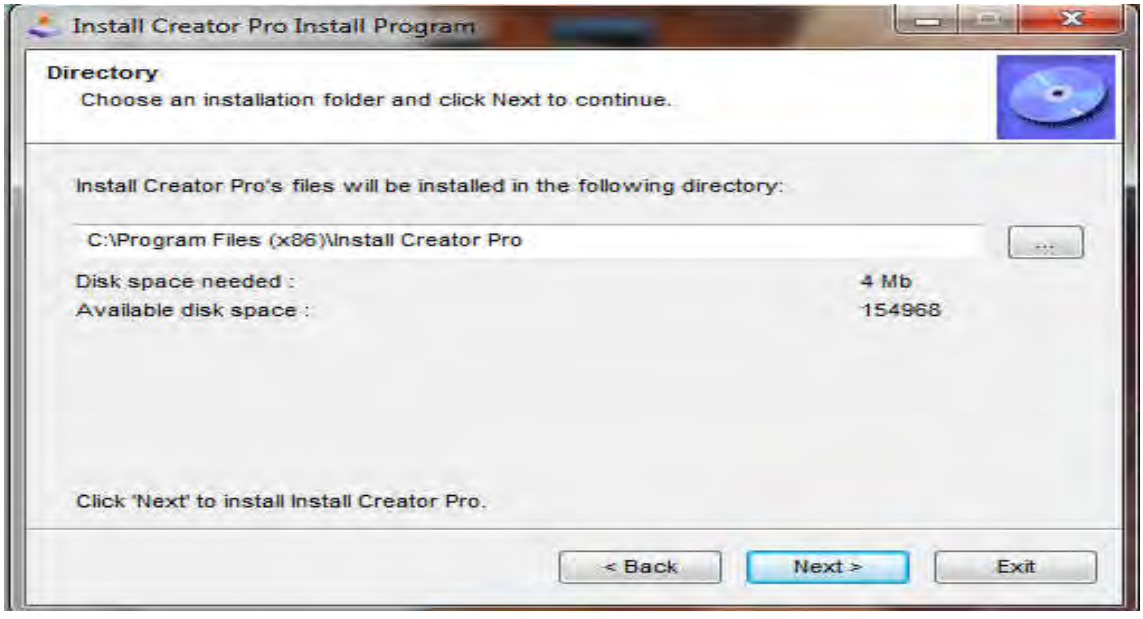
Una vez instalada la traducción en español. Se procede a registrar el programa

Se procede a copiar el código de licencia que se encuentra en la carpeta RedCode.txt



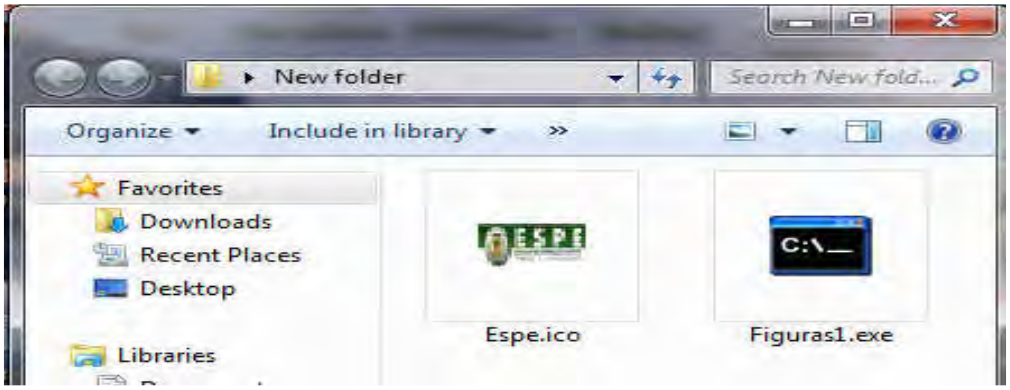


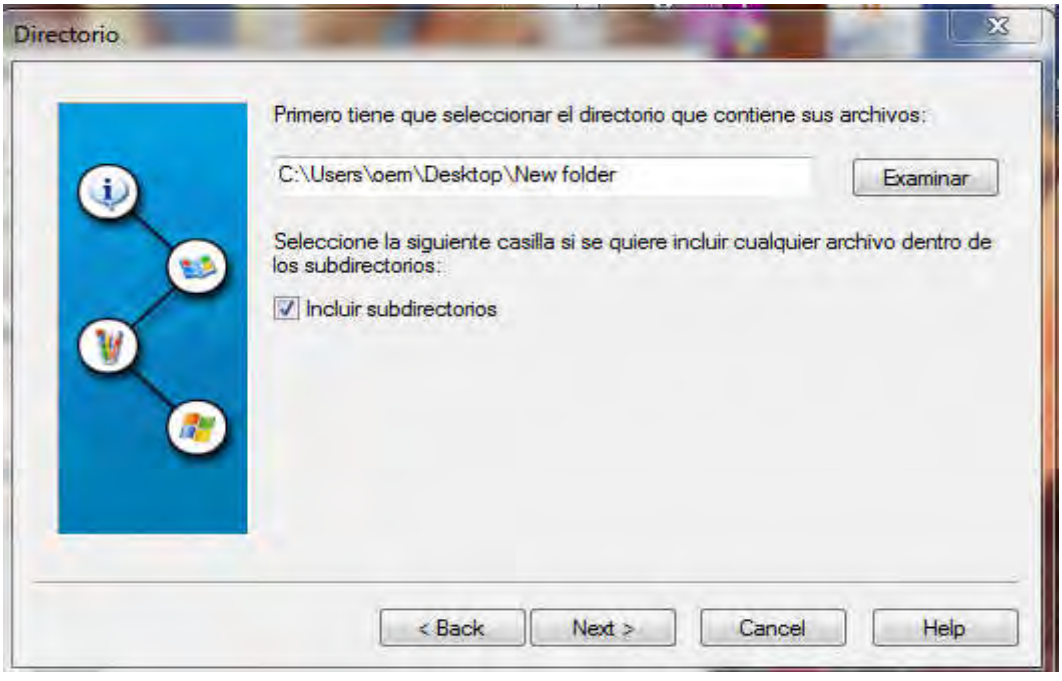
Se instala fácilmente con SIGUIENTE, creando una carpeta específica para este programa.



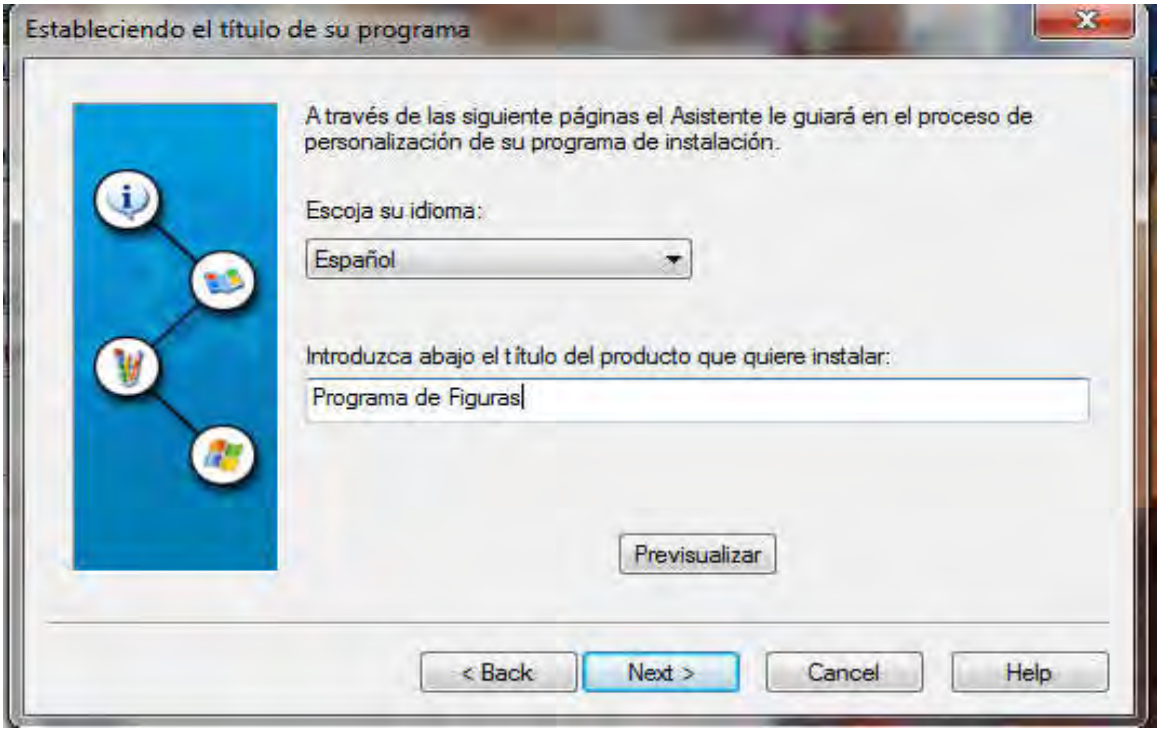
Se inicia dando *click* en NUEVO el asistente para crear instaladores

Se elige la carpeta donde se encuentre el punto *exe*, en este caso se encuentra en NEW FOLDER en Desktop (como se puede observar)

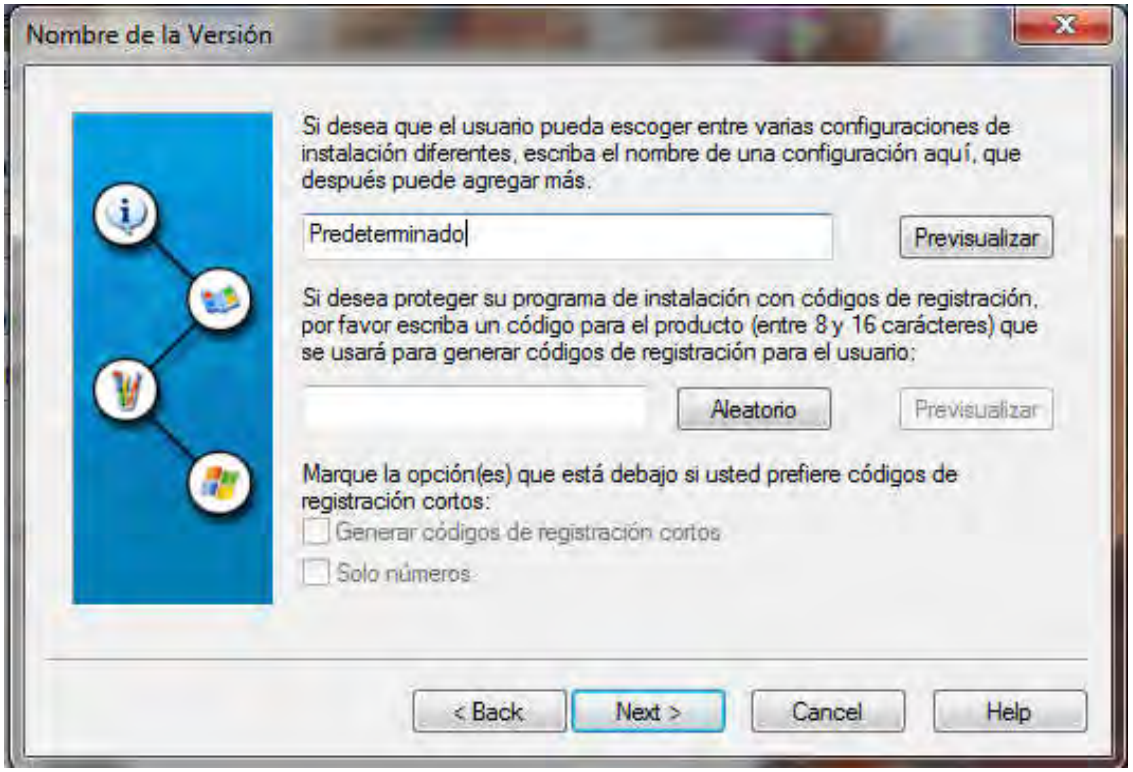




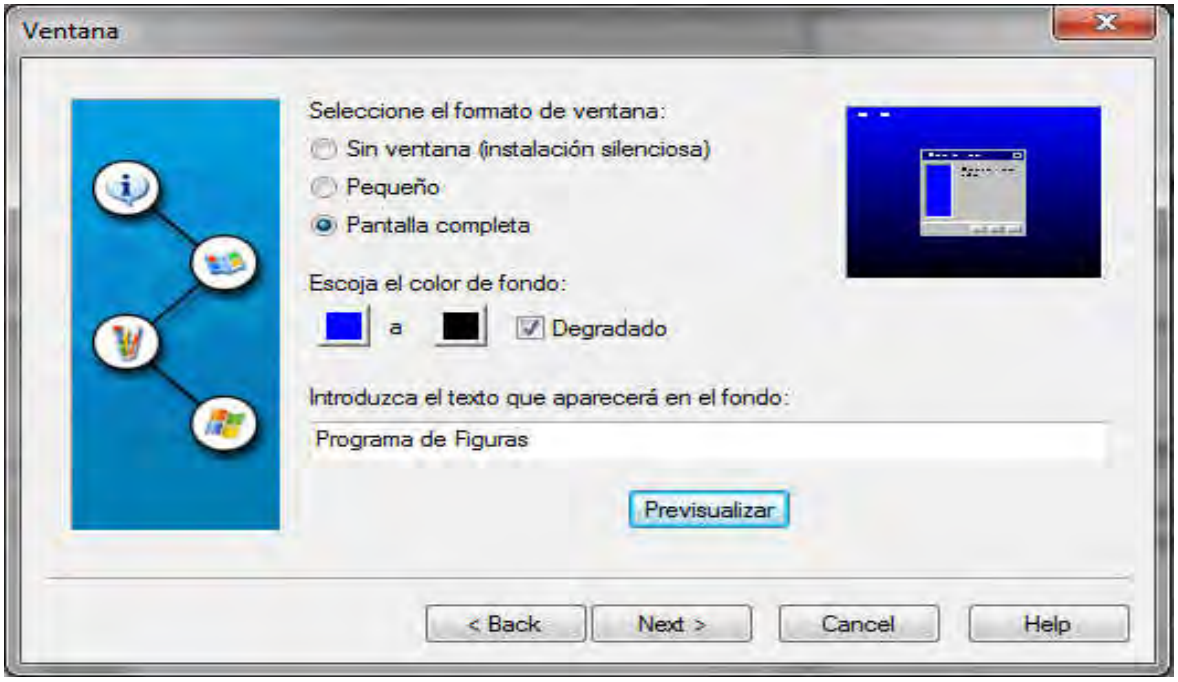
Se elige el idioma y el nombre del instalador



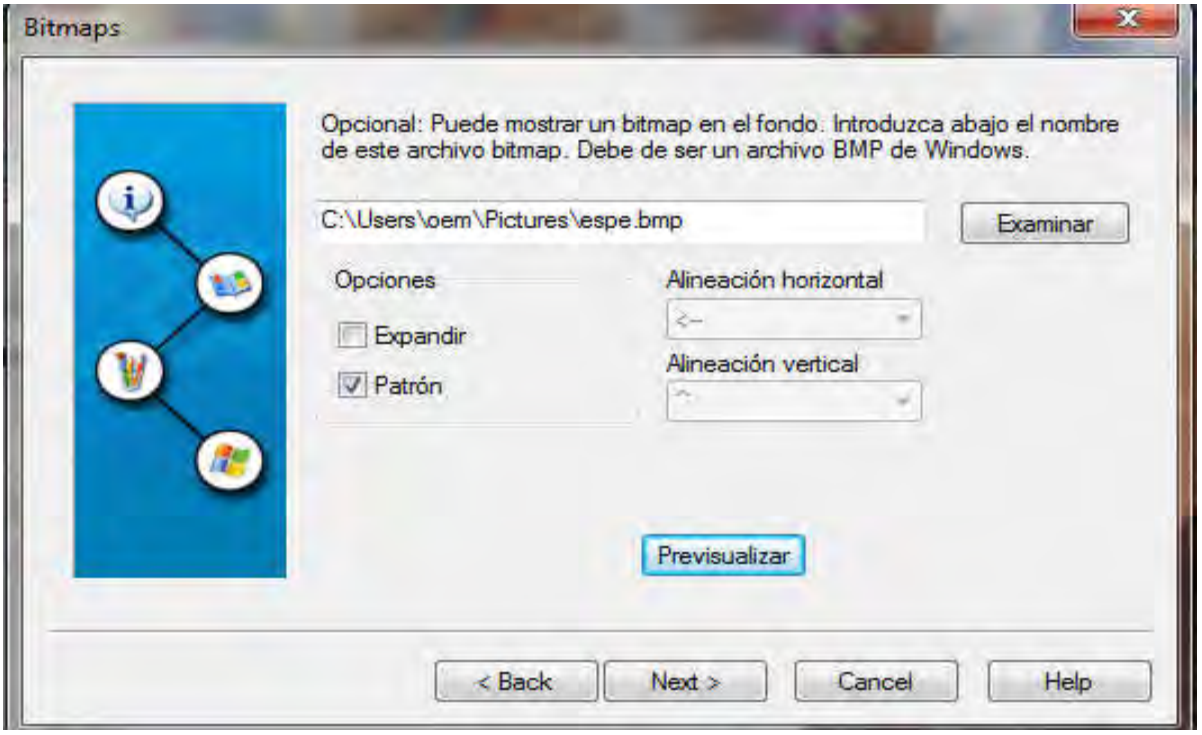
En las siguiente ventana, se elige predeterminado para una instalación común, si se desea modificar las opciones avanzadas se deja a su criterio, igualmente si se requiere pedir un código de licencia



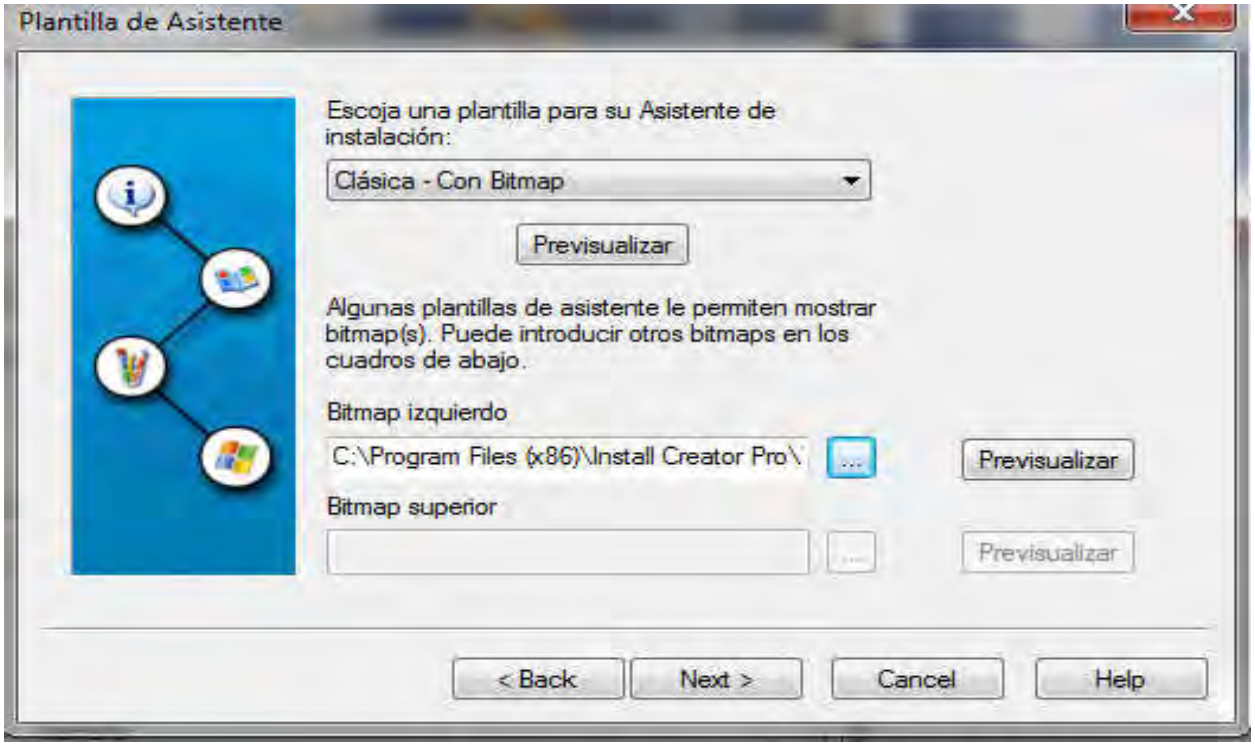
Ahora se selecciona el tamaño de pantalla y texto de fondo, es decir opciones visuales al momento de instalar



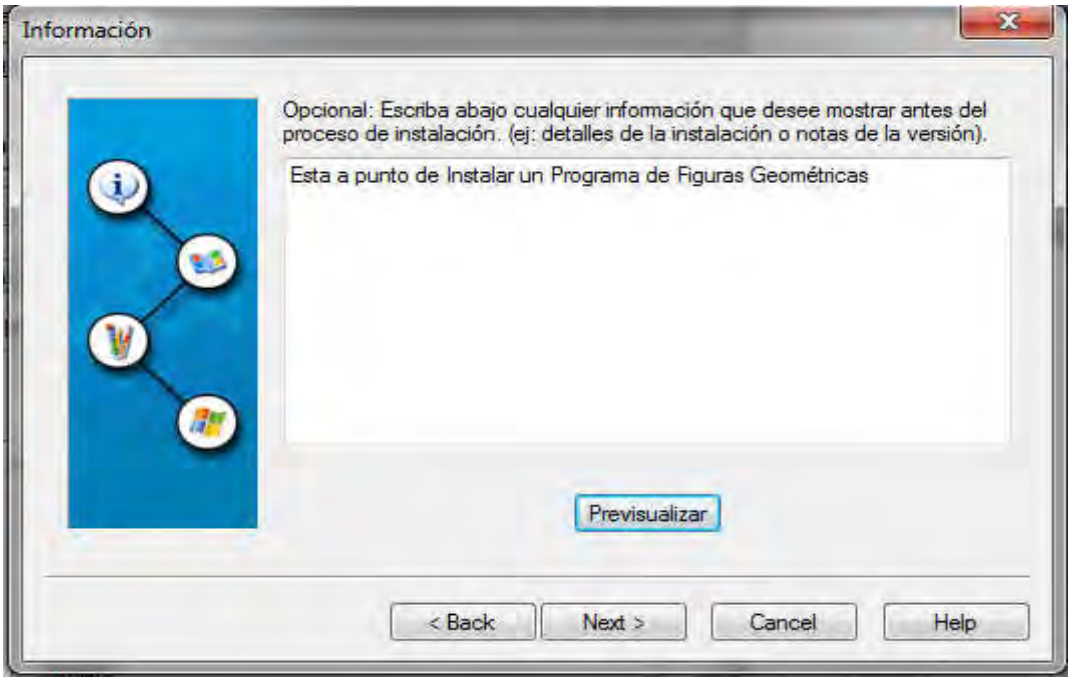
En la siguiente ventana se selecciona si se desea una imagen tipo BMP para el fondo



En esta ventana se selecciona más opciones gráficas que son para mejorar la presentación básicamente

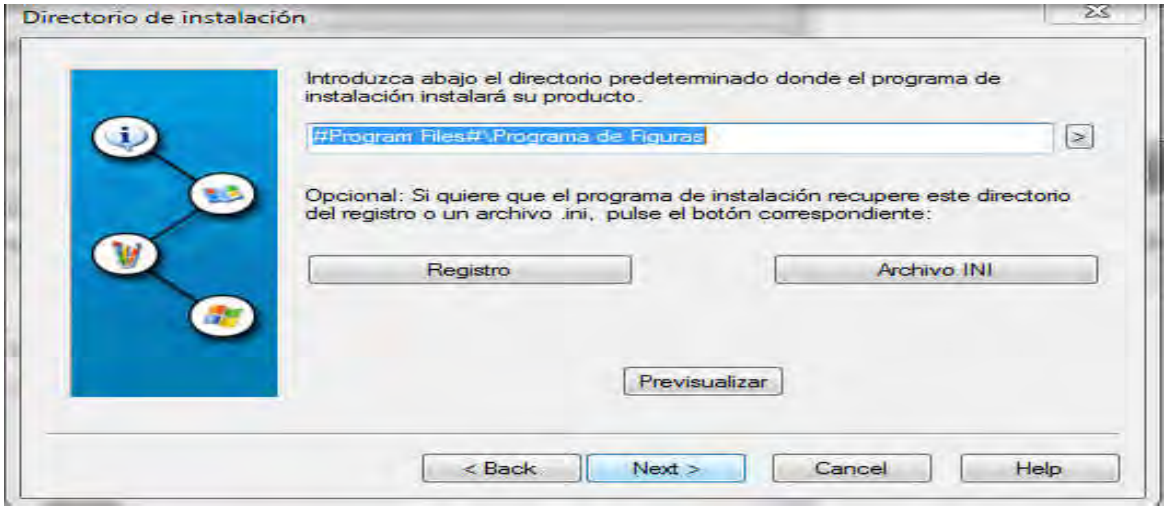


Se escribe un mensaje para hacer más amigable con el usuario el instalador

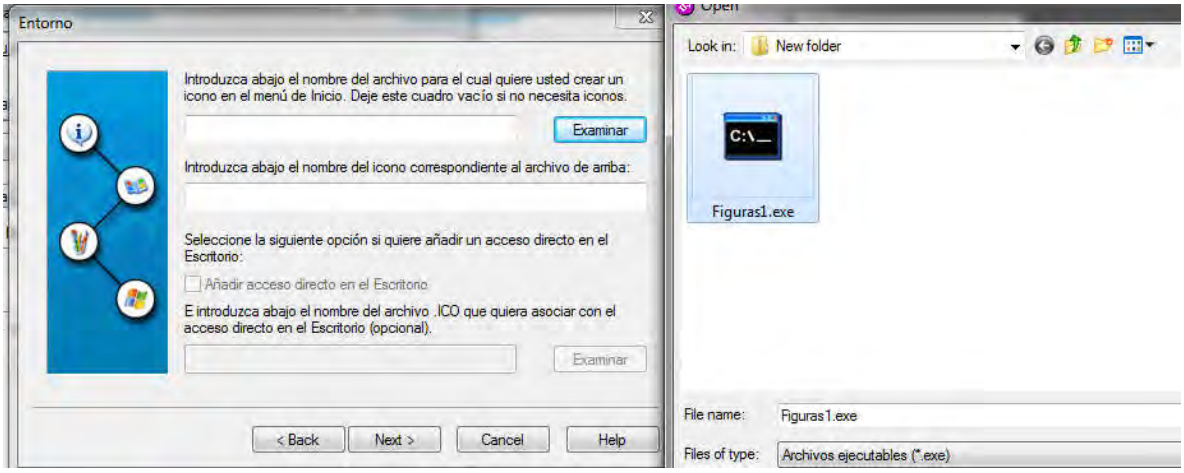


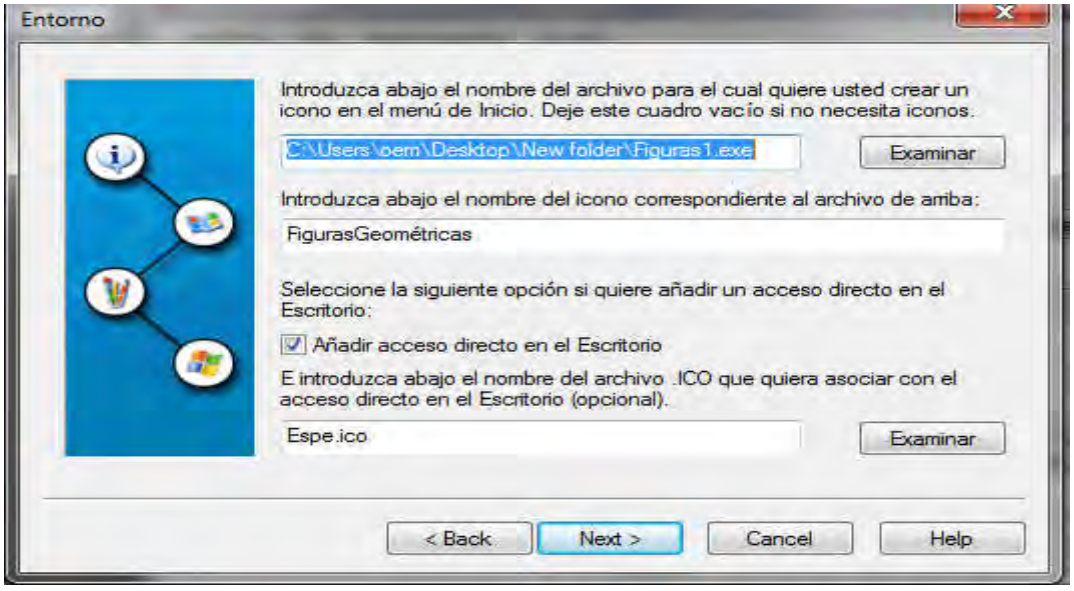
Se introduce opcionalmente un mensaje de acuerdo con el usuario.

Esta opción se de*jará* por default, ya que el usuario escogerá donde se ubicará el programa

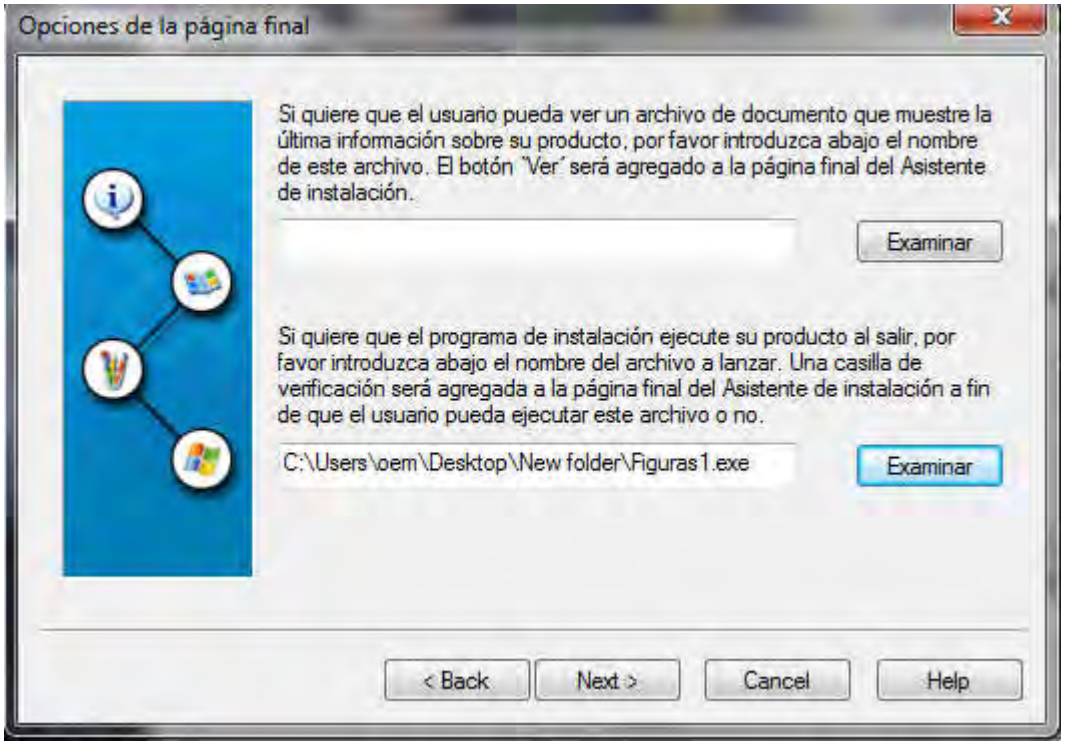


En la siguiente ventana se especificará la dirección del archivo .*exe* y se elige el nombre del ícono

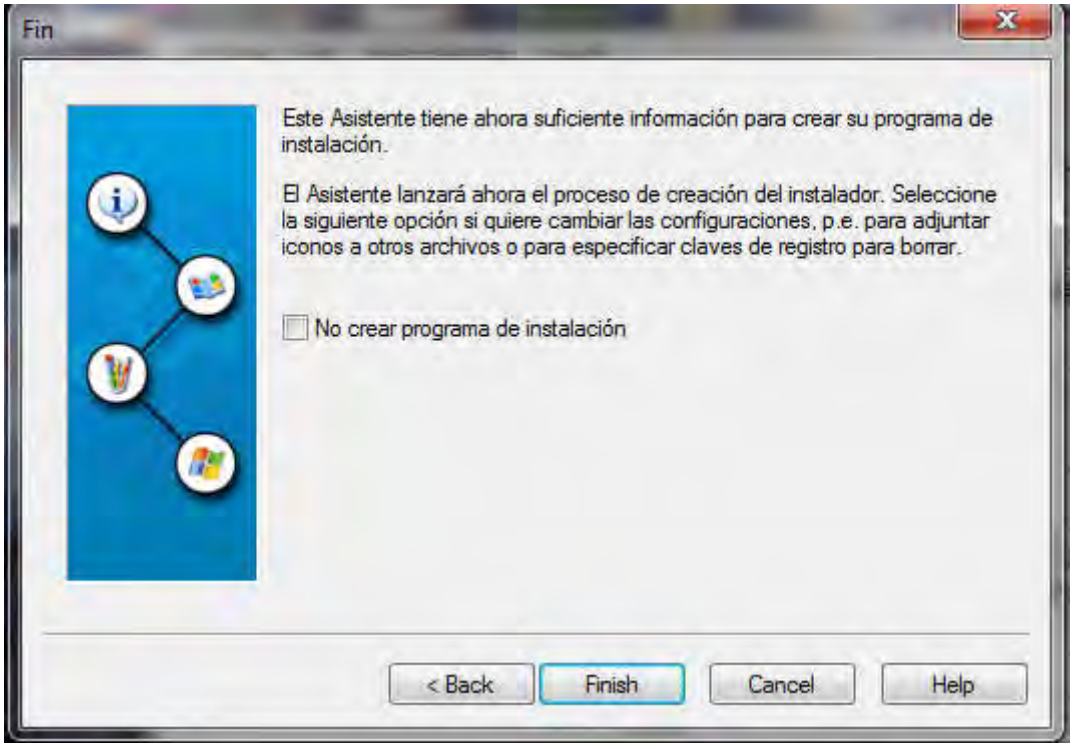




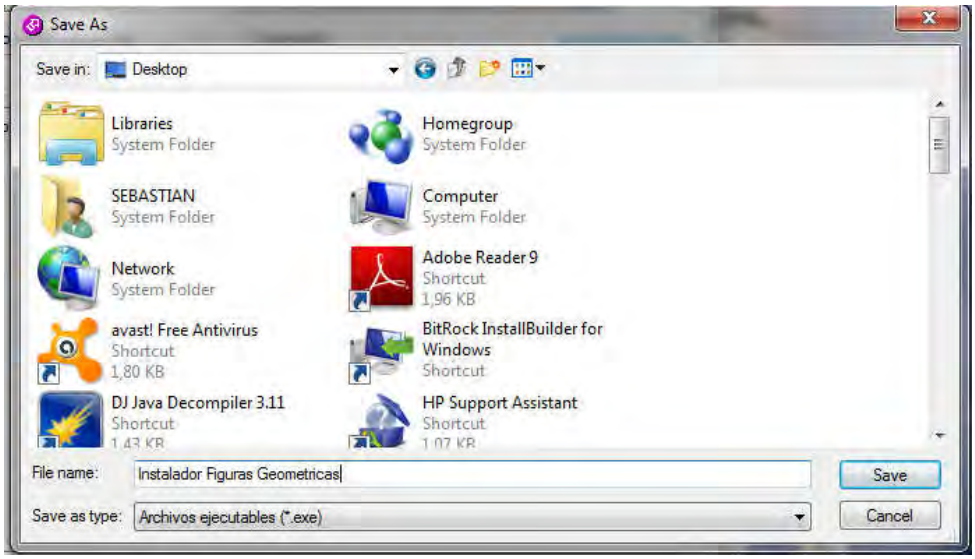
Esta opción escoge el archivo *.exe* para que se ejecute al salir de este ayudante.



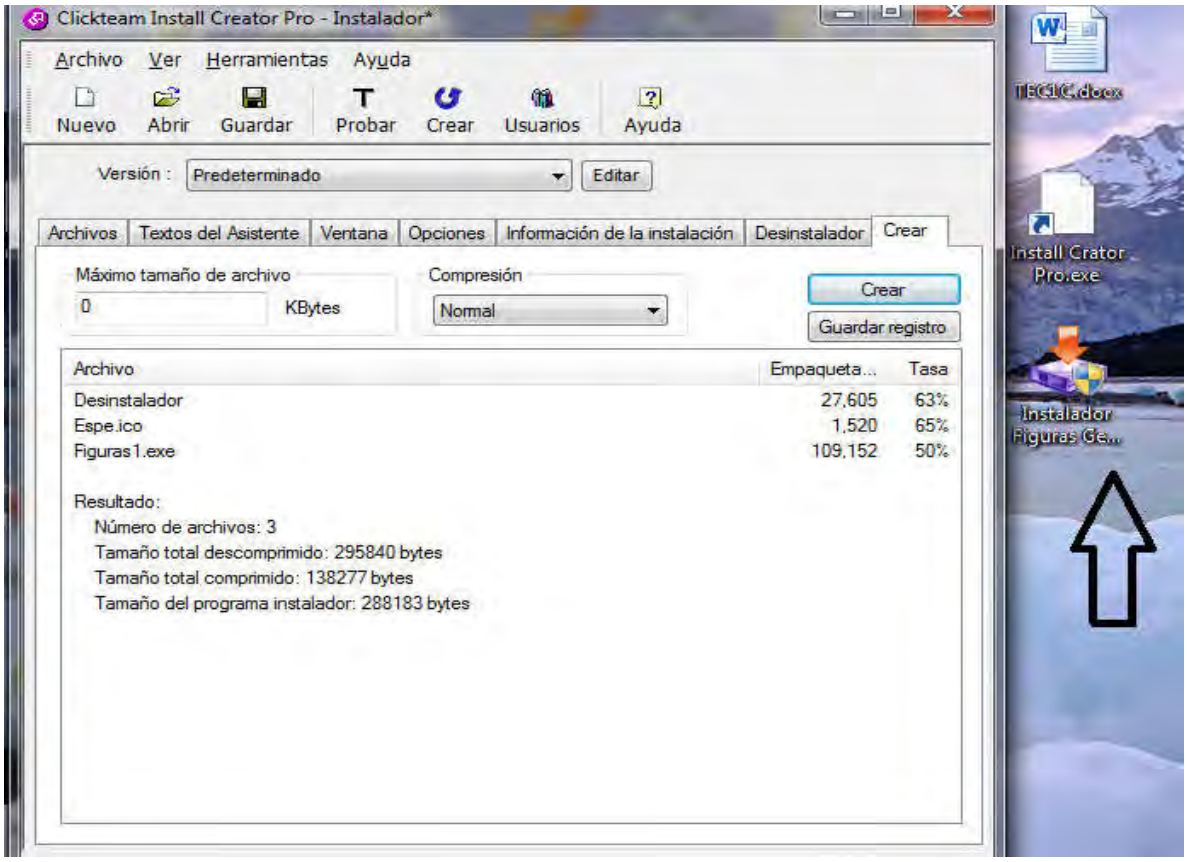
Se finaliza el ayudante.



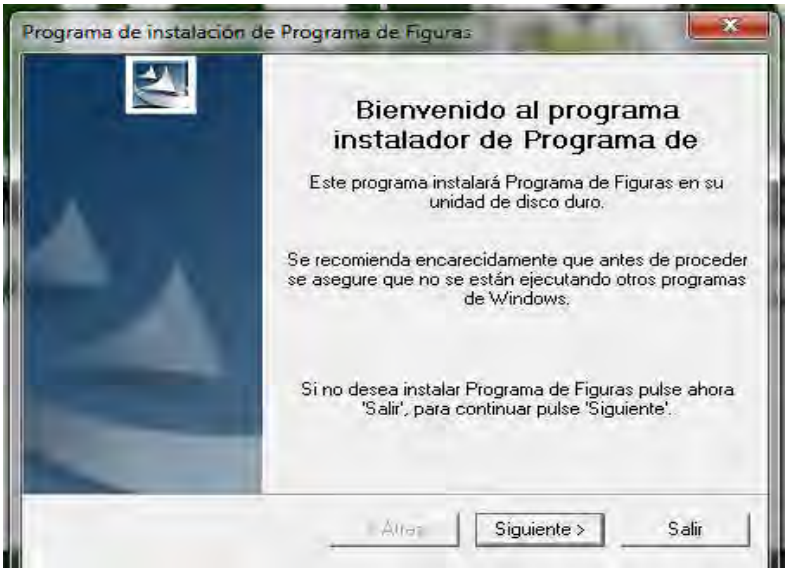
Aquí se guarda el nombre del instalador *.exe*



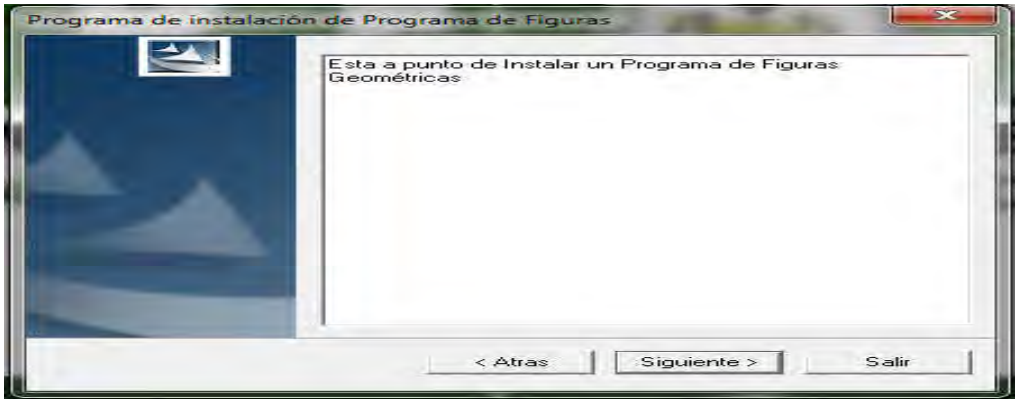
Se cierra el INSTALL CREATOR PRO y como se puede observar, el instalador está creado en el escritorio



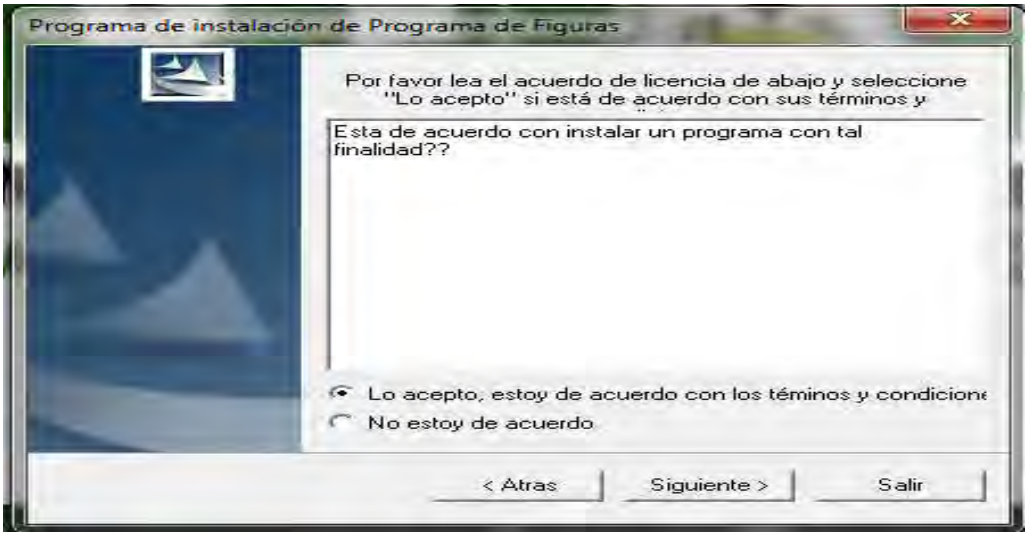
Ahora se prueba el instalador. Doble *click* y se despliega esta ventana.



Son pasos sencillos que basta con SEGUIR las instrucciones del instalador creado.

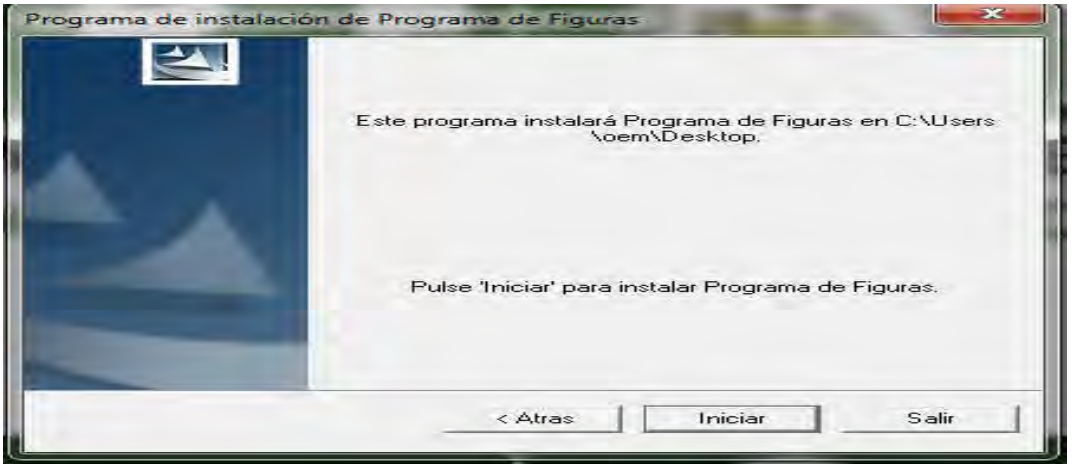


Aquí se acepta la licencia que se creó.

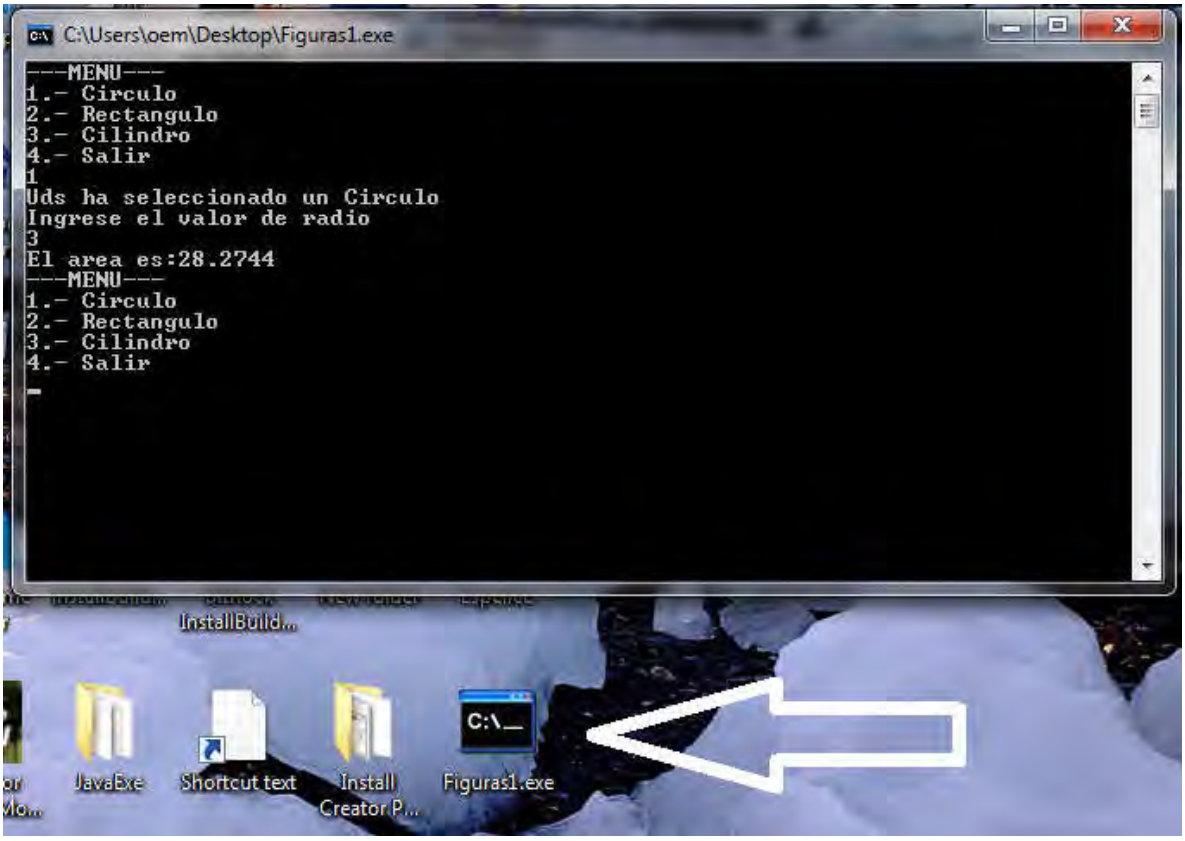


Se elige el destino donde se creará el ejecutable

Prácticamente se ha finalizado la instalaciónA continuación se ejecutará el programa.



Se puede observar que está creado el *.exe* en el escritorio.


C:\Users\oem\Desktop\Figuras1.exe
---MENU---
1.- Circulo
2.- Rectangulo
3.- Cilindro
4.- Salir
1
Uds ha seleccionado un Circulo
Ingrese el valor de radio
3
El area es:28.2744
---MENU---
1.- Circulo
2.- Rectangulo
3.- Cilindro
4.- Salir
InstallBuild...
JavaExe
Shortcut text
Install Creator P...
Figuras1.exe

CAPITULO 16

MANEJO DE LA CPONSOLA DE WINDOWS CON JAVA

La ejecución de aplicaciones Java requiere la existencia de una JVM, la cual es la encargada de interpretar los byte-codes, los cuales son los ejecutables, pero si consideramos que Java es multiplataforma, en la que para la ejecución de aplicaciones de consola java hace uso de Command Prompt (CMD) de Windows, por lo que el usuario debe acceder a esta aplicación para desde ahí ejecutar el programa.

Lo que se pretende en esta sección es evitar tener que acceder de forma manual al CMD, y más bien que sea la aplicación Java la encargada de ejecutar el CMD y cargarse sobre este.

CMD

Es el sucesor de command.com en Windows NT/2000/XP/2003. Se diferencia del anterior ya que este programa es tan sólo una aplicación, no es una parte del sistema operativo.

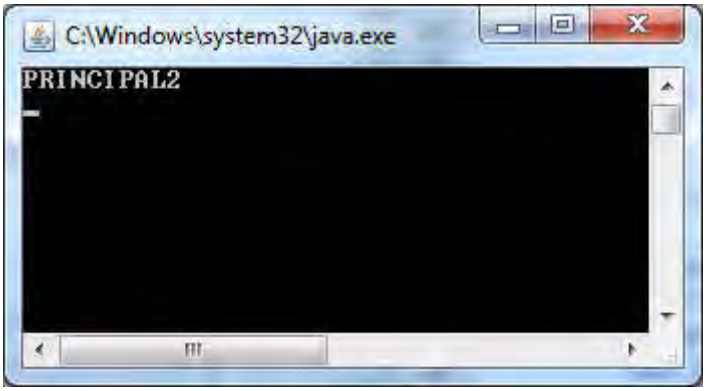
La aplicación consta de partes: en la primera se crea la clase que abrirá el CMD para que éste a su vez ejecute el .class que se necesite que corra en el CMD. Para que se pueda apreciar mejor el funcionamiento se debe crear el .jar.

```
public class Principal {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        try {
            Process eject = Runtime.getRuntime().exec("cmd.exe /K start java Principal2");
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Para la segunda parte se crea el proyecto, pero considerando que el hilo de la aplicación no termine antes de que se pueda apreciar, por ejemplo si el programa tiene una única secuencia, se puede agregar al final la lectura por teclado para que el programa se detenga hasta que se ingrese una tecla cualquiera.

```
import java.util.Scanner;

public class Principal2 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.println("PRINCIPAL2");
        new Scanner(System.in).nextInt();
    }
}
```

Para su ejecución se debe disponer del primer.jar y el .class que se los puede ubicar en cualquier ruta de la máquina. Si se dispone únicamente del .java, se podría agregar una línea adicional en la primera aplicación, para que ésta compile el proyecto (javac) antes de ejecutar (java).



NOTA: Para que la aplicación funcione perfectamente se debe tener configurado el Path de Windows para que reconozca Java automáticamente y agregando la ruta donde se encuentre el java.exe

CAPITULO 17

MANEJO DE LA IMPRESORA CON JA

Dentro de cualquier proceso es necesario contar con una forma en la que el PC da información al usuario, y por lo general esto es por medio de la pantalla, pero para determinadas actividades, es necesaria una constancia física; como una nota de venta, un informe, o una imagen; por lo que es necesario que las aplicaciones puedan controlar la impresora.

La clase que permite controlar la impresora se encuentra en el paquete AWT.

Package java.awt

Se trata de una biblioteca de clases Java para el desarrollo de Interfaces de Usuario Gráficas.
Permite crear entornos graficos de ventanas, a la manera de Windows
Aspecto 'similar' en varias plataformas

import java.awt.Color;	• La clase de color se utiliza encapsular colores en el espacio de color sRGB por defecto o los colores de color arbitrario espacios identificados por un espacio de color.
import java.awt.Font;	• La clase Font representa fuentes, que se utilizan para representar el texto de una manera visible.
import java.awt.PrintJob;	• Una clase abstracta que inicia y ejecuta un trabajo de impresión.
import java.io.BufferedReader;	• Leer texto de un flujo de caracteres de entrada, zonas de separación personajes a fin de proporcionar para la lectura eficiente de los personajes, matrices y líneas.

La aplicación que se presenta permite la apertura de un documento y su impresión.

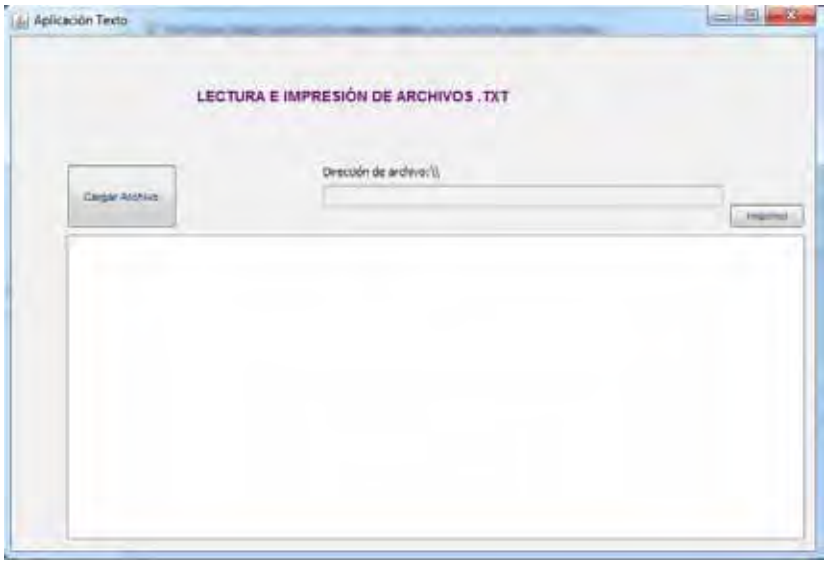
CODIGO FUENTE

```
import java.awt.Color;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.PrintJob;
import java.awt.Toolkit;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import javax.swing.JFileChooser;

public class Principal extends javax.swing.JFrame {
    String archivo="";
      String imprimir="";
     Font fuente = new Font("Dialog", Font.PLAIN, 10);
          PrintJob pj;
          Graphics pagina;

    public Principal() {
        initComponents();
    }

    private void btncargarActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser cuadrobusqueda=new JFileChooser();
        cuadrobusqueda.showOpenDialog(null);
        File aux;
        String separador = System.getProperty("line.separator");
        aux = cuadrobusqueda.getSelectedFile();
        if(aux.equals(null)){

        }
        else{
            txtruta.setText("");
            txtarea.setText("");
            txtruta.setText(String.valueOf(aux));
        archivo=String.valueOf(aux);
        FileReader fr;
        try {
        fr = new FileReader (new File(archivo));
        BufferedReader br = new BufferedReader (fr);
        String line = br.readLine();
        while (line != null) {
          txtarea.setText(txtarea.getText()+line+separador);
          imprimir = imprimir + line +separador;
          line = br.readLine();
      }
```

```
      br.close();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
       }
    }

    private void btnimprimirActionPerformed(java.awt.event.ActionEvent evt) {
       imprimir(imprimir);
     }

public void imprimir(String Cadena)
          {
         pj = Toolkit.getDefaultToolkit().getPrintJob(new Frame(), "SCAT", null);
                    try
                    {
                              pagina = pj.getGraphics();
                              pagina.setFont(fuente);
                              pagina.setColor(Color.black);

                              pagina.drawString(Cadena, 60, 60);

                              pagina.dispose();
                              pj.end();
                    }catch(Exception e)
                    {
                              System.out.println("LA IMPRESION HA SIDO CANCELADA...");
                    }
          }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Principal().setVisible(true);
            }
        });
    }

    private javax.swing.JButton btncargar;
    private javax.swing.JButton btnimprimir;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextArea txtarea;
    private javax.swing.JTextField txtruta;

}
```

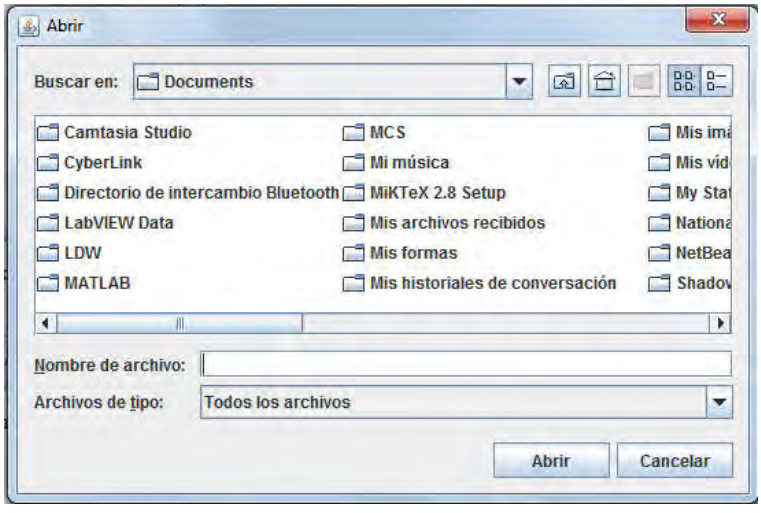
Se trata de una aplicación desarrollada en java que tiene el objetivo de servir al usuario con dos funciones específicas que son:

- Cargar un archivo de texto es decir con extensión *.txt y mostrarlo.
- Imprimir el archivo de texto que fue cargado si el usuario lo requiere.

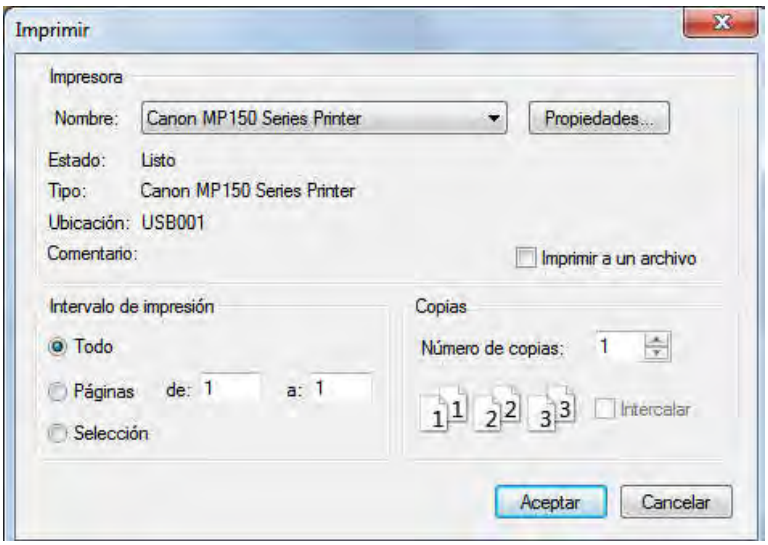


Para poder realizar la búsqueda del archivo del cual pretendemos visualizar su contenido tenemos el siguiente botón:

Si se da un clicken Cargar Archivo, presentará el directorio de búsqueda:



Al encontrar el archivo que deseemos abrir en el computador se presiona el botón Abrir y automáticamente se podrá visualizar el texto que contiene el archivo. Y si desea el usuario imprimir el archivo pulsará o dará un click en el botón Imprimir, el cual presentará la pantalla de Imprimir acostumbrada para elegir las opciones de impresión e impresora:

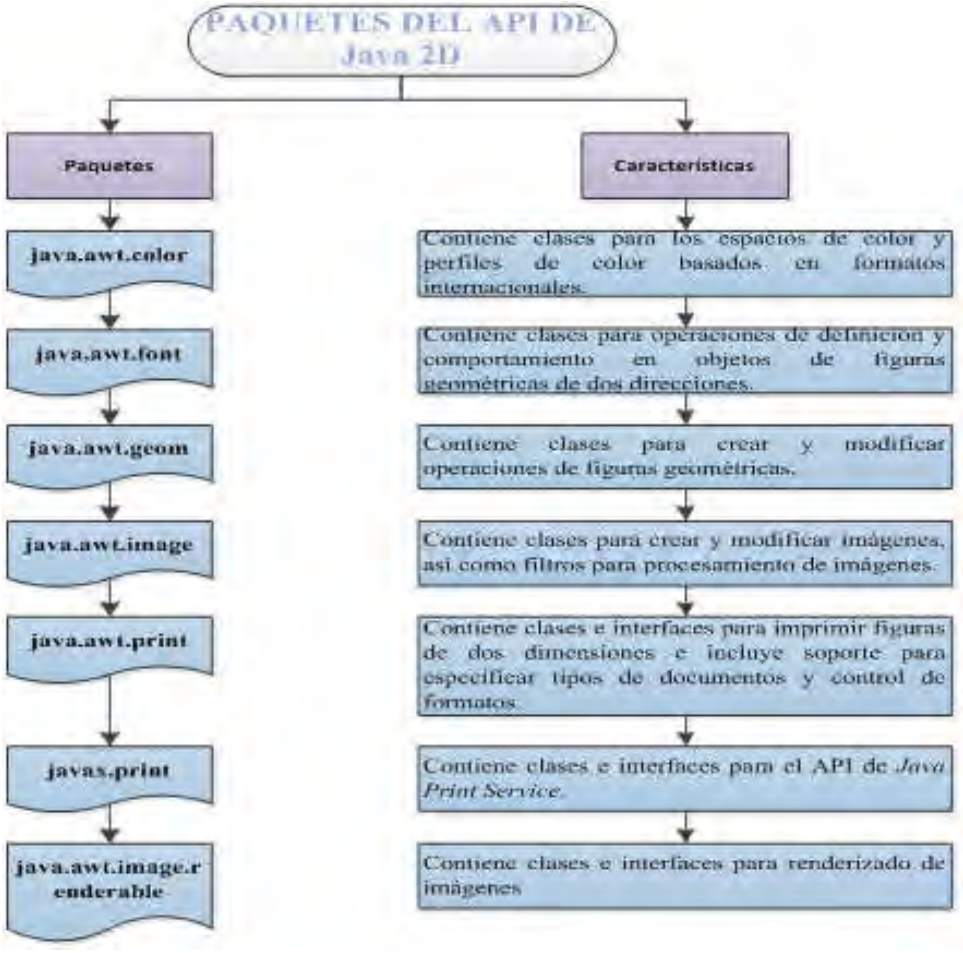

Imprimir
Impresora
Nombre:
Canon MP150 Series Printer
Propiedades...
Estado:
Listo
Tipo:
Canon MP150 Series Printer
Ubicación:
USB001
Comentario:
Imprimir a un archivo
Intervalo de impresión
Todo
Páginas
de: 1
a: 1
Selección
Copias
Número de copias:
1
Intercalar
Aceptar
Cancelar

CAPITULO 18

GRÁFICO DE FUNCIONES CON JAVA 2D

Java permite el manejo de imágenes de múltiples formas, una de ellas es el API de Java 2D, que mejora al modelo previo de la biblioteca *AWT,* de tal forma que admite la creación de interfaces de usuario más complejas que las interfaces creadas solamente con *Java Graphics*. *Java 2D* permite el control básico para la creación de figuras geométricas y de texto (que es tratado como imagen) hasta complementos como transparencias, texturas y rellenos.

La siguiente lista contiene todos los paquetes que forman parte de interfaz de programación de aplicaciones o el API de Java 2D. Cada uno de estos paquetes contiene las clases necesarias para trabajar con objetos específicos, permitiendo mayor libertad en la manipulación de imágenes. Estas son parte necesaria de cada operación de dibujo Java 2D



El *renderizado* básico de *Java 2D* es el siguiente: el sistema de dibujo controla cuándo y cómo dibuja un programa. Cuando un componente necesita ser mostrado, se llama automáticamente a su método *paint* o*update* dentro del contexto *Graphics* apropiado. En

este caso, el *API*de *Java2D* usa *java.awt.Graphics2D*. *Graphics2D* desciende de la clase *Graphics* para proporcionar acceso a las características avanzadas de *renderizado* del *API 2D* de Java. Para esto al objeto *Graphics*se lo debe pasar del objeto de método de dibujo de un componente a un objeto Graphics2D.

Código en el cual se transforma al objeto de tipo *Graphics* en *Graphics2D*.

```
publicvoidPaint (Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    ...
}
```

Java 2D mantiene dos sistemas de coordenadas:

- coordenadas de usuario (*userspace*)
- coordenadas de dispositivo (*devicespace*):

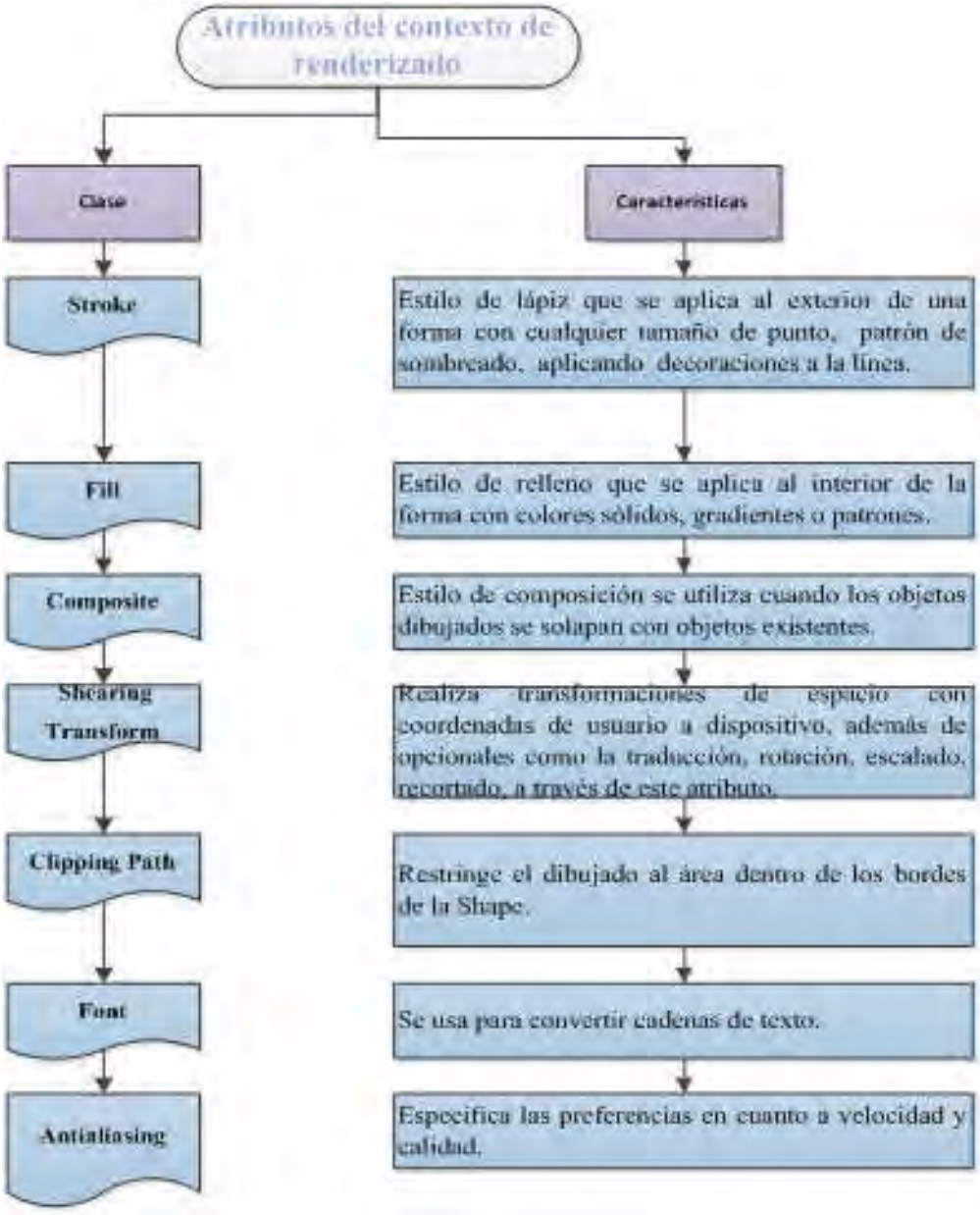
Las coordenadas de usuario constituyen un sistema de coordenadas totalmente independiente del dispositivo final en el que se vaya a hacer el *renderizado*. Las aplicaciones usan exclusivamente este sistema de coordenadas, de forma que el tamaño y posición de todas las figuras geométricas se especifican mediante coordenadas de usuario.Las coordenadas de dispositivo constituyen un sistema de coordenadas que sí depende del dispositivo en el cual va a *renderizarse* realmente.

Java2D ejecuta automáticamente las conversiones entre los dos sistemas de coordenadas en el momento en que se realiza el *renderizado* real sobre un dispositivo.

En la tabla siguiente, a modo de resumen, se muestran las clases que intervienen en el sistema de coordenadas:

Clases en sistemas de coordenadas	
Clase	**Descripción**
GraphicsEnvironment	Define la colección de dispositivos visibles desde una plataforma Java.
GraphicsDevice	Define un dispositivo concreto de gráficos.
GraphicsConfiguration	Define un modo de ese dispositivo concreto.

Contexto de renderizado de Graphics2D.- Es un conjunto de atributos de estado asociados con un objeto Graphics2D. Se usa para mostrar texto, formas o imágenes a través de la configuración de este contexto y luego se llama a uno de los métodos de *renderizado* de la clase Graphics2D.



Para configurar un atributo en el contexto de *renderizado* de Graphics2D, se usan los métodos *set Attribute*:

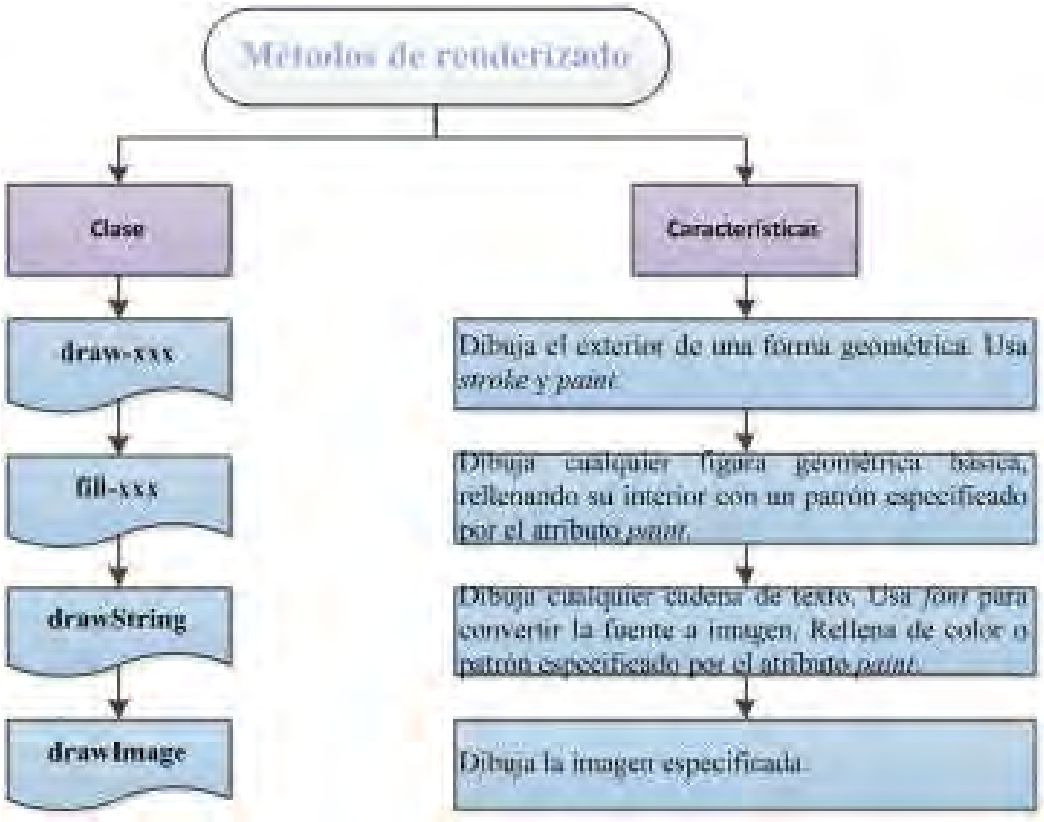
- *setStroke*
- *setPaint*
- *setComposite*
- *setTransform*
- *setClip*
- *setFont*
- *setRenderingHints*

Cuando se configura un atributo, se pasa al objeto el atributo apropiado.

Código en el cual se cambia un atributo *paint* a un relleno de gradiente azul-gris, construyendo el objeto *GradientPaint* y llamando a *setPaint*.

```
public void Paint (Graphics g) {
      Graphics2D g2 = (Graphics2D) g;
      gp = new GradientPaint(0f,0f,blue,0f,30f,green);
       g2.setPaint(gp);
  }
```

Métodos de renderizado de Graphics2D.- Graphics2D tiene métodos generales de dibujado para dibujar cualquier primitivo geométrico, texto o imagen. Además soporta los métodos de *renderizado*de *Graphics* para formas particulares, como *drawOval*y *fillRect*.



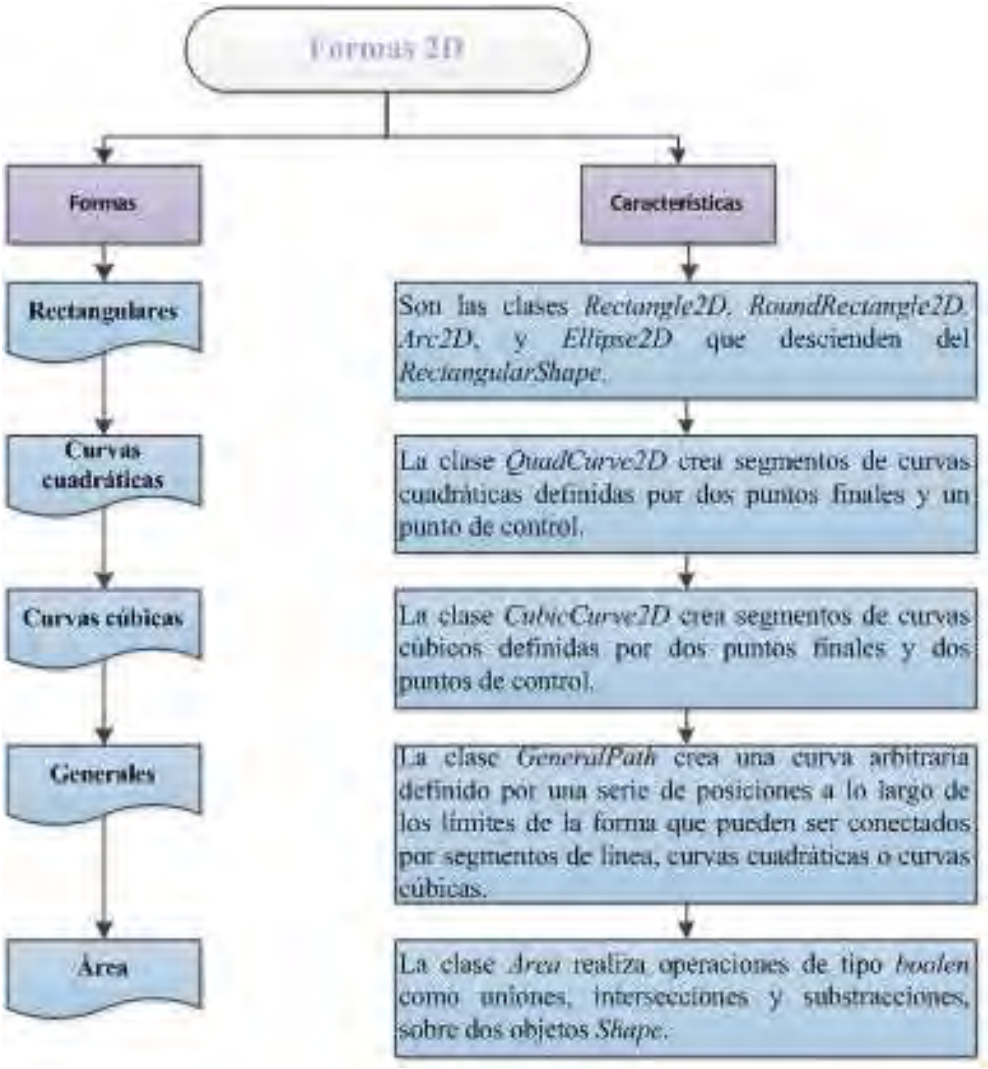
Formas 2D.- Están definidas por el paquete*java.awt.geom.* Se puede construir gráficos primitivos comunes, como puntos, líneas, curvas, arcos, rectángulos y elipses.

Las clases de *java.awt.geom* son:

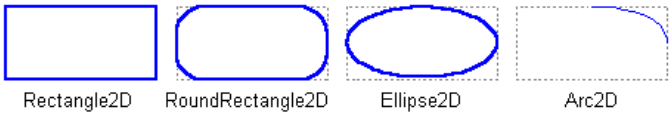
Arc2D	Ellipse2D	QuadCurve2D
Area	GeneralPath	Rectangle2D
CubicCurve2D	Line2D	RectangularShape
Dimension2D	Point2D	RoundRectangle2D

Las clases implementan el interface Shape (a excepción de *Point2D* y *Dimension2D*), que proporciona un conjunto de métodos comunes para describir e inspeccionar objetos geométricos de dos dimensiones. Se crea de modo virtual cualquier forma y se la dibuja a

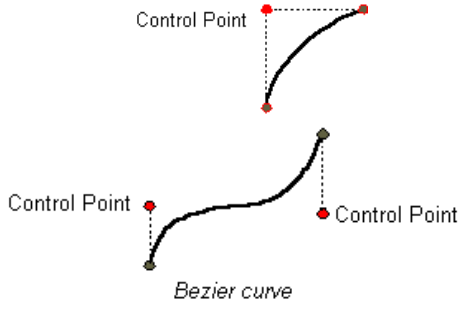
través de *Graphics2D,* llamando al método *draw*o *fill*.



Formas rectangulares



Formas basadas en curvas



Formas generales

Jep - Parser Java de Expresiones Matemáticas

Este parser es muy sencillo de utilizar, sin embargo se pueden crear funciones complejas con muchas variables y a su vez con muchas funciones.Los operadores Aritméticos que se pueden utilizar son:

- + Suma
- - Resta
- / División
- * Multiplicacion
- % Módulo
- ^ Potencia

Pero también soporta operadores lógicos por lo que dispondremos del && (AND), || (OR), ! (NOT), >, <, >=, <=, == (EQUALS), entre otros. Este parser a su vez dispone de funciones standart para ser utilizadas, en el código se tiene que habilitar con la linea jep.addStandardFunctions();

- sin(x): Seno
- cos(x): Coseno
- tan(x): Tangente
- asin(x): ArcoSeno
- acos(x): ArcoCoseno
- atan(x): ArcoTangente
- sinh(x): Seno Hiperbolico
- cosh(x): Coseno Hiperbolico
- tanh(x): Tangente Hiperbolica
- ln(x): Logaritmo Natura
- log(x): Logaritmo en base 10
- lg(x): Logaritmo en base 2
- exp(x): exponencial

Para habilitar las constantes se debe agregar al codigo jep.addStandardConstants()

La página oficial es http://www.singularsys.com/jep/.

La librería transforma y evalúa expresiones matemáticas con sólo unas líneas de código. Esto permite que un usuario ingrese una fórmula como un *String* e inmediatamente lo evalúa.

Esta aplicación tiene como funcionalidad principal la generación de gráficas desde una función matemática ingresada.

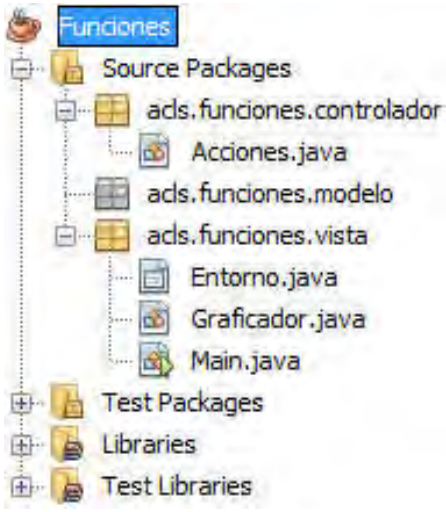
A partir de los conceptos revisas de Java 2D y JEP, se desarrolla la aplicación que permite graficar funciones. El programa tiene la siguiente estructura de funcionamiento:

1. Se ingresa la ecuación a graficar
2. El usuario puede variar las escalas de los ejes para una mejor visualización de los ejes.
3. Salir del programa o pulsar el botón Limpiar para ingresar otra gráfica.

Se deben tener en cuenta las siguientes consideraciones generales en el momento de ejecutar el programa:

- Las funciones ingresadas deben tener como única variable independiente a la letra 'x'.
- Revise la tabla de caracteres permitidos para el ingreso de funciones, tales como signos para suma, multiplicación o logaritmos.

Mapa de Variables:



Paquete Vista	
Class Entorno	*JFrame: Grafico* Crea el Frame en donde se va a graficar la función *String: guardarY* Guarda la escala de y *doublé: guardarX* Guarda la escala de x
Class Graficador	*escala 1: String* Almacena el valor en el eje x *escala 2: double* Almacena el valor en el eje y *función: String* Almacena la función evaluar

Paquete Controlador	
Class Acciones	*ErrorExpresion: booleano* Es verdadero cuando existe error de sintaxis en la función *evaluador: JEP* Objeto de la clase JEP

Explicación de los métodos:

Paquete Controlador		
Class Acciones	Public Acciones()	**Tipo:** void (constructor)
		Argumentos: no tiene argumentos
		Funcion: Inicializa las características del objeto de la clase JEP.
	Public doublé Evaluar(String function, double valorx)	**Tipo:**doublé
		Argumentos: String function, double valorx.
		Funcion:evalua f(x)

Paquete Vista		
Class Graficador	Public Graficador()	**Tipo:** void (constructor)
		Argumentos: no tiene argumentos
		Funcion: Inicializa datos para realizar la gráfica.
	Void paintComponent(Graphics obj)	**Tipo:** void()
		Argumentos: Graphycs obj
		Funcion: Tiene todos los instrumentos para graficar.
		Argumentos: String args [ ] -> Estructura del encabezado para el método main
		Funcion *del Método:* El método main es el encargado de relacionar directa o indirectamente la lógica y secuencia de la aplicación mediante la funcionalidad de los métodos creados en los paquetes Modelo y Controlador. *1. Realizar el proceso*

		1.1 Ingresar función: Realiza un proceso de gráficas de la función ingresada por el usuario, se puede manipular las escalas después de dar clickel botón gráficar 1.2 Salir al Menú Principal *Salir (Finalizar la programación).*
Class Entorno(JFrame)	Public entorno()	**Tipo:** void (constructor)
		Argumentos: Sin argumentos
		Funcion: Inicializa datos
	public static void escalarX(final JFrame grafico)	**Tipo:**void
		Argumentos:final JFrame gráfico
		Funcion: Obtiene el valor del evento del JSlaider del eje x
	public static void escalarY(final JFrame grafico)	**Tipo:**void
		Argumentos:final JFrame grafico
		Funcion: Obtiene el valor del evento del JSlaider del eje y

Clase Acciones:

```
import org.nfunk.jep.*;
import javax.swing.JPanel;

public class Acciones extends JPanel{

    JEP evaluador = new JEP();
    boolean errorExpresion;

    public Acciones() {
        evaluador.addStandardFunctions();
        evaluador.addStandardConstants();
        evaluador.addVariable("x", 0);
        evaluador.setImplicitMul(true);
    }

    public double Evaluar(String funcion, double valorx) {
        evaluador.parseExpression(funcion);
        evaluador.addVariable("x", valorx);
        errorExpresion = evaluador.hasError();
        return evaluador.getValue();
    }
}
```

Clase Main:

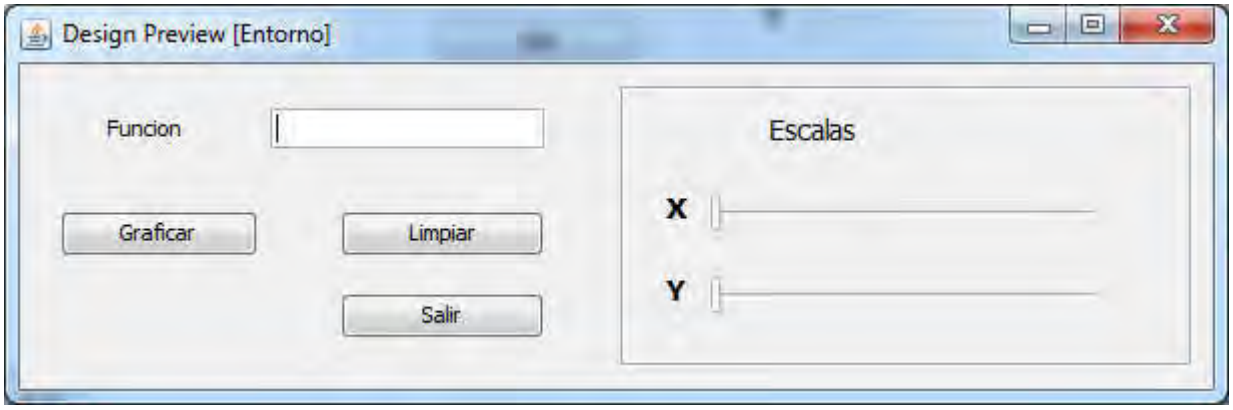
```
public class Main {

    public static void main(String[] args) {
        // TODO code application logic here
        Entorno obj = new Entorno();
        obj.setLocation(200, 540);
        obj.setVisible(true);
    }

}
```

Clase Graficador:

```
import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;
import javax.swing.*;

public class Graficador extends JPanel {

    String funcion;
    String escala1;
    double escala2;

    public Graficador(String funcion, String escalaY, double escalaX) {
        this.funcion = funcion;
        this.escala1=escalaY;
        this.escala2=escalaX;
    }


    @Override
    protected void paintComponent(Graphics obj) {
        super.paintComponent(obj);
        Graphics2D obj2d = (Graphics2D) obj;

        obj2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        obj2d.setPaint(Color.gray);
        obj2d.draw(new Line2D.Double(0, getHeight() / 2, getWidth(), getHeight() / 2));
        obj2d.setPaint(Color.gray);
        for (int i = -getWidth(); i < getWidth(); i = i + 10) {
            double x = i;
```

```
            double y = getHeight() / 2 - 3;
            obj2d.fill(new Rectangle2D.Double(x, y, 1, 6));
        }

        obj2d.setPaint(Color.gray);
        obj2d.draw(new Line2D.Double(getWidth() / 2, getHeight(), getWidth() / 2, 0));
        obj2d.setPaint(Color.gray);
        for (int i = -getHeight(); i < getHeight(); i = i + 10) {
            double x = getWidth() / 2 - 3;
            double y = i;
            obj2d.fill(new Rectangle2D.Double(x, y, 6, 1));
        }

        Font font = obj2d.getFont();
        FontRenderContext frc = obj2d.getFontRenderContext(); //!!!!!!!!!!averiguar

        String tituloEje = "Eje y";
        float tamañoString1 = (float) font.getStringBounds(tituloEje, frc).getHeight();
        obj2d.setPaint(Color.blue);
        obj2d.drawString(tituloEje, getWidth() / 2 + tamañoString1, tamañoString1);

        tituloEje = "Eje x";
        float tamañoString2 = (float) font.getStringBounds(tituloEje, frc).getWidth();
        obj2d.drawString(tituloEje, getWidth() - 2 * tamañoString2, getHeight() / 2 -
tamañoString1);

        double escalaX = getWidth() / 2;
        double escalaY = getHeight() / 2;
        obj2d.setPaint(Color.green.darker());
        for (int i = -1000; i < 1000; i++) {
            double x1 = (i - 0.5) + escalaX;
            double y1 = (-new Acciones().Evaluar(escala1+funcion, (i - 0.5)/escala2)) +
escalaY;

            double x2 = (i + 0.5) + escalaX;
            double y2 = (-new Acciones().Evaluar(escala1+funcion, (i + 0.5)/escala2)) +
escalaY;
            obj2d.draw(new Line2D.Double(x1, y1, x2, y2));
        }
    }
}
```

Clase Entorno:




```
import javax.swing.*;
import javax.swing.event.*;

public class Entorno extends javax.swing.JFrame {

    JFrame grafico;
    public static String guardary;
    public static Double guardarx;

    /** Creates new form Graficador */
    public Entorno() {
        initComponents();
        grafico = new JFrame();
        guardary = "1*";
        guardarx = 1.0;
        escalaY(grafico);
        escalaX(grafico);
    }

    public static void escalaX(final JFrame grafico) {
        sldejeX.setPaintTicks(true);
        sldejeX.addChangeListener(new ChangeListener() {

            public double escala;

            public void stateChanged(ChangeEvent evt) {
                escala = ((1+sldejeX.getValue()) * 2);
                guardarx = escala;
                grafico.add(new Graficador(txtFuncion.getText(), guardary, escala));
                grafico.setSize(600, 500);
                grafico.setLocation(200, 40);
                grafico.setVisible(true);
            }
        });
    }
    public static void escalaY(final JFrame grafico) {
        sldejeY.setPaintTicks(true);
        sldejeY.addChangeListener(new ChangeListener() {
```

```
        public String escala;

        public void stateChanged(ChangeEvent evt) {
            escala = String.valueOf(sldejeY.getValue() * 2) + "*";
            guardary = escala;
            grafico.add(new Graficador(txtFuncion.getText(), escala, guardarx));
            grafico.setSize(600, 500);
            grafico.setLocation(200, 40);
            grafico.setVisible(true);
        }
    });
}


private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    System.exit(0);
}

private void btnGraficarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code he
    grafico.add(new Graficador(txtFuncion.getText(), "1*", 1));
    grafico.setSize(600, 500);
    grafico.setLocation(200, 40);
    grafico.setVisible(true);

    sldejeX.setEnabled(true);
    sldejeY.setEnabled(true);
    sldejeX.setValue(0);
    sldejeY.setValue(0);


}

private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    txtFuncion.setText(null);
    grafico.setVisible(false);
    grafico=new JFrame();

    sldejeX.setEnabled(false);
    sldejeY.setEnabled(false);
    sldejeX.setValue(0);
    sldejeY.setValue(0);
}

private javax.swing.JPanel Controles;
private javax.swing.JPanel Ingreso;
```

```
    private javax.swing.JButton btnGraficar;
    private javax.swing.JButton btnLimpiar;
    private javax.swing.JButton btnSalir;
    private javax.swing.JLabel lblEscala;
    private javax.swing.JLabel lblGrado;
    private javax.swing.JLabel lblejeX;
    private javax.swing.JLabel lblejeY;
    public static javax.swing.JSlider sldejeX;
    public static javax.swing.JSlider sldejeY;
    public static javax.swing.JTextField txtFuncion;
    // End of variables declaration
}
```

Interfaz de la aplicación:

1. Ventana de gráfica: En esta ventana aparecerá la gráfica que tiene como referencia al eje cartesiano.

2. Ventana de control: Es la ventana que contiene los controles para dibujar y modificar la gráfica.

A continuación se especifica con más detalles la ventana que contiene los controles.

1. Escalas: En esta sección se ubican los *sliders* para cambiar la escala de la gráfica, una vez que ya ha sido graficada.

2. Función: Campo de texto en dónde se debe ingresar la función.

3. Graficar: Botón que ejecuta la gráfica que ha sido ingresada.

4. Limpiar: Botón que reinicia los sliders, borra la gráfica previa y el campo de texto.

5. Salir: Botón para salir de la aplicación.

Ingreso de la función

La función debe ingresarse en el campo de texto y para ello debe tener en cuenta las siguientes consideraciones:

- Sólo se ingresa la función- No se debe ingresar por ejemplo f(x) o 'y'.
- La única variable independiente es 'x'.
- Las operaciones cuentan con una forma de ingreso que se detalla a continuación:

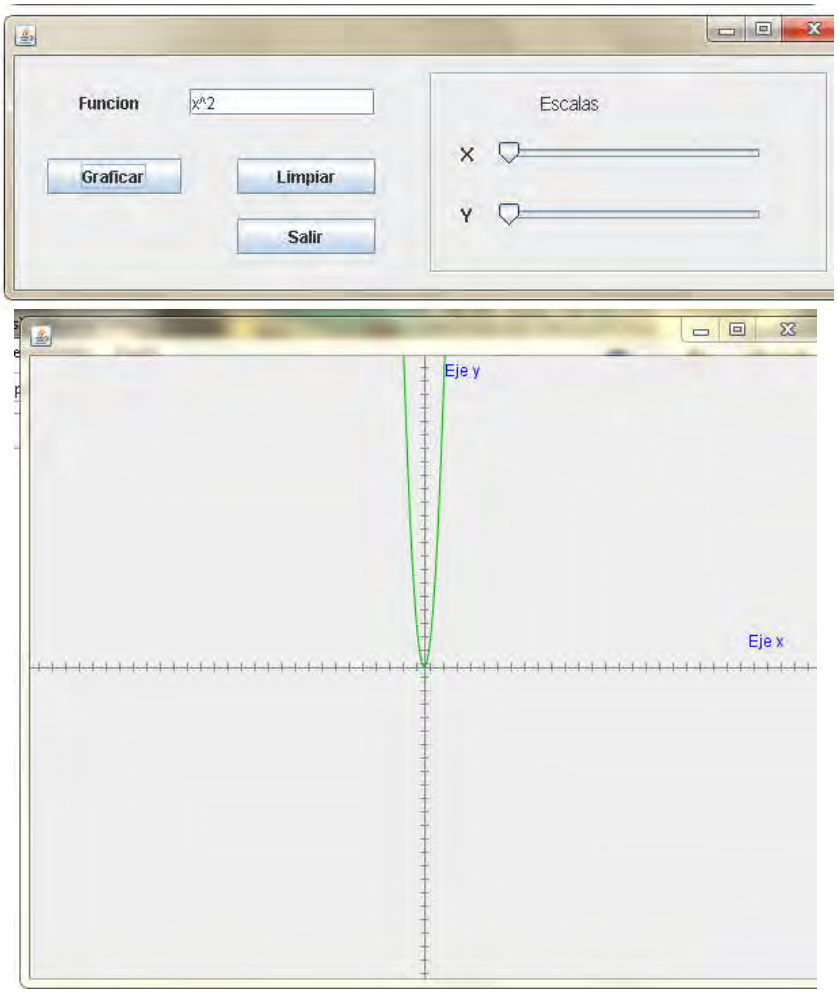
Símbolo matemático	Carácter
Suma	+
Resta	-
Multiplicación	*
División	/
Raíz cuadrada	sqrt(var)
Exponente	^
Seno	sin(var)
Coseno	cos(var)
Tangente	tan(var)
Logaritmo decimal	ln(var)
Logaritmo natural	log(var)

Número pi	pi
Número e	e
agrupación	(x+2)/(3*x)

Donde ‘var’ representa el número o variable.

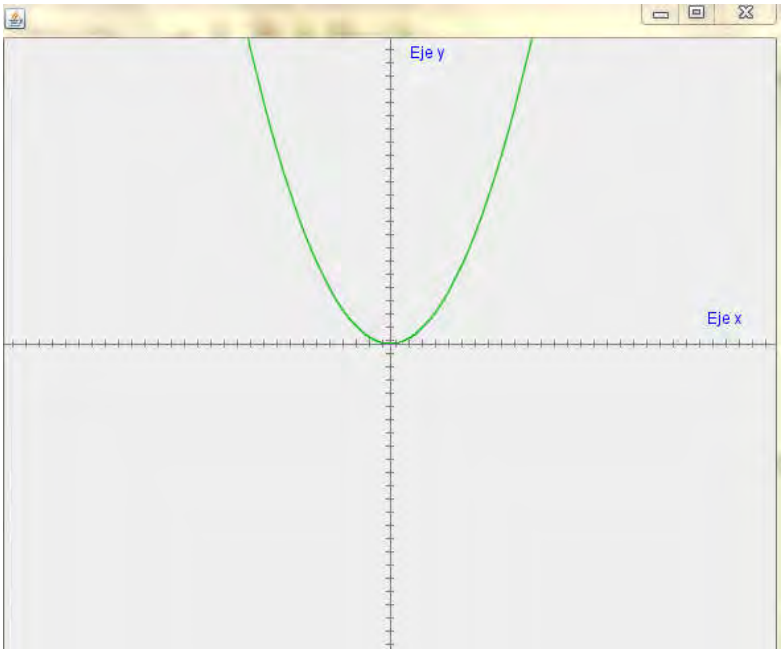
Generar gráfica

- Una vez ingresada la gráfica se debe presiona el botón ‘Graficar’.
- El resultado se mostrará a continuación.



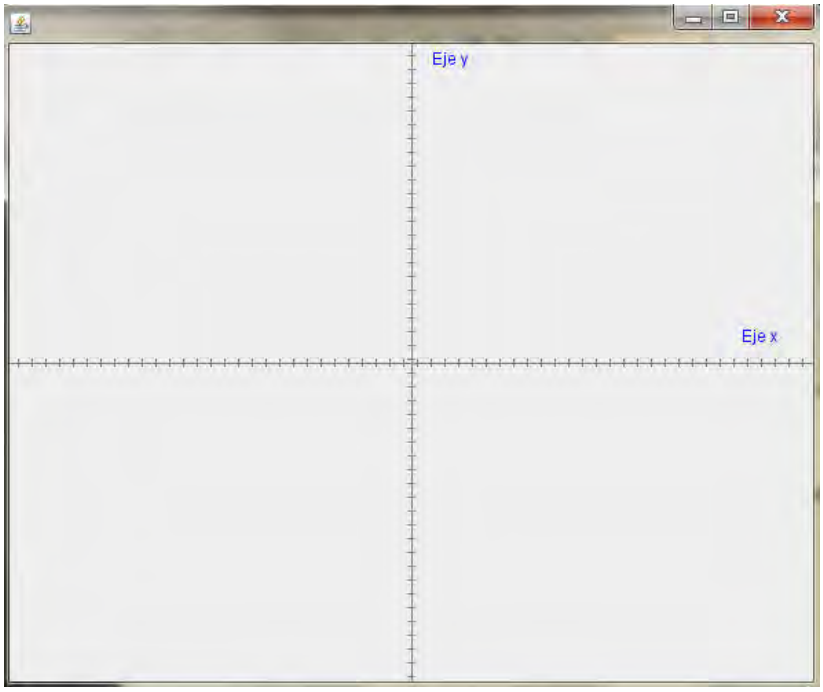
Modificación de parámetros

Una vez generada la gráfica, se puede modificar los valores de las escalas a través de los sliders de la sección escalas.



Limpiar

En caso de que desee ingresar otra función, presione el botón 'Limpiar'. Este botón reinicia los sliders, borra la gráfica previa y el campo de texto.


Eje y
Eje x

CAPITULO 19

COMPONENTES HDMI

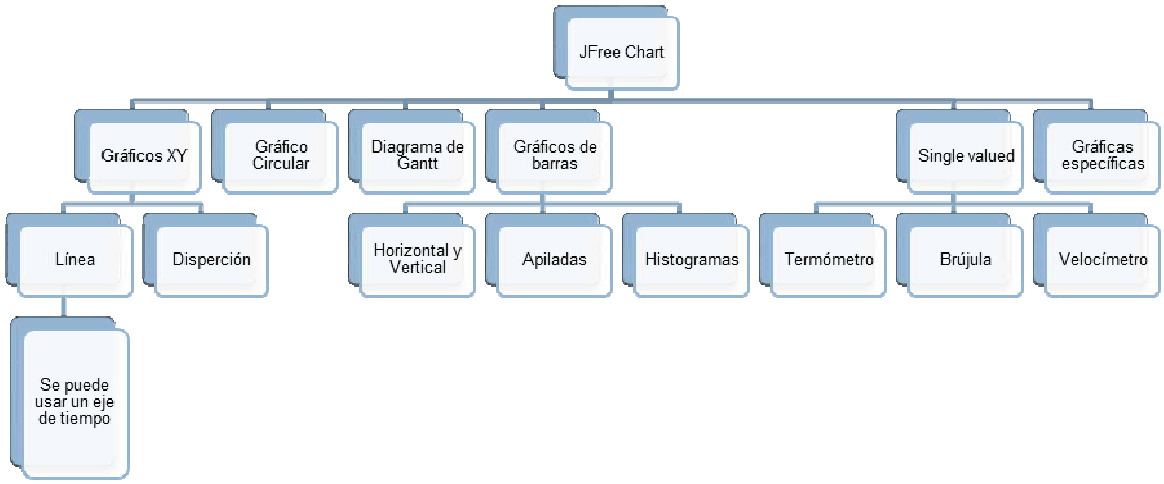
HMI (Human Machine Interface), hace referencia al proceso a través del cual las personas pueden intercambiar información con un computador; con el objetivo de que el intercambio sea más eficiente: minimizando errores, incrementando la satisfacción, disminuyendo la frustración y, en definitiva, haciendo más productivas las tareas que rodean a las personas y los computadores.

Es muy importante diseñar sistemas que sean efectivos, eficientes, sencillos y amenos a la hora de utilizarlos. Los diseñadores crean una interacción con mundos virtuales integrándolos con el mundo físico, es decir, que todo componente basado en la interface HMI debe ser lo más comprensible y preciso posible, para que represente un elemento determinado lo más cercano a la realidad. Estos componentes permiten al usuario un manejo mucho más ágil, ya que al representar objetos físicos, las funciones y las opciones que ofrecen estos componentes son prácticamente las mismas que se aplicarían en la realidad. Por lo que pueden ser aplicados a una infinidad de campos científicos.

Cuando un programa es desarrollado, la interacción de este con el usuario o potenciales usuarios del software es un aspecto muy importante a ser tomado en cuenta por el programador, para una persona será mucho más fácil entender o interactuar con un programa si su presentación es de forma gráfica en vez de textual. JFreeChart fue creado justamente por este motivo, para desarrollar interfaces gráficas que llamen la atención del usuario y provoquen en él una mejor comprensión de lo que está realizando. JFreeChart es un conjunto de clases que forman una librería gráfica con un manejo no tan complicado y que permite presentaciones de programa mucho más amigables con el usuario.

JFREECHART

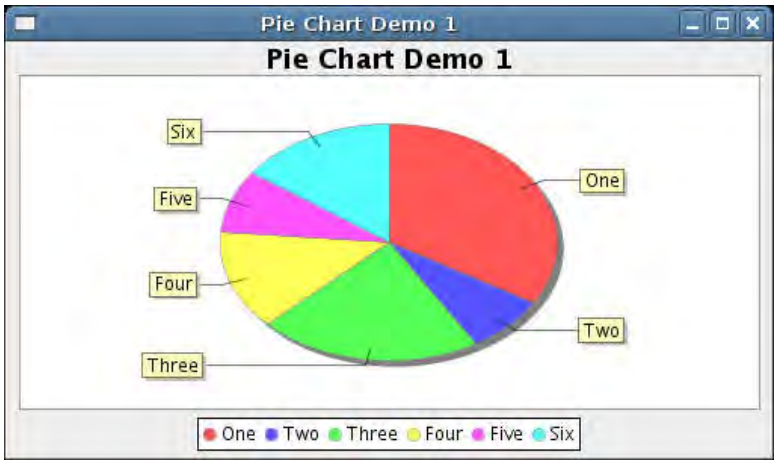
JFreeChart se desarrolla en el año 2000 como una librería de código abierto (*Open Source*) para Java, que permite crear diagramas de forma práctica y sencilla. Con JFreeChart es posible hacer diferentes tipos de gráficos que van desde los tipos comunes tales como gráficos circulares , gráficos de barras , áreas , gráficos de línea , histogramas, diagramas de Gantt y más específicos y menos frecuentemente utilizados como tipos *Candlestick* , Viento y *Wafer Map*.



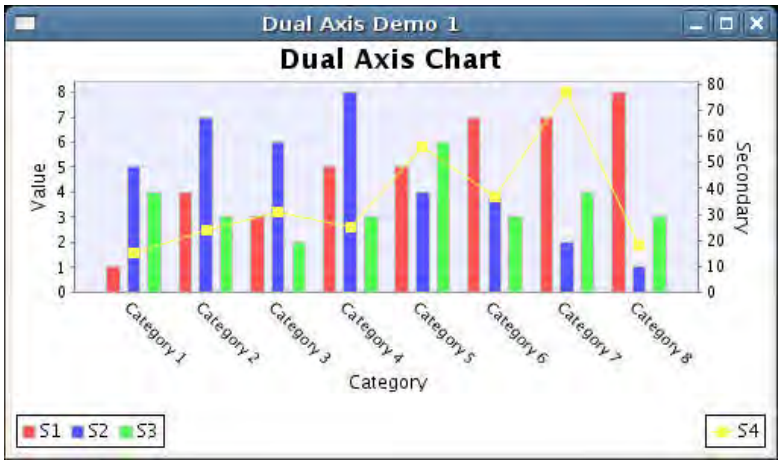
JFreeChart dibuja automáticamente las escalas de los ejes y leyendas. Con el ratón informático se puede hacer zoom en la interfaz de la gráfica automáticamente y cambiar algunos ajustes a través del menú local.

JFreeChart permite tener una amplia gama de aplicaciones, esto se da porque permite generar gráficas estadísticas de diferente índole las cuales pueden ser usadas dependiendo la necesidad del usuario. Entre las gráficas que se pueden generar con esta librería se encuentran:

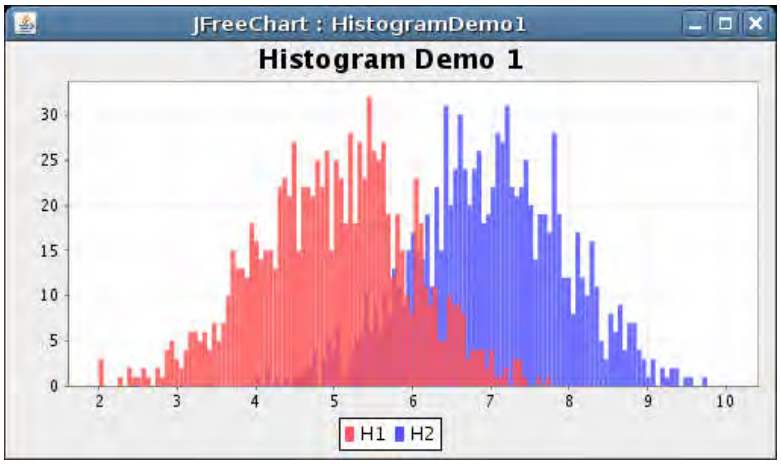
- Diagrama de pastel



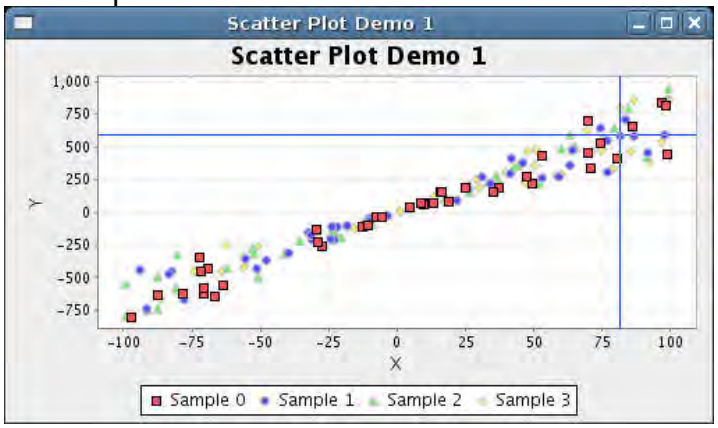
- Diagrama de barras



- Histogramas



- Diagramas de Dispersión



- Componentes HMI

Los componentes HMI que ofrece el JFreeChart, permiten un primer acercamiento al desarrollo de aplicaciones enfocadas al control de temperatura que en un momento determinado se integran con sus componentes de hardware, por medio de los puertos de la PC.

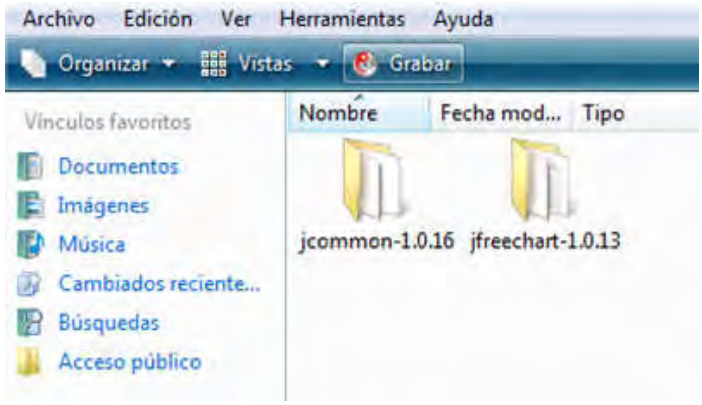
Termómetro como componente HMI

Dentro de la librería de JFreeChart existen varios componentes HMI, uno de ellos es el termómetro, este termómetro se genera mediante la clase "ThermometerPlot" la cual requiere un argumento de tipo ValueDataset, este tipo de dato es una interface que retorna un valor específico, en este caso será el valor que marcará el termómetro.

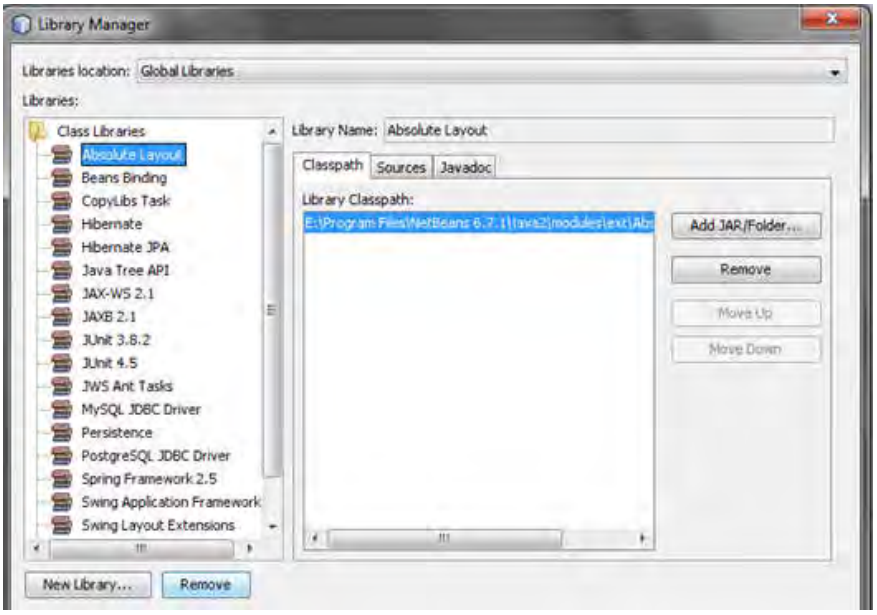
Al crear un objeto de tipo ThermometerPlot, es posible acceder a los diferentes métodos que este tiene, y de esta forma poder establecer el tamaño del termómetro, la posición, si se quiere marcar en °C o °F, el radio de la base del termómetro entre otros parámetros que pueden ser establecidos.

A continuación se indica los pasos para poder configurar JFreeChart en el IDE NetBeans:

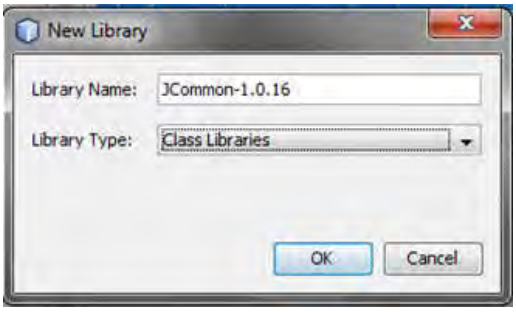
1. Una vez que se descarga el paquete JFreeChart y el paquete JCommon se lo descomprime en un directorio de la PC.

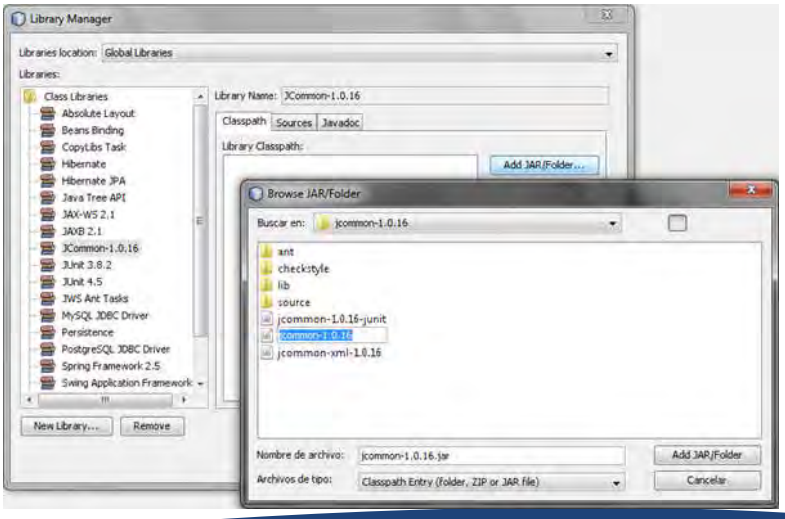


2. A continuación se abre NetBeans para configurar las librerías de JFreeChart y JCommon , por lo tanto en NetBeans se accede al menú Tools y se selecciona Libraries:

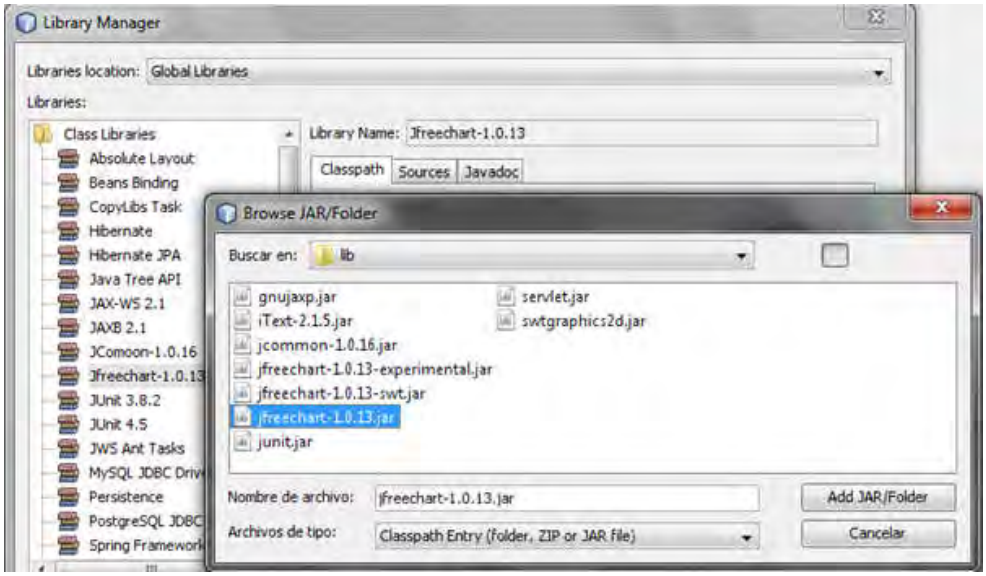


3. Luego se da un click en New Library y se escribe “JCommon-1.0.16” como nombre de la librería luego hay que dirigirse a la pestaña Classpath → Add Jar/Folder y luego buscar la dirección del JCommon descargado anteriormente y se selecciona el archivo JCommon-1.0.16.jar:

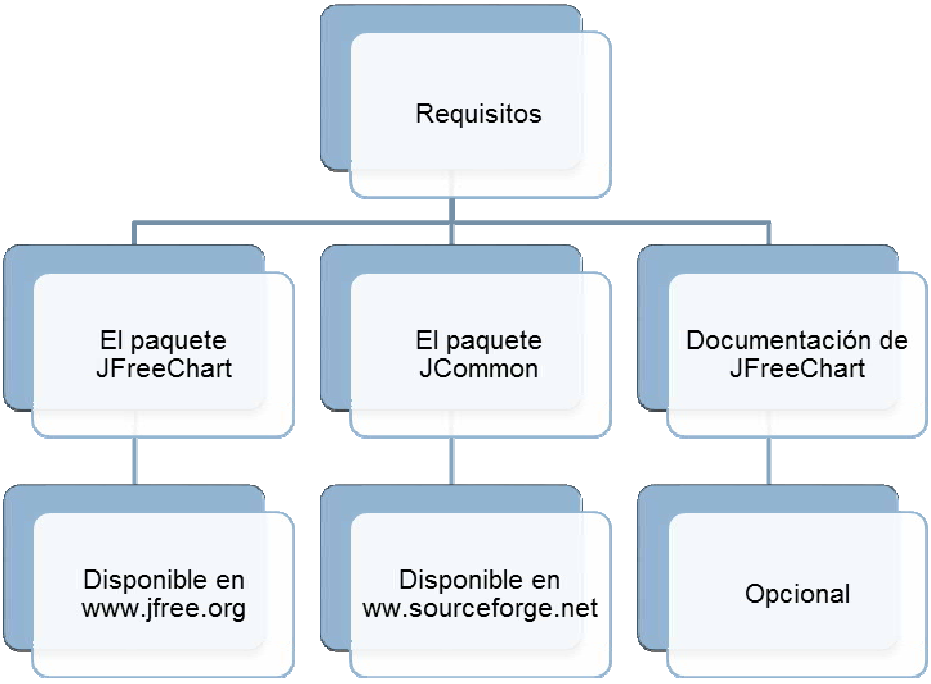




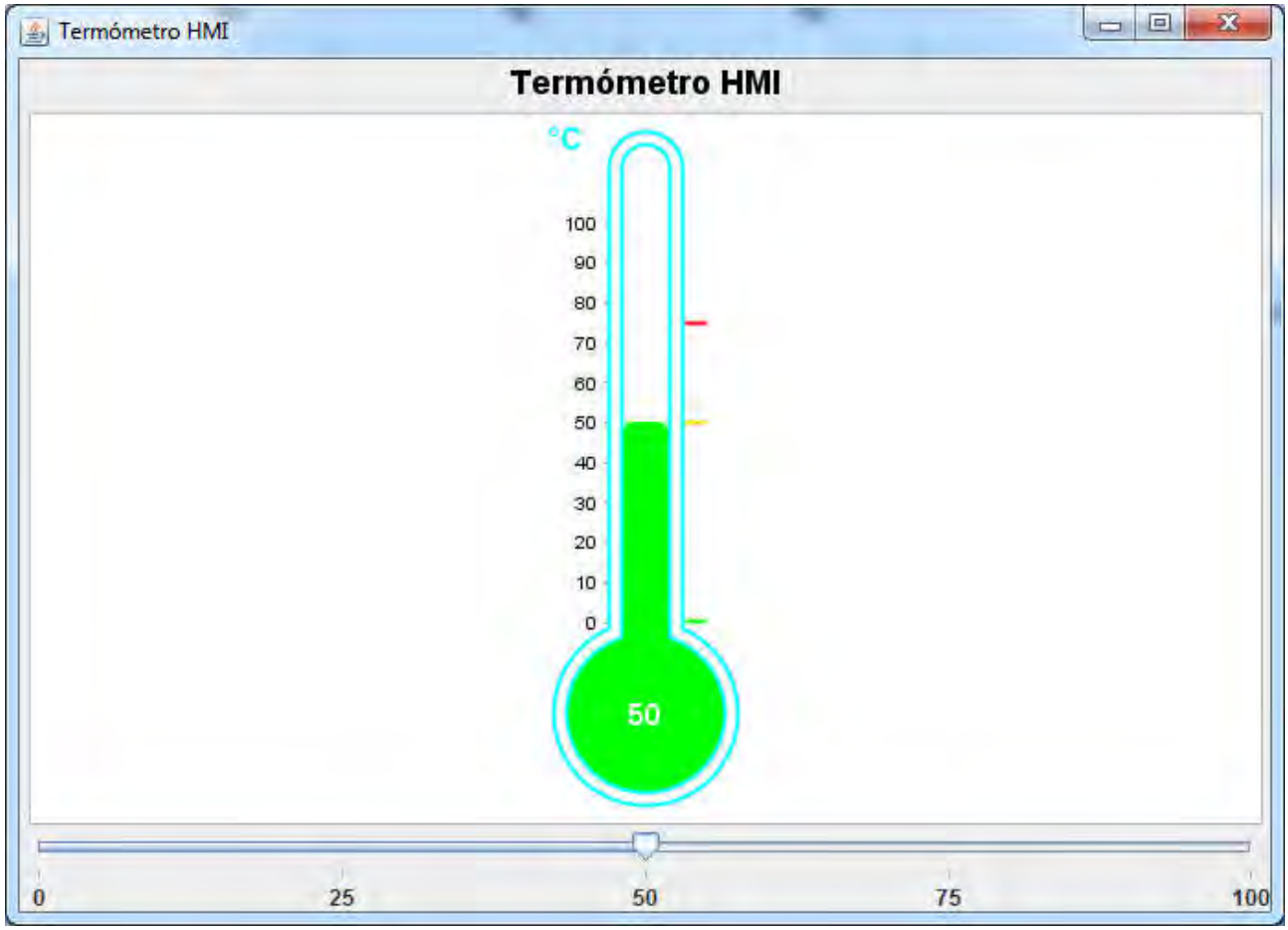
4. Se realiza un procedimiento similar con el .jar que se descargo del JfreeChart, Pero el jar que se selecciona en este caso es el de la carpeta lib.



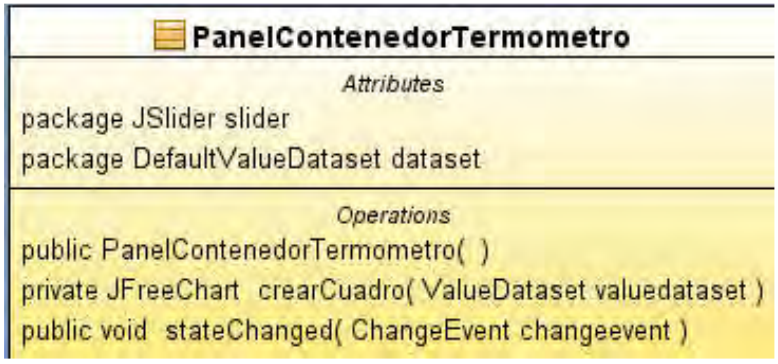
5. Hecho esto se tienen configuradas las librerías de JFreechart y JCommon en NetBeans.

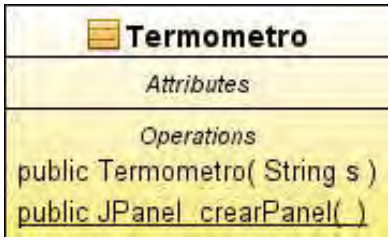


A continuación se ilustra una aplicación Java que permite controlar el HMI de un termómetro, donde se simula su comportamiento mediante un slider.



Diagramas UML.





Clase Main:

```
public class Main {

    public static void main(String[] args) {
        Termometro termometro = new Termometro("Termómetro HMI");
        termometro.setLocation(300, 200);
        termometro.pack();
        termometro.setVisible(true);
    }
}
```

Clase Termómetro:

```
import java.awt.Button;
import javax.swing.JPanel;
import org.jfree.ui.ApplicationFrame;


public class Termometro extends ApplicationFrame {

   public Termometro(String s) {
        super(s);
        JPanel jpanel = crearPanel();
        setContentPane(jpanel);
    }

    public static JPanel crearPanel() {
        return new PanelContenedorTermometro();
    }
}
```

Clase PanelContenedorTermómetro:

```
import java.awt.*;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.ThermometerPlot;
import org.jfree.data.general.DefaultValueDataset;
import org.jfree.data.general.ValueDataset;
import org.jfree.ui.RectangleInsets;

public class PanelContenedorTermometro extends JPanel implements ChangeListener {
        JSlider slider;
    DefaultValueDataset dataset;

    /**
```

```
     * Constructor del la clase PanelContenedor que permite crear el cuadro con
     * el slider y el diagrama del termómetro
     */

    public PanelContenedorTermometro() {


        super(new BorderLayout());

        slider.setPaintLabels(true);
        slider.setPaintTicks(true);
        slider.setMajorTickSpacing(25);
        slider.addChangeListener(this);
        add(slider, "South");
        dataset=new DefaultValueDataset(slider.getValue());
        add(new ChartPanel(crearCuadro(dataset)));
       }



    /**
     * Función que permite crear un diagrama de un termómetro a partir de un valor
     * en este caso del valor del slider obtenido de la función statechanged
     */
    private JFreeChart crearCuadro(ValueDataset valuedataset) {

        ThermometerPlot thermometerplot = new ThermometerPlot(valuedataset);
        JFreeChart jfreechart = new JFreeChart("Termómetro HMI",
JFreeChart.DEFAULT_TITLE_FONT, thermometerplot, true);
        //parámetros para el gráfico del termómetro
        thermometerplot.setInsets(new RectangleInsets(5, 5, 5, 5));// distancia de la ventana
al frame
        thermometerplot.setPadding(new RectangleInsets(10, 10, 10, 10));//distancia del
termometro al frame
        thermometerplot.setThermometerStroke(new BasicStroke(2F, 0, 2, 1.5F));// borde del
termometro
        thermometerplot.setThermometerPaint(Color.CYAN);//color del borde del termometro
        thermometerplot.setUnits(2);// UNIDADES DE MEDIDA ºC
        thermometerplot.setGap(7);// ANCHO DEL BORDE DEL TERMOMETRO
        thermometerplot.setBulbRadius(50);// radio de la base del termómetro
        thermometerplot.setMercuryPaint(Color.blue);// color en cero 0
        thermometerplot.setLowerBound(0);// desde donde empieza a marcar el termómetro
        thermometerplot.setRange(0, 100);// de donde a donde mide el termómetro
        return jfreechart;
    }

    public void stateChanged(ChangeEvent changeevent) {
        dataset.setValue(new Integer(slider.getValue()));
    }
```

CAPITULO 20

CONTROL DE 32 SALIDAS DE POTENCIA Y 5 ENTRADAS POR EL PUERTO PARALELO

Uno de los ambientes en los cuales sigue ingresando la PC es en el hogar, con la idea de la automatización o domótica, pero si consideramos que la mayor parte de los electrodomésticos que se tiene trabajan con 110 o 220 v, debe existir la forma de poder controlar por ejemplo luces o motores de apertura de puertas, con nuestra PC, y en particular con el puerto paralelo.

El problema es si es bastante elemental, primeramente se debe saber cómo escribir y cómo leer por el puerto paralelo, y después considerar, que se debe integrar la etapa de control con la de potencia.

PUERTO PARALELO

Las comunicaciones en paralelo se realizan mediante la transferencia simultánea de todos los bits que constituyen el dato (byte o palabra). En realidad, para la transferencia de las señales de datos y de control a través de la tarjeta de interface paralelo sólo se requieren 18 líneas, las restantes son líneas de masa que se enrollan alrededor de los cables de señal para proporcionarles apantallamiento y protección contra interferencias. Las líneas son latcheadas, esto es, mantienen siempre el último valor establecido en ellas mientras no se cambien expresamente y los niveles de tensión y de corriente coinciden con los niveles de la lógica TTL. Lógica TTL:

- Tensión de nivel alto: 5 V.
- Tensión de nivel bajo: 0 v.
- Intensidad de salida máxima: 26 mA.
- Intensidad de entrada máxima: 24 mA

El puerto consiste de tres registros de 8 bits ubicados en direcciones adyacentes del espacio de I/O de la PC. Los registros se definen relativos a la dirección de I/O base (variable IO Base) y son:

- Registro de datos.- Escribiendo un dato al registro, causa que el mismo aparezca en los pines 2 a 9 del conector del puerto (DB 25). Leyendo el registro, se lee el último dato escrito (NO lee el estado de los pines; para ello hay que usar un puerto bidireccional).
  Es el registro donde el procesador, en operaciones de salida (OUT), pone el dato que se quiere enviar y su dirección coincide con la dirección base del puerto paralelo (0x378 en LPT 1).

Nro.Bit 7	6	5	4	3	2	1	0	Descripción
x	.	.	.	.	.	.	.	D7 (pin 9), 1=Alto, 0=Bajo
.	x	.	.	.	.	.	.	D6 (pin 8), 1=Alto, 0=Bajo
.	.	x	.	.	.	.	.	D5 (pin 7), 1=Alto, 0=Bajo
.	.	.	x	.	.	.	.	D4 (pin 6), 1=Alto, 0=Bajo
.	.	.	.	x	.	.	.	D3 (pin 5), 1=Alto, 0=Bajo
.	.	.	.	.	x	.	.	D2 (pin 4), 1=Alto, 0=Bajo
.	.	.	.	.	.	x	.	D1 (pin 3), 1=Alto, 0=Bajo
.	.	.	.	.	.	.	x	D0 (pin 2), 1=Alto, 0=Bajo

- Registro de Estado.- La lectura da el estado de los cinco pines de entrada al momento de la lectura. Se trata de un registro de entrada (Lectura) de información, su dirección se obtiene sumando 1 a la dirección base del puerto (0x379 en LPT1).

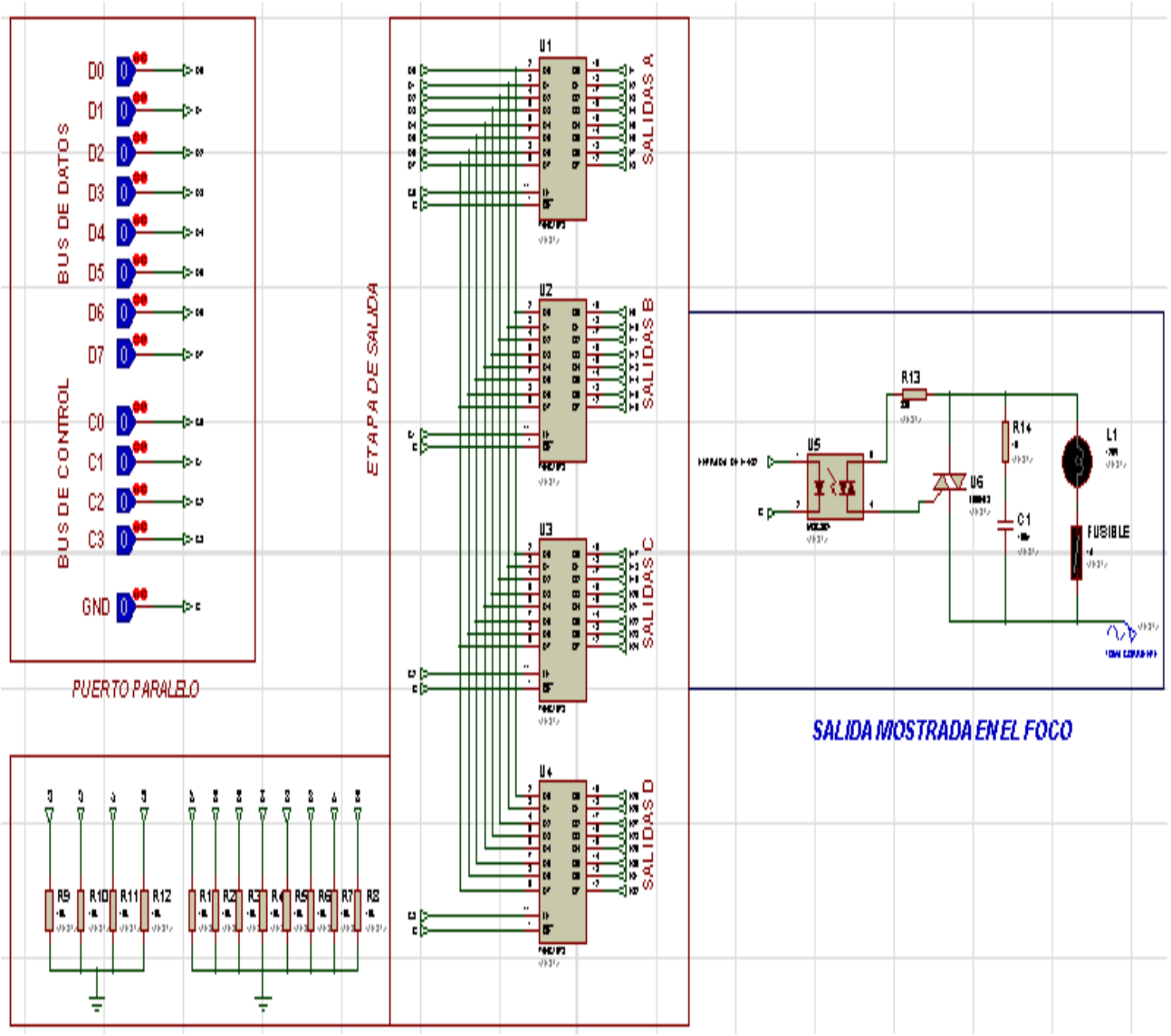
Nro.Bit 7	6	5	4	3	2	1	0	Descripción
x	.	.	.	.	.	.	.	S7 : Busy (pin 11), 0=Alto, 1=Bajo
.	x	.	.	.	.	.	.	S6 : Ack (pin 10), 1=Alto, 0=Bajo
.	.	x	.	.	.	.	.	S5 : No paper (pin 12), 1=Alto, 0=Bajo
.	.	.	x	.	.	.	.	S4 : Selected (pin 13), 1=Alto, 0=Bajo
.	.	.	.	x	.	.	.	S3 : Error (pin 15), 1=Alto, 0=Bajo
.	.	.	.	.	x	x	x	Sin definir

- Registro de control.- El registro de control permite controlar las transferencias de información, y puede ser escrito y leído. Es un registro de entrada/salida cuya dirección se obtiene sumando 2 a la dirección base del puerto (0x37A en L PT 1). Se usa principalmente para la generación de pulsos eléctricos que son usados como señales (1 y 0) que permiten ser manipuladas fácilmente.

Nro.Bit 7	6	5	4	3	2	1	0	
x	x	.	.	.	.	.	.	Sin usar
.	.	x	.	.	.	.	.	C5 : Control puerto bidireccional
.	.	.	x	.	.	.	.	C4 : Interrupt control, 1=enable, 0=disable
.	.	.	.	x	.	.	.	C3 : Select (pin 17), 1=Bajo, 0=Alto
.	.	.	.	.	x	.	.	C2 : Initialize (pin 16), 1=Alto, 0=Bajo
.	.	.	.	.	.	x	.	C1 : Auto Feed (pin 14), 1=Bajo, 0=Alto
.	.	.	.	.	.	.	x	C0 : Strobe (pin 01), 1=Bajo, 0=Alto

CONTROL DE POTENCIA POR PC (Ampliación a 32 canales por puerto paralelo).- Este circuito permite conectar hasta cuatro módulos de control de potencia a un mismo puerto

paralelo del PC. Dicho puerto no necesariamente debe ser bidireccional, por lo que cualquier PC por más antiguo que sea servirá para controlar este sistema.



Cada integrado es un latch octal; éstos sirven para retener un dato (presente en su entrada) en su salida solo cuando una señal específica se presente. Para hacerlo más simple: Los pines 2 al 9 de cada integrado son las entradas de datos, los pines 12 al 19 son las salidas, el pin 11 se denomina en inglés Latch Enable, una entrada de control que causa que los pines 12 al 19 reflejen el dato presente en los pines 2 al 9. Esto quiere decir, a su vez, que los datos presentes en las salidas del integrado no sufren cambios por más que los datos en la entrada del mismo cambien constantemente siempre y cuando la entrada de control (pin 11) esté a masa. Cuando esta entrada de control va a estado alto (a 5v), las salidas quedan conectadas con las entradas haciendo que lo presente en ellas quede reflejado en las salidas. Si dicho terminal de control (pin 11) se mantiene alto y el dato presente en las entradas cambia, el presente en las salidas cambiará también. Como se ve, las entradas de datos de los cuatro integrados están unidas en paralelo. Esto quiere decir que el dato presente en los pines 2 al 9 del puerto

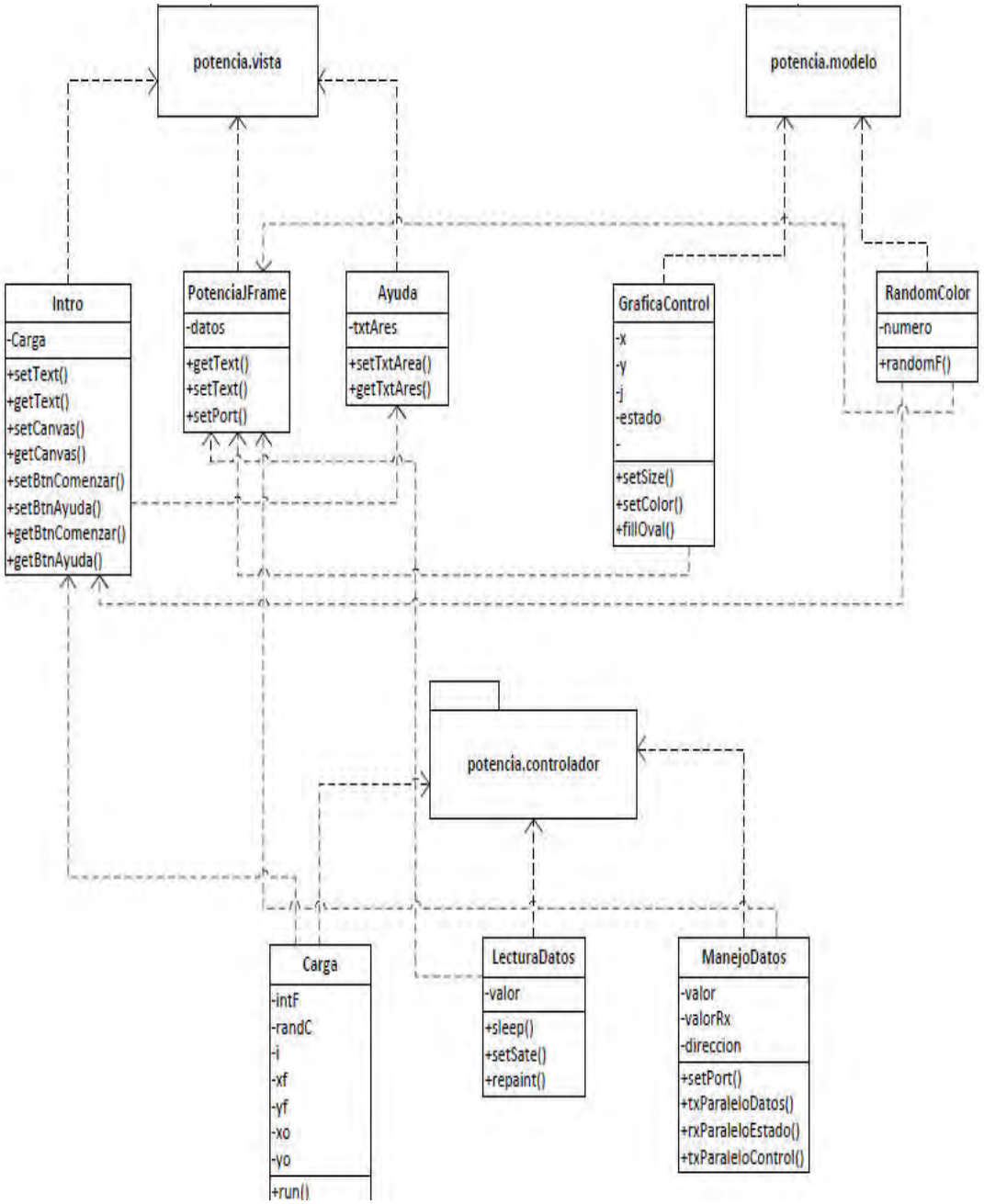
paralelo de la PC (los datos presentes en el bus de datos del puerto paralelo) estará presente en los cuatro integrados al mismo tiempo. Para que un dato presente en el puerto paralelo del PC solo vaya a modificar un grupo de salidas y no los cuatro , usamos los pines de control para determinar cual o cuales integrados deseamos accionar. Esto se logra gracias a que cada pin de control del puerto paralelo maneja solo un integrado. De esta forma logramos controlar 32 salidas independientes (en grupos de 8 salidas por activación).

Como no se sabe en qué estado se encuentra el puerto paralelo y, por ende, la placa de expansión, lo primero que tiene que hacer el software es inicializar el circuito. Para eso, pone en 0 el bus de datos del puerto paralelo, con lo que todos sus pines (del 2 al 9) quedan a masa. Luego, espera 10mS para que el dato se establezca en las entradas de los integrados. Luego de transcurridos los 10mS se activan los cuatro integrados poniendo altos los cuatro pines de control del puerto paralelo. Seguidamente se espera otros 10mS para que los latches retengan los datos en las salidas y por último se pone en bajos (en cero) todos los pines de control del puerto paralelo con lo que los integrados dejan en las salidas todas los pines apagados (a masa) sin importar el dato que aparezca en sus entradas de ahora en masa. Con esto el módulo quedará inicializado y todas las salidas apagadas.

Cuando se quiera modificar el estado de un grupo de salidas (cada grupo es de ocho salidas y están indicados como Salidas A, Salidas B, Salidas C, Salidas D), se deberá poner en el puerto paralelo (en el bus de datos de éste) el dato que se desea colocar en la salidas del integrado. Luego esperar 10mS para que el dato se establezca correctamente en las entradas de los integrados. Luego poner en alto (en uno) la salida de control del puerto paralelo que comande el integrado que se desea modificar y esperar otros 10mS para que el dato se fije correctamente en los latches de salida del mismo. Transcurrido este tiempo volver a bajar (poner a cero) la salida de control que se subió y el proceso habrá concluido. Es recomendable que, tanto la rutina de inicialización como la de control, esperen 10mS luego de terminar de ejecutarse, a fin de dar un tiempo entre cada ejecución para evitar posibles fallas de activación.

Otro factor muy importante a tener en cuenta es que algunos de los pines de control del puerto paralelo presentan un estado lógico invertido con respecto a la tensión. Esto quiere decir que, un pin con estado lógico normal presenta tensión cuando el bit que lo controla está a 1 y está a masa cuando su bit se pone en cero. Pero, un pin con lógica inversa, presentará tensión cuando su bit esté en cero y masa cuando esté en uno. Hay que prestar atención a esto para evitar problemas de control con los integrados o activaciones erráticas.

DIAGRAMA UML



MAPA DE VARIABLES

Variable	Tipo	Función
Int datos[]	Private	Arreglo que contiene los focos que se va a encender o apagar
Int x	Private	Variable que contiene el tamaño del lienzo en x
Int y	Private	Variable que contiene el tamaño del lienzo en y
Int j	Private	Indica la posición del led

		que cambiará de color
Short Estado	Private	Indica el estado del led si está encendido o apagado
Short valor	Private	Indica el valor que se enviará al bus de datos, control o estado.
Short valorRx	Private	Indica el valor de cambio de estado de un led
Short direccion	Private	Contiene la dirección del puerto

Clase PotenciaJFrame

```
private void controlFoco1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (controlFoco1.getText().equals("Off")) {
        controlFoco1.setText("On");
        tarjeta.txParaleloControl(10);
        datos[0] |= 1;
        tarjeta.txParaleloDatos(datos[0]);
    } else {
        controlFoco1.setText("Off");
        tarjeta.txParaleloControl(10);
        datos[0] &= 0xfffe;
        tarjeta.txParaleloDatos(datos[0]);
    }

}
```

Este método es el que cambia la etiqueta del botón al cual se le hace click de on a off, o viceversa, en el caso de on se envía un 1 y en el caso de off se envía un 0, además de esto se envía las señales adecuadas al bus de control para habilitar el latch que corresponde al foco.

Clase GraficaControl

```
public GraficaControl(int x, int y) {
    this.setSize(500, 500);
    this.x = x;
    this.y = y;
}
```

Este constructor de la clase inicializa los valores correspondientes al tamaño en x e y de la gráfica para el control.

```
public void paint(Graphics g) {
```

```
        int j = 1;
        for (int i = 0; i < 5; i++) {
            if ((estado & j) != 0) {
                g.setColor(Color.blue);
            } else {
                g.setColor(Color.yellow);
            }
            j *= 2;
            g.fillOval((x - i * 40), y+30, 30, 30);
        }
    }
```

Este método sirve para seleccionar el color ya sea azul o amarillo dependiendo del estado del led (entrada de control), si se ha ingresado un 1 es de color azul , caso contrario es de color amarillo.

```
    public void setState(short valor) {
        estado = (short) valor;
    }
```

Este método ayuda a determina el estado actual del led.

Clase ManejoDatos

```
    public void setPort(int puerto) {
        direccion = (short) puerto;
        System.out.println("Dirección del puerto establecida a: " + puerto);
    }
```

Este método determina la dirección del puerto que se va a usar.

```
    public void txParaleloDatos(int dato) {
        try {
            valor = (short) dato;
            puerto.output(direccion, valor);
            System.out.println("Datos.- Se ha enviado: " + valor + "\n");
        } catch (Exception e) {
        }
    }
```

Este es un método que indica qué dato se ha enviado al puerto paralelo.

```
    public short rxParaleloEstado() {
        try {
            valor = puerto.input((short) (direccion + 1));
            valor /= 8;
            if (valor >= 16) {
                valor -= 16;
```

```
        } else {
            valor += 16;
        }
        if (valor != valorRx) {
            System.out.println("Estado.- Se ha leído: " + valor + "\n");
        }
        valorRx = valor;
    } catch (Exception e) {
    } finally {
        return (valor);
    }
}
```

Este método selecciona la dirección del puerto paralelo e invierte la entrada del puerto paralelo y además indica el estado en el cual se encuentra.

```
public void txParaleloControl(int dato) {
    try {
        valor = (short) dato;
        puerto.output((short) (direccion + 2), valor);
        System.out.println("Control.- Se ha enviado: " + valor);
    } catch (Exception e) {
    }
}
```

Este en cambio selecciona la dirección del puerto paralelo e indica cuál es el dato enviado al bus de control.

Clase LecturaParalelo

```
public void run() {
    while(true) {
        try {
            Thread.sleep(50);
        } catch (InterruptedException ex) {
            System.out.println("Error al ejecutar el timer: " + ex);
        }
        valor = tarjeta.rxParaleloEstado();
        focos.setState(valor);
        focos.repaint();
    }
}
```

Este método comprueba cada 50 milisegundos si el valor de un led ha cambiado o no, cambia su estado y vuelve a pintarlo de acuerdo a su estado.

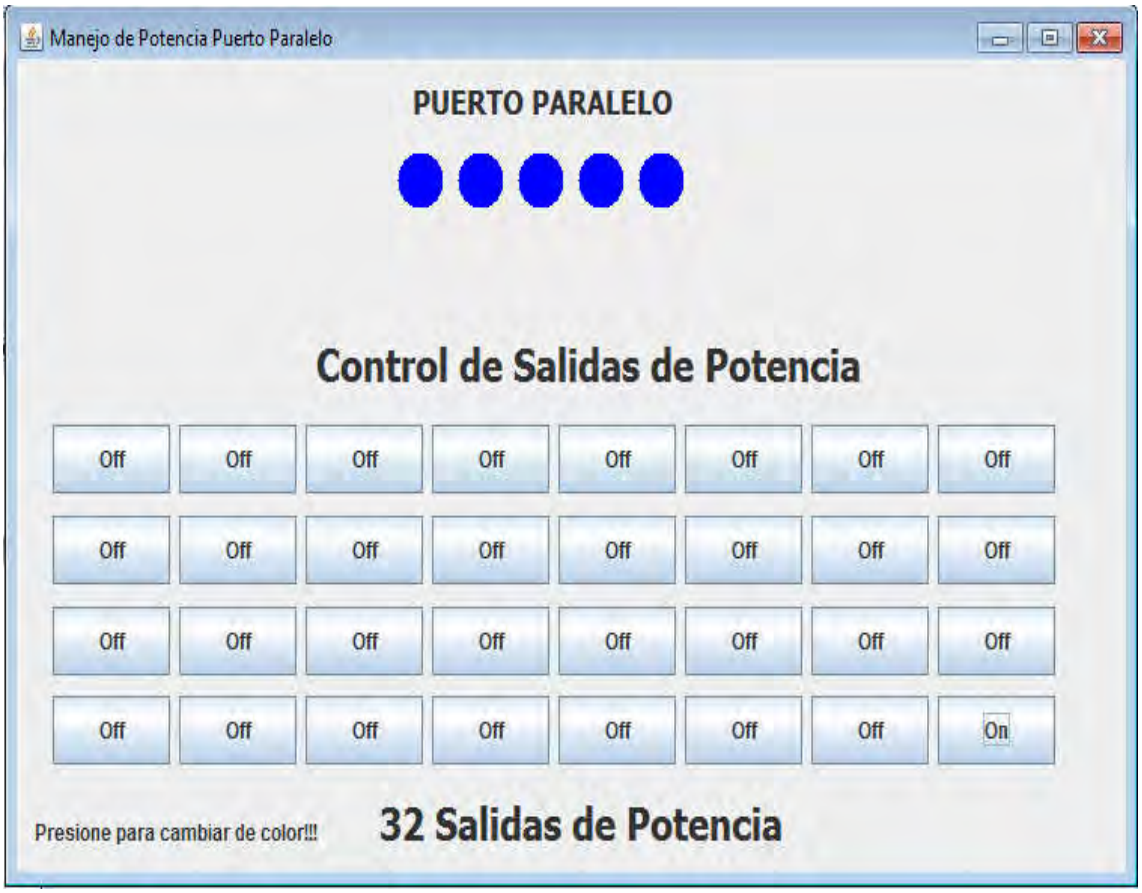
Clase Carga

```
public void run() {
    this.canvas = intF.getCanvas();
```

```
    xf = canvas.getWidth();
    yf = canvas.getHeight();
    xo = canvas.getX();
    yo = canvas.getY();
    intF.getBtnAceptar().setEnabled(false);
    intF.getBtnAyuda().setEnabled(false);
    Graphics g = intF.getCanvas().getGraphics();
    while (true) {
        try {
            this.sleep(10);
            Color col = new Color(randC.RandomF(), randC.RandomF(), randC.RandomF());
            g.setColor(col);
            if ((xf - 50) == i) {
                intF.getBtnAceptar().setEnabled(true);
                intF.getBtnAyuda().setEnabled(true);
                this.stop();
            } else {
                g.fillRect(25, 50, i, 50);
                g.draw3DRect(25, 50, i, 50, true);
                g.drawOval((xo + randC.RandomF() + 50), (yo + randC.RandomF() - 50), 10, 5);
                g.fillOval((xo + randC.RandomF() + 50), (yo + randC.RandomF() - 50), 5, 10);
            }
        } catch (InterruptedException ex) {
            Logger.getLogger(Carga.class.getName()).log(Level.SEVERE, null, ex);
        }
        i++;
    }
}
```

Este método es un **Override** del método run, sirve para mostrar una barra de carga al ejecutar el programa.

En la interfaz que se presenta se muestran las 32 salidas, en donde la primera fila representa al primer circuito integrado 74573, la segunda fila al segundo y así sucesivamente. Los botones representan cada salida y al presionarlos se verá que se ilumina su correspondiente led(Esto ya se refiere a la parte del circuito).



Cabe mencionar en este punto que para probar que las 5 señales de ingreso se lo puede realizar con un dipswitch.

CAPITULO 21

CONTROL DE UN MODULO LCD POR EL PUERTO PARELO

La aparición de la IBM PC, permitió la masificación de las computadoras, llegando a todos los ámbitos de nuestra vida, para lo cual incorporó una serie de periféricos como el puerto paralelo, que permitía que se incorporen distintos accesorios según las necesidades de cada usuario. Lo que se ilustra a continuación es el control de un módulo LCD 2x16, muy utilizado en la ingeniería electrónica para presentar información al usuario.

LCD (LIQUID CRYSTAL DISPLAY)

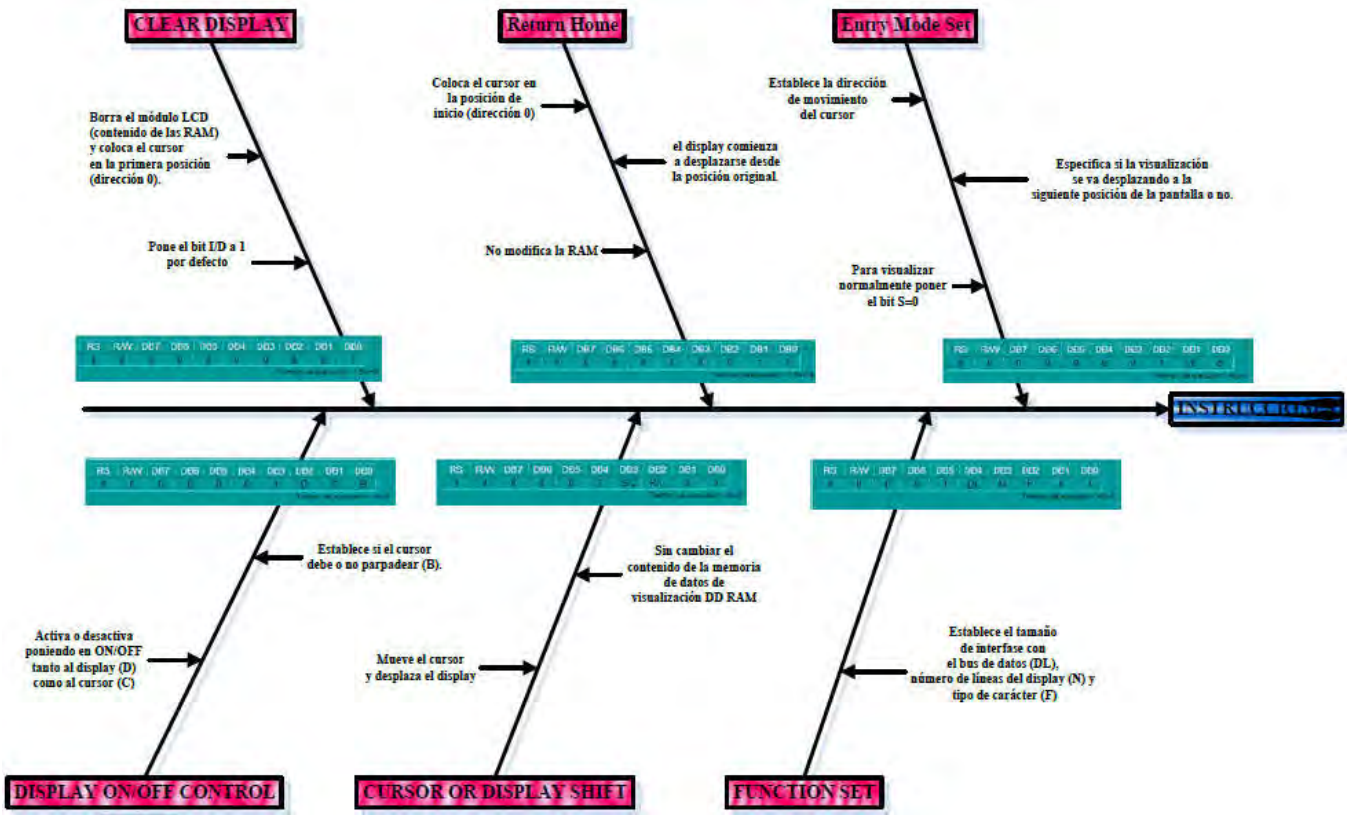
Una pantalla de cristal líquido o LCD, es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora. Tiene la capacidad de mostrar cualquier carácter alfanumérico, la pantalla consta de una matriz de caracteres (normalmente de 5x7 o 5x8puntos) distribuidos en una, dos, tres o cuatro líneas de 16 hasta 40 caracteres cada línea. El proceso de visualización es controlado por un microcontrolador, siendo el Hitachi 44780 el más utilizado.

A continuación se presenta la descripción de señales empleadas por el módulo LCD así como el número de pin a la que corresponden**.** En los módulos de 16 pines, los pines 15 y 16 corresponden a la intensidad del backlight del LCD.

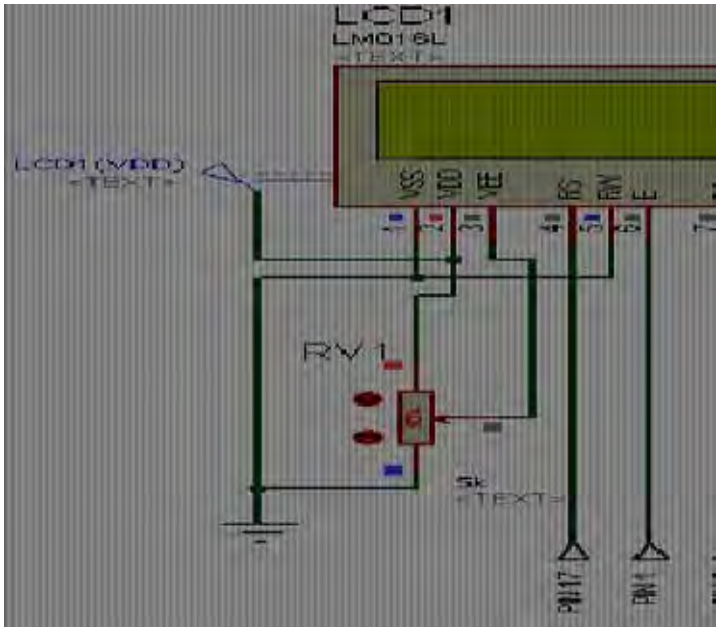
PIN Nº	SÍMBOLO	DESCRIPCIÓN
1	V_{SS}	Patilla de tierra de alimentación
2	V_{DD}	Patilla de alimentación de 5 V
3	V_O	Patilla de contraste del cristal líquido.Normalmente se conecta a un potenciómetro a través del cual se aplica una tensión variable entre 0 y +5V que permite regualr el contraste del cristal líquido.
4	RS	Selección del registro de control/registro de datos: RS=0 Selección del registro de control RS=1 Selección del registro de datos
5	R/W	Señal de lectura/escritura R/W=0 El módulo LCD es escrito R/W=1 El módulo LCD es leido
6	E	Señal de activación del módulo LCD: E=0 Módulo desconectado E=1 Módulo conectado
7-14	D0-D7	Bus de datos bi-direccional. A través de estas líneas se realiza la transferencia de información entre el módulo LCD y el sistema informático que lo gestiona

El LCD dispone de una zona de memoria interna no volátil llamada CGROM donde se almacena una tabla con los 192 caracteres que pueden ser visualizados. Cada uno de los caracteres tiene su representación binaria de 8 bits. Para visualizar un carácter debe recibir por el bus de datos el código correspondiente.

El modulo LCD cuenta con un set de instrucciones para realizar diferentes acciones en el módulo.



Par la conexión del puerto paralelo con el LCD 2x16, se puede utilizar el siguiente esquema.



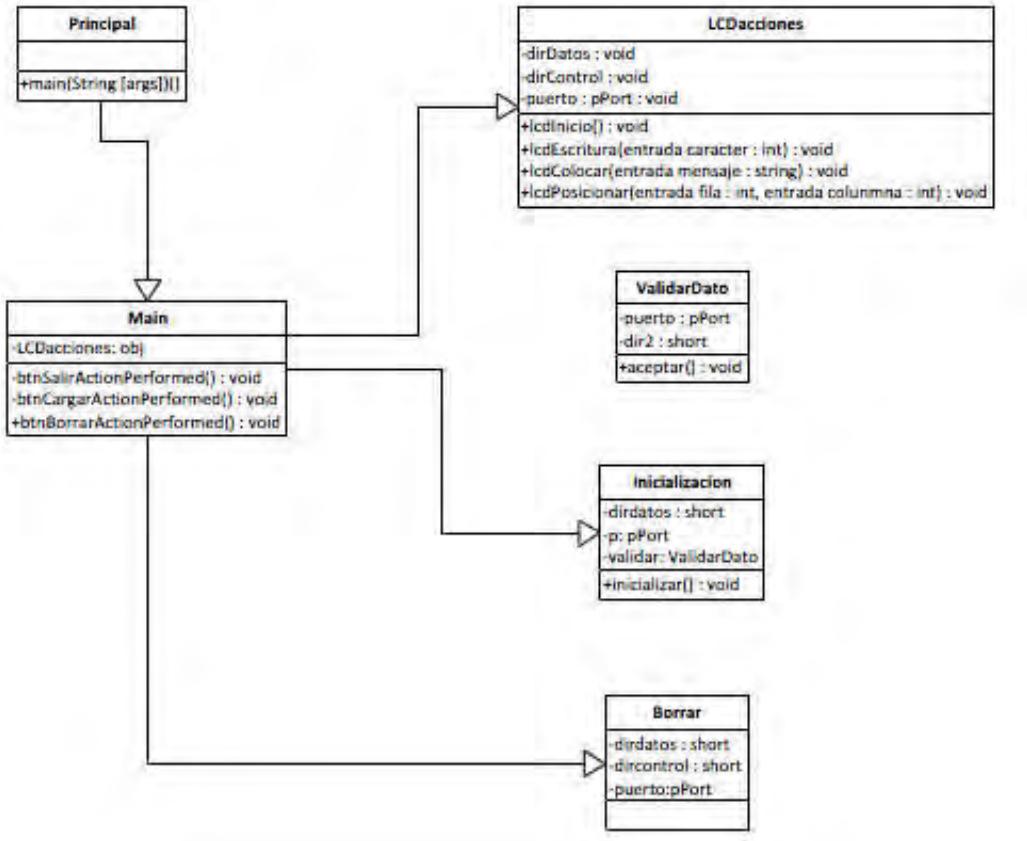
Pin LCD		Pin del paralelo	
1	GND		GND
2	Vcc		Vcc
3	Contraste		Potenciómetro de 5[kΩ]
4	RS	17	C3 (L)
5	R/W		GND
6	Enable	1	C0 (L)
7	D0	2	D0
8	D1	3	D1
9	D2	4	D2
10	D3	5	D3
11	D4	6	D4
12	D5	7	D5
13	D6	8	D6
14	D7	9	D7
15	Backligth +		Vcc
16	Backligth -		Resistencia de 100 [Ω]

Al iniciar el programa se desplegará una ventana que contiene las diferentes opciones a realizarse en una serie de botones:

La ventana cuenta con cuatro campos de texto modificables, en los campos ubicados a la derecha del frame se ingresarán los mensajes que se quieran visualizar en la correspondiente línea del módulo LCD. En los dos campos de texto ubicados a la derecha del frame se ingresará el desplazamiento horizontal que queremos dar al mensaje ingresado en la correspondiente línea, es importante que el mensaje ingresado

en cada línea no supere los 16 caracteres contando los de desplazamiento si esto ocurre solo se visualizarán los caracteres que estén dentro de este rango.

El programa consta de 5 clases tal como se ilustra en el diagrama de UML.



A continuación se describen las principales variables utilizadas:

Paquete Controlador	
Class Borrar (public)	*short dirdatos:* Guarda la dirección de los datos. *short dircontrol:* Guarda la dirección del registro de control. *pPort puerto:* Un objeto que simula el puerto paralelo.
Class Inicializacion (public)	*short dirdatos:* Guarda la dirección de los datos. *pPort P:* Un objeto que simula el puerto paralelo. *validarDato validar:* Valida la palabra de control para que se pueda escribir en el LCD.
Class LCDacciones (public)	*short dirDatos:* Guarda la dirección de salida. *short dirControl:* Guarda la dirección de conrol. *pPort puerto:* Un objeto que simula el puerto paralelo.
Class ValidarDato (public)	*pPort puerto:* Un objeto que simula el puerto paralelo. *short dir2:* tiene la dirección de control del puerto.

Paquete Vista	
Class Main (public)	*LCDacciones obj:* Objeto que inicializa los métodos que posicionan el cursor, escriben y selecciona la fila.

Clase Borrar.

En esta clase lo que se realiza es el envío de las palabras de control necesarias para borrar el LCD mediante las instrucciones output.

```
import jnpout32.*;

public class Borrar {

    private short dirdatos;
    private short dircontrol;
    private pPort puerto;

    public Borrar(){
        dirdatos = 0x378;
        dircontrol = 0x37a;
        puerto = new pPort();
        try{
            puerto.output(dirdatos,(short)1);
            puerto.output(dircontrol,(short)0xc);
            Thread.sleep(1);
            puerto.output(dircontrol, (short)0xa);
            Thread.sleep(1);
            puerto.output(dircontrol, (short)0xb);
            Thread.sleep(5);
```

```
        }catch (Exception e){}
    }
}
```

Clase Inicialización.

Esta clase sirve para inicializar el LCD, y setear el LCD en modo que pueda leer y escribir datos.

```
import jnpout32.*;

public class Inicializacion {

    private short dirdatos = 0x378;
    private pPort P = new pPort();
    private ValidarDato validar = new ValidarDato();

    public void inicializar(){
        try{
            Thread.sleep(20);
            try{
            P.output(dirdatos, (short)0x30);

            validar.aceptar();
            }catch(Exception e){}
            Thread.sleep(5);
            try{
            P.output(dirdatos, (short)0x30);

            validar.aceptar();
            }catch(Exception e){}
            Thread.sleep(1);
            try{
            P.output(dirdatos, (short)0x30);

            validar.aceptar();
            }catch(Exception e){}
            try{
            P.output(dirdatos, (short)0x38);

            validar.aceptar();
            }catch(Exception e){}
            try{
            P.output(dirdatos, (short)0x08);

            validar.aceptar();
            }catch(Exception e){}
            try{
```

```
            P.output(dirdatos, (short)0x0c);

            validar.aceptar();
            }catch(Exception e){}
            try{
            P.output(dirdatos, (short)0x06);

            validar.aceptar();
            }catch(Exception e){}
        }
        catch(Exception e){}

    }

}
```

Clase LCD acciones.

En esta clase se implementa todos los métodos necesarios para escribir un String en el LCD, algunos de los métodos son privados para que solo se tenga acceso a ellos desde la clase.

```
import java.util.logging.Level;
import java.util.logging.Logger;
import jnpout32.*;

public class LCDacciones {

    private short dirDatos = 0x378;
    private short dirControl = 0x37a;
    private pPort puerto = new pPort();

    /**
     * Método para inicializar el modo de escritura
     */

    public void lcdInicio() {
        try {
            puerto.output(dirControl, (short) (puerto.input(dirControl) & 0xDF));
            puerto.output(dirControl, (short) (puerto.input(dirControl) | 0x08));
            lcdEscritura(0x0f);
            Thread.sleep(20);
            lcdEscritura(0x01);
            Thread.sleep(20);
            lcdEscritura(0x38);
            Thread.sleep(20);
        } catch (InterruptedException ex) {
            Logger.getLogger(LCDacciones.class.getName()).log(Level.SEVERE, null, ex);
```

```
        }
    }

    /**
     * Método para escribir en el LCD
     * @param caracter : int
     */

    private void lcdEscritura(int caracter) {
        try {
            puerto.output(dirDatos, (short) caracter);
            puerto.output(dirControl, (short) (puerto.input(dirControl) | 0x01));
            Thread.sleep(2);
            puerto.output(dirControl, (short) (puerto.input(dirControl) & 0xFE));
            Thread.sleep(2);
        } catch (InterruptedException ex) {
            Logger.getLogger(LCDacciones.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * Método para convertir un String en codigo ascii
     * @param mensaje : String
     */

    public void lcdColocar(String mensaje) {
        puerto.output(dirControl, (short) (puerto.input(dirControl) & 0xF7));
        for (int i = 0; i < mensaje.length(); i++) {
            int ascii = String.valueOf(mensaje.charAt(i)).hashCode();
            lcdEscritura(ascii);
        }
    }

    /**
     * Método para posicionar el cursor en el LCD
     * @param fila : int
     * @param columna : int
     */

    public void lcdPosicionar(int fila, int columna) {
        puerto.output(dirControl, (short) (puerto.input(dirControl) | 0x08));
        if (fila == 2) {
            columna += 0x40;
        }
        lcdEscritura(0x80 | columna);
    }
}
```

Clase Validar Dato.

Esta clase la utilizamos para setear el LCD en modo de escritura, de esta manera podemos realizar la escritura del String en el LCD.

```
import jnpout32.*;

public class ValidarDato {

    private pPort puerto = new pPort();
    private short dir2 = 0x37a;

    /*
     * Método para ingresar la palabra de control
     */

    public void aceptar(){
            try{
                Thread.sleep(1);
                puerto.output(dir2, (short)0x0a);
                Thread.sleep(1);
                puerto.output(dir2, (short)0x0c);
            }catch(Exception e){}
    }
}
```

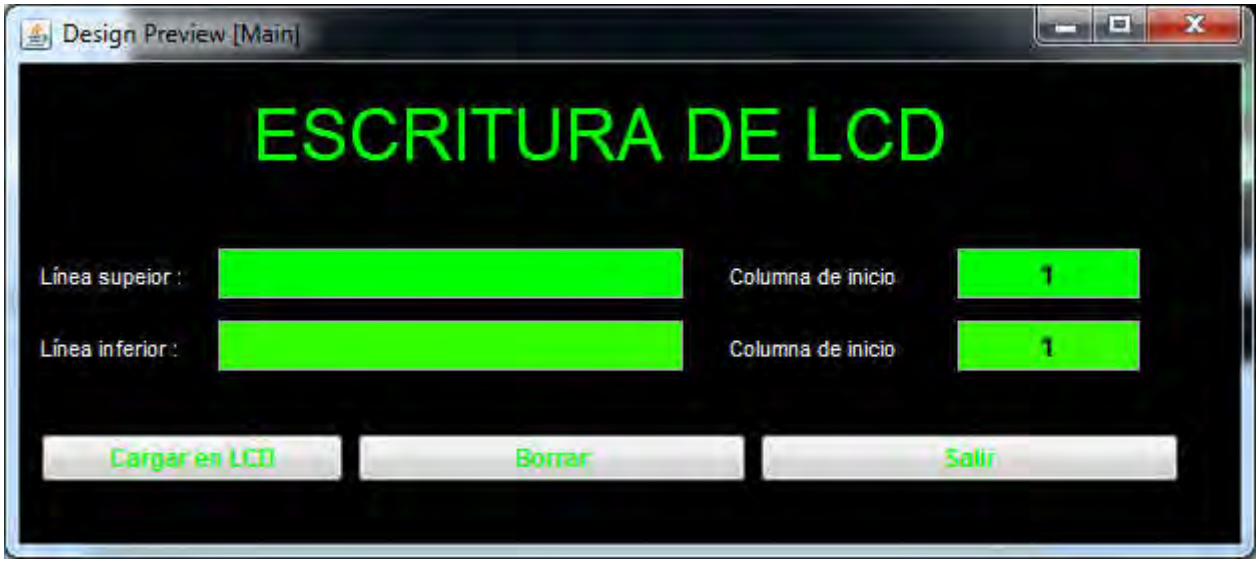
Clase Principal.

En esta clase se encuentra el método principal y es desde donde se ejecuta la aplicación.

```
import java.awt.Color;

public class Principal {

    public static void main(String[] args) {
        Main m = new Main();
        m.setLocation(300, 300);
        m.setBackground(Color.black);
        m.setVisible(true);
    }
}
```

Clase Main.

En esta clase se define la interfaz con la que el usuario interactúa con el programa y controla el LCD.




```
public class Main extends javax.swing.JFrame {

    /** Creates new form Main */
    LCDacciones obj;
    public Main() {
        initComponents();
        new Borrar();
        new Inicializacion();
        obj=new LCDacciones();
    }

    private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
        new Borrar();
        System.exit(0);
    }

    private void btnCargarActionPerformed(java.awt.event.ActionEvent evt) {
        obj.lcdInicio();
        obj.lcdPosicionar(1, Integer.parseInt(txtColumnaSup.getText())-1);
        obj.lcdColocar(txtSuperior.getText());
        obj.lcdPosicionar(2, Integer.parseInt(txtColumnaInf.getText())-1);
        obj.lcdColocar(txtInferior.getText());
    }

    private void btnBorrarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        new Borrar();
        txtColumnaInf.setText("1");
        txtColumnaSup.setText("1");
        txtSuperior.setText(null);
        txtInferior.setText(null);
```

```
    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Main().setVisible(true);
            }
        });
    }

    private javax.swing.JButton btnBorrar;
    private javax.swing.JButton btnCargar;
    private javax.swing.JButton btnSalir;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JPanel panelContenedor;
    private javax.swing.JTextField txtColumnaInf;
    private javax.swing.JTextField txtColumnaSup;
    private javax.swing.JTextField txtInferior;
    private javax.swing.JTextField txtSuperior;
}
```

Para el manejo de esta aplicación con interfaz gráfica se debe:

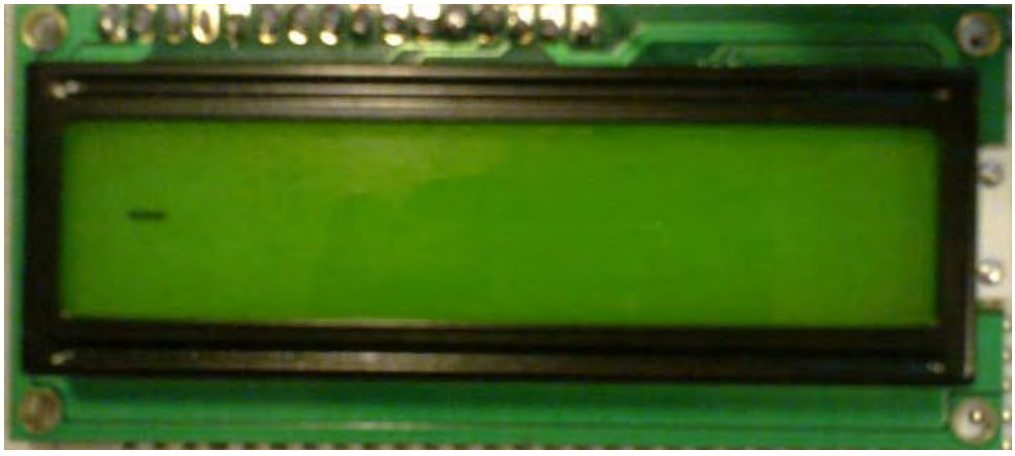
1. Posicionar el cursor sobre la opción deseada.
2. Hacer “Click”
3. Modificar los parámetros de las nuevas ventanas acorde a la opción seleccionada

Solo los campos de color verde serán modificables, en los dos campos ubicados a la derecha de la ventana se ingresarán los mensajes que se visualizarán en el módulo LCD, cada uno de ellos representará una de las líneas del módulo, la interfaz le dice al usuario

en qué línea se visualizará dependiendo del campo en el que escriba el mensaje. En los otros dos se debe ingresar un valor número entero, estos valores corresponden al desplazamiento horizontal de izquierda a derecha en la línea correspondiente en el módulo LCD.

Al dar click en el botón "*Cargar en LCD*" los mensajes escritos en las líneas se visualizarán en el módulo, incluyendo los desplazamientos ingresados.

Al dar click en la opción "Borrar", el LCD y los campo de texto se limpian.



CAPITULO 22

CONTROL DE UN SERVOMOTOR CON EL PUERTO PARALELO

Una aplicación a más de mostrar información, la cual es importante, también en algún momento puede necesitar interactuar con su ambiente, así como en el capitulo anterior en el que se puede controlar el encendido o apagado de luces; esto, para el caso de etapas de potencia, pero si lo que se necesita es el ámbito de la electrónica analógica, es más sencillo, por así decirlo, ya que tanto el control y el actuador están en el mismo dominio.

Lo que se plantea es una aplicación que permita el control por medio del puerto paralelo de un servo motor (dirección y velocidad). El procedimiento consiste en enviar señales activas para realizar el control del servo motor controlando así el movimiento con la precisión de las tareas a realizar.

Al programar la rutina de movimiento de los motores hay que tener en cuenta el tiempo de retardo entre dato y dato, si el primer dato no alcanza a ejecutar una acción sobre el motor se regresa y acumula la información en el puerto sobrecargándolo.

SERVOMOTOR

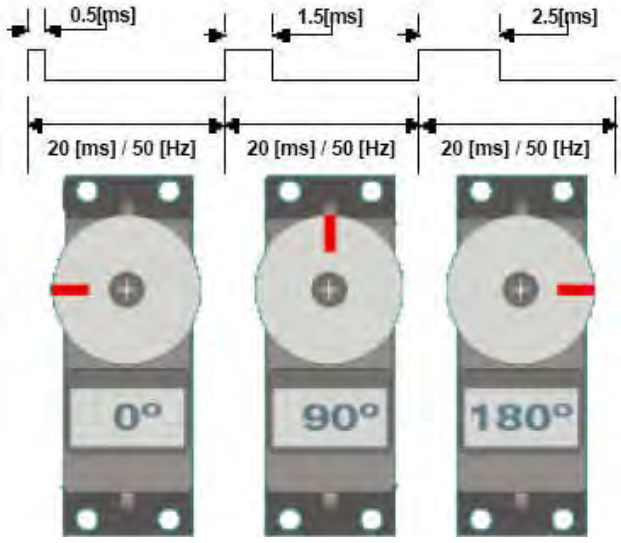
Es un tipo de motor de corriente continua que se caracteriza por su capacidad para posicionarse de forma inmediata en cualquier posición dentro de su intervalo de operación. Para esto, el servomotor espera un tren de pulsos que se corresponde con el movimiento a realizar.

Los servomotores están generalmente formados por un amplificador, un motor, un sistema reductor formado por ruedas dentadas y un circuito de realimentación, todo en una misma caja de pequeñas dimensiones. Dentro del servomotor, una tarjeta controladora le dice a un pequeño motor de corriente directa cuántas vueltas girar para acomodar la flecha (el eje de plástico que sale al exterior) en la posición que se le ha pedido. El resultado es un servo de posición con un margen de operación de 180° aproximadamente.

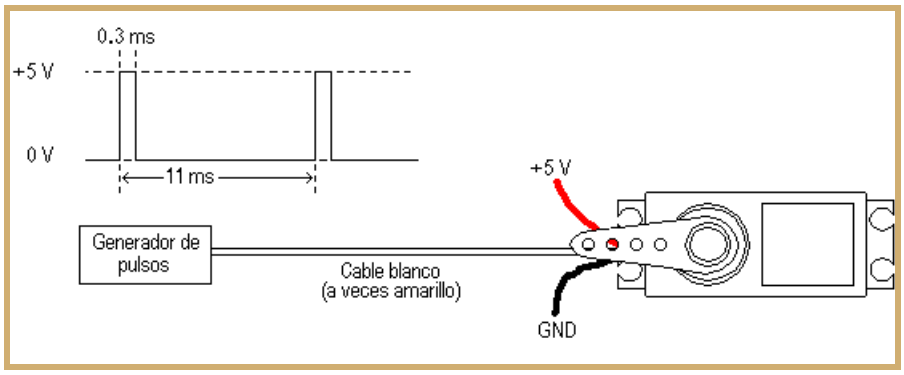
De los tres cables que salen de su cubierta. El rojo es de voltaje de alimentación (+5V), el negro es de tierra (0V ó GND). El cable blanco (a veces amarillo) es el cable por el cual se le instruye al servomotor en qué posición ubicarse (entre 0 grados y 180).

El sistema de control de un servo se limita a indicar la posición que se debe situar. Esto se lleva a cabo mediante una serie de pulsos, tal que la duración del pulso indica el ángulo de giro del motor. Cada servo tiene sus márgenes de operación, que se corresponden con el ancho del pulso máximo y mínimo que el servo entiende. Los valores más generales se corresponden con pulsos de entre 1 ms y 2 ms de anchura, que dejarían al motor en ambos extremos (0º y 180º).

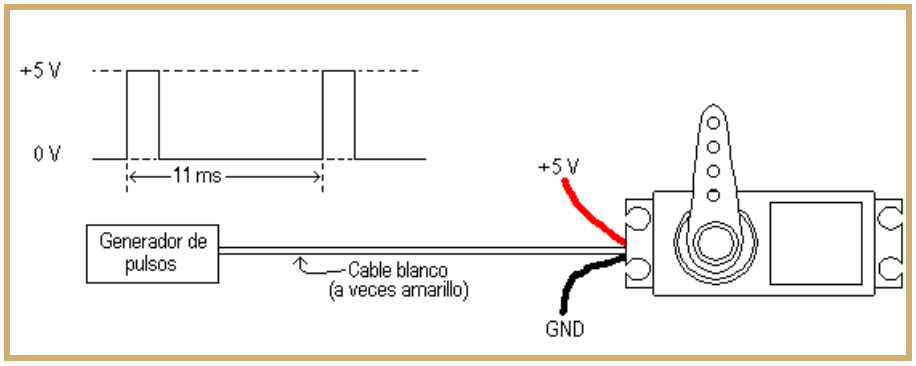
El valor 1.5 ms indicaría la posición central o neutra (90º), mientras que otros valores del pulso lo dejan en posiciones intermedias. Estos valores suelen ser los recomendados, sin embargo, es posible emplear pulsos menores de 1 ms o mayores de 2 ms, pudiéndose conseguir ángulos mayores de 180°. Si se sobrepasan los límites de movimiento del servo, éste comenzará a emitir un zumbido, indicando que se debe cambiar la longitud del pulso. El factor limitante es el tope del potenciómetro y los límites mecánicos constructivos.



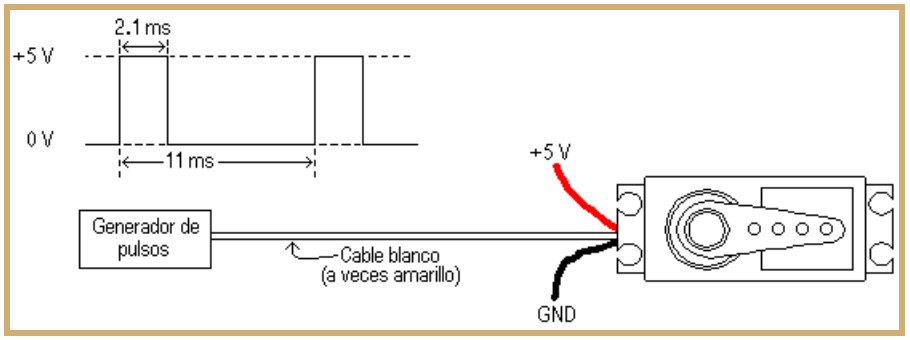
El período entre pulso y pulso (tiempo de OFF) no es crítico, e incluso puede ser distinto entre uno y otro pulso. Se suelen emplear valores ~ 20 ms (entre 10 ms y 30 ms). Si el intervalo entre pulso y pulso es inferior al mínimo, puede interferir con la temporización interna del servo, causando un zumbido, y la vibración del eje de salida. Si es mayor que el máximo, entonces el servo pasará a estado dormido entre pulsos. Esto provoca que se mueva con intervalos pequeños.



Señal de pulsos que controla al servo



Señal de pulsos que controla al servo

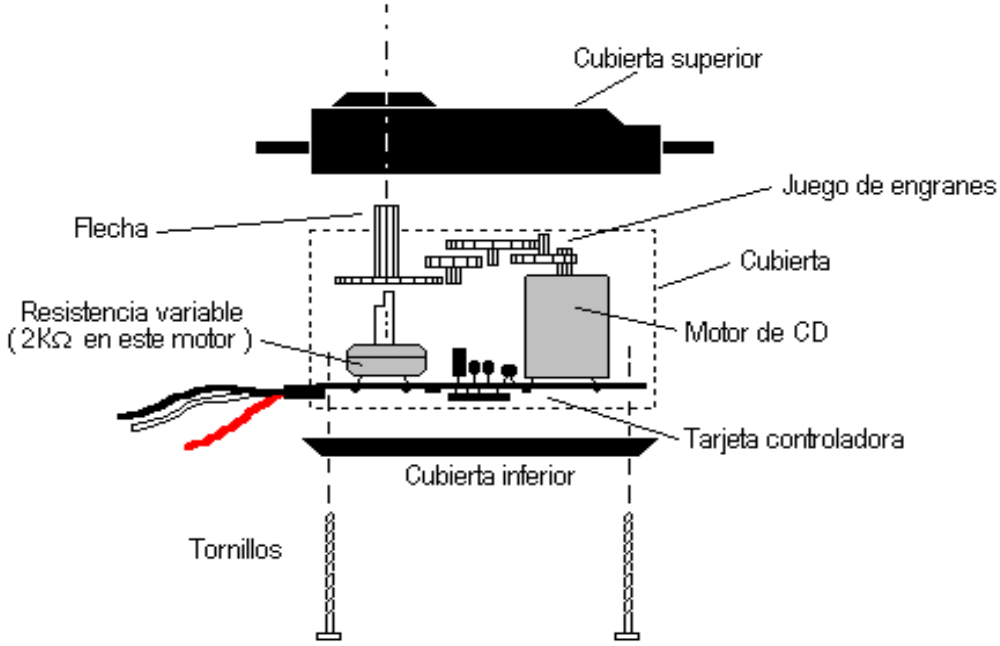


Señal de pulsos que controla al servo

MODIFICACIÓN DEL SERVOMOTOR

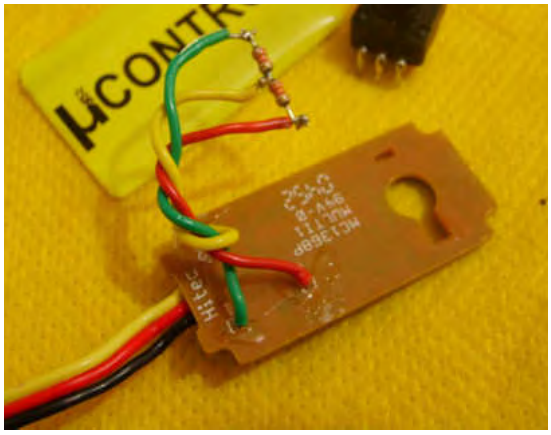
El potenciómetro del sistema de control del servo es un potenciómetro de menos de una vuelta, de modo que no puede dar giros completos en un mismo sentido. Para evitar que el motor pudiera dañar el potenciómetro, el fabricante del servo añade una pequeña pestaña en la caja reductora del motor, que impide que éste gire más de lo debido. Es por

ello que los servos tienen una cantidad limitada de giro, y no pueden girar continuamente en un mismo sentido. Es posible, sin embargo, realizar modificaciones al servo de modo que esta limitación se elimine, a costa de perder el control de posición



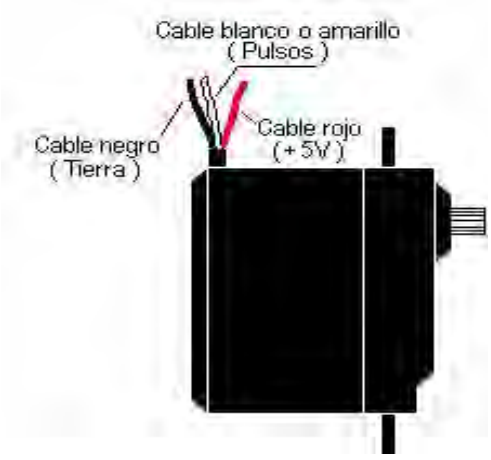
Para esto ese realiza un cambio en el sistema de control, de modo que se obtenga un sistema de control de velocidad. Para ello, se desacopla el potenciómetro de realimentación del eje del motor, y se hace que permanezca estático en una misma posición. Así, la señal de error del sistema de control dependerá directamente del valor deseado que se ajuste (que seguirá indicándose mediante pulsos de duración variable). Además se requiere la eliminación física de la pestaña limitadora de la caja reductora. El procedimiento será explicado de mejor forma, a continuación:

1. Retirar los tornillos de la parte posterior del servomotor y retirar la cubierta superior.
2. Remover con cuidado los engranajes.
3. Retirar la placa electrónica de la caja del servomotor.
4. Cortar la 'patilla' que limita el movimiento del engranaje principal.
5. Desoldar el potenciómetro o resistencia variable y separarlo de la placa electrónica.
6. Reemplazarlo por dos resistencias que sean la mitad del valor del potenciómetro. Por ejemplo, si es de 4.7k, se debe reemplazarlo por dos resistencias de 3.3k, Como se ilustra en la imagen siguiente.
7. Rearmar el motor.



Esta es una guía para el trucamiento de un servomotor *HITEC HS 311*, uno de los motores más comunes en el mercado. Sin embargo, puede servir de guía para cualquier tipo de motor.

Para la conexión del servomotor a la PC, es necesesario recordar que, el servomotor cuenta con tres cables de distintos colores para su conexión.

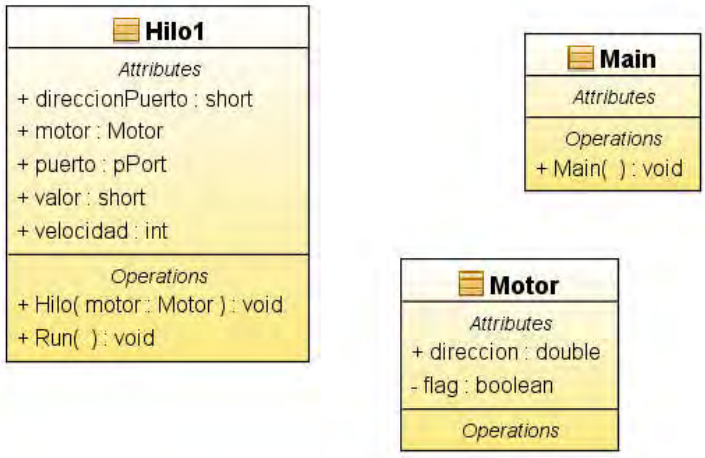


Cable amarillo: Conexión a uno de los puertos de salida del puerto paralelo.
Cable negro: Conexión al pin que corresponde a la tierra del puerto paralelo.
Cable rojo: Se recomienda conectar una fuente dependiente de VCC por los requerimientos de corriente.

El programa maneja los siguientes parámetros:

- Dirección a la que se desplaza el servomotor: horario y anti horario.
- Velocidad de desplazamiento.

A continuación se muestra el diagrama de UML de las clases de la aplicación:



Las principales variables son las que se detallan a continuación

ClassHilo1	• *direccionpuerto:short* Variable que almacena la dirección del puerto de salida • motor:Motor Sirve para hacer principio de encapsulamiento • puerto:pPort Envía los datos • valor:short Variable en la que se guarda , es el dato que se envía • velocidad:int Almacena el número del slider

CLASE HILO1

```
import acls.controlservomotor.vista.Motor;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import jnpout32.*;

public class Hilo1 extends Thread {

    pPort puerto = new pPort();
    Motor motor;
    int velocidad;

    short valor = 0;
    short direccionPuerto = 0x378;

    public Hilo1(Motor motor) {
        this.motor=motor;
        txParaleloDatos(0);
    }

    void txParaleloDatos(int dato) {
        valor = (short) dato;
        puerto.output(direccionPuerto, valor);
    }

    @SuppressWarnings("static-access")
    public void run() {
        //Control de Velocidad
        motor.getSliderVelocidad().addChangeListener(new ChangeListener(){
            public void stateChanged(ChangeEvent evt)
            {
                velocidad=motor.getSliderVelocidad().getValue();
            }
        });

        while (true) {
            try {
                txParaleloDatos(0);
                this.sleep(velocidad);
                txParaleloDatos(1);
                this.sleep((long) motor.getDireccion());
                System.out.println("Contador: " +velocidad);
            } catch (Exception e) {
            }
        }
    }
}
```

CLASE MOTOR

```
import acls.controlservomotor.controlador.Hilo1;
import javax.swing.JSlider;

public class Motor extends javax.swing.JFrame {

    Hilo1 hilo;
    double direccion;
    boolean flag;

    public Motor() {
        initComponents();
        hilo = new Hilo1(this);
        hilo.start();
    }

    public double getDireccion() {
        return direccion;
    }

    public JSlider getSliderVelocidad() {
        return sliderVelocidad;
    }
    private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
        System.exit(0);
    }

    private void btnAccionActionPerformed(java.awt.event.ActionEvent evt) {

        if (flag) {
            hilo.suspend();
            flag = false;
        } else {
            System.out.println("resume");
            flag = true;
            hilo.resume();
        }
    }

    private void rbIzquierdaActionPerformed(java.awt.event.ActionEvent evt) {
        direccion = 1;
    }

    private void rbDerechaActionPerformed(java.awt.event.ActionEvent evt) {
        direccion = 0.7;
    }
```

```
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {

            public void run() {
                new Motor().setVisible(true);
            }
        });
    }
    private javax.swing.ButtonGroup Direccion;
    private javax.swing.JButton btnAccion;
    private javax.swing.JButton btnSalir;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JRadioButton rbDerecha;
    private javax.swing.JRadioButton rbIzquierda;
    private javax.swing.JSlider sliderVelocidad;
}
```

CLASE MAIN

```
public class Main {

    public static void main(String[] args) {
        Motor principal=new Motor();
        principal.setLocation(400,300);
        principal.setVisible(true);
    }

}
```

CAPITULO 23

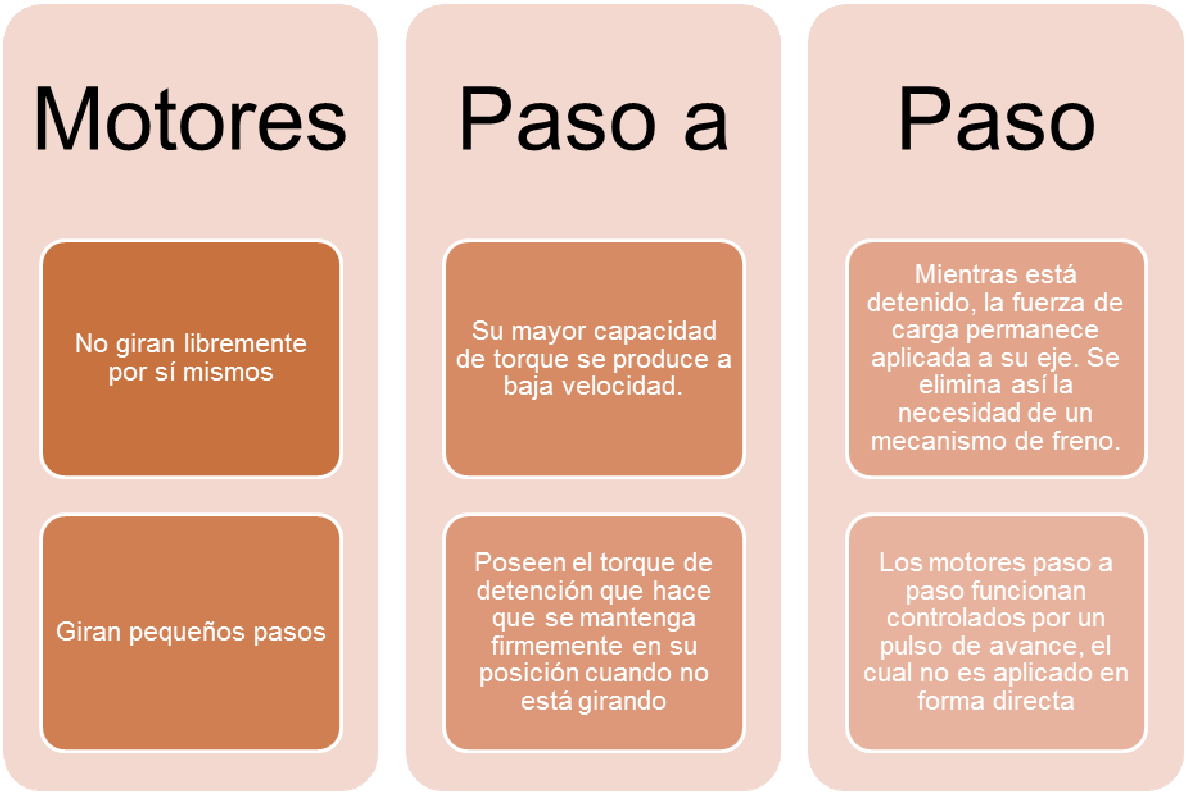
CONTROL DE UN MOTOR PASO A PASO CON EL PUERTO PARALELO

La necesidad de una aplicación para controlar exactamente donde se encuentrar su actuador, se la puede solventar mediante el empleo de motores paso a paso, los cuales, aunque no tien un torque tan grande como los servo motores, tiene la venjaja de ser mas precisos ya que se puede definir exactamente donde se encuentra el eje del motor, que para determinadas aplicaciones, que requieren acciones repetitivas constituye una exelente solución.

MOTORES PASO A PASO

Motores eléctricos sin escobillas, la mayoría de los bobinados del motor son parte del estator, y el rotor puede ser un imán permanente o, en el caso de los motores de reluctancia variable, un cilindro sólido construido con un material magnéticamente "blando" (como el hierro dulce). La conmutación se debe manejar de manera externa con un controlador electrónico.

A continuación se describe el comportamiento de los motores paso a paso.



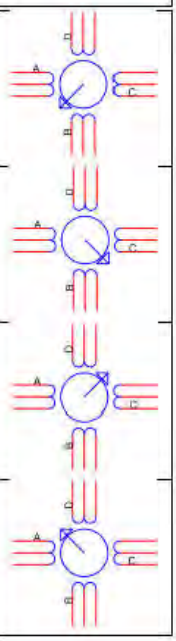
Estos motores tienen varios bobinados que, para producir el avance de ese paso, deben ser alimentados en una adecuada secuencia. Si se invierte el orden de esta secuencia, se logra que el motor gire en sentido opuesto. Si los pulsos de alimentación no se proveen en el orden correcto, el motor no se moverá apropiadamente. Puede ser que zumbe y no se mueva, o puede ser que gire, pero de una manera tosca e irregular.

Esto significa que hacer girar un motor paso a paso no es tan simple como hacerlo con un motor de corriente continua, al que se le entrega una corriente y listo. Se requiere un circuito de control, que será el responsable de convertir las señales de avance de un paso y sentido de giro en la necesaria secuencia de energización de los bobinados.

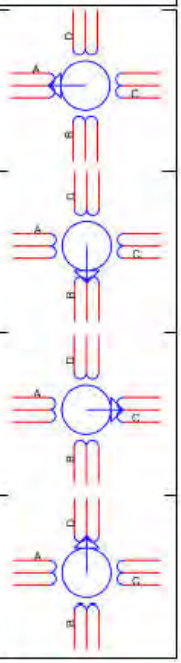
Para el control de este tipo de motores es necesario el envio de sevuencias de control, de tal manera que se consigue activar y desactivar las distintas bobinas internas.

Las Secuencia para manejar motores paso a paso unipolares, pueden ser las siguientes:

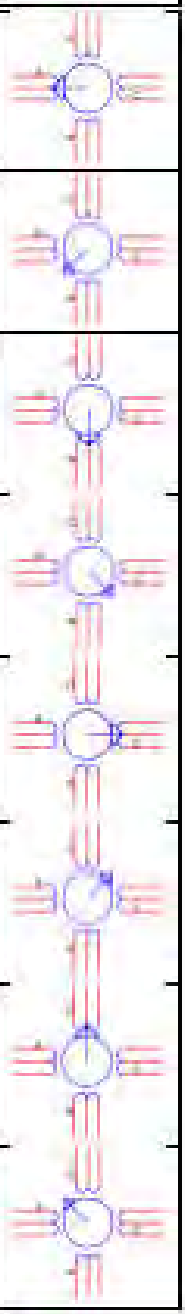
1. Secuencia Normal: el motor avanza un paso por vez debido a que hay 2 bobinas activas se obtiene un alto torque de paso y de retención

Paso	Bobina A	Bobina B	Bobina C	Bobina D	
1	On	On	Off	Off	
2	Off	On	On	Off	
3	Off	Off	On	On	
4	On	Off	Off	On	

2. Secuencia del tipo wave drive: Se activa una sola bobina a la vez el torque de paso y retención es menor

Paso	Bobina A	Bobina B	Bobina C	Bobina D	
1	On	Off	Off	Off	
2	Off	On	Off	Off	
3	Off	Off	On	Off	
4	Off	Off	Off	On	

3. Secuencia del tipo de medio paso: brindan un movimiento igual a la mitad del paso real, primero se activan 2 bobinas y luego 1 y asi sucesivamente

Paso	Bobina A	Bobina B	Bobina C	Bobina D	
1	On	Off	Off	Off	
2	On	On	Off	Off	
3	Off	On	Off	Off	
4	Off	On	On	Off	
5	Off	Off	On	Off	
6	Off	Off	On	On	
7	Off	Off	Off	On	
8	On	Off	Off	On	

Identificación de tipos de motores paso a paso y sus cables

IMAN PERMANENTE

son utilizados en el avance de papel y del cabezal de impresión de las impresoras, disketteras, etc.

Poseen un imán que aporta el campo magnético para la operación.

RELUCTANCIA VARABLE

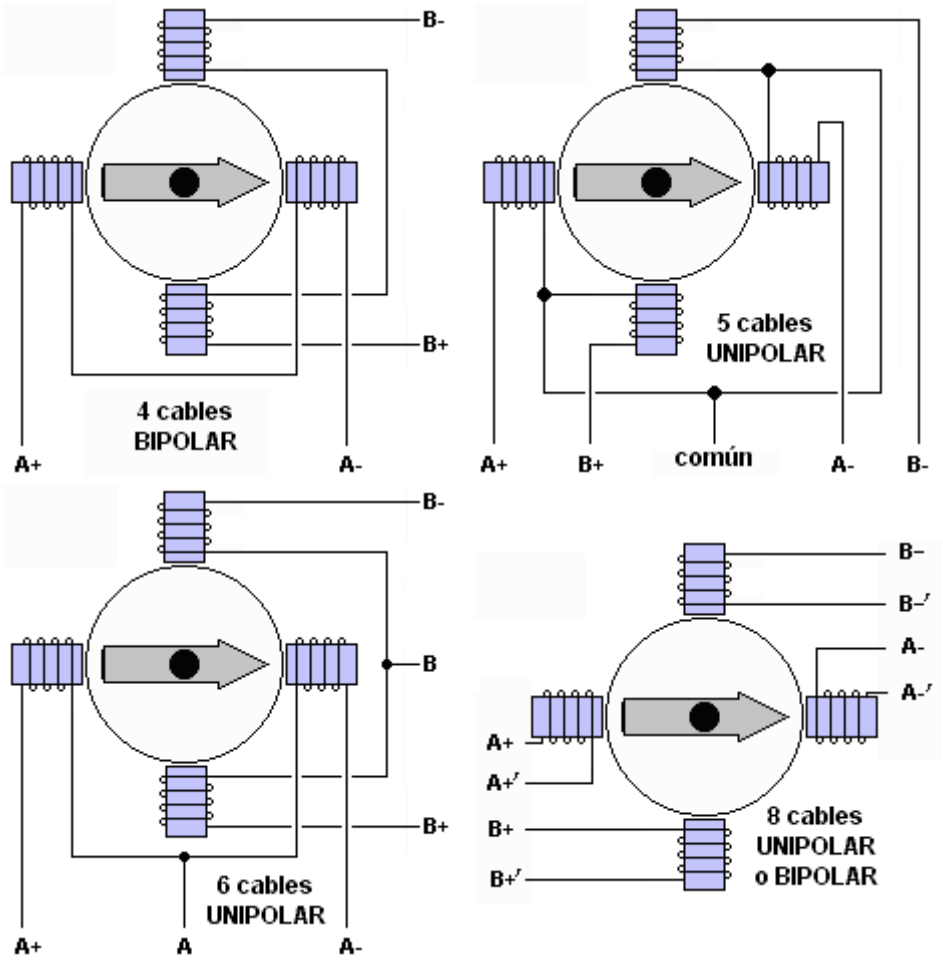
Poseen un rotor de hierro dulce que en condiciones de excitación del estator, y bajo la acción de su campo magnético, ofrece menor resistencia a ser atravesado por su flujo en la posición de equilibrio

HÍBRIDOS

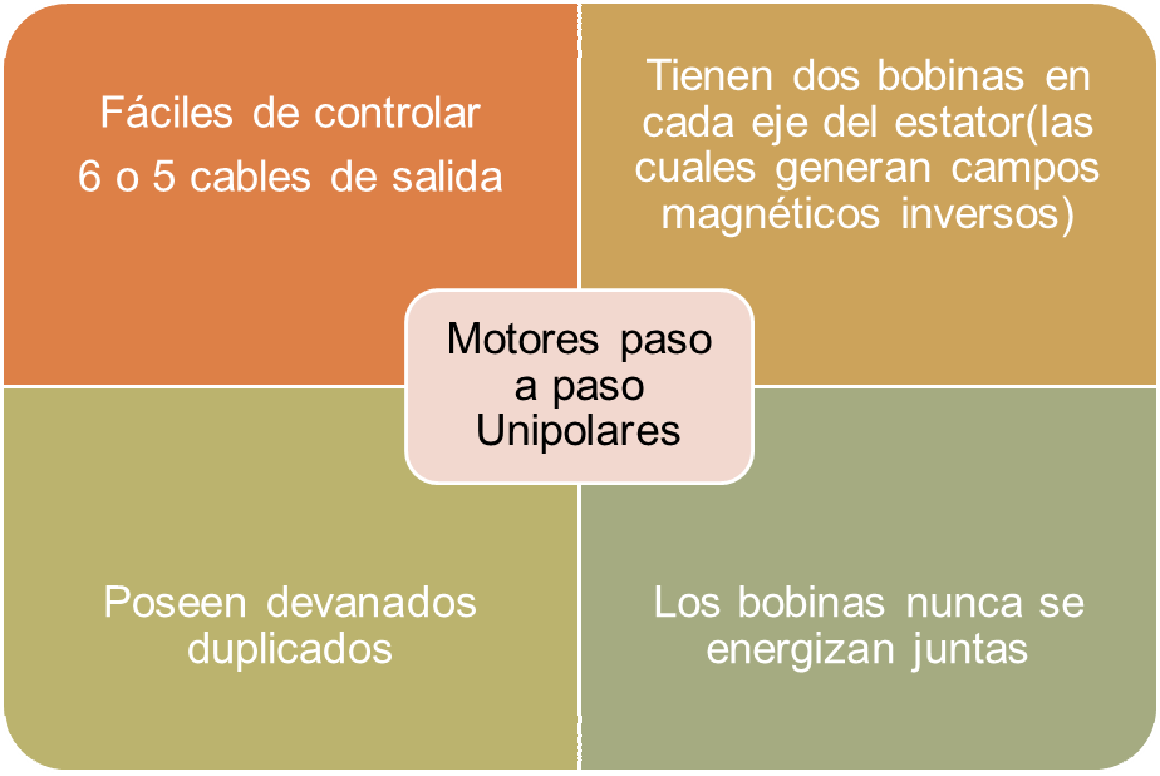
Combinación de los dos motores anteriores.

Se construyen con estatores multidentados y un rotor de imán permanente. Los motores híbridos estándar tienen 200 dientes en el rotor y giran en pasos de 1,8 grados.

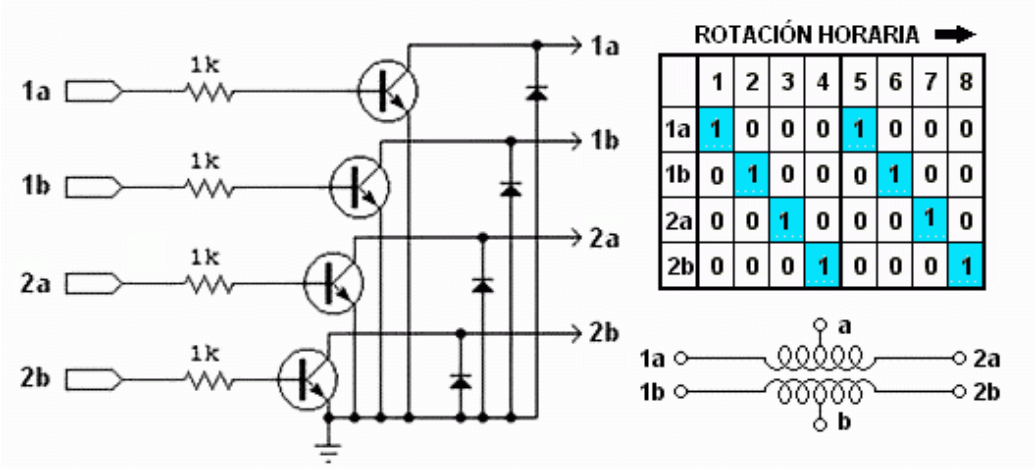
Diferentes tipos de cableado de las bobinas.



Cada uno de estos tipos requerirá un diferente circuito de control.



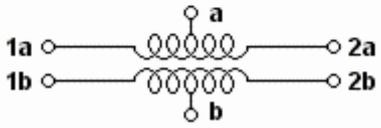
En el esquema más común de conexión se unen los "puntos medios" de ambos ejes (**a** y **b** en el dibujo) y se les conecta al positivo de la alimentación del motor. El circuito de control de potencia, entonces, se limita a poner a masa los bobinados de manera secuencial.



Circuito y secuencia para controlar un motor unipolar.

ROTACIÓN HORARIA ➡

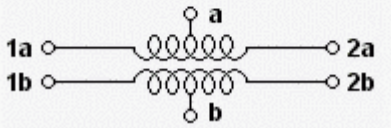
	1	2	3	4	5	6	7	8
1a	1	1	0	0	1	1	0	0
1b	0	1	1	0	0	1	1	0
2a	0	0	1	1	0	0	1	1
2b	1	0	0	1	1	0	0	1



Secuencia para lograr más fuerza

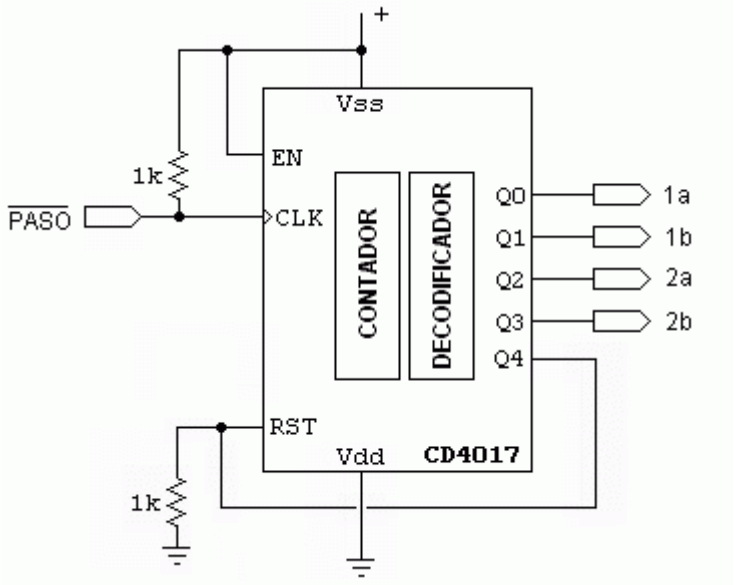
ROTACIÓN HORARIA ➡

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1a	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1
1b	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0
2a	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0
2b	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1



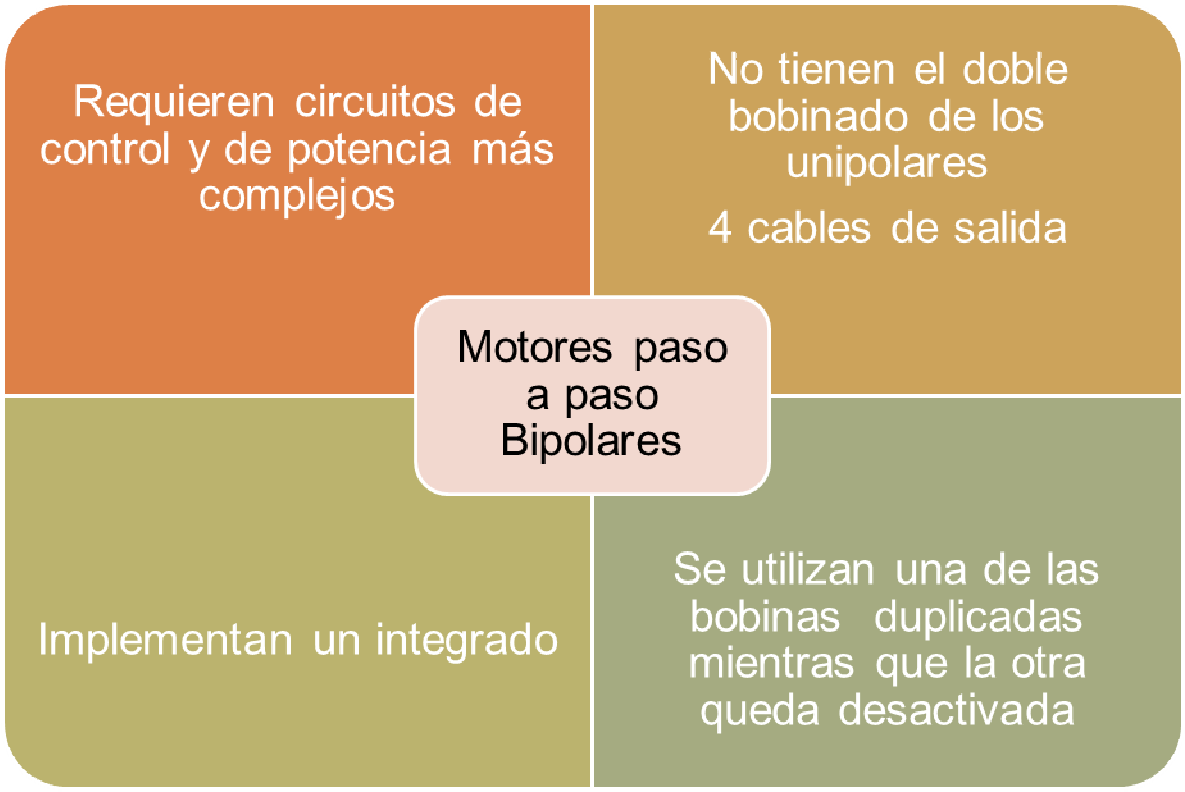
Secuencia para lograr medio-paso

La secuencia de pulsos de un motor unipolar se puede controlar con un contador binario de dos bits con un decodificador, como por ejemplo el integrado CD 4017. La parte de potencia puede ser implementada con un único transistor en cada bobinado.



Control de avance con un único integrado CD 4017

Motores paso a paso bipolares:



Ventajas.

- Son convenientes para el control exacto de su rotación, puesto que la operación del motor paso a paso se sincroniza con las señales de pulso del comando generadas de los pulsos en enviados por los puertos.
- Se puede controlar fácilmente su velocidad desde el computador.
- Es fácil de invertir el sentido de rotación del motor paso a paso.
- Bajo costo.

Desventajas.

- El flujo actual de un driver a la bobina del motor no se puede aumentar o disminuir durante la operación. Por lo tanto, si el motor se carga con una carga más pesada que la característica diseñada del esfuerzo de torsión del motor, saldrá de paso con los pulsos.
- El motor paso a paso produce más ruido y vibración que los servos.
- El motor paso a paso no se puede utilizar para tareas de rotación de alta velocidad.

A continuación se presenta el circuito para conectar el motor paso a paso al puerto paralelo.

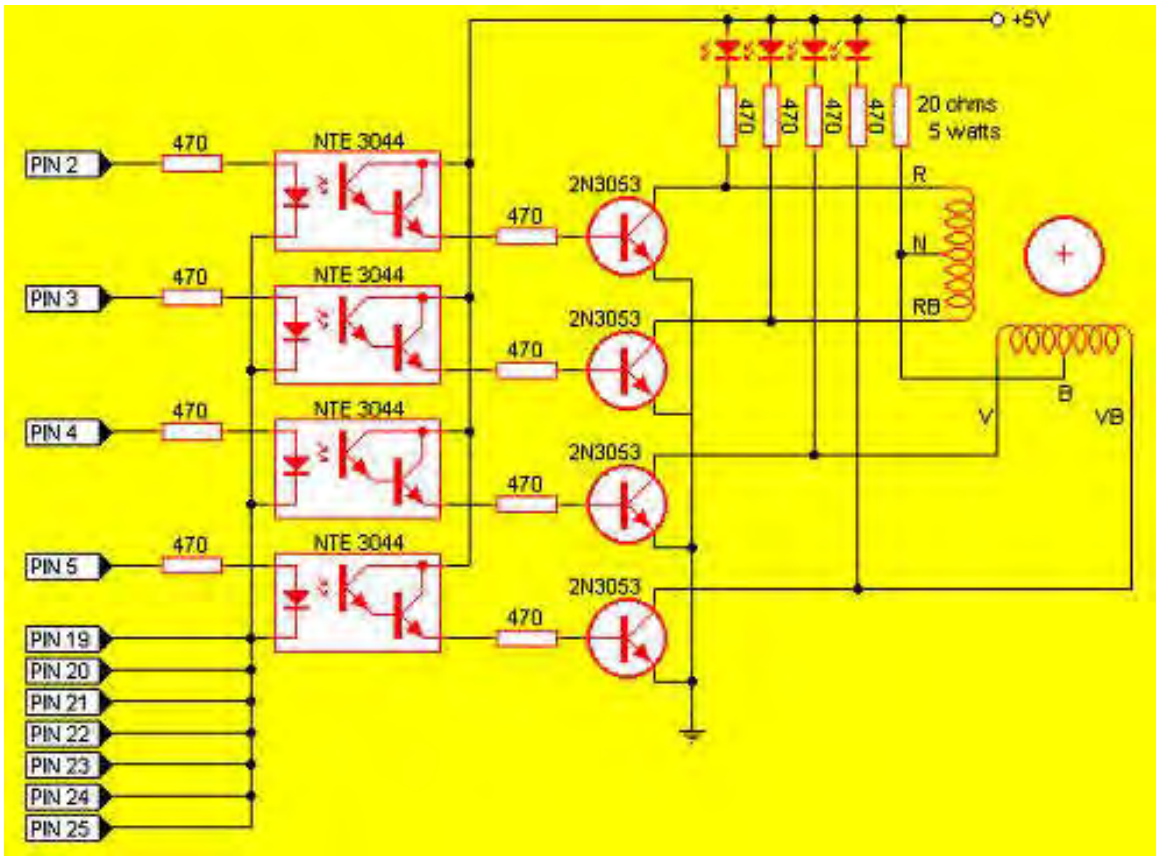


DIAGRAMA UML.

Clase Monitoreo.

Class Monitoreo.
-serial commRxTX -horario: Timer -antihorario: Timer -leer: Timer int xxx int tiempo
-MONITOREO() - jButton2ActionPerformed(java.awt.event.ActionEvent evt): void -jButton1ActionPerformed(java.awt.event.ActionEvent evt):void -formWindowActivated(java.awt.event.WindowEvent evt):void -INCREMENTOActionPerformed(java.awt.event.ActionEvent evt):void -LEE_PPActionPerformed(java.awt.event.ActionEvent evt):void +static main(String args[]):void

Clase serial.

Class Serial
-static CommPortIdentifier portId -static CommPortIdentifier saveportId -static Enumeration portList -InputStream inputStream -int Dato_Enviar=0 -static int Dato_Recibido=0 -static int Dato=99 -static OutputStream outputStream -static boolean outputBufferEmptyFlag -static boolean portFound
-initwritetoport() : void -writetoport() : void +StatePort():int

Clase Monitoreo

Esta clase es un JFrame llamado Monitoreo, que es hija de javax.swing.JFrame por lo que junto a la declaración de la clase tenemos la palabra extends. En esta clase se tiene 4 botones: Giro Horario, Giro Antihorario, Salir, Leer Puerto; 1 JSlider que permite controlar la velocidad de giro del motor paso a paso y 2 Labels: JLabel8 y JLabel9. También se emplea 2 timers, los mismos que permiten controlar la ejecución de cada pulso de nuestro motor paso a paso.

A continuación describiremos el propósito y funcionamiento de cada uno.

Este método es un constructor sobrecargado sin argumentos, que permite inicializar todo los componentes de la clase Monitoreo además de realizar varias tareas más. También este método selecciona el tiempo, para que cada 10 [ms] el timer varíe y por lo tanto varíe los pulsos enviados al motor.

Se declara la dirección 379 que es la salida del puerto paralelo y se crea un pobjeto de tipo puerto paralelo. La variable xxx inicializada en 0, entra en un switch que está compuesto por 8 casos, el 1º caso envía el numero 16 al motor, el mismo que en código binario representa el numero 10000, cabe explicar que para la rotación de l motor paso a paso se utliliza los bits más significativos del puerto paralelo es decir D7, D6, D5, D4 y el D3, D2, D1 y D0 permanecerán siempre en 0 y el numero enviado recorrerá siempre una posición hacia la izquierda , el caso 2 se da cuando xxx se incrementa en 1 y el valor que se envia al puerto es el 32, en el 3º caso el 64, el 4º 128, el 5º 16, el 6º 32, el 7º 64 y el 8º 128, de esta manera se forma:

00010000
00100000
01000000
10000000
00010000

00100000
01000000
10000000

Los números 1 forman el pulso enviado. Si xxx es mayor que 7, es decir esta en 8, la variable xxx vuelve a ponerse en 0 y por lo tanto vuelve a entrar en el switch.

Para el giro en sentido antihorario, se realiza el mismo proceso pero en ves de que xxx se incrementa, se decrementa en 1.

JBoton2, es el botón que permite salir del programa, y además muestra una ventana con el mensaje “Gracias por utilizar el sistema”.

El botón JButton1, representa el Botón de sentido Antihorario, este detiene el sentido horario con horario.stop() y activa el antihorario con antihorario.start().

Se realiza la lectura del puerto y se envía a la dirección 379 los datos, aparece en el JLabel8 las direcciones ocupadas por cada pin del puerto paralelo.

Clase serial

Esta clase es una interfaz por lo que a lado de la declaración de la clase lleva implements SerialPortEventListener, los métodos que tiene son:

private void initwritetoport()

Lleva el try y el catch, que permiten realizar lectura lectura por teclado.

private void writetoport()

Se encarga de enviar los datos y escribirlos en el puerto.

CODIGO FUENTE:

Clase Serie

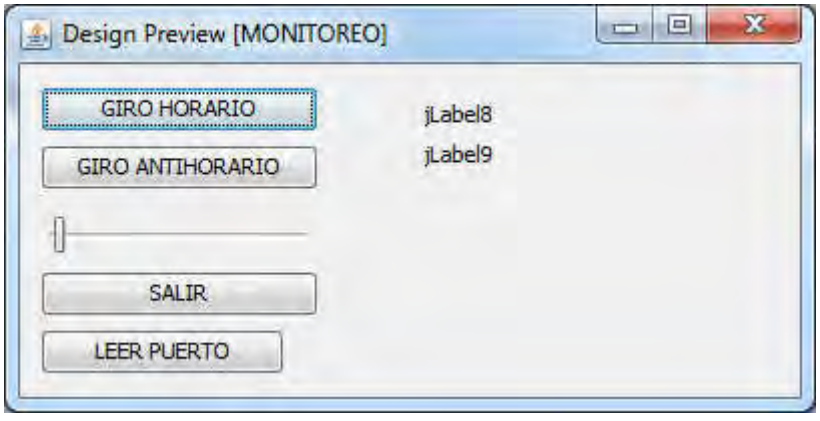
```
import java.io.*;
import java.util.*;
import javax.comm.*;
import java.awt.event.*;

public class serial implements SerialPortEventListener  {
   private static CommPortIdentifier portId;
   private static CommPortIdentifier saveportId;
   private static Enumeration        portList;
   private InputStream           inputStream=null;
   private SerialPort            serialPort=null;
   private int Dato_Enviar=0;//        messageString = "Hello, world!";
```

```
private static int Dato_Recibido=0;
private static int Dato=99;
private static OutputStream      outputStream=null;
private static boolean        outputBufferEmptyFlag = false;
static private boolean           portFound=false;

private void initwritetoport() {

   try {
      outputStream = serialPort.getOutputStream();
   } catch (IOException e) {}

   try {
      serialPort.notifyOnOutputEmpty(true);
   } catch (Exception e) {
      System.out.println("Error setting event notification");
      System.out.println(e.toString());
      System.exit(-1);
   }
}

private void writetoport() {
   try {
       this.Dato=this.Dato_Enviar;
      outputStream.write(Dato_Enviar);//messageString.getBytes());
   } catch (IOException e) {}
}
public serial(){
}
 public serial(int Aux){
   String          defaultPort;
   String osname = System.getProperty("os.name","").toLowerCase();
   if ( osname.startsWith("windows") ) {
      defaultPort = "COM8";
   }
    else {
      System.out.println("Sorry, your operating system is not supported");
      return;
   }
   System.out.println("INICIANDO BUSQUEDA DE PUERTOS DISPONIBLES");

                  // parse ports and if the default port is found, initialized the reader
   portList = CommPortIdentifier.getPortIdentifiers();
   while (portList.hasMoreElements())
   {
      portId = (CommPortIdentifier) portList.nextElement();
      if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
      {
         //if (portId.getName().equals(defaultPort))
```

```
        if (portId.getName().equals(portId.getName()))
        {
          System.out.println("Coneccion satisfactoria con: "+portId.getName());
          portFound = true;
          // init reader thread
           serial reader = new serial(true);
         //Conectar();
        }
      }
    }
    if (!portFound) {
       System.out.println(" "+portId);
      System.out.println("port " + defaultPort + " not found.");
    }
  }

  public serial(boolean G) {
    // initalize serial port
    try {
      serialPort = (SerialPort) portId.open("SimpleReadApp", 2000);
    } catch (PortInUseException e) {}
    try {
      inputStream = serialPort.getInputStream();
    } catch (IOException e) {}
    try {
      serialPort.addEventListener(this);
    } catch (TooManyListenersException e) {}
    // activate the DATA_AVAILABLE notifier
    serialPort.notifyOnDataAvailable(true);
    try {
      serialPort.setSerialPortParams(9600, SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1,
                SerialPort.PARITY_NONE);
    } catch (UnsupportedCommOperationException e) {}
     initwritetoport();
  }

  public void serialEvent(SerialPortEvent event) {
    switch (event.getEventType()) {
    case SerialPortEvent.BI:
    case SerialPortEvent.OE:
    case SerialPortEvent.FE:
    case SerialPortEvent.PE:
    case SerialPortEvent.CD:
    case SerialPortEvent.CTS:
    case SerialPortEvent.DSR:
    case SerialPortEvent.RI:
    case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
      break;
```

```
      case SerialPortEvent.DATA_AVAILABLE:
        try {
          while (inputStream.available() > 0) {
                this.Dato_Recibido = inputStream.read();
          }
        } catch (IOException e) {}

        break;
      }
    }

    public void SetDatoSerial (int dato){
        this.Dato_Enviar=dato;
         writetoport();
    }

    public int GetDatoSerial (){
        System.out.println("Read: "+this.Dato_Recibido);
        return this.Dato_Recibido;
    }
    public int StatePort(){
        if(this.portFound)
        {
            return (1);
        }
            return (0);
    }
}
```

Clase MONITOREO




```
import java.awt.*;
import java.io.*;
import java.awt.event.*;
import javax.swing.*;
import javax.comm.*;
```

```
import jnpout32.*;


public class MONITOREO extends javax.swing.JFrame {

    private serial commRxTX=new serial();
    private Timer horario;
    private Timer antihorario;
    private Timer leer;
    int xxx = 0 ;
    int tiempo=70;

    public MONITOREO() {
        //tiempo=jSlider1.getValue();
        initComponents();


            leer = new Timer(10,new ActionListener(){
            public void actionPerformed(ActionEvent a){
            short direccion1 = 0x379;
            pPort puertoParalelo = new pPort();
            jLabel9.setText(Short.toString(puertoParalelo.input(direccion1)));
            }
            });

            horario= new Timer(tiempo,new ActionListener(){

            public void actionPerformed(ActionEvent a){

            xxx = xxx + 1;

            switch(xxx)

       {
           case 1:
           {
             //String letra = "14";
             String letra = "16";
             jLabel8.setText(letra);
             short direccion1 = 0x378;
             pPort puertoParalelo = new pPort();
             short valor=(short)Integer.parseInt(letra);
             puertoParalelo.output(direccion1, valor);
             break;
           }
           case 2:
           {
             //String letra = "13";
             //String letra = "2";
```

```
   String letra = "32";
   jLabel8.setText(letra);
   short direccion1 = 0x378;
   pPort puertoParalelo = new pPort();
   short valor=(short)Integer.parseInt(letra);
   puertoParalelo.output(direccion1, valor);
    break;
}
case 3:
{
   //String letra = "11";
    //String letra = "4";
   String letra = "64";
   jLabel8.setText(letra);
   short direccion1 = 0x378;
   pPort puertoParalelo = new pPort();
   short valor=(short)Integer.parseInt(letra);
   puertoParalelo.output(direccion1, valor);
    break;
}
case 4:
{
   //String letra = "7";
   //String letra = "8";
   String letra = "128";
   jLabel8.setText(letra);
   short direccion1 = 0x378;
   pPort puertoParalelo = new pPort();
   short valor=(short)Integer.parseInt(letra);
   puertoParalelo.output(direccion1, valor);

   break;
}
case 5:
{
   //String letra = "14";
    String letra = "16";
   jLabel8.setText(letra);
   short direccion1 = 0x378;
   pPort puertoParalelo = new pPort();
   short valor=(short)Integer.parseInt(letra);
   puertoParalelo.output(direccion1, valor);
   break;
}
case 6:
{
   //String letra = "13";
   //String letra = "2";
   String letra = "32";
```

```
        jLabel8.setText(letra);
        short direccion1 = 0x378;
        pPort puertoParalelo = new pPort();
        short valor=(short)Integer.parseInt(letra);
        puertoParalelo.output(direccion1, valor);
         break;
      }
      case 7:
      {
        //String letra = "11";
         //String letra = "4";
        String letra = "64";
        jLabel8.setText(letra);
        short direccion1 = 0x378;
        pPort puertoParalelo = new pPort();
        short valor=(short)Integer.parseInt(letra);
        puertoParalelo.output(direccion1, valor);
         break;
      }
      case 8:
      {
        //String letra = "7";
        //String letra = "8";
        String letra = "128";
        jLabel8.setText(letra);
        short direccion1 = 0x378;
        pPort puertoParalelo = new pPort();
        short valor=(short)Integer.parseInt(letra);
        puertoParalelo.output(direccion1, valor);

        break;
      }
  }
       if (xxx > 7)
       {
           xxx = 0;
       }


       }
       });

       antihorario = new Timer(tiempo,new ActionListener(){
       public void actionPerformed(ActionEvent a){

       xxx = xxx - 1;

       switch(xxx)
```

```
{
    case 1:
    {
      //String letra = "14";
       String letra = "16";
      jLabel8.setText(letra);
      short direccion1 = 0x378;
      pPort puertoParalelo = new pPort();
      short valor=(short)Integer.parseInt(letra);
      puertoParalelo.output(direccion1, valor);
      break;
    }
    case 2:
    {
      //String letra = "13";
      //String letra = "2";
      String letra = "32";
      jLabel8.setText(letra);
      short direccion1 = 0x378;
      pPort puertoParalelo = new pPort();
      short valor=(short)Integer.parseInt(letra);
      puertoParalelo.output(direccion1, valor);
       break;
    }
    case 3:
    {
      //String letra = "11";
       //String letra = "4";
      String letra = "64";
      jLabel8.setText(letra);
      short direccion1 = 0x378;
      pPort puertoParalelo = new pPort();
      short valor=(short)Integer.parseInt(letra);
      puertoParalelo.output(direccion1, valor);
       break;
    }
    case 4:
    {
      //String letra = "7";
      //String letra = "8";
      String letra = "128";
      jLabel8.setText(letra);
      short direccion1 = 0x378;
      pPort puertoParalelo = new pPort();
      short valor=(short)Integer.parseInt(letra);
      puertoParalelo.output(direccion1, valor);

      break;
    }
```

```
    }
        if (xxx < 0)
        {
            xxx = 5;
        }


        }
        });
 }

  private void semaforo(int num)
{


}
  private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
      jOptionPane1.showMessageDialog(null, "GRACIAS POR UTILIZAR EL SISTEMA",
"salida", 1, null);
      System.exit(0);
  }

  private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
      tiempo=jSlider1.getValue();
      horario.stop();
      antihorario.start();
  }

  private void formWindowActivated(java.awt.event.WindowEvent evt) {
      if(commRxTX.StatePort()==0)
      {
        commRxTX= new serial(1);
      }
      leer.start();
  }

  private void INCREMENTOActionPerformed(java.awt.event.ActionEvent evt) {
      tiempo=jSlider1.getValue();
      horario.start();
      antihorario.stop();
  }

  private void LEE_PPActionPerformed(java.awt.event.ActionEvent evt) {

      short direccion1 = 0x379;
      pPort puertoParalelo = new pPort();
```

```
        jLabel8.setText(Short.toString(puertoParalelo.input(direccion1)));

    }

    private void jSlider1StateChanged(javax.swing.event.ChangeEvent evt) {
        tiempo = jSlider1.getValue();

    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new MONITOREO().setVisible(true);
            }
        });
    }

    private javax.swing.JButton INCREMENTO;
    private javax.swing.JButton LEE_PP;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JOptionPane jOptionPane1;
    private javax.swing.JSlider jSlider1;
}
```

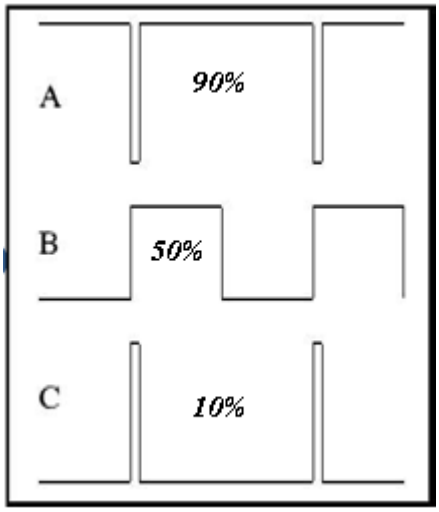
CAPITULO 24

CPNTROL DE UN MOTOR DC CON EL PUERTO PARALELO

Ya anteriormente se comentó acerca del control de un servomotor, ahora lo que se plantea es el control de un motor DC. A diferencia de los servomotor, los motores DC, son de los más comunes y económicos, y se los puede encontrar en la mayoría de los juguetes a pilas por ejemplo; además de sus ventajas en cuanto a obtención y precio, existe una gran ventaja en lo que a manejo del mismo se refiere, ya que estos motores permiten controlar su velocidad y giro con relativa facilidad.

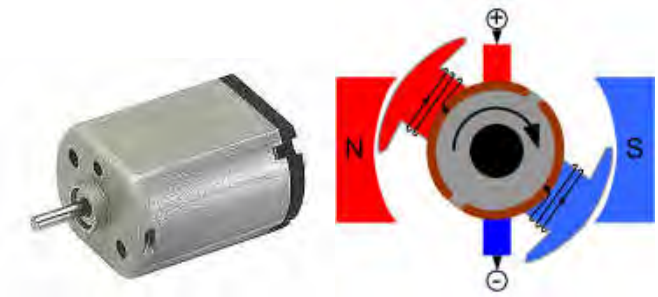
PWM

PWM, técnica de modulación de ancho de pulso utilizado comúnmente para el control. Si el concepto le extiendo a la alimentación de un motor se vería que si se enciende la alimentación y se apaga rápidamente, el motor tomará una velocidad comprendida entre velocidad cero y velocidad máxima; es decir que PWM alimenta el motor suministrándole una serie de pulsos, para controlar la velocidad del motor.

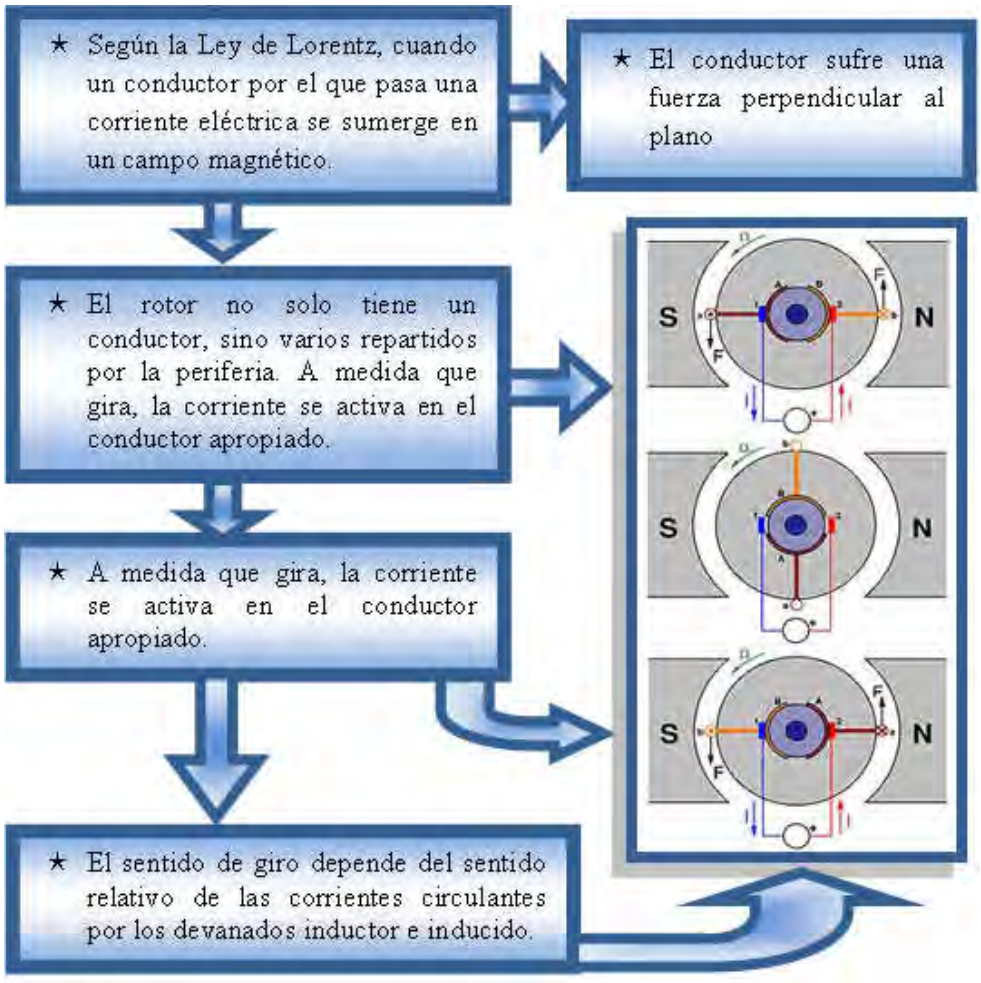


Motor DC

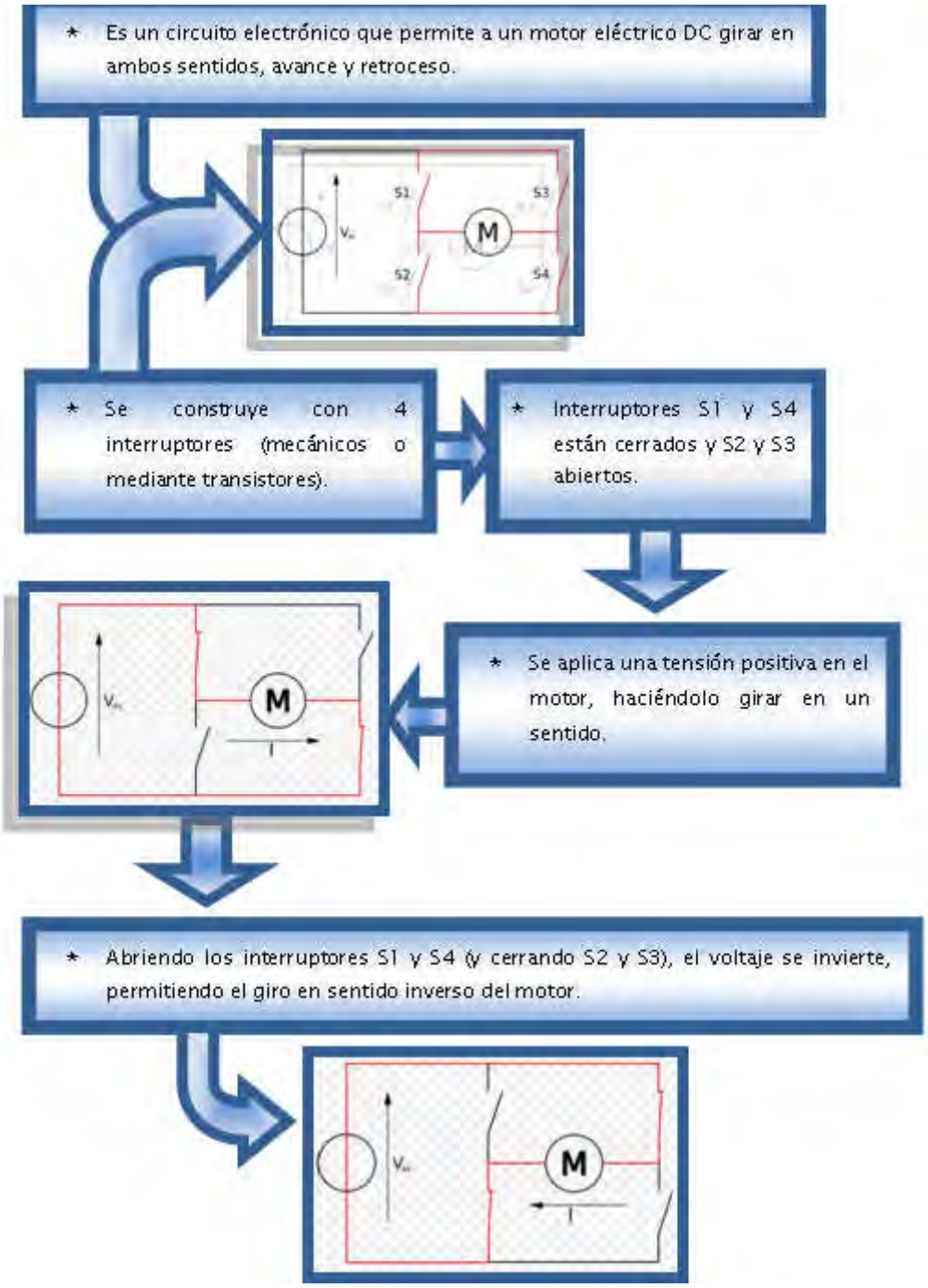
Es una máquina que convierte la energía eléctrica en mecánica; funciona con corriente continua y es una de las más versátiles en la industria, ya que permite el fácil control de posición, paro y velocidad. La principal característica del motor de corriente continua es la posibilidad de regular la velocidad desde vacío a plena carga.



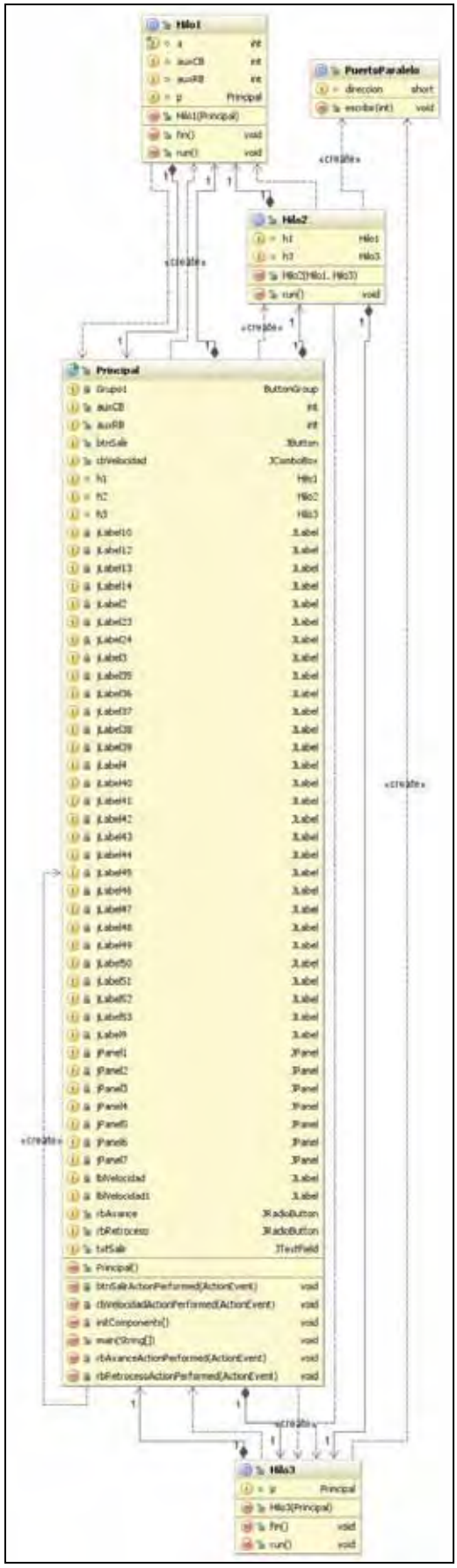
Su funcionamiento se ilustra en el siguiente esquema.



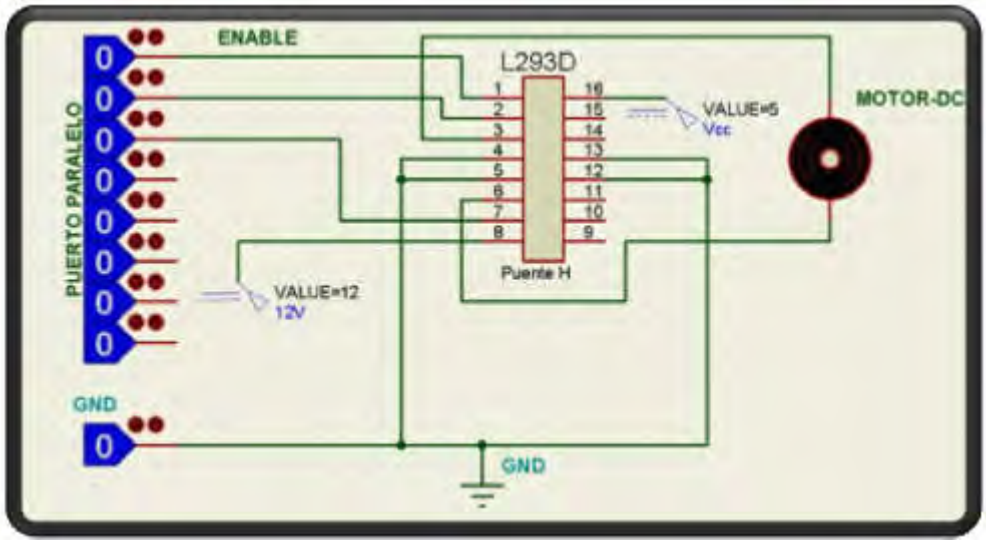
Puente H



A continuación se presenta el diagrama de UML de la aplicación:


Hilo1
PuertoParalelo
Hilo2
Principal
Hilo3
«create»

El esquema para la conexión es el siguiente:



Mapa de Variables

Control de Motor DC (Velocidad y Sentido de Giro)	
Hilo1	La clase Hilo1 es el proceso encargado de actualizar el campo de texto txtSalir de acuerdo a las variaciones en velocidad y giro resultado de la interacción del usuario con la interfaz. ***Atributos*** • Las *variables auxcB, auxrB* y a son constantes que permiten controlar la escritura de un nuevo texto únicamente cuando se cambie las opciones contenidas en la interfaz, ya que si esta escritura se ejecuta durante un proceso continuo descrito por hilos sin restricciones, los datos son casi imperceptibles y titilantes para su visualización.
Hilo2	La clase Hilo2 es el proceso encargado finalizar los Hilos 1 y 2 cuando el botón salir de la ventana principal es seleccionado. ***Atributos*** • Los *objetos de las clases Hilo 1 y 3* son necesarios como variables de clase ya que representan los hilos ejecutados desde la frame Principal que deben culminar su proceso cuando se presione el botón salir.
Hilo3	La clase Hilo3 es el proceso encargado de la interacción con el puerto paralelo, que para esta aplicación se utiliza de forma unidireccional - escritura en el puerto a través del bus de datos. ***Atributos*** • *Principal p* es un objeto tipo frame requerido como argumento de clase para captar los estados y cambios dados en el frame Principal.

PuertoParalelo	Clase que permite la escritura en el puerto paralelo de un dato entero a través de las funciones nativas de interacción con el puerto paralelo a las que podemos acceder a través de la librería jnpout32 y su respectivo dll. ***Argumentos*** • *short dirección:* establece el dato respectivo a la dirección del bus de datos del puerto paralelo (0x378) para habilitar la comunicación.
Clase Principal	Frame que permite la iteración del usuario y el control de velocidad - giro del motor. ***Argumentos*** • *auxCB, auxRB:* atributos de clase auxiliares que cambian su valor únicamente cuando se registran variaciones en los botones de selección del frame. • *h1,h2,h3:* Hilos de la clase Hilo 1,2,3 que permiten el control de los procesos necesarios para ejecutar los cambios solicitados desde el frame en el dispositivo exterior.

Explicación de los métodos

A continuación se describe detalladamente el tipo, argumentos y descripción de los métodos correspondientes a las tres aplicaciones desarrolladas para implementar componentes de interfaz gráfica humano – máquina.

Control de Motor DC (Velocidad y Sentido de Giro)		
Clase Hilo1	**public Hilo1 (Principal p)**	* *Tipo:* Void (Constructor).
		* *Argumentos:* Principal p >> de la clase Hilo1 arranca el hilo y asigna al atributo de clase el frame respectivo a la ventana principal.
		* *Descripción del método:* Constructor de la clase Hilo1 arranca el hilo y asigna al atributo de clase el frame respectivo a la ventana principal.
	Public void fin( )	* *Tipo:* Void
		* *Argumentos:* No tiene argumentos.
		* *Descripción del método:* Detiene la ejecución del proceso controlado por el Hilo1.
	private void run ( )	* *Tipo:* Void.
		* *Argumentos:* No tiene argumentos.
		* *Descripción del método*: Método que engloba el proceso de ejecución del Hilo1, actualiza el campo de texto txtSalir cuando varía la selección de

<table>
<tr><td></td><td></td><td>parámetros en el Frame Principal.

No se requiere de la ejecución de un proceso continuo sin restricciones ya que provoca una visualización intermitente del texto debido a la rapidez de ejecución del proceso respecto al tiempo del ciclo de trabajo del procesador.

El número de revoluciones por minuto (RPM) fueron extraídos de tablas sobre control de motores dc:

• Velocidad lenta >> 149 RPM.
• Velocidad media >> 746 RPM.
• Velocidad rápida >> 1493 RPM.</td></tr>
<tr><td rowspan="5">Clase Hilo2</td><td rowspan="2">public Hilo2(Hilo1 h1, Hilo3 h3)</td><td>* Tipo: Void (Constructor).</td></tr>
<tr><td>* Argumentos: Hilo h1, Hilo h2, Hilo h3 >> objetos de las clases Hilo1, Hilo2 e Hilo3.</td></tr>
<tr><td>* Descripción del Método: Inicia el proceso del hilo, encargado de inicializar los atributos de clase igualándolos a los argumentos del constructor que representan los hilos que interactúan con el frame Principal. Inicia el proceso respectivo.</td></tr>
<tr><td rowspan="3">public void run()</td><td>* Tipo:Void</td></tr>
<tr><td>* Argumentos: No tiene argumentos.</td></tr>
<tr><td>* Descripción del Método: Presenta en el cuadro de texto txtSalir el mensaje respectivo a fin de aplicación, envía al bus de datos del puerto paralelo nivel bajo a todos sus pines para detener el movimiento del motor. Detiene el proceso del hilo 3 encargado de la escritura en el puerto según corresponda la selección de velocidad - giro y del hilo 1 encargado de actualizar los estados del motor en el cuadro de texto.

Finalmente provoca un retardo de 2 segundos para cerrar la aplicación.</td></tr>
<tr><td rowspan="2">Clase Hilo3</td><td rowspan="2">public Hilo3(Principal p)</td><td>* Tipo: Void (Constructor).</td></tr>
<tr><td>* Argumentos: Principal p >> Representa el frame de la clase Principal requerido para inicializar el atributo de clase y reconocer el estado de los botones que indican cambio de acciones o permanencia.</td></tr>
</table>

		* *Descripción del Método:* Inicializa el proceso y el argumento de clase principal (p).
	public void fin()	* *Tipo:* Void.
		* *Argumentos:* No tiene argumentos.
		* *Descripción del Método:* Termina el proceso del Hilo 3.
	public void run ( )	* *Tipo:* Void.
		* *Argumentos:* No tiene argumentos.
		* *Descripción del Método:* A través del concepto de PWM se varía la velocidad del motor conectado al puente H que controla su giro. Se utiliza 3 pines del bus de datos del puerto Paralelo: • ***D0*** >> controla el terminal Enable del puente H a través del cual se programan retardos para enviar niveles altos (1) o bajos (0) según lo requiera los pulsos necesarios para una determinada velocidad. • ***D1*** >> controla la terminal Input 1 del puente H para establecer los niveles lógicos requeridos para controlar el giro del motor. • ***D2*** >> controla la terminal Input 2 del puente H para establecer los niveles lógicos requeridos para controlar el giro del motor. Niveles de control de giro ***D0(Enable) D1(Input1) D2(Input2)*** Giro a la derecha (Avance) 1 0 1 Giro a la izquierda (Retroceso) 1 1 0 Detención del motor (Ninguna) 0 0 0 *La combinación 111 no se puede presentar ya que fuerza el funcionamiento del puente H. El período establecido para el control a través de

pulsos es 12ms, distribuidos de la siguiente forma para generar los pulsos requeridos por cada velocidad:

	Permanencia en alto	***Permanencia en bajo***
Velocidad Lenta:	4 ms.	8 ms.
Velocidad Media:	6 ms.	6 ms.
Velociad Rápida:	12 ms.	0 ms.

De acuerdo a la selección de una de las opciones de velocidad (Lenta, Media, Rápida) del ComboBox (cbVelocidad) ingresa en el caso respectivo que compara el estado activo de los radio buttons avance o retroceso para escribir en bus de datos del puerto paralelo el número correspondiente a la activación de los pines empleados:

Escritura en el bus de datos para velocidad Lenta (selección del índice 1 del ComboBox cbVelocidad)

D2 D1 D0
Avance: Nivel alto por 4 ms >> 5 (dec) = 1 0 1 (bin)
Nivel bajo por 8 ms >> 4 (dec) = 1 0 0 (bin)
Retroceso: Nivel alto por 4 ms >> 3 (dec) = 0 1 1 (bin)
Nivel bajo por 8 ms >> 2 (dec) = 0 1 0 (bin)

Escritura en el bus de datos para velocidad Media (selección del índice 2 del ComboBox cbVelocidad)

D2 D1 D0
Avance: Nivel alto por 6 ms -> 5 (dec) = 1 0 1 (bin)
Nivel bajo por 6 ms -> 4 (dec) = 1 0 0 (bin)
Retroceso: Nivel alto por 6 ms -> 3 (dec) = 0 1 1 (bin)
Nivel bajo por 6 ms -> 2 (dec) = 0 1 0 (bin)

Escritura en el bus de datos para velocidad Rápida (selección del índice 3 del ComboBox cbVelocidad)

		D2 D1 D0 Avance: Nivel alto por 12 ms >> 5 (dec) = 1 0 1 (bin) Retroceso: Nivel alto por 12 ms >> 3 (dec) = 0 1 1 (bin) ***Escritura en el bus de datos para Ninguna velocidad (selección del índice 3 del ComboBox cbVelocidad)*** No existen retardos ya que no se requiere control de velocidad y las combinaciones 0 0 de input 1 y 2 representan detención del motor >> 0 (dec) = 0 0 0 (bin).
Clase PuertoParalelo	public void escribir(int dato)	** Tipo:Void.* ** Argumentos: int dato -> dato a escribir en la dirección del bus de datos del puerto paralelo.* ** Descripción del Método: Realiza un cast a dato tipo short del argumento que se recibe como int.* ** El nuevo dato se envía a través de la dirección definida para el bus de datos del puerto.*
JFrame Principal	**public Principal()**	** Tipo: Void (Constructor).* ** Argumentos: No tiene argumentos.* ** Descripción del Método:* Inicia los procesos correspondientes a los Hilos 1 y 3, configura la fuente del texto y tamaño para el cuadro de texto txtSalir encargado de desplegar los mensajes de configuración del motor. Al inicio de la aplicación la ninguna velocidad está activada por defecto, por lo que los botones de selección de giro están desactivados.
	private void btnSalirActionPerformed (java.awt.event.ActionEvent evt)	** Tipo: Void.* ** Argumentos:* Evento de escucha tipo java.awt.event. ** Descripción del Método:* Inicia el proceso del Hilo2 encargado de culminar con los procesos de los Hilos 1 y 3.

	private void cbVelocidad ActionPerforme d (java.awt.event. ActionEvent evt)	** Descripción del Método:* Evento del componente ComboBox cbVelocidad que guarda el valor del índice de selección y hace invisibles a los botones radio buttons de selección de giro si la opción de velocidad ninguna es seleccionada, caso contrario los habilita su visibilidad.
	private void rbAvanceActio n Performed (java.awt.event. ActionEvent evt)	** Descripción del Método:* Evento asociado al radiobutton avance que asigna 1 a la variable auxRB para reconocer que esta opción se ha seleccionado y compararla con su estado anterior para de modo afirmativo cambiar la literatura del cuadro de texto txtSalir.
	private void rbRetrocesoAct ionPerformed (java.awt.event. ActionEvent evt)	** Descripción del Método:* Evento asociado al radiobutton retroceso que asigna 1 a la variable auxRB para reconocer que esta opción se ha seleccionado y compararla con su estado anterior para de modo afirmativo cambiar la literatura del cuadro de texto txtSalir.

CODIGO FUENTE:

Clase Hilo1

```
import jnpout32.*;

public class Hilo1 extends Thread{
    static final int  a=444;
    Principal p;
    int auxCB=a;
    int auxRB=a;

    public Hilo1(Principal p){
        this.p=p;
        this.start();}

    public void fin(){
        this.stop();}

    public void run(){  //f=1 kHz, T=1ms
        while (true){
            try{
            if(((auxRB!=p.auxRB)&&(auxCB==p.auxCB))||(auxCB!=p.auxCB)){
                  auxCB=p.auxCB;
                  auxRB=p.auxRB;

                  if(p.auxCB== 1){  //Cambia parametros de giro respecto a  velocidad lenta
```

```
                if(p.rbAvance.isSelected()==true){
                  p.txtSalir.setText("^.^ Avance Lento -> 149 RPM.");}
                else{
                  p.txtSalir.setText("^.^ Retroceso Lento -> 149 RPM.");}
              }

              else if(p.auxCB == 2){ //Cambia parametros de giro respecto a velocidad
media
                    if(p.rbAvance.isSelected()==true){
                      p.txtSalir.setText("^.^ Avance Medio -> 746 RPM.");}
                    else{
                      p.txtSalir.setText("^.^ Retroceso Medio -> 746 RPM.");}
              }

              else if(p.auxCB==3){ //Cambia parametros de giro respecto a velocidad rapida
                    if(p.rbAvance.isSelected()==true){
                      p.txtSalir.setText("^.^ Avance Rapido -> 1493 RPM.");}
                    else{
                       p.txtSalir.setText("^.^ Retroceso Rapido -> 1493 RPM.");}
              }

              else{  //Ninguna velocidad seleccionada
                 p.txtSalir.setText("^.^ Detención del Motor ^.^");}
          }
          } catch(Exception e){p.txtSalir.setText("^-^ Error de Iteracción con el Puerto || ^-
^");}
        }
      }
}
```

Clase Hilo2

```
public class Hilo2 extends Thread{
   Hilo1 h1;
   Hilo3 h3;

   public Hilo2(Hilo1 h1, Hilo3 h3){
      this.h3=h3;
      this.h1=h1;
      this.start();}

   public void run(){
      while (true){
         try {

            h1.p.txtSalir.setText(">>>> Fin de la Aplicación <<<<");
            new PuertoParalelo().escribir(0);
```

```
            h3.fin();
            h1.fin();
            this.sleep(2000);
            System.exit(0);
            } catch (Exception e) {}
    }
    }
}
```

Clase Hilo3

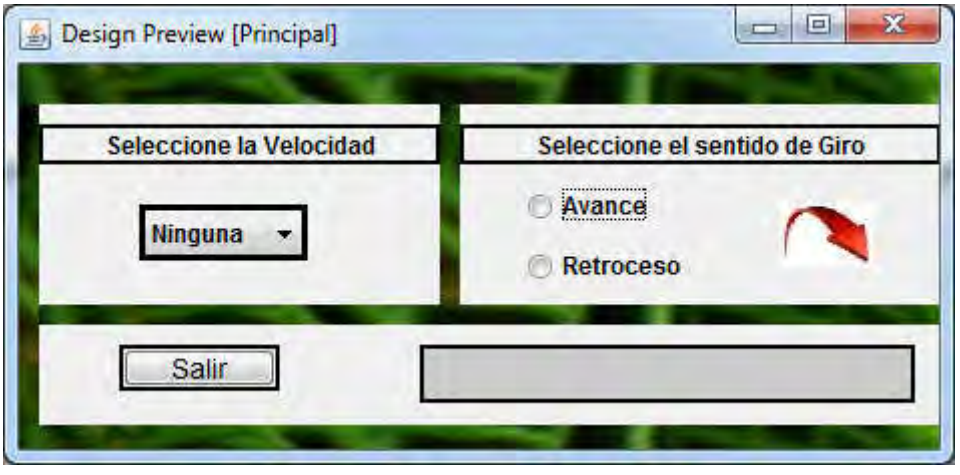
```
public class Hilo3 extends Thread{
    Principal p;

    public Hilo3(Principal p){
        this.p=p;
        this.start();}

    public void fin(){
        this.stop();}

    public void run(){  //f=1 kHz, T=1ms
        while (true){
            try{
                if(p.cbVelocidad.getSelectedIndex()== 1){  //Velicidad lenta
                  if(p.rbAvance.isSelected()==true){
                    new PuertoParalelo().escribir(5);
                    this.sleep(4);
                    new PuertoParalelo().escribir(4);
                    this.sleep(8);
                  }
                  else{
                    new PuertoParalelo().escribir(3);
                    this.sleep(4);
                    new PuertoParalelo().escribir(2);
                    this.sleep(8);}
                }

                else if(p.cbVelocidad.getSelectedIndex() == 2){ //Velocidad media
                      if(p.rbAvance.isSelected()==true){
                        new PuertoParalelo().escribir(5);
                        this.sleep(6);
                        new PuertoParalelo().escribir(4);
                        this.sleep(6);}
                      else{
                        new PuertoParalelo().escribir(3);
                        this.sleep(6);
                        new PuertoParalelo().escribir(2);
                        this.sleep(6);}
```

```
                }

                else if(p.cbVelocidad.getSelectedIndex()==3){ //Velocidad rapida
                        if(p.rbAvance.isSelected()==true){
                           new PuertoParalelo().escribir(5);
                           this.sleep(12);}
                        else{
                            new PuertoParalelo().escribir(3);
                            this.sleep(12);}
                }

                else{  //Ninguna
                   new PuertoParalelo().escribir(0);}

            }catch(Exception e){p.txtSalir.setText("^-^ Error de Iteracción con el Puerto || ^-^");}
        }
    }
}
```

Clase PuertoParalelo

```
import jnpout32.*;

public class PuertoParalelo {
    short direccion=0x378;

    public void escribir(int dato){
        new pPort().output(direccion,(short)dato);
    }

}
```

Clase Principal



```
import java.awt.Font;

public class Principal extends javax.swing.JFrame {
    public int auxCB=0;
    public int auxRB=1;
    Hilo1 h1;
    Hilo2 h2;
    Hilo3 h3;

    public Principal(){
        initComponents();
        h3=new Hilo3(this);
        h1=new Hilo1(this);
        txtSalir.setFont(new Font("Candara",Font.BOLD+Font.ITALIC,14));
        rbAvance.setVisible(false);
        rbRetroceso.setVisible(false);
    }

    private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
       h2=new Hilo2(h1,h3);
    }

    private void cbVelocidadActionPerformed(java.awt.event.ActionEvent evt) {
           auxCB=cbVelocidad.getSelectedIndex();
           rbAvance.setVisible(true);
           rbRetroceso.setVisible(true);

           if(cbVelocidad.getSelectedIndex()==0){
              rbAvance.setVisible(false);
              rbRetroceso.setVisible(false);}
    }

    private void rbAvanceActionPerformed(java.awt.event.ActionEvent evt) {
        auxRB=1;
    }

    private void rbRetrocesoActionPerformed(java.awt.event.ActionEvent evt) {
       auxRB=2;
    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Principal().setVisible(true);
            }
        });
    }
```

```
    private javax.swing.ButtonGroup Grupo1;
    public javax.swing.JButton btnSalir;
    public javax.swing.JComboBox cbVelocidad;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    public javax.swing.JRadioButton rbAvance;
    public javax.swing.JRadioButton rbRetroceso;
    public javax.swing.JTextField txtSalir;
}
```

CAPITULO 25

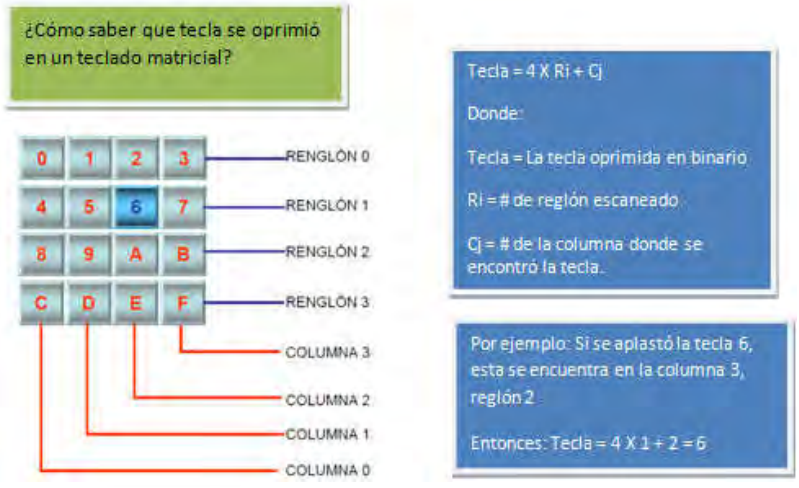
CONEXIÓN DE UN TECLADO MATRICIAL AL PUERTO PARALELO

El puerto paralelo a más de permitir sacar información al mundo exterior, también permite el ingreso, y eso es precisamente lo que se pretende en esta sección, el diseñar una pequeña aplicación, que conecte un teclado matricial al puerto paralelo.

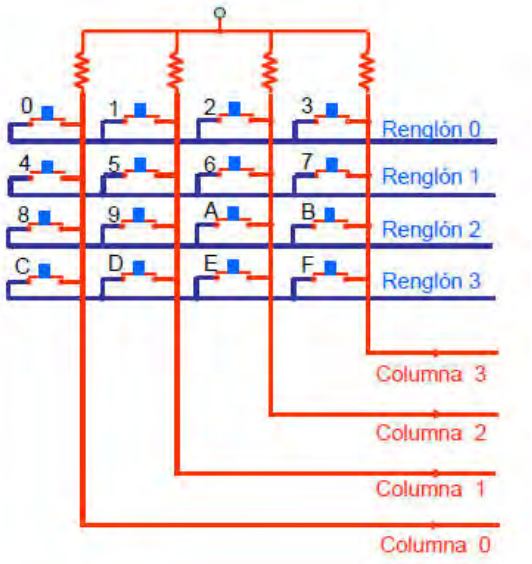
TECLADO MATRICIAL

Los teclados matriciales son ensamblados en forma de matriz. Un teclado como una matriz 4X4 – 16 teclas configuradas en 4 columnas y 4 renglones.

Cuando no se ha oprimido ninguna tecla, (todas las teclas abiertas) no hay conexiones entre renglones y columnas.



Cuando se oprime una tecla se hace una conexión entre la columna y el renglón de la tecla.



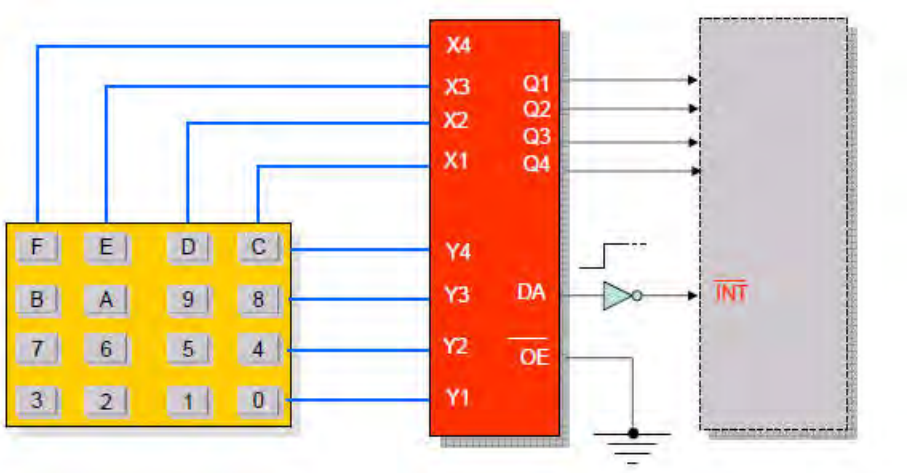
DECODIFICADOR DE TECLADO

Muchos teclados comerciales ya traen incluido un decodificador que escanea el teclado y si, una tecla es presionada, regresa un número que identifica la tecla. Otra alternativa es adquirir por separado un chip decodificador y conectarlo al teclado.

El decodificador mencionado tiene 8 entradas; las 4 entradas “X” son conectadas a las 4 columnas del teclado y las 4 entradas “Y” son conectadas a los 4 renglones. No se muestran los capacitores que gobiernan la rapidez a la que se escanea el teclado.

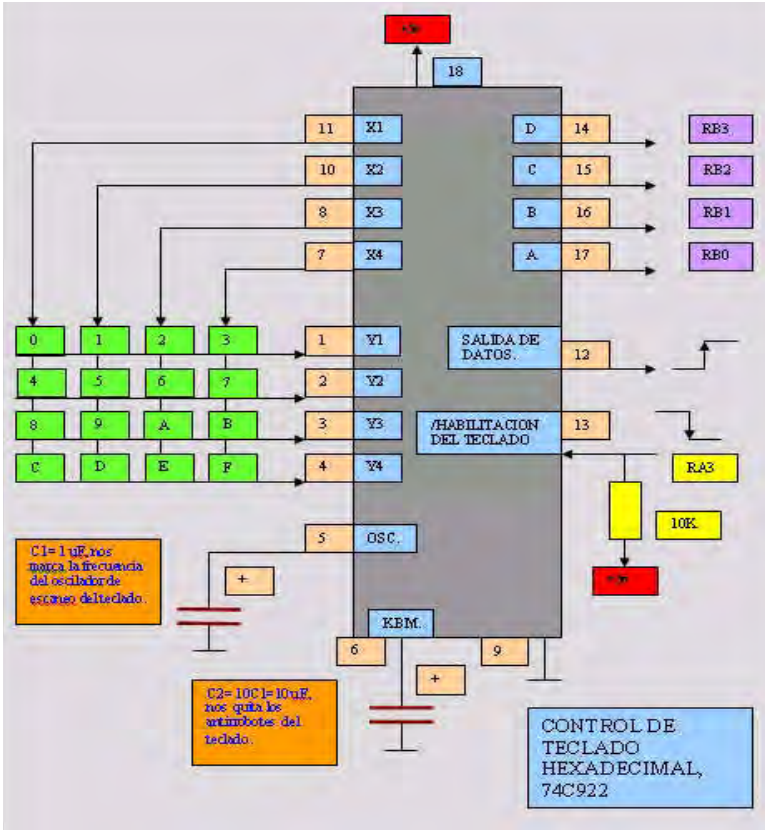
Cuando se oprime una tecla el código de 4 bits de la tecla (con 16 teclas, los códigos están entre 0000 y 1111 en binario) aparecerá las 4 líneas de salida y la línea de dato disponible (DA) se pone en BAJO. Si se conecta a una línea de interrupción el microprocesador será interrumpido cuando se oprima alguna tecla. La rutina de servicio de la Interrupción, entonces lee los 4 bits y procesa el dato.

El chip del decodificador se encarga de eliminar el rebote de las teclas, lo que libera al programador de esta responsabilidad, esta es una ventaja al usar un chip decodificador.



DECODIFICADOR MM74C922

El circuito consta de 8 entradas X1, X2, X3 y X4 y Y1, Y2, Y3, Y4, donde se conectan las 4 filas y 4 columnas del teclado, la salida en función de la tecla pulsada es en formato binario en las patillas A, B, C y D y van a través de una báscula tipo D, con lo que el valor de una tecla pulsada se mantiene hasta que se pulsa otra y se sobre escribe el antiguo valor.



Mapa de variables

Clase hilo		
Nombre	**Tipo**	**Descripción**
valor	short	Almacena los datos que se obtienen desde el puerto paralelo
direccion	short	Almacena la dirección del puerto de estado
puerto	pPort	Objeto del puerto paralelo
tecla	int	Variable de control del bus de estado, cuyo valor define qué tecla se ha presionado
mensaje	String	Cadena de caracteres que almacena el texto del cuadro de texto en el frame
cont	int	Variable que funciona como contador, el cual aumenta cada vez que se realice una pulsación en una misma tecla
i	int	Bucle que controla las teclas repetidas
j	int	Bucle que controla las teclas distintas
mensaje1	String	Cadena de caracteres que se envía a imprimir en el cuadro de texto en el frame
mensaje2	String	Representa a un caracter que se añadirá a la cadena de caracteres que se va a imprimir
Clase Principal		
obj	Mensaje	Objeto del frame Mensaje, el cual visibilizará el frame al momento de correr el programa el cual se abrirá con un título y un tamaño definido en la clase principal
Frame mensaje		
btnsalir	JButton	Botón que al momento de darle click, se cierra la aplicación del programa
menuSalir	JMenuItem	Opción del menú Archivo que al momento de darle click, se cierra la aplicación del programa
menuGuia	JMenuItem	Opción del menú Ayuda que al momento de darle click, se abre la ventana ayuda que muestra cómo se deben ingresar las variables de texto
txtMensaje	JTextField	Campo de texto donde se mostrarán los caracteres ingresados por el teclado matricial
Frame Ayuda		
btnsalir	JButton	Botón que al momento de darle click, se cierra la aplicación del programa

Explicación de los Métodos o Funciones

Clase Hilo

```
import jnpout32.*;


public class Hilo extends Thread {
    short valor=0;
    short direccion=0x379;
```

```
pPort puerto=new pPort();
int tecla=0;
String mensaje;

int cont=0,i=4,j=3;
int numero=0;
int bandera=0;

public Hilo() {
   //Inicio de la clase Hilo
   this.start();
}
```

En el constructor **Hilo()** se inicia el hilo de ejecución que se está utilizando en el programa.

```
public void run()
{
   //Asigna pantalla=texto de txtPantalla del Frame
   mensaje=Mensaje.txtMensaje.getText();
   //Llama funcion Dato()
   Dato();
   //Bucle para control de teclas distintas
   while (j!=0)
   {
      //Asigna a i=3 que activa al bucle de teclas repetidas
      i=3;
      //Llama Funcion Dato()
      Dato();
      //Compara la tecla presionada
        switch (tecla)
        {
           //Caso tecla1//tecla B
           case 11:
              //Llama a Funcion Dato()
              Dato();
              //Control de presion de tecla repetitiva
              if (tecla==119)
              {
                 //Inicio de bucle de teclas repetidas
                 while (i!=0){
                    //Control de impresion de carácter
                    cont++;
                    //Compara las veces que se presiona la
                    //misma tecla
                    switch(cont)
                    {
                       case 1:
```

```
                    //Llama Funcion Imprimir mensaje
                    imprimirMensaje(mensaje, ".");
                    //Llama a la funcion Dato()
                    Dato();
                     if (tecla!=119){
                        //se encera la variable de
                        //de control del segundo bucle
                        i=0;
                        //Se guarda en pantalla
                        //el texto que ha sido ingresado
                        mensaje=Mensaje.txtMensaje.getText();
                        //termina el switch(cont)
                        break;
                     }
                    //termina el switch(cont)
                     break;
                case 2:
                    imprimirMensaje(mensaje, ",");
                    Dato();
                     if (tecla!=7)
                     {
                        i=0;
                        mensaje=Mensaje.txtMensaje.getText();
                        break;
                     }
                     break;
                case 3:
                    imprimirMensaje(mensaje, "?");
                    Dato();
                     if (tecla!=119)
                     {
                        i=0;
                        mensaje=Mensaje.txtMensaje.getText();
                        break;
                     }
                     break;
                case 4:
                    imprimirMensaje(mensaje, "1"); Dato();
                     //numero=H1.tecla;
                     if (tecla!=119){
                        i=0;
                        mensaje=Mensaje.txtMensaje.getText();break;}
                     break;
                default: cont=0;
                break;
            }
        }
    }
break;
```

```
//Tecla2
case 23:
    Dato();
    if (tecla==23){
    while (i!=0){
        cont++;
        switch(cont)
        {
            case 1:imprimirMensaje(mensaje, "a"); Dato();
                if (tecla!=23){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 2:imprimirMensaje(mensaje, "b"); Dato();
                if (tecla!=23){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 3:imprimirMensaje(mensaje, "c"); Dato();
                if (tecla!=23){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 4:imprimirMensaje(mensaje, "2"); Dato();
                if (tecla!=23){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;

            case 5:imprimirMensaje(mensaje, " "); Dato();
                if (tecla!=23){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            default: cont=0;break;
        }
    } }break;

//Tecla3
case 87:
    Dato();
    if (tecla==87){
    while (i!=0){
        cont++;
        switch(cont)
        {
            case 1:imprimirMensaje(mensaje, "d"); Dato();
                //numero=H1.aux;
```

```
                if (tecla!=87){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 2:imprimirMensaje(mensaje, "e"); Dato();
                //numero=H1.aux;
                if (tecla!=87){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 3:imprimirMensaje(mensaje, "f"); Dato();
                //numero=H1.aux;
                if (tecla!=87){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 4:imprimirMensaje(mensaje, "3"); Dato();
                //numero=H1.aux;
                if (tecla!=87){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            default: cont=0;break;
        }
    } }break;

//Tecla4
case 55:
    Dato();
    if (tecla==55){
    while (i!=0){
        cont++;
        switch(cont)
        {
            case 1:imprimirMensaje(mensaje, "g"); Dato();
                if (tecla!=55){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 2:imprimirMensaje(mensaje, "h"); Dato();
                if (tecla!=55){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 3:imprimirMensaje(mensaje, "i"); Dato();
                if (tecla!=55){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
```

```
            case 4:imprimirMensaje(mensaje, "4"); Dato();
                if (tecla!=55){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            default: cont=0;break;
        }
    } }break;

//Tecla5
case 31:
    Dato();
    if (tecla==31){
    while (i!=0){
        cont++;
        switch(cont)
        {
            case 1:imprimirMensaje(mensaje, "j"); Dato();
                if (tecla!=31){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 2:imprimirMensaje(mensaje, "k"); Dato();
                if (tecla!=31){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 3:imprimirMensaje(mensaje, "l"); Dato();
                if (tecla!=31){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 4:imprimirMensaje(mensaje, "5"); Dato();
                if (tecla!=31){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            default: cont=0;break;
        }
    } }break;
//Tecla6
case 95:
    Dato();
    if (tecla==95){
    while (i!=0){
        cont++;
        switch(cont)
        {
            case 1:imprimirMensaje(mensaje, "m"); Dato();
```

```
                if (tecla!=95){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 2:imprimirMensaje(mensaje, "n"); Dato();
                if (tecla!=95){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 3:imprimirMensaje(mensaje, "o"); Dato();
                if (tecla!=95){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 4:imprimirMensaje(mensaje, "6"); Dato();
                if (tecla!=95){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            default: cont=0;break;
        }
    } }break;
//Tecla7
case 63: Dato();
    if (tecla==63){
    while (i!=0){
        cont++;
        switch(cont)
        {
            case 1:imprimirMensaje(mensaje, "p"); Dato();
                if (tecla!=63){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 2:imprimirMensaje(mensaje, "q"); Dato();
                if (tecla!=63){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 3:imprimirMensaje(mensaje, "r"); Dato();
                if (tecla!=63){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 4:imprimirMensaje(mensaje, "s"); Dato();
                if (tecla!=63){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
```

```
            case 5:imprimirMensaje(mensaje, "7"); Dato();
                if (tecla!=63){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            default: cont=0;break;
        }
    } }break;
//Tecla8
case 119:
    Dato();
    if (tecla==119){
    while (i!=0){
        cont++;
        switch(cont)
        {
            case 1:imprimirMensaje(mensaje, "t"); Dato();
                if (tecla!=119){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 2:imprimirMensaje(mensaje, "u"); Dato();
                if (tecla!=119){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 3:imprimirMensaje(mensaje, "v"); Dato();
                if (tecla!=119){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 4:imprimirMensaje(mensaje, "8"); Dato();
                if (tecla!=119){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            default: cont=0;break;
        }
    } }break;
//Tecla9
case 127:
    Dato();
    if (tecla==127){
    while (i!=0){
        cont++;
        switch(cont)
        {
            case 1:imprimirMensaje(mensaje, "w"); Dato();
                if (tecla!=127){
```

```
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 2:imprimirMensaje(mensaje, "x"); Dato();
                if (tecla!=127){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 3:imprimirMensaje(mensaje, "y"); Dato();
                if (tecla!=127){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 4:imprimirMensaje(mensaje, "z"); Dato();
                if (tecla!=127){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 5:imprimirMensaje(mensaje, "9"); Dato();
                if (tecla!=127){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            default: cont=0;break;
        }
    } }break;
//Tecla*

//Tecla0
case 5:
    Dato();
    if (tecla==95){
    while (i!=0){
        cont++;
        switch(cont)
        {
            case 1:imprimirMensaje(mensaje, " "); Dato();
                if (tecla!=95){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            case 2:imprimirMensaje(mensaje, "0"); Dato();
                if (tecla!=95){
                    i=0;
                    mensaje=Mensaje.txtMensaje.getText();break;}
                break;
            default: cont=0;break;
        }
    } }break;
```

```
                // Tecla#
                case 6:
                    Dato();
                    if (tecla==63){
                    while (i!=0){
                        cont++;
                        switch(cont)
                        {
                            case 1:imprimirMensaje(mensaje, "#"); Dato();
                                if (tecla!=63){
                                    i=0;
                                    mensaje=Mensaje.txtMensaje.getText();break;}
                                break;
                            default: cont=0;break;
                        }
                    } }break;

                //Tecla_enter
                case 3:
                    Dato();
                    if (tecla==31){
                    while (i!=0){
                        cont++;
                        switch(cont)
                        {
                            case 1:imprimirMensaje(mensaje, "\n"); Dato();
                                //numero=H1.aux;
                                if (tecla!=31){
                                    i=0;
                                    mensaje=Mensaje.txtMensaje.getText();break;}
                                break;
                            default: cont=0;break;
                        }
                    } }break;

                // default: if(tecla==119 ||tecla==111 ||tecla==127){
                            //System.exit(0);break;}
                            //break;
            }
        }
}
    //Implementacion de la Funcion de Capturar Dato del bus de estado
public void Dato()
{
    try{
        //Tiempo de pulso
        this.sleep(300);
        //aux guarda el valor que retorna la funcion obtenerDatos()
```

```
        tecla=obtenerDatos();
        //Imprime en la ventana de salida los datos capturados
        //en el bus de estado
        System.out.println(String.valueOf(tecla));
    }
        catch (Exception e){ }
}
```

En el método sobrecargado **run()** se programa el código del programa, en donde a partir de la tecla ingresada, se verifica por medio de un swith que tecla ha sido ingresada, posteriormente se verifica cuantas veces se ha presionado la misma tecla. Dependiendo del número de veces que se ha presionado la tecla, por medio de otro switch se define que carácter debe imprimirse en pantalla dependiendo de la tecla ingresada.

Si se presiona la primera tecla, al pulsar una sola vez se mandará imprimir el carácter “.”; a la segunda se mandará a imprimir “,”; a la tercera “?”; y a la cuarta “1”.

El mismo proceso se repetirá con las otras 12 teclas, donde el resto de caracteres que se deben imprimir dependerán del valor del entero **tecla** que se obtiene al pulsar cualquier botón y del número de veces que presione la misma tecla.

```
public void Dato()
{
    try{
        //Tiempo de pulso
        this.sleep(300);
        //aux guarda el valor que retorna la funcion obtenerDatos()
        tecla=obtenerDatos();
        //Imprime en la ventana de salida los datos capturados
        //en el bus de estado
        System.out.println(String.valueOf(tecla));
    }
        catch (Exception e){ }
}
```

En el método **Dato()**, se define la velocidad en que se debe ejecutar el programa, a 300 ms. Luego se guarda el valor que se obtiene del método **obtenerDatos** en el entero **tecla** y se imprime dicho valor.

```
public int obtenerDatos()
{
    valor=puerto.input(direccion);
    //this.stop();
    return(valor);
}
```

En el método **obtenerDatos()** en el entero **valor** se guarda el dato obtenido en el puerto de estado del puerto paralelo, dicho valor será retornado para su uso en los demás métodos.

```
    public void imprimirMensaje(String mensaje1,String mensaje2)
    {
        //Imprime el carácter en el Frame
          Mensaje.txtMensaje.setText(mensaje1.concat(mensaje2));
    }
}
```

En el método **imprimirMensaje()** se imprime en el textfield **txtMensaje** los caracteres ingresados por el teclado matricial, dicho texto es una cadena de caracteres, donde otros caracteres se van añadiendo a dicho texto dependiendo de la tecla ingresada.

Clase Mensaje

```
public class Mensaje extends javax.swing.JFrame {

    /** Creates new form Teclado */
    public Mensaje() {
        initComponents();
        new Hilo();

    }
```

En el constructor **Mensaje** se inicializan los componentes del frame y además se llama al método **Hilo()** para que el hilo de ejecución se inicialice y empiece a funcionar la aplicación.

```
private void btnLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
    txtMensaje.setText(" ");
}

    /**
    * @param args the command line arguments
    */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Mensaje().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton btnLimpiar;
    private javax.swing.JButton btnSalir;
```

```
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenu menuArchivo;
    private javax.swing.JMenu menuAyuda;
    private javax.swing.JMenuItem menuGuia;
    private javax.swing.JMenuItem menuSalir;
    public static javax.swing.JTextField txtMensaje;
    // End of variables declaration

}
```

Clase Principal

```
public class Principal {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Mensaje obj=new Mensaje();
        obj.setVisible(true);
        obj.setTitle("Teclado matricial");
        obj.setLocation(300, 290);
    }

    }
```

CAPITULO 26

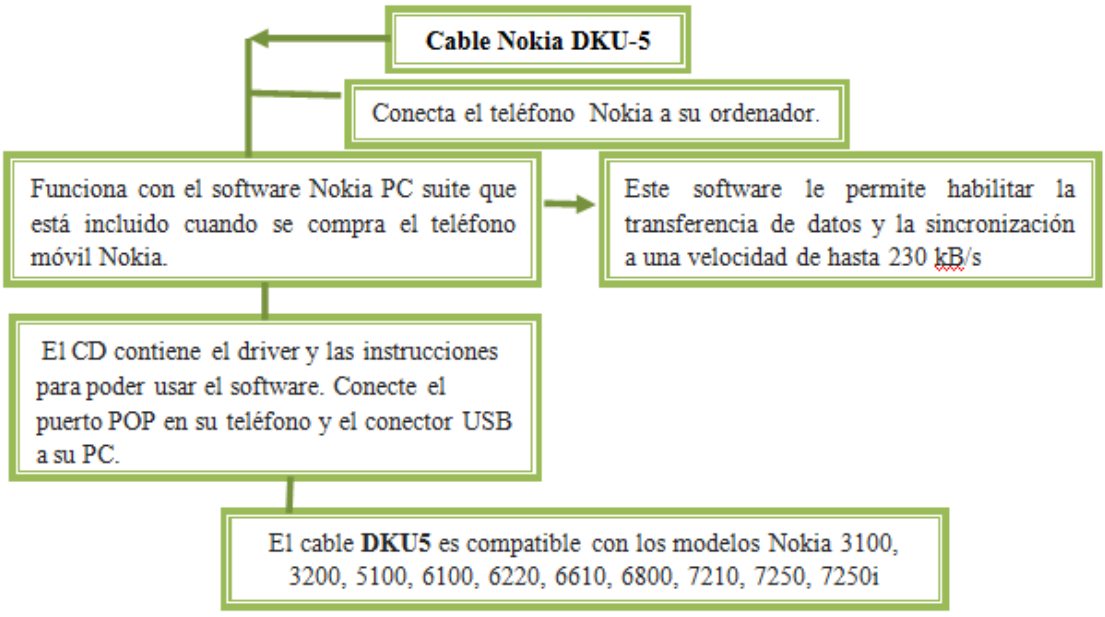
CONTROL DE MODEMS CELULARES CON COMANDOS AT

La PC permite interactuar con el mundo y con el usuario, pero si la necesidad de interacción va mas alla de unos pocos metros, debe existir un medio que permita la transmisión de información a otras personas o equipos, que se encuentren a grandes distancias. Es así que el principio de la comunicación en paralelo, no cumple con esta necesidad, pero el principio de la comunicación serial aunque un poco más lenta presenta ciertas características que permite transmisión a mayores distancias. Muchos dispositivos a más de las computadoras incorporaron la comunicación serial como los celulares que por medio del estándar RS-232 permite su control a través de una serie de instrucciones llamado comandos AT.

La aplicación que se presenta permite el control por medio del puerto serie de un celular, empleando comandos AT, para el envío y la recepción de mensajes de texto, para ello se va a utilizar un celular (Nokia 3220) por medio del cable serial DKU-5 y los comandos AT para el envío y la recepción de mensajes SMS.

La comunicación serial consiste en el envío de un bit de información de manera secuencial, esto es, un bit a la vez y a un ritmo acordado entre el emisor y el receptor. La comunicación serial en computadores ha seguido los estándares definidos en 1969 por el RS-232 (Recommended Standard 232) que establece niveles de voltaje, velocidad de transmisión de los datos, etc. Por ejemplo, este protocolo establece un nivel de -12v como un uno lógico y un nivel de voltaje de +12v como un cero lógico (por su parte, los microcontroladores emplean por lo general 5v como un uno lógico y 0v como un cero lógico).

El celular que se utilizará para realizar esta investigación es el Nokia 3220 ya que tiene muchos beneficios para esta comunicación.



COMANDOS AT

Los comandos AT son instrucciones codificadas que conforman un lenguaje de comunicación entre el hombre y un Terminal MODEM, para así poder configurarlo y proporcionarle instrucciones, tales como marcar un número de teléfono.

Aunque la finalidad principal de los comandos AT es la comunicación con módems, la telefonía móvil GSM también ha adoptado como estándar este lenguaje para poder comunicarse con sus terminales. De esta forma, todos los teléfonos móviles GSM poseen un juego de comandos AT específico que sirve de interfaz para configurar y proporcionar instrucciones a los terminales, permiten acciones tales como realizar llamadas de datos o de voz, leer y escribir en la agenda de contactos y enviar mensajes SMS, además de muchas otras opciones de configuración del terminal.

Comandos para SMS

a) AT+CPMS: Seleccionar lugar de almacenamiento de los SMS

b) AT+CMGF: Seleccionar formato de los mensajes SMS
c) AT+CMGR: Leer un mensaje SMS almacenado
d) AT+CMGL: Listar los mensajes almacenados
e) AT+CMGS: Enviar mensaje SMS
f) AT+CMGW: Almacenar mensaje en memoria
g) AT+CMSS: Enviar mensaje almacenado
h) AT+CSCA: Establecer el Centro de mensajes a usar
i) AT+ WMSC: Modificar el estado de un mensaje.

Comandos del servicio de red

a) AT+CSQ: Obtener calidad de la señal
b) AT+COPS: Selección de un operador
c) AT+CREG: Registrarse en una red
d) AT+WOPN: Leer nombre del operador

Comandos de seguridad:

a) AT+CPIN: Introducir el PIN
b) AT+CPINC: Obtener el número de reintentos que quedan
c) AT+CPWD: Cambiar password

Como Enviar SMS con Hyperterminal y Comandos AT

Este es un caso para el envío de un SMS a través de un celular conectado a un computador por medio de puerto serie. Obviamente para que se logre esta comunicación se necesita del hardware y el software necesario; y para manipular el teléfono celular se utiliza Comandos AT+.

Para este ejemplo se emplea un telefono Nokia 3220 y para la conexión a un computador el cable DKU-5 el cual viene con su driver PL 2303 USB-serial debidamente conectado el celular al computador e instalado el driver se puede hacer el envío de datos.

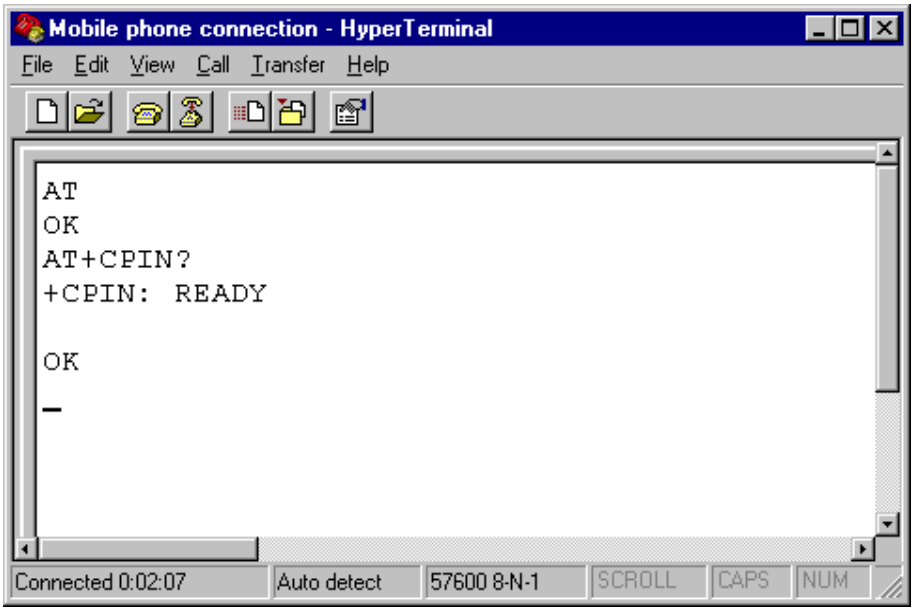
1) Abrir el hyperterminal
2) Ingresar un nombre en la conexión
3) Seleccionar el puerto COM correspondiente del teléfono celular
4) Configurar el puerto con 9600 bps; 8 bits de datos; ninguna paridad; 1 bit de parada; control de flujo vía hardware.

Terminada la configuración se puede iniciar las pruebas de comandos AT en el HyperTerminal. Para hacer una rápida prueba se escribe el comando AT+CGMM para saber el modelo del teléfono.

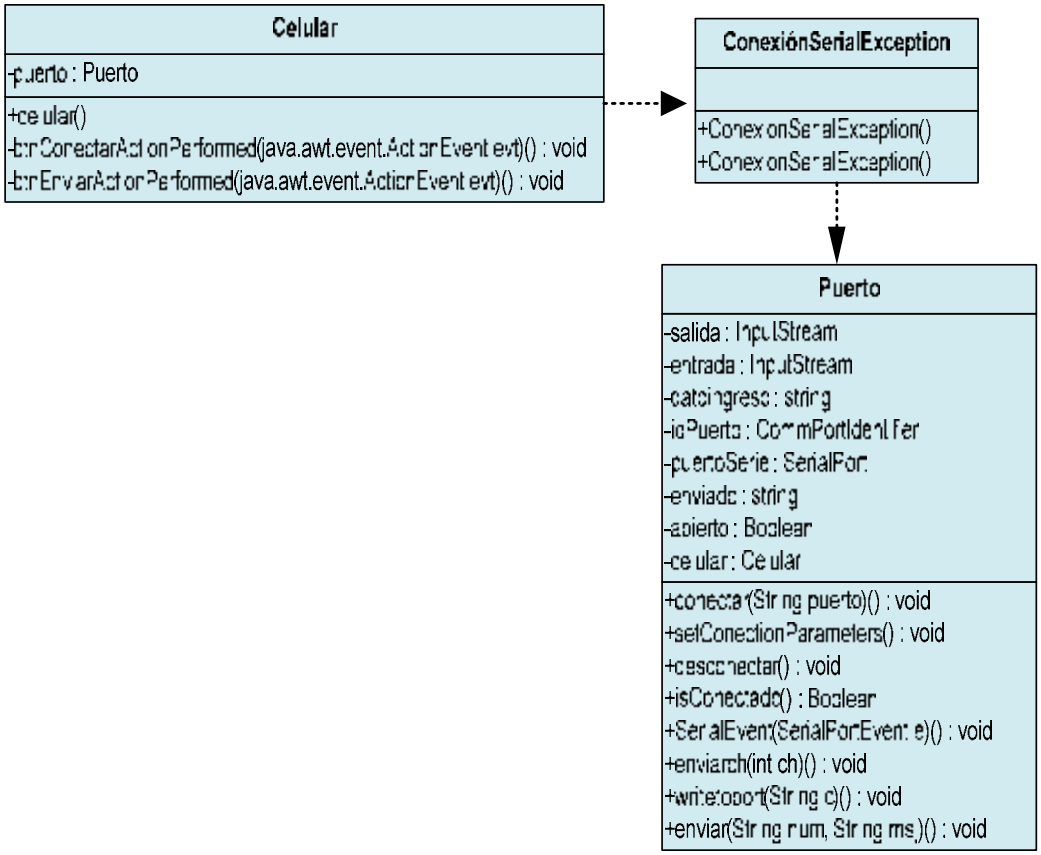
Ahora si para el envío de un SMS se debe escribir el comando AT+CMGS con el número celular del destinatario y se presiona “Enter” seguido que salga el signo (>) se ingresa el texto del mensaje terminado el mensaje se presiona CTRL + Z.

Si ha sido un envío exitoso el comando emitirá un OK caso contrario devolverá un número de error , esto puede ser porque no se tiene saldo en el celular, se perdió la señal o

simplemente está alguna parte mal escrita en el comando. Con esto se da una pauta de como sería la manipulación de un teléfono celular GSM utilizando comandos AT+.



A continuación se presenta el diagrama de UML del proyecto.



Las principales variables empleadas son las siguientes:

ConexionSerialException	No contiene variables
Puerto	**celular**.es un objeto de la clase Celular que se lo usa en la clase Puerto para ser llamado.
	idPuerto: Es una variable privada que es quien identifica y guarda al puerto que se está utilizando, en nuestro caso COM3.
	puertoSerie: Es el puerto serial en si, primero por medio del idPuerto se lo identifica,sin embargo es aquí en el SerialPort que se guarda el puerto y tiene la posibilidad de activarlo o desactivarlo cuando sea necesario.
	salida: Es la variable del puerto de salida que es quien notifica si se desea mandar datos desde la computadora hacia algo externo.
	entrada:Es la variable del puerto de entrada que es quien notifica si se desea mandar datos de un dispositivo externo a la computadora.
	abierto: Es una variable boleana que determina si está abierto el puerto me envía un true caso contrario un false
	datoingreso: Es una variable String que es en donde se escribe el mensaje.
	enviado: Es una variable String en donde se escribe el remitente.
Celular	**puerto:** declaración de un objeto de tipo puerto, es para que poder acceder a los métodos, de la clase Puerto, necesarios para desarrollar la interfaz con el usuario.

Explicación de los métodos o funciones:

CONTROL DE UN CELULAR POR MEDIO DE COMANDOS AT	
ConexiónSerialException	En esta clase se encuentran dos constructores: a) Constructor sin parámetros: Llama al constructor con parámetros tipo String. b) Constructor con parámetros tipo String: Llama al constructor principal e inicializa la variable usando el texto enviado por el anterior constructor.
	En esta clase se encuentran un constructor con parámetros recibe una variable de tipo Celular e inicializa dicha variable.

Puerto	**conectar:** recibe una variable de tipo String y no retorna valores. Éste método es para establecer la conexión entre la PC y el celular mediante el cable DKU-5 y el uso de los comandos AT.
	setConnectionParameters():éste método es sin atributos y permite establecer los parámetros de la comunicación serial . En este método es en donde se establecen los parámetros más importantes del protocolo, estos valores son : velocidad con que se envía = 9600 bps, 8 bits de datos, 1 bit de parada y sin paridad.
	desconectar: no recibe ni retorna valores. Éste método permite finalizar la comunicación con nuestro celular.
	isConectado: El método indicará el estado de la conexión retornando un valor de false o true en la variable correspondiente.
	serialEvent: recibe un parámetro, pero no retorna valores. Comprueba si el puerto está habilitado, lee la respuesta que envía el celular a la PC mediante comandos AT.
	enviarch: recibe un parámetro de tipo entero y no envía ningún valor. Permite enviar letra por letra por el puerto serial, obviamente utilizando comandos AT.
	writetoport: recibe un parámetro de tipo String. Éste método se utiliza para poder emplear los comandos AT.
	enviar: recibe dos parámetros de tipo String que corresponden al número de teléfono al cual se va a enviar el sms y el texto del mensaje. Éste método se encarga de enviar el mensaje por el puerto hacia el celular.

Celular	**Celular:** es un constructor de la clase.
	btnConectarActionPerformed: permite llamar al método **conectar()** de la clase Puerto para establecer conexión, se debe enviar como atributo el nombre del puerto que se escribe en el campo de texto que se encuentra en la interfaz. Una vez conectados se habilita el botón para poder desconectar el teléfono.
	btnEnviarActionPerformed: en este método se realiza la llamada método ***enviar()*** que requiere como parámetros el número telefónico que se escribe en el campo de texto correspondiente localizado en la interfaz, y el mensaje de texto que se escribe en el área de texto destinada para realizar ésta operación**.**

CODIGO FUENTE:

Clase ConexionSerialException

```
public class ConexionSerialException extends Exception {

    public ConexionSerialException(String msg) {
        super(msg);
    }

    public ConexionSerialException() {
        super();
    }
}
```

Clase Puerto

```
import java.util.logging.Level;
import java.util.logging.Logger;
import java.io.*;
import java.util.TooManyListenersException;
import javax.comm.CommPortIdentifier;
import javax.comm.PortInUseException;
import javax.comm.SerialPort;
import javax.comm.SerialPortEvent;
import javax.comm.SerialPortEventListener;
import javax.comm.UnsupportedCommOperationException;

public class Puerto implements SerialPortEventListener {

    private OutputStream salida;
    private InputStream entrada;
    private String datoingreso;
    private CommPortIdentifier idPuerto;
    private SerialPort puertoSerie;
    private String enviado;
    private boolean abierto = false;
    Celular celular;

    public Puerto(Celular celular) {
        this.celular = celular;
    }

    public void conectar(String puerto) throws ConexionSerialException {
        try {
            idPuerto = CommPortIdentifier.getPortIdentifier(puerto);
        } catch (Exception e) {
            e.printStackTrace();
            throw new ConexionSerialException(e.getMessage());
```

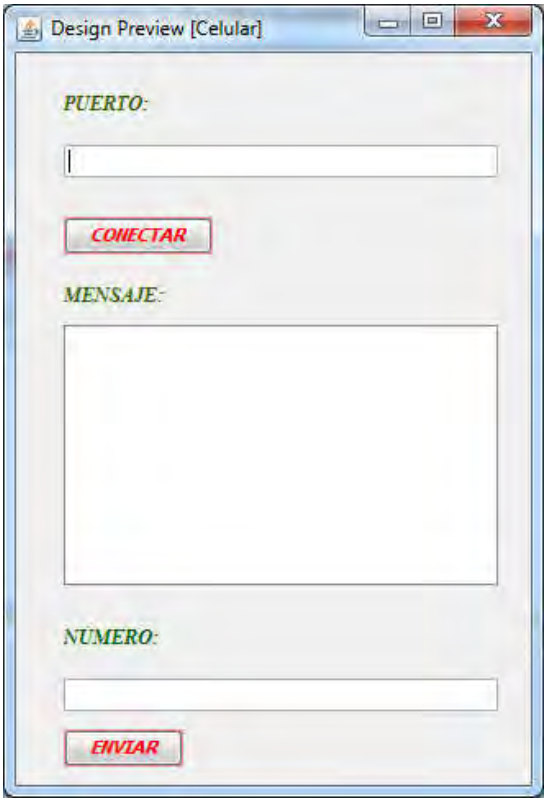
```
        }

        try {
            puertoSerie = (SerialPort) idPuerto.open("SerialDemo", 30000);
        } catch (PortInUseException e) {
            throw new ConexionSerialException(e.getMessage());
        }

        try {
            setConnectionParameters();
        } catch (ConexionSerialException e) {
            puertoSerie.close();
            throw e;
        }
        try {
            salida = puertoSerie.getOutputStream();
            entrada = puertoSerie.getInputStream();
        } catch (IOException e) {
            puertoSerie.close();
            throw new ConexionSerialException("Error abriendo puerto I/O");
        }
        try {
            puertoSerie.addEventListener(this);
        } catch (TooManyListenersException e) {
            puertoSerie.close();
            throw new ConexionSerialException("Demasiados listeners añadidos");
        }
        puertoSerie.notifyOnDataAvailable(true);
        puertoSerie.notifyOnBreakInterrupt(true);
        try {
            puertoSerie.enableReceiveTimeout(30);
        } catch (UnsupportedCommOperationException e) {
        }
        abierto = true;
        this.writetoport("ate0\r");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException ex) {
            Logger.getLogger(Puerto.class.getName()).log(Level.SEVERE, null, ex);
        }
        this.writetoport("at+cnmi=1,1,0,0,0\r");

    }

    public void setConnectionParameters() throws ConexionSerialException {
        try {
            puertoSerie.setSerialPortParams(9600,                SerialPort.DATABITS_8,
SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
        } catch (UnsupportedCommOperationException e) {
```

```
            throw new ConexionSerialException("Unsupported parameter");
        }
    }

    public void desconectar() {
        if (!abierto) {
            return;
        }
        if (puertoSerie != null) {
            try {
                salida.close();
                entrada.close();
            } catch (IOException e) {
                System.err.println(e);
            }
            puertoSerie.close();
        }
        abierto = false;
    }

    public boolean isConectado() {
        return abierto;
    }

    public void serialEvent(SerialPortEvent e) {
        StringBuffer inputBuffer = new StringBuffer();
        int newData = 0;
        switch (e.getEventType()) {
            case SerialPortEvent.DATA_AVAILABLE:
                datoingreso = "";
                while (newData != -1) {
                    try {
                        newData = entrada.read();
                        if (newData == -1) {
                            break;
                        }
                        if ('\r' == (char) newData) {
                            inputBuffer.append('\n');
                        } else {
                            inputBuffer.append((char) newData);
                        }
                        datoingreso = datoingreso + (char) newData;
                    } catch (IOException ex) {
                        System.err.println(ex);
                        return;
                    }
                }

                if (datoingreso.length() > 5) {
```

```
                for (int i = 0; i < datoingreso.length() - 5; i++) {
                    if (datoingreso.substring(i, i + 5).equals("ERROR")) {
                        celular.mensaje("Mensaje No Enviado");
                        enviado = "no";
                    } else if (datoingreso.substring(i, i + 5).equals("+CMGS")) {
                        celular.mensaje("Mensaje Enviado");
                        enviado = "si";
                    } else if (datoingreso.substring(i, i + 5).equals("+CMTI")) {
                        this.writetoport("at+cmgf=1\r");
                        //System.out.println(datain);
                        try {
                            Thread.sleep(2000);
                        } catch (InterruptedException ex) {
                            Logger.getLogger(Puerto.class.getName()).log(Level.SEVERE,     null,
ex);
                        }
                        this.writetoport("at+cmgr=" + datoingreso.substring(i + 12, i + 14).trim() +
"\r");
                        //System.out.println(datain.substring(i+12, i + 14).trim());
                    } else if (datoingreso.substring(i, i + 5).equals("+CMGR")) {
                        int j = datoingreso.indexOf("+", i + 1);
                        int k = datoingreso.indexOf(",", j);
                        int l = datoingreso.indexOf("-", j);
                        celular.mensaje("Remitente:"  +  datoingreso.substring(j,  k  -  1)  +
"\nMensaje:" + datoingreso.substring(l + 4, datoingreso.length() - 4));
                        //System.out.println("Remitente:"+datain.substring(j,                    k-
1)+"\nMensaje:"+datain.substring(l+4, datain.length()-4));
                    }
                }
            }

            case SerialPortEvent.BI:
        }
    }

    public void enviarch(int ch) {
        enviado = "";
        int iCHR34Val = ch;
        char cCHR34 = (char) iCHR34Val;
        this.writetoport("" + cCHR34);
    }

    public void writetoport(String c) {
        try {
            this.salida.write(c.getBytes());
        } catch (IOException ex) {
            Logger.getLogger(Puerto.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

```
    public void enviar(String num, String msj) {

        int iCHR34Val = 34;
        char cCHR34 = (char) iCHR34Val;
        int iCHR26Val = 26;
        char cCHR26 = (char) iCHR26Val;
        this.writetoport("at+cmgf=1\r");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException ex) {
            Logger.getLogger(Puerto.class.getName()).log(Level.SEVERE, null, ex);
        }
        this.writetoport("at+cmgs=" + cCHR34 + num + cCHR34 + "\r");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException ex) {
            Logger.getLogger(Puerto.class.getName()).log(Level.SEVERE, null, ex);
        }
        this.writetoport(msj + cCHR26);
    }
}
```

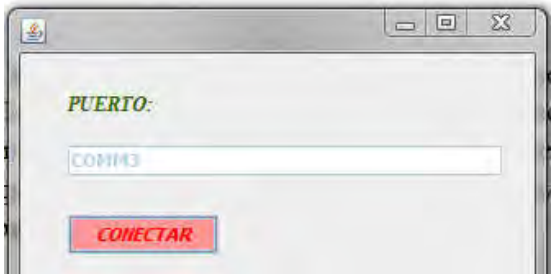
Clase Celular



```
import java.awt.Frame;
import javax.swing.JOptionPane;

public class Celular extends javax.swing.JFrame {

    Puerto puerto = new Puerto(this);

    public Celular() {
        initComponents();
    }
    private void btnConectarActionPerformed(java.awt.event.ActionEvent evt) {

        if (this.btnConectar.getText().equals("CONECTAR")) {
            try {

                puerto.conectar(this.txtPuerto.getText());
                btnConectar.setText("DESCONECTAR");
                this.btnEnviar.setEnabled(true);
                this.txtMensaje.setEnabled(true);
                this.txtNumero.setEnabled(true);
                this.btnEnviar.setEnabled(true);

            } catch (ConexionSerialException ex) {
            }
        } else {

            puerto.desconectar();
            btnConectar.setText("CONECTAR");
            this.btnEnviar.setEnabled(false);
            this.txtMensaje.setEnabled(false);
            this.txtNumero.setEnabled(false);
            this.btnEnviar.setEnabled(false);

        }
        this.txtPuerto.setEnabled(false);

    }

    private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt) {
        puerto.enviar(this.txtNumero.getText(), this.txtMensaje.getText());
        this.txtMensaje.setText("");
    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {

            public void run() {
                new Celular().setVisible(true);
            }
```

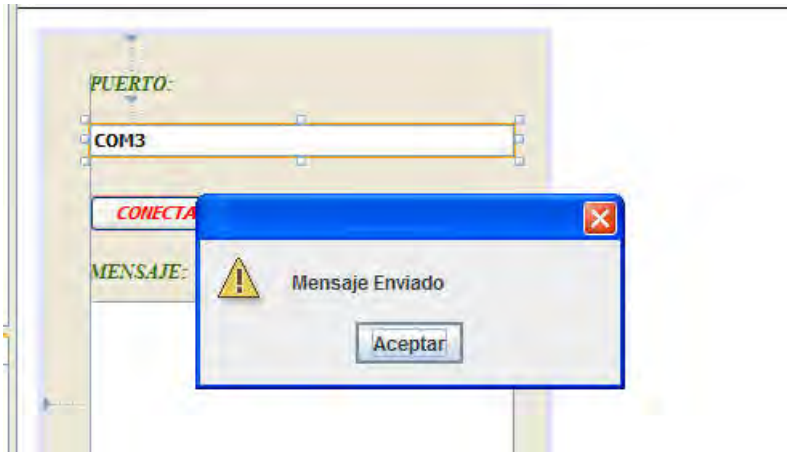
```
        });
    }

    private javax.swing.JButton btnConectar;
    private javax.swing.JButton btnEnviar;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JLabel lblMensaje;
    private javax.swing.JLabel lblNumero;
    private javax.swing.JLabel lblPuerto;
    private javax.swing.JTextArea txtMensaje;
    private javax.swing.JTextField txtNumero;
    private javax.swing.JTextField txtPuerto;

    public void mensaje(String mensaje) {
        JOptionPane.showMessageDialog(new Frame(), mensaje, "", 2);
    }
}
```

Durante el proceso de ejecución, primero se debe llenar los campos correspondientes al envío de mensajes.

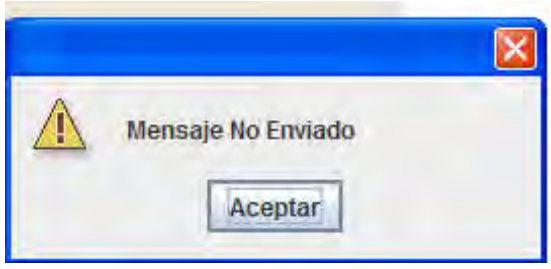


Se escribe el puerto, en este caso es el COMM3.

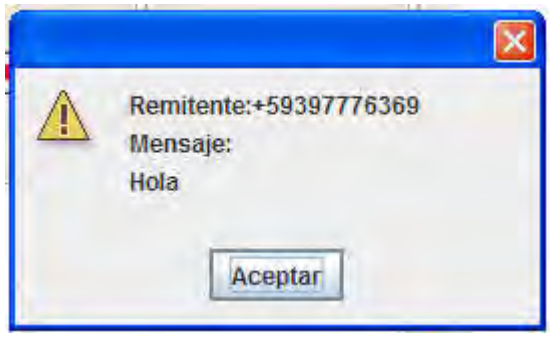
Se escribe el mensaje que se desea enviar y el número de destino y se presiona enviar. A continuación aparecerá un mensaje confirmando que se envió correctamente.



En el caso que ocurra un error al enviar el mensaje (celular sin saldo o número incorrecto) debe aparecer el siguiente mensaje:



Para la recepción de mensajes, aparecerá un cuadro de diálogo con el remitente, el mensaje.



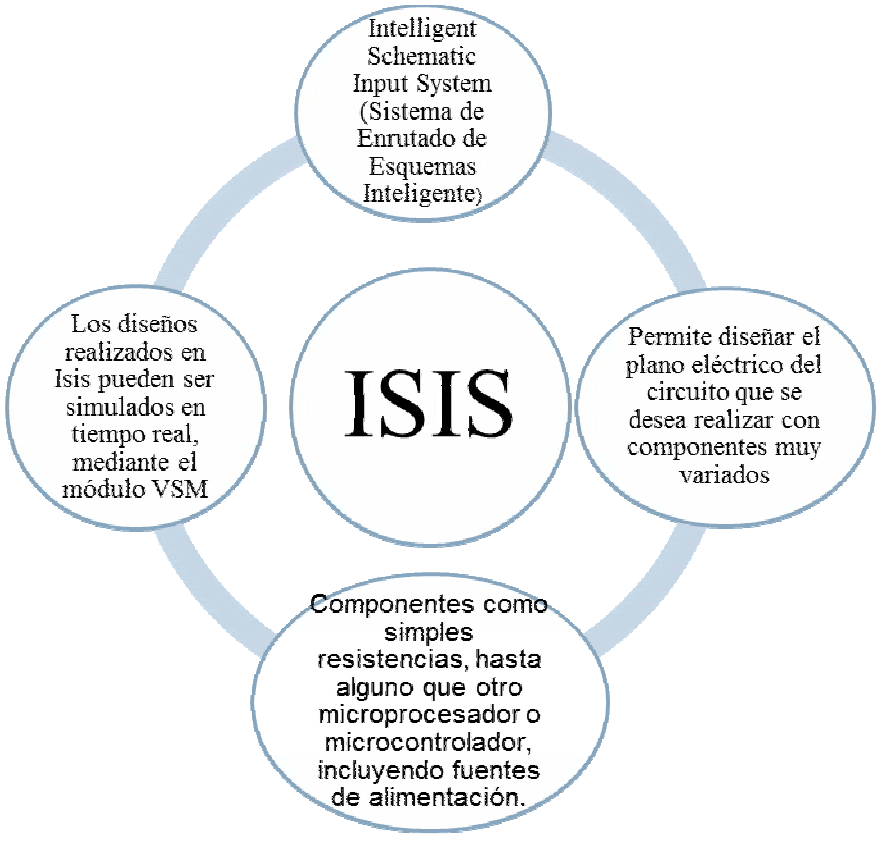
CAPITULO 27

SIMULACION DE PUERTOS CON PROTEUS

Hasta el momento para realizar pruebas con los puertos, es necesario diponer de todo el hadware y que esté correctamente conectado, pero existe una alternativa, para no tener que trabajar físicamente con el hadware, y es la simulación, claro está, que no se debe considerar estrictamente los resultados de la simulación, pero da un buena idea de lo que sería en la realidad.

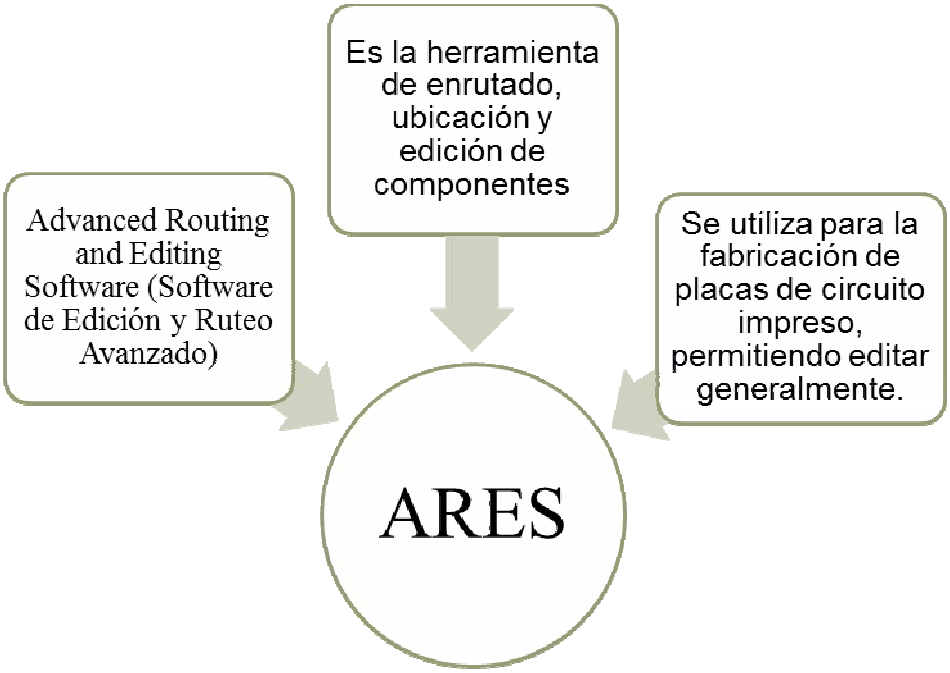
PROTEUS

Proteus es una compilación de programas de diseño y simulación electrónica, desarrollado por Labcenter Electronics que consta de los dos programas principales: Ares e Isis, y los módulos VSM y Electra.



Una de las prestaciones de Proteus, integrada con ISIS, es VSM, el Virtual System Modeling (Sistema Virtual de Modelado), una extensión integrada con ISIS, con la cual se puede simular, en tiempo real, con posibilidad de más rapidez. Se pueden simular circuitos con microcontroladores conectados a distintos dispositivos, como motores, lcd´s, teclados en matriz, etc. Incluye, entre otras, las familias de PIC's PIC10, PIC12, PIC16, PIC18, PIC24 y dsPIC33. ISIS es el corazón del entorno integrado PROTEUS. Combina

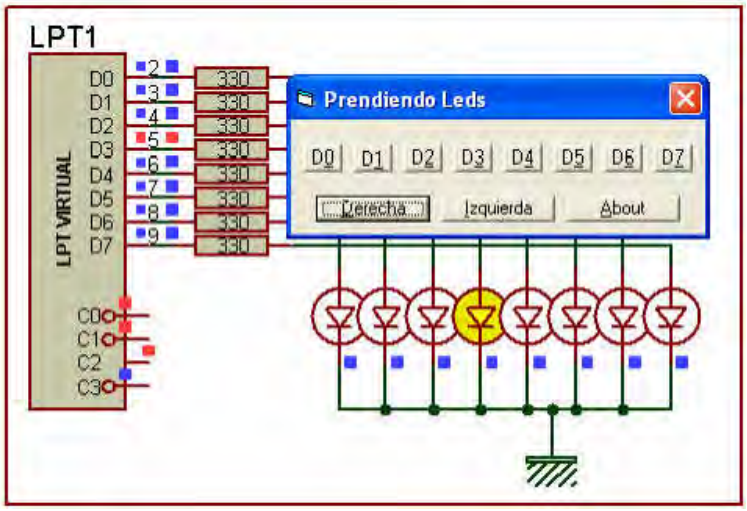
un entorno de diseño de una potencia excepcional con una enorme capacidad de controlar la apariencia final de los dibujos.



PUERTO VIRTUAL

En el caso de aplicaciones especiales, es posible que la PC con el *Drivervirtual* instalado trabaje también como servidor, la conexión puede ser inicializada del dispositivo externo por enviar datos entre los puertos. El convertidor abre la conexión a PC y envía los datos a *COM* virtual. Toda la situación se parece a la situación de puerto serie normal.

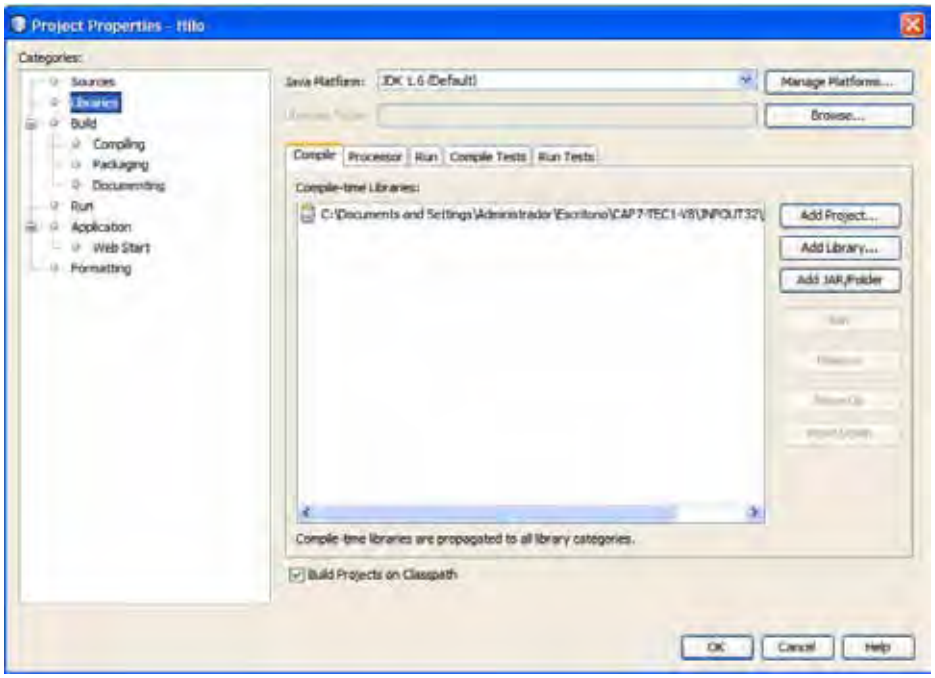
El puerto serie virtual *HW* no se puede usar para conectar dos puertos físicos de dos PC, ya que siempre trabaja con los puertos series virtuales.



SIMULACION DEL PUERTO PARALELO

Para la creación de la aplicación son necesarias ciertas librerías básicas tanto para el control del puerto paralelo como para el control de Proteus.

La librería para la aplicación creada en Netbeans es jnpout32pkg.jar la cual es agregada accediendo a las propiedades del proyecto creado.



Aquí se redirecciona la librería agregada como ya se ha venido trabajando en otras aplicaciones.

A su vez se debe copiar el archivo jnpout32pkg.dll dentro de la carpeta Windows:

jnpout32pkg.dll → WINDOWS\system32

Una vez realizado esto es indispensable importar la librería con la siguiente sentencia:

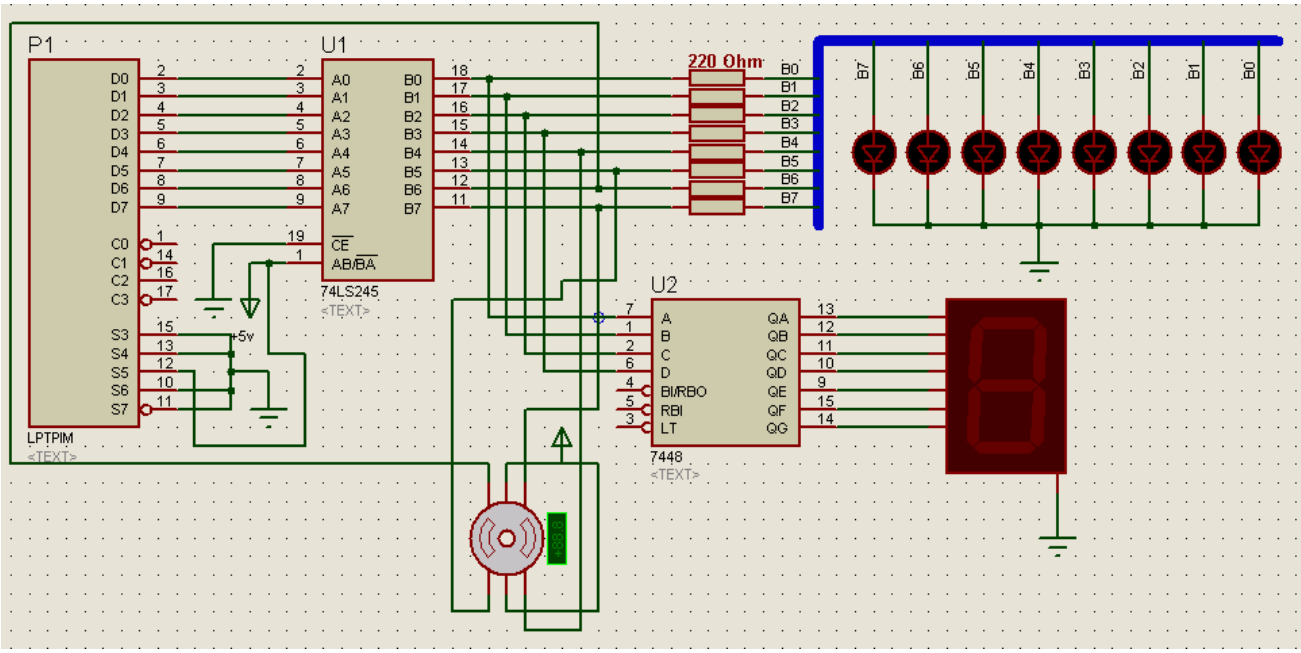
```
import jnpout32.*;
```

A su vez crear un objeto puerto de la clase pPort el cual permitirá acceder a todos los métodos de esta librería:

- puerto.input(port);
- puerto.output(port);

Dos métodos importantes para la entrada y salida de datos por el puerto LPT los cuales necesitan como parámetro la dirección del puerto a ser utilizado, debido a que se tiene 3 registros; control, estado y datos.

La aplicación controla un contador ascendente y descendente, en el cual cuando la cuenta está ascendiendo un motor gira en forma antihoraria, mientras que cuando la cuenta se encuentra descendiendo el motor comienza a girar en sentido horario.

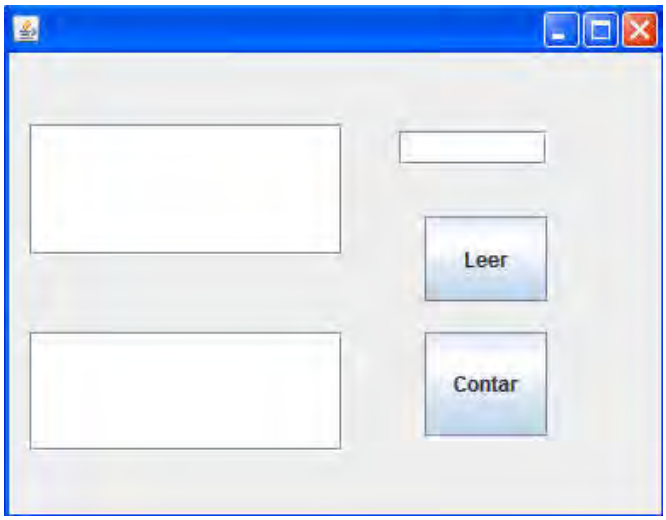


Para poder simular el puerto LPT en Proteus se hace uso de librerías y modelos para poder incorporar este dispositivo en la simulación, las librerías y modelos usados son los siguientes:

- Librerias para el sistema:
  - INPOUT32.DLL ⇒ WINDOWS\system32
  - hwinterface.ocx ⇒ WINDOWS\system32
- Librería y modelo para Proteus:
  - LPT02.LIB
  - Port.dll

Las cuales se colocan en las carpetas LIBRARY y MODELS respectivamente.

El ingreso de datos en la aplicación se realiza mediante un teclado matricial, es decir fue desarrolla por hardware enviando datos al puerto LPT por medio del cable correspondiente que permitirá la comunicación entre el ordenador y este. Esto debido a que a pesar de estar presentes las líneas de control en el modelo del puerto en PROTEUS, estas no funcionan correctamente.



La aplicación viene a ser una interfaz que simplemente va a manipular los datos ingresados por el puerto paralelo simulado en Proteus, y posteriormente transferirlos al puerto LPT para ser visualizados en los displays y funcionamiento del motor.

La ejecución de la aplicación en Java, es exactamente igual a cualquier otra, se la ejecuta sin ninguna consideración en particular.

CODIGO FUENTE:

Clase Hilo:

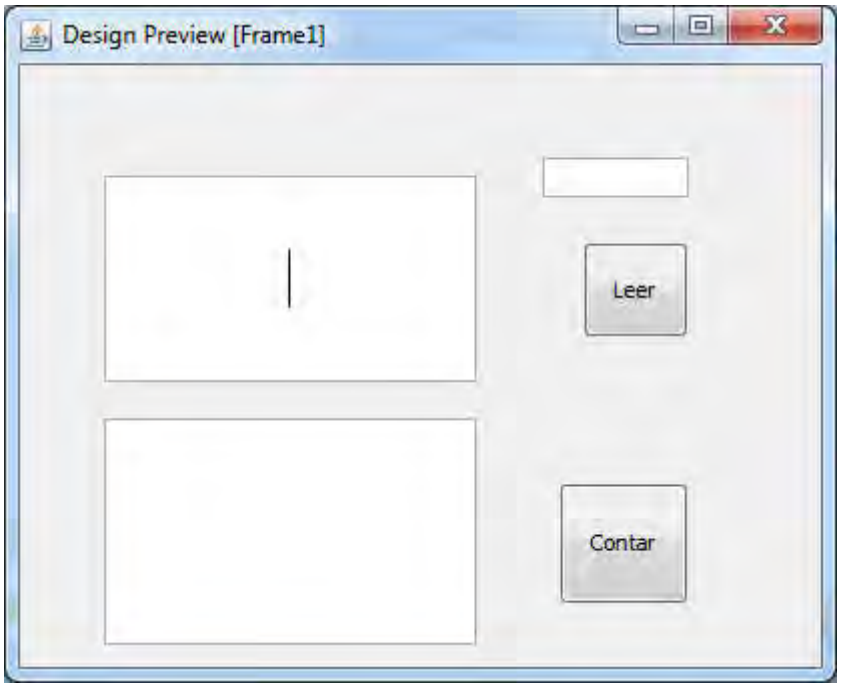
```
import java.util.logging.Level;
import java.util.logging.Logger;
import tec1c.hilo.vista.Frame1;
import jnpout32.*;

public class Contar extends Thread {

    Frame1 ventana;
    int fin;
    int bandera;
    pPort puerto = new pPort();
    private int psalida = 0x378;
    private int pentrada = 0x379;
    public int dato;
     private int psalida2 = 0x37A;

    public Contar(Frame1 ventana, int fin, int bandera) {
        this.ventana = ventana;
        this.fin = fin;
        this.bandera = bandera;
    }

    public Contar() {
```

```
}

 public void leer() {
    dato = puerto.input((short) pentrada);
    if(dato==15)
        dato=1;
    if(dato==31)
        dato=2;
    if(dato==47)
        dato=3;
    if(dato==79)
        dato=4;
    if(dato==95)
        dato=5;
    if(dato==111)
        dato=6;
    if(dato==143)
        dato=7;
    if(dato==159)
        dato=8;
    if(dato==175)
        dato=9;
    if(dato==223)
        dato=0;
    if(dato==63)
        dato=10;
    if(dato==127)
        dato=11;
    if(dato==191)
        dato=12;
    if(dato==255)
        dato=13;
    if(dato==207)
        dato=14;
    if(dato==239)
        dato=15;
}

public int getDato() {
    return dato;
}

public void setDato(int dato) {
    this.dato = dato;
}

public void run() {
    int a=16;
    int b=128;
```

```
int contador = 0;
int contador2 =Integer.parseInt(ventana.getTxtFinal().getText());
int c =contador2;
int bandera2=1;
while (true) {
    try {
        this.sleep(1000);
    } catch (InterruptedException ex) {
        Logger.getLogger(Contar.class.getName()).log(Level.SEVERE, null, ex);
    }
    if (bandera == 1) {

        ventana.getTxtMostrar().setText(String.valueOf(contador));
        puerto.output((short) psalida, (short) (contador));

      if(contador<c)
     puerto.output((short) psalida, (short) (a+contador));
        a=a*2;
        if (a>128)
            a=16;

    } else {
        ventana.getTxtMostrar2().setText(String.valueOf(contador2));
        puerto.output((short) psalida, (short) contador2);
        if(contador2>0)
            puerto.output((short) psalida, (short) (b+contador2));
        b=b/2;
        if(b<16)
        b=128;

        bandera2=2;
    }

    if (fin == contador) {

        Contar contar = new Contar(ventana, fin, 2);
        if(bandera==1)
        puerto.output((short) psalida, (short) (c));
        contar.start();

        fin(bandera, contador, contar);

        this.stop();

    }
    contador++;
    contador2--;
}
```

```
    }

    public void fin(int bandera, int contador, Contar contar){
        if(fin == contador & bandera == 2){
            contar.stop();
        }

    }
}
```

Clase Frame1:




```
import javax.swing.JTextField;

public class Frame1 extends javax.swing.JFrame {

    /** Creates new form Frame1 */
    public Frame1() {
        initComponents();
    }

    public JTextField getTxtFinal() {
        return txtFinal;
    }

    public void setTxtFinal(JTextField txtFinal) {
        this.txtFinal = txtFinal;
    }
```

```
    public JTextField getTxtMostrar() {
        return txtMostrar;
    }

    public void setTxtMostrar(JTextField txtMostrar) {
        this.txtMostrar = txtMostrar;
    }

    public JTextField getTxtMostrar2() {
        return txtMostrar2;
    }

    public void setTxtMostrar2(JTextField txtMostrar2) {
        this.txtMostrar2 = txtMostrar2;
    }
    private void btnContarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        Contar contar = new Contar(this, Integer.parseInt(txtFinal.getText()),1);
        contar.start();
    }

    private void jbnLeerActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here
        Contar contar =new Contar();
        contar.leer();
        txtFinal.setText(String.valueOf(contar.getDato()));
    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Frame1().setVisible(true);
            }
        });
    }

    private javax.swing.JButton btnContar;
    private javax.swing.JButton jbnLeer;
    private javax.swing.JTextField txtFinal;
    private javax.swing.JTextField txtMostrar;
    private javax.swing.JTextField txtMostrar2;
}
```

SIMULACIÓN DEL PUERTO SERIAL

Para esta aplicación se realiza una conexión a través del puerto virtual serie de Proteus para simular el comportamiento de un termómetro.

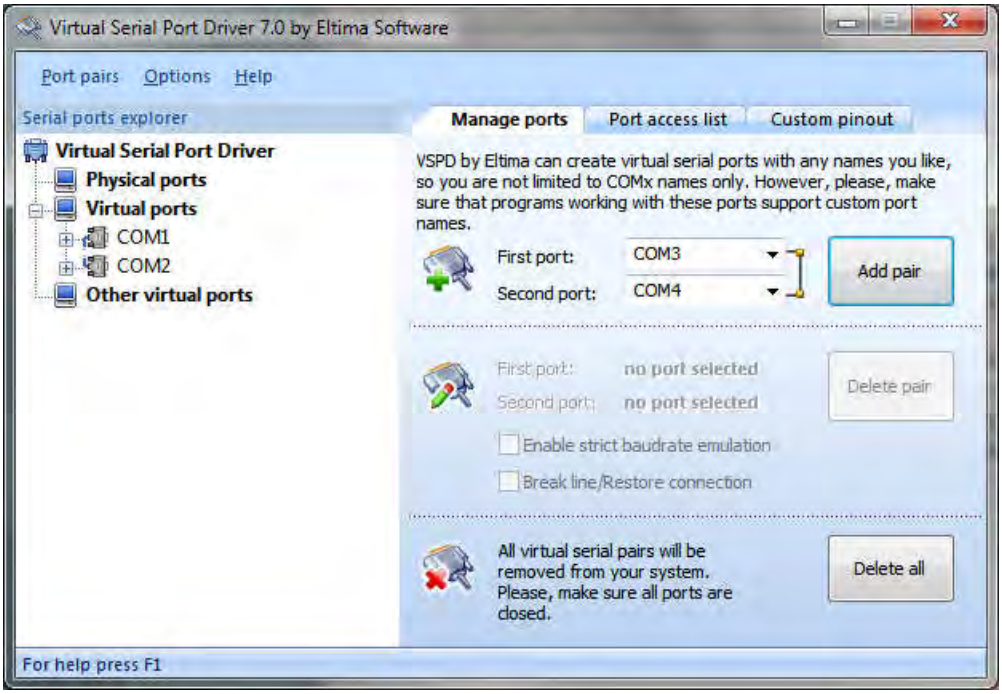
1. Para obtener la interfaz, primero se debe crear el puerto virtual que se lo hace con la instalación del programa Vsp2.2.1:

   Para su instalación se ejecuta el archivo de instalación de "Vsp2.2.1".

   Paso1: El Setup Wizard será visualizado. Para continuar la instalación haga click en el botón "Next".
   Paso2: La información básica sobre el producto será visualizada.
   Paso3: Elige el sitio para instalar el driver.
   Paso4: Escoge el nombre de la carpeta que será creada en Start menú.
   Paso5: Decide si quiere crear o no el método abreviado en el escritorio.
   Paso6: Los datos eligidos serán visualizados, recomendamos controlarlos.
   Paso7: La instalación del programa será procedido.

El programa se ejecutará, después de acabar la instalación. No es necesario resetear la computadora después de la instalación. HW VSP se ejecuta por hacer el click en el icono „ **VSP** “ – icono con la flecha roja.

Creación de un puerto Par de puertos virtuales.



En Netbeans se importa una librería para realizar la comunicación, para este caso la librería: ***giovynet***, esta permite realizar comunicaciones seriales utilizando java.

En el programa se implementa cuatro métodos:

Abrir puertos: con este método se habilita los puertos y además se verifica cuales de los puertos serie están libres para que se pueda entablar la comunicación.

```
public void abrirpuerto(List<String> portsFree) throws Exception {
```

```
        parameters = new Parameters();
        parameters.setPort(portsFree.get(0));
        parameters.setBaudRate(Baud._9600);
        com = new Com(parameters);
    }
```

Cerrar puerto: Una vez utilizados los puertos que sean necesarios si se los abre, se los debe volver a cerrar para no crear conflictos de operación en el futuro con estos puertos.

```
    public void cerrarpuerto(List<String> portsFree) throws Exception {
        com.close();
    }
```

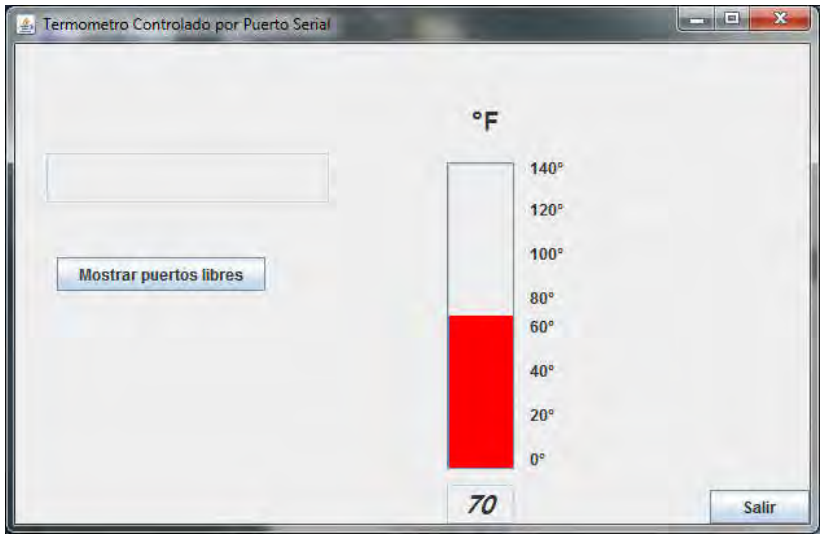
Envio datos: este método permite enviar datos desde el puerto serial hacia el otro extremo de la comunicación.

```
public void envioDato(List<String> portsFree) throws Exception {
com.sendSingleData(data);
}
```

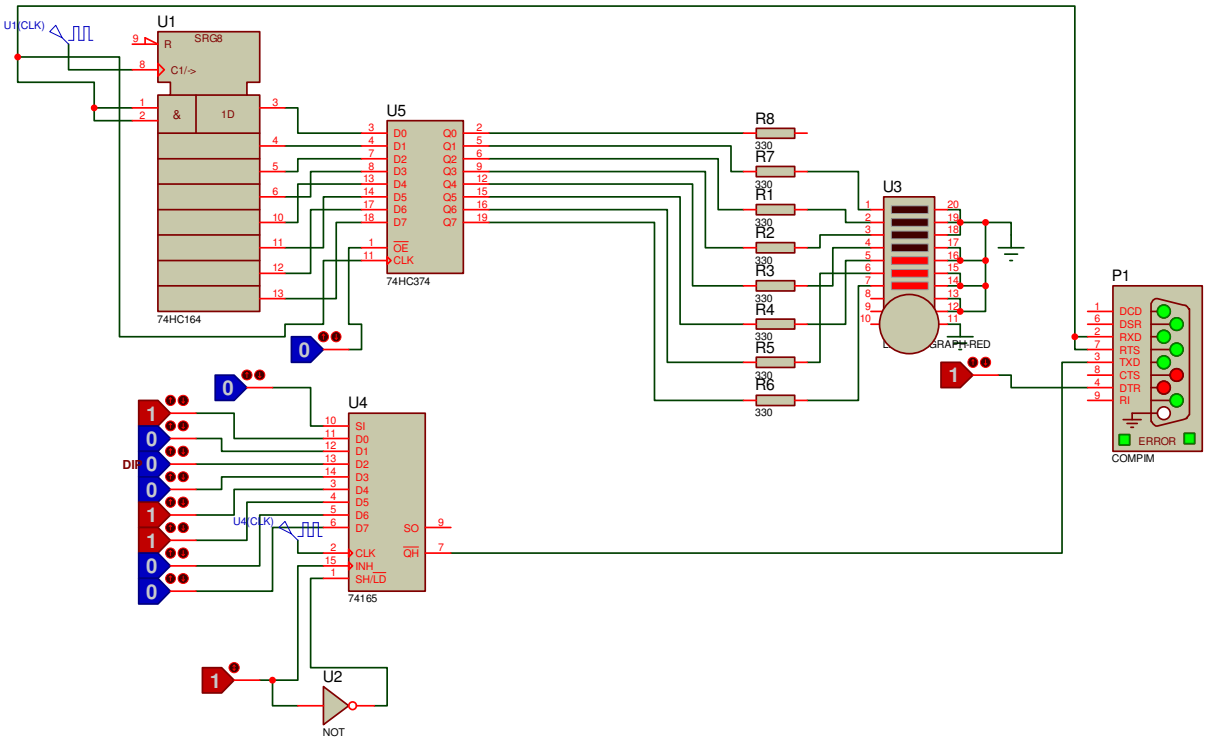
Recibir datos: se la implementa para que el puerto recepte la información que le fue enviada.

```
public int reciboDato(List<String> portsFree) throws Exception {
 int num = 0;
 do {
                num = 0;
                num = com.receiveSingleDataInt();
} while (num == 0);
return num;
 }
```

Una vez realizada la programación necesaria en la aplicación se procede a realizar el circuito en el software para simulación Proteus.



Y por medio del Proteus se manipulala temperatura a través de sus contactores de pulso.



CODIGO FUENTE:

Clase Datos:

```
import giovynet.serial.Baud;
import giovynet.serial.Com;
import giovynet.serial.Parameters;
import java.util.List;
```

```
public class Datos {

    int data;
    Parameters parameters;
    Com com;

    public Datos(int data, Parameters parameters, Com com) {
        this.data = data;
        this.parameters = parameters;
        this.com = com;
    }

    public Datos() {
        String data = null;
    }

    public int getData() {
        return data;
    }

    public void setData(int data) {
        this.data = data;
    }

    public void abrirpuerto(List<String> portsFree) throws Exception {
        parameters = new Parameters();
        parameters.setPort(portsFree.get(0));
        parameters.setBaudRate(Baud._9600);
        com = new Com(parameters);
    }

    public void cerrarpuerto(List<String> portsFree) throws Exception {
        com.close();
    }

    public void envioDato(List<String> portsFree) throws Exception {
        com.sendSingleData(data);
    }

    public int reciboDato(List<String> portsFree) throws Exception {
        int num = 0;
        do {
            num = 0;
            num = com.receiveSingleDataInt();
             System.out.println(num);
        } while (num == 0);

        return num;
```

```
    }
}
```

Clase Termómetro

```
import java.util.List;


public class Termometro extends javax.swing.JFrame {

    String libre;
    List<String> puertoLibre;
    int aux,contador=77;

    public Termometro() {
        initComponents();
    }
    private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {

        System.exit(0);
    }

    public void puertolibre(String libre, List<String> portsFree) {
        this.libre = libre;
        puertoLibre = portsFree;
    }

    public void manejotemperatura(int valor){
        contador=valor;
        tempvar.setValue(contador);
        txtvalor.setText(String.valueOf(contador));
    }

    private void btnpuertoActionPerformed(java.awt.event.ActionEvent evt) {
        txtpuerto.setText(libre);
    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {

            public void run() {
                new Termometro().setVisible(true);
            }
        });
    }
    private javax.swing.JToggleButton btnSalir;
    private javax.swing.JButton btnpuerto;
    private javax.swing.JLabel jLabel1;
```

```
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JProgressBar tempvar;
    private javax.swing.JTextField txtpuerto;
    private javax.swing.JTextField txtvalor;
}
```

CAPITULO 28

CONEXIÓN ENTRE MATLAB Y JAVA

Hasta este punto, ya se ha mostrado rápidamente las posibilidades que Java presta, cuando se trabaja en Ingeniería, pero a pesar de todo, Java es un lenguaje que no incorpora ciertas funciones especificas, que otras herramientas como Matlab si las incorpora, pero existe la posibilidad de unir estas dos herramientas, para obtener aplicaciones, mucho más versátiles.

MATLAB

MATLAB es un software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio (lenguaje M). Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. Dispone de dos herramientas adicionales que expanden sus prestaciones, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolboxes); y las de Simulink con los paquetes de bloques (blocksets).

JMATLINK

JMatLink conecta Java con Matlab, empleando métodos nativos. Para su utilización basta con importar al proyecto las tres clases principales que proporcionan el funcionamiento de Matlab mediante Java: CoreJMatLink, JMatLink y JMatLinkException. Adicionalmente se debe crear un paquete llamado jmatlink, donde se aloja las clases anteriores.

A continuación se presenta una tabla resumida de los métodos disponibles dentro de la clase JMatlink.

void	**engClose** () Cierra la conexión con Matlab.
void	**engCloseAll** () Cierra todas las conexiones con matlab
void	**engEvalString** (java.lang.String evalS) Evalua una expresión en el espacio de trabajo de Matlab.
double[][]	**engGetArray** (java.lang.String arrayS) Obtiene una matriz de espacio de trabajo de Matlab.
java.lang.String[]	**engGetCharArray** (java.lang.String arrayS) Obtiene un 'char' array (cadena) de espacio de trabajo de Matlab.

java.awt.Image	**engGetFigure** (int figure, int dx, int dy) Retorna la imagen de la figura de Matlab
java.awt.Image	**engGetFigure** (long epl, int figure, int dx, int dy) Retorna la imagen de la figura de Matlab
java.lang.String	**engGetOutputBuffer** () Retorna la salida de los comandos anteriores a partir de una instancia especificada de forma matlab.
java.lang.String	**engGetOutputBuffer** (long epl) Retorna la salida de los comandos anteriores en el espacio de trabajo de Matlab.
double	**engGetScalar** (java.lang.String arrayS) Obtiene un valor escalar de espacio de trabajo de Matlab.
boolean	**engGetVisible** (long epl) Retorna el estado de visibilidad de la ventana de Matlab
void	**engOpen** () Abre el motor.
int	**engOutputBuffer** () Retorna la salida de los comandos anteriores al espacio de trabajo de Matlab.
void	**engPutArray** (long epl, java.lang.String arrayS, double valueD) Pone una matriz en una área de trabajo especificada
void	**engPutArray** (long epl, java.lang.String arrayS, double[] valuesD) Pone un array (1 dimensiones) en una instancia
void	**engPutArray** (long epl, java.lang.String arrayS, double[][] valuesDD) Pone un array (2 dimensiones) en una instancia
void	**engPutArray** (java.lang.String arrayS, double valueD) Pone una matriz en espacio de trabajo de Matlab.
void	**engSetVisible** (long epl, boolean visB) Establece la visibilidad de la ventana de Matlab
java.lang.String	**getVersion** () Devuelve la versión actual de JMatLink
void	**setDebug** (boolean debugB) Activa o desactiva la información de depuración la salida estándar.

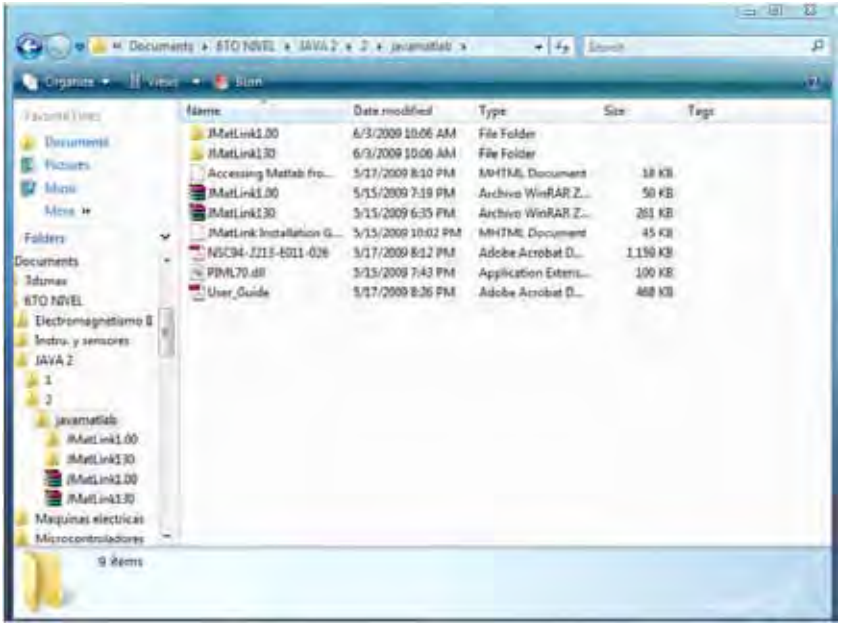
CONEXION MATLAB-JAVA

Para empezar con la conexion de Java y Matlab se debe tomar en cuenta los siguientes aspectos:

- Tener instalado una versión de Matlab que sobrepase o sea igual a la versión MATLAB 7.0.
- Tomar en cuenta que NeatBeans se encuentre instalado en la PC.

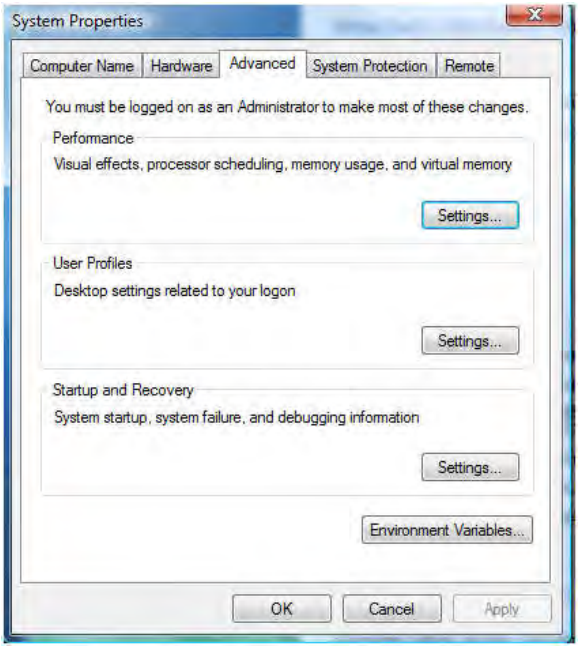
- Descargar la versión de JMATLINK1.30 que será la base fundamental para esta conexión.

Para empezar básicamente el proceso de conexión se debe descomprimir la carpeta JMATLINK1.30.

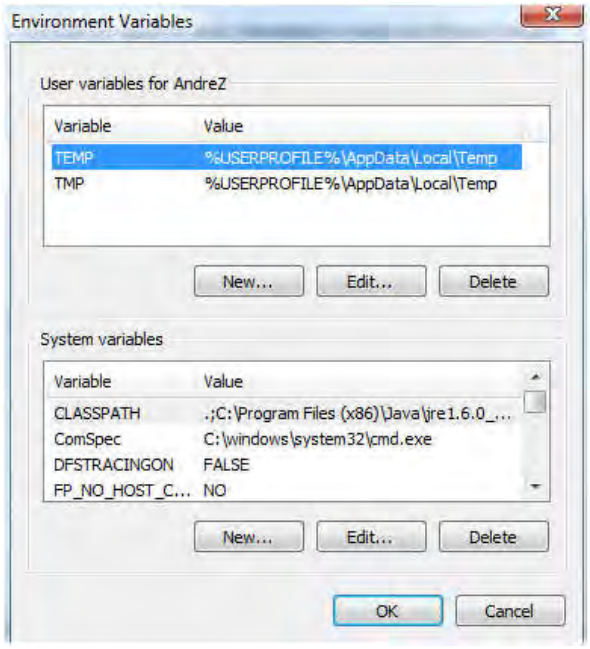


Se debe crear una variable de ambiente como se muestra en las figuras para lo cual se debe:

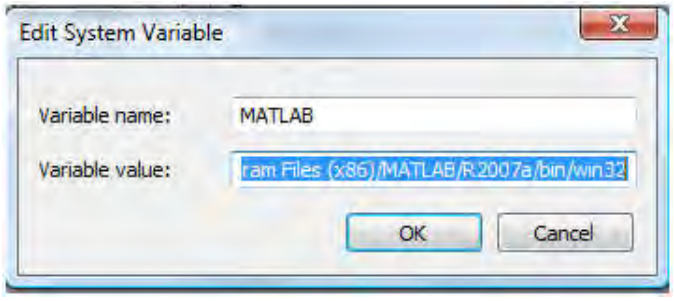
- Ingresar en propiedades de Mi PC.
- Buscar y entrar en propiedades del sistema (Si se está trabajando en Windows Vista se debe dar los debidos permisos para ingresar).



- Se escoge la opción de “Environment Variable”.
- Ingresar el nombre de la variable MATLAB y el valor de la variable será la ubicación del archivo “win32”, que se encuentra en la carpeta “BIN” dentro de MATLAB que se encuentra obviamente ya instalado en el computador a ejecutar la tarea.

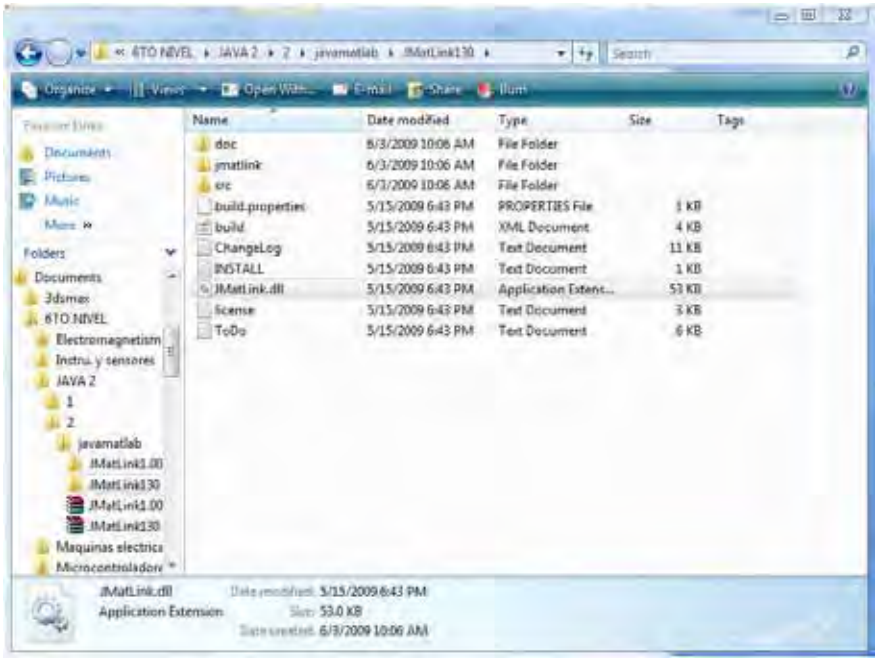


- Aceptar y Aplicar al final de la ventana para obtener nuestra variable creada.

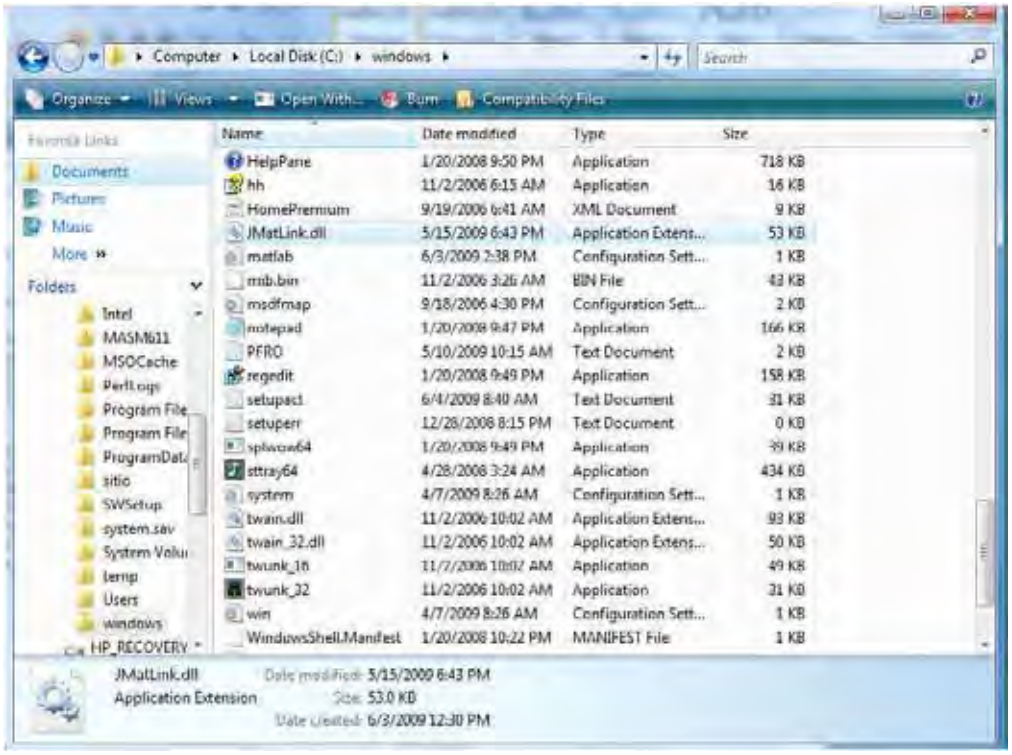


Para este paso se debe tomar en cuenta la carpeta JMatLink1.30:

- Utilizando la carpeta previamente mencionada se utiliza el archivo nombrado: JMatLink.dll



- Se localiza la carpeta llamada Windows la cual se encontrará en algún disco local dependiendo del PC (en el caso de tener más sistemas operativos tipo Microsoft se deberá buscar dicha carpeta con la característica de que sea el sistema operativo en el cual se va a utilizar la conexión y no sea obsoleto).
- Una vez ubicado en la carpeta Windows se procede a pegar JMatLink.dll en esa carpeta.

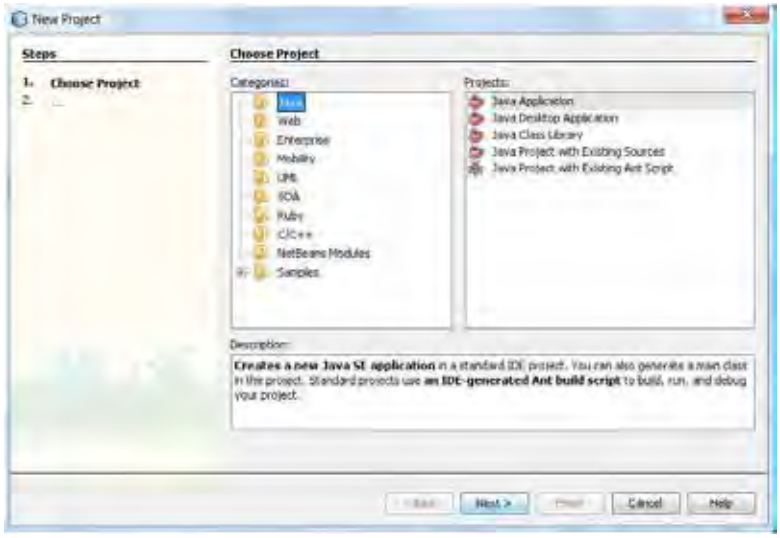


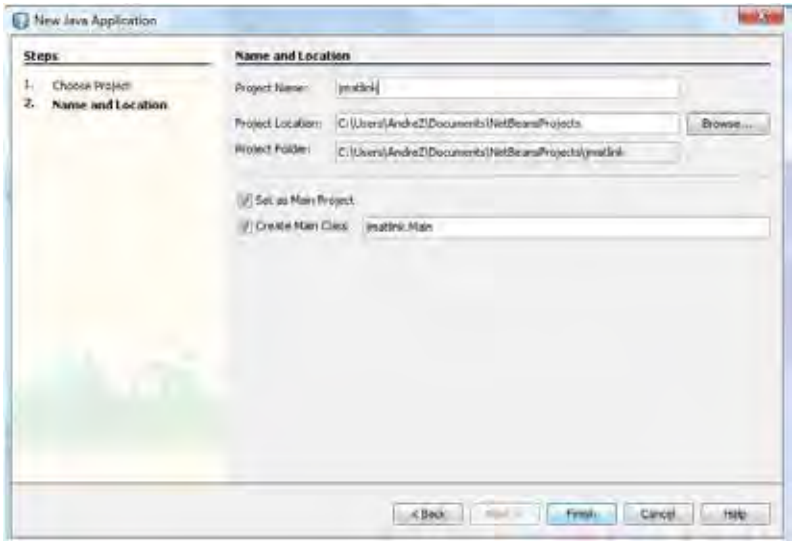
CREACION DE UN NUEVO PROYECTO:

A continuación se procede de manera explicativa a la creación de un nuevo Proyecto de Java en NeatBeans utilizando la conexión JMatLink como medio entre Matlab y Java.

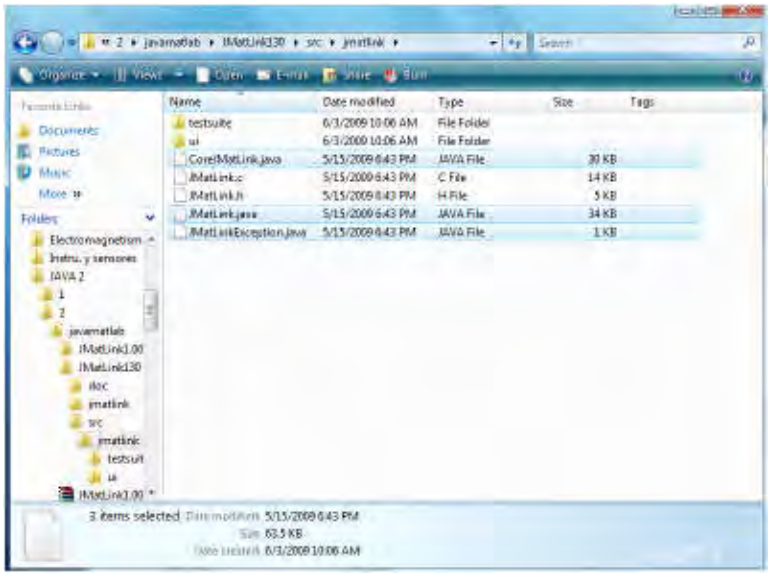
- Se ejecuta NeatBeans (para el caso de Windws Vista y 7, se debe ejecutar como administrador, haciendo click con el botón derecho y abrir como Administrador).

- Crear un nuevo proyecto y dentro de éste un paquete llamado jmatlink, para llamar a las clases que vienen preestablecidas en la carpeta que se suministró.

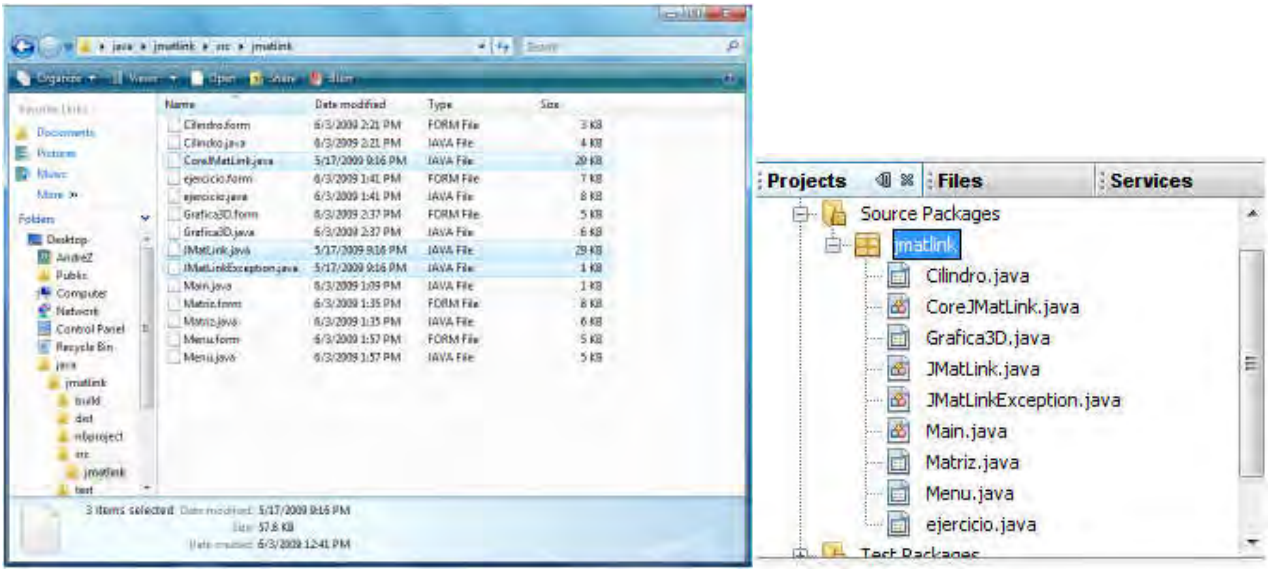




- Utilizando de nuevo la carpeta JMatLink1.30 proporcionada debemos ingresar al src y a “jmatlink” de donde se toma las 3 clases antes mencionadas:



- En la carpeta del proyecto ya creado se debe copiar 3 clases que vienen preestablecidas, se deben pegar en el paquete “jmtalink” que se encuentra en el “src” del proyecto jmatlink creado.



CODIGO FUENTE:

Clase Principal

```
import java.awt.Image;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.PixelGrabber;
import javax.swing.JPanel;
import javax.swing.*;
import java.math.*;

public class Principal extends javax.swing.JFrame {

    public Principal() {
        initComponents();

        this.setVisible(true);
        this.setSize(500, 500);
        this.setTitle("GRUPO 3");
        jLabel13.setVisible(false);
        jLabel14.setVisible(false);
        jLabel15.setVisible(false);
        jButton9.setVisible(false);
        jLabel16.setVisible(false);
        jLabel17.setVisible(false);
        jButton13.setVisible(false);
        jLabel20.setVisible(false);
        jLabel21.setVisible(false);
        jLabel22.setVisible(false);
        jButton17.setVisible(false);
        jButton21.setVisible(false);
```

```
    }

private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    Funciones.setVisible(true);
    Funciones.setSize(500, 500);
    Funciones.setTitle("EJERCICIOS CON FUNCIONES");
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    Graficias.setVisible(true);
    Graficias.setSize(550, 550);
    Graficias.setTitle("EJERCICIOS CON GRAFICAS");
}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
     Matrices.setVisible(true);
    Matrices.setSize(500, 500);
    Matrices.setTitle("EJERCICIOS CON MATRICES");
}

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    Otros.setVisible(true);
    Otros.setSize(800, 700);
    Otros.setTitle("EJERCICIOS ADICIONALES");
}

private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
    JMatLink engine = new JMatLink();
        double a;
        String c;
        String d;
        c=txtx.getText();
        d=c+";";
        engine.engOpen();
        engine.engEvalString("a="+d+" f=sin(a);");
        a = engine.engGetScalar("f");
        engine.engClose();
        txtr.setText(Double.toString(a));
}

private void jButton7ActionPerformed(java.awt.event.ActionEvent evt) {
    JMatLink engine = new JMatLink();
        double a1;
        String c1;
        String d1;
```

```
        c1=txtx1.getText();
        d1=c1+";";
        engine.engOpen();
        engine.engEvalString("a="+d1+" f=cos(a);");
        a1 = engine.engGetScalar("f");
        engine.engClose();
        txtr1.setText(Double.toString(a1));
}

private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {
    JMatLink engine = new JMatLink();
        double a2;
        String c2;
        String d2;
        c2=txtx2.getText();
        d2=c2+";";
        engine.engOpen();
        engine.engEvalString("a="+d2+" f=a^2+2*a+1");
        a2 = engine.engGetScalar("f");
        engine.engClose();
        txtr2.setText(Double.toString(a2));
        jLabel13.setVisible(true);
        jLabel14.setVisible(true);
        jLabel15.setVisible(true);
        jButton9.setVisible(true);
}

private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {
    Funciones.setVisible(false);
    jLabel13.setVisible(false);
    jLabel14.setVisible(false);
    jLabel15.setVisible(false);
    jButton9.setVisible(false);
}

private void jButton10ActionPerformed(java.awt.event.ActionEvent evt) {
    JMatLink engine = new JMatLink();
        String c3;
        String d3;
        Image im;
        Graphics g=canvas1.getGraphics();
        c3=String.valueOf(Math.PI/2);
        d3=c3+";";
        engine.engOpen();
        engine.engEvalString("a="+d3+" f=sin(a); x=-3*pi:0.2:3*pi; plot(sin(x))");
        im = engine.engGetFigure(1, 400, 400);
        engine.engClose();
        g.drawImage(im,0,0,400,400,this);
}
```

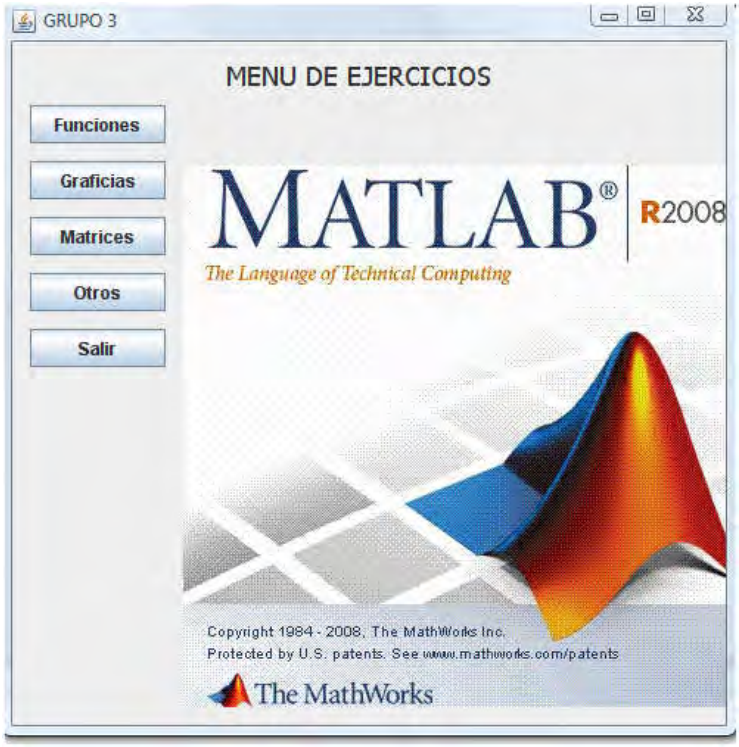
```
private void jButton11ActionPerformed(java.awt.event.ActionEvent evt) {
    JMatLink engine = new JMatLink();
        Image im;
        Graphics g=canvas1.getGraphics();
        engine.engOpen();

engine.engEvalString("r=(0:0.1:2*pi)';t=(0:0.1:2*pi);X=(3+cos(r))*cos(t);Y=(3+cos(r))*sin(t);
Z=sin(r*ones(size(t)));surf(X,Y,Z)");
        im = engine.engGetFigure(1, 400, 400);
        engine.engClose();
        g.drawImage(im,0,0,400,400,this);
}

private void jButton12ActionPerformed(java.awt.event.ActionEvent evt) {
    JMatLink engine = new JMatLink();
        Image im;
        Graphics g=canvas1.getGraphics();
        engine.engOpen();
        engine.engEvalString("theta=[0:0.1:2*pi];r=cos(3*theta);polar(theta,r,'--b')");
        im = engine.engGetFigure(1, 400, 400);
        engine.engClose();
        g.drawImage(im,0,0,400,400,this);
        jLabel16.setVisible(true);
        jLabel17.setVisible(true);
        jButton13.setVisible(true);
}

private void jButton13ActionPerformed(java.awt.event.ActionEvent evt) {
    Graficias.setVisible(false);
    jLabel16.setVisible(false);
    jLabel17.setVisible(false);
    jButton13.setVisible(false);
}

private void jButton14ActionPerformed(java.awt.event.ActionEvent evt) {
    double[][] array={{1.0 , 2.0 , 3.0}, {4.0 , 5.0 , 6.0}, {7.0 , 8.0 , 9.0}};
    double[][] array1=null;
   JMatLink engine = new JMatLink();

   engine.engOpen();

   engine.engPutArray("array", array);
   engine.engEvalString("a1=array'");
        array1 = engine.engGetArray("a1");
        engine.engClose();

        for(int i=0;i<3;i++){
            for(int k=0;k<3;k++){
```

```
                jTable1.setValueAt(array[i][k], i, k);
                jTable2.setValueAt(array1[i][k], i, k);
            }
        }
}

private void jButton15ActionPerformed(java.awt.event.ActionEvent evt) {
    double[][] array={{1.1 , 2.2 , 3.3}, {4.4 , 5.5 , 6.6}, {7.7 , 8.8 , 9.9}};
    double[][] array11=null;
   JMatLink engine = new JMatLink();

   engine.engOpen();

   engine.engPutArray("array", array);
   engine.engEvalString("unos=eye(3,3);a1=array.*unos");
        array11 = engine.engGetArray("a1");
        engine.engClose();

        for(int i=0;i<3;i++){
            for(int k=0;k<3;k++){
                jTable1.setValueAt(array[i][k], i, k);
                jTable2.setValueAt(array11[i][k], i, k);
            }
        }
}

private void jButton16ActionPerformed(java.awt.event.ActionEvent evt) {
    double[][] array={{0.1 , 0.2 , 0.3}, {0.4 , 0.5 , 0.6}, {0.7 , 0.8 , 0.9}};
    double[][] array11=null;
   JMatLink engine = new JMatLink();

   engine.engOpen();

   engine.engPutArray("array", array);
   engine.engEvalString("a1=inv(array)");
        array11 = engine.engGetArray("a1");
        engine.engClose();

        for(int i=0;i<3;i++){
            for(int k=0;k<3;k++){
                jTable1.setValueAt(array[i][k], i, k);
                jTable2.setValueAt(array11[i][k], i, k);
            }
        }
        jLabel20.setVisible(true);
        jLabel21.setVisible(true);
        jLabel22.setVisible(true);
        jButton17.setVisible(true);
}
```

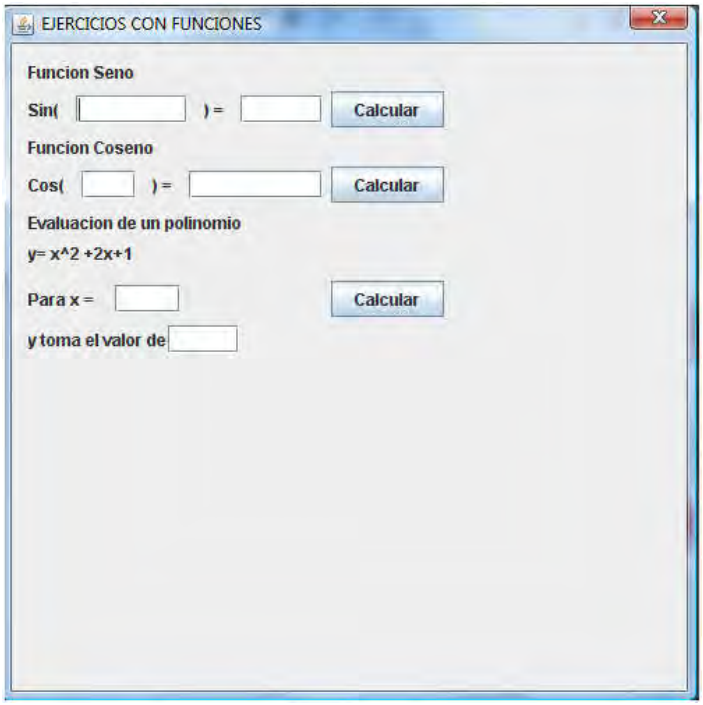
```
private void jButton17ActionPerformed(java.awt.event.ActionEvent evt) {
        Matrices.setVisible(false);
        jLabel20.setVisible(false);
        jLabel21.setVisible(false);
        jLabel22.setVisible(false);
        jButton17.setVisible(false);
}

private void jButton18ActionPerformed(java.awt.event.ActionEvent evt) {
    JMatLink engine = new JMatLink();
        engine.engOpen();
        engine.engEvalString("cancion");
        engine.engClose();
}

private void jButton19ActionPerformed(java.awt.event.ActionEvent evt) {
    JMatLink engine = new JMatLink();
        engine.engOpen();
        engine.engEvalString("util");
        //engine.engClose();
}

private void jButton20ActionPerformed(java.awt.event.ActionEvent evt) {
    JMatLink engine = new JMatLink();
        engine.engOpen();
        engine.engEvalString("conectar");
        engine.engClose();
        jButton21.setVisible(true);
}

private void jButton21ActionPerformed(java.awt.event.ActionEvent evt) {
    Otros.setVisible(false);
    jButton21.setVisible(false);
}

private void jButton22ActionPerformed(java.awt.event.ActionEvent evt) {
    JMatLink engine = new JMatLink();
     Image im2;
        Graphics g2=canvas2.getGraphics();
        engine.engOpen();
        engine.engEvalString("camara");
        im2 = engine.engGetFigure(1, 400, 400);
        engine.engClose();
        g2.drawImage(im2,0,0,400,400,this);
        engine.engClose();
}

    public static void main(String args[]) {
```

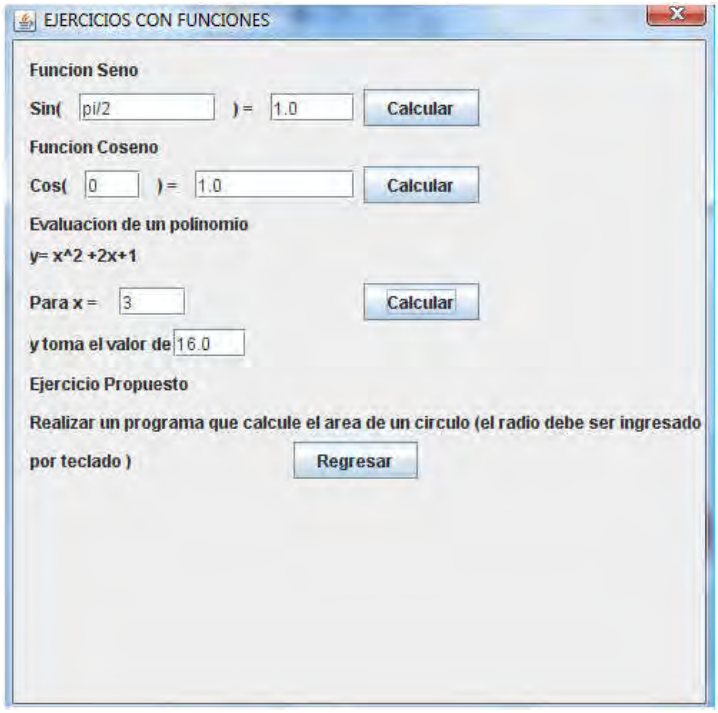
```
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Principal().setVisible(true);
        }
    });
}

private javax.swing.JDialog Funciones;
private javax.swing.JDialog Graficias;
private javax.swing.JDialog Matrices;
private javax.swing.JDialog Otros;
private java.awt.Canvas canvas1;
private java.awt.Canvas canvas2;
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton10;
private javax.swing.JButton jButton11;
private javax.swing.JButton jButton12;
private javax.swing.JButton jButton13;
private javax.swing.JButton jButton14;
private javax.swing.JButton jButton15;
private javax.swing.JButton jButton16;
private javax.swing.JButton jButton17;
private javax.swing.JButton jButton18;
private javax.swing.JButton jButton19;
private javax.swing.JButton jButton2;
private javax.swing.JButton jButton20;
private javax.swing.JButton jButton21;
private javax.swing.JButton jButton22;
private javax.swing.JButton jButton3;
private javax.swing.JButton jButton4;
private javax.swing.JButton jButton5;
private javax.swing.JButton jButton6;
private javax.swing.JButton jButton7;
private javax.swing.JButton jButton8;
private javax.swing.JButton jButton9;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel12;
private javax.swing.JLabel jLabel13;
private javax.swing.JLabel jLabel14;
private javax.swing.JLabel jLabel15;
private javax.swing.JLabel jLabel16;
private javax.swing.JLabel jLabel17;
private javax.swing.JLabel jLabel18;
private javax.swing.JLabel jLabel19;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel20;
private javax.swing.JLabel jLabel21;
```

```
    private javax.swing.JLabel jLabel22;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTable jTable1;
    private javax.swing.JTable jTable2;
    private javax.swing.JTextField txtr;
    private javax.swing.JTextField txtr1;
    private javax.swing.JTextField txtr2;
    private javax.swing.JTextField txtx;
    private javax.swing.JTextField txtx1;
    private javax.swing.JTextField txtx2;
}
```

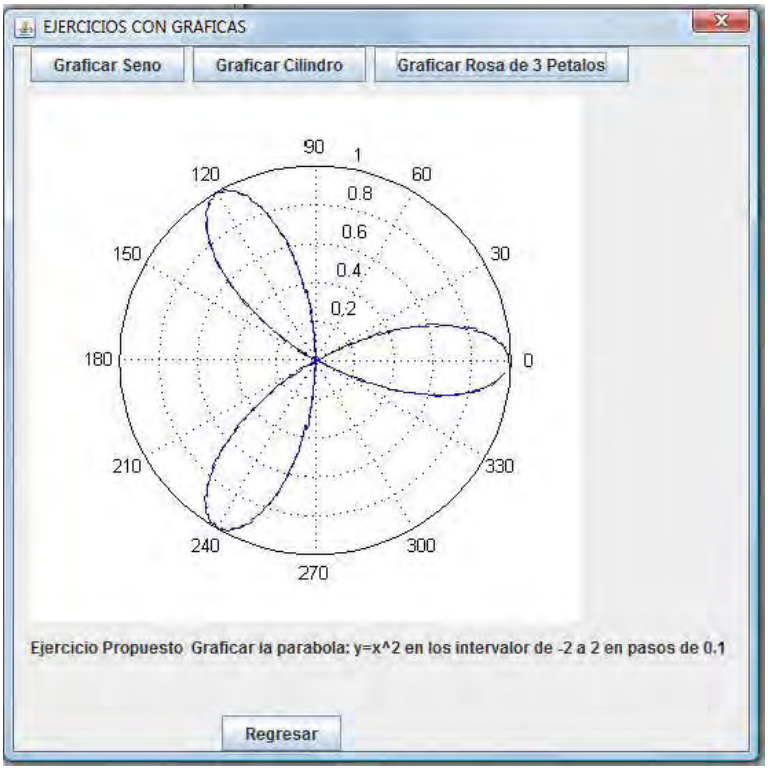


Cuando se presiona el botón Funciones se despliega el siguiente frame, con el que la aplicación permite que el usario ingrese datos de funciones trigonométricas y éstos sean evaluados en Matlab.

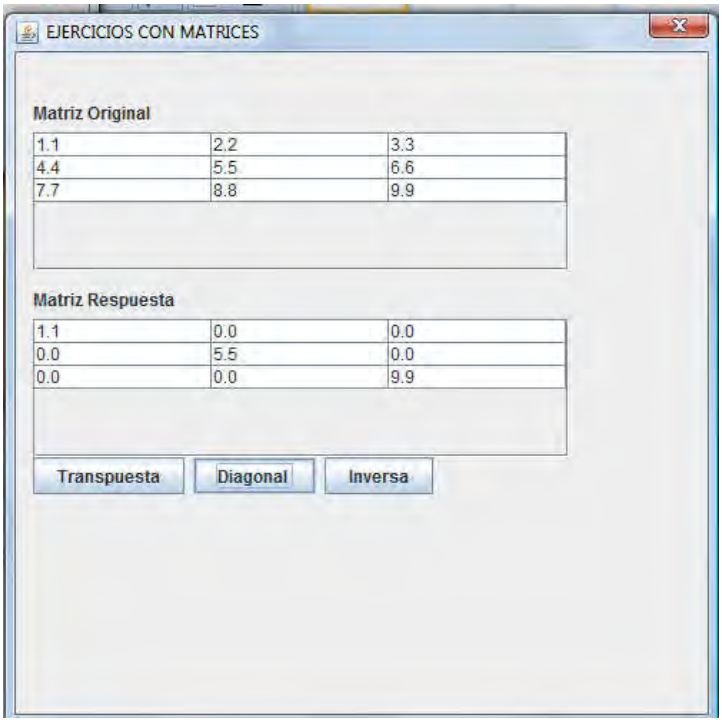




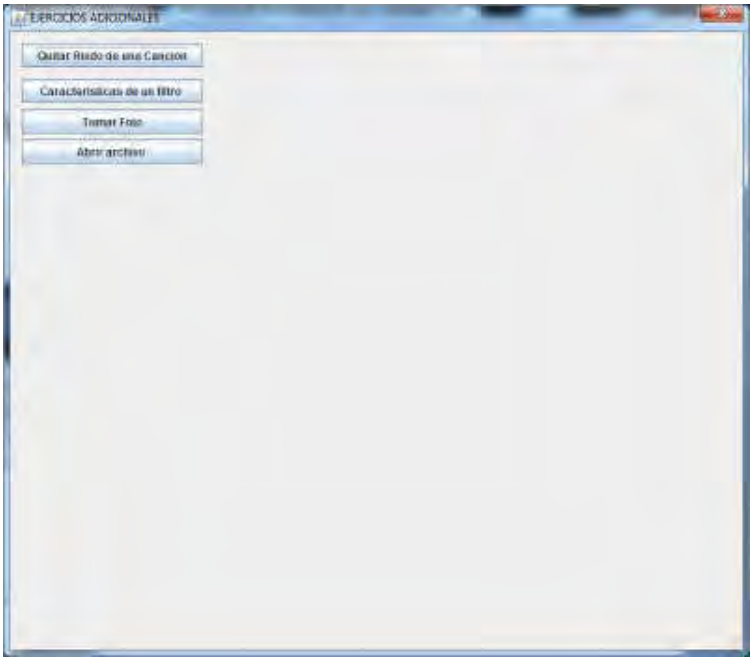
Cuando se presiona el botón Gráficos se despliega el siguiente frame, con el que la aplicación permite que el usario grafique funciones trigonométricas desde Matlab y que sean capturados en java, para evitar que se despliegue el frame propio de Matlab.



Cuando se presiona el botón Matrices se despliega el siguiente frame, con el que la aplicación permite que el usario ingrese matrices y permita operar con las mismas desde Matlab.



Cuando se presiona el botón Otros se despliega el siguiente frame, con el que la aplicación permite que al usario tomar fotos desde una cámara web y llame un filtro digital.



A continuación se presenta algunos de los métodos por separado más importantes

PARA EL CÁLCULO DEL SENO

```
JMatLink engine = new JMatLink();
    double a;
    String c;
    String d;
    c=txtx.getText();
    d=c+";";
    engine.engOpen();
    engine.engEvalString("a="+d+" f=sin(a);");
    a = engine.engGetScalar("f");
    engine.engClose();
    txtr.setText(Double.toString(a));
```

PARA EL CÁLCULO DEL COSENO

```
JMatLink engine = new JMatLink();
    double a1;
    String c1;
    String d1;
    c1=txtx1.getText();
    d1=c1+";";
    engine.engOpen();
    engine.engEvalString("a="+d1+" f=cos(a);");
    a1 = engine.engGetScalar("f");
    engine.engClose();
    txtr1.setText(Double.toString(a1));
```

PARA EL CÁLCULO DE LA ECUACIÓN CUADRÁTICA

```
JMatLink engine = new JMatLink();
    double a2;
    String c2;
    String d2;
    c2=txtx2.getText();
    d2=c2+";";
    engine.engOpen();
    engine.engEvalString("a="+d2+" f=a^2+2*a+1");
    a2 = engine.engGetScalar("f");
    engine.engClose();
    txtr2.setText(Double.toString(a2));
    jLabel13.setVisible(true);
    jLabel14.setVisible(true);
    jLabel15.setVisible(true);
    jButton9.setVisible(true);
```

PARA LA GRÁFICA DEL SENO

```
JMatLink engine = new JMatLink();

    String c3;
    String d3;
    Image im;
    Graphics g=canvas1.getGraphics();
    c3=String.valueOf(Math.PI/2);
    d3=c3+";";
    engine.engOpen();
    engine.engEvalString("a="+d3+" f=sin(a); x=-3*pi:0.2:3*pi; plot(sin(x))");
    im = engine.engGetFigure(1, 400, 400);
    engine.engClose();
    g.drawImage(im,0,0,400,400,this);
```

PARA LA GRÁFICA CILINDRO

```
JMatLink engine = new JMatLink();

    Image im;
    Graphics g=canvas1.getGraphics();

    engine.engOpen();
    engine.engEvalString("r=(0:0.1:2*pi)';t=(0:0.1:2*pi);X=(3+cos(r))*cos(t);Y=(3+cos(r))*sin(t);Z=sin(r*ones(si
    im = engine.engGetFigure(1, 400, 400);
    engine.engClose();
    g.drawImage(im,0,0,400,400,this);
```

PARA LA GRÁFICA ROSA DE 3 PÉTALOS

```
JMatLink engine = new JMatLink();
    Image im;
    Graphics g=canvas1.getGraphics();
    engine.engOpen();
    engine.engEvalString("theta=[0:0.1:2*pi];r=cos(3*theta);polar(theta,r,'--b')");
    im = engine.engGetFigure(1, 400, 400);
    engine.engClose();
    g.drawImage(im,0,0,400,400,this);
    jLabel16.setVisible(true);
    jLabel17.setVisible(true);
    jButton13.setVisible(true);
```

PARA EL CÁLCULO DE MATRIZ TRANSPUESTA

```
double[][] array={{1.0 , 2.0 , 3.0}, {4.0 , 5.0 , 6.0}, {7.0 , 8.0 , 9.0}};
      System.out.println(array[0][0]);
    double[][] array1=null;
    JMatLink engine = new JMatLink();
    engine.engOpen();
    engine.engEvalString("a=[1 2 3; 4 5 6; 7 8 9]; a1=a'");
    array1 = engine.engGetArray("a1");
    engine.engClose();

    for(int i=0;i<3;i++){
        for(int k=0;k<3;k++){
                jTable1.setValueAt(array[i][k], i, k);
                jTable2.setValueAt(array1[i][k], i, k);
        }
    }
```

PARA EL CÁLCULO DE LA DIAGONAL DE UNA MATRIZ

```
 double[][] array={{1.1 , 2.2 , 3.3}, {4.4 , 5.5 , 6.6}, {7.7 , 8.8 , 9.9}};
 double[][] array11=null;
JMatLink engine = new JMatLink();

engine.engOpen();

engine.engPutArray("array", array);
engine.engEvalString("unos=eye(3,3);a1=array.*unos");
     array11 = engine.engGetArray("a1");
     engine.engClose();

     for(int i=0;i<3;i++){
         for(int k=0;k<3;k++){
                 jTable1.setValueAt(array[i][k], i, k);
                 jTable2.setValueAt(array11[i][k], i, k);
         }
     }
```

PARA EL CÁLCULO DE LA MATRIZ INVERSA

```
 double[][] array={{0.1 , 0.2 , 0.3}, {0.4 , 0.5 , 0.6}, {0.7 , 0.8 , 0.9}};
 double[][] array11=null;
JMatLink engine = new JMatLink();

engine.engOpen();

engine.engPutArray("array", array);
engine.engEvalString("a1=inv(array)");
     array11 = engine.engGetArray("a1");
     engine.engClose();

     for(int i=0;i<3;i++){
         for(int k=0;k<3;k++){
                 jTable1.setValueAt(array[i][k], i, k);
                 jTable2.setValueAt(array11[i][k], i, k);
         }
     }
     jLabel20.setVisible(true);
     jLabel21.setVisible(true);
     jLabel22.setVisible(true);
     jButton17.setVisible(true);
```

PARA QUITAR RUIDO DE UNA CANCIÓN

```
    JMatLink engine = new JMatLink();
        engine.engOpen();
        engine.engEvalString("cancion");
        engine.engClose();
```

CÓDIGO EN MATLAB DE LA FUNCIÓN:

```
function [y]= cancion(x)
x=x+1;
```

```
clear all;
clc;
[senial,Fs]=wavread('SWEETHOME.WAV'); % Se lee el archivo y se guarda la frecuencia
 figure(1)
 plot(senial)% Se grafica la señal de voz
 figure(2)
 [frec,Wi]=freqz(senial,1,1000);
 specgram(senial)
 figure(3)
plot(Wi/pi,frec)
 load filtro4.mat
 Num2;
 y=filter(Num2,1,senial);
 specgram(y)
 wavplay(y,22150)
 y=0;
```

CARACTERÍSTICAS DE UN FILTRO

```
JMatLink engine = new JMatLink();
    engine.engOpen();
    engine.engEvalString("util");
    engine.engClose();
```

CÓDIGO EN MATLAB:

```
function [y]= util(x)
clear all;
clc;
%N=2*[1 -0.5 -0.5];
%M=[1 -1.13 2.26];
%zplane(N,M)
%fvtool(N,M)
N=1;
M=[1 0 -0.81];
fvtool(N,M)
```

ABRIR ARCHIVO

```
JMatLink engine = new JMatLink();
    engine.engOpen();
    engine.engEvalString("conectar");
    engine.engClose();
    jButton21.setVisible(true);
```

CÓDIGO EN MATLAB:

```
function [y]= conectar(x)
winopen('c:\gracias.pptx')
```

CAPITULO 29

MANEJO DEL PUETO SERIE Y PARALELO CON JAVA Y MATLAB

Detro de la ingeniería electrónica más de una vez, es necesario realizar algún proyecto que se enlace con el mundo exterior, y esto previamente ya se comentó, mediante el API de comunicaciones de Java, con lo que se podía trabajar con los puertos, pero esa no es la única forma con la que Java permite, el manejo de los puertos. A continuación se comenta cómo controlar los puertos serie y paralelo, pero esta vez con Matlab.

CONTROL DEL PUERTO PARALELO LPT

A continuación se presenta las instrucciones básicas para el control del puerto paralelo en Matlab, con línea de comandos.

En el command window se ejecuta:

```
>> out = daqhwinfo;
>> out.InstalledAdaptors
```

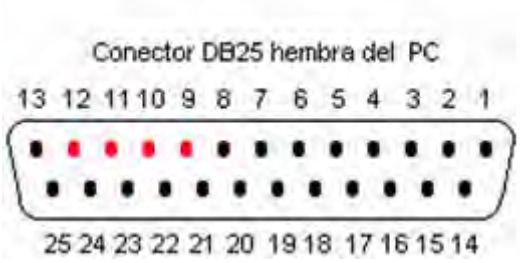
Lo que dará como resultado en el caso de mi PC:

```
ans =
'parallel'
'winsound'
```

Una vez que Matlab ha reconocido el Puerto Paralelo, se ejecuta:

```
>> daqhwinfo('parallel')
ans =
AdaptorDllName: [1x50 char]
AdaptorDllVersion: '2.7 (R14SP3)'
AdaptorName: 'parallel'
BoardNames: {'PC Parallel Port Hardware'}
InstalledBoardIds: {'LPT1'}
ObjectConstructorName:{'"digitalio('parallel','LPT1')'}
```

La última línea indica el nombre de la entrada digital: digitalio ('parallel','LPT1').
Como se sabe, el Puerto Paralelo puede ser de entrada y salida.



En este conector:

8 pines son para salida de datos (bits de DATOS), y van desde el pin 2 (bit menos significativo) hasta el pin 9 (bit más significativo).
5 pines son de entrada de datos (bits de ESTADO). Estos pines son: 15, 13, 12, 10 y 11, del menos al más significativo.
4 pines son de control (bits de CONTROL). Tienen la característica de ser bidireccionales es decir que los puedes utilizar tanto de entrada como de salida. Estos pines son: 1, 14, 16 y 17, del menos al más significativo.

Sin embargo, configurando en la BIOS del PC (accesible en unos PCs con la tecla F12, en otros con la tecla Del o Supr, y a veces con F10) el puerto paralelo (LPT1) como de entrada y salida, es posible usar los pines del 2 al 9 como de entrada.

Para determinar cuántos pines es posible podemos usar en el Puerto Paralelo, se ejecuta:

```
>> parport = digitalio('parallel','LPT1');
>> hwinfo = daqhwinfo(parport)
hwinfo =
AdaptorName: 'parallel'
DeviceName: 'PC Parallel Port Hardware'
ID: 'LPT1'
Port: [1x3 struct]
SubsystemType: 'DigitalIO'
TotalLines: 17
VendorDriverDescription: 'Win I/O'
VendorDriverVersion: '1.3'
```

Como se puede ver en la información, 17 de los 25 pines del PP se los puede utilizar como I/O. Los restantes pines son tierra.

Una vez creada la entrada digital del Puerto Paralelo, lo que sigue es asignar qué pines serán para entrada y cuáles para salida, para lo cual se usa la función addline, cuya sintaxis es:

```
>> dato2= addline(parport,0:7,'in'); %Para valores de entrada
>> dato = addline(parport,0:7,'out'); %Para valores de salida
```

Y se obtiene el dato de entrada con dato3=getvalue(dato2). Se asigna el dato al puerto con: putvalue(dato,255).

Por ejemplo, con el siguiente script se puede encender un led:

```
%Puerto paralelo
parport = digitalio('parallel','LPT1');
dato = addline(parport,0:7,'out');
putvalue(dato,2)
```

CONTROL DEL PUERTO SERIAL COM

A continuación se presenta las instrucciones básicas para el control del puerto serie en Matlab, con línea de comandos.

A diferencia del puerto paralelo que lo pueden usar simultáneamente dos o más programas a la vez, el puerto serial será de uso exclusivo de un solo programa. Para usar este puerto es necesario establecer los parámetros de su funcionamiento, como son: baudrate, bits de datos, bit de parada, etc.

```
%ABRIR el puerto COM1
clc; disp('BEGIN')
SerPIC = serial('COM1');
set(SerPIC,'BaudRate',2400);
set(SerPIC,'DataBits',8);
set(SerPIC,'Parity','none');
set(SerPIC,'StopBits',1);
set(SerPIC,'FlowControl','none');
fopen(SerPIC);
%*-*-*-*-*-*-
```

Para escribir datos se usa la función fprintf:

```
fprintf(SerPIC,'%c',char(100));%Envía en código ASCII
fprintf(SerPIC,'%s',’100’);%Envía un string
```

Para leer los datos del puerto se usa la función fscanf.

```
s1 = serial('COM1');
s1.BaudRate=9600;
fopen(s1);
fscanf(s1)
```

Luego de realizar la comunicación, los pasos para cerrar el puerto son:

```
%CERRAR el puerto COM1 al finalizar
fclose(SerPIC);
delete(SerPIC)
clear SerPIC
disp('STOP')
```

CODIGO FUENTE

Clase Princpipal


```
public class Principal extends javax.swing.JFrame {

    public Principal() {
        initComponents();
    }

    private void enviarparaleloActionPerformed(java.awt.event.ActionEvent evt) {
        // ENVIA EL NUMERO AL PUERTO PARALELO maximo 255
        JMatLink engine = new JMatLink();
        String c;
        String d;
        c=NUMENVIAR.getText();
        d=c+";";
        engine.engOpen();
        engine.engEvalString("parport = digitalio('parallel','LPT1'); dato =
addline(parport,0:7,'out'); n1="+d+" putvalue(dato,n1); pause(0.004); delete(dato); clear
dato; ");
        engine.engClose();
    }

    private void recibeparaleloActionPerformed(java.awt.event.ActionEvent evt) {
        // RECIBE NUMERO DEL 1 AL 15 DEL  PUERTO PARALELO
        JMatLink engine = new JMatLink();
        double a2;
```

```
        engine.engOpen();
        engine.engEvalString("parport = digitalio('parallel','LPT1');dato = addline(parport,9:12,0,'in');out = getvalue(dato);entro= binvec2dec(out); pause(0.004); delete(dato); clear dato;s=entro ");
        a2 = engine.engGetScalar("s");
        engine.engClose();
        int c = (int)a2;
        String muestra1=String.valueOf(c);
        NUMRECIBIDO.setText(muestra1);
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // ENVIA POR EL PUERTO SERIAL maximo 6 caracteres
        JMatLink engine = new JMatLink();
        String PALABRAENVIA;
        String d;
        PALABRAENVIA=CADENAENVIA.getText();
        d="'"+PALABRAENVIA+"'";
        engine.engOpen();
        engine.engEvalString("clear all;close all;clc;PS=serial('COM1');set(PS,'Baudrate',9600);set(PS,'StopBits',1);set(PS,'DataBits',8); set(PS,'Parity','none');set(PS,'Terminator','CR/LF');set(PS,'OutputBufferSize',6);set(PS,'InputBufferSize'
,6);set(PS,'Timeout',1);fopen(PS);fprintf(PS,'%s',"+d+");fclose(PS);delete(PS);clear PS;");
        engine.engClose();
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        // Lee por el serial un caracter
        JMatLink engine = new JMatLink();
        double a2;
        String p[];
        String q="";
        engine.engOpen();
        engine.engEvalString("clear all;close all;clc;PS=serial('COM1');set(PS,'Baudrate',9600);set(PS,'StopBits',1);set(PS,'DataBits',8); set(PS,'Parity','none');set(PS,'Terminator','CR/LF');set(PS,'OutputBufferSize',6);set(PS,'InputBufferSize'
,6);set(PS,'Timeout',2);fopen(PS);variable=fread(PS,1,'uchar');u=double(variable);fclose(PS);delete(PS);clear PS;p=u");
        a2= engine.engGetScalar("p");
        engine.engClose();
        int c = (int)a2;
        char d=(char)c;
        String muestra=String.valueOf(d);
        CADENARECIBI.setText(muestra);
    }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
```

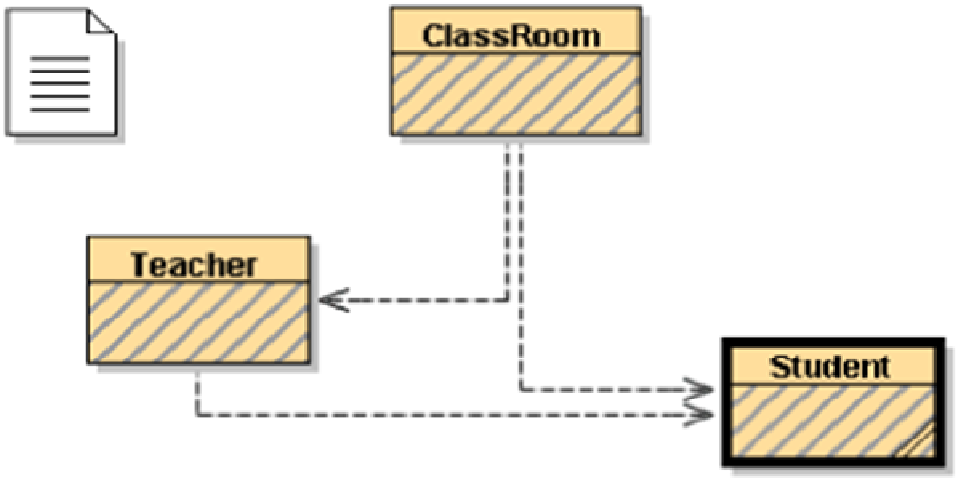
```
        // BOTON SALIR
            System.exit(0);
    }
public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Principal().setVisible(true);
            }
        });
    }
    public static javax.swing.JTextField CADENAENVIA;
    public static javax.swing.JTextField CADENARECIBI;
    public static javax.swing.JTextField NUMENVIAR;
    public static javax.swing.JTextField NUMRECIBIDO;
    public static javax.swing.JButton enviarparalelo;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JPanel jPanel5;
    private javax.swing.JPanel jPanel6;
    public static javax.swing.JButton recibeparalelo;
}
```

ANEXO I

UML

LENGUAJES Y SÍMBOLOS DE MODELIZACIÓN

Los métodos de diseño actuales de OOP utilizan el UML para definir construcciones y modelos OO. El UML es resultado de los esfuerzos por estandarizar la terminología y la diagramación de los modelos de objetos. Muchos productos de software proporcionan herramientas gráficas para crear diagramas UML. UML tiene muchos tipos diferentes de diagramas que pueden utilizarse para describir modelos de objetos.



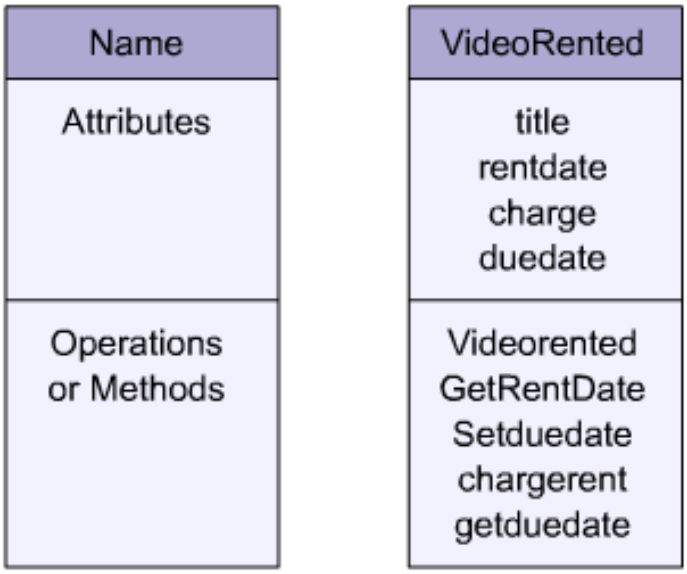
¿Qué es exactamente UML?

UML es una colección de símbolos y patrones de símbolos. Se lo utiliza para describir objetos, la relación entre objetos, el uso de objetos y el estado de los mismos.

El propósito de la introducción de este lenguaje es proporcionar una comprensión y una familiaridad muy elementales con el diagrama de Clase.

Esta técnica de diagramación ayuda a comunicar a un programador de manera simple las definiciones de una clase y sus relaciones con otras clases en una forma fácil de comprender.

Diagrama de clase. Un diagrama de clase describe los atributos de los objetos del sistema.



Hay varios símbolos asociados con un diagrama de clase:

- Los rectángulos describen la clase
- Las líneas describen las relaciones
- Los símbolos especiales describen la accesibilidad y la fortaleza de las relaciones

Un rectángulo representa una clase de objetos. Está dividido en tres compartimientos, el compartimiento nombre, el compartimiento atributo, y el compartimiento operación o método.

Los símbolos se utilizan para indicar accesibilidad. La línea de puntos se utiliza para representar una implementación y/o una asociación. Los símbolos +, -, # se utilizan para describir modificadores de acceso para cada atributo o método de la clase. Estos diagramas representan un pequeño conjunto de símbolos UML.

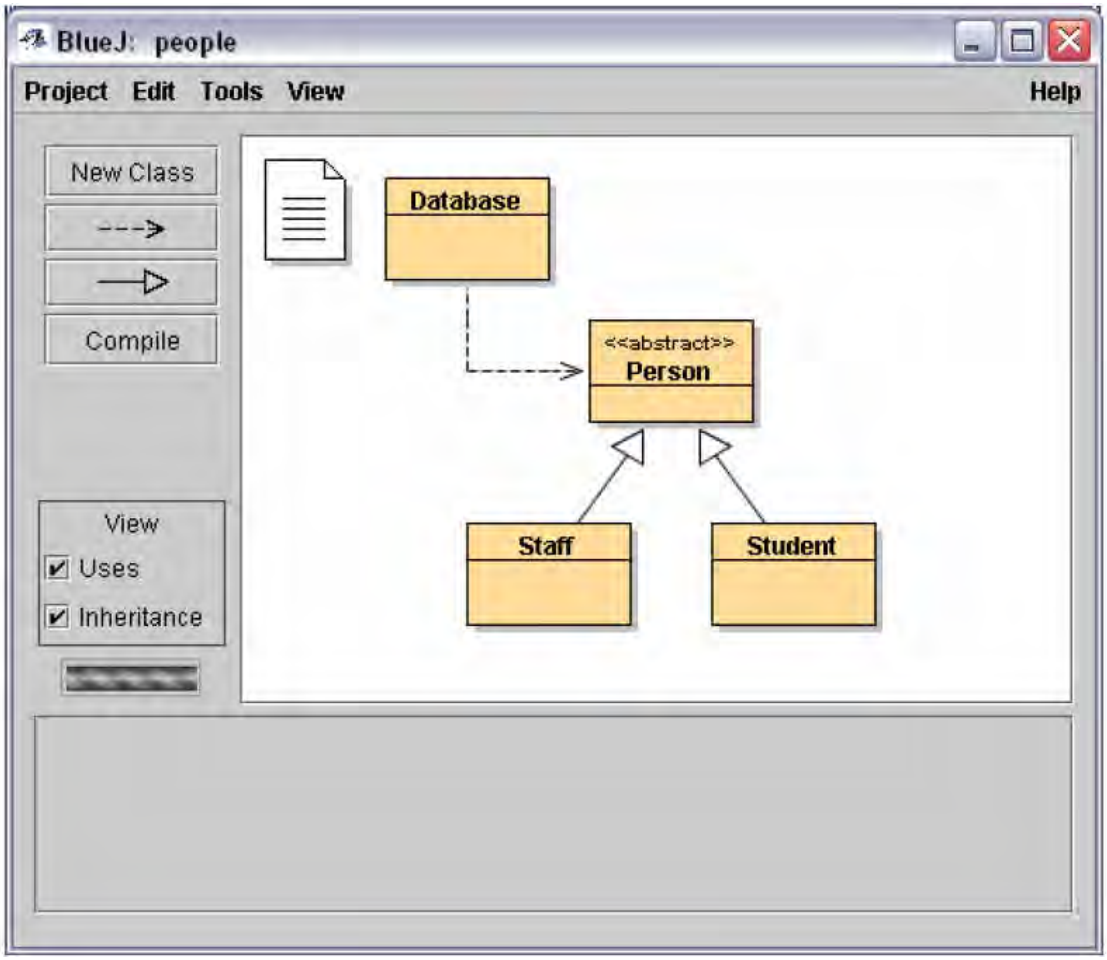
- "+" — public
- "-" — private
- "#" — protected
- " " — no symbol indicates default access.

ANEXO II

BLUEJ

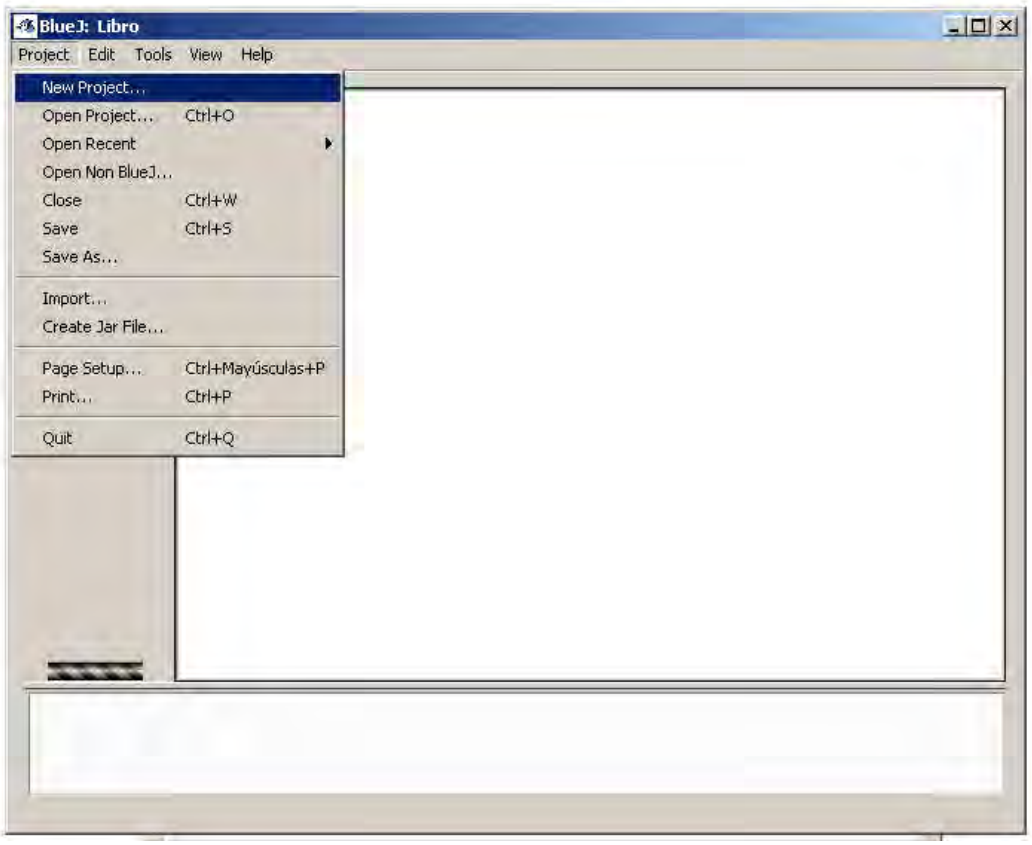
BlueJ. BlueJ es un entorno de desarrollo para Java™ diseñado específicamente para la enseñanza en un curso introductorio. Fue diseñado e implementado por el equipo de BlueJ en la Universidad de Monash, Melbourne, Australia, y la Universidad de Southern Denmark, Odense.

BlueJ es distribuido en tres formatos diferentes: uno para sistemas Windows, uno para MacOS, y uno para los otros sistemas. Para la instalación se debe tener J2SE v1.3 (o. JDK 1.3) o posterior instalado en el sistema para utilizar BlueJ.



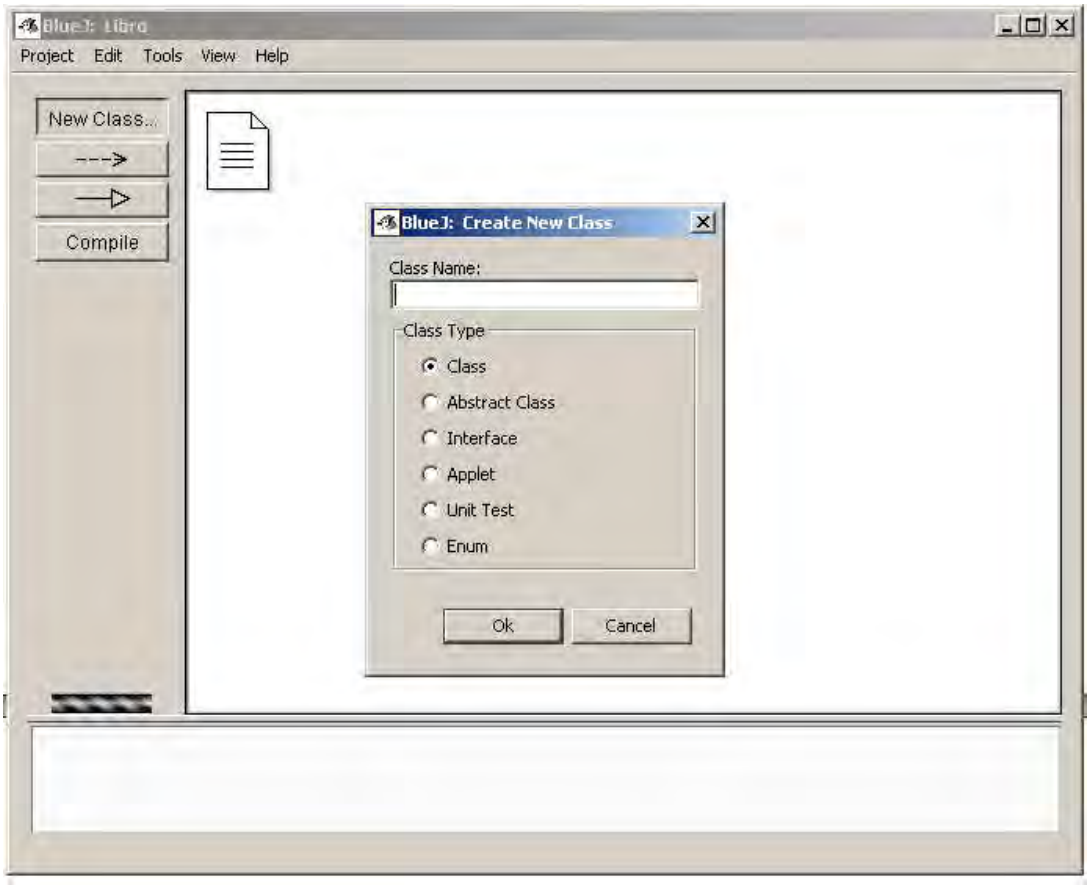
CREACION DE UN PROYECTO

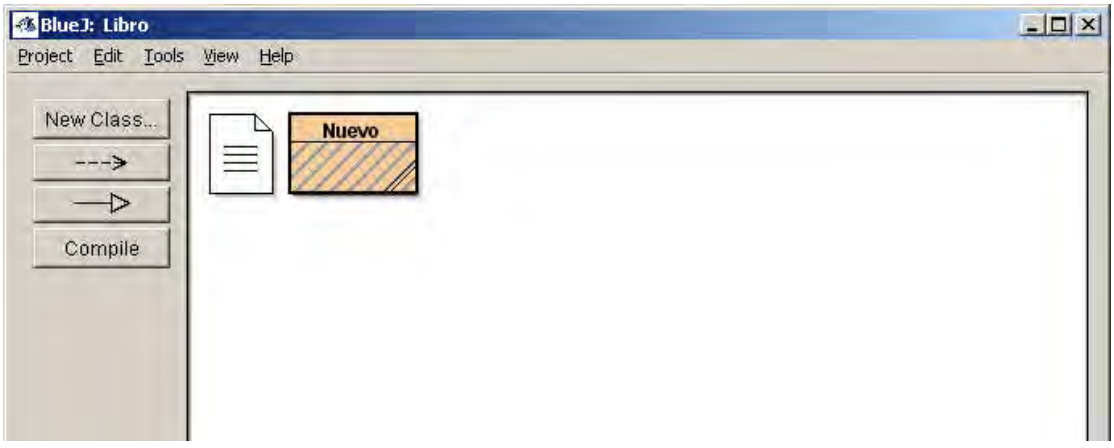
Para la creación de un proyecto se accede al menú Proyect y se selecciona New Proyect, a continuación establezca el nombre del proyecto.



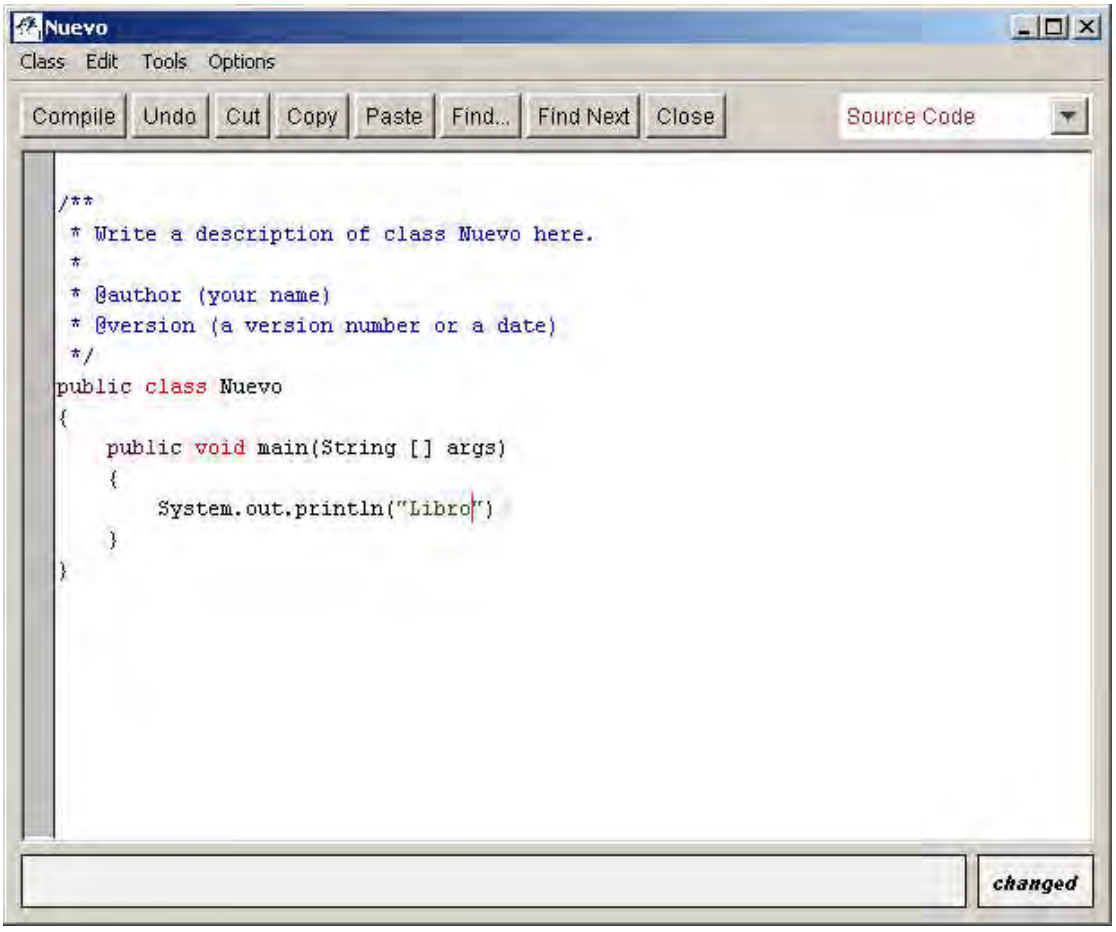
CREACION DE UNA CLASE

Para crear una clase selección en botón New Class con lo que se despliega un nueva ventana donde se especifica el nombre de la clase y su tipo.

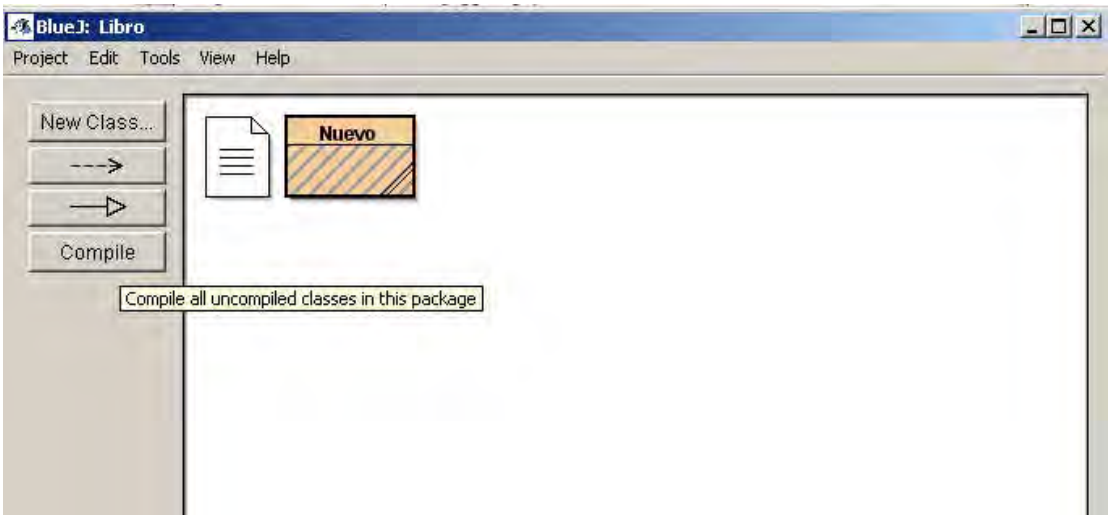




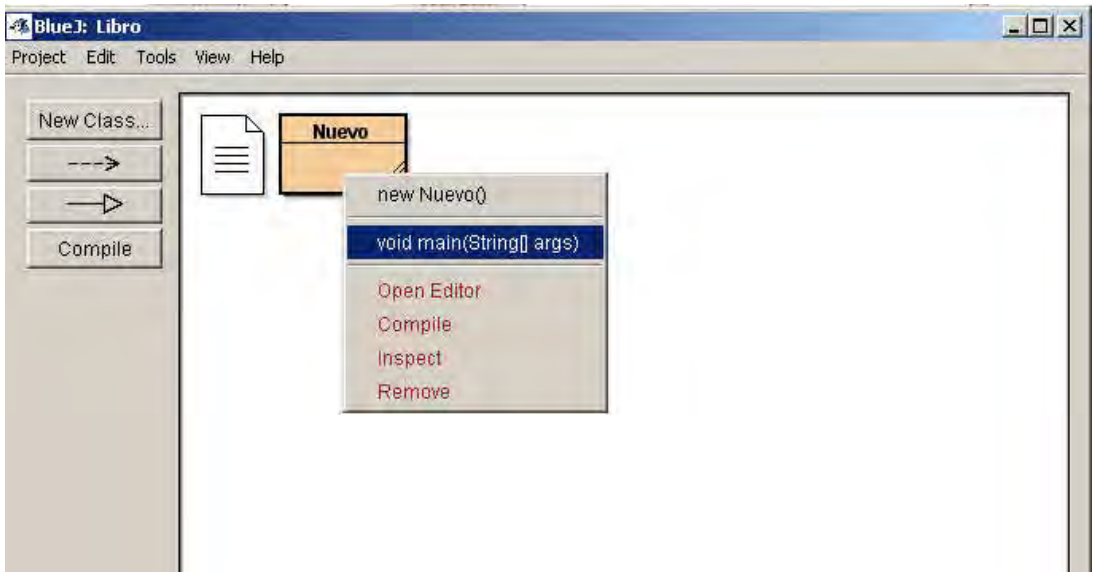
Una vez creada la clase de da un doble clic sobre el rectángulo que representa la clase y se reemplaza el código auto generado.

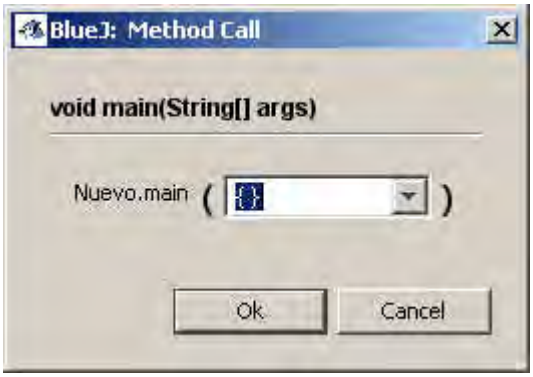


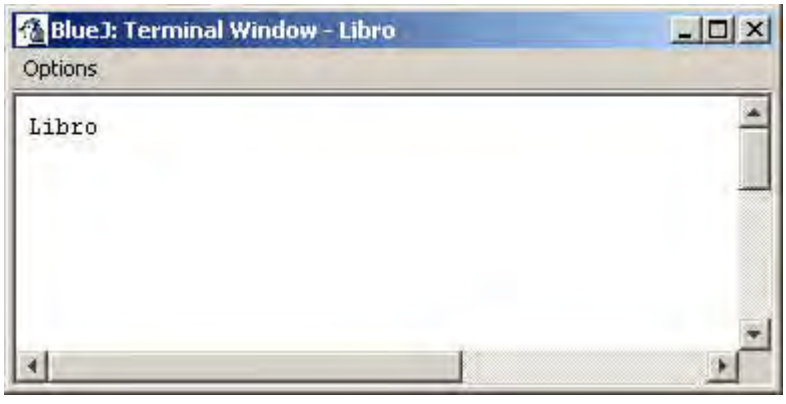
Para la compilación se debe hacer un presionar el botón Compile.



Para ejecutar la aplicación se debe hacer un click con el botón derecho del mouse y seleccionar el método main.


BlueJ: Libro
Project Edit Tools View Help
New Class...
Compile
Nuevo
new Nuevo()
void main(String[] args)
Open Editor
Compile
Inspect
Remove


BlueJ: Method Call
void main(String[] args)
Nuevo.main ( )
Ok
Cancel


BlueJ: Terminal Window - Libro
Options
Libro

ANEXO III

NETBEANS

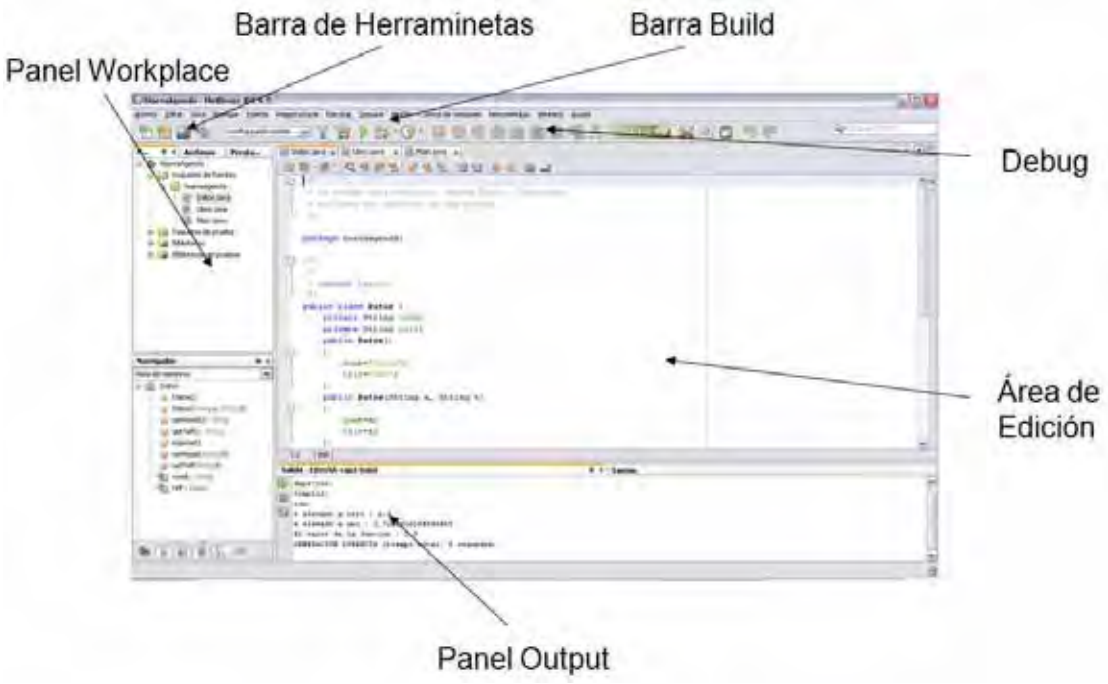
Para poder escribir programas se necesita de un entorno de desarrollo Java, por lo que Sun Microsystems, proporciona uno de forma gratuita, el J2SE Develoment Kit. Opcionalmente se puede trabajar con un entorno de desarrollo integrado (IDE) que facilitan las tareas de creación de la interfaz gráfica de usuario, edición de código, compilación, ejecución y depuración. Como es Netbeans.

Netbeans es un IDE multilenguaje modular que presentar algunas características como:

- Soporta para Java SE, Java EE y Java ME
- Soporta módulos de terceros (plugins)
- Permite el desarrollo intuitivo drag-and-drop.
- Dispone de Debugger, Porfiler, Refactoring y Completación de código
- Es gratuito.
- Es de código abierto desde Junio de 2000
- Existe una gran comunidad de usuarios y desarrolladores.
- Permite aplicaciones completas para cliente
- Crea ventanas, menús, barras de herramientas y acciones fácilmente.

Netbeans nace como un proyecto estudiantil en la República Checa en 1996, con el nombre de Xelfi y se constituye en el primer IDE para Java escrito en Java. Jorda Tulach, miembro del equipo original propone el nombre de Netbeans (Network + Java Beans)

Netnbeans permite desarrollar aplicaciones de escritorio, Web, Mobile y Enterprise, con Java, C/C++, PHP, Groovy, Phyton, JavaScript, etc.



El Workplace muestra las partes de la aplicación en dos partes principales:

- Project: Permite navegar y manipular el código fuente.
- File: Permite encontrar y editar cada uno de los diferentes recursos de la aplicación, incluyendo los diseños, iconos y menús de la ventana.

El panel Output es el lugar donde se presenta cualquier información. Es donde se podrá observar las instrucciones de progreso del compilador, las advertencias y los mensajes de error. Este se abrirá automáticamente cuando NetBeans necesite desplegar un mensaje.

El área de edición, es el lugar donde se desplegaran las ventanas necesarias para editar el código fuente, así como el lugar donde se desplegará la ventana necesaria para diseñar la aplicación.

Al ejecutar NetBeans por primera vez se desplegaran dos barras de herramientas debajo de la barra de menú. Las dos barras de herramientas que se abren inicialmente son:

- *La barra de herramientas Standard:* contiene la mayoría de las herramientas estándar para abrir y guardar archivos, cortar, copiar, pegar y para una gran variedad de comandos que son sumamente útiles.
- *La barra Build:* que ofrece los comandos de construcción y de ejecución que seguramente se utilizara cuando desarrolle y pruebe las aplicaciones.

La barra de Debug, muestra las herramientas necesarias para hacer la depuración del programa. Permite ejecutar el programa paso a paso. Permite mediante “Watch” observar los valores que tienen los objetos.

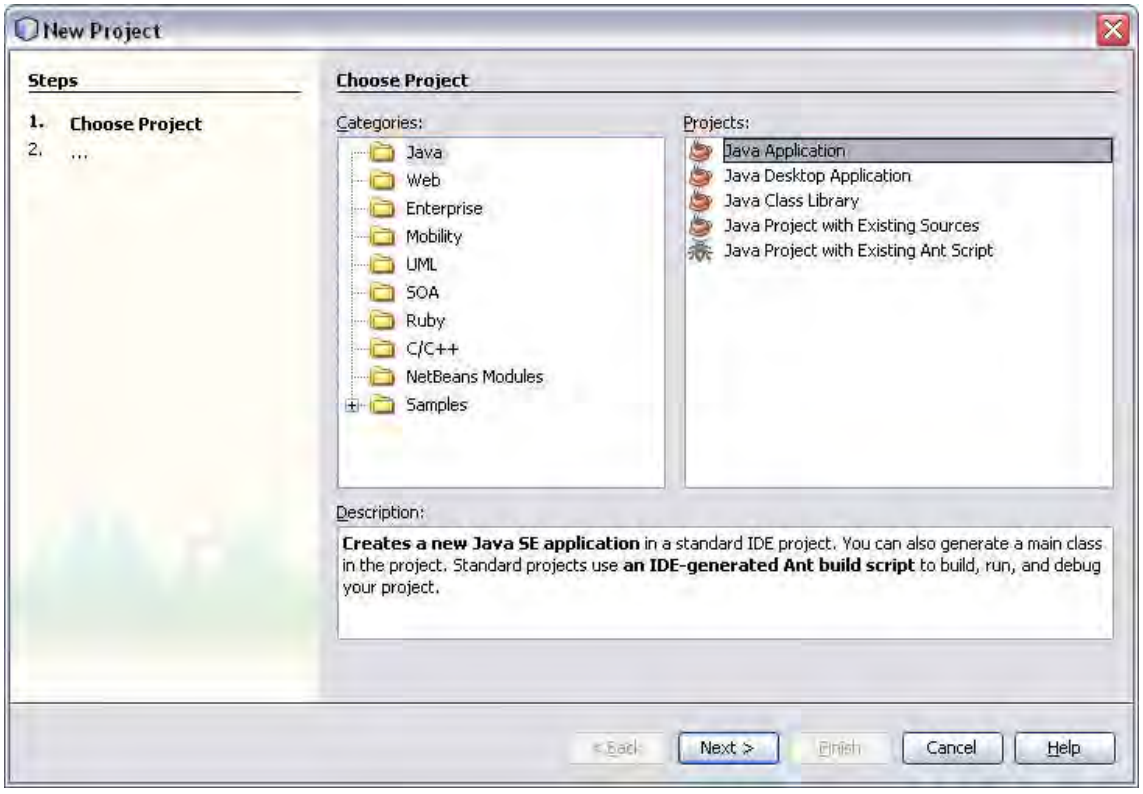
COMO CREAR UNA NUEVA APLICACIÓN EN NETBEANS

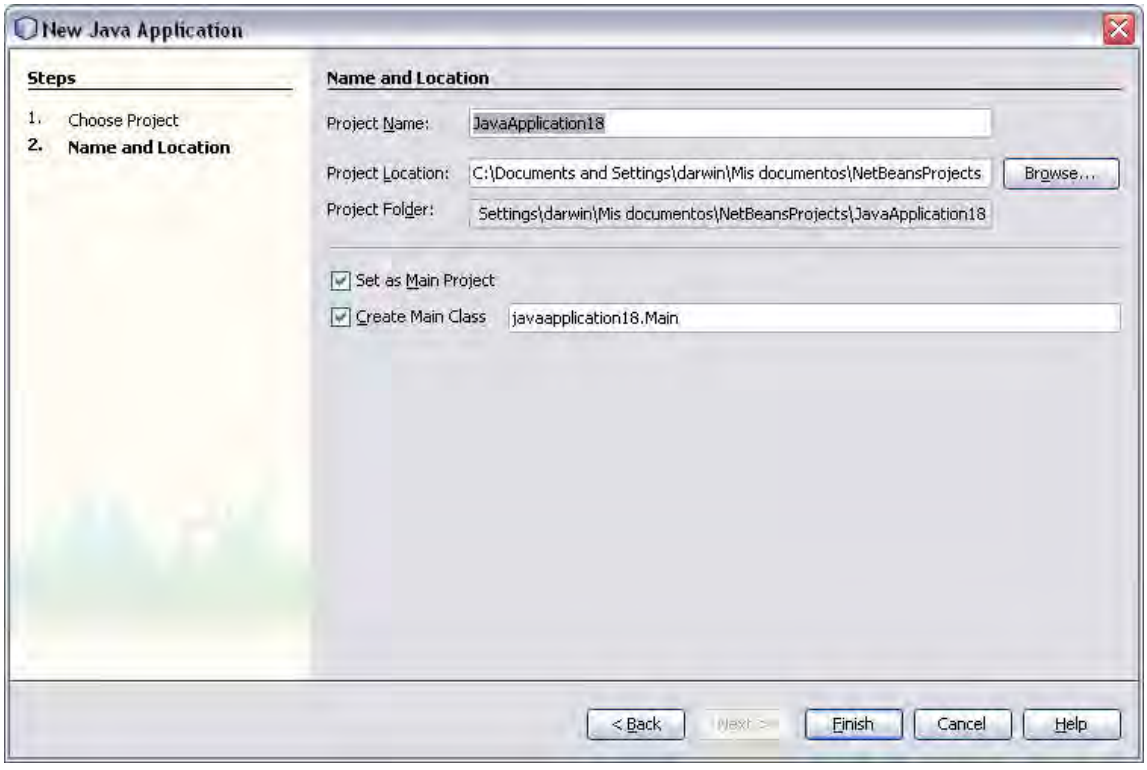
Un proyecto permite administrar los archivos con el código fuente y compilado de una aplicación.

1. Ejecute el programa NetBeans

2. Del menú principal de NetBeans, seleccione la opción **File**/**New Project … ,** presione las teclas **Ctrl+Mayúsculas+N** o haga clic en el icono **New Project**.

3. Aparecerá la primera ventana del asistente para crear un nuevo proyecto

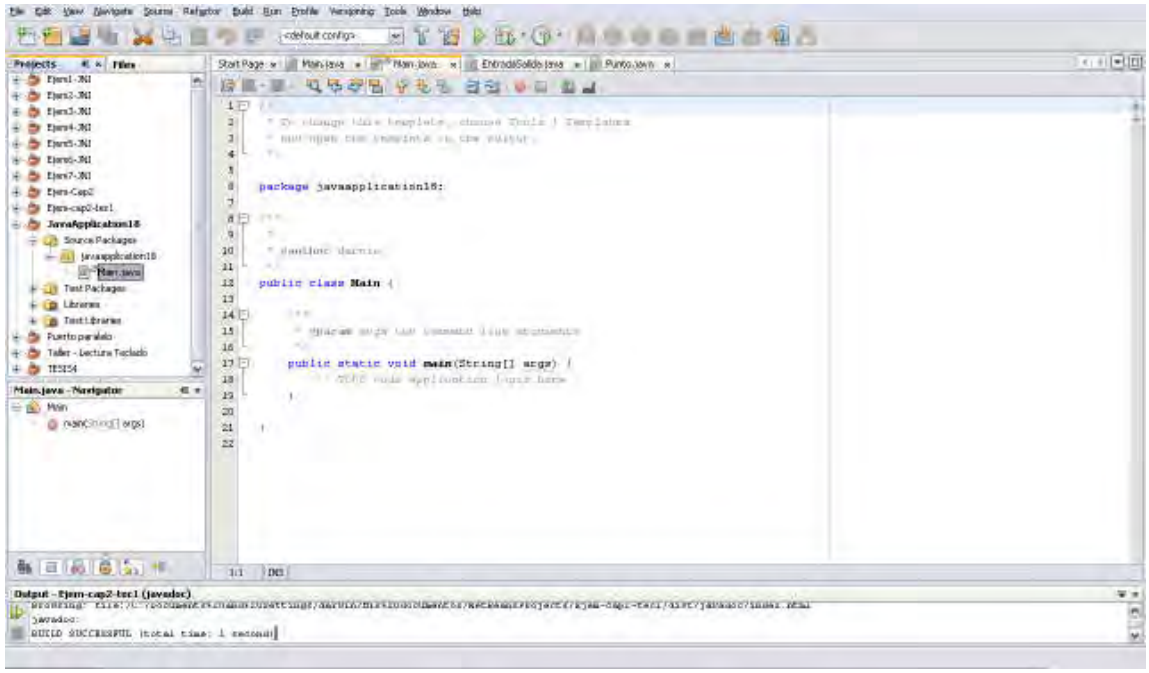




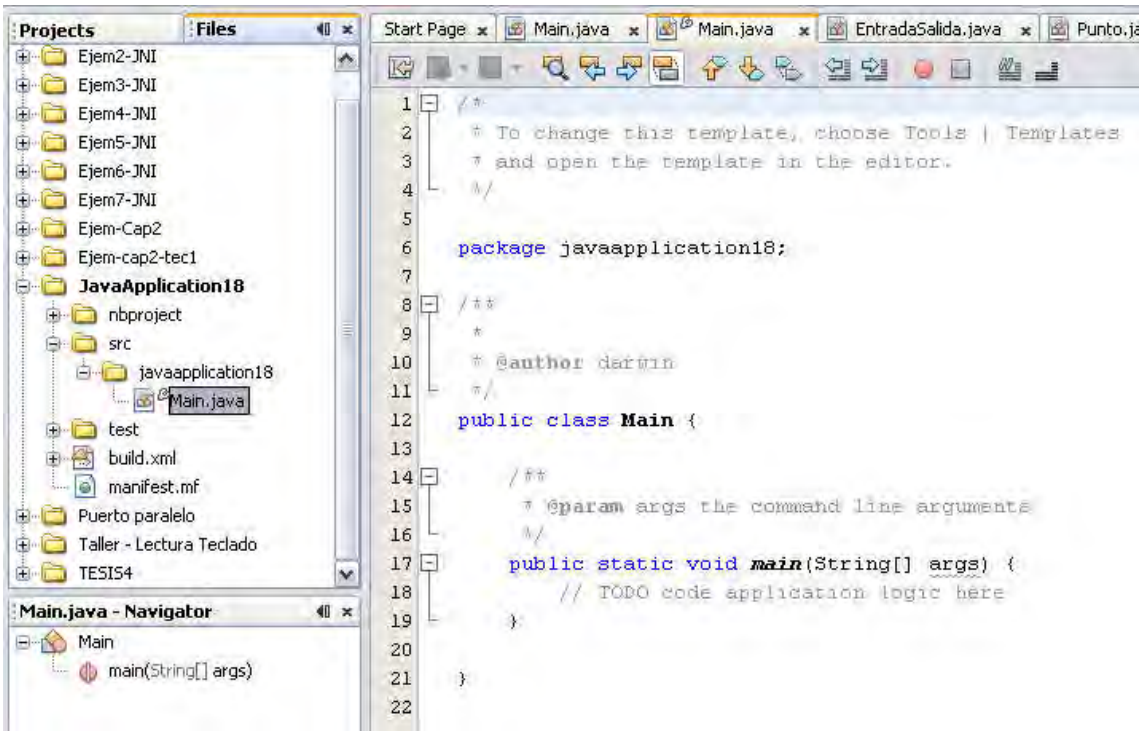
Set as Main Project (Hace que este proyecto sea el proyecto principal)

Create Main Class (Cree la clase principal, la clase con el método main())

4. Desaparecerá el asistente para crear un nuevo proyecto. Del lado derecho aparece el editor de NetBeans con el esqueleto de la clase principal, mientras que del lado izquierdo aparece el árbol de los proyectos.



5. El recuadro del árbol de los proyectos hacemos clic en la pestaña Files, aparecerá un árbol con todos los archivos de los proyectos

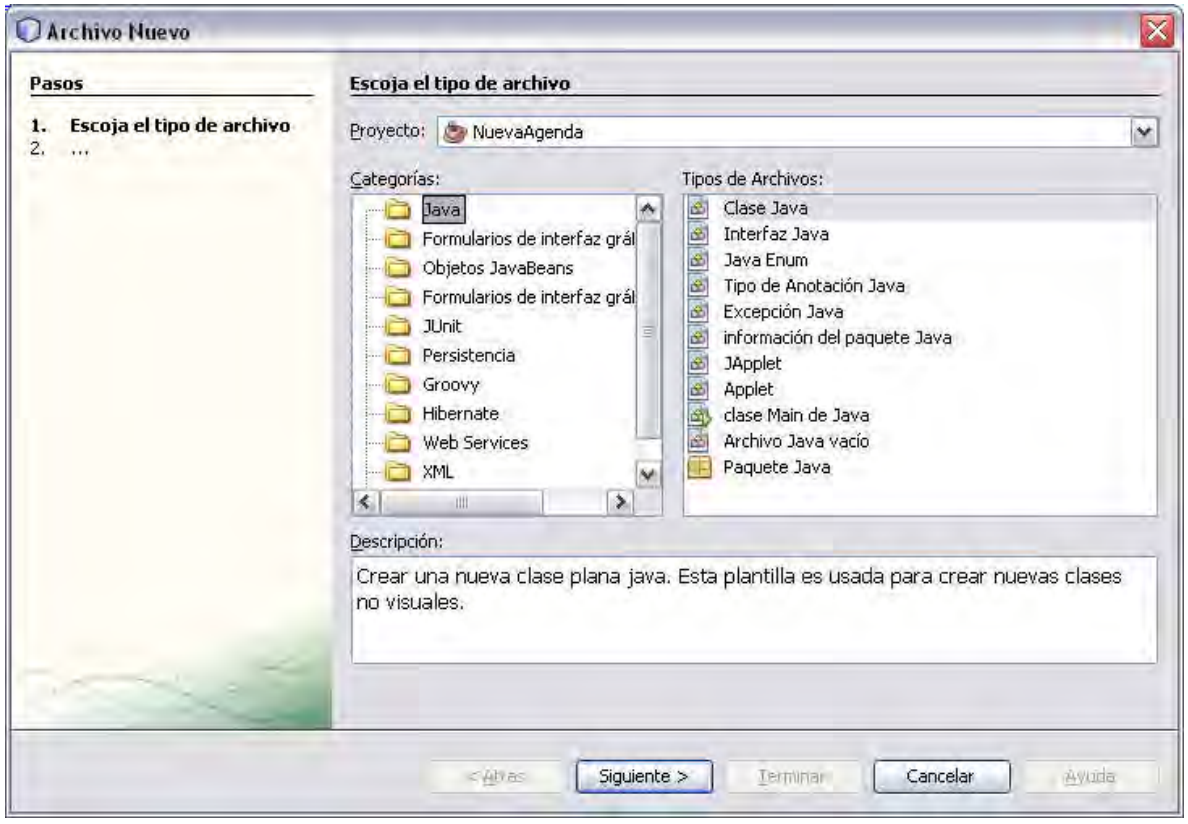


CREACIÓN DE UNA CLASE

Para crear una clase se sigue el siguiente procedimiento:

1. Del menú principal de NetBeans, seleccione la opción **Files/New File**, presione las teclas **Ctrl+ N** o haga clic en el icono **New File**.

2. Aparecerá la primera ventana del asistente para crear una nueva clase. En esta ventana del asistente seleccionaremos el tipo de clase que deseamos crear.



3. Aparecerá la segunda ventana del asistente para crear clases. En esta ventana seleccionaremos el nombre y la ubicación de la clase.

a) Establezca el nombre de la clase (**Class Name**).

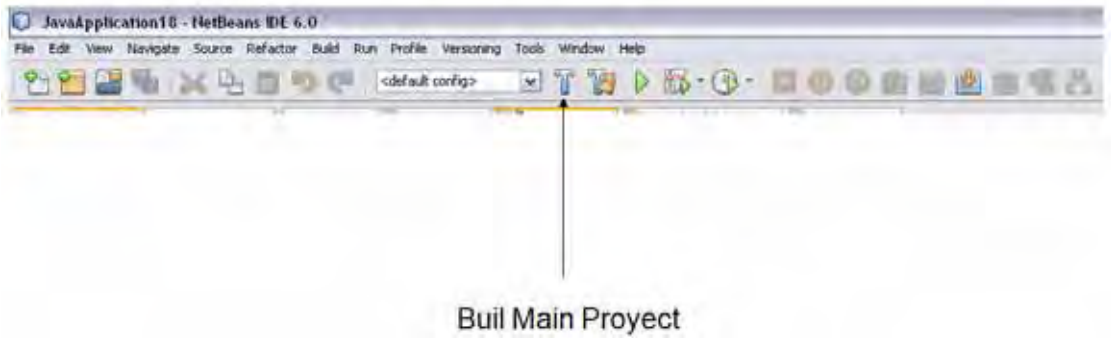
b) Establezca el paquete donde estará la clase (**Package**).

COMPILACIÓN DE UNA CLASE

Para compilar la clase que se encuentra en la ventana de edición seleccione del menú principal la opción **Build/Compile "*NombreClase*.java". "*NombreClase*"** es el nombre de la clase a compilar.

COMPILACIÓN DEL PROYECTO

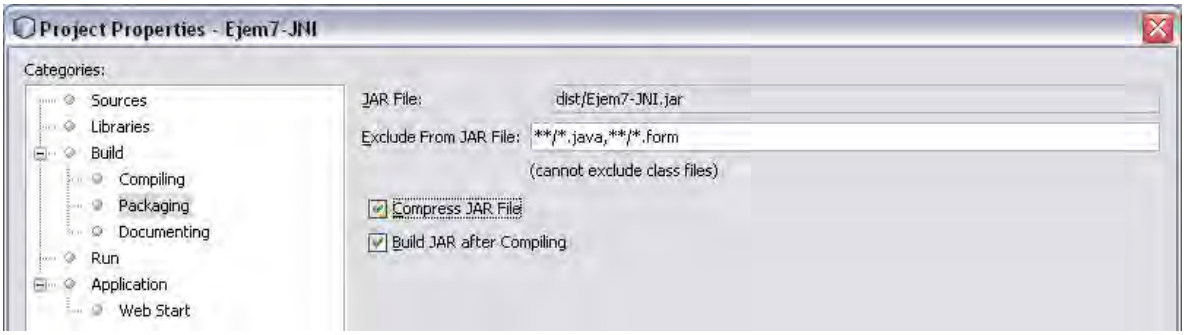
Para compilar todas las clases de un proyecto seleccione del menú principal la opción **Build/Build Main Project**, presione la tecla **F11** o presione el icono **Build Main Project**.



Durante la compilación, NetBeans muestra los mensajes resultantes del proceso, como se muestra en la figura.

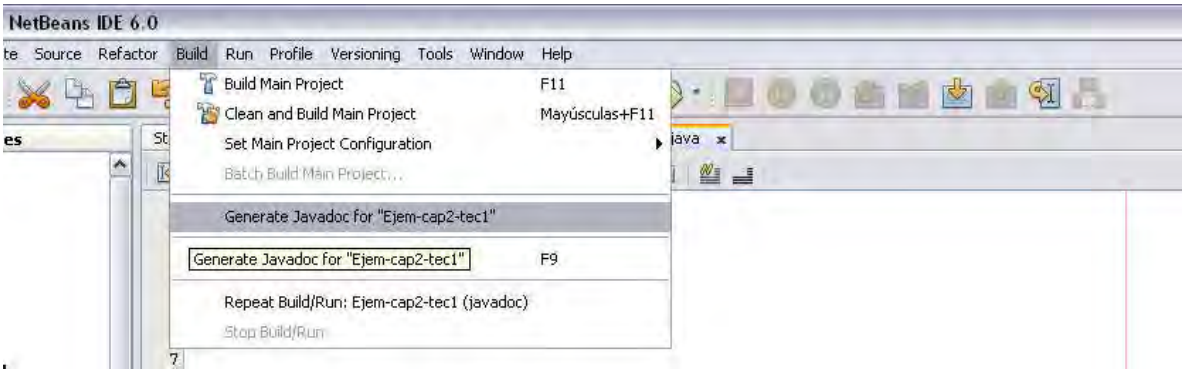
GENERACION DE EJECUTABLES

Para poder generar un “ejecutable” con la extensión jar se debe marcar la opción *Compress JAR File*, dentro de cuadro de propiedades del proyecto en *Packaging*.



GENERADOR DE DOCUMENTACIÓN

Es una herramienta que permite la generación de documentación API directamente desde el código fuente Java. Genera páginas HTML basadas en las declaraciones y comentarios

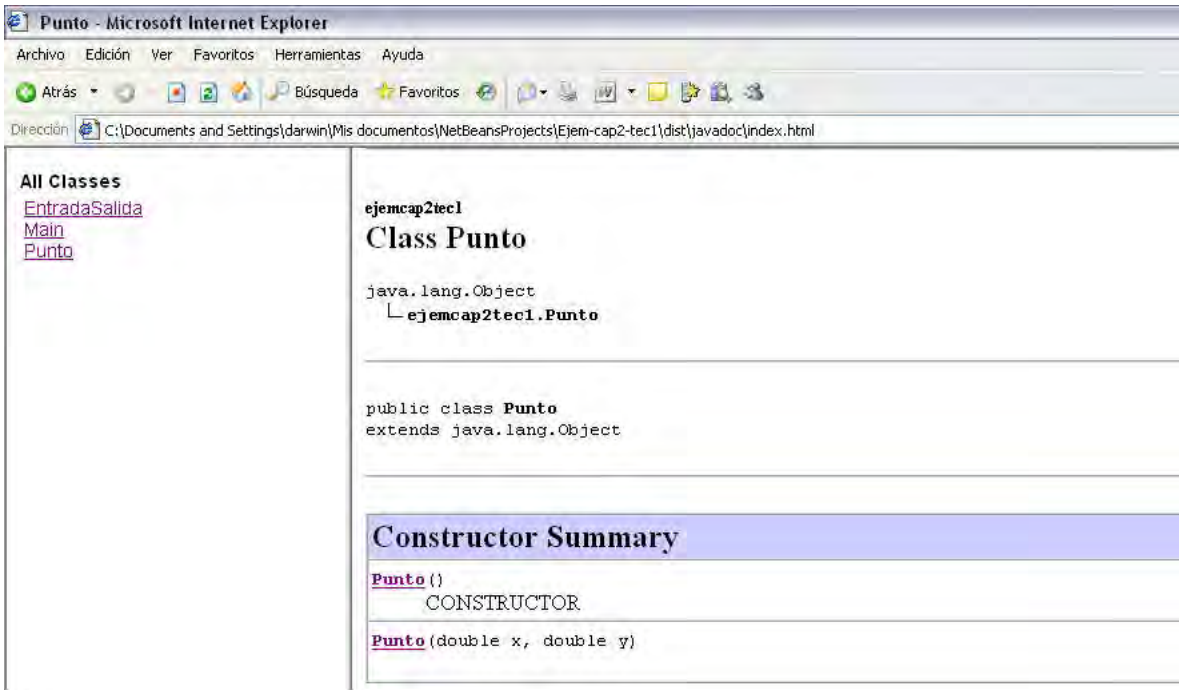


Un comentario de documentación empieza con los caracteres /** y termina con los caracteres */. Cada comentario consiste de una descripción seguida de una o más etiquetas. Se pueden usar etiquetas de formateo HTML en los comentarios de documentación.

```java
package nuevaagenda;

/**
 *
 * @author Darwin
 * @version 2.0
 */
public class Datos {
    private String nomb;
    private String telf;
    /**
    Constructor sin argumentos
    */
    public Datos()
    {
        nomb="vacio";
        telf="000";
    }
        /**
    Constructor con argumentos
         * @param n = nombre
         * @param t = telefono
    */
    public Datos(String n, String t)
    {
        nomb=n;
        telf=t;
    }
```

Se pueden insertar comentarios de documentación para clases, interfaces, métodos, atributos y constructores:

- Los comentarios de la clase o interfaz se colocan en la parte superior del archivo, después de las sentencias **import** e inmediatamente antes de la declaración de la clase o interfaz.
- Los comentarios de los atributos se colocan inmediatamente antes de su declaración.
- Los comentarios de los métodos y constructores se colocan inmediatamente antes de la declaración de su firma.


Punto - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Favoritos
Dirección C:\Documents and Settings\darwin\Mis documentos\NetBeansProjects\Ejem-cap2-tec1\dist\javadoc\index.html
All Classes
EntradaSalida
Main
Punto
ejemcap2tec1
Class Punto
java.lang.Object
ejemcap2tec1.Punto
public class Punto
extends java.lang.Object
Constructor Summary
Punto()
CONSTRUCTOR
Punto(double x, double y)

nuevaagenda

Class Datos

```
java.lang.Object
  └nuevaagenda.Datos
```

Direct Known Subclasses:

Libro

```
public class Datos
extends java.lang.Object
```

Version:

2.0

Author:

Darwin

Constructor Summary

Datos() Constructor sin argumentos
Datos(java.lang.String n, java.lang.String t) Constructor con argumentos

Constructor Detail

Datos

```
public Datos()
```

Constructor sin argumentos

Datos

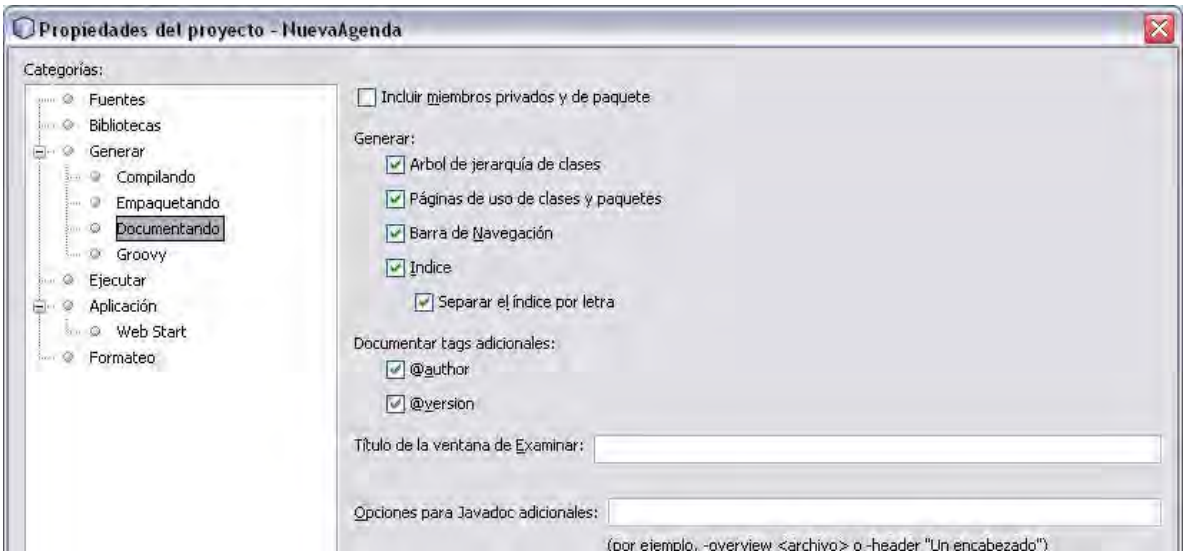
```
public Datos(java.lang.String n,
             java.lang.String t)
```

Constructor con argumentos

Parameters:

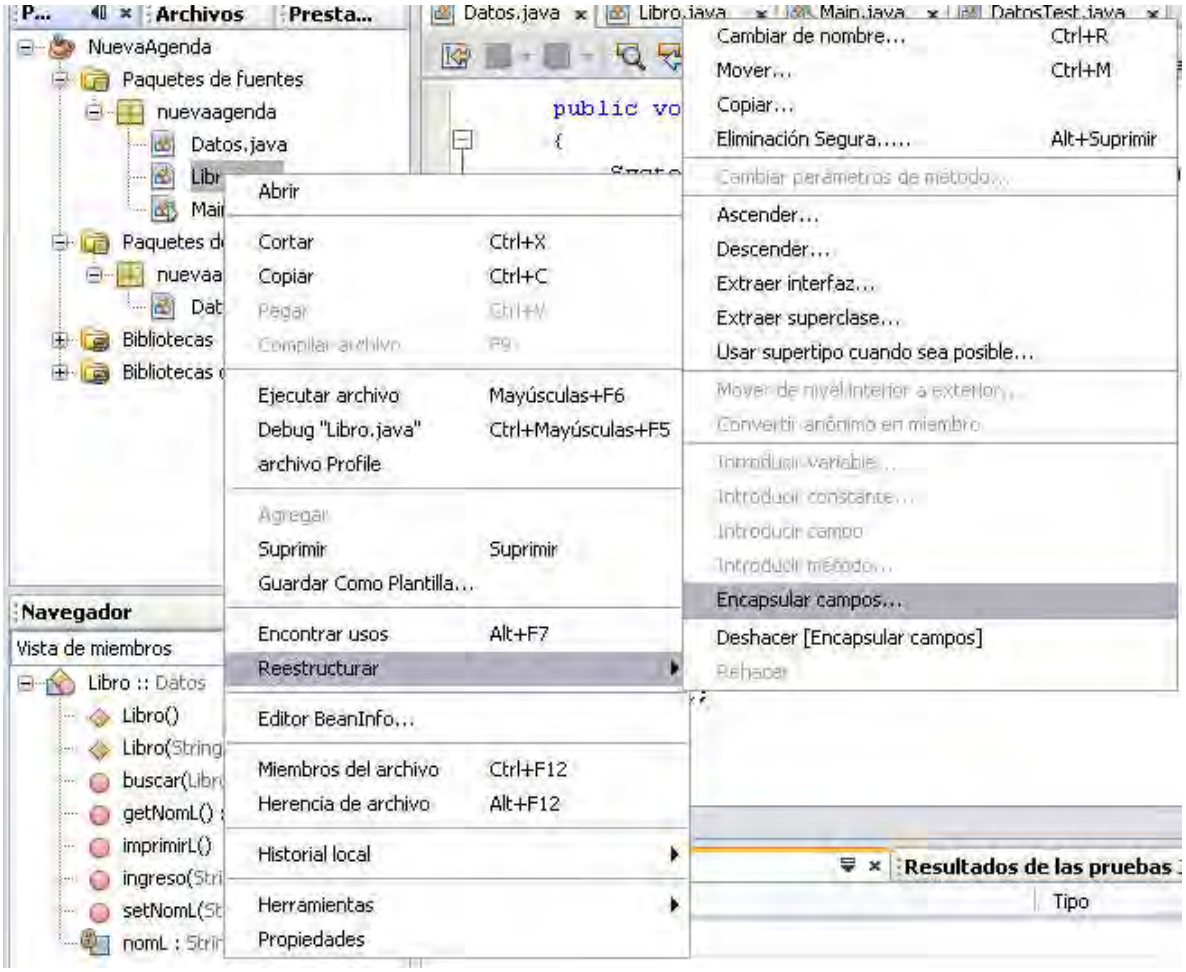
n - = nombre

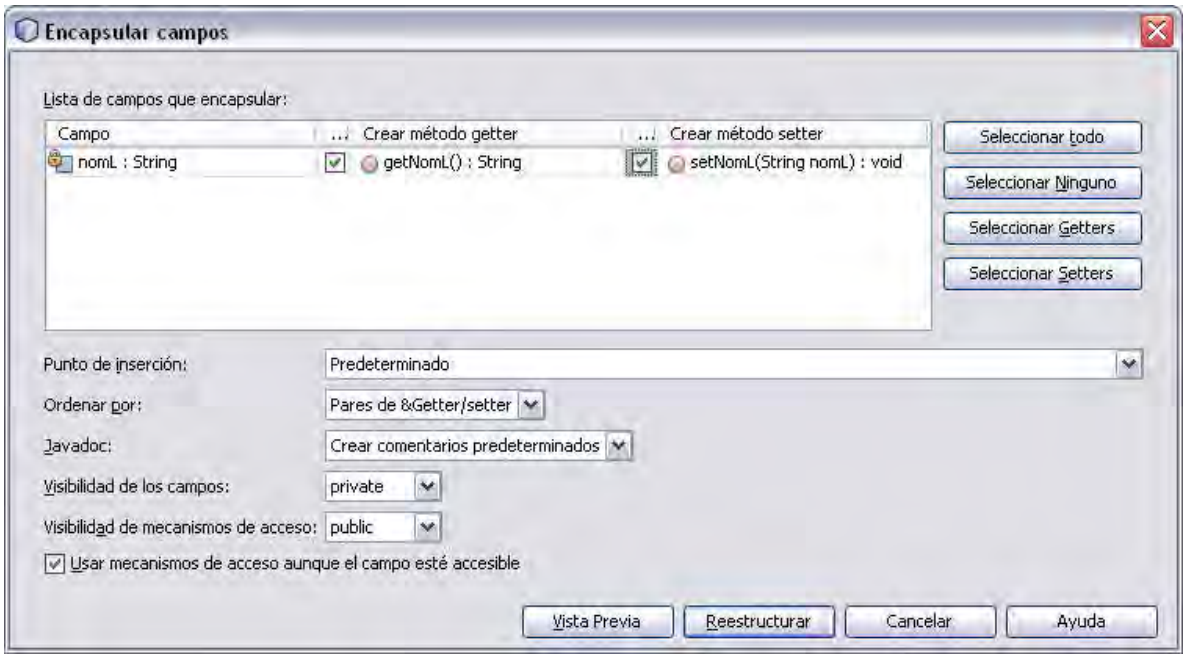
t - = telefono



GENERACION DE M´´ETODOS DE CONSULTA Y DE MODIFICACION

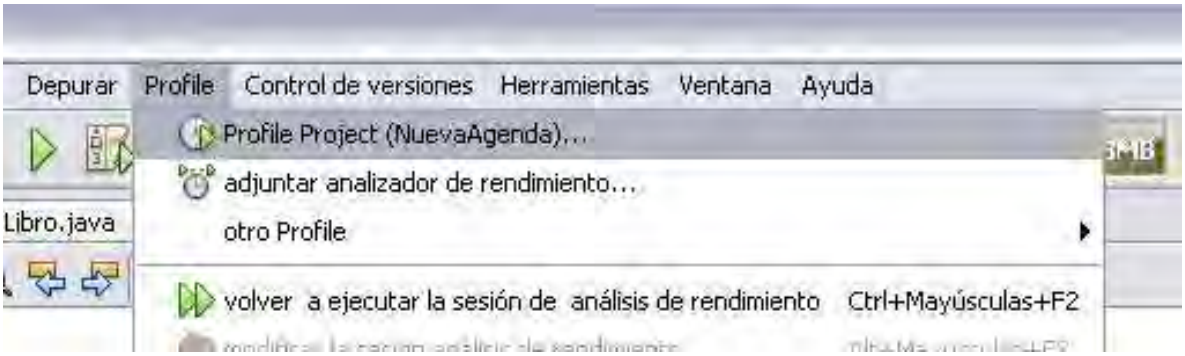
Para generar métodos de acceso (consulta y modificación), se procede de la siguiente manera:

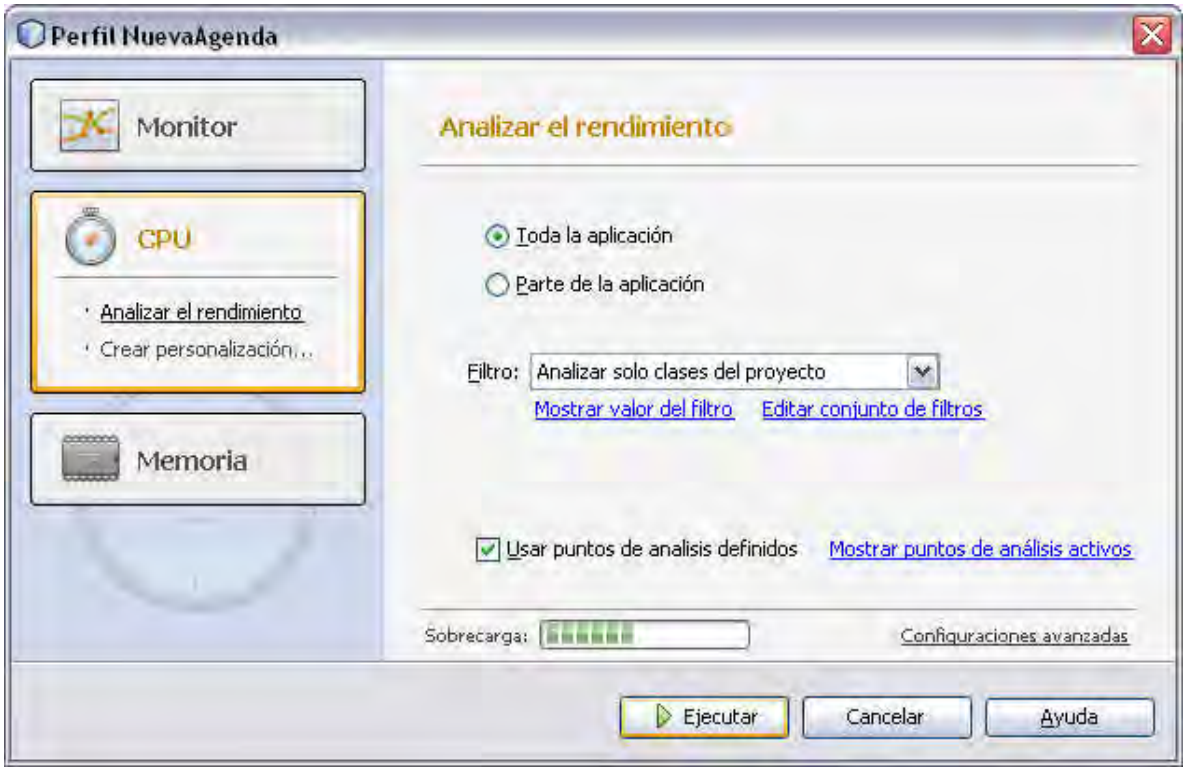

Archivos
Presta...
NuevaAgenda
Paquetes de fuentes
nuevaagenda
Datos.java
Bibliotecas
Datos.java
Libro.java
Main.java
DatosTest.java
public vo
Abrir
Cortar Ctrl+X
Copiar Ctrl+C
Pegar Ctrl+V
Compilar archivo F9
Ejecutar archivo Mayúsculas+F6
Debug "Libro.java" Ctrl+Mayúsculas+F5
archivo Profile
Agregar
Suprimir Suprimir
Guardar Como Plantilla...
Encontrar usos Alt+F7
Reestructurar
Editor BeanInfo...
Miembros del archivo Ctrl+F12
Herencia de archivo Alt+F12
Historial local
Herramientas
Propiedades
Cambiar de nombre... Ctrl+R
Mover... Ctrl+M
Copiar...
Eliminación Segura..... Alt+Suprimir
Cambiar parámetros de método...
Ascender...
Descender...
Extraer interfaz...
Extraer superclase...
Usar supertipo cuando sea posible...
Mover de nivel interior a exterior...
Convertir anónimo en miembro
Introducir variable...
Introducir constante...
Introducir campo
Introducir método...
Encapsular campos...
Deshacer [Encapsular campos]
Rehacer
Navegador
Vista de miembros
Libro :: Datos
Libro()
buscar(
getNomL()
imprimirL()
ingreso(
setNomL(
nomL : Strir
Resultados de las pruebas
Tipo

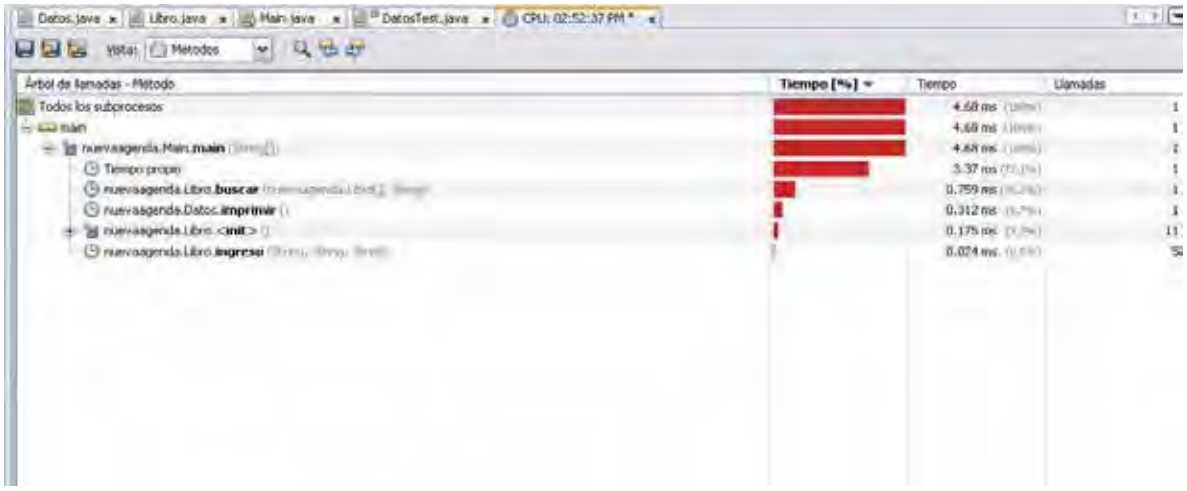

Encapsular campos
Lista de campos que encapsular:
Campo
Crear método getter
Crear método setter
nomL : String
getNomL() : String
setNomL(String nomL) : void
Seleccionar todo
Seleccionar Ninguno
Seleccionar Getters
Seleccionar Setters
Punto de inserción:
Predeterminado
Ordenar por:
Pares de &Getter/setter
Javadoc:
Crear comentarios predeterminados
Visibilidad de los campos:
private
Visibilidad de mecanismos de acceso:
public
Usar mecanismos de acceso aunque el campo esté accesible
Vista Previa
Reestructurar
Cancelar
Ayuda

```
/**
 * @return the nomL
 */
public String getNomL() {
    return nomL;
}

/**
 * @param nomL the nomL to set
 */
public void setNomL(String nomL) {
    this.nomL = nomL;
}
```

MANEJO DEL PROFILE

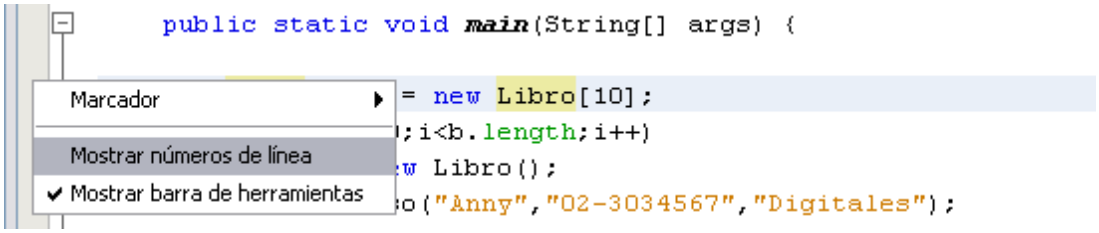






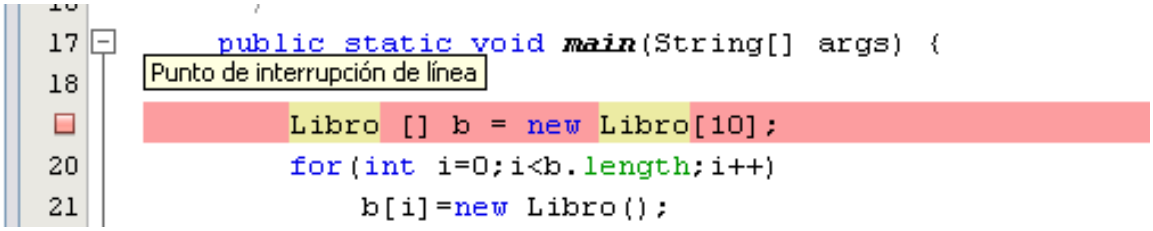
NÚMERO EN LAS LÍNEAS DE CÓDIGO

Para establecer número en las líneas de código haga click con el izquierdo del ratón sobre el borde gris del área de trabajo y selecciones Mostrar números de líneas.



DEPURACION DE APLICACIONES

Establecer un punto de parada inicial. Para ello, haga click con el izquierdo del ratón sobre el número de línea que quiere ejecutar paso a paso.



Debug Main Proyect o Ctrl+F5.- Inicia la ejecución de la aplicación en modo depuración hasta encontrar un punto de parada o hasta el final si no hay puntos de parada.

Toggle Line Brakpoint o Ctrl+F8.- Pone o quita un punto de parada en la línea sobre la que está el punto de inserción.

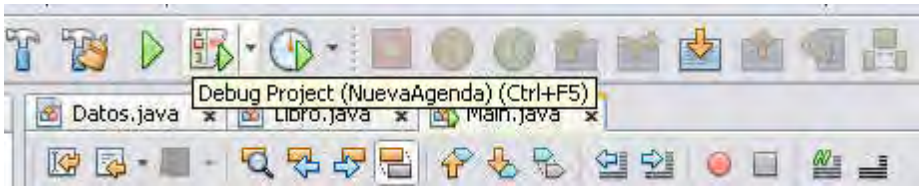
Finish Debugger Session o Mayús + F5.- Detiene el proceso de depuración.

Step Into o F7.- Ejecuta la aplicación paso a paso. Si la línea a ejecutar coincide con una llamada a un método definido por el usuario, dicho método también se ejecuta paso a paso.

Step Over o F8.- Ejecuta la aplicación paso a paso. Si la línea a ejecutar coincide con una llamada a un método definido por el usuario, dicho método no se ejecuta paso a paso, sino de una sola vez.

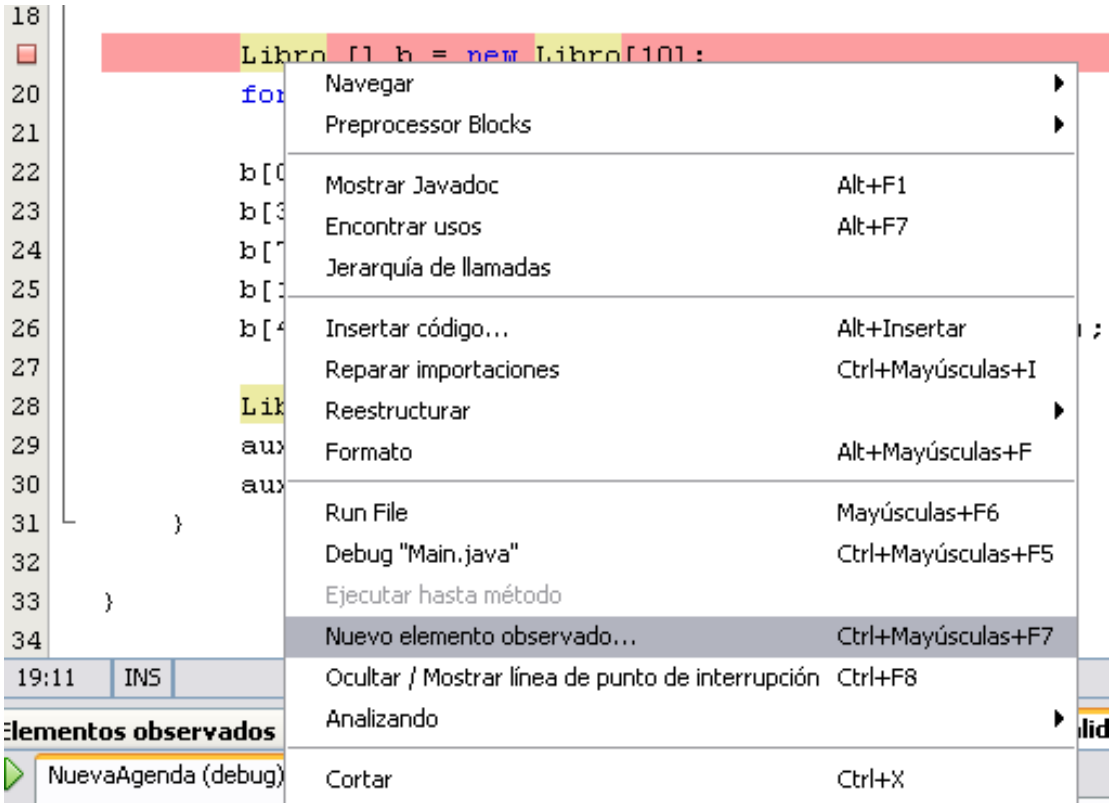
Step Out o Crtl+F7.- Cuando un método definido por el usuario ha sido invocado para ejecutarse paso a paso, utilizando esta orden se puede finalizar su ejecución en un solo paso.

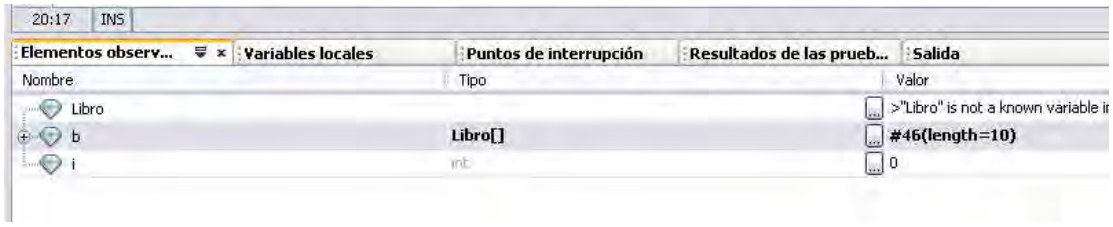
Run to Cursor o F4.- Ejecuta el código que hay entre la última línea ejecutada y la línea donde se encuentra el punto de inserción.



ELEMENTOS OBSERVADOS

Para ver los valores intermedios que van tomando las variables ponga el cursor sobre ellas, y seleccione Nuevo elemento observado


Navegar
Preprocessor Blocks
Mostrar Javadoc Alt+F1
Encontrar usos Alt+F7
Jerarquía de llamadas
Insertar código... Alt+Insertar
Reparar importaciones Ctrl+Mayúsculas+I
Reestructurar
Formato Alt+Mayúsculas+F
Run File Mayúsculas+F6
Debug "Main.java" Ctrl+Mayúsculas+F5
Ejecutar hasta método
Nuevo elemento observado... Ctrl+Mayúsculas+F7
Ocultar / Mostrar línea de punto de interrupción Ctrl+F8
Analizando
Cortar Ctrl+X
19:11
INS
Elementos observados
NuevaAgenda (debug)


20:17
INS
Elementos observ...
Variables locales
Puntos de interrupción
Resultados de las prueb...
Salida
Nombre
Tipo
Valor
Libro
>"Libro" is not a known variable i
b
Libro[]
#46(length=10)
i
int
0

BIBLIOGRAFIA

http://es.wikipedia.org/wiki/Ofuscaci%C3%B3n

http://es.wikipedia.org/wiki/Descompilador

Hozner, S. (2003). Creación de Paquetes, Interfaces, Archivos jar y Java Beens. Java 2. Coriolis. (941-948)

http://download.cnet.com/Cavaj-Java-Decompiler/3000-2213_4-10071619.html.

Descripción de ProGuard. http://mac.softpedia.com/es/programa-ProGuard-54793.html

Ofuscación. http://es.wikipedia.org/wiki/Ofuscaci%C3%B3n.

Álvarez, G. (1997-1998). Ofuscación de Código. http://www.iec.csic.es/criptonomicon/java/ofuscacion.html.

Canales, R. (2003). Decompilación de Código Java. file:///F:/tutoriales.php.htm.

ClickTeam. (2011). http://www.clickteam.com/website/usa/downloads/index/10.

González, D. (2009). Ofuscar Código Java. http://mundogeek.net/archivos/2009/06/08/ofuscar-codigo-java/.

Guachún, A. (2008).Crear un instalador para programas Java. http://www.javahispano.org/contenidos/es/crear_un_instalador_para_programas_java__andres_guachun/.

Kumar, S. (2001-2003). Decompilation. http://www.debugmode.com/dcompile/.

Lafortune, E. (2002-2011). ProGuard Version 4.6.http://proguard.sourceforge.net/.

Lic. Valerio A. (2005). http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=15.

Martínez, B. (2009). ¿Cómo crear un instalador para una aplicación Java? . http://api.ning.com/files/sxxo6HxbwDRv-O*lZINVmykCnug9VnIMkM2gukyYqfD5KLksBY2VnIOlzpmzSNNKHBCPSUycsfnikonRoAJX6LxHRKj3Q-xs/CmocrearuninstaladorparaunaaplicacinJava.pdf.

NetBeans. (2011). http://netbeans.org/downloads/zip.html.

Nolan, G. (2004). Decompiling Java.
http://adrastea.ugr.es/search*spi/?searchtype=t&searcharg=Decompiling+Java.

Olivares, J. C. (2007). Construcción de Proyectos de Software.
http://www.slidefinder.net/c/construcci%C3%B3n_proyectos_software_juan_carlos/24893007.

RegExLab.com. (2009). http://www.brothersoft.com/jar2exe-wizard-47319.html.

Salvador, M. (2007). Decompiladores. http://draxus.org/upload/decompiladores.pdf.

TechSmith. (2011). http://www.techsmith.com/download/camtasia/.

Barkodar. (2007). A. UML: Adding Class Elements. http://www.youtube.com/watch?v=a-I24gTEdHg&feature=BFa&list=PL564558906264BE8B&index=3

Barkodar. (2007). B. UML: Adding Elements.
http://www.youtube.com/watch?v=j1NB33IvwFU&feature=BFa&list=PL564558906264BE8B&index=6 .

Barkodar. (2007). C. UML: Adding Elements.
http://www.youtube.com/watch?v=XIbOXhgfq2w&feature=BFa&list=PL564558906264BE8B&index=7 .

Barkodar. (2007). D. UML: Compiling and Executing Class Elements.
http://www.youtube.com/watch?v=OhdUneAyiB4&feature=BFa&list=PL564558906264BE8B&index=4.

Barkodar. (2007). E. UML: Generating Java Source Code.
http://www.youtube.com/watch?v=EoHkCYW72Ew&playnext=1&list=PL564558906264BE8B.

Barkodar. (2007). F. UML: UML: Identify Association
http://www.youtube.com/watch?v=I33tM9DTnAI&feature=BFa&list=PL564558906264BE8B&index=2.

Wfrancootero. (2011). Instalacion Plugin Uml para Netbeans 6.9.1.
http://www.youtube.com/watch?v=NgyyoFrOB_0.

Diccionario Informático
http://www.webdeveloper.com/forum/showthread.php?t=125418.

Ejemplos/Comandos Runtime
http://www.chuidiang.com/java/ejemplos/Runtime/runtime.php

Comando.net, búsqueda, cuestiones generales, consultado el
http://www.algoritmia.net/articles.php?i

http://www.programacion.com/articulo/graficos_con_java_2d_111.

http://laurel.datsi.fi.upm.es/_media/docencia/cursos/java/java2d.pdf

http://es.wikipedia.org/wiki/API_(inform%C3%A1tica)

http://es.scribd.com/doc/51574853/76/formas-2D

http://nereida.deioc.ull.es/~cleon/pa/0304/prct/p8/node1.html

http://www.cidse.itcr.ac.cr/revistamate/HERRAmInternet/Graficador-Swing-java2D/node4.html

http://sourceforge.net/projects/jep/

http://www.singularsys.com/jep/

Gilbert D. (2007). “Practica 7”. PDF. Extraído el 17 de julio de 2011.

Harsha Perla. (2010). “LCD configuración final”.PDF .

Pérez G. (2009). “Java Chart”. http://www.java2s.com/Code/Java/Chart/CatalogChart.htm

Ing. Emilio La Torre. (2010). “Display LCD”.PDF.

Puerto paralelo. Disponible:
http://www.modelo.edu.mx/univ/virtech/circuito/paralelo.htm

Hardware del puerto paralelo. Disponible:
http://www.modelo.edu.mx/univ/virtech/circuito

Ejercicio1: Servomotor., Disponible: http://hombrepie.wordpress.com/2009/09/23/ejercicio-1-arduino-y-servomotor/

Servomotores. Disponible:
http://www.monografias.com/trabajos60/servo-motores/servo-motores.shtml

El servomotor.Disponible:
http://www.info-ab.uclm.es/labelec/solar/electronica/elementos/servomotor.htm

Cómo trucar un motor. Disponible:
http://www.ucontrol.com.ar/wiki/index.php?title=Como_trucar_un_servo#Trucaje

Hytherion. Disponible:
http://www.hytherion.com/beattidp/comput/pport.htm

The oficialUserPort homepage., Disponible: http://hem.passagen.se/tomasf/UserPort/

Puerto Serie [en línea] http://www.informaticamoderna.com/El_puerto_serial.htm

Puerto Serie [en línea] http://es.wikipedia.org/wiki/Puerto_serie [6 de julio de 2011]

Comandos AT[en línea] http://alarmagsm.googlecode.com/files/COMANDOS AT.doc

Aplicación de los comandos AT[en línea] http://www.foroswebgratis.com/mensaje-los_comandos_at-129640-1449569-1-4561138.htm

Bryan Veloso (2008). Implementación de código para utilizar Puerto Paralelo. http://drwn.wordpress.com/2008/06/05/ejemplo-de-termometro-en-java/.

Edward Crean (1998). Descargarse las librerías y bibliotecas para controlar el puerto paralelo. http://www.filetransit.com/download.php?id=84452/.

Sebastián Herrera (2002). Descargarse la biblioteca Java.comm para controlar el puerto paralelo. http://www.topshareware.com/javax-comm-jar/downloads/1.htm.

Francisco José Molina López (2004). Funcionamiento de un Motor de Corriente Continua. http://www.electronicafacil.net/tutoriales/El-MotorCC.php

Instituto de Tecnologías Educativas (2010). Funcionamiento y Aspectos Generales de un Dispositivo Relé. http://platea.pntic.mec.es/~pcastela/tecno/documentos/apuntes/rele.pdf.

Albert Parker (2010). Introducción y Partes que constituyen un Motor de Corriente Continua. http://www.todorobot.com.ar/documentos/dc-motor.pdf.

Marcel Nory Durán (2005). Como trabaja y el funcionamiento de un Dispositivo Relé. http://www.unicrom.com/Tut_relay.asp.

Fred G. Martin (2004). Las librerías necesarias para la implementación del control de motor. http://www.cs.uml.edu/~fredm/courses/91.305-fall04/javasetup.shtml.

Mario Sacco (2010). Puerto H. http://www.neoteo.com/puente-h-para-motores-cc-parte-ii.neo.

Luftawaffe (2009). Funcionamiento y la construcción de un Puerto H para control de un Motor DC. http://fuhrer-luftwaffe.blogspot.com/2009/04/puente-h-con-transistores-npn.html.

http://galia.fc.uaslp.mx/~cantocar/microprocesadores/EL_Z80_PDF_S/20_TECLADO_MATRICIAL.PDF

http://www.x-robotics.com/rutinas.htm#Teclado Matricial 4x4

http://www.forosdeelectronica.com/f25/problema-utilizando-jk-memoria-20957/

http://es.wikipedia.org/wiki/Decodificador

Anónimo (2002). Conexión y programación con el puerto paralelo.

http://mimosa.pntic.mec.es/~flarrosa/puerto.pdf

Anónimo (2003). El puerto de impresora http://arantxa.ii.uam.es/ ~gdrivera/varios/notas_lpt.htm

Anónimo (2011). El puerto serial http://www.informaticamoderna.com /El_puerto_serial.htm.

Anónimo (2009). Java Comm Api http://lefunes.wordpress.com/2009/02/27/ instalacion-del-java-comm-api-en-windows/.

Monillo007 (2009), Leer la entrada de un puerto serial desde Java http://es.debugmodeon.com/articulo/leer-la-entrada-de-un-puerto-serial-desde-java

Wikipedia (2011), Proteus http://es.wikipedia.org/wiki/Proteus_(electr%C3%B3nica).

http://192.9.162.102/thread.jspa?threadID=5165624&messageID=9633447

http://www.held-mueller.de/Puertoserial/

http://tech.groups.yahoo.com/group/PuertoParalelo/

www.mathkb.com/Uwe/Forum.aspx/matlab/94235/GiovynetDriver_1.2_x86

http://www.programacion.com/articulo/jdc_tech_tips_22_de_enero_de_2002_160

http://www.held-mueller.de/JMatLink/

http://foros.solocodigo.com/viewtopic.php?f=72&t=37198

www.held-mueller.de/JMatLink/download.html
http://es.wikipedia.org/wiki/MATLAB

http://www.matpic.com/MATLAB/MATLAB_LPT_SERIAL.html

http://www.monografias.com/trabajos5/matlab/matlab.shtml

http://spanish.osstrans.net/software/jmatlink.html

Technical Reports (Informes Técnicos)
http://www.javahispano.org/contenidos/es/puertoparalelo_en_java_parte_1/;jsessionid=788802248797D4A9887D741D4E09FFAB

http://msdn.microsoft.com/es-es/library/system.pport%28VS.80%29.aspx

http://www.monografias.com/trabajos16/java/java.shtml

http://docentes.uni.edu.ni/fec/Giovanni.Saenz/Archivos%20Curso%20Java%20-%20Unidad%20II/Solo%20Java2D%20-%20Explicacion.pdf

Tutorial de BlueJ. Iván Alfonso Guarín V. Michael Kölling, Mærsk Insitute, University of Southern Denmark.

Microcontroladores PIC Programación en Basic. Carlos A. Reyes. Segunda edición. 2006

Aprenda Java como si estuviera en primero. García de Jalón, Rodriguez, Mingo, Imaz, Brazales, Laszarbal, Calleja, Gacrcía.

Java a travez de ejemplos. J. Bobadilla S.

JNI: Java Native Interface. MacProgramadores

http://www.itapizaco.edu.mx

http://www.programacion.net

http://www.javahispano.org

ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

ISBN: 978-9978-301-46-3

Publicaciones Científicas