

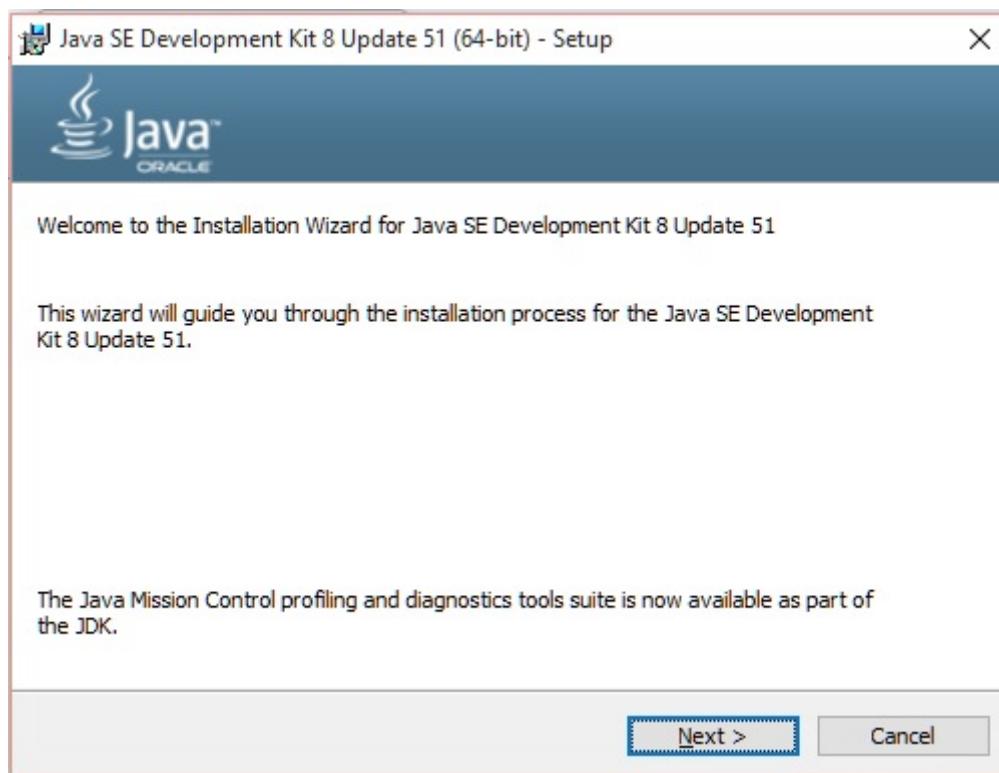
Tutorial  
Java Ya

## Descarga

Para poder hacer este curso debemos instalar el compilador de Java y la máquina virtual de Java. Estas herramientas las podemos descargar de:

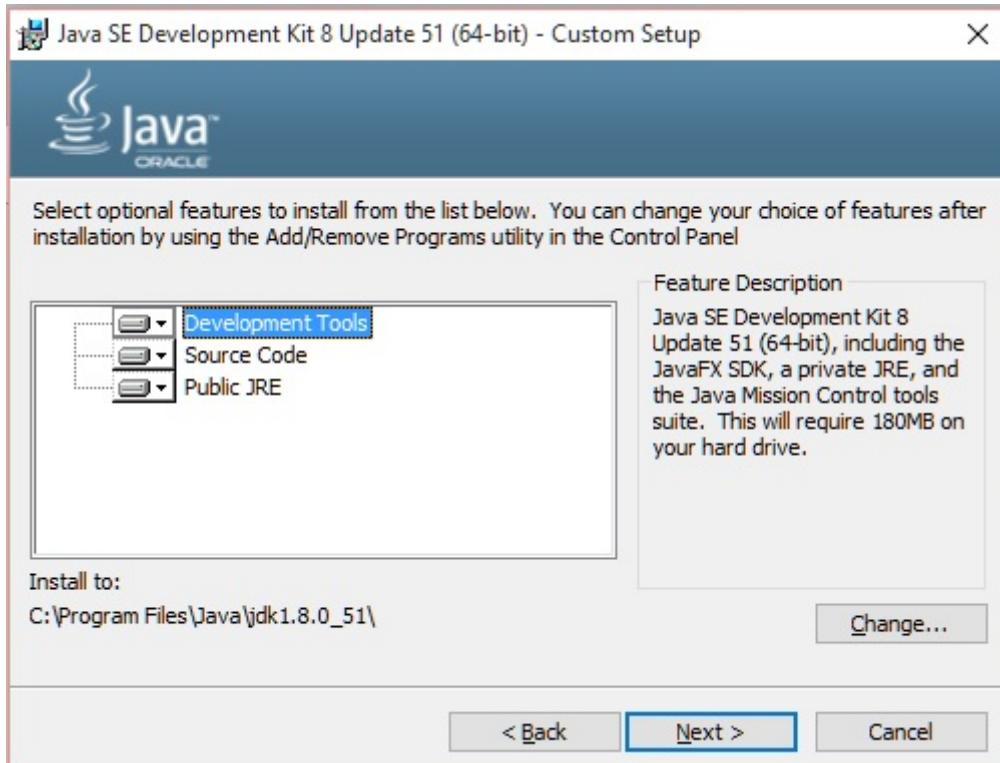
[Java SE Development Kit \(descargar el Windows x64 o si tiene un sistema operativo de 32 bits instale el Windows x86\).](#)

Una vez que tenemos el JDK (Java Development Kit) procedemos a instalarlo:

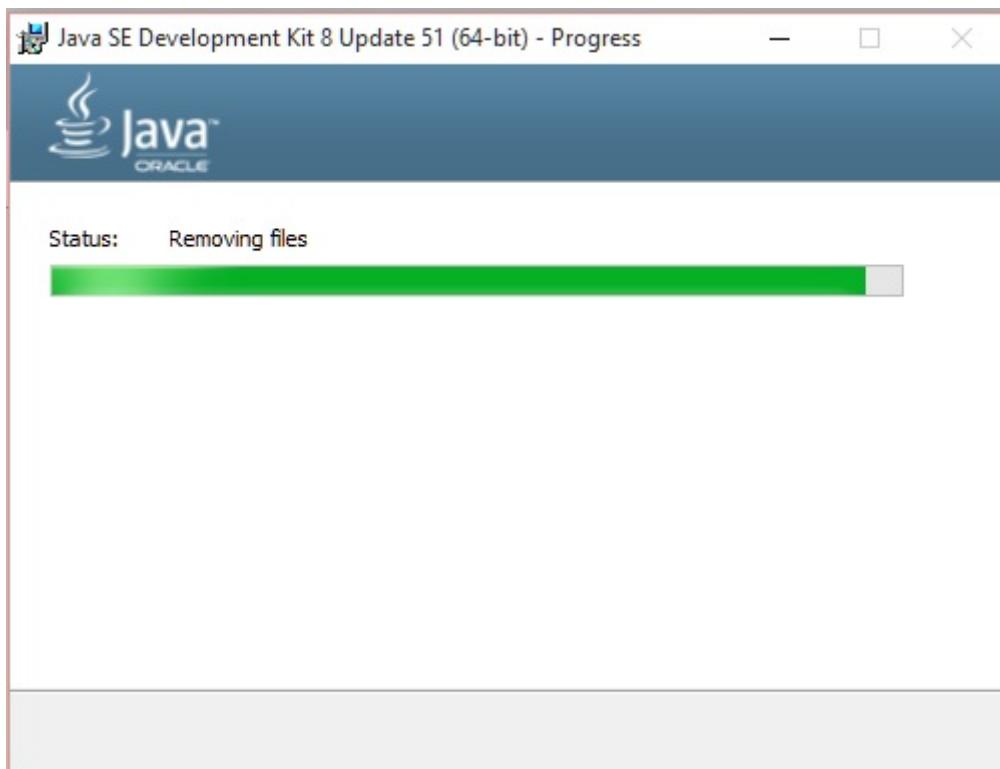


La versión a instalar conviene que sea la última (en este momento disponemos la versión 8)

Presionamos el botón "next". Haremos la instalación por defecto por lo que presionamos el botón next nuevamente:

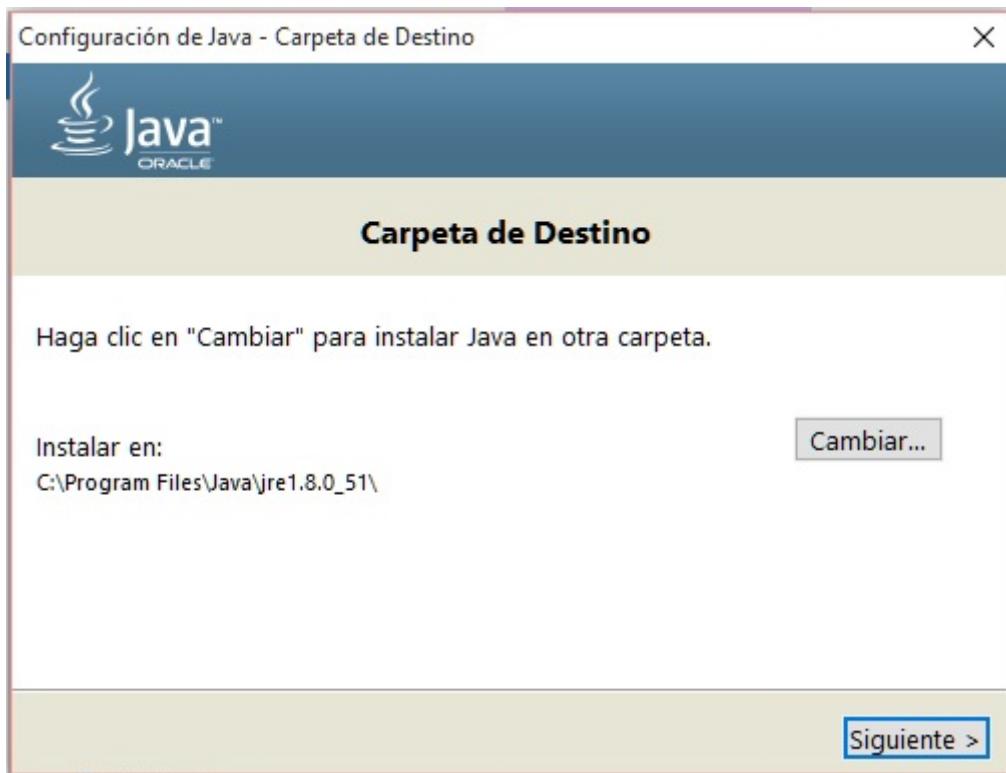


Esperamos unos minutos mientras se procede a la instalación de Java:



Luego aparece un diálogo donde debemos especificar el directorio donde almacenar el JRE y también procedemos a efectuar la instalación por defecto presionando el botón

"next":



Una vez finalizado el proceso de instalación debe aparecer un diálogo similar a este y presionamos el botón Close (por el momento no vamos a instalar "Tutorials, Api documentation etc.):



[Retornar](#)

## 2 - Instalación del editor Eclipse

[Listado completo de tutoriales](#)

En el concepto anterior procedimos a instalar el lenguaje Java (con dichas herramientas podemos ejecutar y compilar programas codificados en java), pero ahora necesitamos instalar el Eclipse que es un editor para codificar los programas (si bien podemos utilizar otros editores, nosotros utilizaremos este para seguir el curso)

### Descarga

Para la descarga del editor Eclipse lo hacemos del sitio:

[Eclipse](#)

La versión a instalar es la Eclipse OXIGEN:



Una vez que descargamos el instalador de Eclipse procedemos a ejecutarlo y debemos elegir el entorno "Eclipse IDE for Java Developers":

eclipseinstaller by Oomph

type filter text

**Eclipse IDE for Java Developers**  
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration

**Eclipse IDE for Java EE Developers**  
Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn, EGit and others.

**Eclipse IDE for C/C++ Developers**  
An IDE for C/C++ developers with Mylyn integration.

**Eclipse IDE for JavaScript and Web Developers**  
The essential tools for any JavaScript developer, including JavaScript, HTML, CSS, XML languages support, Git client, and Mylyn.

**Eclipse IDE for PHP Developers**  
The essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, HTML, CSS and XML.

Seleccionamos la carpeta donde se instalará el Eclipse:



### Eclipse IDE for Java Developers

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration.

Installation Folder

D:\java-oxygen



- create start menu entry
- create desktop shortcut

**INSTALL**

**BACK**

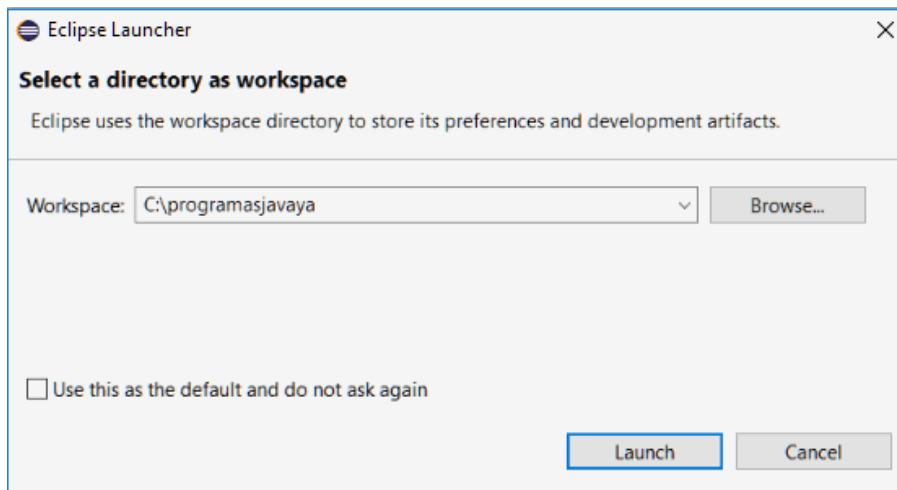
Una vez instalado en forma completa el "Eclipse IDE for Java Developers" nos dirigimos a la carpeta donde se instalaron los archivos y procedemos a ejecutar el programa `eclipse.exe` (podemos ingresar desde el acceso directo que se crea en el escritorio)

Primero aparece un mensaje de inicio del Eclipse:



Luego la primera vez que ejecutemos el editor Eclipse aparece un diálogo para seleccionar la carpeta donde se almacenarán los programas que desarrollaremos (podemos crear una carpeta donde almacenaremos todos los proyectos que desarrollaremos en el curso, si indicamos una carpeta que no existe el mismo Eclipse la

crea):



Luego de configurar la carpeta donde se crearán los proyecto aparece el editor con una pantalla de presentación (Welcome):

File Edit Navigate Search Project Run Window Help

Welcome

# eclipse

Welcome to the Eclipse IDE for Java Developers

Workbench

**Review IDE configuration settings**  
Review the IDE's most fiercely contested preferences

**Create a Hello World application**  
A guided walkthrough to create the famous Hello World in Eclipse

**Create a new Java project**  
Create a new Java Eclipse project

**Checkout projects from Git**  
Checkout Eclipse projects hosted in a Git repository

**Import existing projects**  
Import existing Eclipse projects from the filesystem or archive

**Launch the Eclipse Marketplace**  
Enhance your IDE with additional plugins

**Open an existing file**  
Open a file from the filesystem

**Overview**  
Get an overview of the features

**Tutorials**  
Go through tutorials

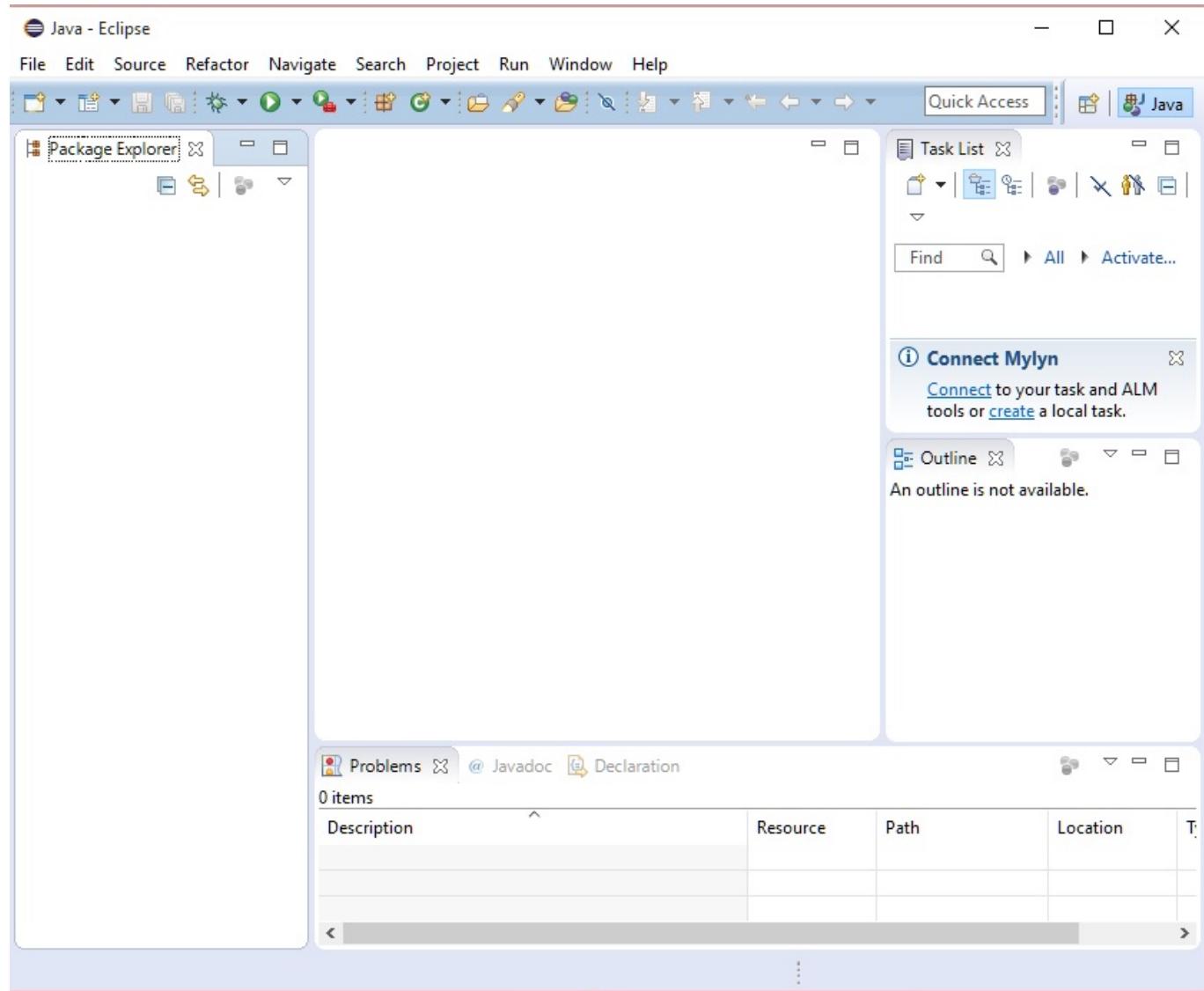
**Samples**  
Try out the samples

**What's New**  
Find out what is new

Always show Welcome at start up

Esta ventana de bienvenida la podemos cerrar seleccionando el ícono: "Workbench", con lo que aparece el entorno de trabajo del Eclipse (si queremos nuevamente ver la ventana de bienvenida podemos activarla desde el menú de opciones: Help -> Welcome)

El entorno de trabajo del Eclipse es:



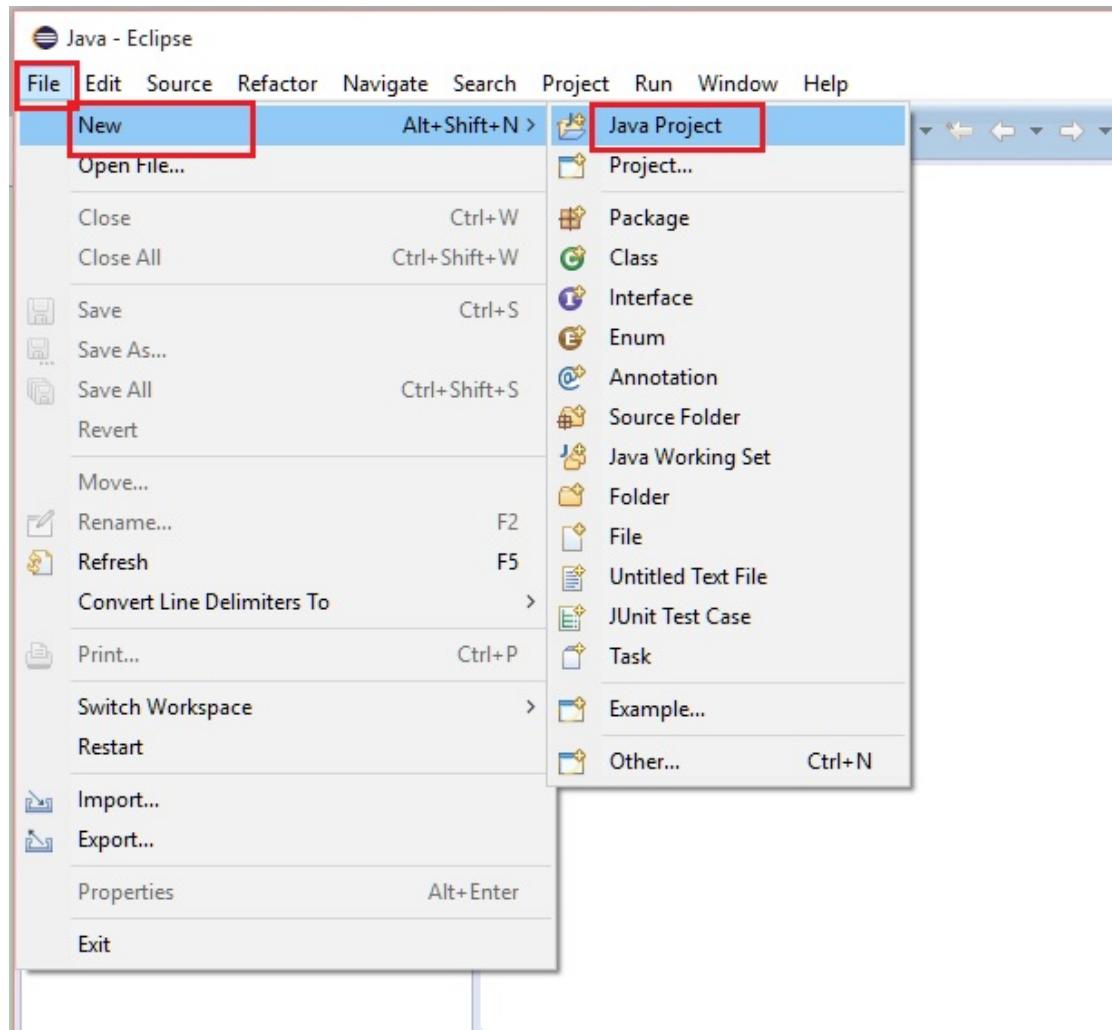
[Retornar](#)

# 3 - Pasos para crear un programa con Eclipse

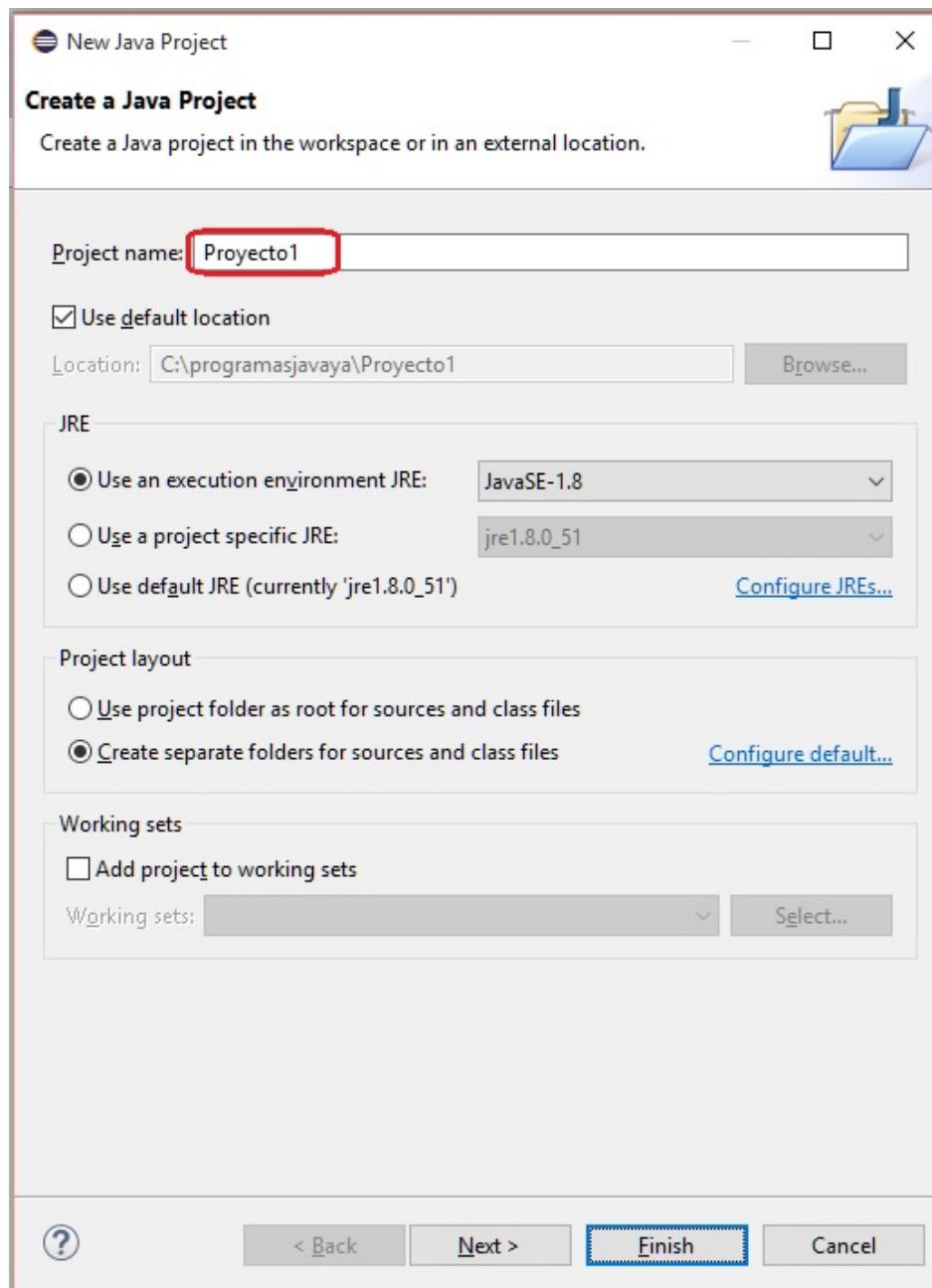
[Listado completo de tutoriales](#)

EI Eclipse es un entorno de trabajo profesional, por lo que en un principio puede parecer complejo el desarrollo de nuestros primeros programas.

Todo programa en Eclipse requiere la creación de un "Proyecto", para esto debemos seleccionar desde el menú de opciones:

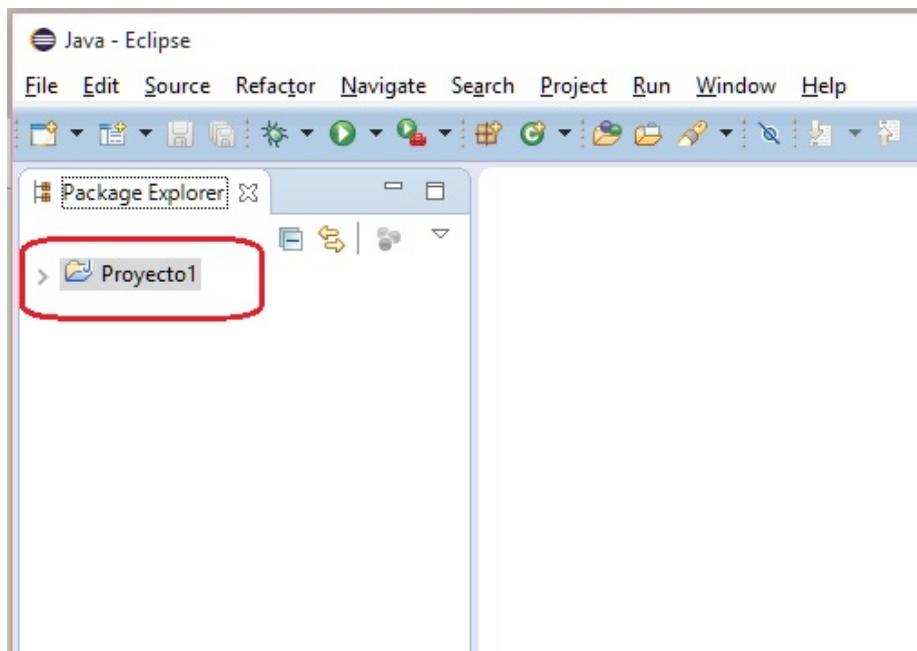


Ahora aparece el diálogo donde debemos definir el nombre de nuestro proyecto:

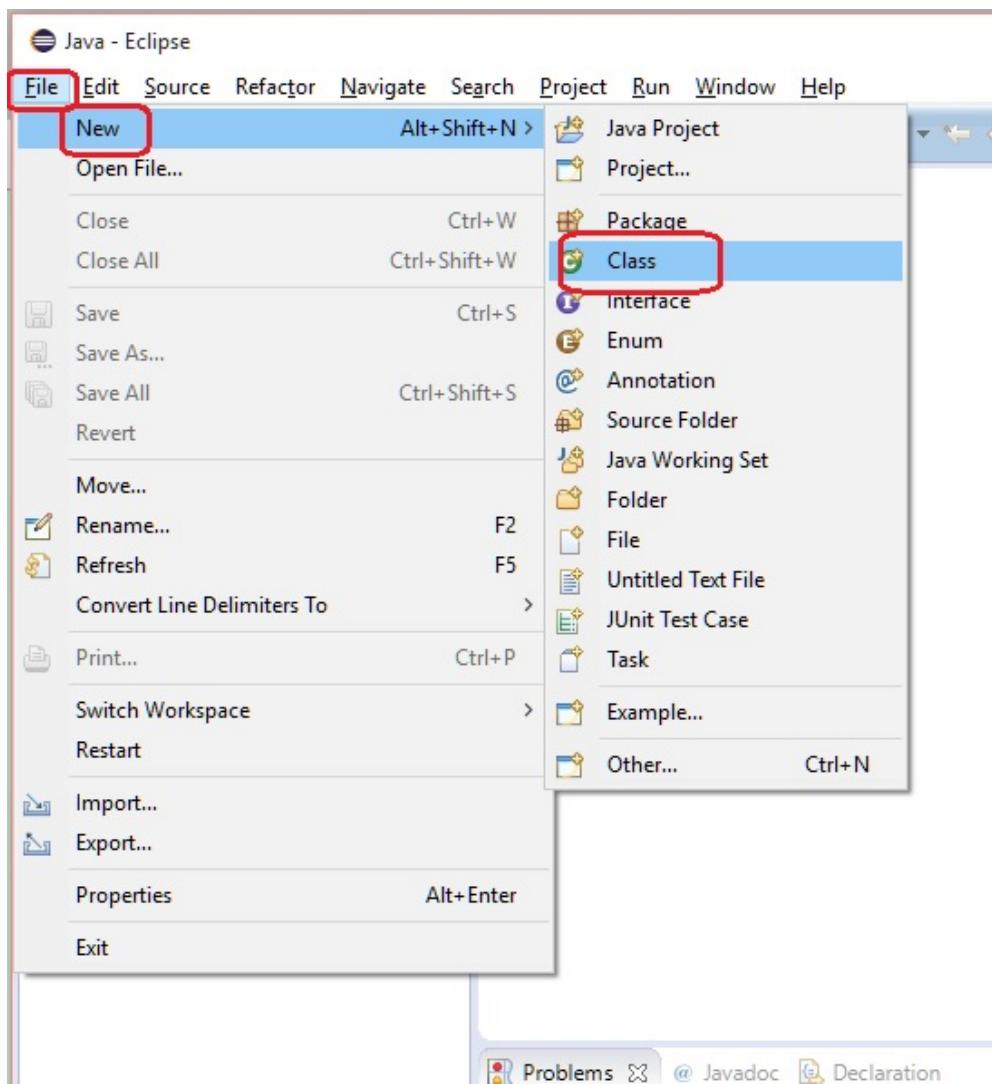


En el campo de texto "Project Name" ingresamos como nombre: Proyecto1 y dejamos todas las otras opciones del diálogo con los valores por defecto. Presionamos el botón "Finish".

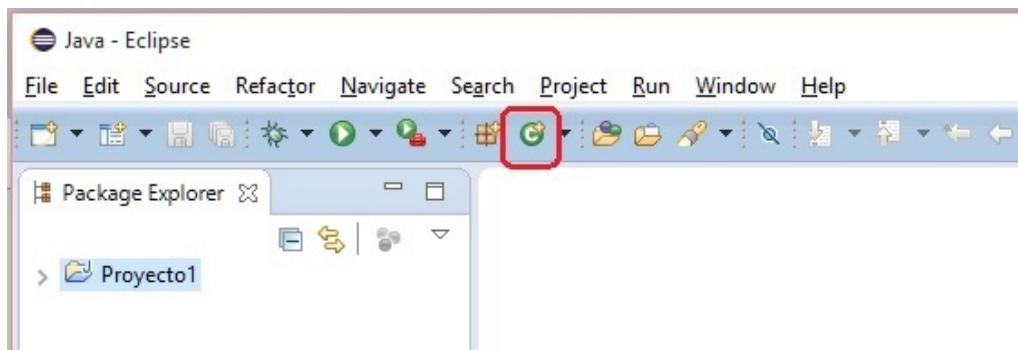
Ahora en la ventana de "Package Explorer" aparece el proyecto que acabamos de crear:



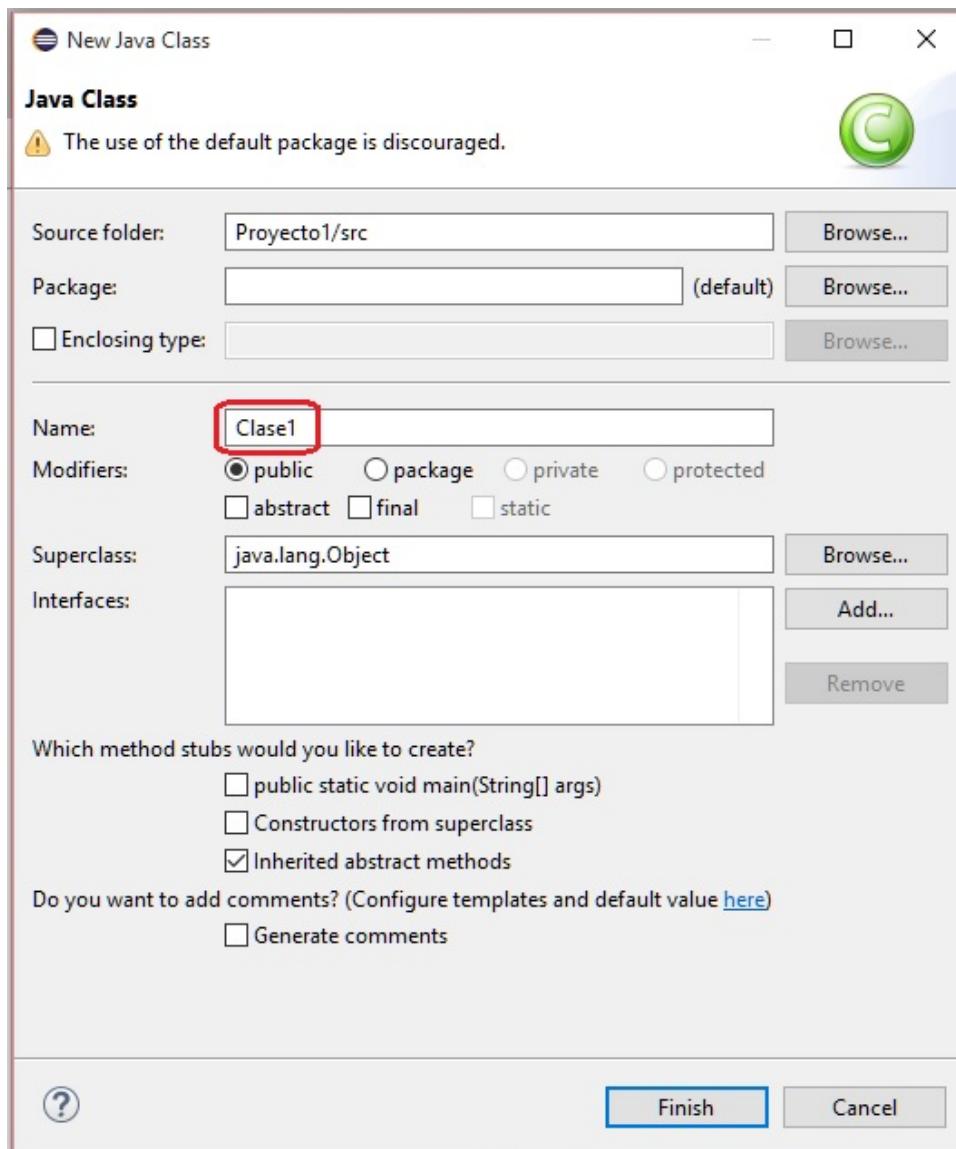
Como segundo paso veremos que todo programa en Java requiere como mínimo una clase. Para crear una clase debemos seleccionar desde el menú de opciones:



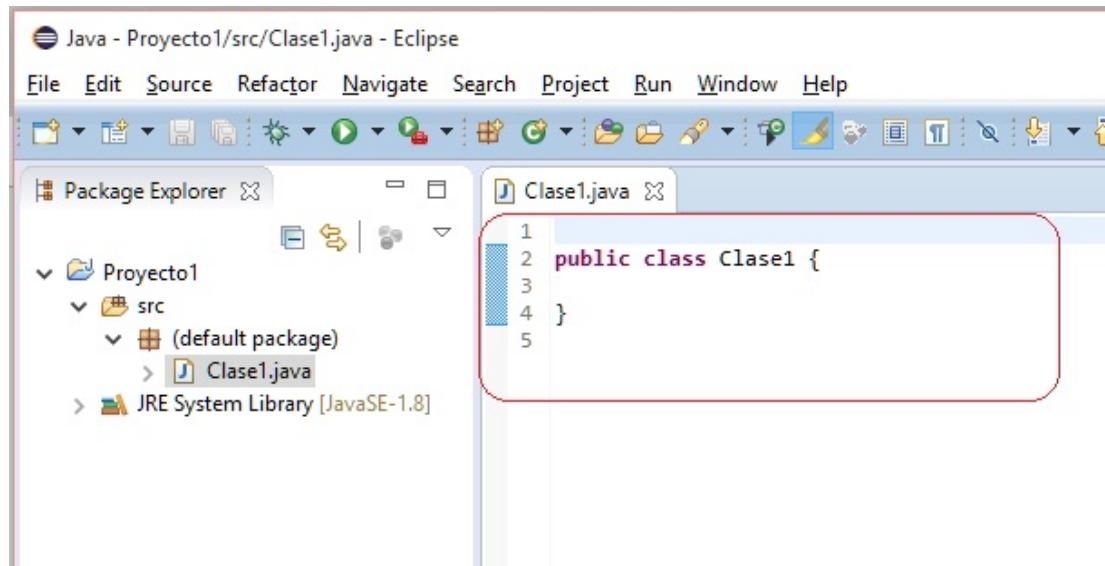
O desde la barra de íconos del Eclipse:



En el diálogo que aparece debemos definir el nombre de la clase (en nuestro primer ejemplo la llamaremos Clase1 (con mayúscula la letra C), luego veremos que es importante definir un nombre que represente al objetivo de la misma), los otros datos del diálogo los dejamos con los valores por defecto:



Luego de presionar el botón "Finish" tenemos el archivo donde podemos codificar nuestro primer programa:



Más adelante veremos los archivos que se crean en un proyecto, ahora nos dedicaremos a codificar nuestro primer programa. En la ventana de edición ya tenemos el esqueleto de una clase de Java que el entorno Eclipse nos creó automáticamente.

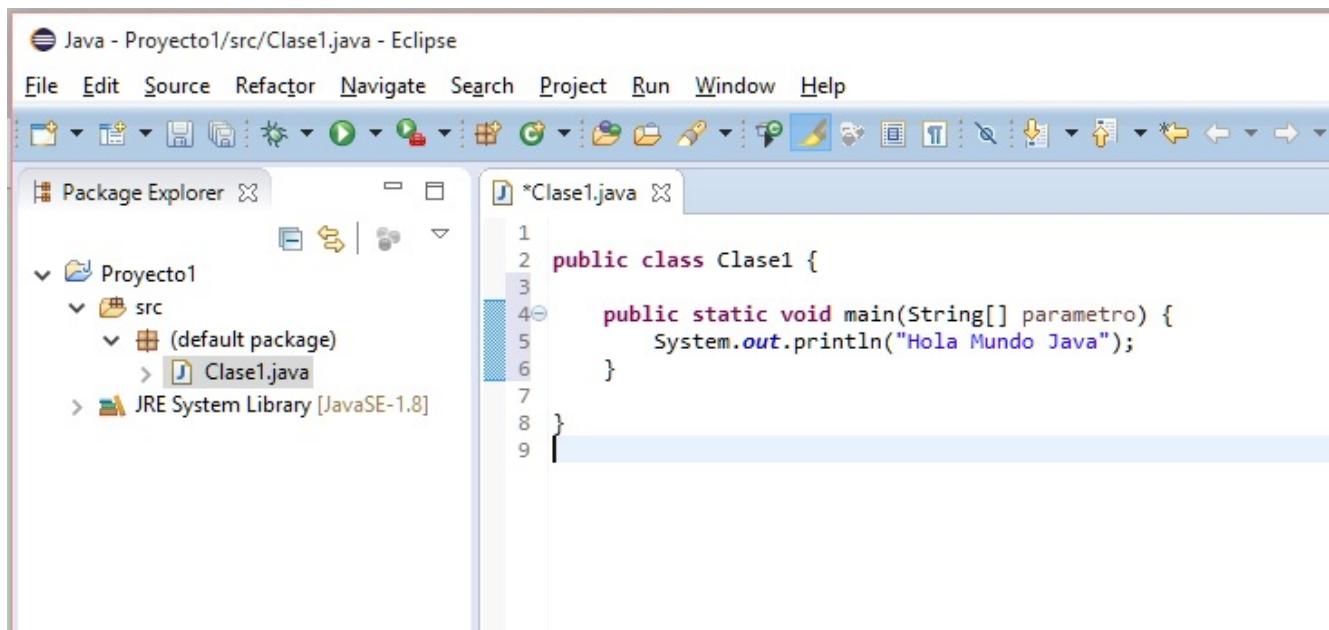
```
public class Clase1 {  
}  
}
```

Todo programa en Java debe definir la función main. Esta función la debemos codificar dentro de la clase: "Clase1".

Procedemos a tipear lo siguiente:

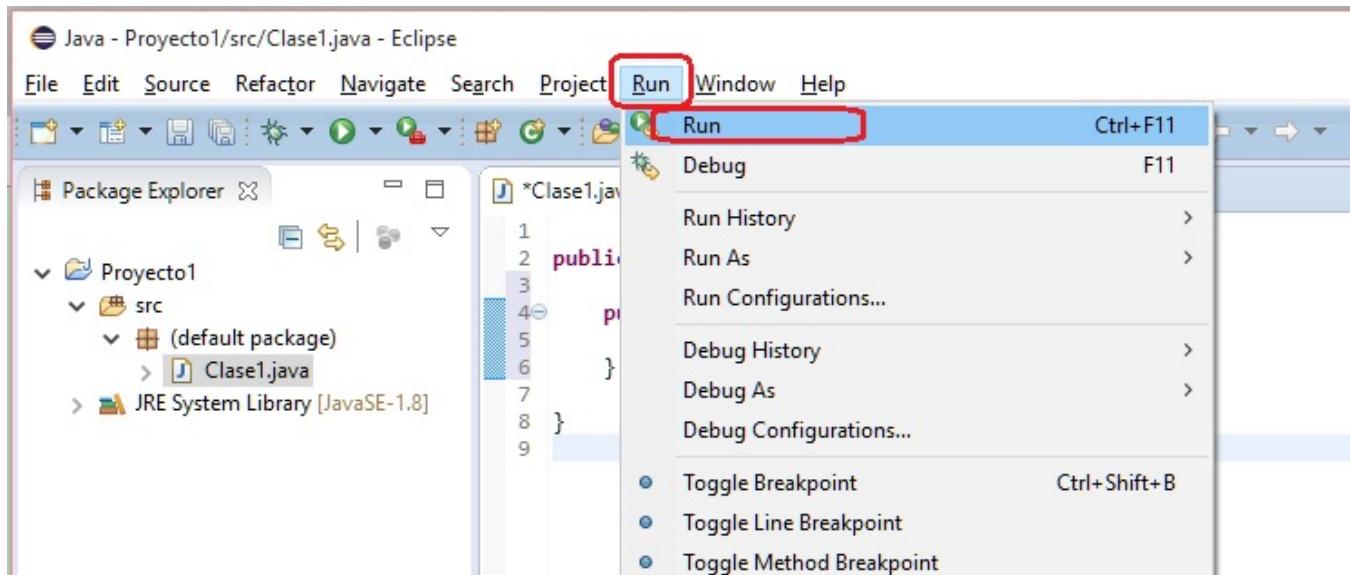
```
public class Clase1 {  
  
    public static void main(String[] parametro) {  
        System.out.println("Hola Mundo Java");  
    }  
  
}
```

Es decir tenemos codificado en el entorno del Eclipse nuestro primer programa:

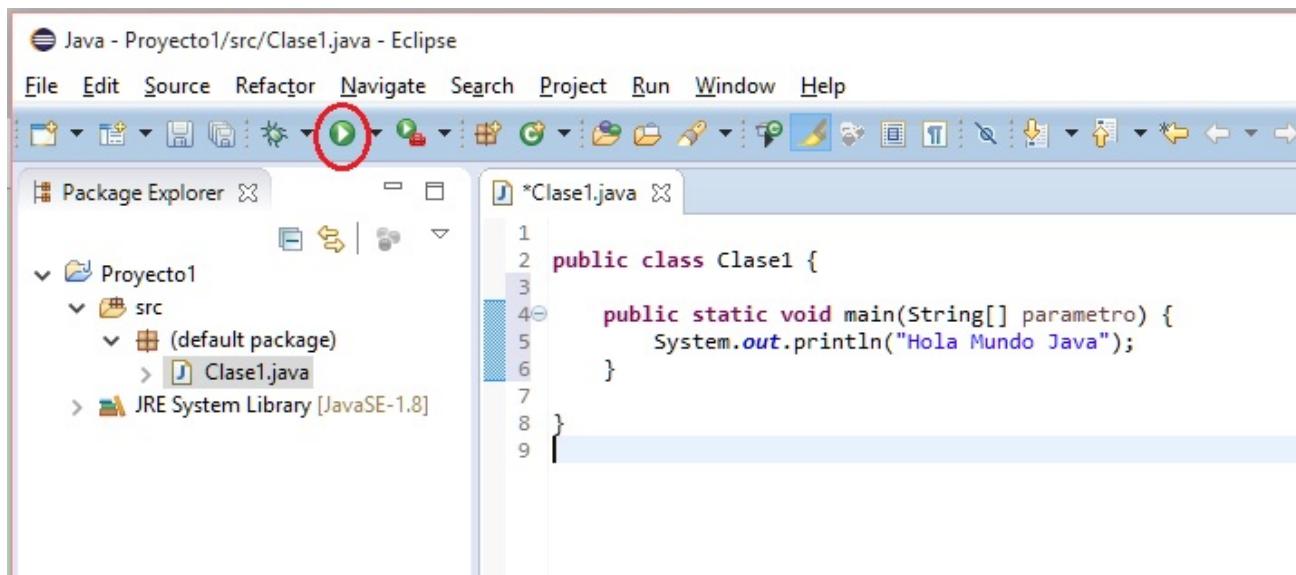


```
Java - Proyecto1/src/Clase1.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer *Clase1.java
1
2 public class Clase1 {
3
4     public static void main(String[] parametro) {
5         System.out.println("Hola Mundo Java");
6     }
7
8 }
9
```

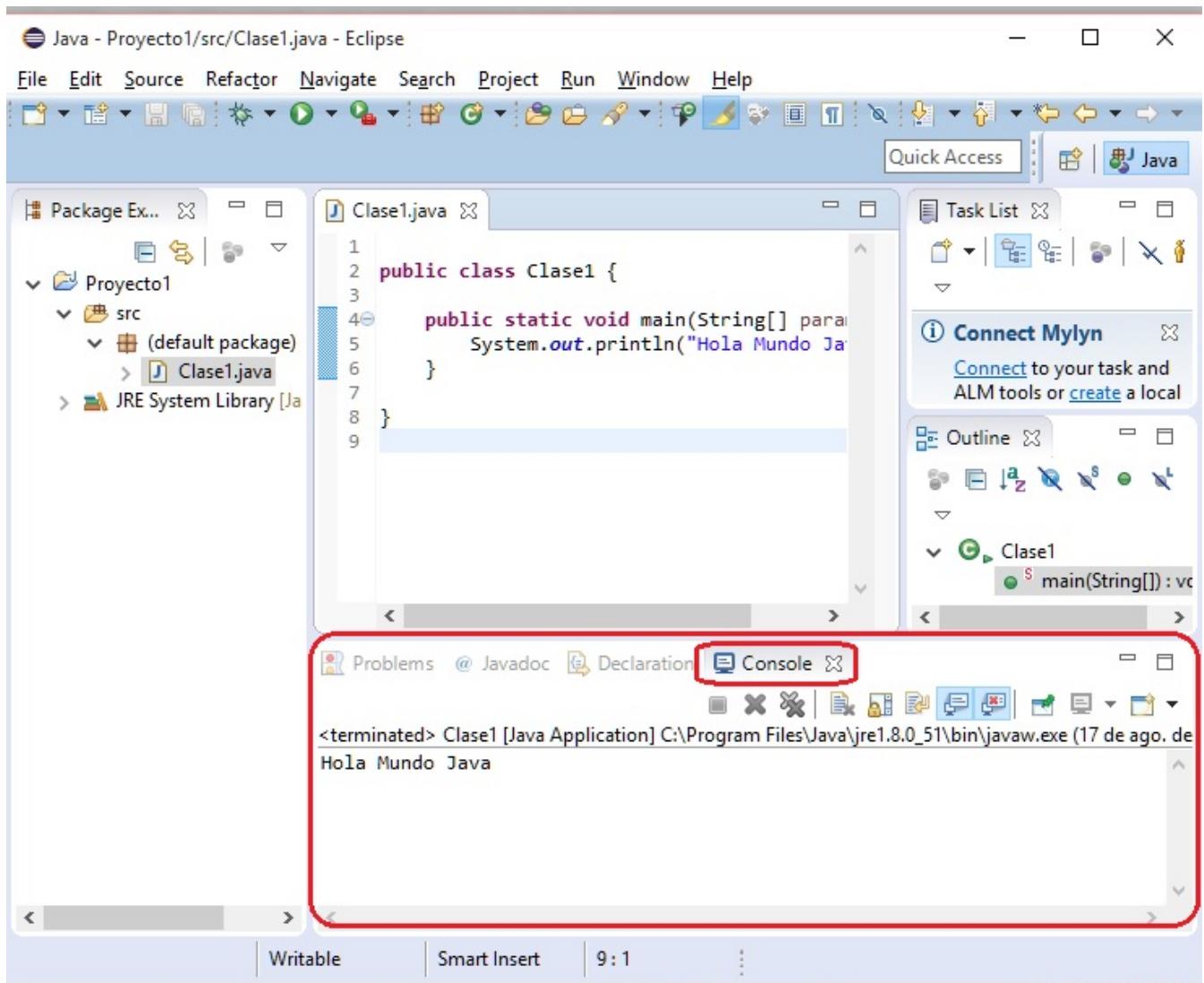
Como último paso debemos compilar y ejecutar el programa, esto lo podemos hacer desde el menú de opciones:



O desde la barra de íconos del Eclipse:



Si no hay errores de codificación debemos ver el resultado de la ejecución en una ventana del Eclipse llamada "Console" que aparece en la parte inferior (puede aparecer un diálogo pidiendo que grabemos el archivo, el cual confirmamos):



Lo más importante es que quede claro los pasos que debemos dar para crear un proyecto en Java. El objetivo de una clase, la función main etc. los veremos a lo largo de este curso.

[Retornar](#)

## 4 - Objetivos del curso y nociones básicas indispensables

[Listado completo de tutoriales](#)

El curso está ideado para ser desarrollado por una persona que no conoce nada de programación y se utilice Java como primer lenguaje.

El objetivo fundamental de este tutorial es permitir que el estudiante pueda resolver problemas de distinta índole (matemáticos, administrativos, gráficos, contables etc.) empleando como herramienta la computadora.

Hay que tener en cuenta que para llegar a ser programador se debe recorrer un largo camino donde cada tema es fundamental para conceptos futuros. Es importante no dejar temas sin entender y relacionar.

La programación a diferencia de otras materias como podría ser la historia requiere un estudio metódico y ordenado (en historia se puede estudiar la edad media sin tener grandes conocimientos de la edad antigua)

La programación es una actividad nueva para el estudiante, no hay en los estudios primarios y secundarios una materia parecida.

Es bueno tenerse paciencia cuando los problemas no se resuelven por completo, pero es de fundamental importancia dedicar tiempo al análisis individual de los problemas.

### Qué es un programa?

Programa: Conjunto de instrucciones que entiende un ordenador para realizar una actividad.

Todo programa tiene un objetivo bien definido: un procesador de texto es un programa que permite cargar, modificar e imprimir textos, un programa de ajedrez permite jugar al ajedrez contra el ordenador u otro contrincante humano.

La actividad fundamental del programador es resolver problemas empleando el ordenador como herramienta fundamental.

Para la resolución de un problema hay que plantear un algoritmo.

Algoritmo: Son los pasos a seguir para resolver un problema.

### Diagrama de flujo

Un diagrama de flujo es la representación gráfica de un ALGORITMO.

Los símbolos gráficos a utilizar para el planteo de diagramas de flujo son:



Inicio y fin del diagrama.



Entrada de Datos

(Vamos a considerar que las entradas de datos se realizan siempre por el teclado de la computadora).

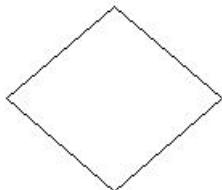


Salida de Datos

(Vamos a considerar que las salidas de datos se realizan siempre por la pantalla de la computadora).



Operación



Condición

Estos son los elementos esenciales que intervienen en el desarrollo de un diagrama de flujo.

## Planteo de un problema utilizando diagramas de flujo.

Para plantear un diagrama de flujo debemos tener muy en claro el problema a resolver.

Ejemplo : Calcular el sueldo mensual de un operario conociendo la cantidad de horas trabajadas y el pago por hora.

Podemos identificar:

Datos conocidos:

Horas trabajadas en el mes.

Pago por hora.

Proceso:

Cálculo del sueldo multiplicando la cantidad de horas por el pago por hora.

Información resultante:

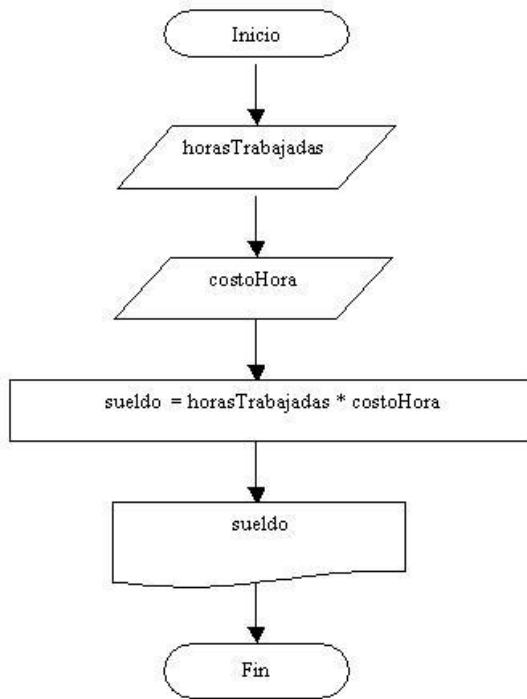
Sueldo mensual.

Si hacemos un análisis todo problema está constituido por:

- Datos conocidos: Datos con los que se cuenta al plantear el problema.
- Proceso: Operaciones a realizar con los datos conocidos.
- Información resultante: Es la información que resuelve el problema.

Esta forma de expresar un problema identificando sus datos conocidos, procesos e información resultante puede llegar a ser engorrosa para problemas complejos donde hay muchos datos conocidos y procesos. Es por eso que resulta mucho más efectivo representar los pasos para la resolución del problema mediante un

diagrama de flujo.



Resulta mucho más fácil entender un gráfico que un texto.

El diagrama de flujo nos identifica claramente los datos de entrada, operaciones y datos de salida.

En el ejemplo tenemos dos datos de entrada: horasTrabajadas y costoHora, a las entradas las representamos con un paralelogramo y hacemos un paralelogramo por cada dato de entrada.

La operación se representa con un rectángulo, debemos hacer un rectángulo por cada operación. A la salida la representamos con la hoja rota.

El diagrama de flujo nos da una idea del orden de ejecución de las actividades en el tiempo. Primero cargamos los datos de entrada, luego hacemos las operaciones necesarias y por último mostramos los resultados.

## Codificación del problema con el lenguaje Java.

No debemos perder de vista que el fin último es realizar un programa de computación que permita automatizar una actividad para que muchos procesos sean desarrollados por la computadora.

El diagrama de flujo es un paso intermedio para poder ser interpretado por la computadora.

El paso siguiente es la codificación del diagrama de flujo en un lenguaje de computación, en nuestro caso emplearemos el lenguaje Java.

Lenguaje de computación: Conjunto de instrucciones que son interpretadas por una computadora para realizar operaciones, mostrar datos por pantalla, sacar listados por impresora, entrar datos por teclado, etc.

## Conceptos básicos para codificar un programa.

Variable: Es un depósito donde hay un valor. Consta de un nombre y pertenece a un tipo.

Para el ejemplo planteado la variable horasTrabajadas almacena la cantidad de horas trabajadas por el operario. La variable valorHora almacena el precio de una hora de trabajo. La variable sueldo almacena el sueldo a abonar al operario.

En el ejemplo tenemos tres variables.

Tipos de variable:

Una variable puede almacenar:

- Valores Enteros (100, 260, etc.)
- Valores Reales (1.24, 2.90, 5.00, etc.)
- Cadenas de caracteres ("Juan", "Compras", "Listado", etc.)

Elección del nombre de una variable:

Debemos elegir nombres de variables representativas. En el ejemplo el nombre horasTrabajadas es lo suficientemente claro para darnos una idea acabada sobre su contenido. Podemos darle otros buenos nombres. Otros no son tan representativos, por ejemplo hTr. Posiblemente cuando estemos resolviendo un problema dicho nombre nos recuerde que almacenamos las horas trabajadas por el operario pero cuando pase el tiempo y leamos el diagrama probablemente no recordemos ni entendamos qué significa hTr.

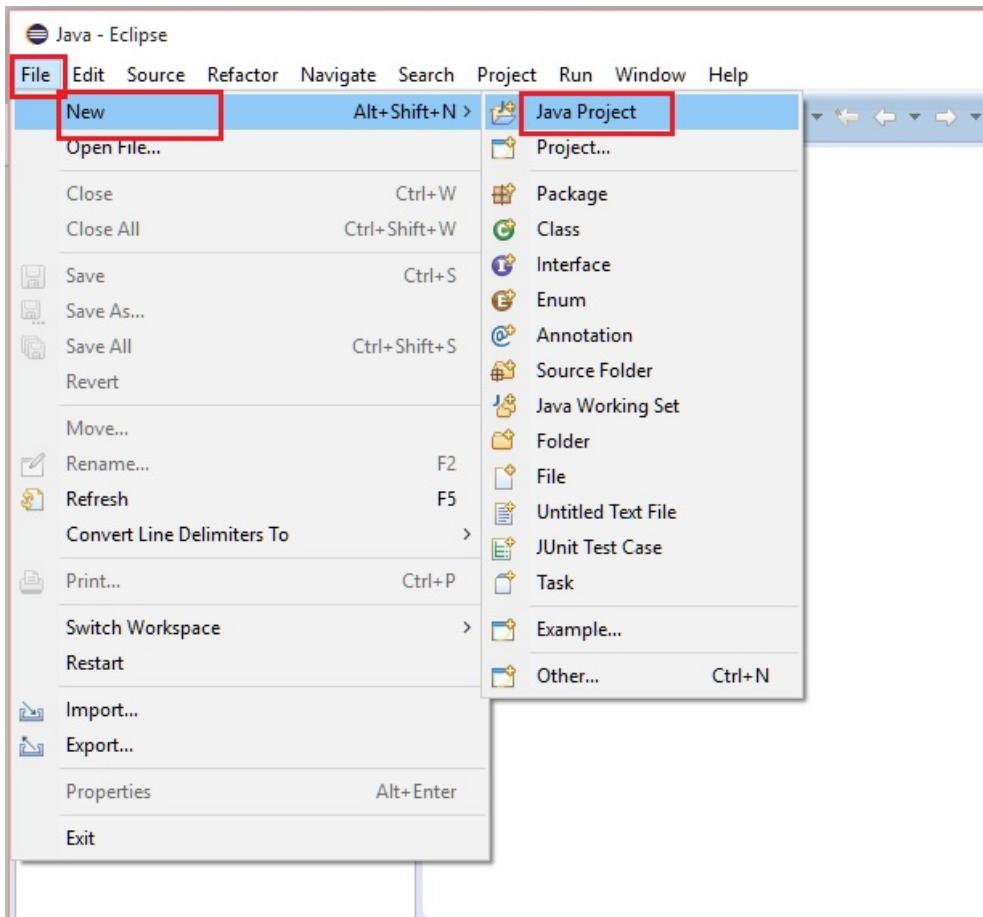
### **Consideraciones a tener en cuenta en cada proyecto.**

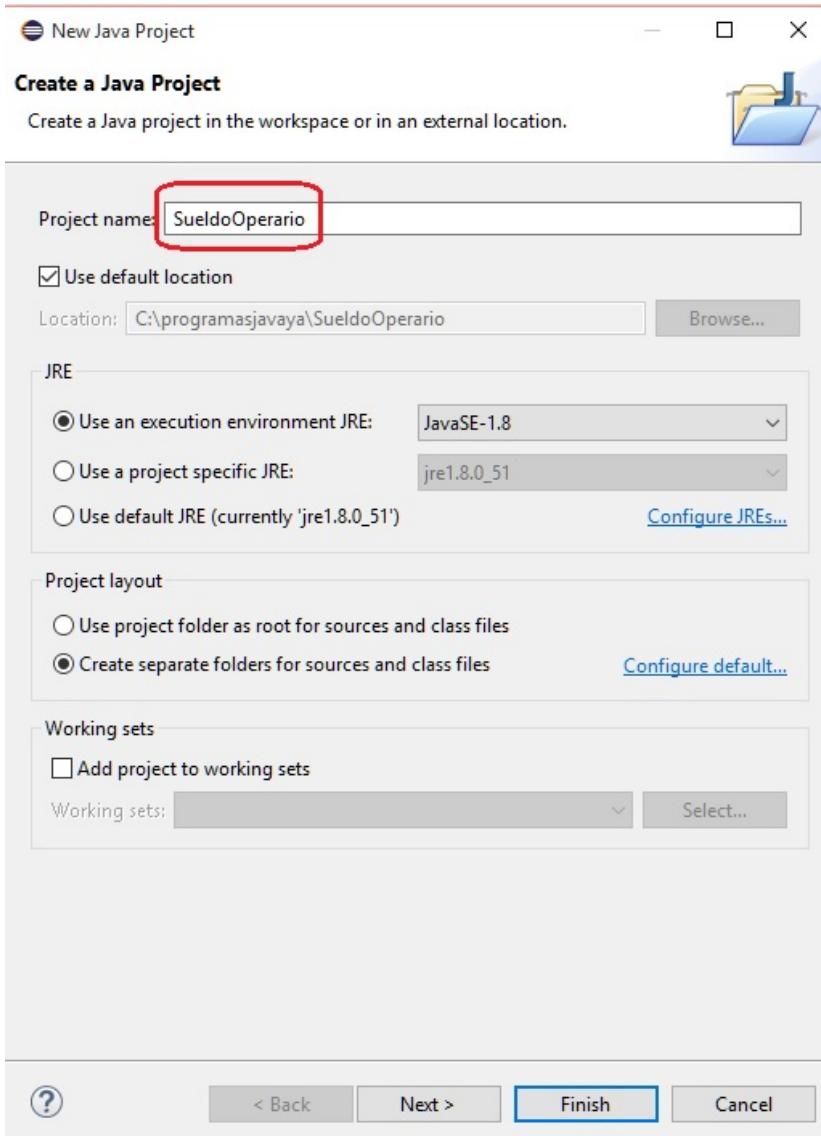
Hay que tener en cuenta que el entorno de programación "Eclipse" no a sido desarrollado pensando en un principiante de la programación. Lo mismo ocurre con el propio lenguaje Java, es decir su origen no tiene como principio el aprendizaje de la programación. Debido a estos dos puntos veremos que a medida que avanzamos con el tutorial muchos conceptos que iremos dejando pendientes se irán aclarando.

Codificaremos el problema propuesto para repasar los pasos para la creación de un proyecto en Eclipse, creación de la clase principal, definición de la función main y el posterior desarrollo del algoritmo del problema.

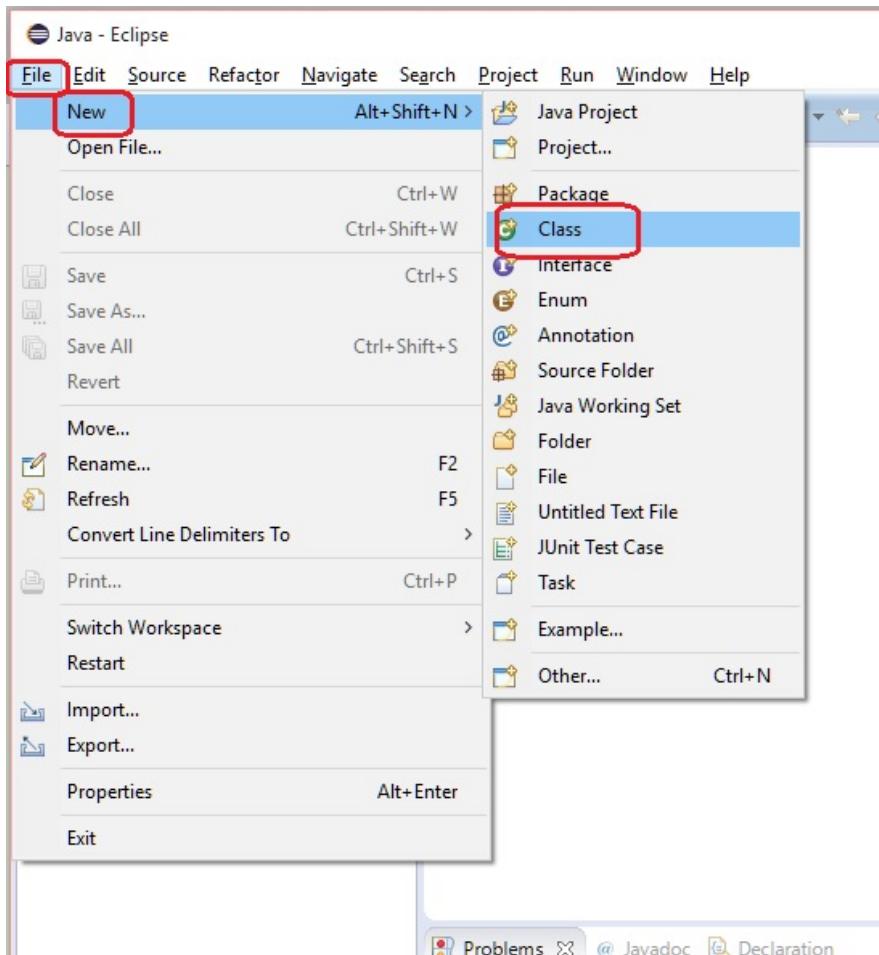
### **Pasos.**

1 - Creación del proyecto (tema visto anteriormente). Podemos asignarle como nombre: SueldoOperario (normalmente uno busca un nombre representativo al programa que desarrolla)





2 - Creación de la clase. Definiremos como nombre el mismo que le asignamos al proyecto (esto no es obligatorio como veremos más adelante un proyecto puede contener varias clases)  
Es decir disponemos como nombre de la clase: SueldoOperario.



Inicializamos el campo que solicita el "Name" con "SueldoOperario".

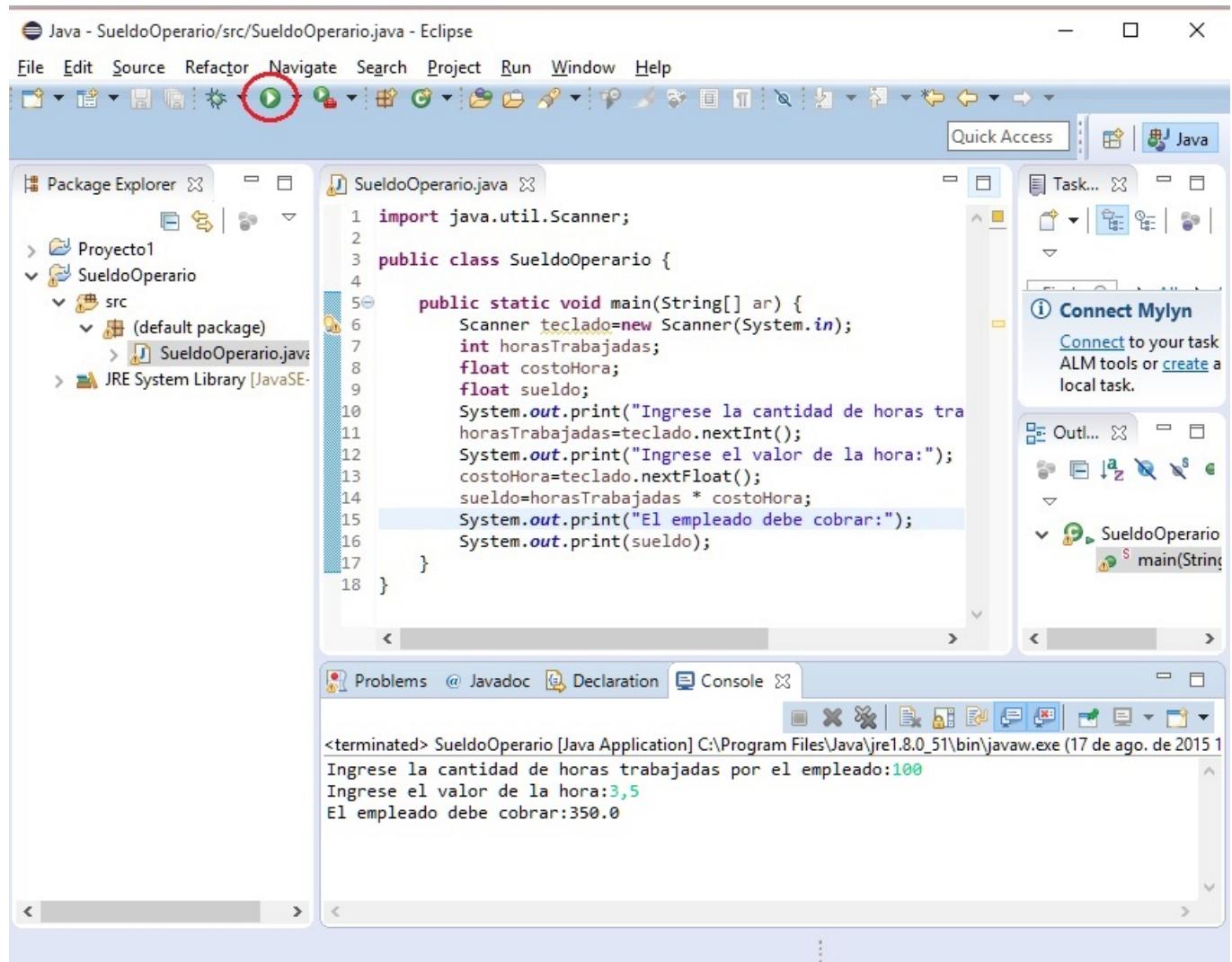
3 - Codificamos el algoritmo en la clase: SueldoOperario.

```
import java.util.Scanner;

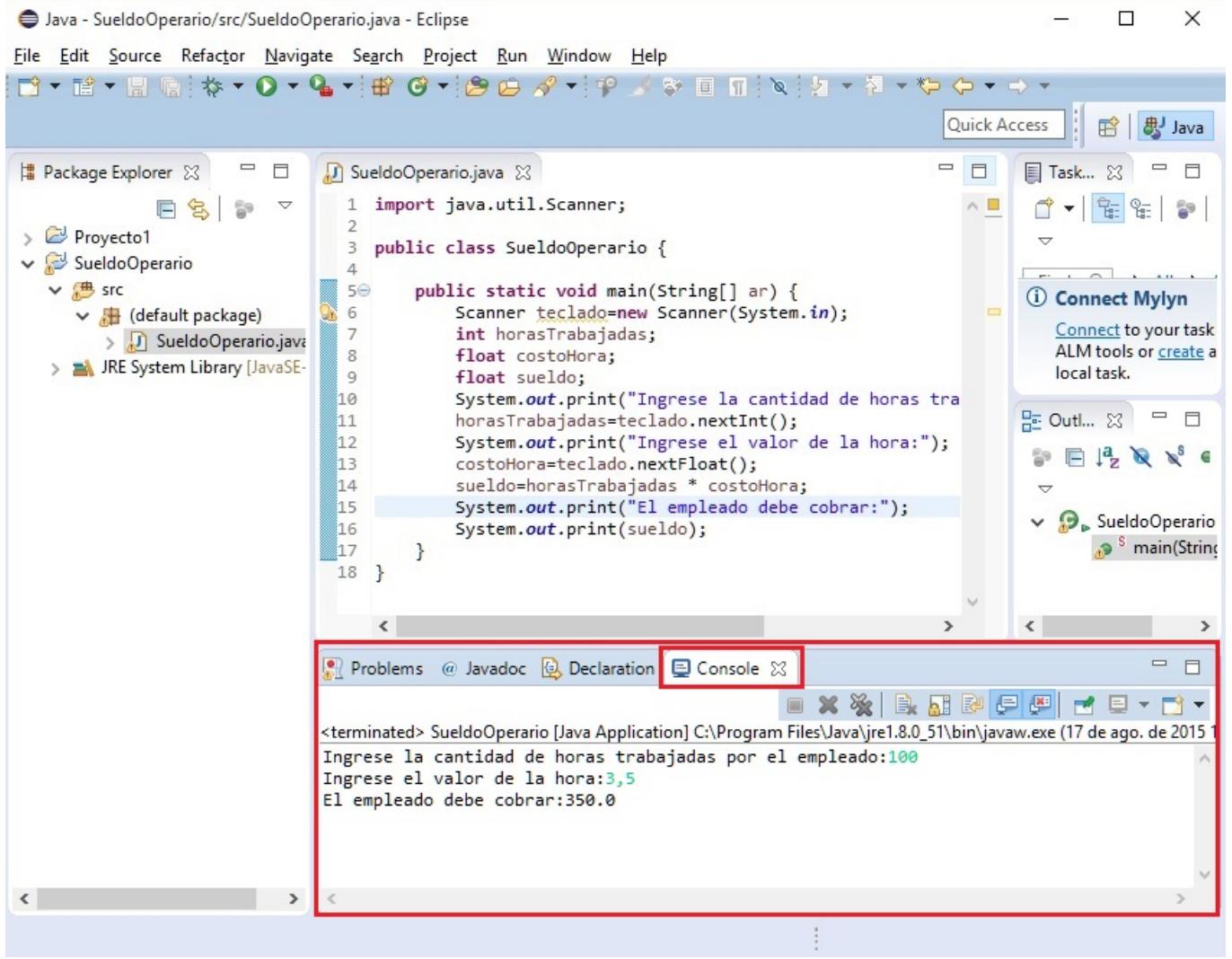
public class SueldoOperario {

    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int horasTrabajadas;
        float costoHora;
        float sueldo;
        System.out.print("Ingrese la cantidad de horas trabajadas por");
        horasTrabajadas=teclado.nextInt();
        System.out.print("Ingrese el valor de la hora:");
        costoHora=teclado.nextFloat();
        sueldo=horasTrabajadas * costoHora;
        System.out.print("El empleado debe cobrar:");
        System.out.print(sueldo);
    }
}
```

4 - Ejecutamos el programa:



5 - Si no hay errores sintácticos procedemos a activar la ventana de la "Console" con el mouse y cargamos por teclado los dos datos que se solicitan (la cantidad de horas trabajadas y el precio de la hora):



Estos cinco pasos fundamentales debemos llevar a cabo cada vez que desarrollemos un nuevo programa en Java.

## Explicación.

Ahora veremos una explicación de varias partes de nuestro programa y otras partes quedarán pendientes para más adelante ya que en este momento difícilmente se entiendan.

Conceptos que quedarán pendientes para explicar:

1. Concepto de una clase. Veremos más adelante que en Java todo debe estar contenido en clases, por lo que hasta el problema más elemental debe estar contenido en una clase. Para declarar una clase utilizamos la sintaxis:

```
public class SueldoOperario {  
}
```

El nombre de la clase no puede tener espacios en blanco, comienza con una letra mayúscula y en caso de estar constituida por dos o más palabras el primer carácter va en mayúsculas, no puede empezar con un número, pero si puede llevar números a partir del segundo carácter. Toda clase debe tener una llave de apertura y una llave de cierre.

2. Todo programa constituido por una única clase debe tener definida la función main:

```
public static void main(String[] ar) {  
}
```

La función main es la primera que se ejecuta y debe llevar la sintaxis indicada anteriormente (más adelante veremos que significa el parámetro ar, las palabras claves public, static y void. La función main tiene una llave de apertura y una llave de cierre (similar a la clase). La función main debe estar contenida en la clase.

3. Cuando se requieren utilizar otras clases debemos importarlas previo a la declaración de la clase (en nuestro problema utilizamos la clase Scanner que se encuentra en el paquete java.util por lo que la importamos con la siguiente sintaxis:

```
import java.util.Scanner;
```

En la main creamos un objeto de la clase Scanner que nos permitirá ingresar por teclado los valores:

```
Scanner teclado=new Scanner(System.in);
```

Conceptos que deben quedar claros:

1. Por el momento haremos todo el algoritmo dentro de la función main. Es decir el resto siempre será lo mismo (declarar un proyecto, declarar una clase, definir una función main)
2. Si observamos el diagrama de flujos vemos que debemos definir tres variables: (horasTrabajadas, costoHora,sueldo), aquí es donde debemos definir que tipos de datos se almacenarán en las mismas. La cantidad de horas normalmente será un valor entero (ej. 100 - 150 - 230 etc.), pero el costo de la hora es muy común que sea un valor real (ej. 5,35 - 7,50 etc.) y como el sueldo resulta de multiplicar las horas trabajadas por el costo por hora el mismo deberá ser real.

La definición de las variables la hacemos en la main:

```
int horasTrabajadas;  
float costoHora;  
float sueldo;
```

Utilizamos la palabra clave int para definir variables enteras (en Java las palabras claves deben ir obligatoriamente en minúsculas, sino se produce un error sintáctico) Luego de la palabra clave debemos indicar el nombre de la variable, por ejemplo: horasTrabajadas (se propone que el nombre de la variable comience con minúsculas y en caso de estar constituida por dos palabras o más a partir de la segunda palabra el primer carácter se especifique con mayúsculas (un nombre de variable no puede tener espacios en blanco, empezar con un número, ni tampoco utilizar caracteres especiales))

Debemos buscar siempre nombres de variables que nos indiquen que almacenan (no es conveniente llamar a nombres de variables con letras individuales)

3. Para mostrar mensajes en la "Console" utilizamos la siguiente sintaxis:

```
System.out.print("Ingrese la cantidad de horas trabajadas por el empleo
```

Con esta sintaxis todo lo que se encuentra contenido entre comillas aparecerá exactamente en la ventana de la "Console".

Si disponemos una variable:

```
System.out.print(sueldo);
```

Aparecerá el contenido de la variable. Es decir el valor almacenado en la variable sueldo y no el mensaje "sueldo".

4. Para hacer la entrada de datos por teclado en Java se complica. Utilizaremos una clase llamada Scanner que nos facilita el ingreso de datos. Por eso tuvimos que importar la clase Scanner que se encuentra en el paquete java.util en la primer línea de nuestro programa.

En la función main debemos crear un objeto de la clase Scanner con la siguiente sintaxis:

```
Scanner teclado=new Scanner(System.in);
```

Luego para cargar valores enteros por teclado debemos implementar la siguiente sintaxis:

```
horasTrabajadas=teclado.nextInt();
```

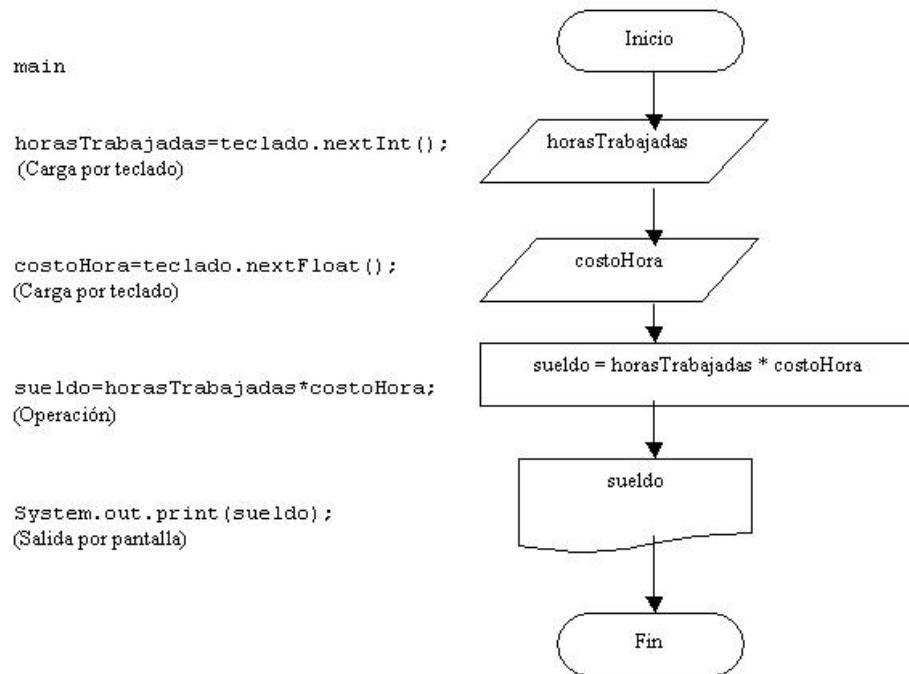
Pero si el dato a cargar se trata de un valor float luego debemos utilizar la siguiente sintaxis:

```
costoHora=teclado.nextFloat();
```

5. Las operaciones que indicamos en el diagrama de flujo mediante la figura rectángulo la codificamos tal cual:

```
sueldo=horasTrabajadas * costoHora;
```

Podemos ver una relación entre las instrucciones que debemos utilizar para cada símbolo del diagrama de flujo:



En el diagrama de flujo no indicamos la definición de variables:

```
int horasTrabajadas;  
float costoHora;  
float sueldo;
```

No indicamos la creación del objeto de la clase Scanner:

```
Scanner teclado = new Scanner(System.in);
```

No representamos con símbolos los mensajes a mostrar previo a la carga de datos por teclado:

```
System.out.print("Ingrese la cantidad de horas trabajadas por el empleado:");
```

Como hemos visto hasta ahora hay muchas partes de nuestro código que no entendemos pero son indispensables para la implementación de nuestros programas, a medida que avancemos con el curso muchos de estos conceptos se irán aclarando.

[Retornar](#)

# 5 - Errores sintácticos y lógicos

[Listado completo de tutoriales](#)

Confeccionaremos un problema y agregaremos adrede una serie de errores tipográficos. Este tipo de errores siempre son detectados por el COMPILADOR, antes de ejecutar el programa.

A los errores tipográficos, como por ejemplo la falta de puntos y comas, nombres de variables incorrectas, falta de paréntesis, palabras claves mal escritas, etc. los llamamos errores SINTACTICOS.

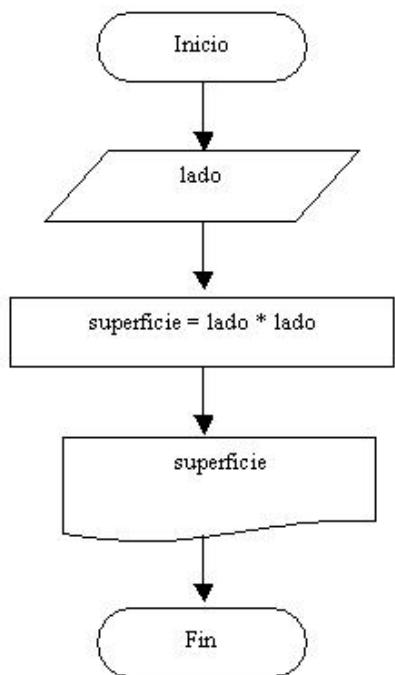
Un programa no se puede ejecutar sin corregir absolutamente todos los errores sintácticos.

Existe otro tipo de errores llamados ERRORES LOGICOS. Este tipo de errores en programas grandes (miles de líneas) son más difíciles de localizar. Por ejemplo un programa que permite hacer la facturación pero la salida de datos por impresora es incorrecta.

## Problema:

Hallar la superficie de un cuadrado conociendo el valor de un lado.

## Diagrama de flujo:

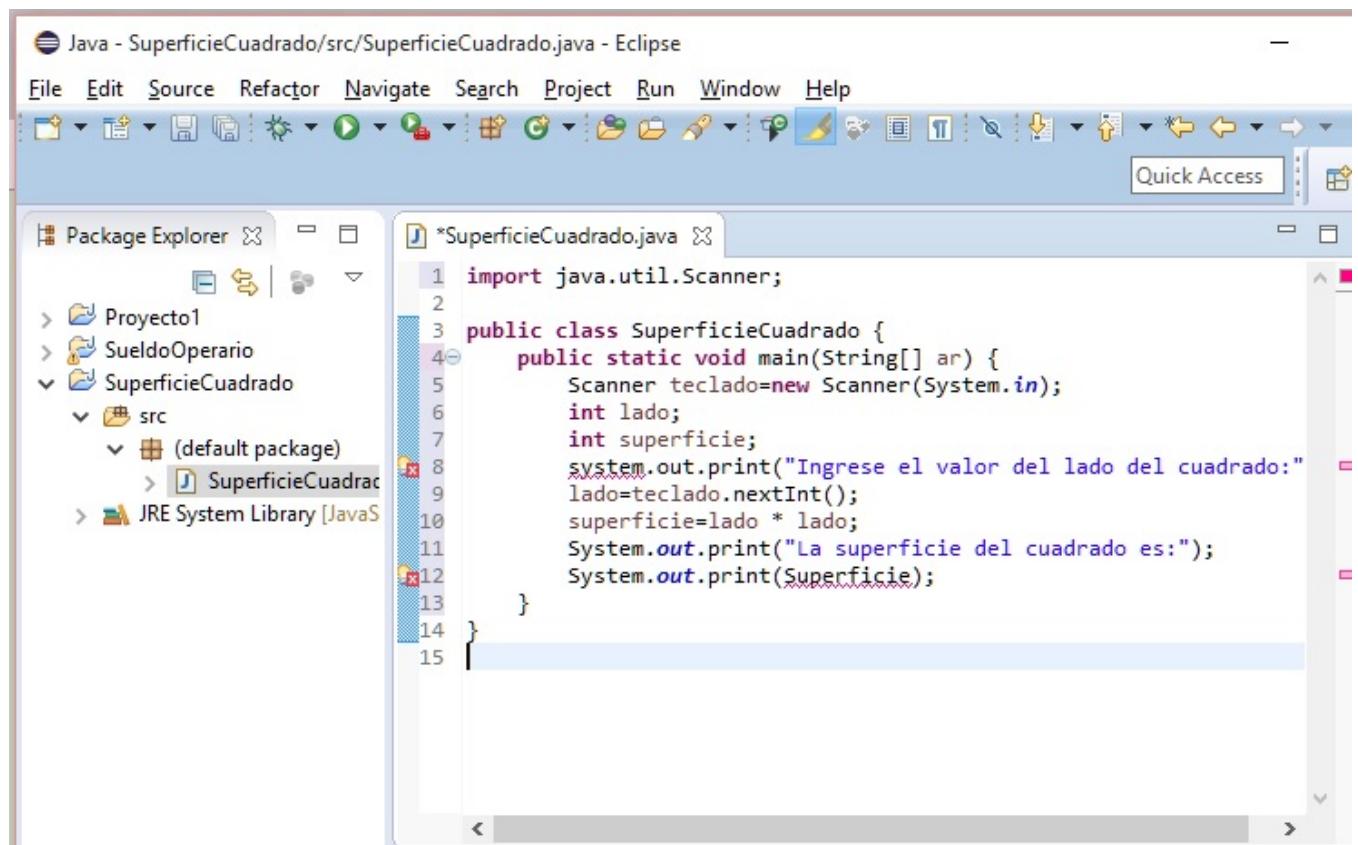


## Proyecto:

Creemos un proyecto llamado SuperficieCuadrado y una clase llamada SuperficieCuadrado.

Codificamos el algoritmo en Java e introducimos dos errores sintáctico:

- 1 - Disponemos el nombre del objeto System con minúsculas.
- 2 - Tratamos de imprimir el nombre de la variable superficie con el primer carácter en mayúsculas.



Como podemos observar aparece subrayado la línea donde disponemos System con minúsculas como en la línea que imprimimos la variable superficie con mayúsculas. Si modificamos y corregimos los dos errores sintácticos podremos ejecutar nuestro programa.

#### Programa correctamente codificado:

```
import java.util.Scanner;

public class SuperficieCuadrado {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int lado;
        int superficie;
        System.out.print("Ingrese el valor del lado del cuadrado:");
        lado=teclado.nextInt();
        superficie=lado * lado;
        System.out.print("La superficie del cuadrado es:");
        System.out.print(superficie);
    }
}
```

Programa con un error lógico:

```
import java.util.Scanner;

public class SuperficieCuadrado {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int lado;
        int superficie;
        System.out.print("Ingrese el valor del lado del cuadrado:");
        lado=teclado.nextInt();
        superficie=lado * lado * lado;
        System.out.print("La superficie del cuadrado es:");
        System.out.print(superficie);
    }
}
```

Como podemos observar si ejecutamos el programa no presenta ningún error de compilación. Pero luego de ingresar el valor del lado del cuadrado (por ejemplo el valor 10) obtenemos como resultado un valor incorrecto (imprime el 1000), esto debido que definimos incorrectamente la fórmula para calcular la superficie del cuadrado:

```
superficie=lado * lado * lado;
```

[Retornar](#)

# 6 - Estructura de programación secuencial

[Listado completo de tutoriales](#)

Cuando en un problema sólo participan operaciones, entradas y salidas se la denomina una estructura secuencial.

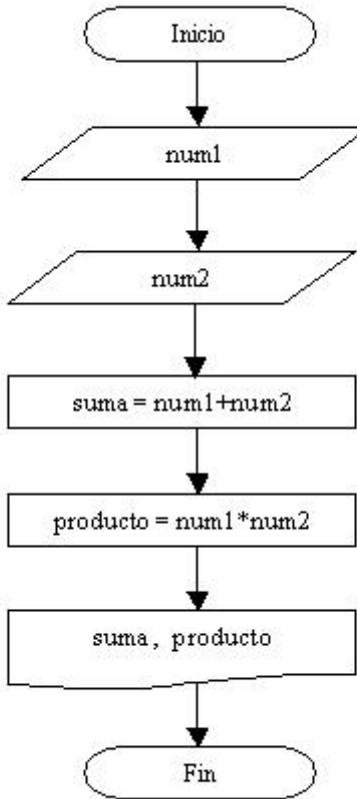
Los problemas diagramados y codificados previamente emplean solo estructuras secuenciales.

La programación requiere una práctica ininterrumpida de diagramación y codificación de problemas.

## **Problema:**

Realizar la carga de dos números enteros por teclado e imprimir su suma y su producto.

## **Diagrama de flujo:**



Tenemos dos entradas num1 y num2 (recordar cuáles son los nombres de variables correctas), dos operaciones: realización de la suma y del producto de los valores ingresados y dos salidas, que son los resultados de la suma y el producto de los valores ingresados. En el símbolo de impresión podemos indicar una o más salidas, eso queda a criterio del programador, lo mismo para indicar las entradas por teclado.

### Programa:

```

import java.util.Scanner;

public class SumaProductoNumeros {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int num1,num2,suma,producto;
        System.out.print("Ingrese primer valor:");
        num1=teclado.nextInt();
        System.out.print("Ingrese segundo valor");
        num2=teclado.nextInt();
        suma=num1 + num2;
        producto=num1 * num2;
        System.out.print("La suma de los dos valores es ")
    }
}
  
```

```
        System.out.println(suma);
        System.out.print("El producto de los dos valore
        System.out.println(producto);
    }
}
```

Recordemos que tenemos que seguir todos los pasos vistos para la creación de un proyecto, su clase, definición de la función main y la codificación del diagrama de flujo.

Algunas cosas nuevas que podemos notar:

- Podemos definir varias variables en la misma línea:

```
int num1, num2, suma, producto;
```

- Si llamamos a la función println en lugar de print, la impresión siguiente se efectuará en la próxima línea:

```
System.out.println(suma);
```

## Problemas propuestos

1. Realizar la carga del lado de un cuadrado, mostrar por pantalla el perímetro del mismo (El perímetro de un cuadrado se calcula multiplicando el valor del lado por cuatro)
2. Escribir un programa en el cual se ingresen cuatro números, calcular e informar la suma de los dos primeros y el producto del tercero y el cuarto.
3. Realizar un programa que lea cuatro valores numéricos e informar su suma y promedio.
4. Se debe desarrollar un programa que pida el ingreso del precio de un artículo y la cantidad que lleva el cliente. Mostrar lo que debe abonar el comprador.

[Solución](#)

[Retornar](#)

# 7 - Estructuras condicionales simples y compuestas

[Listado completo de tutoriales](#)

No todos los problemas pueden resolverse empleando estructuras secuenciales.

Cuando hay que tomar una decisión aparecen las estructuras condicionales.

En nuestra vida diaria se nos presentan situaciones donde debemos decidir.

¿Elijo la carrera A o la carrera B?

¿Me pongo este pantalón?

Para ir al trabajo, ¿elijo el camino A o el camino B?

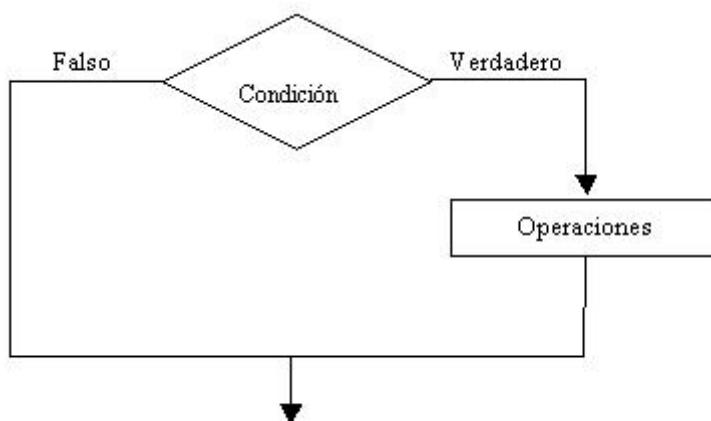
Al cursar una carrera, ¿elijo el turno mañana, tarde o noche?

Por supuesto que en un problema se combinan estructuras secuenciales y condicionales.

## Estructura condicional simple.

Cuando se presenta la elección tenemos la opción de realizar una actividad o no realizar ninguna.

Representación gráfica:



Podemos observar: El rombo representa la condición. Hay dos opciones que se pueden tomar. Si la condición da verdadera se sigue el camino del verdadero, o sea el de la derecha, si la condición da falsa se sigue el camino de la izquierda.

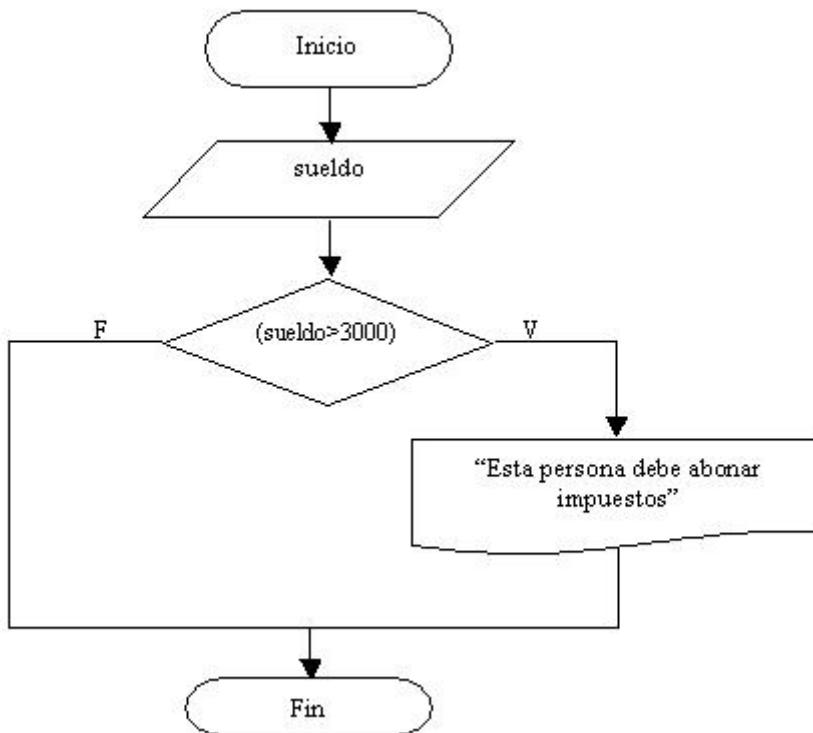
Se trata de una estructura CONDICIONAL SIMPLE porque por el camino del verdadero hay actividades y por el camino del falso no hay actividades.

Por el camino del verdadero pueden existir varias operaciones, entradas y salidas, inclusive ya veremos que puede haber otras estructuras condicionales.

## Problema:

Ingresar el sueldo de una persona, si supera los 3000 pesos mostrar un mensaje en pantalla indicando que debe abonar impuestos.

## Diagrama de flujo:



Podemos observar lo siguiente: Siempre se hace la carga del sueldo, pero si el sueldo que ingresamos supera 3000 pesos se mostrará por pantalla el mensaje "Esta persona debe abonar impuestos", en caso que la persona cobre 3000 o menos no aparece nada por pantalla.

## Programa:

```
import java.util.Scanner;

public class EstructuraCondicionalSimple1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        float sueldo;
        System.out.print("Ingresé el sueldo:");
        sueldo=teclado.nextFloat();
```

```
if (sueldo>3000) {  
    System.out.println("Esta persona debe abonar impuestos")  
}  
}
```

La palabra clave "if" indica que estamos en presencia de una estructura condicional; seguidamente disponemos la condición entre paréntesis. Por último encerrada entre llaves las instrucciones de la rama del verdadero.

Es necesario que las instrucciones a ejecutar en caso que la condición sea verdadera estén encerradas entre llaves {}, con ellas marcamos el comienzo y el fin del bloque del verdadero.

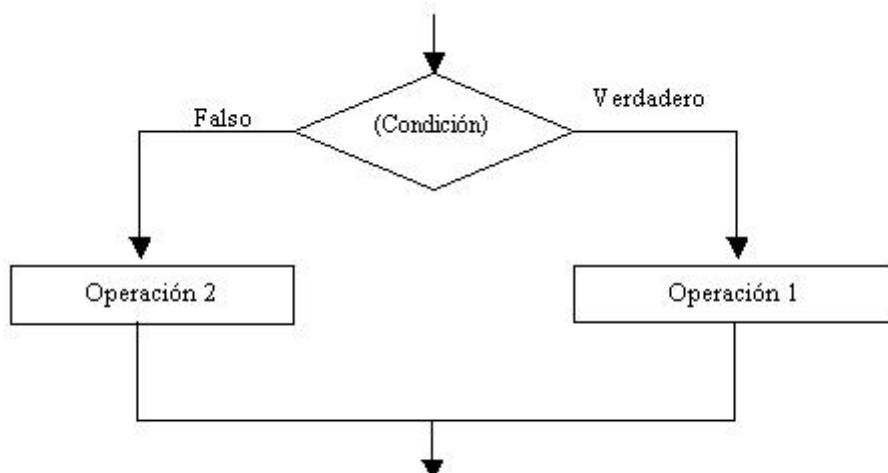
Ejecutando el programa e ingresamos un sueldo superior a 3000 pesos. Podemos observar como aparece en pantalla el mensaje "Esta persona debe abonar impuestos", ya que la condición del if es verdadera.

Volvamos a ejecutar el programa y carguemos un sueldo menor o igual a 3000 pesos. No debe aparecer mensaje en pantalla.

## Estructura condicional compuesta.

Cuando se presenta la elección tenemos la opción de realizar una actividad u otra. Es decir tenemos actividades por el verdadero y por el falso de la condición. Lo más importante que hay que tener en cuenta que se realizan las actividades de la rama del verdadero o las del falso, NUNCA se realizan las actividades de las dos ramas.

Representación gráfica:

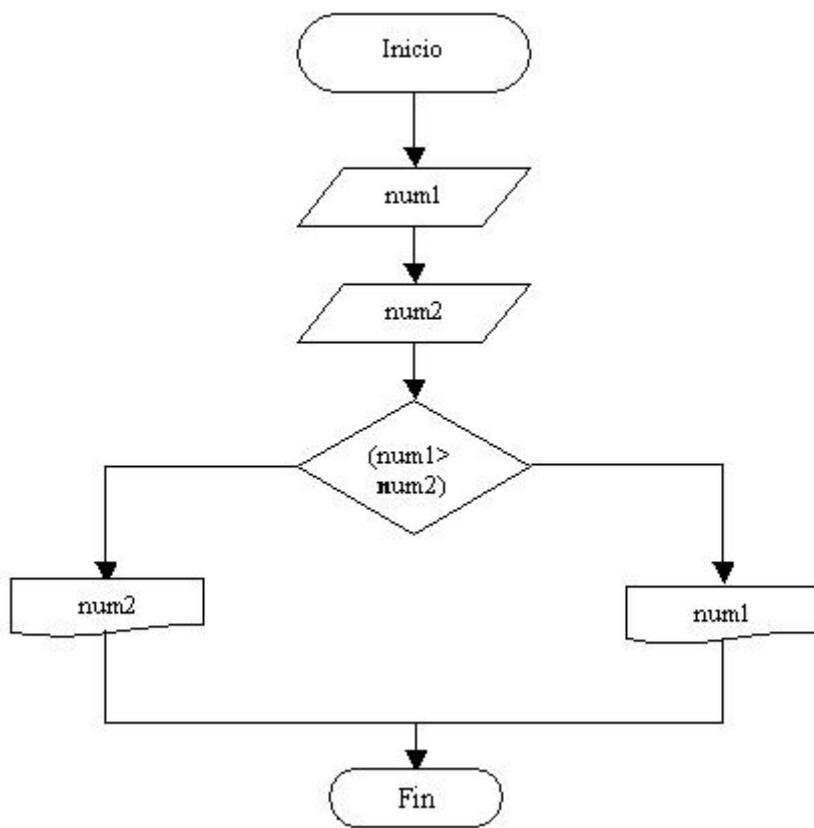


En una estructura condicional compuesta tenemos entradas, salidas, operaciones, tanto por la rama del verdadero como por la rama del falso.

### Problema:

Realizar un programa que solicite ingresar dos números distintos y muestre por pantalla el mayor de ellos.

### Diagrama de flujo:



Se hace la entrada de num1 y num2 por teclado. Para saber cual variable tiene un valor mayor preguntamos si el contenido de num1 es mayor ( $>$ ) que el contenido de num2, si la respuesta es verdadera vamos por la rama de la derecha e imprimimos num1, en caso que la condición sea falsa vamos por la rama de la izquierda (Falsa) e imprimimos num2.

Como podemos observar nunca se imprimen num1 y num2 simultáneamente.

Estamos en presencia de una ESTRUCTURA CONDICIONAL COMPUESTA ya que tenemos actividades por la rama del verdadero y del falso.

### Programa:

```
import java.util.Scanner;

public class EstructuraCondicionalCompuesta1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int num1,num2;
        System.out.print("Ingrese primer valor:");
        num1=teclado.nextInt();
        System.out.print("Ingrese segundo valor:");
        num2=teclado.nextInt();
        if (num1>num2) {

            System.out.print(num1);
        } else {
            System.out.print(num2);
        }
    }
}
```

Cotejemos el diagrama de flujo y la codificación y observemos que el primer bloque de llaves después del if representa la rama del verdadero y el segundo bloque de llaves representa la rama del falso.

Compilemos el programa, si hubo errores sintácticos corrijamos y carguemos dos valores, como por ejemplo:

```
Ingrese el primer valor: 10
Ingrese el segundo valor: 4
10
```

Si ingresamos los valores 10 y 4 la condición del if retorna verdadero y ejecuta el primer bloque.

Un programa se controla y corrige probando todos sus posibles resultados.

Ejecutemos nuevamente el programa e ingresemos:

```
Ingrese el primer valor: 10
Ingrese el segundo valor: 54
54
```

Cuando a un programa le corregimos todos los errores sintácticos y lógicos ha terminado nuestra tarea y podemos entregar el mismo al USUARIO que nos lo solicitó.

# Operadores

En una condición deben disponerse únicamente variables, valores constantes y operadores relacionales.

## >Operadores Relacionales:

```
> (mayor)
< (menor)
>= (mayor o igual)
<= (menor o igual)
== (igual)
!= (distinto)
```

## Operadores Matemáticos

```
+ (más)
- (menos)
* (producto)
/ (división)
% (resto de una división) Ej.: x=13%5; {se guarda 3}
```

Hay que tener en cuenta que al disponer una condición debemos seleccionar que operador relacional se adapta a la pregunta.

Ejemplos:

Se ingresa un número multiplicarlo por 10 si es distinto a  
Se ingresan dos números mostrar una advertencia si son iguales

Los problemas que se pueden presentar son infinitos y la correcta elección del operador sólo se alcanza con la práctica intensiva en la resolución de problemas.

## Problemas propuestos

1. Realizar un programa que lea por teclado dos números, si el primero es mayor al segundo informar su suma y diferencia, en caso contrario informar el producto y la división del primero respecto al segundo.
2. Se ingresan tres notas de un alumno, si el promedio es mayor o igual a siete mostrar un mensaje "Promocionado".
3. Se ingresa por teclado un número positivo de uno o dos dígitos (1..99) mostrar un mensaje indicando si el número tiene uno o dos dígitos.  
(Tener en cuenta que condición debe cumplirse para tener dos dígitos, un número entero)

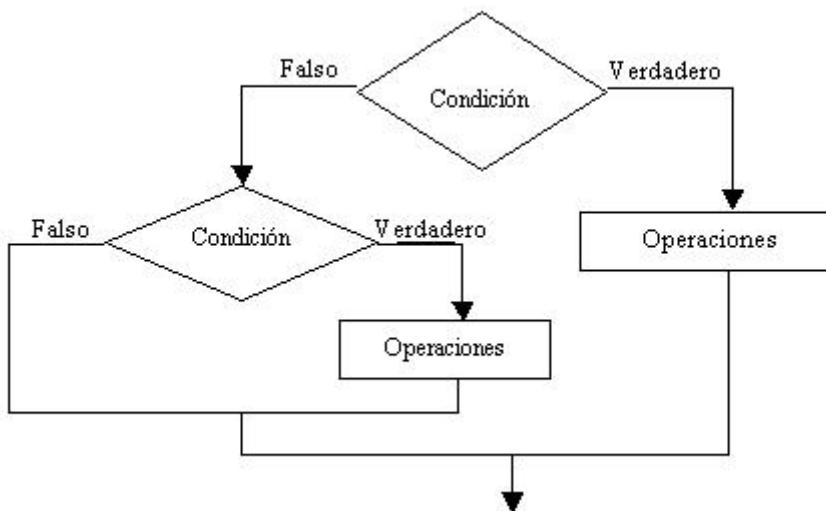
[Solución](#)

[\*\*Retornar\*\*](#)

# 8 - Estructuras condicionales anidadas

[Listado completo de tutoriales](#)

Decimos que una estructura condicional es anidada cuando por la rama del verdadero o el falso de una estructura condicional hay otra estructura condicional.



El diagrama de flujo que se presenta contiene dos estructuras condicionales. La principal se trata de una estructura condicional compuesta y la segunda es una estructura condicional simple y está contenida por la rama del falso de la primer estructura.

Es común que se presenten estructuras condicionales anidadas aún más complejas.

## Problema:

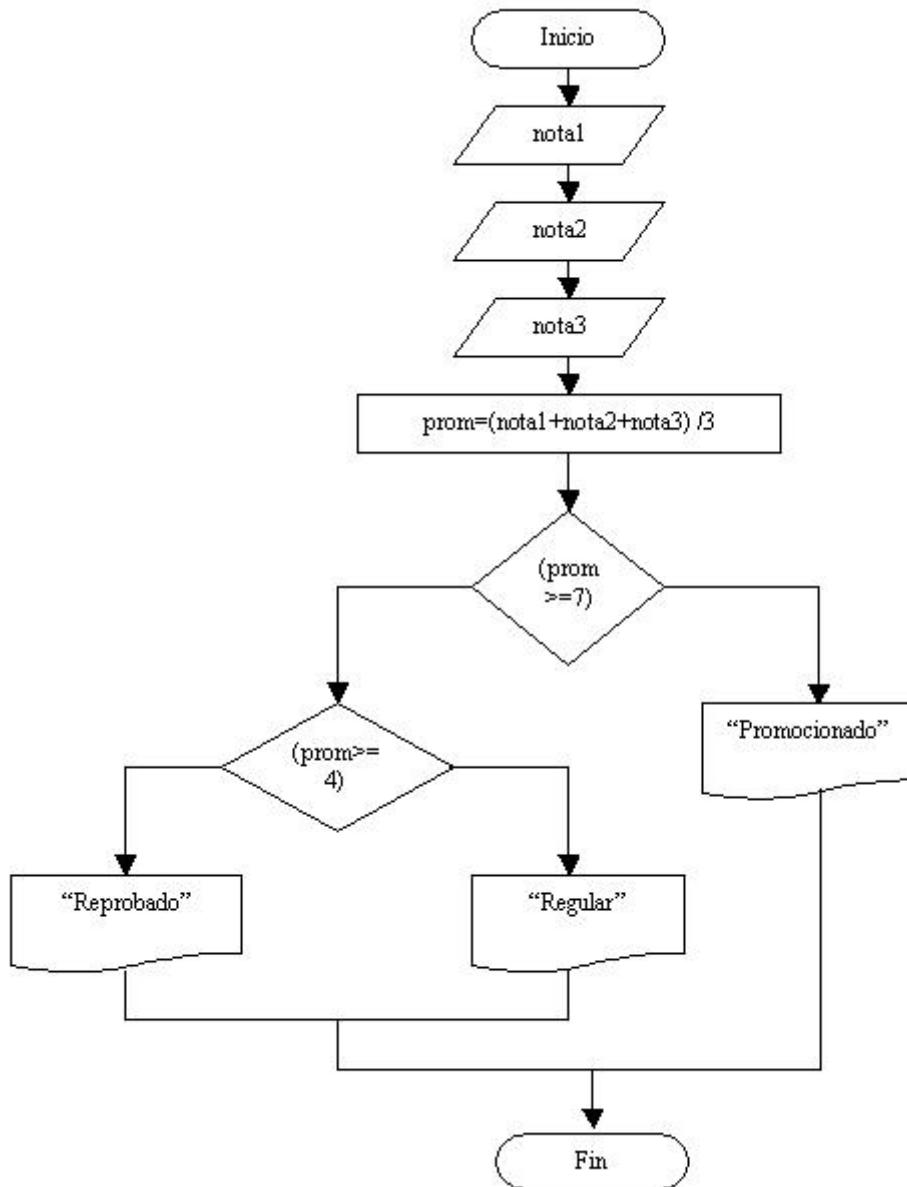
Confeccionar un programa que pida por teclado tres notas de un alumno, calcule el promedio e imprima alguno de estos mensajes:

Si el promedio es  $\geq 7$  mostrar "Promocionado".

Si el promedio es  $\geq 4$  y  $< 7$  mostrar "Regular".

Si el promedio es  $< 4$  mostrar "Reprobado".

## Diagrama de flujo:



Analicemos el siguiente diagrama. Se ingresan tres valores por teclado que representan las notas de un alumno, se obtiene el promedio sumando los tres valores y dividiendo por 3 dicho resultado (Tener en cuenta que si el resultado es un valor real solo se almacena la parte entera).

Primeramente preguntamos si el promedio es superior o igual a 7, en caso afirmativo va por la rama del verdadero de la estructura condicional mostramos un mensaje que indica "Promocionado" (con comillas indicamos un texto que debe imprimirse en pantalla).

En caso que la condición nos de falso, por la rama del falso aparece otra estructura condicional, porque todavía debemos averiguar si el promedio del alumno es superior o igual a cuatro o inferior a cuatro.

Estamos en presencia de dos estructuras condicionales compuestas.

## Programa:

```
import java.util.Scanner;

public class EstructuraCondicionalAnidada1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int nota1,nota2,nota3;
        System.out.print("Ingrese primer nota:");
        nota1=teclado.nextInt();
        System.out.print("Ingrese segunda nota:");
        nota2=teclado.nextInt();
        System.out.print("Ingrese tercer nota:");
        nota3=teclado.nextInt();
        int promedio=(nota1 + nota2 + nota3) / 3;
        if (promedio>=7) {
            System.out.print("Promocionado");
        } else {
            if (promedio>=4) {
                System.out.print("Regular");
            } else {
                System.out.print("Reprobado");
            }
        }
    }
}
```

Codifiquemos y ejecutemos este programa. Al correr el programa deberá solicitar por teclado la carga de tres notas y mostrarnos un mensaje según el promedio de las mismas.

Podemos definir un conjunto de variables del mismo tipo en una misma línea:

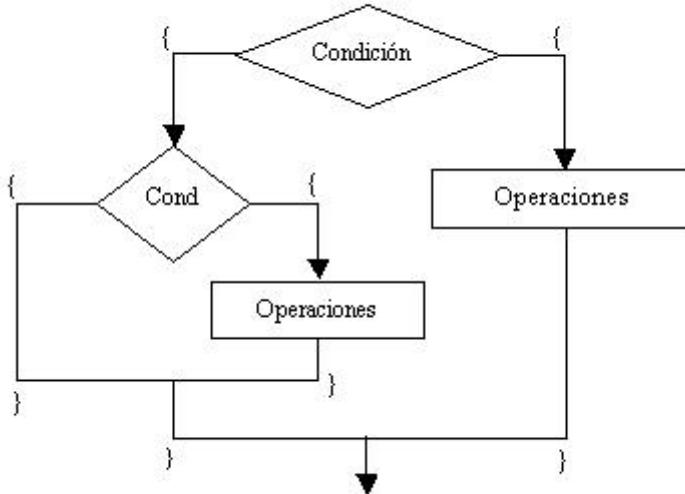
```
int nota1,nota2,nota3;
```

Esto no es obligatorio pero a veces, por estar relacionadas, conviene.

A la codificación del if anidado podemos observarla por el else del primer if.

Para no tener problemas (olvidarnos) con las llaves de apertura y cerrado podemos ver la siguiente regla:

Cada vértice representa una llave de apertura y una de cierre:



## Problemas propuestos

1. Se cargan por teclado tres números distintos. Mostrar por pantalla el mayor de ellos.
2. Se ingresa por teclado un valor entero, mostrar una leyenda que indique si el número es positivo, nulo o negativo.
3. Confeccionar un programa que permita cargar un número entero positivo de hasta tres cifras y muestre un mensaje indicando si tiene 1, 2, o 3 cifras. Mostrar un mensaje de error si el número de cifras es mayor.
4. Un postulante a un empleo, realiza un test de capacitación, se obtuvo la siguiente información: cantidad total de preguntas que se le realizaron y la cantidad de preguntas que contestó correctamente. Se pide confeccionar un programa que ingrese los dos datos por teclado e informe el nivel del mismo según el porcentaje de respuestas correctas que ha obtenido, y sabiendo que:

Nivel máximo: Porcentaje $\geq 90\%$ .  
 Nivel medio: Porcentaje $\geq 75\% \text{ y } < 90\%$ .  
 Nivel regular: Porcentaje $\geq 50\% \text{ y } < 75\%$ .  
 Fuera de nivel: Porcentaje $< 50\%$ .

[Solución](#)

[Retornar](#)

# 9 - Condiciones compuestas con operadores lógicos

[Listado completo de tutoriales](#)

Hasta ahora hemos visto los operadores:

relacionales ( $>$ ,  $<$ ,  $\geq$ ,  $\leq$ ,  $\neq$ ,  $\equiv$ )  
matemáticos ( $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$ )

pero nos están faltando otros operadores imprescindibles:

lógicos ( $\&\&$ ,  $\|$ ).

Estos dos operadores se emplean fundamentalmente en las estructuras condicionales para agrupar varias condiciones simples.

## Operador $\&\&$



Traducido se lo lee como ?Y?. Si la Condición 1 es verdadera Y la condición 2 es verdadera luego ejecutar la rama del verdadero.

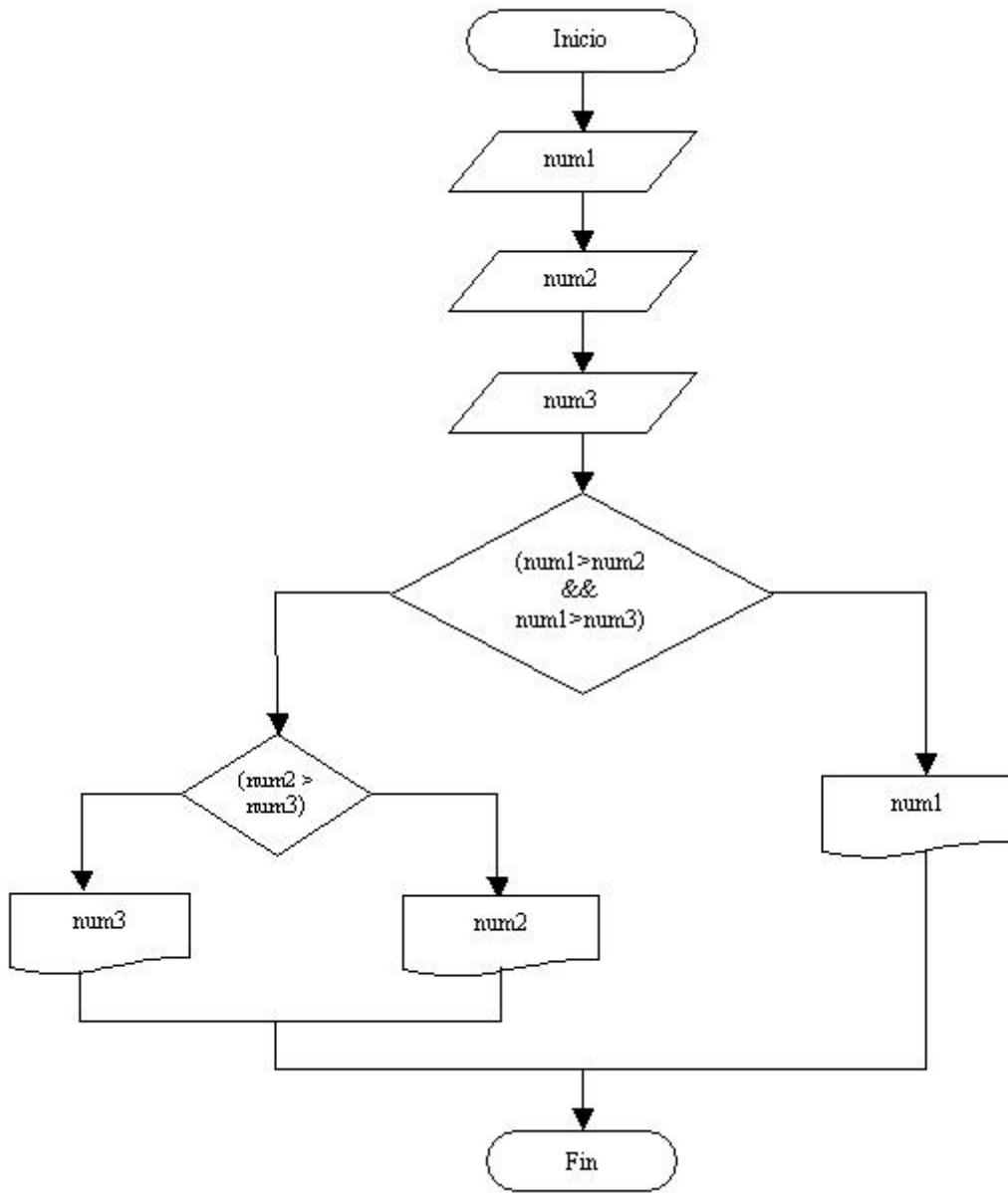
Cuando vinculamos dos o más condiciones con el operador ? $\&\&$ ?, las dos condiciones deben ser verdaderas para que el resultado de la condición compuesta de Verdadero y continúe por la rama del verdadero de la estructura condicional.

La utilización de operadores lógicos permiten en muchos casos plantear algoritmos más cortos y comprensibles.

## Problema:

Confeccionar un programa que lea por teclado tres números distintos y nos muestre el mayor.

## Diagrama de flujo:



Este ejercicio está resuelto sin emplear operadores lógicos en un concepto anterior del tutorial. La primera estructura condicional es una ESTRUCTURA CONDICIONAL COMPUESTA con una CONDICION COMPUESTA.

Podemos leerla de la siguiente forma:

Si el contenido de la variable num1 es mayor al contenido de la variable num2 Y si el contenido de la variable num1 es mayor al contenido de la variable num3 entonces la CONDICION COMPUESTA resulta Verdadera.

Si una de las condiciones simples da falso la CONDICION COMPUESTA da Falso y continua por la rama del falso.

Es decir que se mostrará el contenido de num1 si y sólo si  $\text{num1} > \text{num2}$  y  $\text{num1} > \text{num3}$ . En caso de ser Falsa la condición, analizamos el contenido de num2 y num3 para ver

cual tiene un valor mayor.

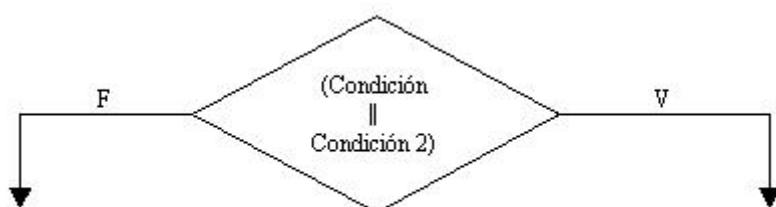
En esta segunda estructura condicional no se requieren operadores lógicos al haber una condición simple.

### Programa:

```
import java.util.Scanner;

public class CondicionesCompuestas1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int num1,num2,num3;
        System.out.print("Ingrese primer valor:");
        num1=teclado.nextInt();
        System.out.print("Ingrese segundo valor:");
        num2=teclado.nextInt();
        System.out.print("Ingrese tercer valor:");
        num3=teclado.nextInt();
        if (num1>num2 && num1>num3) {
            System.out.print(num1);
        } else {
            if (num2>num3) {
                System.out.print(num2);
            }else {
                System.out.print(num3);
            }
        }
    }
}
```

### Operador ||



Traducido se lo lee como ?O?. Si la condición 1 es Verdadera O la condición 2 es Verdadera, luego ejecutar la rama del Verdadero.

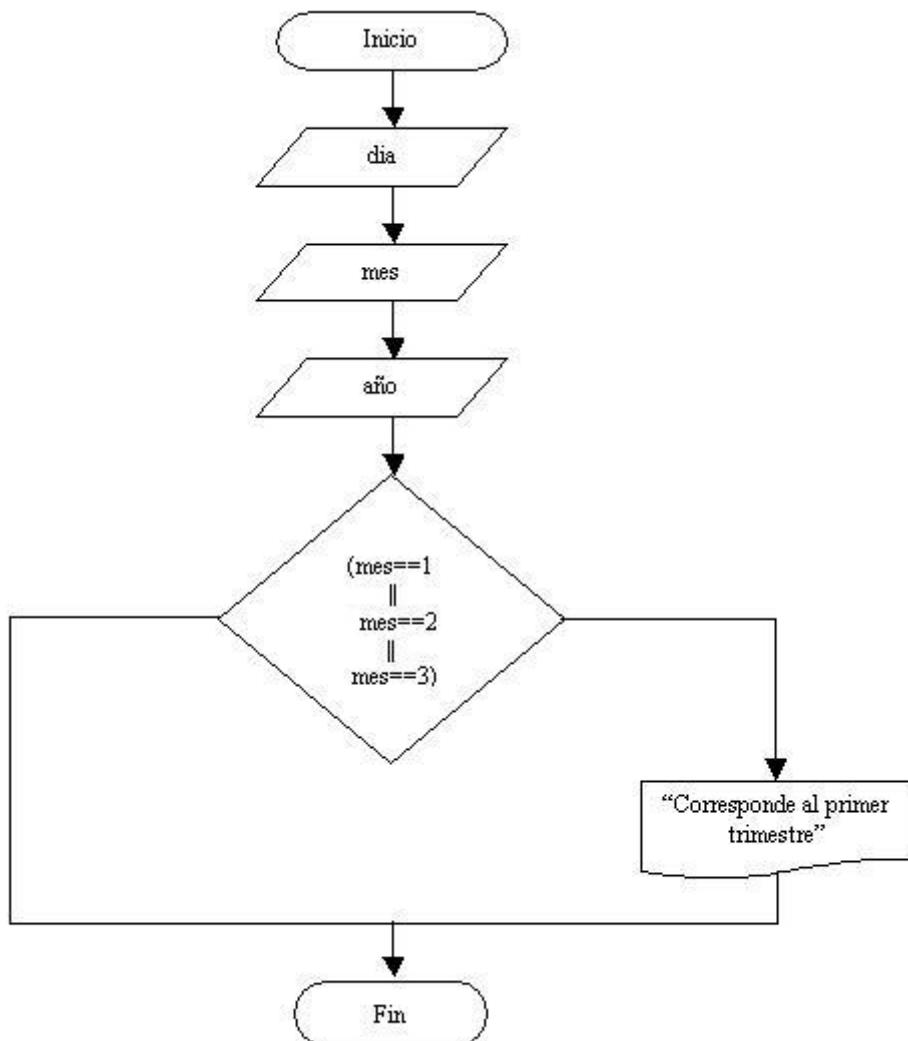
Cuando vinculamos dos o más condiciones con el operador ?Or", con que una de las dos condiciones sea Verdadera alcanza para que el resultado de la condición compuesta sea Verdadero.

### Problema:

Se carga una fecha (día, mes y año) por teclado. Mostrar un mensaje si corresponde al primer trimestre del año (enero, febrero o marzo) Cargar por teclado el valor numérico del día, mes y año.

Ejemplo: dia:10 mes:1 año:2010.

### Diagrama de flujo:



La carga de una fecha se hace por partes, ingresamos las variables dia, mes y año. Mostramos el mensaje "Corresponde al primer trimestre" en caso que el mes ingresado por teclado sea igual a 1, 2 ó 3. En la condición no participan las variables dia y año.

### Programa:

```
import java.util.Scanner;

public class CondicionesCompuestas2 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int dia,mes,año;
        System.out.print("Ingrese nro de día:");
        dia=teclado.nextInt();
        System.out.print("Ingrese nro de mes:");
        mes=teclado.nextInt();
        System.out.print("Ingrese nro de año:");
        año=teclado.nextInt();
        if (mes==1 || mes==2 || mes==3) {
            System.out.print("Corresponde al primer tri
        }
    }
}
```

## Problemas propuestos

1. Realizar un programa que pida cargar una fecha cualquiera, luego verificar si dicha fecha corresponde a Navidad.
2. Se ingresan tres valores por teclado, si todos son iguales se imprime la suma del primero con el segundo y a este resultado se lo multiplica por el tercero.
3. Se ingresan por teclado tres números, si todos los valores ingresados son menores a 10, imprimir en pantalla la leyenda "Todos los números son menores a diez".
4. Se ingresan por teclado tres números, si al menos uno de los valores ingresados es menor a 10, imprimir en pantalla la leyenda "Alguno de los números es menor a diez".
5. Escribir un programa que pida ingresar la coordenada de un punto en el plano, es decir dos valores enteros x e y (distintos a cero). Posteriormente imprimir en pantalla en que cuadrante se ubica dicho punto. (1º Cuadrante si x > 0 Y y > 0 , 2º Cuadrante: x < 0 Y y > 0, etc.)

6. De un operario se conoce su sueldo y los años de antigüedad. Se pide confeccionar un programa que lea los datos de entrada e informe:
  - a) Si el sueldo es inferior a 500 y su antigüedad es igual o superior a 10 años, otorgarle un aumento del 20 %, mostrar el sueldo a pagar.
  - b) Si el sueldo es inferior a 500 pero su antigüedad es menor a 10 años, otorgarle un aumento de 5 %.
  - c) Si el sueldo es mayor o igual a 500 mostrar el sueldo en pantalla sin cambios.
7. Escribir un programa en el cual: dada una lista de tres valores numéricos distintos se calcule e informe su rango de variación (debe mostrar el mayor y el menor de ellos)

[Solución](#)

[\*\*Retornar\*\*](#)

# 10 - Estructura repetitiva while

[Listado completo de tutoriales](#)

Hasta ahora hemos empleado estructuras SECUENCIALES y CONDICIONALES. Existe otro tipo de estructuras tan importantes como las anteriores que son las estructuras REPETITIVAS.

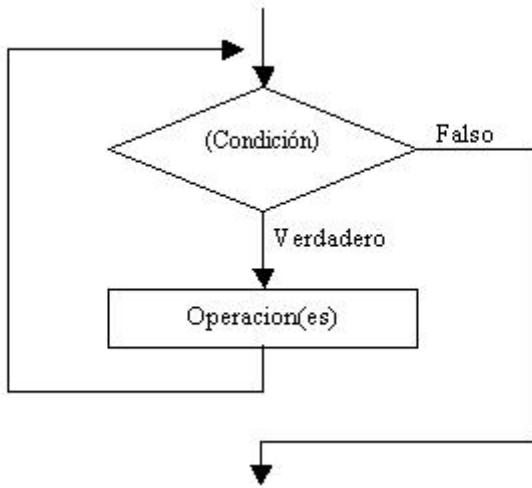
Una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces.

Una ejecución repetitiva de sentencias se caracteriza por:

- La o las sentencias que se repiten.
- El test o prueba de condición antes de cada repetición, que motivará que se repitan o no las sentencias.

## Estructura repetitiva while.

Representación gráfica de la estructura while:



No debemos confundir la representación gráfica de la estructura repetitiva while (Mientras) con la estructura condicional if (Si)

Funcionamiento: En primer lugar se verifica la condición, si la misma resulta verdadera se ejecutan las operaciones que indicamos por la rama del Verdadero.

A la rama del verdadero la graficamos en la parte inferior de la condición. Una línea al final del bloque de repetición la conecta con la parte superior de la estructura repetitiva. En caso que la condición sea Falsa continúa por la rama del Falso y sale de la estructura repetitiva para continuar con la ejecución del algoritmo.

El bloque se repite MIENTRAS la condición sea Verdadera.

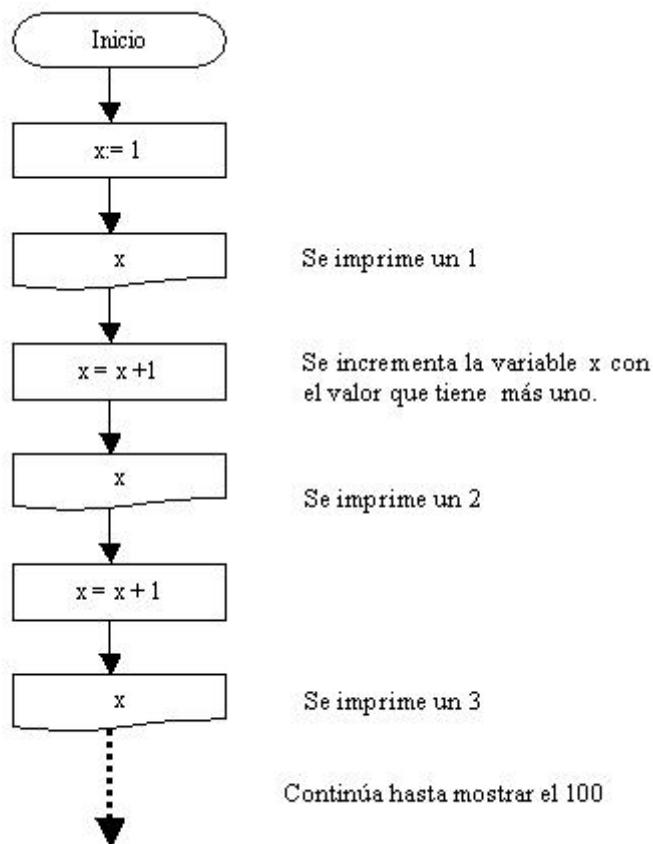
**Importante:** Si la condición siempre retorna verdadero estamos en presencia de un ciclo repetitivo infinito. Dicha situación es un error de programación, nunca finalizará el programa.

### Problema 1:

Realizar un programa que imprima en pantalla los números del 1 al 100.

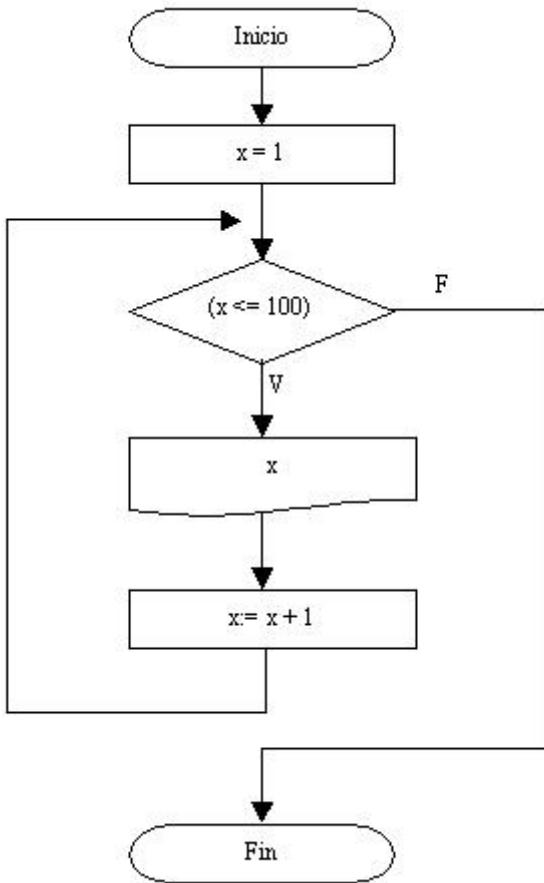
Si no conocemos las estructuras repetitivas podemos resolver el problema empleando una estructura secuencial. Inicializamos una variable con el valor 1, luego imprimimos la variable, incrementamos nuevamente la variable y así sucesivamente.

### Diagrama de flujo:



Si continuamos con el diagrama no nos alcanzarían las próximas 5 páginas para finalizarlo. Emplear una estructura secuencial para resolver este problema produce un diagrama de flujo y un programa en Java muy largo.

Ahora veamos la solución empleando una estructura repetitiva while:



Es muy importante analizar este diagrama:

La primera operación inicializa la variable  $x$  en 1, seguidamente comienza la estructura repetitiva while y disponemos la siguiente condición ( $x \leq 100$ ), se lee MIENTRAS la variable  $x$  sea menor o igual a 100.

Al ejecutarse la condición retorna VERDADERO porque el contenido de  $x$  (1) es menor o igual a 100. Al ser la condición verdadera se ejecuta el bloque de instrucciones que contiene la estructura while. El bloque de instrucciones contiene una salida y una operación.

Se imprime el contenido de  $x$ , y seguidamente se incrementa la variable  $x$  en uno.

La operación  $x=x + 1$  se lee como "en la variable  $x$  se guarda el contenido de  $x$  más 1". Es decir, si  $x$  contiene 1 luego de ejecutarse esta operación se almacenará en  $x$  un 2.

Al finalizar el bloque de instrucciones que contiene la estructura repetitiva se verifica nuevamente la condición de la estructura repetitiva y se repite el proceso explicado anteriormente.

Mientras la condición retorne verdadero se ejecuta el bloque de instrucciones; al retornar falso la verificación de la condición se sale de la estructura repetitiva y continua el algoritmo, en este caso finaliza el programa.

Lo más difícil es la definición de la condición de la estructura while y qué bloque de instrucciones se van a repetir. Observar que si, por ejemplo, disponemos la condición  $x \geq 100$  ( si x es mayor o igual a 100) no provoca ningún error sintáctico pero estamos en presencia de un error lógico porque al evaluarse por primera vez la condición retorna falso y no se ejecuta el bloque de instrucciones que queríamos repetir 100 veces.

No existe una RECETA para definir una condición de una estructura repetitiva, sino que se logra con una práctica continua solucionando problemas.

Una vez planteado el diagrama debemos verificar si el mismo es una solución válida al problema (en este caso se debe imprimir los números del 1 al 100 en pantalla), para ello podemos hacer un seguimiento del flujo del diagrama y los valores que toman las variables a lo largo de la ejecución:

```
x
1
2
3
4
.
.
100
101 Cuando x vale 101 la condición de la estructura repetitiva se cumple
en este caso finaliza el diagrama.
```

**Importante:** Podemos observar que el bloque repetitivo puede no ejecutarse ninguna vez si la condición retorna falso la primera vez.

La variable x debe estar inicializada con algún valor antes que se ejecute la operación  $x=x + 1$  en caso de no estar inicializada aparece un error de compilación.

### Programa:

```
public class EstructuraRepetitivaWhile1 {
    public static void main(String[] ar) {
        int x;
        x=1;
        while (x<=100) {
            System.out.print(x);
            System.out.print(" - ");
            x = x + 1;
        }
    }
}
```

---

**Importante:** Como podemos observar no hemos creado un objeto de la clase Scanner. Esto debido a que en este programa no hay que ingresar datos por teclado. Para las salidas utilizamos la función print, que se encuentra creada por defecto en cualquier programa que codifiquemos en Java.

Recordemos que un problema no estará 100% solucionado si no hacemos el programa en Java que muestre los resultados buscados.

Probemos algunas modificaciones de este programa y veamos que cambios se deberían hacer para:

- 1 - Imprimir los números del 1 al 500.
- 2 - Imprimir los números del 50 al 100.
- 3 - Imprimir los números del -50 al 0.
- 4 - Imprimir los números del 2 al 100 pero de 2 en 2 (2,4,6,8 ....).

Respuestas:

- 1 - Debemos cambiar la condición del while con  $x \leq 500$ .
- 2 - Debemos inicializar  $x$  con el valor 50.
- 3 - Inicializar  $x$  con el valor -50 y fijar la condición  $x \leq 0$ .
- 4 - Inicializar a  $x$  con el valor 2 y dentro del bloque repetitivo :  
$$( x = x + 2 ).$$

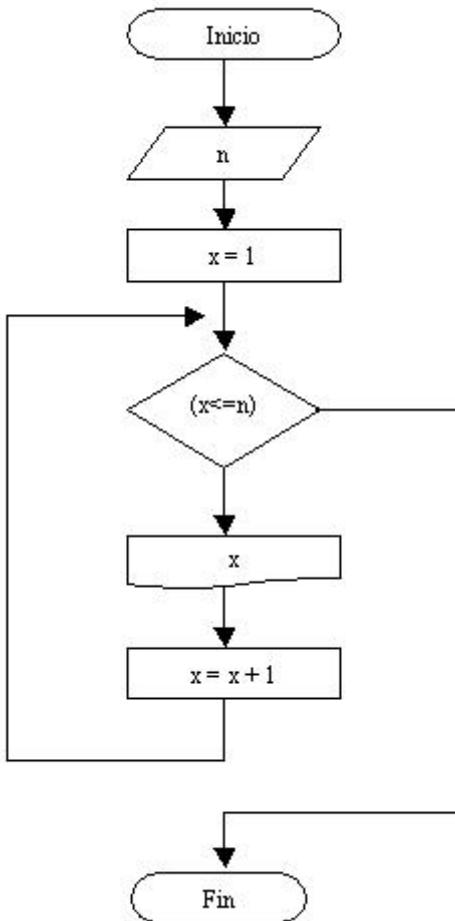
## Problema 2:

Escribir un programa que solicite la carga de un valor positivo y nos muestre desde 1 hasta el valor ingresado de uno en uno.

Ejemplo: Si ingresamos 30 se debe mostrar en pantalla los números del 1 al 30.

Es de FUNDAMENTAL importancia analizar los diagramas de flujo y la posterior codificación en Java de los siguientes problemas, en varios problemas se presentan otras situaciones no vistas en el ejercicio anterior.

## Diagrama de flujo:



Podemos observar que se ingresa por teclado la variable n. El operador puede cargar cualquier valor.

Si el operador carga 10 el bloque repetitivo se ejecutará 10 veces, ya que la condición es ?Mientras  $x \leq n$  ?, es decir ?mientras x sea menor o igual a 10?; pues x comienza en uno y se incrementa en uno cada vez que se ejecuta el bloque repetitivo.

A la prueba del diagrama la podemos realizar dándole valores a las variables; por ejemplo, si ingresamos 5 el seguimiento es el siguiente:

|   |   |
|---|---|
| n | x   |
| 5 | 1 (Se imprime el contenido de x)                    |
|   | 2   |
|   | 3   |
|   | 4   |
|   | 5   |
|   | 6 (Sale del while porque 6 no es menor o igual a 5) |

### Programa:

```

import java.util.Scanner;

public class EstructurasPorIteracionWhile {
  
```

```
public class estructurarepetitiva{
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int n,x;
        System.out.print("Ingrese el valor final:");
        n=teclado.nextInt();
        x=1;
        while (x<=n) {
            System.out.print(x);
            System.out.print(" - ");
            x = x + 1;
        }
    }
}
```

Los nombres de las variables n y x pueden ser palabras o letras (como en este caso)

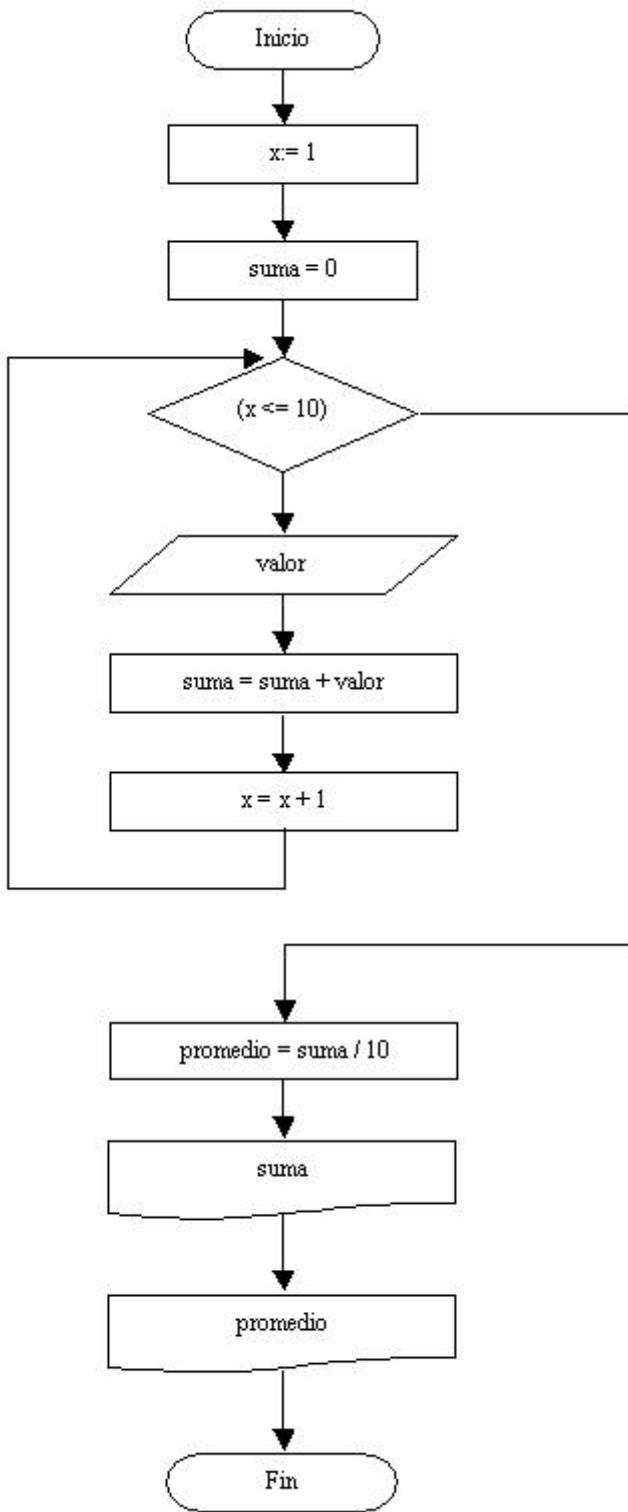
La variable x recibe el nombre de CONTADOR. Un contador es un tipo especial de variable que se incrementa o decrementa con valores constantes durante la ejecución del programa.

El contador x nos indica en cada momento la cantidad de valores impresos en pantalla.

### Problema 3:

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio.

### Diagrama de flujo:



En este problema, a semejanza de los anteriores, llevamos un CONTADOR llamado  $x$  que nos sirve para contar las vueltas que debe repetir el while.

También aparece el concepto de ACUMULADOR (un acumulador es un tipo especial de variable que se incrementa o decrementa con valores variables durante la ejecución del programa)

Hemos dado el nombre de suma a nuestro acumulador. Cada ciclo que se repita la estructura repetitiva, la variable suma se incrementa con el contenido ingresado en la variable valor.

La prueba del diagrama se realiza dándole valores a las variables:

| valor  | suma | x  | promedio |
|--|------|----|----------|
|  | 0    | 0  |          |
| (Antes de entrar a la estructura repetitiva estos son los valores) |      |    |          |
| 5  | 5    | 1  |          |
| 16   | 21   | 2  |          |
| 7  | 28   | 3  |          |
| 10   | 38   | 4  |          |
| 2  | 40   | 5  |          |
| 20   | 60   | 6  |          |
| 5  | 65   | 7  |          |
| 5  | 70   | 8  |          |
| 10   | 80   | 9  |          |
| 2  | 82   | 10 |          |
| 8  | 90   | 11 |          |
|  |      |    | 9        |

Este es un seguimiento del diagrama planteado. Los números que toma la variable valor dependerá de qué cifras cargue el operador durante la ejecución del programa. El promedio se calcula al salir de la estructura repetitiva (es decir primero sumamos los 10 valores ingresados y luego los dividimos por 10)

Hay que tener en cuenta que cuando en la variable valor se carga el primer valor (en este ejemplo 5) al cargarse el segundo valor (16) el valor anterior 5 se pierde, por ello la necesidad de ir almacenando en la variable suma los valores ingresados.

### Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaWhile3 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int x,suma,valor,promedio;
        x=1;
        suma=0;
        while (x<=10) {
            System.out.print("Ingrese un valor:");
            valor=teclado.nextInt();
```

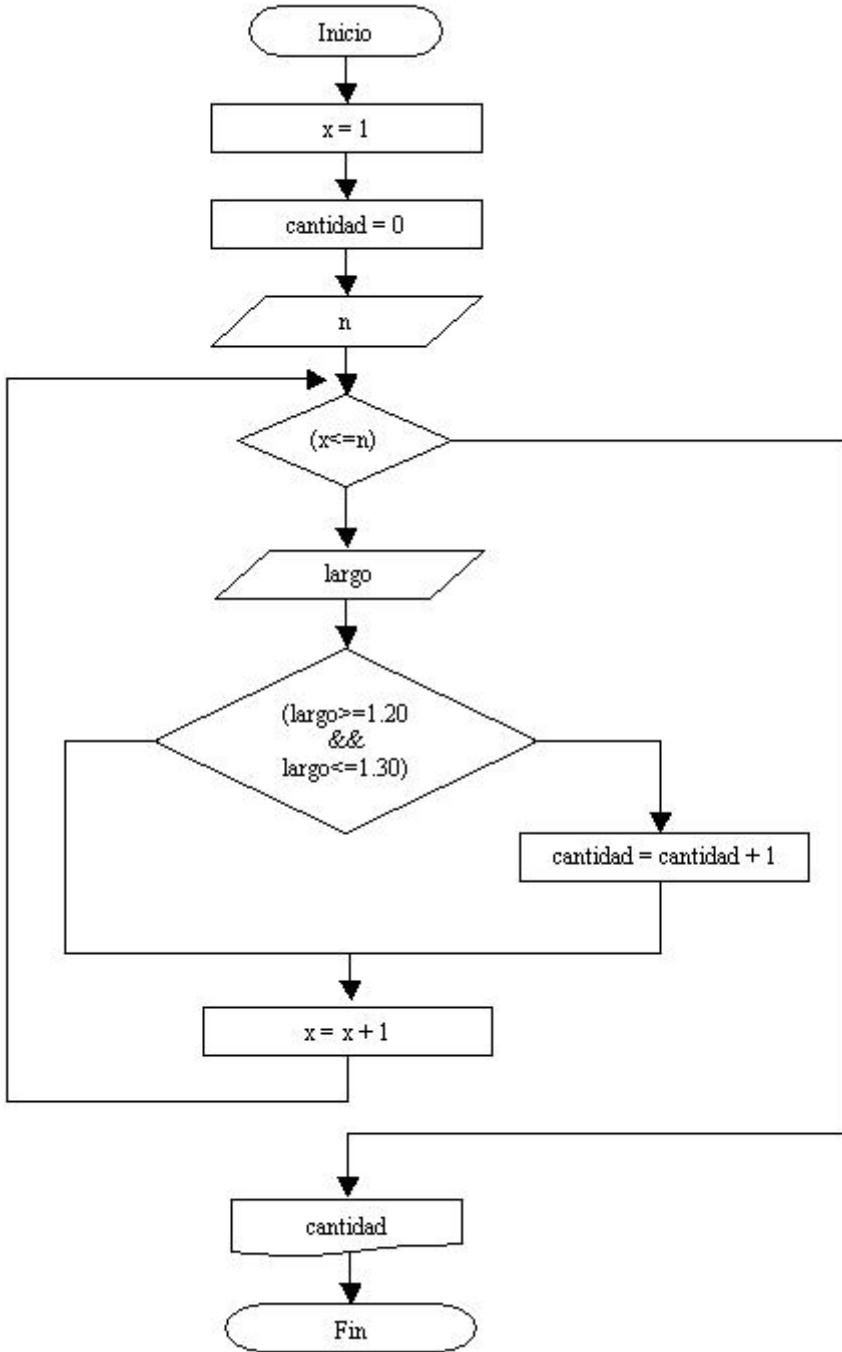
```
        suma=suma+valor;
        x=x+1;
    }
    promedio=suma/10;
    System.out.print("La suma de los 10 valores es:");
    System.out.println(suma);
    System.out.print("El promedio es:");
    System.out.print(promedio);
}
}
```

#### Problema 4:

Una planta que fabrica perfiles de hierro posee un lote de n piezas.

Confeccionar un programa que pida ingresar por teclado la cantidad de piezas a procesar y luego ingrese la longitud de cada perfil; sabiendo que la pieza cuya longitud esté comprendida en el rango de 1,20 y 1,30 son aptas. Imprimir por pantalla la cantidad de piezas aptas que hay en el lote.

#### Diagrama de flujo:



Podemos observar que dentro de una estructura repetitiva puede haber estructuras condicionales (inclusive puede haber otras estructuras repetitivas que veremos más adelante)

En este problema hay que cargar inicialmente la cantidad de piezas a ingresar ( $n$ ), seguidamente se cargan  $n$  valores de largos de piezas.

Cada vez que ingresamos un largo de pieza ( $largo$ ) verificamos si es una medida correcta (debe estar entre 1.20 y 1.30 el largo para que sea correcta), en caso de ser correcta la CONTAMOS (incrementamos la variable  $cantidad$  en 1)

Al contador cantidad lo inicializamos en cero porque inicialmente no se ha cargado ningún largo de medida.

Cuando salimos de la estructura repetitiva porque se han cargado n largos de piezas mostramos por pantalla el contador cantidad (que representa la cantidad de piezas aptas)

En este problema tenemos dos CONTADORES:

x  
cantidad

(Cuenta la cantidad de piezas cargadas hasta el momento)  
(Cuenta los perfiles de hierro aptos)

**Programa:**

```
import java.util.Scanner;

public class EstructuraRepetitivaWhile4 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int x,cantidad,n;
        float largo;
        x=1;
        cantidad=0;
        System.out.print("Cuantas piezas procesará:");
        n=teclado.nextInt();
        while (x<=n) {
            System.out.print("Ingrese la medida de la pieza:");
            largo=teclado.nextFloat();
            if (largo>=1.20 && largo<=1.30) {
                cantidad = cantidad +1;
            }
            x=x + 1;
        }
        System.out.print("La cantidad de piezas aptas es:");
        System.out.print(cantidad);
    }
}
```

## Problemas propuestos

Ha llegado la parte fundamental, que es el momento donde uno desarrolla individualmente un algoritmo para la resolución de problemas.

El tiempo a dedicar a esta sección EJERCICIOS PROPUESTOS debe ser mucho mayor que el empleado a la sección de EJERCICIOS RESUELTOS.

La experiencia dice que debemos dedicar el 80% del tiempo a la resolución individual de problemas y el otro 20% al análisis y codificación de problemas ya resueltos por otras personas.

Es de vital importancia para llegar a ser un buen PROGRAMADOR poder resolver problemas en forma individual.

1. Escribir un programa que solicite ingresar 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores.
2. Se ingresan un conjunto de n alturas de personas por teclado. Mostrar la altura promedio de las personas.
3. En una empresa trabajan n empleados cuyos sueldos oscilan entre \$100 y \$500, realizar un programa que lea los sueldos que cobra cada empleado e informe cuántos empleados cobran entre \$100 y \$300 y cuántos cobran más de \$300. Además el programa deberá informar el importe que gasta la empresa en sueldos al personal.
4. Realizar un programa que imprima 25 términos de la serie 11 - 22 - 33 - 44, etc. (No se ingresan valores por teclado)
5. Mostrar los múltiplos de 8 hasta el valor 500. Debe aparecer en pantalla 8 - 16 - 24, etc.
6. Realizar un programa que permita cargar dos listas de 15 valores cada una. Informar con un mensaje cual de las dos listas tiene un valor acumulado mayor (mensajes "Lista 1 mayor", "Lista 2 mayor", "Listas iguales")  
Tener en cuenta que puede haber dos o más estructuras repetitivas en un algoritmo.
7. Desarrollar un programa que permita cargar n números enteros y luego nos informe cuántos valores fueron pares y cuántos impares.  
Emplear el operador `?%?` en la condición de la estructura condicional:

```
if (valor%2==0) //Si el if da verdadero luego
```

[Solución](#)

[Retornar](#)

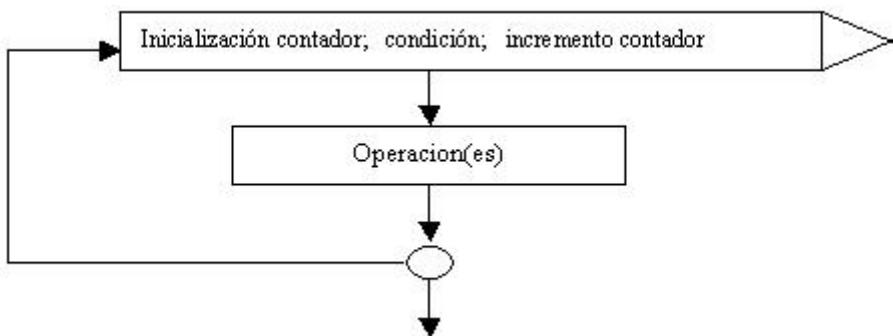
# 11 - Estructura repetitiva for

[Listado completo de tutoriales](#)

Cualquier problema que requiera una estructura repetitiva se puede resolver empleando la estructura while. Pero hay otra estructura repetitiva cuyo planteo es más sencillo en ciertas situaciones.

En general, la estructura for se usa en aquellas situaciones en las cuales CONOCEMOS la cantidad de veces que queremos que se ejecute el bloque de instrucciones. Ejemplo: cargar 10 números, ingresar 5 notas de alumnos, etc. Conocemos de antemano la cantidad de veces que queremos que el bloque se repita. Veremos, sin embargo, que en el lenguaje Java la estructura for puede usarse en cualquier situación repetitiva, porque en última instancia no es otra cosa que una estructura while generalizada.

Representación gráfica:

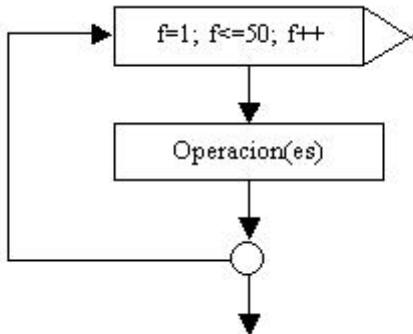


En su forma más típica y básica, esta estructura requiere una variable entera que cumple la función de un **CONTADOR** de vueltas. En la sección indicada como "inicialización contador", se suele colocar el nombre de la variable que hará de contador, asignándole a dicha variable un valor inicial. En la sección de "condición" se coloca la condición que deberá ser verdadera para que el ciclo continúe (en caso de un falso, el ciclo se detendrá). Y finalmente, en la sección de "incremento contador" se coloca una instrucción que permite modificar el valor de la variable que hace de contador (para permitir que alguna vez la condición sea falsa)

Cuando el ciclo comienza, antes de dar la primera vuelta, la variable del for toma el valor indicado en la sección de "inicialización contador". Inmediatamente se verifica, en forma automática, si la condición es verdadera. En caso de serlo se ejecuta el bloque de operaciones del ciclo, y al finalizar el mismo se ejecuta la instrucción que se haya colocado en la tercer sección.

Seguidamente, se vuelve a controlar el valor de la condición, y así prosigue hasta que dicha condición entregue un falso.

Si conocemos la cantidad de veces que se repite el bloque es muy sencillo emplear un for, por ejemplo si queremos que se repita 50 veces el bloque de instrucciones puede hacerse así:



La variable del for puede tener cualquier nombre. En este ejemplo se la ha definido con el nombre f.

Analicemos el ejemplo:

- La variable f toma inicialmente el valor 1.
- Se controla automáticamente el valor de la condición: como f vale que 50, la condición da verdadero.
- Como la condición fue verdadera, se ejecutan la/s operación/es.
- Al finalizar de ejecutarlas, se retorna a la instrucción f++, por variable f se incrementa en uno.
- Se vuelve a controlar (automáticamente) si f es menor o igual a 50. Como ahora su valor es 2, se ejecuta nuevamente el bloque de instrucción incrementa nuevamente la variable del for al terminar el mismo.
- El proceso se repetirá hasta que la variable f sea incrementada a 50. En este momento la condición será falsa, y el ciclo se detendrá.

La variable f PUEDE ser modificada dentro del bloque de operaciones del for, aunque esto podría causar problemas de lógica si el programador es inexperto.

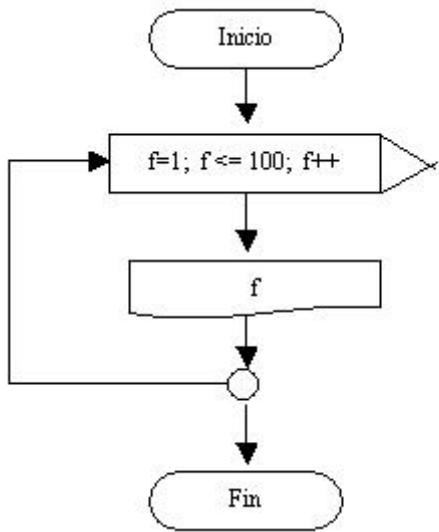
La variable f puede ser inicializada en cualquier valor y finalizar en cualquier valor. Además, no es obligatorio que la instrucción de modificación sea un incremento del tipo contador (f++).

Cualquier instrucción que modifique el valor de la variable es válida. Si por ejemplo se escribe  $f=f+2$  en lugar de  $f++$ , el valor de f será incrementado de a 2 en cada vuelta, y no de a 1. En este caso, esto significará que el ciclo no efectuará las 50 vueltas sino sólo 25.

## Problema 1:

Realizar un programa que imprima en pantalla los números del 1 al 100.

### Diagrama de flujo:



Podemos observar y comparar con el problema realizado con el while. Con la estructura while el CONTADOR x sirve para contar las vueltas. Con el for el CONTADOR f cumple dicha función.

Inicialmente f vale 1 y como no es superior a 100 se ejecuta el bloque, imprimimos el contenido de f, al finalizar el bloque repetitivo se incrementa la variable f en 1, como 2 no es superior a 100 se repite el bloque de instrucciones.

Cuando la variable del for llega a 101 sale de la estructura repetitiva y continúa la ejecución del algoritmo que se indica después del círculo.

La variable f (o como sea que se decida llamarla) debe estar definida como una variable más.

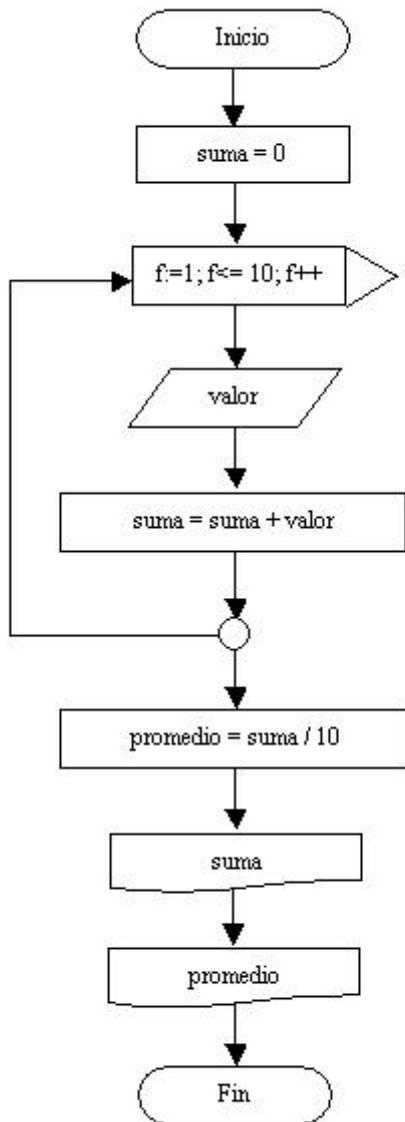
### Programa:

```
public class EstructuraRepetitivaFor1 {  
    public static void main(String[] ar) {  
        int f;  
        for(f=1;f<=100;f++) {  
            System.out.print(f);  
            System.out.print("-");  
        }  
    }  
}
```

### Problema 2:

: Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio. Este problema ya lo desarrollamos , lo resolveremos empleando la estructura for.

### Diagrama de flujo:



En este caso, a la variable del for (f) sólo se la requiere para que se repita el bloque de instrucciones 10 veces.

### Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaFor2 {
    public static void main (String [] args) {
        Scanner sc = new Scanner (System.in);
        int sum = 0;
        int f = 1;
        int promedio;
        System.out.println ("Cargue 10 valores");
        for (f=1; f<=10; f++) {
            System.out.print ("Cargue el valor " + f + ": ");
            int valor = sc.nextInt();
            sum = sum + valor;
        }
        promedio = sum / 10;
        System.out.println ("La suma es: " + sum);
        System.out.println ("El promedio es: " + promedio);
    }
}
```

```

public static void main(String[] ar) {
    Scanner teclado=new Scanner(System.in);
    int suma,f,valor,promedio;
    suma=0;
    for(f=1;f<=10;f++) {
        System.out.print("Ingrese valor:");
        valor=teclado.nextInt();
        suma=suma+valor;
    }
    System.out.print("La suma es:");
    System.out.println(suma);
    promedio=suma/10;
    System.out.print("El promedio es:");
    System.out.print(promedio);
}
}

```

El problema requiere que se carguen 10 valores y se sumen los mismos.

Tener en cuenta encerrar entre llaves bloque de instrucciones a repetir dentro del for.

El promedio se calcula fuera del for luego de haber cargado los 10 valores.

### Problema 3:

Escribir un programa que lea 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores.

Para resolver este problema se requieren tres contadores:

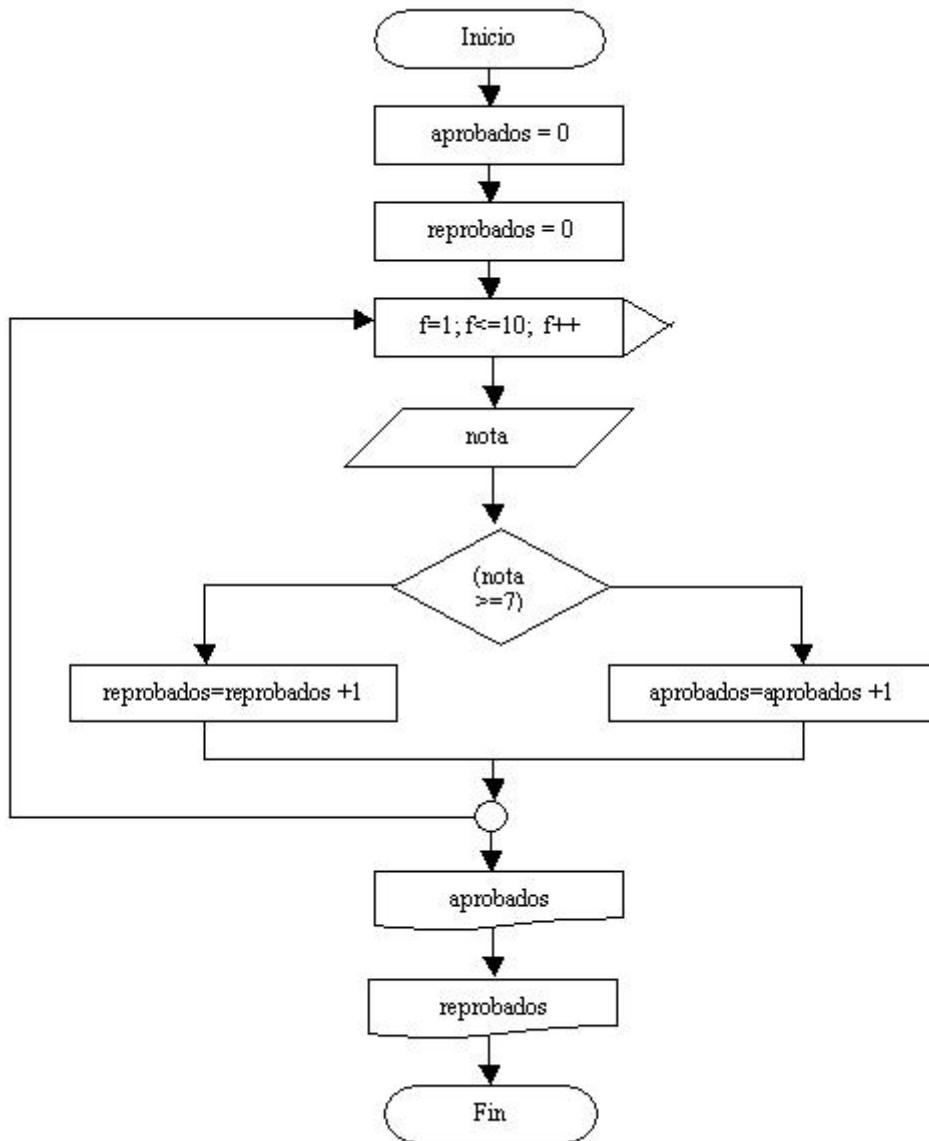
aprobados (Cuenta la cantidad de alumnos aprobados)

reprobados (Cuenta la cantidad de reprobados)

f (es el contador del for)

Dentro de la estructura repetitiva debemos hacer la carga de la variable nota y verificar con una estructura condicional si el contenido de la variable nota es mayor o igual a 7 para incrementar el contador aprobados, en caso de que la condición retorne falso debemos incrementar la variable reprobados.

### Diagrama de flujo:



Los contadores aprobados y reprobados deben imprimirse FUERA de la estructura repetitiva.

Es fundamental inicializar los contadores aprobados y reprobados en cero antes de entrar a la estructura for.

**Importante:** Un error común es inicializar los contadores dentro de la estructura repetitiva. En caso de hacer esto los contadores se fijan en cero en cada ciclo del for, por lo que al finalizar el for como máximo el contador puede tener el valor 1.

### Programa:

```

import java.util.Scanner;

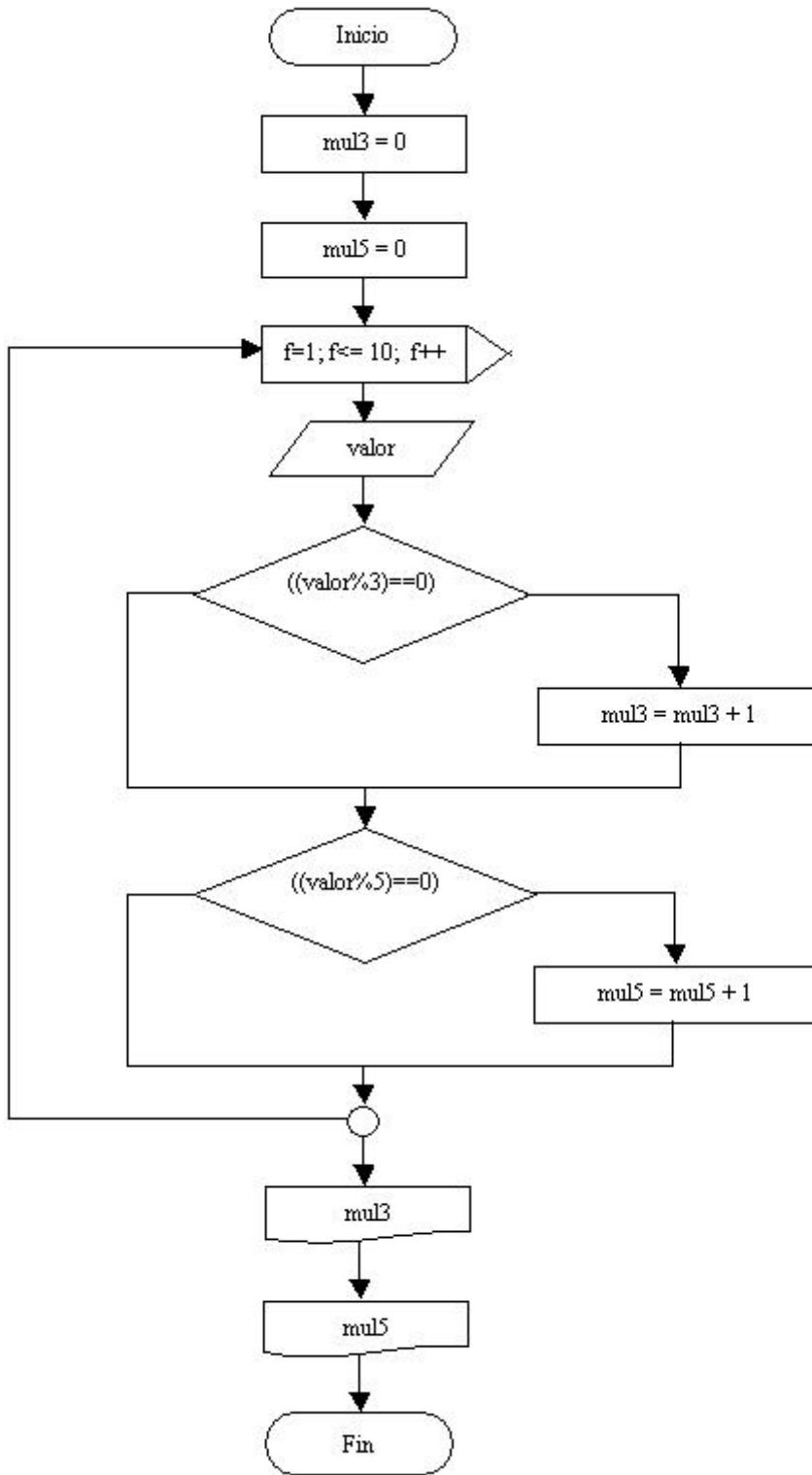
public class EstructuraRepetitivaFor3 {
    public static void main(String[] args) {
    }
  
```

```
public static void main(String[] ar) {
    Scanner teclado=new Scanner(System.in);
    int aprobados,reprobados,f,nota;
    aprobados=0;
    reprobados=0;
    for(f=1;f<=10;f++) {
        System.out.print("Ingrese la nota:");
        nota=teclado.nextInt();
        if (nota>=7) {
            aprobados=aprobados+1;
        } else {
            reprobados=reprobados+1;
        }
    }
    System.out.print("Cantidad de aprobados:");
    System.out.println(aprobados);
    System.out.print("Cantidad de reprobados:");
    System.out.print(reprobados);
}
```

#### Problema 4:

Escribir un programa que lea 10 números enteros y luego muestre cuántos valores ingresados fueron múltiplos de 3 y cuántos de 5. Debemos tener en cuenta que hay números que son múltiplos de 3 y de 5 a la vez.

#### Diagrama de flujo:



Tengamos en cuenta que el operador matemático `%` retorna el resto de dividir un valor por otro, en este caso: `valor%3` retorna el resto de dividir el valor que ingresamos por teclado, por tres.

Veamos: si ingresamos 6 el resto de dividirlo por 3 es 0, si ingresamos 12 el resto de dividirlo por 3 es 0. Generalizando: cuando el resto de dividir por 3 al valor que ingresamos por teclado es cero, se trata de un múltiplo de dicho valor.

Ahora bien ¿por qué no hemos dispuesto una estructura if anidada? Porque hay valores que son múltiplos de 3 y de 5 a la vez. Por lo tanto con if anidados no podríamos analizar los dos casos.

Es importante darse cuenta cuando conviene emplear if anidados y cuando no debe emplearse.

### Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaFor4 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int mul3,mul5,valor,f;
        mul3=0;
        mul5=0;
        for(f=1;f<=10;f++) {
            System.out.print("Ingrese un valor:");
            valor=teclado.nextInt();
            if (valor%3==0) {
                mul3=mul3+1;
            }
            if (valor%5==0) {
                mul5=mul5+1;
            }
        }
        System.out.print("Cantidad de valores ingresados:");
        System.out.println(mul3);
        System.out.print("Cantidad de valores ingresados:");
        System.out.print(mul5);
    }
}
```

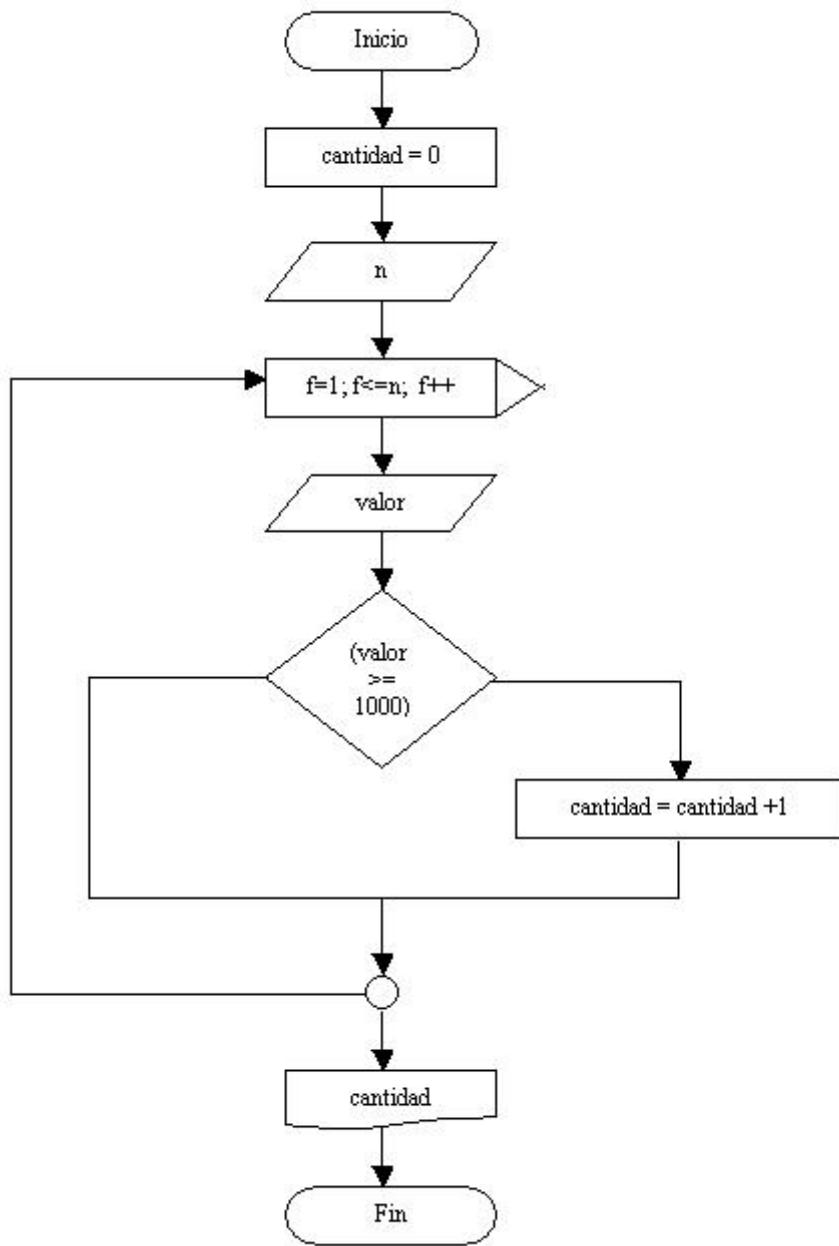
### Problema 5:

Escribir un programa que lea  $n$  números enteros y calcule la cantidad de valores mayores o iguales a 1000.

Este tipo de problemas también se puede resolver empleando la estructura repetitiva for. Lo primero que se hace es cargar una variable que indique la cantidad de valores a ingresar. Dicha variable se carga antes de entrar a la estructura repetitiva for.

La estructura for permite que el valor inicial o final dependa de una variable cargada previamente por teclado.

### Diagrama de flujo:



Tenemos un contador llamado cantidad y f que es el contador del for.

La variable entera n se carga previo al inicio del for, por lo que podemos fijar el valor final del for con la variable n.

Por ejemplo si el operador carga 5 en n la estructura repetitiva for se ejecutará 5 veces. La variable valor se ingresa dentro de la estructura repetitiva, y se verifica si el valor de la misma es mayor o igual a 1000, en dicho caso se incrementa en uno el contador cantidad.

Fuera de la estructura repetitiva imprimimos el contador cantidad que tiene almacenado la cantidad de valores ingresados mayores o iguales a 1000.

### Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaFor5 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int cantidad,n,f,valor;
        cantidad=0;
        System.out.print("Cuantos valores ingresará:");
        n=teclado.nextInt();
        for(f=1;f<=n;f++) {
            System.out.print("Ingrese el valor:");
            valor=teclado.nextInt();
            if (valor>=1000) {
                cantidad=cantidad+1;
            }
        }
        System.out.print("La cantidad de valores ingresados es:");
        System.out.print(cantidad);
    }
}
```

## Problemas propuestos

Ha llegado nuevamente la parte fundamental, que es el momento donde uno desarrolla individualmente un algoritmo para la resolución de un problema.

1. Confeccionar un programa que lea  $n$  pares de datos, cada par de datos corresponde a la medida de la base y la altura de un triángulo. El programa deberá informar:
  - a) De cada triángulo la medida de su base, su altura y su superficie.
  - b) La cantidad de triángulos cuya superficie es mayor a 12.
2. Desarrollar un programa que solicite la carga de 10 números e imprima la suma de los últimos 5 valores ingresados.
3. Desarrollar un programa que muestre la tabla de multiplicar del 5 (del 5 al 50)
4. Confeccionar un programa que permita ingresar un valor del 1 al 10 y nos muestre la tabla de multiplicar del mismo (los primeros 12 términos)

Ejemplo: Si ingreso 3 deberá aparecer en pantalla los valores 3, 6, 9, hasta el 36.

5. Realizar un programa que lea los lados de n triángulos, e informar:
  - a) De cada uno de ellos, qué tipo de triángulo es: equilátero (tres lados iguales), isósceles (dos lados iguales), o escaleno (ningún lado igual)
  - b) Cantidad de triángulos de cada tipo.
  - c) Tipo de triángulo que posee menor cantidad.
6. Escribir un programa que pida ingresar coordenadas (x,y) que representan puntos en el plano.  
Informar cuántos puntos se han ingresado en el primer, segundo, tercer y cuarto cuadrante. Al comenzar el programa se pide que se ingrese la cantidad de puntos a procesar.
7. Se realiza la carga de 10 valores enteros por teclado. Se desea conocer:
  - a) La cantidad de valores ingresados negativos.
  - b) La cantidad de valores ingresados positivos.
  - c) La cantidad de múltiplos de 15.
  - d) El valor acumulado de los números ingresados que son pares.
8. Se cuenta con la siguiente información:  
Las edades de 50 estudiantes del turno mañana.  
Las edades de 60 estudiantes del turno tarde.  
Las edades de 110 estudiantes del turno noche.  
Las edades de cada estudiante deben ingresarse por teclado.
  - a) Obtener el promedio de las edades de cada turno (tres promedios)
  - b) Imprimir dichos promedios (promedio de cada turno)
  - c) Mostrar por pantalla un mensaje que indique cual de los tres turnos tiene un promedio de edades mayor.

[Solución](#)

[\*\*Retornar\*\*](#)

# 12 - Estructura repetitiva do while

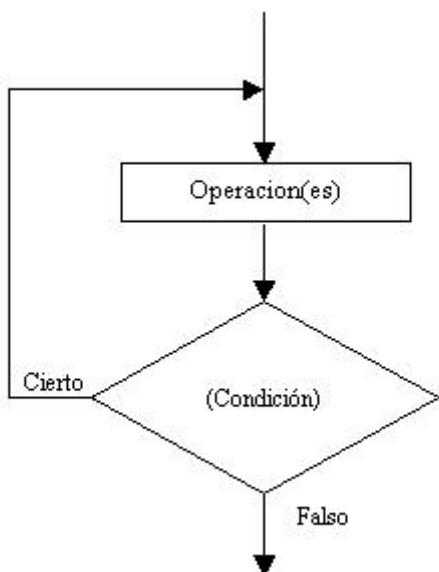
[Listado completo de tutoriales](#)

La estructura do while es otra estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo, a diferencia del while o del for que podían no ejecutar el bloque.

Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo.

La condición de la estructura está abajo del bloque a repetir, a diferencia del while o del for que está en la parte superior.

Representación gráfica:



El bloque de operaciones se repite MIENTRAS que la condición sea Verdadera.

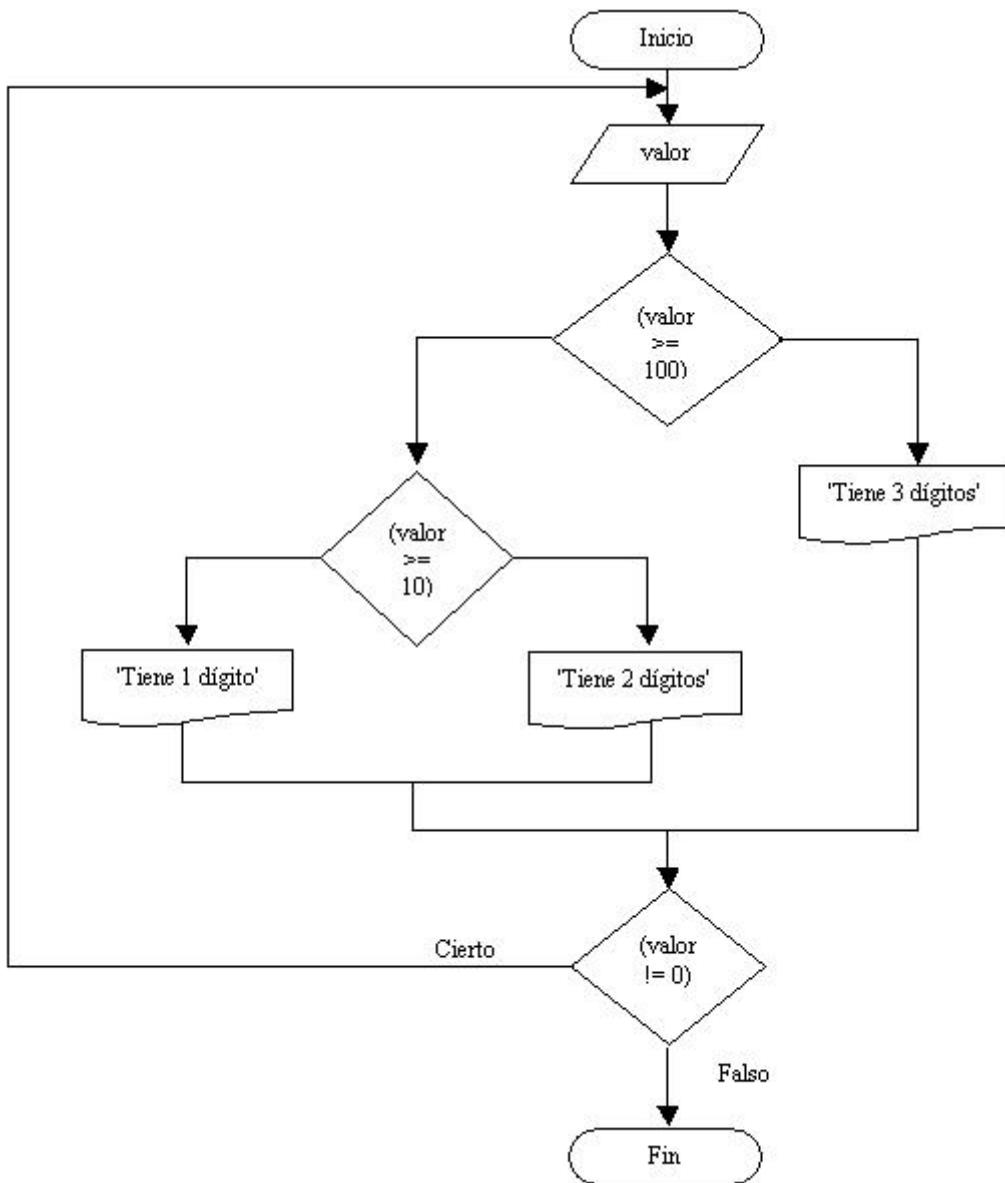
Si la condición retorna Falso el ciclo se detiene. En Java, todos los ciclos repiten por verdadero y cortan por falso.

Es importante analizar y ver que las operaciones se ejecutan como mínimo una vez.

## Problema 1:

Escribir un programa que solicite la carga de un número entre 0 y 999, y nos muestre un mensaje de cuántos dígitos tiene el mismo. Finalizar el programa cuando se cargue el valor 0.

## Diagrama de flujo:



No hay que confundir los rombos de las estructuras condicionales con los de las estructuras repetitivas do while.

En este problema por lo menos se carga un valor. Si se carga un valor mayor o igual a 100 se trata de un número de tres cifras, si es mayor o igual a 10 se trata de un valor de dos dígitos, en caso contrario se trata de un valor de un dígito. Este bloque se repite hasta que se ingresa en la variable valor el número 0 con lo que la condición de la estructura do while retorna falso y sale del bloque repetitivo finalizando el programa.

### Programa:

```

import java.util.Scanner;

public class EstructuraRepetitivaDoWhile1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int valor;
    }
}

```

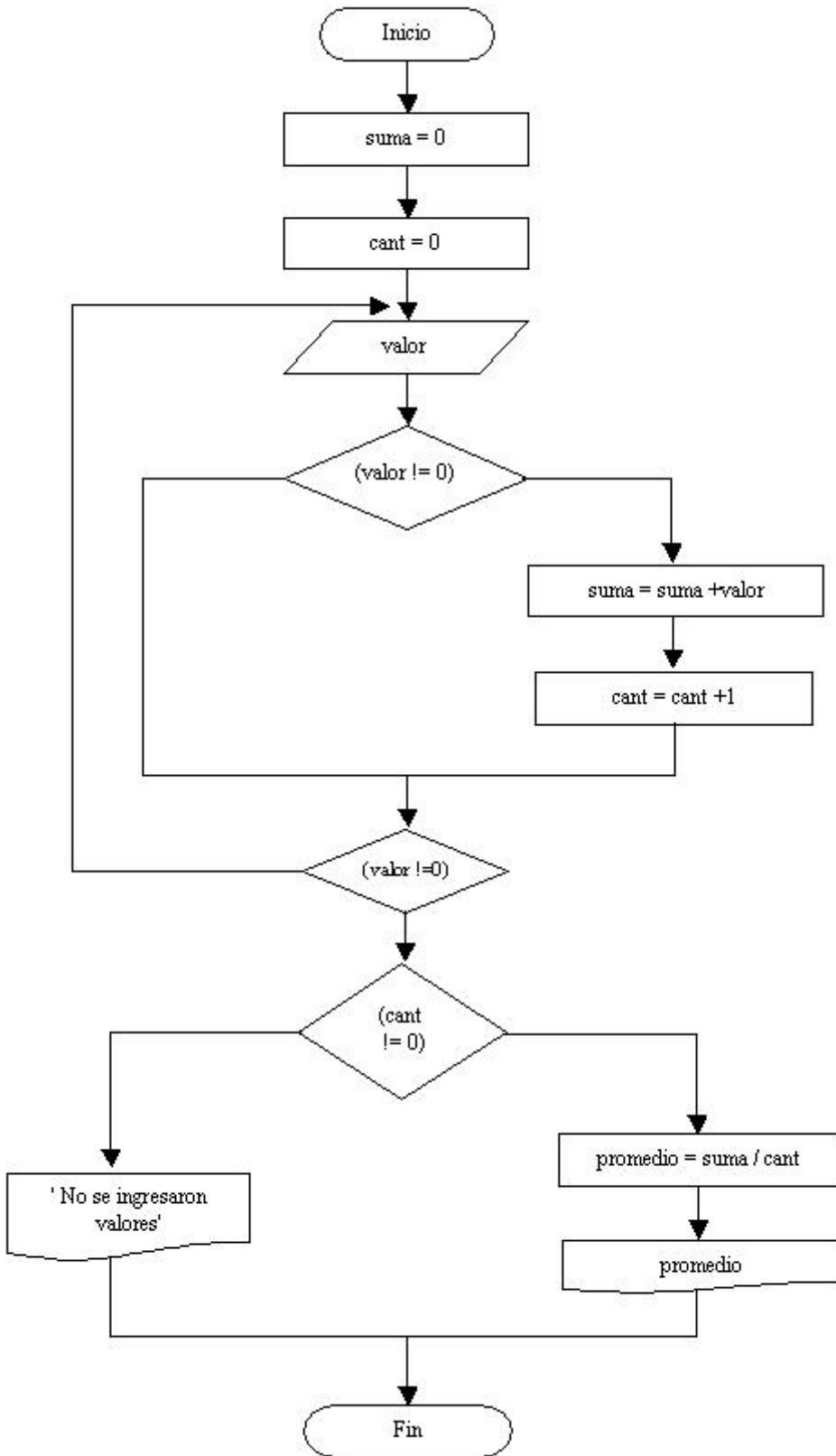
```
do {
    System.out.print("Ingrese un valor entre 0 y 999 (0
finaliza):");
    valor=teclado.nextInt();
    if (valor>=100) {
        System.out.println("Tiene 3 dígitos.");
    } else {
        if (valor>=10) {
            System.out.println("Tiene 2 dígitos.");
        } else {
            System.out.println("Tiene 1 dígito.");
        }
    }
} while (valor!=0);
}
```

## Problema 2:

Escribir un programa que solicite la carga de números por teclado, obtener su promedio. Finalizar la carga de valores cuando se cargue el valor 0.

Cuando la finalización depende de algún valor ingresado por el operador conviene el empleo de la estructura do while, por lo menos se cargará un valor (en el caso más extremo se carga 0, que indica la finalización de la carga de valores)

## Diagrama de flujo:



Es importante analizar este diagrama de flujo.

Definimos un contador **cant** que cuenta la cantidad de valores ingresados por el operador (no lo incrementa si ingresamos 0)

El valor 0 no es parte de la serie de valores que se deben sumar.

Definimos el acumulador suma que almacena todos los valores ingresados por teclado. La estructura repetitiva do while se repite hasta que ingresamos el valor 0. Con dicho valor la condición del ciclo retorna falso y continúa con el flujo del diagrama.

Disponemos por último una estructura condicional para el caso que el operador cargue únicamente un 0 y por lo tanto no podemos calcular el promedio ya que no existe la división por 0.

En caso que el contador cant tenga un valor distinto a 0 el promedio se obtiene dividiendo el acumulador suma por el contador cant que tiene la cantidad de valores ingresados antes de introducir el 0.

### Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaDoWhile2 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int suma,cant,valor,promedio;
        suma=0;
        cant=0;
        do {
            System.out.print("Ingrese un valor (0 para finalizar):");
            valor=teclado.nextInt();
            if (valor!=0) {
                suma=suma+valor;
                cant++;
            }
        } while (valor!=0);
        if (cant!=0) {
            promedio=suma/cant;
            System.out.print("El promedio de los valores ingresados es:");
            System.out.print(promedio);
        } else {
            System.out.print("No se ingresaron valores.");
        }
    }
}
```

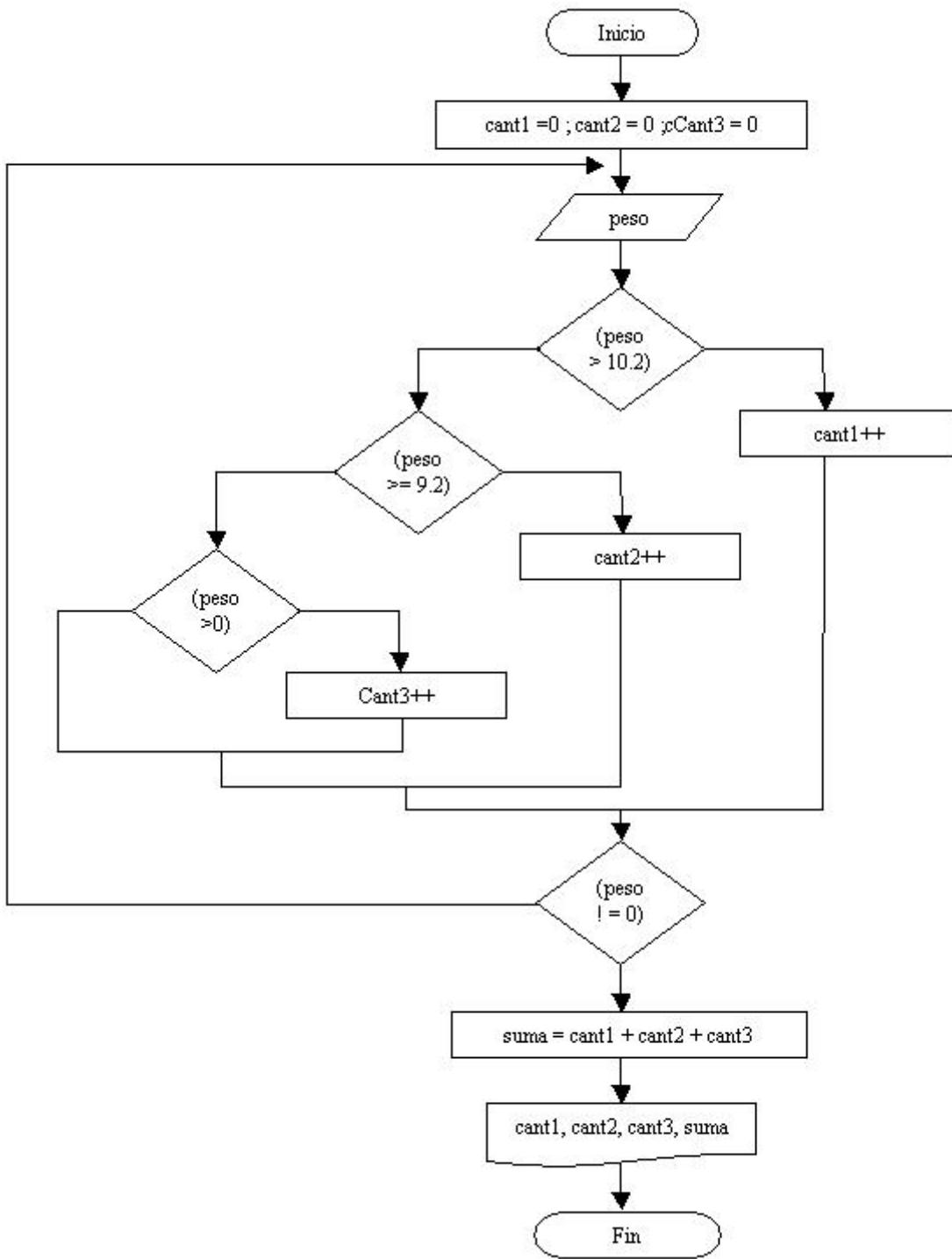
El contador cant DEBE inicializarse antes del ciclo, lo mismo que el acumulador suma. El promedio se calcula siempre y cuando el contador cant sea distinto a 0.

### Problema 3:

Realizar un programa que permita ingresar el peso (en kilogramos) de piezas. El proceso termina cuando ingresamos el valor 0. Se debe informar:

- Cuántas piezas tienen un peso entre 9.8 Kg. y 10.2 Kg.?, cuántas con más de 10.2 Kg.? y cuántas con menos de 9.8 Kg.?
- La cantidad total de piezas procesadas.

## Diagrama de flujo:



Los tres contadores **cont1**, **cont2**, y **cont3** se inicializan en 0 antes de entrar a la estructura repetitiva.

A la variable **suma** no se la inicializa en 0 porque no es un acumulador, sino que guarda la suma del contenido de las variables **cont1**, **cont2** y **cont3**.

La estructura se repite hasta que se ingresa el valor 0 en la variable **peso**. Este valor no

se lo considera un peso menor a 9.8 Kg., sino que indica que ha finalizado la carga de valores por teclado.

### Programa:

```
import java.util.Scanner;

public class EstructuraRepetitivaDoWhile3 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int cant1,cant2,cant3,suma;
        float peso;
        cant1=0;
        cant2=0;
        cant3=0;
        do {
            System.out.print("Ingrese el peso de la pieza (0 para
finalizar):");
            peso=teclado.nextFloat();
            if (peso>10.2) {
                cant1++;
            } else {
                if (peso>=9.8) {
                    cant2++;
                } else {
                    if (peso>0) {
                        cant3++;
                    }
                }
            }
        } while(peso!=0);
        suma=cant1+cant2+cant3;
        System.out.print("Piezas aptas:");
        System.out.println(cant2);
        System.out.print("Piezas con un peso superior a 10.2:");
        System.out.println(cant1);
        System.out.print("Piezas con un peso inferior a 9.8:");
        System.out.println(cant3);
    }
}
```

## Problemas propuestos

1. Realizar un programa que acumule (sume) valores ingresados por teclado hasta ingresar el 9999 (no sumar dicho valor, indica que ha finalizado la carga). Imprimir el valor acumulado e informar si dicho valor es cero, mayor a cero o menor a cero.
2. En un banco se procesan datos de las cuentas corrientes de sus clientes. De cada cuenta corriente se conoce: número de cuenta y saldo actual. El ingreso de datos debe finalizar al ingresar un valor negativo en el número de cuenta.  
Se pide confeccionar un programa que lea los datos de las cuentas corrientes e informe:  
a)De cada cuenta: número de cuenta y estado de la cuenta según su saldo, sabiendo que:

|                     |   |
|---------------------|---|
| Estado de la cuenta | 'Acreedor' si el saldo es >0.<br>'Deudor' si el saldo es <0.<br>'Nulo' si el saldo es =0. |
|---------------------|---|

b) La suma total de los saldos acreedores.

## Solución

## Retornar

# 13 - Cadenas de caracteres en Java

[Listado completo de tutoriales](#)

En Java hemos visto que cuando queremos almacenar un valor entero definimos una variable de tipo int, si queremos almacenar un valor con decimales definimos una variable de tipo float. Ahora si queremos almacenar una cadena de caracteres (por ejemplo un nombre de una persona) debemos definir un objeto de la clase String.

Más adelante veremos en profundidad y detenimiento los conceptos de CLASE y OBJETO, por ahora solo nos interesa la mecánica para trabajar con cadenas de caracteres.

## Problema 1:

Solicitar el ingreso del nombre y edad de dos personas. Mostrar el nombre de la persona con mayor edad.

### Programa:

```
import java.util.Scanner;

public class CadenaDeCaracteres1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        String nombre1,nombre2;
        int edad1,edad2;
        System.out.print("Ingrese el nombre:");
        nombre1=teclado.next();
        System.out.print("Ingrese edad:");
        edad1=teclado.nextInt();
        System.out.print("Ingrese el nombre:");
        nombre2=teclado.next();
        System.out.print("Ingrese edad:");
        edad2=teclado.nextInt();
        System.out.print("La persona de mayor edad es:");
        if (edad1>edad2) {
            System.out.print(nombre1);
        } else {
```

```
        System.out.print(nombre2);
    }
}
```

Para almacenar un nombre debemos definir una variable de tipo String y su ingreso por teclado se hace llamando al método next() del objeto teclado:

```
nombre1=teclado.next();
```

La primera salvedad que tenemos que hacer cuando utilizamos el método next() es que solo nos permite ingresar una cadena de caracteres con la excepción del espacio en blanco (es decir debemos ingresar un nombre de persona y no su nombre y apellido separado por un espacio en blanco)

Veamos que existe otro método llamado nextLine() que nos permite cargar espacios en blanco pero para su uso se complica cuando cargamos otras valores de tipo distinto a String (por ejemplo int, float etc.)

## Problema 2:

Solicitar el ingreso del apellido, nombre y edad de dos personas. Mostrar el nombre de la persona con mayor edad. Realizar la carga del apellido y nombre en una variable de tipo String.

### Programa:

```
import java.util.Scanner;

public class CadenaDeCaracteres2 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        String apenom1,apenom2;
        int edad1,edad2;
        System.out.print("Ingrese el apellido y el nombre:");
        apenom1=teclado.nextLine();
        System.out.print("Ingrese edad:");
        edad1=teclado.nextInt();
        System.out.print("Ingrese el apellido y el nombre:");
        teclado.nextLine();
        apenom2=teclado.nextLine();
```

```

        System.out.print("Ingrese edad:");
        edad2=teclado.nextInt();
        System.out.print("La persona de mayor edad es:");
        if (edad1>edad2) {
            System.out.print(apenom1);
        } else {
            System.out.print(apenom2);
        }
    }
}

```

Cuando se ingresa una cadena con caracteres en blanco debemos tener en cuenta en llamar al método `nextLine()`

Una dificultad se presenta si llamamos al método `nextLine()` y previamente hemos llamado al método `nextInt()`, esto debido a que luego de ejecutar el método `nextInt()` queda almacenado en el objeto de la clase `Scanner` el carácter "Enter" y si llamamos inmediatamente al método `nextLine()` este almacena dicho valor de tecla y continúa con el flujo del programa. Para solucionar este problema debemos generar un código similar a:

```

System.out.print("Ingrese edad:");
edad1=teclado.nextInt();
System.out.print("Ingrese el apellido y el nombre:");
teclado.nextLine();
apenom2=teclado.nextLine();

```

Como vemos llamamos al método `nextLine()` dos veces, la primera retorna la tecla "Enter" y la segunda se queda esperando que ingresemos el apellido y nombre (tener en cuenta que esto es necesario solo si previamente se llamó al método `nextInt()` o `nextFloat()`).

### Problema 3:

Solicitar el ingreso de dos apellidos. Mostrar un mensaje si son iguales o distintos.

### Programa:

```

import java.util.Scanner;

public class CadenaDeCaracteres3 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);

```

```
String apellido1,apellido2;
System.out.print("Ingrese primer apellido:");
apellido1=teclado.next();
System.out.print("Ingrese segundo apellido:");
apellido2=teclado.next();
if (apellido1.equals(apellido2)) {
    System.out.print("Los apellidos son iguales")
} else {
    System.out.print("Los apellidos son distintos")
}
}
```

Para comparar si el contenido de dos String son iguales no podemos utilizar el operador `==`. Debemos utilizar un método de la clase String llamado `equals` y pasar como parámetro el String con el que queremos compararlo:

```
if (apellido1.equals(apellido2)) {
```

El método `equals` retorna verdadero si los contenidos de los dos String son exactamente iguales, esto hace que se ejecute el bloque del verdadero.

Recordemos que hemos utilizado el método `next()` para la carga de los String, luego esto hace que no podamos ingresar un apellido con espacios en blanco (podemos probar que si ingresamos por ejemplo "Rodriguez Rodriguez" en el primer apellido, luego se carga la cadena "Rodriguez" en la variable `apellido1` y "Rodriguez" en la variable `apellido2` (con esto hacemos notar que cada vez que ingresamos un espacio en blanco cuando utilizamos el método `next()` los caracteres que siguen al espacio en blanco son recuperados en la siguiente llamada al método `next()`)

El método `equals` retorna verdadero si los contenidos de los dos String son exactamente iguales, es decir si cargamos "Martinez" en `apellido1` y "martinez" en `apellido2` luego el método `equals` retorna falso ya que no es lo mismo la "M" mayúscula y la "m" minúscula.

En el caso que necesitemos considerar igual caracteres mayúsculas y minúsculas podemos utilizar el método `equalsIgnoreCase`:

```
if (apellido1.equalsIgnoreCase(apellido2)) {
    System.out.print("Los apellidos son iguales sin tener en cuenta el caso")
} else {
    System.out.print("Los apellidos son distintos sin tener en cuenta el caso")
}
```

[Retornar](#)

# 14 - Declaración de una clase y definición de objetos.

[Listado completo de tutoriales](#)

La programación orientada a objetos se basa en la programación de clases; a diferencia de la programación estructurada, que está centrada en las funciones.

Una clase es un molde del que luego se pueden crear múltiples objetos, con similares características.

Una clase es una plantilla (molde), que define atributos (variables) y métodos (funciones)

La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

Debemos crear una clase antes de poder crear objetos (instancias) de esa clase. Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.

La estructura de una clase es:

```
class [nombre de la clase] {  
    [atributos o variables de la clase]  
    [métodos o funciones de la clase]  
    [main]  
}
```

## Problema 1:

Confeccionar una clase que permita cargar el nombre y la edad de una persona. Mostrar los datos cargados. Imprimir un mensaje si es mayor de edad (edad>=18)

## Programa:

```
import java.util.Scanner;  
public class Persona {  
    private Scanner teclado;  
    private String nombre;  
    private int edad;  
  
    public void inicializar() {  
        teclado=new Scanner(System.in);  
        System.out.print("Ingrese nombre:");  
        nombre=teclado.next();  
        System.out.print("Ingrese edad:");
```

```

        edad=teclado.nextInt();
    }

    public void imprimir() {
        System.out.println("Nombre:" + nombre);
        System.out.println("Edad:" + edad);
    }

    public void esMayorEdad() {
        if (edad >= 18) {
            System.out.print(nombre + " es mayor de edad.");
        } else {
            System.out.print(nombre + " no es mayor de edad.");
        }
    }

    public static void main(String[] ar) {
        Persona personal;
        personal=new Persona();
        personal.inicializar();
        personal.imprimir();
        personal.esMayorEdad();
    }
}

```

El nombre de la clase debe hacer referencia al concepto (en este caso la hemos llamado Persona):

```
public class Persona {
```

Los atributos los definimos dentro de la clase pero fuera de la main:

```

private Scanner teclado;
private String nombre;
private int edad;

```

Veremos más adelante que un atributo es normalmente definido con la cláusula `private` (con esto no permitimos el acceso al atributo desde otras clases)

A los atributos se tiene acceso desde cualquier función o método de la clase (salvo la main)

Luego de definir los atributos de la clase debemos declarar los métodos o funciones de la clase. La sintaxis es parecida a la main (sin la cláusula `static`):

```

public void inicializar() {
    teclado=new Scanner(System.in);
    System.out.print("Ingrese nombre:");
    nombre=teclado.next();
    System.out.print("Ingrese edad:");
    edad=teclado.nextInt();
}

```

En el método inicializar (que será el primero que deberemos llamar desde la main) creamos el objeto de la clase Scanner y cargamos por teclado los atributos nombre y edad. Como podemos ver el método inicializar puede hacer acceso a los tres atributos de la clase Persona.

El segundo método tiene por objetivo imprimir el contenido de los atributos nombre y edad (los datos de los atributos se cargaron al ejecutarse previamente el método inicializar):

```
public void imprimir() {  
    System.out.println("Nombre: "+nombre);  
    System.out.println("Edad: "+edad);  
}
```

El tercer método tiene por objetivo mostrar un mensaje si la persona es mayor o no de edad:

```
public void esMayorEdad() {  
    if (edad>=18) {  
        System.out.print(nombre+" es mayor de edad.");  
    } else {  
        System.out.print(nombre+" no es mayor de edad.");  
    }  
}
```

Por último en la main declaramos un objeto de la clase Persona y llamamos a los métodos en un orden adecuado:

```
public static void main(String[] ar) {  
    Persona personal;  
    personal=new Persona();  
    personal.inicializar();  
    personal.imprimir();  
    personal.esMayorEdad();  
}
```

```
Persona persona1; //Declaración del objeto  
persona1=new Persona(); //Creación del objeto  
persona1.inicializar(); //Llamada de un método
```

## Problema 2:

Desarrollar un programa que cargue los lados de un triángulo e implemente los siguientes métodos: inicializar los atributos, imprimir el valor del lado mayor y otro método que muestre si es equilátero o no.

## Programa:

```

import java.util.Scanner;
public class Triangulo {
    private Scanner teclado;
    private int lado1,lado2,lado3;

    public void inicializar() {
        teclado=new Scanner(System.in);
        System.out.print("Medida lado 1:");
        lado1=teclado.nextInt();
        System.out.print("Medida lado 2:");
        lado2=teclado.nextInt();
        System.out.print("Medida lado 3:");
        lado3=teclado.nextInt();
    }

    public void ladoMayor() {
        System.out.print("Lado mayor:");
        if (lado1>lado2 && lado1>lado3) {
            System.out.println(lado1);
        } else {
            if (lado2>lado3) {
                System.out.println(lado2);
            } else {
                System.out.println(lado3);
            }
        }
    }

    public void esEquilatero() {
        if (lado1==lado2 && lado1==lado3) {
            System.out.print("Es un triángulo equilátero");
        } else {
            System.out.print("No es un triángulo equilátero");
        }
    }

    public static void main(String []ar) {
        Triangulo triangulol=new Triangulo();
        triangulol.inicializar();
        triangulol.ladoMayor();
        triangulol.esEquilatero();
    }
}

```

Todos los problemas que requieran la entrada de datos por teclado debemos definir un atributo de la clase Scanner:

```
private Scanner teclado;
```

Este problema requiere definir tres atributos de tipo entero donde almacenamos los valores de los lados del triángulo:

```
private int lado1,lado2,lado3;
```

El primer método que deberá llamarse desde la main es el inicializar donde creamos el objeto de la clase Scanner y cargamos los tres atributos por teclado:

```
public void inicializar() {  
    teclado=new Scanner(System.in);  
    System.out.print("Medida lado 1:");  
    lado1=teclado.nextInt();  
    System.out.print("Medida lado 2:");  
    lado2=teclado.nextInt();  
    System.out.print("Medida lado 3:");  
    lado3=teclado.nextInt();  
}
```

El método ladoMayor muestra el valor mayor de los tres enteros ingresados:

```
public void ladoMayor() {  
    System.out.print("Lado mayor:");  
    if (lado1>lado2 && lado1>lado3) {  
        System.out.println(lado1);  
    } else {  
        if (lado2>lado3) {  
            System.out.println(lado2);  
        } else {  
            System.out.println(lado3);  
        }  
    }  
}
```

Como podemos observar cuando un problema se vuelve más complejo es más fácil y ordenado separar los distintos algoritmos en varios métodos y no codificar todo en la main.

El último método de esta clase verifica si los tres enteros ingresados son iguales:

```
public void esEquilatero() {  
    if (lado1==lado2 && lado1==lado3) {  
        System.out.print("Es un triángulo equilátero");  
    } else {  
        System.out.print("No es un triángulo equilátero");  
    }  
}
```

En la main creamos un objeto de la clase Triangulo y llamamos los métodos respectivos:

```
public static void main(String []ar) {  
    Triangulo triangulo1=new Triangulo();  
    triangulo1.inicializar();  
    triangulo1.ladoMayor();  
    triangulo1.esEquilatero();  
}
```

### Problema 3:

Desarrollar una clase que represente un punto en el plano y tenga los siguientes métodos: cargar los valores de x e y, imprimir en que cuadrante se encuentra dicho punto (concepto matemático, primer cuadrante si x e y son positivas, si x<0 e y>0 segundo cuadrante, etc.)

### Programa:

```
import java.util.Scanner;
public class Punto {
    private Scanner teclado;
    int x,y;

    public void inicializar() {
        teclado=new Scanner(System.in);
        System.out.print("Ingrese coordenada x :");
        x=teclado.nextInt();
        System.out.print("Ingrese coordenada y :");
        y=teclado.nextInt();
    }

    void imprimirCuadrante() {
        if (x>0 && y>0) {
            System.out.print("Se encuentra en el primer cuadrante.");
        } else {
            if (x<0 && y>0) {
                System.out.print("Se encuentra en el segundo cuadrante.");
            } else {
                if (x<0 && y<0) {
                    System.out.print("Se encuentra en el tercer
cuadrante.");
                } else {
                    if (x>0 && y<0) {
                        System.out.print("Se encuentra en el cuarto
cuadrante.");
                    } else {
                        System.out.print("El punto no está en un
cuadrante.");
                    }
                }
            }
        }
    }

    public static void main(String[] ar) {
        Punto punto1;
        punto1=new Punto();
        punto1.inicializar();
        punto1.imprimirCuadrante();
    }
}
```

Definimos tres atributos (el objeto de la clase Scanner y los dos enteros donde almacenamos la coordenada x e y del punto:

```
private Scanner teclado;
int x,y;
```

El método inicializar crea el objeto de la clase Scanner y pide cargar las coordenadas x e y:

```
public void inicializar() {
    teclado=new Scanner(System.in);
    System.out.print("Ingrese coordenada x :");
    x=teclado.nextInt();
    System.out.print("Ingrese coordenada y :");
    y=teclado.nextInt();
}
```

El segundo método mediante un conjunto de if verificamos en que cuadrante se encuentra el punto ingresado:

```
void imprimirCuadrante() {
    if (x>0 && y>0) {
        System.out.print("Se encuentra en el primer cuadrante.");
    } else {
        if (x<0 && y>0) {
            System.out.print("Se encuentra en el segundo cuadrante.");
        } else {
            if (x<0 && y<0) {
                System.out.print("Se encuentra en el tercer
cuadrante.");
            } else {
                if (x>0 && y<0) {
                    System.out.print("Se encuentra en el cuarto
cuadrante.");
                } else {
                    System.out.print("El punto no está en un
cuadrante.");
                }
            }
        }
    }
}
```

La main no tiene grandes diferencias con los problemas realizados anteriormente, declaramos un objeto de la clase Punto, creamos el objeto mediante el operador new y seguidamente llamamos a los métodos inicializar e imprimirCuadrante en ese orden:

```
public static void main(String[] ar) {
    Punto punto1;
    punto1=new Punto();
    punto1.inicializar();
    punto1.imprimirCuadrante();
}
```

## Problema 4:

Desarrollar una clase que represente un Cuadrado y tenga los siguientes métodos: cargar el valor de su lado, imprimir su perímetro y su superficie.

### Programa:

```
import java.util.Scanner;
public class Cuadrado {
    private Scanner teclado;
    int lado;

    public void inicializar() {
        teclado=new Scanner(System.in);
        System.out.print("Ingrese valor del lado :");
        lado=teclado.nextInt();
    }

    public void imprimirPerimetro() {
        int perimetro;
        perimetro=lado*4;
        System.out.println("El perímetro es:"+perimetro);
    }

    public void imprimirSuperficie() {
        int superficie;
        superficie=lado*lado;
        System.out.println("La superficie es:"+superficie);
    }

    public static void main(String[] ar) {
        Cuadrado cuadrado1;
        cuadrado1=new Cuadrado();
        cuadrado1.inicializar();
        cuadrado1.imprimirPerimetro();
        cuadrado1.imprimirSuperficie();
    }
}
```

En este problema es interesante ver como no definimos dos atributos donde se almacenan la superficie y el perímetro del cuadrado, esto debido a que solo estos datos se los requiere en el método donde se imprimen:

```
public void imprimirPerimetro() {
    int perimetro;
    perimetro=lado*4;
    System.out.println("El perímetro es:"+perimetro);
}
```

Esto significa que la variable `perimetro` es una variable local al método `imprimirPerimetro`. Esta variable es local a dicho método y solo se la puede acceder dentro del método. La diferencia fundamental entre una variable local y un atributo de

la clase es que al atributo se lo puede acceder desde cualquier método de la clase y la variable local solo existe mientras se está ejecutando el método.

## Problemas propuestos

1. Confeccionar una clase que represente un empleado. Definir como atributos su nombre y su sueldo. Confeccionar los métodos para la carga, otro para imprimir sus datos y por último uno que imprima un mensaje si debe pagar impuestos (si el sueldo supera a 3000)
2. Implementar la clase operaciones. Se deben cargar dos valores enteros, calcular su suma, resta, multiplicación y división, cada una en un método, imprimir dichos resultados.

[Solución](#)

[Retornar](#)

# 15 - Declaración de métodos.

[Listado completo de tutoriales](#)

Cuando uno plantea una clase en lugar de especificar todo el algoritmo en un único método (lo que hicimos en los primeros pasos de este tutorial) es dividir todas las responsabilidades de la clase en un conjunto de métodos.

Un método hemos visto que tiene la siguiente sintaxis:

```
public void [nombre del método]() {  
    [algoritmo]  
}
```

Veremos que hay varios tipos de métodos:

## Métodos con parámetros.

Un método puede tener parámetros:

```
public void [nombre del método]([parámetros]) {  
    [algoritmo]  
}
```

Los parámetros los podemos imaginar como variables locales al método, pero su valor se inicializa con datos que llegan cuando lo llamamos.

## Problema 1:

Confeccionar una clase que permita ingresar valores enteros por teclado y nos muestre la tabla de multiplicar de dicho valor. Finalizar el programa al ingresar el -1.

## Programa:

```
import java.util.Scanner;  
public class TablaMultiplicar {  
    public void cargarValor() {  
        Scanner teclado=new Scanner(System.in);  
        int valor;  
        do {  
            System.out.print("Ingrese valor:");  
            valor=teclado.nextInt();  
        } while (valor>0);  
        for (int i=1; i<=10; i++) {  
            System.out.println(valor + " x " + i + " = " + (valor*i));  
        }  
    }  
}
```

```

        valor=scanner.nextInt();
        if (valor!=-1) {
            calcular(valor);
        }
    } while (valor!=-1);
}

public void calcular(int v) {
    for(int f=v;f<=v*10;f=f+v) {
        System.out.print(f+"-");
    }
}

public static void main(String[] ar) {
    TablaMultiplicar tabla;
    tabla=new TablaMultiplicar();
    tabla.cargarValor();
}
}

```

En esta clase no hemos definido ningún atributo, ya que el objeto de la clase Scanner lo requerimos en un solo método, por ello lo definimos como una variable local.

El método calcular recibe un parámetro de tipo entero, luego lo utilizamos dentro del método para mostrar la tabla de multiplicar de dicho valor, para esto inicializamos la variable f con el valor que llega en el parámetro. Luego de cada ejecución del for incrementamos el contador f con el valor de v.

```

public void calcular(int v) {
    for(int f=v;f<=v*10;f=f+v) {
        System.out.print(f+"-");
    }
}

```

Un método puede no tener parámetros como hemos visto en problemas anteriores o puede tener uno o más parámetros (en caso de tener más de un parámetro los mismos se separan por coma)

El método cargarValores no tiene parámetros y tiene por objetivo cargar un valor entero por teclado y llamar al método calcular para que muestre la tabla de multiplicar del valor que le pasamos por teclado:

```

public void cargarValor() {
    Scanner teclado=new Scanner(System.in);
    int valor;
    do {
        System.out.print("Ingrese valor:");
        valor=teclado.nextInt();
        if (valor!=-1) {
            calcular(valor);
        }
    } while (valor!=-1);
}

```

Como vemos al método calcular lo llamamos por su nombre y entre paréntesis le pasamos el dato a enviar (debe ser un valor o variable entera)

En este problema en la main solo llamamos al método cargarValor, ya que el método calcular luego es llamado por el método cargarValor:

```

public static void main(String[] ar) {
    TablaMultiplicar tabla;
    tabla=new TablaMultiplicar();
    tabla.cargarValor();
}

```

## Métodos que retornan un dato.

Un método puede retornar un dato:

```

public [tipo de dato] [nombre del método]([parámetros]) {
    [algoritmo]
    return [tipo de dato]
}

```

Cuando un método retorna un dato en vez de indicar la palabra clave void previo al nombre del método indicamos el tipo de dato que retorna. Luego dentro del algoritmo en el momento que queremos que finalice el mismo y retorne el dato empleamos la palabra clave return con el valor respectivo.

## Problema 2:

Confeccionar una clase que permita ingresar tres valores por teclado. Luego mostrar el mayor y el menor.

### Programa:

```
import java.util.Scanner;
public class MayorMenor {
    public void cargarValores() {
        Scanner teclado=new Scanner(System.in);
        System.out.print("Ingrese primer valor:");
        int valor1=teclado.nextInt();
        System.out.print("Ingrese segundo valor:");
        int valor2=teclado.nextInt();
        System.out.print("Ingrese tercer valor:");
        int valor3=teclado.nextInt();
        int mayor,menor;
        mayor=calcularMayor(valor1,valor2,valor3);
        menor=calcularMenor(valor1,valor2,valor3);
        System.out.println("El valor mayor de los tres");
        System.out.println("El valor menor de los tres");
    }

    public int calcularMayor(int v1,int v2,int v3) {
        int m;
        if(v1>v2 && v1>v3) {
            m=v1;
        } else {
            if(v2>v3) {
                m=v2;
            } else {
                m=v3;
            }
        }
        return m;
    }

    public int calcularMenor(int v1,int v2,int v3) {
        int m;
        if(v1<v2 && v1<v3) {
            m=v1;
        } else {
            if(v2<v3) {
                m=v2;
            } else {
                m=v3;
            }
        }
    }
}
```

```

        return m;
    }

    public static void main(String[] ar) {
        MayorMenor maymen=new MayorMenor();
        maymen.cargarValores();
    }
}

```

Si vemos la sintaxis que calcula el mayor de tres valores enteros es similar al algoritmo visto en conceptos anteriores:

```

public int calcularMayor(int v1,int v2,int v3) {
    int m;
    if(v1>v2 && v1>v3) {
        m=v1;
    } else {
        if(v2>v3) {
            m=v2;
        } else {
            m=v3;
        }
    }
    return m;
}

```

Lo primero que podemos observar que el método retorna un entero y recibe tres parámetros:

```
public int calcularMayor(int v1,int v2,int v3) {
```

Dentro del método verificamos cual de los tres parámetros almacena un valor mayor, a este valor lo almacenamos en una variable local llamada "m", al valor almacenado en esta variable lo retornamos al final con un return.

La llamada al método calcularMayor lo hacemos desde dentro del método cargarCalores:

```
mayor=calcularMayor(valor1,valor2,valor3);
```

Debemos asignar a una variable el valor devuelto por el método calcularMayor. Luego el contenido de la variable mayor lo mostramos:

```
System.out.println("El valor mayor de los tres es:"+mayor);
```

La lógica es similar para el cálculo del menor.

[Retornar](#)

# 16 - Estructura de datos tipo vector.

[Listado completo de tutoriales](#)

Hemos empleado variables de distinto tipo para el almacenamiento de datos (variables int, float, String) En esta sección veremos otros tipos de variables que permiten almacenar un conjunto de datos en una única variable.

Un vector es una estructura de datos que permite almacenar un CONJUNTO de datos del MISMO tipo.

Con un único nombre se define un vector y por medio de un subíndice hacemos referencia a cada elemento del mismo (componente)

## Problema 1:

Se desea guardar los sueldos de 5 operarios.

Según lo conocido deberíamos definir 5 variables si queremos tener en un cierto momento los 5 sueldos almacenados en memoria.

Empleando un vector solo se requiere definir un único nombre y accedemos a cada elemento por medio del subíndice.

| sueldos    |            |            |            |            |
|------------|------------|------------|------------|------------|
| 1200       | 750        | 820        | 550        | 490        |
| sueldos[0] | sueldos[1] | sueldos[2] | sueldos[3] | sueldos[4] |

## Programa:

```
import java.util.Scanner;
public class PruebaVector1 {
    private Scanner teclado;
    private int[] sueldos;

    public void cargar()
    {
        teclado=new Scanner(System.in);
        sueldos=new int[5];
        for(int f=0;f<5;f++)
        {
            System.out.print("Ingrese valor de la componente:");
            sueldos[f]=teclado.nextInt();
        }
    }
}
```

```

public void imprimir() {
    for(int f=0;f<5;f++) {
        System.out.println(sueldos[f]);
    }
}

public static void main(String[] ar) {
    PruebaVector1 pv=new PruebaVector1();
    pv.cargar();
    pv.imprimir();
}
}

```

Para la declaración de un vector le antecedemos al nombre los corchetes abiertos y cerrados:

```
private int[] sueldos;
```

Lo definimos como atributo de la clase ya que lo utilizaremos en los dos métodos.

En el método de cargar lo primero que hacemos es crear el vector (en java los vectores son objetos por lo que es necesario proceder a su creación mediante el operador new):

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

Cuando creamos el vector indicamos entre corchetes la cantidad de elementos que se pueden almacenar posteriormente en el mismo.

Para cargar cada componente debemos indicar entre corchetes que elemento del vector estamos accediendo:

```

for(int f=0;f<5;f++) {
    System.out.print("Ingrese valor de la componente:");
    sueldos[f]=teclado.nextInt();
}

```

La estructura de programación que más se adapta para cargar en forma completa las componentes de un vector es un for, ya que sabemos de antemano la cantidad de valores a cargar.

Cuando f vale cero estamos accediendo a la primer componente del vector (en nuestro caso sería):

```
sueldos[0]=teclado.nextInt();
```

Lo mas común es utilizar una estructura repetitiva for para recorrer cada componente del vector.

Utilizar el for nos reduce la cantidad de código, si no utilizo un for debería en forma secuencial implementar el siguiente código:

```
System.out.print("Ingrese valor de la componente:");
sueldos[0]=teclado.nextInt();
System.out.print("Ingrese valor de la componente:");
sueldos[1]=teclado.nextInt();
System.out.print("Ingrese valor de la componente:");
sueldos[2]=teclado.nextInt();
System.out.print("Ingrese valor de la componente:");
sueldos[3]=teclado.nextInt();
System.out.print("Ingrese valor de la componente:");
sueldos[4]=teclado.nextInt();
```

La impresión de las componentes del vector lo hacemos en el otro método:

```
public void imprimir() {
    for(int f=0;f<5;f++) {
        System.out.println(sueldos[f]);
    }
}
```

Siempre que queremos acceder a una componente del vector debemos indicar entre corchetes la componente, dicho valor comienza a numerarse en cero y continua hasta un número menos del tamaño del vector, en nuestro caso creamos el vector con 5 elementos:

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

Por último en este programa creamos un objeto en la main y llamamos a los métodos de cargar e imprimir el vector:

```
public static void main(String[] ar) {
    PruebaVector1 pv=new PruebaVector1();
    pv.cargar();
    pv.imprimir();
}
```

## Problema 2:

Definir un vector de 5 componentes de tipo float que representen las alturas de 5 personas.

Obtener el promedio de las mismas. Contar cuántas personas son más altas que el promedio y cuántas más bajas.

## Programa:

```

import java.util.Scanner;
public class PruebaVector2 {
    private Scanner teclado;
    private float[] alturas;
    private float promedio;

    public void cargar() {
        teclado=new Scanner(System.in);
        alturas=new float[5];
        for(int f=0;f<5;f++) {
            System.out.print("Ingrese la altura de la persona:");
            alturas[f]=teclado.nextFloat();
        }
    }

    public void calcularPromedio() {
        float suma;
        suma=0;
        for(int f=0;f<5;f++) {
            suma=suma+alturas[f];
        }
        promedio=suma/5;
        System.out.println("Promedio de alturas:"+promedio);
    }

    public void mayoresMenores() {
        int may,men;
        may=0;
        men=0;
        for(int f=0;f<5;f++) {
            if (alturas[f]>promedio) {
                may++;
            } else {
                if (alturas[f]<promedio) {
                    men++;
                }
            }
        }
        System.out.println("Cantidad de personas mayores al promedio:"+may);
        System.out.println("Cantidad de personas menores al promedio:"+men);
    }

    public static void main(String[] ar) {
        PruebaVector2 pv2=new PruebaVector2();
        pv2.cargar();
        pv2.calcularPromedio();
        pv2.mayoresMenores();
    }
}

```

Definimos como atributo un vector donde almacenaremos las alturas:

```
private float[] alturas;
```

En la carga creamos el vector indicando que reserve espacio para 5 componentes:

```
alturas=new float[5];
```

Procedemos seguidamente a cargar todos sus elementos:

```
for(int f=0;f<5;f++) {  
    System.out.print("Ingrese la altura de la persona:");  
    alturas[f]=teclado.nextFloat();  
}
```

En otro método procedemos a sumar todas sus componentes y obtener el promedio. El promedio lo almacenamos en un atributo de la clase ya que lo necesitamos en otro método:

```
public void calcularPromedio() {  
    float suma;  
    suma=0;  
    for(int f=0;f<5;f++) {  
        suma=suma+alturas[f];  
    }  
    promedio=suma/5;  
    System.out.println("Promedio de alturas:"+promedio);  
}
```

Por último en un tercer método comparamos cada componente del vector con el atributo promedio, si el valor almacenado supera al promedio incrementamos un contador en caso que sea menor al promedio incrementamos otro contador:

```
public void mayoresMenores() {  
    int may,men;  
    may=0;  
    men=0;  
    for(int f=0;f<5;f++) {  
        if (alturas[f]>promedio) {  
            may++;  
        } else {  
            if (alturas[f]<promedio) {  
                men++;  
            }  
        }  
    }  
    System.out.println("Cantidad de personas mayores al promedio:"+may);  
    System.out.println("Cantidad de personas menores al promedio:"+men);  
}
```

### Importante:

En este problema podemos observar una ventaja de tener almacenadas todas las alturas de las personas. Si no conociéramos los vectores tenemos que cargar otra vez las alturas por teclado para compararlas con el promedio.

Mientras el programa está en ejecución tenemos el vector alturas a nuestra disposición.

Es importante tener en cuenta que cuando finaliza la ejecución del programa se pierde el contenido de todas las variables (simples y vectores)

### Problema 3:

Una empresa tiene dos turnos (mañana y tarde) en los que trabajan 8 empleados (4 por la mañana y 4 por la tarde)

Confeccionar un programa que permita almacenar los sueldos de los empleados agrupados por turno.

Imprimir los gastos en sueldos de cada turno.

### Programa:

```
import java.util.Scanner;
public class PruebaVector3 {
    private Scanner teclado;
    private float[] turnoMan;
    private float[] turnoTar;

    public void cargar() {
        teclado=new Scanner(System.in);
        turnoMan=new float[4];
        turnoTar=new float[4];
        System.out.println("Sueldos de empleados del turno de la mañana.");
        for(int f=0;f<4;f++) {
            System.out.print("Ingrese sueldo:");
            turnoMan[f]=teclado.nextFloat();
        }
        System.out.println("Sueldos de empleados del turno de la tarde.");
        for(int f=0;f<4;f++) {
            System.out.print("Ingrese sueldo:");
            turnoTar[f]=teclado.nextFloat();
        }
    }

    public void calcularGastos() {
        float man=0;
        float tar=0;
        for(int f=0;f<4;f++){
            man=man+turnoMan[f];
            tar=tar+turnoTar[f];
        }
        System.out.println("Total de gastos del turno de la mañana:"+man);
        System.out.println("Total de gastos del turno de la tarde:"+tar);
    }

    public static void main(String[] ar){
        PruebaVector3 pv=new PruebaVector3();
        pv.cargar();
        pv.calcularGastos();
    }
}
```

Definimos dos atributos de tipo vector donde almacenaremos los sueldos de los empleados de cada turno:

```
private float[ ] turnoMan;  
private float[ ] turnoTar;
```

Creamos los vectores con cuatro elementos cada uno:

```
turnoMan=new float[4];  
turnoTar=new float[4];
```

Mediante dos estructuras repetitivas procedemos a cargar cada vector:

```
System.out.println("Sueldos de empleados del turno de la mañana.");  
for(int f=0;f<4;f++) {  
    System.out.print("Ingrese sueldo:");  
    turnoMan[f]=teclado.nextFloat();  
}  
System.out.println("Sueldos de empleados del turno de la tarde.");  
for(int f=0;f<4;f++) {  
    System.out.print("Ingrese sueldo:");  
    turnoTar[f]=teclado.nextFloat();  
}
```

En otro método procedemos a sumar las componentes de cada vector y mostrar dichos acumuladores:

```
float man=0;  
float tar=0;  
for(int f=0;f<4;f++){  
    man=man+turnoMan[f];  
    tar=tar+turnoTar[f];  
}  
System.out.println("Total de gastos del turno de la mañana:"+man);  
System.out.println("Total de gastos del turno de la tarde:"+tar);
```

## Problemas propuestos

1. Desarrollar un programa que permita ingresar un vector de 8 elementos, e informe:  
El valor acumulado de todos los elementos del vector.  
El valor acumulado de los elementos del vector que sean mayores a 36.  
Cantidad de valores mayores a 50.
2. Realizar un programa que pida la carga de dos vectores numéricos enteros de 4 elementos. Obtener la suma de los dos vectores, dicho resultado guardarlo en un tercer vector del mismo tamaño. Sumar componente a componente.
3. Se tienen las notas del primer parcial de los alumnos de dos cursos, el curso A y el curso B, cada curso cuenta con 5 alumnos.

Realizar un programa que muestre el curso que obtuvo el mayor promedio general.

4. Cargar un vector de 10 elementos y verificar posteriormente si el mismo está ordenado de menor a mayor.

[Solución](#)

[\*\*Retornar\*\*](#)

# 17 - Vector (Tamaño de un vector)

[Listado completo de tutoriales](#)

Como hemos visto cuando se crea un vector indicamos entre corchetes su tamaño:

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

Luego cuando tenemos que recorrer dicho vector disponemos una estructura repetitiva for:

```
for(int f=0;f<5;f++) {  
    System.out.print("Ingrese valor de la componente:");  
    sueldos[f]=teclado.nextInt();  
}
```

Como vemos el for se repite mientras el contador f vale menos de 5. Esta estructura repetitiva es idéntica cada vez que recorremos el vector.

Que pasa ahora si cambiamos el tamaño del vector cuando lo creamos:

```
sueldos=new int[7];
```

Con esto tenemos que cambiar todos los for que recorren dicho vector. Ahora veremos que un vector al ser un objeto tiene un atributo llamado length que almacena su tamaño. Luego podemos modificar todos los for con la siguiente sintaxis:

```
for(int f=0;f<sueldos.length;f++) {  
    System.out.print("Ingrese valor de la componente:");  
    sueldos[f]=teclado.nextInt();  
}
```

También podemos pedir al usuario que indique el tamaño del vector en tiempo de ejecución, en estos casos se hace imprescindible el empleo del atributo length.

## Problema 1:

Se desea almacenar los sueldos de operarios. Cuando se ejecuta el programa se debe pedir la cantidad de sueldos a ingresar. Luego crear un vector con dicho tamaño.

## Programa:

```
import java.util.Scanner;
public class PruebaVector8 {
    private Scanner teclado;
    private int[] sueldos;

    public void cargar()
    {
        teclado=new Scanner(System.in);
        System.out.print("Cuantos sueldos cargará:");
        int cant;
        cant=teclado.nextInt();
        sueldos=new int[cant];
        for(int f=0;f<sueldos.length;f++) {
            System.out.print("Ingrese sueldo:");
            sueldos[f]=teclado.nextInt();
        }
    }

    public void imprimir() {
        for(int f=0;f<sueldos.length;f++) {
            System.out.println(sueldos[f]);
        }
    }

    public static void main(String[] ar) {
        PruebaVector8 pv=new PruebaVector8();
        pv.cargar();
        pv.imprimir();
    }
}
```

La definición del vector no varía:

```
private int[] sueldos;
```

Luego para la creación del mismo ingresamos una variable entera y la utilizamos como subíndice en el momento de la creación del vector:

```
System.out.print("Cuantos sueldos cargará:");  
int cant;  
cant=teclado.nextInt();  
sueldos=new int[cant];
```

Luego las estructuras repetitivas las acotamos accediendo al atributo length del vector:

```
for(int f=0;f<sueldos.length;f++) {  
    System.out.print("Ingrese sueldo:");  
    sueldos[f]=teclado.nextInt();  
}
```

## Problemas propuestos

1. Desarrollar un programa que permita ingresar un vector de n elementos, ingresar n por teclado. Luego imprimir la suma de todos sus elementos

[Solución](#)

[Retornar](#)

# 18 - Vectores paralelos

[Listado completo de tutoriales](#)

Este concepto se da cuando hay una relación entre las componentes de igual subíndice (misma posición) de un vector y otro.

|                |      |     |        |       |       |
|----------------|------|-----|--------|-------|-------|
| <i>nombres</i> | Juan | Ana | Marcos | Pablo | Laura |
| <i>edades</i>  | 12   | 21  | 27     | 14    | 21    |

Si tenemos dos vectores de 5 elementos cada uno. En uno se almacenan los nombres de personas en el otro las edades de dichas personas.

Decimos que el vector *nombres* es paralelo al vector *edades* si en la componente 0 de cada vector se almacena información relacionada a una persona (Juan - 12 años)

Es decir hay una relación entre cada componente de los dos vectores.

Esta relación la conoce únicamente el programador y se hace para facilitar el desarrollo de algoritmos que procesen los datos almacenados en las estructuras de datos.

## Problema 1:

Desarrollar un programa que permita cargar 5 nombres de personas y sus edades respectivas. Luego de realizar la carga por teclado de todos los datos imprimir los nombres de las personas mayores de edad (mayores o iguales a 18 años)

## Programa:

```
import java.util.Scanner;
public class PruebaVector10 {
    private Scanner teclado;
    private String[] nombres;
    private int[] edades;

    public void cargar() {
        teclado=new Scanner(System.in);
        nombres=new String[5];
        edades=new int[5];
        for(int f=0;f<nombres.length;f++) {
            System.out.print("Ingrese nombre:");
        }
    }
}
```

```

        nombres[f]=teclado.next();
        System.out.print("Ingrese edad:");
        edades[f]=teclado.nextInt();
    }
}

public void mayoresEdad() {
    System.out.println("Personas mayores de edad.");
    for(int f=0;f<nombres.length;f++) {
        if (edades[f]>=18) {
            System.out.println(nombres[f]);
        }
    }
}

public static void main(String[] ar) {
    PruebaVector10 pv=new PruebaVector10();
    pv.cargar();
    pv.mayoresEdad();
}
}

```

Definimos los dos vectores:

```

private String[] nombres;
private int[] edades;

```

Creamos los dos vectores con 5 elementos cada uno:

```

nombres=new String[5];
edades=new int[5];

```

Mediante un for procedemos a la carga de los elementos de los vectores:

```

for(int f=0;f<nombres.length;f++) {
    System.out.print("Ingrese nombre:");
    nombres[f]=teclado.next();
    System.out.print("Ingrese edad:");
    edades[f]=teclado.nextInt();
}

```

Podemos utilizar el length de cualquiera de los dos vectores, ya que tienen el mismo tamaño.

Para imprimir los nombres de las personas mayores de edad verificamos cada componente del vector de edades, en caso que sea igual o mayor o 18 procedemos a mostrar el elemento de la misma posición del otro vector:

```
for(int f=0; f<nombres.length; f++) {  
    if (edades[f]>=18) {  
        System.out.println(nombres[f]);  
    }  
}
```

[Retornar](#)

# 19 - Vectores (mayor y menor elemento)

[Listado completo de tutoriales](#)

Es una actividad común la búsqueda del mayor y menor elemento de un vector, lo mismo que su posición.

| <i>sueldos</i> |            |            |            |            |
|----------------|------------|------------|------------|------------|
| 120            | 750        | 820        | 550        | 490        |
| sueldos[0]     | sueldos[1] | sueldos[2] | sueldos[3] | sueldos[4] |

El mayor elemento es el 820 y se encuentra en la posición nº 2.

## Problema 1:

Confeccionar un programa que permita cargar los nombres de 5 operarios y sus sueldos respectivos. Mostrar el sueldo mayor y el nombre del operario.

## Programa:

```
import java.util.Scanner;
public class PruebaVector11 {
    private Scanner teclado;
    private String[] nombres;
    private float[] sueldos;

    public void cargar() {
        teclado=new Scanner(System.in);
        nombres=new String[5];
        sueldos=new float[5];
        for(int f=0;f<nombres.length;f++) {
            System.out.print("Ingrese el nombre del empleado:");
            nombres[f]=teclado.next();
            System.out.print("Ingrese el sueldo:");
            sueldos[f]=teclado.nextFloat();
        }
    }

    public void mayorSueldo() {
        float mayor;
        int pos;
        mayor=sueldos[0];
        pos=0;
        for(int f=1;f<nombres.length;f++) {
            if (sueldos[f]>mayor) {
                mayor=sueldos[f];
            }
        }
    }
}
```

```

        pos=f;
    }
}
System.out.println("El empleado con sueldo mayor es
"+nombres[pos]);
System.out.println("Tiene un sueldo:"+mayor);
}

public static void main(String[] ar) {
    PruebaVector11 pv=new PruebaVector11();
    pv.cargar();
    pv.mayorSueldo();
}
}

```

Definimos los dos vectores paralelos donde almacenaremos los nombres y los sueldos de los operarios:

```

private String[ ] nombres;
private float[ ] sueldos;

```

Creamos los dos vectores y procedemos a cargar sus elementos:

```

nombres=new String[5];
sueldos=new float[5];
for(int f=0;f<nombres.length;f++) {
    System.out.print("Ingrese el nombre del empleado:");
    nombres[f]=teclado.next();
    System.out.print("Ingrese el sueldo:");
    sueldos[f]=teclado.nextFloat();
}

```

Para obtener el mayor sueldo y el nombre del operario realizar los siguientes pasos:

Inicializamos una variable mayor con la primer componente del vector sueldos:

```

mayor=sueldos[0];

```

Inicializamos una variable pos con el valor 0, ya que decimos primeramente que el mayor es la primer componente del vector:

```

pos=0;

```

Recorremos las componentes del vector que faltan analizar, o sea, de la 1 a la 4:

```

for(int f=1;f<nombres.length;f++) {

```

Accedemos a cada componente para controlar si supera lo que tiene la variable mayor:

```
if (sueldos[f]>mayor) {
```

En caso de ser verdadera la condición asignamos a la variable mayor este nuevo valor sueldos[f]

```
mayor=sueldos[f];
```

y a la variable pos le cargamos la variable f que indica la componente que estamos analizando:

```
pos=f
```

Cuando salimos de la estructura repetitiva imprimimos la variable mayor que contiene el mayor sueldo y para imprimir el nombre del operario conociendo la posición del mayor sueldo imprimimos el elemento que ocupa la posición que indica la variable pos en el vector paralelo:

```
System.out.println("El empleado con sueldo mayor es  
"+nombres[pos]);  
System.out.println("Tiene un sueldo:"+mayor);
```

## Problemas propuestos

1. Cargar un vector de n elementos. imprimir el menor y un mensaje si se repite dentro del vector.

[Solución](#)

[Retornar](#)

# 20 - Vectores (ordenamiento)

[Listado completo de tutoriales](#)

El ordenamiento de un vector se logra intercambiando las componentes de manera que:

$\text{vec}[0] \leq \text{vec}[1] \leq \text{vec}[2]$  etc.

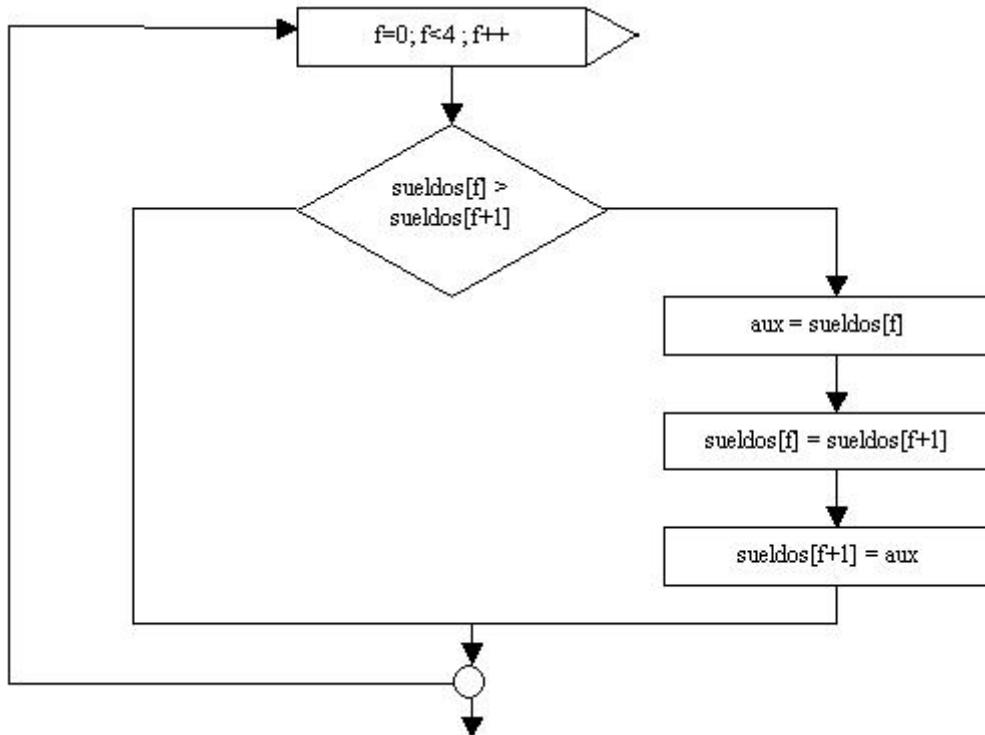
El contenido de la componente  $\text{vec}[0]$  sea menor o igual al contenido de la componente  $\text{vec}[1]$  y así sucesivamente.

Si se cumple lo dicho anteriormente decimos que el vector está ordenado de menor a mayor. Igualmente podemos ordenar un vector de mayor a menor.

Se puede ordenar tanto vectores con componentes de tipo int, float como String. En este último caso el ordenamiento es alfabético.

## Problema 1:

Se debe crear un vector donde almacenar 5 sueldos. Ordenar el vector sueldos de menor a mayor.



Esta primera aproximación tiene por objetivo analizar los intercambios de elementos dentro del vector.

El algoritmo consiste en comparar si la primera componente es mayor a la segunda, en caso que la condición sea verdadera, intercambiamos los contenidos de las componentes.

Vamos a suponer que se ingresan los siguientes valores por teclado:

1200  
750  
820  
550  
490

En este ejemplo: ¿es 1200 mayor a 750? La respuesta es verdadera, por lo tanto intercambiamos el contenido de la componente 0 con el de la componente 1.

Luego comparamos el contenido de la componente 1 con el de la componente 2: ¿Es 1200 mayor a 820?

La respuesta es verdadera entonces intercambiamos.

Si hay 5 componentes hay que hacer 4 comparaciones, por eso el for se repite 4 veces. Generalizando: si el vector tiene N componentes hay que hacer N-1 comparaciones.

| Cuando | f = 0 | f = 1 | f = 2 | f =  |
|--------|-------|-------|-------|------|
|        | 750   | 750   | 750   | 750  |
|        | 1200  | 820   | 820   | 820  |
|        | 820   | 1200  | 550   | 550  |
|        | 550   | 550   | 1200  | 490  |
|        | 490   | 490   | 490   | 1200 |

Podemos ver cómo el valor más grande del vector desciende a la última componente. Empleamos una variable auxiliar (aux) para el proceso de intercambio:

```
aux=sueldos[f];
sueldos[f]=sueldos[f+1];
sueldos[f+1]=aux;
```

Al salir del for en este ejemplo el contenido del vector es el siguiente:

750  
820  
550  
490  
1200

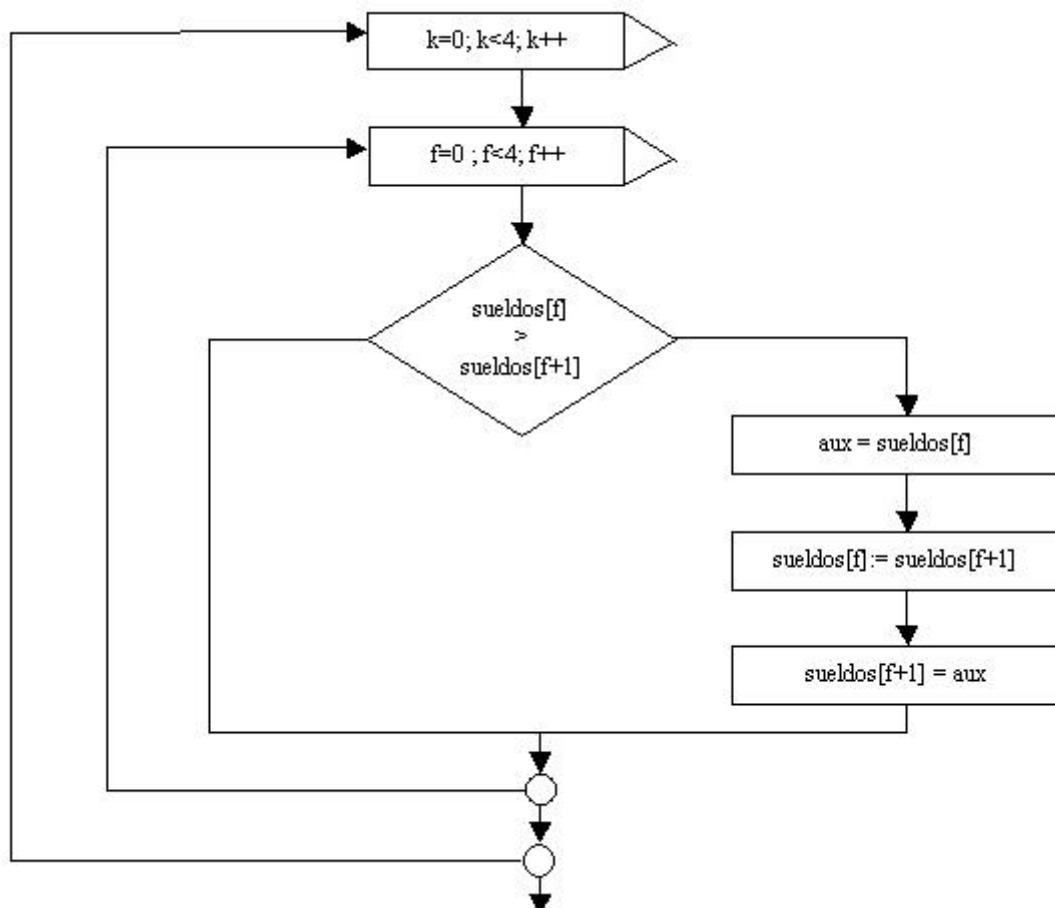
Analizando el algoritmo podemos comprobar que el elemento mayor del vector se ubica ahora en el último lugar.

Podemos definir otros vectores con distintos valores y comprobar que siempre el elemento mayor queda al final.

Pero todavía con este algoritmo no se ordena un vector. Solamente está ordenado el último elemento del vector.

Ahora bien, con los 4 elementos que nos quedan podemos hacer el mismo proceso visto anteriormente, con lo cual quedará ordenado otro elemento del vector. Este proceso lo repetiremos hasta que quede ordenado por completo el vector.

Como debemos repetir el mismo algoritmo podemos englobar todo el bloque en otra estructura repetitiva.



Realicemos una prueba del siguiente algoritmo:

Cuando  $k = 0$

| $f = 0$ | $f = 1$ | $f = 2$ | $f = 3$ |
|---------|---------|---------|---------|
| 750     | 750     | 750     | 750     |
| 1200    | 820     | 820     | 820     |
| 820     | 1200    | 550     | 550     |
| 550     | 550     | 1200    | 490     |

490                    490                    490                    120

Cuando k = 1

|       |       |       |     |
|-------|-------|-------|-----|
| f = 0 | f = 1 | f = 2 | f = |
| 750   | 750   | 750   | 750 |
| 820   | 550   | 550   | 550 |
| 550   | 820   | 490   | 490 |
| 490   | 490   | 820   | 820 |
| 1200  | 1200  | 1200  | 120 |

Cuando k = 2

|       |       |       |     |
|-------|-------|-------|-----|
| f = 0 | f = 1 | f = 2 | f = |
| 550   | 550   | 550   | 550 |
| 750   | 490   | 490   | 490 |
| 490   | 750   | 750   | 750 |
| 820   | 820   | 820   | 820 |
| 1200  | 1200  | 1200  | 120 |

Cuando k = 3

|       |       |       |     |
|-------|-------|-------|-----|
| f = 0 | f = 1 | f = 2 | f = |
| 490   | 490   | 490   | 490 |
| 550   | 550   | 550   | 550 |
| 750   | 750   | 750   | 750 |
| 820   | 820   | 820   | 820 |
| 1200  | 1200  | 1200  | 120 |

¿Porque repetimos 4 veces el for externo?

Como sabemos cada vez que se repite en forma completa el for interno queda ordenada una componente del vector. A primera vista diríamos que deberíamos repetir el for externo la cantidad de componentes del vector, en este ejemplo el vector sueldos tiene 5 componentes.

Si observamos, cuando quedan dos elementos por ordenar, al ordenar uno de ellos queda el otro automáticamente ordenado (podemos imaginar que si tenemos un vector con 2 elementos no se requiere el for externo, porque este debería repetirse una única vez)

Una última consideración a este ALGORITMO de ordenamiento es que los elementos que se van ordenando continuamos comparándolos.

Ejemplo: En la primera ejecución del for interno el valor 1200 queda ubicado en la posición 4 del vector. En la segunda ejecución comparamos si el 820 es mayor a 1200, lo cual seguramente será falso.

Podemos concluir que la primera vez debemos hacer para este ejemplo 4 comparaciones, en la segunda ejecución del for interno debemos hacer 3 comparaciones y en general debemos ir reduciendo en uno la cantidad de comparaciones.

Si bien el algoritmo planteado funciona, un algoritmo más eficiente, que se deriva del

anterior es el plantear un for interno con la siguiente estructura: (f=0 ; f<4-k; f++)  
Es decir restarle el valor del contador del for externo.

**Programa:**

```
import java.util.Scanner;
public class PruebaVector13 {
    private Scanner teclado;
    private int[] sueldos;

    public void cargar() {
        teclado=new Scanner(System.in);
        sueldos=new int[5];
        for(int f=0;f<sueldos.length;f++) {
            System.out.print("Ingrese el sueldo:");
            sueldos[f]=teclado.nextInt();
        }
    }

    public void ordenar() {
        for(int k=0;k<4;k++) {
            for(int f=0;f<4-k;f++) {
                if (sueldos[f]>sueldos[f+1]) {
                    int aux;
                    aux=sueldos[f];
                    sueldos[f]=sueldos[f+1];
                    sueldos[f+1]=aux;
                }
            }
        }
    }

    public void imprimir() {
        System.out.println("Sueldos ordenados de menor
        for(int f=0;f<sueldos.length;f++) {
            System.out.println(sueldos[f]);
        }
    }

    public static void main(String[] ar) {
        PruebaVector13 pv=new PruebaVector13();
        pv.cargar();
```

```
        pv.ordenar();
        pv.imprimir();
    }
}
```

También podemos ordenar vectores cuyas componentes sean de tipo String. Para esto no podemos utilizar el operador `>` sino debemos utilizar un método de la clase String:

```
String cad1="juan";
String cad2="analia";
if (cad1.compareTo(cad2)>0)
{
    System.out.println(cad1 + " es mayor alfabéticamente que " + cad2)
}
```

El método `compareTo` retorna un valor mayor a cero si `cad1` es mayor alfabéticamente. En este ejemplo `cad1` tiene un valor alfabéticamente mayor a `cad2`, luego el `compareTo` retorna un valor mayor a cero.

Si los dos String son exactamente iguales el método `compareTo` retorna un cero, y finalmente si `cad1` es menor alfabéticamente retorna un valor menor a cero.

## Problema 2:

Definir un vector donde almacenar los nombres de 5 países. Confeccionar el algoritmo de ordenamiento alfabético.

### Programa:

```
import java.util.Scanner;
public class PruebaVector14 {
    private Scanner teclado;
    private String[] paises;

    public void cargar() {
        teclado=new Scanner(System.in);
        paises=new String[5];
        for(int f=0;f<paises.length;f++) {
            System.out.print("Ingrese el nombre del país ");
            paises[f]=teclado.next();
        }
    }
}
```

```

public void ordenar() {
    for(int k=0;k<4;k++) {
        for(int f=0;f<4-k;f++) {
            if (paises[f].compareTo(paises[f+1])>0)
                String aux;
                aux=paises[f];
                paises[f]=paises[f+1];
                paises[f+1]=aux;
        }
    }
}

public void imprimir() {
    System.out.println("Paises ordenados en forma ascendente");
    for(int f=0;f<paises.length;f++) {
        System.out.println(paises[f]);
    }
}

public static void main(String[] ar) {
    PruebaVector14 pv=new PruebaVector14();
    pv.cargar();
    pv.ordenar();
    pv.imprimir();
}
}

```

Definimos un vector de tipo String:

```
private String[] paises;
```

Lo creamos indicando que almacenará cinco elementos:

```
paises=new String[5];
```

Procedemos a cargar el vector:

```
for(int f=0;f<paises.length;f++) {
    System.out.print("Ingrese el nombre del pais:");
    paises[f]=teclado.next();
```

```
}
```

Para el ordenamiento utilizamos el método `compareTo` para verificar si tenemos que intercambiar las componentes:

```
if (paises[f].compareTo(paises[f+1])>0) {
```

En el caso que si tenemos que intercambiarla utilizamos un auxiliar de tipo `String`:

```
String aux;
aux=paises[f];
paises[f]=paises[f+1];
paises[f+1]=aux;
```

## Problemas propuestos

1. Cargar un vector de  $n$  elementos de tipo entero. Ordenar posteriormente el vector.

[Solución](#)

[Retornar](#)

# 21 - Vectores (ordenamiento con vectores paralelos)

[Listado completo de tutoriales](#)

Cuando se tienen vectores paralelos y se ordena uno de ellos hay que tener la precaución de intercambiar los elementos de los vectores paralelos.

## Problema 1:

Confeccionar un programa que permita cargar los nombres de 5 alumnos y sus notas respectivas. Luego ordenar las notas de mayor a menor. Imprimir las notas y los nombres de los alumnos.

## Programa:

```
import java.util.Scanner;
public class PruebaVector16 {
    private Scanner teclado;
    private String[] nombres;
    private int[] notas;

    public void cargar() {
        teclado=new Scanner(System.in);
        nombres=new String[5];
        notas=new int[5];
        System.out.println("Carga de nombres y notas");
        for(int f=0;f<nombres.length;f++) {
            System.out.print("Ingrese el nombre del alumno:");
            nombres[f]=teclado.next();
            System.out.print("Ingrese la nota del alumno:");
            notas[f]=teclado.nextInt();
        }
    }

    public void ordenar() {
        for(int k=0;k<notas.length;k++) {
            for(int f=0;f<notas.length-1-k;f++) {
                if (notas[f]<notas[f+1]) {
                    int auxnota;
                    auxnota=notas[f];
                    notas[f]=notas[f+1];
                    notas[f+1]=auxnota;
                    String auxnombre;
                    auxnombre=nombres[f];
                    nombres[f]=nombres[f+1];
                    nombres[f+1]=auxnombre;
                }
            }
        }
    }
}
```

```

public void imprimir() {
    System.out.println("Nombres de alumnos y notas de mayor a menor");
    for(int f=0;f<notas.length;f++) {
        System.out.println(nombres[f] + " - " + notas[f]);
    }
}

public static void main(String[] ar) {
    PruebaVector16 pv=new PruebaVector16();
    pv.cargar();
    pv.ordenar();
    pv.imprimir();
}
}

```

Definimos los dos vectores:

```

private String[] nombres;
private int[] notas;

```

Creamos los dos vectores paralelos con cinco elementos cada uno:

```

nombres=new String[5];
notas=new int[5];

```

En el proceso de ordenamiento dentro de los dos for verificamos si debemos intercambiar los elementos del vector notas:

```

for(int k=0;k<notas.length;k++) {
    for(int f=0;f<notas.length-1-k;f++) {
        if (notas[f]<notas[f+1]) {

```

En el caso que la nota de la posición 'f' sea menor a de la posición siguiente 'f+1' procedemos a intercambiar las notas:

```

        int auxnota;
        auxnota=notas[f];
        notas[f]=notas[f+1];
        notas[f+1]=auxnota;
    }
}

```

y simultáneamente procedemos a intercambiar los elementos del vector paralelo (con esto logramos que los dos vectores continuen siendo vectores paralelos):

```

        String auxnombre;
        auxnombre=nombres[f];
        nombres[f]=nombres[f+1];
        nombres[f+1]=auxnombre;
    }
}

```

Como vemos utilizamos dos auxiliares distintos porque los elementos de los dos vectores son de distinto tipo (int y String)

Si deseamos ordenar alfabéticamente la condición dependerá del vector nombres.

## Problemas propuestos

1. Cargar en un vector los nombres de 5 países y en otro vector paralelo la cantidad de habitantes del mismo. Ordenar alfabéticamente e imprimir los resultados. Por último ordenar con respecto a la cantidad de habitantes (de mayor a menor) e imprimir nuevamente.

[Solución](#)

[Retornar](#)

# 22 - Estructura de datos tipo matriz

[Listado completo de tutoriales](#)

Una matriz es una estructura de datos que permite almacenar un CONJUNTO de datos del MISMO tipo.

Con un único nombre se define la matriz y por medio de DOS subíndices hacemos referencia a cada elemento de la misma (componente)

|       |   | <i>mat</i> |    |    |     |     |
|-------|---|------------|----|----|-----|-----|
|       |   | Columnas   |    |    |     |     |
|       |   | 1          | 2  | 3  | 4   | 5   |
| Filas | 0 | 50         | 5  | 27 | 400 | 7   |
|       | 1 | 0          | 67 | 90 | 6   | 97  |
|       | 2 | 30         | 14 | 23 | 251 | 490 |

Hemos graficado una matriz de 3 filas y 5 columnas. Para hacer referencia a cada elemento debemos indicar primero la fila y luego la columna, por ejemplo en la componente 1,4 se almacena el valor 97.

En este ejemplo almacenamos valores enteros. Todos los elementos de la matriz deben ser del mismo tipo (int, float, String etc.)

Las filas y columnas comienzan a numerarse a partir de cero, similar a los vectores.

Una matriz se la puede representar por un conjunto de vectores.

## Problema 1:

Crear una matriz de 3 filas por 5 columnas con elementos de tipo int, cargar sus componentes y luego imprimirlas.

## Programa:

```
import java.util.Scanner;
public class Matriz1 {
    private Scanner teclado;
    private int[][] mat;
```

```

public void cargar() {
    teclado=new Scanner(System.in);
    mat=new int[3][5];
    for(int f=0;f<3;f++) {
        for(int c=0;c<5;c++) {
            System.out.print("Ingrese componente:");
            mat[f][c]=teclado.nextInt();
        }
    }
}

public void imprimir() {
    for(int f=0;f<3;f++) {
        for(int c=0;c<5;c++) {
            System.out.print(mat[f][c]+" ");
        }
        System.out.println();
    }
}

public static void main(String[] ar) {
    Matriz1 ma=new Matriz1();
    ma.cargar();
    ma.imprimir();
}
}

```

Para definir una matriz debemos antecederle los corchetes abiertos y cerrados dos veces:

```
private int[][][] mat;
```

De esta forma el compilador de Java puede diferenciar los vectores de las matrices.

Para crear la matriz, es decir hacer la reserva de espacio de todas sus componentes debemos utilizar el operador new y mediante dos subíndices indicamos la cantidad de filas y columnas que tendrá la matriz:

```
mat=new int[3][5];
```

Luego debemos pasar a cargar sus 15 componentes (cada fila almacena 5 componentes y tenemos 3 filas)

Lo más cómodo es utilizar un for anidado, el primer for que incrementa el contador f lo utilizamos para recorrer las filas y el contador interno llamado c lo utilizamos para recorrer las columnas.

Cada vez que se repite en forma completa el for interno se carga una fila completa, primero se carga la fila cero en forma completa, luego la fila uno y finalmente la fila 2.

Siempre que accedemos a una posición de la matriz debemos disponer dos subíndices que hagan referencia a la fila y columna mat[f][c]):

```
for(int f=0;f<3;f++) {  
    for(int c=0;c<5;c++) {  
        System.out.print("Ingrese componente:");  
        mat[f][c]=teclado.nextInt();  
    }  
}
```

Para imprimir la matriz de forma similar utilizamos dos for para acceder a cada elemento de la matriz:

```
for(int f=0;f<3;f++) {  
    for(int c=0;c<5;c++) {  
        System.out.print(mat[f][c]+" ");  
    }  
    System.out.println();  
}
```

Cada vez que se ejecuta todas las vueltas del for interno tenemos en pantalla una fila completa de la matriz, por eso pasamos a ejecutar un salto de línea (con esto logramos que en pantalla los datos aparezcan en forma matricial):

```
System.out.println();
```

## Problema 2:

Crear y cargar una matriz de 4 filas por 4 columnas. Imprimir la diagonal principal.

|   |   |   |   |
|---|---|---|---|
| x | - | - | - |
| - | x | - | - |
| - | - | x | - |
| - | - | - | x |

### Programa:

```
import java.util.Scanner;
public class Matriz2 {
    private Scanner teclado;
    private int[][] mat;

    public void cargar() {
        teclado=new Scanner(System.in);
        mat=new int[4][4];
        for(int f=0;f<4;f++) {
            for(int c=0;c<4;c++) {
                System.out.print("Ingrese componente:");
                mat[f][c]=teclado.nextInt();
            }
        }
    }

    public void imprimirDiagonalPrincipal() {
        for(int k=0;k<4;k++) {
            System.out.print(mat[k][k]+" ");
        }
    }

    public static void main(String[] ar) {
        Matriz2 ma=new Matriz2();
        ma.cargar();
        ma.imprimirDiagonalPrincipal();
    }
}
```

La definición, creación y carga de la matriz no varían con el ejemplo anterior.

Para imprimir la diagonal principal de la matriz lo más conveniente es utilizar un for que se repita 4 veces y disponer como subíndice dicho contador (los elementos de la diagonal principal coinciden los valores de la fila y columna):

```
for(int k=0;k<4;k++) {
    System.out.print(mat[k][k]+" ");
}
```

### Problema 3:

Crear y cargar una matriz de 3 filas por 4 columnas. Imprimir la primer fila. Imprimir la última fila e imprimir la primer columna.

**Programa:**

```
import java.util.Scanner;
public class Matriz3 {
    private Scanner teclado;
    private int[][] mat;

    public void cargar() {
        teclado=new Scanner(System.in);
        mat=new int[3][4];
        for(int f=0;f<3;f++) {
            for(int c=0;c<4;c++) {
                System.out.print("Ingrese componente:");
                mat[f][c]=teclado.nextInt();
            }
        }
    }

    public void primerFila() {
        System.out.println("Primer fila de la matriz:");
        for(int c=0;c<4;c++) {
            System.out.println(mat[0][c]);
        }
    }

    public void ultimaFila() {
        System.out.println("Ultima fila de la matriz:");
        for(int c=0;c<4;c++) {
            System.out.println(mat[2][c]);
        }
    }

    public void primerColumna() {
        System.out.println("Primer columna:");
        for(int f=0;f<3;f++) {
            System.out.println(mat[f][0]);
        }
    }
}
```

```
public static void main(String[] ar) {  
    Matriz3 ma=new Matriz3();  
    ma.cargar();  
    ma.primerFila();  
    ma.ultimaFila();  
    ma.primerColumna();  
}  
}
```

Creamos una matriz de 3 filas y 4 columnas:

```
mat=new int[3][4];
```

Luego de cargarla el primer método que codificamos es el que imprime la primer fila. Disponemos un for para recorrer las columnas, ya que la fila siempre será la cero. Como son cuatro los elementos de la primer fila el for se repite esta cantidad de veces:

```
System.out.println("Primer fila de la matriz:");  
for(int c=0;c<4;c++) {  
    System.out.println(mat[0][c]);  
}
```

Para imprimir la última fila el algoritmo es similar, disponemos un for que se repite 4 veces y en el subíndice de la fila disponemos el valor 2 (ya que la matriz tiene 3 filas):

```
System.out.println("Ultima fila de la matriz:");  
for(int c=0;c<4;c++) {  
    System.out.println(mat[2][c]);  
}
```

Para imprimir la primer columna el for debe repetirse 3 veces ya que la matriz tiene 3 filas. Dejamos constante el subíndice de la columna con el valor cero:

```
System.out.println("Primer columna:");  
for(int f=0;f<3;f++) {  
    System.out.println(mat[f][0]);  
}
```

## Problemas propuestos

1. Crear una matriz de 2 filas y 5 columnas. Realizar la carga de componentes por columna (es decir primero ingresar toda la primer columna, luego la segunda

columna y así sucesivamente)  
Imprimir luego la matriz.

[Solución](#)

[\*\*Retornar\*\*](#)

# 23 - Matrices (cantidad de filas y columnas)

[Listado completo de tutoriales](#)

Como hemos visto para definir y crear la matriz utilizamos la siguiente sintaxis:

```
int[][] mat;
```

Creación:

```
mat=new int[3][4];
```

Como las matrices son objetos en Java disponemos por un lado del atributo `length` que almacena la cantidad de filas de la matriz:

```
System.out.println("Cantidad de filas de la matriz:" + mat.length);
```

También podemos preguntarle a cada fila de la matriz la cantidad de elementos que almacena:

```
System.out.println("Cantidad de elementos de la primer fila:" + mat[0].length);
```

## Problema 1:

Crear una matriz de  $n * m$  filas (cargar  $n$  y  $m$  por teclado) Imprimir la matriz completa y la última fila.

## Programa:

```
import java.util.Scanner;
public class Matriz5 {
    private Scanner teclado;
    private int[][] mat;

    public void cargar() {
        teclado=new Scanner(System.in);
        System.out.print("Cuantas fila tiene la matriz:");
        int filas=teclado.nextInt();
        System.out.print("Cuantas columnas tiene la matriz:");
        int columnas=teclado.nextInt();
        mat=new int[filas][columnas];
        for(int f=0;f<mat.length;f++) {
            for(int c=0;c<mat[f].length;c++) {
```

```

        System.out.print("Ingrese componente:");
        mat[f][c]=teclado.nextInt();
    }
}

public void imprimir() {
    for(int f=0;f<mat.length;f++) {
        for(int c=0;c<mat[f].length;c++) {
            System.out.print(mat[f][c]+" ");
        }
        System.out.println();
    }
}

public void imprimirUltimaFila() {
    System.out.println("Ultima fila");
    for(int c=0;c<mat[mat.length-1].length;c++) {
        System.out.print(mat[mat.length-1][c]+" ");
    }
}

public static void main(String[] ar) {
    Matriz5 ma=new Matriz5();
    ma.cargar();
    ma.imprimir();
    ma.imprimirUltimaFila();
}
}

```

En este ejemplo cada vez que se ejecute el programa el tamaño de la matriz lo define el usuario, para ello ingresamos por teclado dos enteros y seguidamente procedemos a crear la matriz con dichos valores:

```

System.out.print("Cuantas fila tiene la matriz:");
int filas=teclado.nextInt();
System.out.print("Cuantas columnas tiene la matriz:");
int columnas=teclado.nextInt();
mat=new int[filas][columnas];

```

Ahora las estructuras repetitivas las acotamos preguntando a la misma matriz la cantidad de filas y la cantidad de elementos de cada fila(mat.length almacena la cantidad de filas de la matriz y mat[f].length cuando f vale cero accedemos a la cantidad de elementos de la fila cero y así sucesivamente para cada valor de f):

```

for(int f=0;f<mat.length;f++) {
    for(int c=0;c<mat[f].length;c++) {
        System.out.print("Ingrese componente:");
        mat[f][c]=teclado.nextInt();
    }
}

```

El algoritmo de impresión es idéntico al visto anteriormente con la modificación de las condiciones de los for:

```
public void imprimir() {  
    for(int f=0;f<mat.length;f++) {  
        for(int c=0;c<mat[f].length;c++) {  
            System.out.print(mat[f][c]+" ");  
        }  
        System.out.println();  
    }  
}
```

Para imprimir la última fila debemos disponer un valor fijo en el subíndice de la fila (en este caso no podemos disponer un número fijo sino preguntarle a la misma matriz la cantidad de filas y restarle uno ya que las filas comienzan a numerarse a partir de cero: mat[mat.length-1][c])

También la condición del for debemos acceder al atributo length de la última fila mat[mat.length-1].length

```
for(int c=0;c<mat[mat.length-1].length;c++) {  
    System.out.print(mat[mat.length-1][c]+" ");  
}
```

## Problema 2:

Crear una matriz de  $n * m$  filas (cargar  $n$  y  $m$  por teclado) Imprimir el mayor elemento y la fila y columna donde se almacena.

## Programa:

```
import java.util.Scanner;  
public class Matriz6 {  
    private Scanner teclado;  
    private int[][] mat;  
  
    public void cargar() {  
        teclado=new Scanner(System.in);  
        System.out.print("Cuantas fila tiene la matriz:");  
        int filas=teclado.nextInt();  
        System.out.print("Cuantas columnas tiene la matriz:");  
        int columnas=teclado.nextInt();  
        mat=new int[filas][columnas];  
        for(int f=0;f<mat.length;f++) {  
            for(int c=0;c<mat[f].length;c++) {  
                System.out.print("Ingrese componente:");  
                mat[f][c]=teclado.nextInt();  
            }  
        }  
    }
```

```

public void imprimirMayor() {
    int mayor=mat[0][0];
    int filamay=0;
    int columnamay=0;
    for(int f=0;f<mat.length;f++) {
        for(int c=0;c<mat[f].length;c++) {
            if (mat[f][c]>mayor) {
                mayor=mat[f][c];
                filamay=f;
                columnamay=c;
            }
        }
    }
    System.out.println("El elemento mayor es:"+mayor);
    System.out.println("Se encuentra en la fila:"+filamay+ " y en la
columna: "+columnamay);
}

public static void main(String[] ar) {
    Matriz6 ma=new Matriz6();
    ma.cargar();
    ma.imprimirMayor();
}
}

```

Para obtener el mayor elemento de la matriz y la fila y columna donde se ubica debemos inicializar una variable mayor con el elemento de la fila cero y columna cero (esto lo hacemos suponiendo que en dicha posición se almacena el mayor):

```

int mayor=mat[0][0];
int filamay=0;
int columnamay=0;

```

Luego mediante dos for recorremos todos los elementos de la matriz y cada vez que encontramos un elemento mayor al actual procedemos a actualizar la variable mayor y la posición donde se almacena:

```

for(int f=0;f<mat.length;f++) {
    for(int c=0;c<mat[f].length;c++) {
        if (mat[f][c]>mayor) {
            mayor=mat[f][c];
            filamay=f;
            columnamay=c;
        }
    }
}

```

## Problemas propuestos

1. Crear una matriz de  $n * m$  filas (cargar  $n$  y  $m$  por teclado) Intercambiar la primer fila con la segundo. Imprimir luego la matriz.

2. Crear una matriz de  $n * m$  filas (cargar  $n$  y  $m$  por teclado) Imprimir los cuatro valores que se encuentran en los vértices de la misma (mat[0][0] etc.)

[Solución](#)

[\*\*Retornar\*\*](#)

# 24 - Matrices y vectores paralelos

[Listado completo de tutoriales](#)

Dependiendo de la complejidad del problema podemos necesitar el empleo de vectores y matrices paralelos.

## Problema 1:

Se tiene la siguiente información:

- Nombres de 4 empleados.
  - Ingresos en concepto de sueldo, cobrado por cada empleado, en los últimos 3 meses.
- Confeccionar el programa para:

- Realizar la carga de la información mencionada.
- Generar un vector que contenga el ingreso acumulado en sueldos en los últimos 3 meses para cada empleado.
- Mostrar por pantalla el total pagado en sueldos a todos los empleados en los últimos 3 meses
- Obtener el nombre del empleado que tuvo el mayor ingreso acumulado

| empleados | sueldos | sueldostot |
|-----------|---------|------------|
| Marcos    | 540     | 760        |
| Ana       | 200     | 250        |
| Luis      | 760     | 760        |
| Maria     | 605     | 810        |

## Programa:

```
import java.util.Scanner;
public class Matriz9 {
    private Scanner teclado;
    private String[] empleados;
    private int[][] sueldos;
    private int[] sueldostot;

    public void cargar() {
        teclado=new Scanner(System.in);
        empleados=new String[4];
```

```

        sueldos=new int[4][3];
        for(int f=0;f<empleados.length;f++){
            System.out.print("Ingrese el nombre del empleado:");
            empleados[f]=teclado.next();
            for(int c=0;c<sueldos[f].length;c++) {
                System.out.print("Ingrese sueldo:");
                sueldos[f][c]=teclado.nextInt();
            }
        }
    }

    public void calcularSumaSueldos() {
        sueldostot=new int[4];
        for(int f=0;f<sueldos.length;f++) {
            int suma=0;
            for(int c=0;c<sueldos[f].length;c++) {
                suma=suma+sueldos[f][c];
            }
            sueldostot[f]=suma;
        }
    }

    public void imprimirTotalPagado() {
        System.out.println("Total de sueldos pagados por empleado.");
        for(int f=0;f<sueldostot.length;f++) {
            System.out.println(empleados[f] + " - " + sueldostot[f]);
        }
    }

    public void empleadoMayorSueldo() {
        int may=sueldostot[0];
        String nom=empleados[0];
        for(int f=0;f<sueldostot.length;f++) {
            if (sueldostot[f]>may) {
                may=sueldostot[f];
                nom=empleados[f];
            }
        }
        System.out.println("El empleado con mayor sueldo es "+ nom + " que
tiene un sueldo de "+may);
    }

    public static void main(String[] ar){
        Matriz9 ma=new Matriz9();
        ma.cargar();
        ma.calcularSumaSueldos();
        ma.imprimirTotalPagado();
        ma.empleadoMayorSueldo();
    }
}

```

Para resolver este problema lo primero que hacemos es definir una matriz donde se almacenarán los sueldos mensuales de cada empleado, un vector de tipo String donde almacenaremos los nombre de cada empleado y finalmente definimos un vector paralelo a la matriz donde almacenaremos la suma de cada fila de la matriz:

```

private String[] empleados;
private int[][] sueldos;
private int[] sueldostot;

```

En el método de cargar inicializamos el vector con los nombres de los empleados y la matriz paralela donde se almacenan los últimos tres sueldos (previo a cargar procedemos a crear el vector y la matriz):

```
empleados=new String[4];
sueldos=new int[4][3];
for(int f=0;f<empleados.length;f++){
    System.out.print("Ingrese el nombre del empleado:");
    empleados[f]=teclado.next();
    for(int c=0;c<sueldos[f].length;c++) {
        System.out.print("Ingrese sueldo:");
        sueldos[f][c]=teclado.nextInt();
    }
}
```

El método sumar sueldos crea el vector donde se almacenará la suma de cada fila de la matriz. Mediante dos for recorremos toda la matriz y sumamos cada fila:

```
sueldostot=new int[4];
for(int f=0;f<sueldos.length;f++) {
    int suma=0;
    for(int c=0;c<sueldos[f].length;c++) {
        suma=suma+sueldos[f][c];
    }
    sueldostot[f]=suma;
}
```

El método imprimirTotalPagado tiene por objetivo mostrar los dos vectores (el de nombre de los empleados y el que almacena la suma de cada fila de la matriz):

```
for(int f=0;f<sueldostot.length;f++) {
    System.out.println(empleados[f] + " " + sueldostot[f]);
}
```

Por último para obtener el nombre del empleado con mayor sueldo acumulado debemos inicializar dos variables auxiliares con el primer elemento del vector de empleados y en otra auxiliar guardamos la primer componente del vector sueldostot:

```
int may=sueldostot[0];
String nom=empleados[0];
for(int f=0;f<sueldostot.length;f++) {
    if (sueldostot[f]>may) {
        may=sueldostot[f];
        nom=empleados[f];
    }
}
System.out.println("El empleado con mayor sueldo es " + nom + " que tiene un sueldo de "+may);
```

## Problemas propuestos

1. Se desea saber la temperatura media trimestral de cuatro países. Para ello se tiene como dato las temperaturas medias mensuales de dichos países.  
Se debe ingresar el nombre del país y seguidamente las tres temperaturas medias mensuales.  
Seleccionar las estructuras de datos adecuadas para el almacenamiento de los datos en memoria.
  - a - Cargar por teclado los nombres de los países y las temperaturas medias mensuales.
  - b - Imprimir los nombres de los países y las temperaturas medias mensuales de las mismas.
  - c - Calcular la temperatura media trimestral de cada país.
  - d - Imprimir los nombres de las provincias y las temperaturas medias trimestrales.
  - e - Imprimir el nombre de la provincia con la temperatura media trimestral mayor.

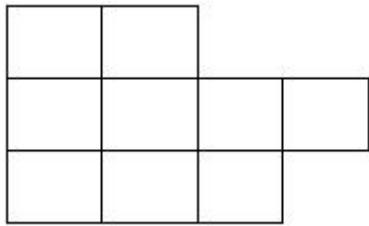
[Solución](#)

[\*\*Retornar\*\*](#)

# 25 - Matrices irregulares

[Listado completo de tutoriales](#)

Java nos permite crear matrices irregulares. Se dice que una matriz es irregular si la cantidad de elementos de cada fila varía. Luego podemos imaginar una matriz irregular:



Como podemos ver la fila cero tiene reservado dos espacios, la fila uno reserva cuatro espacios y la última fila reserva espacio para tres componentes.

Para crear la matriz irregular del gráfico:

La declaración es la misma que para matrices regulares:

```
int [][] mat;
```

Primero creamos la cantidad de filas dejando vacío el espacio que indica la cantidad de columnas:

```
mat=new int[3][];
```

Luego debemos ir creando cada fila de la matriz indicando la cantidad de elementos de la respectiva fila:

```
mat[0]=new int[2];
mat[1]=new int[4];
mat[2]=new int[3];
```

Luego la forma para acceder a sus componentes es similar a las matrices regulares, siempre teniendo en cuenta y validando que exista dicha componente:

```
mat[0][0]=120;
```

Dará un error si queremos cargar la tercer componente de la fila cero (esto debido a que no existe):

```
mat[0][2]=230;
```

Luego si queremos saber la cantidad de filas que tiene la matriz:

```
System.out.println(mat.length);
```

Si queremos saber la cantidad de elementos de una determinada fila:

```
System.out.println("Cantidad de elementos de la fila 0:"+mat[0].length);
System.out.println("Cantidad de elementos de la fila 1:"+mat[1].length);
System.out.println("Cantidad de elementos de la fila 2:"+mat[2].length);
```

## Problema 1:

Confeccionaremos un programa que permita crear una matriz irregular y luego imprimir la matriz en forma completa.

### Programa:

```
import java.util.Scanner;
public class MatrizIrregular1 {
    private Scanner teclado;
    private int[][] mat;

    public void cargar() {
        teclado=new Scanner(System.in);
        System.out.print("Cuantas fila tiene la matriz:");
        int filas=teclado.nextInt();
        mat=new int[filas][];
        for(int f=0;f<mat.length;f++) {
            System.out.print("Cuantas elementos tiene la fila " + f + ":");
            int elementos=teclado.nextInt();
            mat[f]=new int[elementos];
            for(int c=0;c<mat[f].length;c++) {
                System.out.print("Ingrese componente:");
                mat[f][c]=teclado.nextInt();
            }
        }
    }

    public void imprimir() {
        for(int f=0;f<mat.length;f++) {
            for(int c=0;c<mat[f].length;c++) {
                System.out.print(mat[f][c]+ " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] ar) {
        MatrizIrregular1 ma=new MatrizIrregular1();
        ma.cargar();
        ma.imprimir();
    }
}
```

```
    }  
}
```

Primero creamos la cantidad de filas que tendrá la matriz (en los corchetes para las columnas no disponemos valor):

```
System.out.print("Cuantas fila tiene la matriz:");  
int filas=teclado.nextInt();  
mat=new int[filas][];
```

Dentro del primer for pedimos que ingrese la cantidad de elementos que tendrá cada fila y utilizamos el operador new nuevamente, pero en este caso se están creando cada fila de la matriz (Java trata a cada fila como un vector):

```
for(int f=0;f<mat.length;f++) {  
    System.out.print("Cuantas elementos tiene la fila " + f + ":");  
    int elementos=teclado.nextInt();  
    mat[f]=new int[elementos];
```

Dentro del for interno hacemos la carga de las componentes propiamente dicho de la matriz (podemos ir cargando cada fila a medida que las vamos creando):

```
for(int c=0;c<mat[f].length;c++) {  
    System.out.print("Ingrese componente:");  
    mat[f][c]=teclado.nextInt();  
}
```

Luego imprimimos la matriz en forma completa teniendo cuidado las condiciones que disponemos en cada for.

El primer for se repite tantas veces como filas tiene la matriz:  $f < \text{mat.length}$  y el for interno se repite tantas veces como elementos tiene la fila que estamos procesando  $c < \text{mat}[f].length$ :

```
for(int f=0;f<mat.length;f++) {  
    for(int c=0;c<mat[f].length;c++) {  
        System.out.print(mat[f][c]+ " ");  
    }  
    System.out.println();  
}
```

## Problemas propuestos

1. Confeccionar una clase para administrar una matriz irregular de 5 filas y 1 columna la primer fila, 2 columnas la segunda fila y así sucesivamente hasta 5 columnas la última fila (crearla sin la intervención del operador)  
Realizar la carga por teclado e imprimir posteriormente.

2. Confeccionar una clase para administrar los días que han faltado los 3 empleados de una empresa.

Definir un vector de 3 elementos de tipo String para cargar los nombres y una matriz irregular para cargar los días que han faltado cada empleado (cargar el número de día que faltó)

Cada fila de la matriz representan los días de cada empleado.

Mostrar los empleados con la cantidad de inasistencias.

Cuál empleado faltó menos días.

[Solución](#)

[Retornar](#)

# 26 - Constructor de la clase

[Listado completo de tutoriales](#)

En Java podemos definir un método que se ejecute inicialmente y en forma automática. Este método se lo llama constructor.

El constructor tiene las siguientes características:

- Tiene el mismo nombre de la clase.
- Es el primer método que se ejecuta.
- Se ejecuta en forma automática.
- No puede retornar datos.
- Se ejecuta una única vez.
- Un constructor tiene por objetivo inicializar atributos.

## Problema 1:

Se desea guardar los sueldos de 5 operarios en un vector. Realizar la creación y carga del vector en el constructor.

### Programa:

```
import java.util.Scanner;
public class Operarios {
    private Scanner teclado;
    private int[] sueldos;

    public Operarios()
    {
        teclado=new Scanner(System.in);
        sueldos=new int[5];
        for(int f=0;f<5;f++) {
            System.out.print("Ingrese valor de la compo");
            sueldos[f]=teclado.nextInt();
        }
    }
}
```

```

public void imprimir() {
    for(int f=0;f<5;f++) {
        System.out.println(sueldos[f]);
    }
}

public static void main(String[] ar) {
    Operarios op=new Operarios();
    op.imprimir();
}
}

```

Como podemos ver es el mismo problema que resolvimos cuando vimos vectores. La diferencia es que hemos sustituido el método cargar con el constructor:

```

public Operarios()
{
    teclado=new Scanner(System.in);
    sueldos=new int[5];
    for(int f=0;f<5;f++) {
        System.out.print("Ingrese valor de la componente:");
        sueldos[f]=teclado.nextInt();
    }
}

```

Como la clase se llama Operarios el constructor tiene el mismo nombre, no disponemos la palabra clave void ya que el constructor no puede retornar datos.

La ventaja de plantear un constructor en lugar de definir un método con cualquier nombre es que se llamará en forma automática cuando se crea un objeto de esta clase:

```

public static void main(String[] ar) {
    Operarios op=new Operarios();
}

```

Cuando se crea el objeto op se llama al método constructor.

Finalmente llamamos al método imprimir:

```
op.imprimir();
```

## Problema 2:

Plantear una clase llamada Alumno y definir como atributos su nombre y su edad. En el constructor realizar la carga de datos. Definir otros dos métodos para imprimir los datos ingresados y un mensaje si es mayor o no de edad (edad  $\geq 18$ )

**Programa:**

```
import java.util.Scanner;
public class Alumno {
    private Scanner teclado;
    private String nombre;
    private int edad;

    public Alumno() {
        teclado=new Scanner(System.in);
        System.out.print("Ingrese nombre:");
        nombre=teclado.next();
        System.out.print("Ingrese edad:");
        edad=teclado.nextInt();
    }

    public void imprimir() {
        System.out.println("Nombre:"+nombre);
        System.out.println("Edad:"+edad);
    }

    public void esMayorEdad() {
        if (edad>=18) {
            System.out.print(nombre+" es mayor de edad.");
        } else {
            System.out.print(nombre+" no es mayor de edad.");
        }
    }

    public static void main(String[] ar) {
        Alumno alumno1=new Alumno();
        alumno1.imprimir();
        alumno1.esMayorEdad();
    }
}
```

Declaramos la clase Persona, sus tres atributos y definimos el constructor con el mismo nombre de la clase:

```
public class Alumno {  
    private Scanner teclado;  
    private String nombre;  
    private int edad;  
  
    public Alumno() {  
        teclado=new Scanner(System.in);  
        System.out.print("Ingrese nombre:");  
        nombre=teclado.next();  
        System.out.print("Ingrese edad:");  
        edad=teclado.nextInt();  
    }  
}
```

En la main el constructor se llama en forma automática cuando creamos un objeto de la clase Alumno:

```
public static void main(String[] ar) {  
    Alumno alumno1=new Alumno();
```

Los otros dos métodos deben llamarse por su nombre y en el orden que necesitemos:

```
alumno1.imprimir();  
alumno1.esMayorEdad();
```

## Problemas propuestos

1. Confeccionar una clase que represente un empleado. Definir como atributos su nombre y su sueldo. En el constructor cargar los atributos y luego en otro método imprimir sus datos y por último uno que imprima un mensaje si debe pagar impuestos (si el sueldo supera a 3000)
2. Implementar la clase operaciones. Se deben cargar dos valores enteros en el constructor, calcular su suma, resta, multiplicación y división, cada una en un método, imprimir dichos resultados.

[Solución](#)

[Retornar](#)

La clase String está orientada a manejar cadenas de caracteres. Hasta este momento hemos utilizado algunos métodos de la clase String (equals, compareTo)

Ahora veremos otro conjunto de métodos de uso común de la clase String:

## Métodos

- `boolean equals(String s1)`

Como vimos el método equals retorna true si el contenido de caracteres del parámetro s1 es exactamente igual a la cadena de caracteres del objeto que llama al método equals.

- `boolean equalsIgnoreCase(String s1)`

El funcionamiento es casi exactamente igual que el método equals con la diferencia que no tiene en cuenta mayúsculas y minúsculas (si comparamos 'Ana' y 'ana' luego el método equalsIgnoreCase retorna true)

- `int compareTo(String s1)`

Este método retorna un 0 si el contenido de s1 es exactamente igual al String contenido por el objeto que llama al método compareTo. Retorna un valor  $>0$  si el contenido del String que llama al método compareTo es mayor alfabéticamente al parámetro s1.

- `char charAt(int pos)`

Retorna un carácter del String, llega al método la posición del carácter a extraer.

- `int length()`

Retorna la cantidad de caracteres almacenados en el String.

- `String substring(int pos1,int pos2)`

Retorna un substring a partir de la posición indicada en el parámetro pos1 hasta la posición pos2 sin incluir dicha posición.

- `int indexOf(String s1)`

Retorna -1 si el String que le pasamos como parámetro no está contenida en la cadena del objeto que llama al método. En caso que se encuentre contenido el String s1 retorna la posición donde comienza a repetirse.

- `String toUpperCase()`

Retorna un String con el contenido convertido todo a mayúsculas.

- `String toLowerCase()`

Retorna un String con el contenido convertido todo a minúsculas.

## Problema 1:

Confeccionar una clase que solicite el ingreso de dos String y luego emplee los métodos más comunes de la clase String.

### Programa:

```
import java.util.Scanner;
public class Cadenas1 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        String cad1;
        String cad2;
        System.out.print("Ingrese la primer cadena:");
        cad1=teclado.nextLine();
        System.out.print("Ingrese la segunda cadena:");
        cad2=teclado.nextLine();
        if (cad1.equals(cad2)==true) {
            System.out.println(cad1+" es exactamente igual a "+cad2);
        } else {
            System.out.println(cad1+" no es exactamente igual a "+cad2);
        }
        if (cad1.equalsIgnoreCase(cad2)==true) {
            System.out.println(cad1+" es igual a "+cad2+" sin tener en
cuenta mayúsculas/minúsculas");
        } else {
            System.out.println(cad1+" no es igual a "+cad2+" sin tener en
cuenta mayúsculas/minúsculas");
        }
        if (cad1.compareTo(cad2)==0) {
            System.out.println(cad1+" es exactamente igual a "+cad2);
        } else {
            if (cad1.compareTo(cad2)>0) {
```

```

        System.out.println(cad1+ " es mayor alfabéticamente que
"+cad2);
    } else {
        System.out.println(cad2+ " es mayor alfabéticamente que
"+cad1);
    }
}
char carac1=cad1.charAt(0);
System.out.println("El primer caracter de "+cad1+" es "+carac1);
int largo=cad1.length();
System.out.println("El largo del String "+cad1+" es "+largo);
String cad3=cad1.substring(0,3);
System.out.println("Los primeros tres caracteres de "+cad1+" son
"+cad3);
int pos1=cad1.indexOf(cad2);
if (pos1== -1) {
    System.out.println(cad2+ " no está contenido en "+cad1);
} else {
    System.out.println(cad2+ " está contenido en "+cad1+" a partir
de la posición "+pos1);
}
System.out.println(cad1+ " convertido a mayúsculas es
"+cad1.toUpperCase());
System.out.println(cad1+ " convertido a minúsculas es
"+cad1.toLowerCase());
}
}
}

```

Para cargar los dos String utilizamos en este caso el método `nextLine` para permitir ingresar espacios en blanco:

```

System.out.print("Ingrese la primer cadena:");
cad1=teclado.nextLine();
System.out.print("Ingrese la segunda cadena:");
cad2=teclado.nextLine();

```

## Problemas propuestos

1. Realizar una clase, que permita cargar una dirección de mail en el constructor, luego en otro método mostrar un mensaje si contiene el carácter '@'.
2. Cargar un String por teclado e implementar los siguientes métodos:
  - a) Imprimir la primera mitad de los caracteres de la cadena.
  - b) Imprimir el último carácter.
  - c) Imprimirla en forma inversa.
  - d) Imprimir cada carácter del String separado con un guión.
  - e) Imprimir la cantidad de vocales almacenadas.
  - f) Implementar un método que verifique si la cadena se lee igual de izquierda a derecha tanto como de derecha a izquierda (ej. neuquen se lee igual en las dos direcciones)
3. Desarrollar un programa que solicite la carga de una clave. La clase debe tener dos métodos uno para la carga y otro que muestre si la clave es la correcta (la clave a comparar es "123abc")

4. Confeccionar un programa que permita cargar los nombres de 5 personas y sus mail, luego implementar los siguientes métodos:
  - a) Mostrar por pantalla los datos.
  - b) Consulta del mail ingresando su nombre.
  - c) Mostrar los mail que no tienen el carácter @.
5. Codifique un programa que permita cargar una oración por teclado, luego mostrar cada palabra ingresada en una línea distinta.

Por ejemplo si cargo:

La mañana está fría.

Debe aparecer:

La  
mañana  
está  
fría.

[Solución](#)

[\*\*Retornar\*\*](#)

# 28 - Colaboración de clases

[Listado completo de tutoriales](#)

Normalmente un problema resuelto con la metodología de programación orientada a objetos no interviene una sola clase, sino que hay muchas clases que interactúan y se comunican.

Plantearemos un problema separando las actividades en dos clases.

## Problema 1:

Un banco tiene 3 clientes que pueden hacer depósitos y extracciones. También el banco requiere que al final del día calcule la cantidad de dinero que hay depositada.

Lo primero que hacemos es identificar las clases:

Podemos identificar la clase Cliente y la clase Banco.

Luego debemos definir los atributos y los métodos de cada clase:

```
Cliente
    atributos
        nombre
        monto
    métodos
        constructor
        depositar
        extraer
        retornarMonto

Banco
    atributos
        3 Cliente (3 objetos de la clase Cliente)
        1 Scanner (Para poder hacer la entrada de datos por teclado)
    métodos
        constructor
        operar
        depositosTotales
```

Creamos un proyecto en Eclipse llamado: Proyecto1 y dentro del proyecto creamos dos clases llamadas: Cliente y Banco.

## Programa:

```
public class Cliente {
    private String nombre;
    private int monto;
```

```

public Cliente(String nom) {
    nombre=nom;
    monto=0;
}

public void depositar(int m) {
    monto=monto+m;
}

public void extraer(int m) {
    monto=monto-m;
}

public int retornarMonto() {
    return monto;
}

public void imprimir() {
    System.out.println(nombre+" tiene depositado la suma de "+monto);
}
}

public class Banco {
    private Cliente cliente1,cliente2,cliente3;

    public Banco() {
        cliente1=new Cliente("Juan");
        cliente2=new Cliente("Ana");
        cliente3=new Cliente("Pedro");
    }

    public void operar() {
        cliente1.depositar (100);
        cliente2.depositar (150);
        cliente3.depositar (200);
        cliente3.extraer (150);
    }

    public void depositosTotales ()
    {
        int t = cliente1.retornarMonto () + cliente2.retornarMonto () +
cliente3.retornarMonto ();
        System.out.println ("El total de dinero en el banco es:" + t);
        cliente1.imprimir();
        cliente2.imprimir();
        cliente3.imprimir();
    }

    public static void main(String[] ar) {
        Banco bancol=new Banco();
        bancol.operar();
        bancol.depositosTotales();
    }
}

```

Analicemos la implementación del problema.

Los atributos de una clase normalmente son privados para que no se tenga acceso directamente desde otra clase, los atributos son modificados por los métodos de la misma clase:

```
private String nombre;  
private int monto;
```

El constructor recibe como parámetro el nombre del cliente y lo almacena en el atributo respectivo e inicializa el atributo monto en cero:

```
public Cliente(String nom) {  
    nombre=nom;  
    monto=0;  
}
```

Los métodos depositar y extraer actualizan el atributo monto con el dinero que llega como parámetro (para simplificar el problema no hemos validado que cuando se extrae dinero el atributo monto quede con un valor negativo):

```
public void depositar(int m) {  
    monto=monto+m;  
}  
  
public void extraer(int m) {  
    monto=monto-m;  
}
```

El método retornarMonto tiene por objetivo comunicar al Banco la cantidad de dinero que tiene el cliente (recordemos que como el atributo monto es privado de la clase, debemos tener un método que lo retorne):

```
public int retornarMonto() {  
    return monto;  
}
```

Por último el método imprimir muestra nombre y el monto de dinero del cliente:

```
public void imprimir() {  
    System.out.println(nombre+" tiene depositado la suma de "+monto);  
}
```

Como podemos observar la clase Cliente no tiene función main. Entonces donde definimos objetos de la clase Cliente?

La respuesta a esta pregunta es que en la clase Banco definimos tres objetos de la clase Cliente.

Veamos ahora la clase Banco que requiere la colaboración de la clase Cliente.

Primero definimos tres atributos de tipo Cliente:

```
public class Banco {  
    private Cliente cliente1,cliente2,cliente3;
```

En el constructor creamos los tres objetos (cada vez que creamos un objeto de la clase Cliente debemos pasar a su constructor el nombre del cliente, recordemos que su monto de depósito se inicializa con cero):

```
public Banco() {  
    cliente1=new Cliente("Juan");  
    cliente2=new Cliente("Ana");  
    cliente3=new Cliente("Pedro");  
}
```

El método operar del banco (llamamos a los métodos depositar y extraer de los clientes):

```
public void operar() {  
    cliente1.depositar (100);  
    cliente2.depositar (150);  
    cliente3.depositar (200);  
    cliente3.extraer (150);  
}
```

El método depositosTotales obtiene el monto depositado de cada uno de los tres clientes, procede a mostrarlos y llama al método imprimir de cada cliente para poder mostrar el nombre y depósito:

```
public void depositosTotales ()  
{  
    int t = cliente1.retornarMonto () + cliente2.retornarMonto () +  
cliente3.retornarMonto ();  
    System.out.println ("El total de dinero en el banco es:" + t);  
    cliente1.imprimir();  
    cliente2.imprimir();  
    cliente3.imprimir();  
}
```

Por último en la main definimos un objeto de la clase Banco (la clase Banco es la clase principal en nuestro problema):

```
public static void main(String[] ar) {  
    Banco banco1=new Banco();  
    banco1.operar();
```

```
        banco1.depositosTotales();  
    }  
}
```

## Problema 2:

Plantear un programa que permita jugar a los dados. Las reglas de juego son: se tiran tres dados si los tres salen con el mismo valor mostrar un mensaje que "gano", sino "perdió".

Lo primero que hacemos es identificar las clases:

Podemos identificar la clase Dado y la clase JuegoDeDados.

Luego los atributos y los métodos de cada clase:

Dado  
atributos  
    valor  
métodos  
    tirar  
    imprimir  
    retornarValor

JuegoDeDados  
atributos  
    3 Dado (3 objetos de la clase Dado)  
métodos  
    constructor  
    jugar

Creamos un proyecto en Eclipse llamado: Proyecto2 y dentro del proyecto creamos dos clases llamadas: Dado y JuegoDeDados.

## Programa:

```
public class Dado {  
    private int valor;  
  
    public void tirar() {  
        valor=1+(int)(Math.random()*6);  
    }  
  
    public void imprimir() {  
        System.out.println("El valor del dado es:"+valor);  
    }  
  
    public int retornarValor() {  
        return valor;  
    }  
}
```

```

public class JuegoDeDados {
    private Dado dado1,dado2,dado3;

    public JuegoDeDados() {
        dado1=new Dado();
        dado2=new Dado();
        dado3=new Dado();
    }

    public void jugar() {
        dado1.tirar();
        dado1.imprimir();
        dado2.tirar();
        dado2.imprimir();
        dado3.tirar();
        dado3.imprimir();
        if (dado1.retornarValor()==dado2.retornarValor() &&
            dado1.retornarValor()==dado3.retornarValor()) {
            System.out.println("Ganó");
        } else {
            System.out.println("Perdió");
        }
    }

    public static void main(String[] ar){
        JuegoDeDados j=new JuegoDeDados();
        j.jugar();
    }
}

```

La clase dado define el atributo "valor" donde almacenamos un valor aleatorio que representa el número que sale al tirarlo.

```

public class Dado {
    private int valor;

```

El método tirar almacena el valor aleatorio (para generar un valor aleatorio utilizamos el método random de la clase Math, el mismo genera un valor real comprendido entre 0 y 1, pero nunca 0 o 1. Puede ser un valor tan pequeño como 0.0001 o tan grande como 0.9999. Luego este valor generado multiplicado por 6 y antecediendo (int) obtenemos la parte entera de dicho producto):

```

public void tirar() {
    valor=1+(int)(Math.random()*6);
}

```

Como vemos le sumamos uno ya que el producto del valor aleatorio con seis puede generar números enteros entre 0 y 5.

El método imprimir de la clase Dado muestra por pantalla el valor del dado:

```
public void imprimir() {
    System.out.println("El valor del dado es:"+valor);
}
```

Por último el método que retorna el valor del dado (se utiliza en la otra clase para ver si los tres dados generaron el mismo valor):

```
public int retornarValor() {
    return valor;
}
```

La clase JuegoDeDados define tres atributos de la clase Dado (con esto decimos que la clase Dado colabora con la clase JuegoDeDados):

```
public class JuegoDeDados {
    private Dado dado1,dado2,dado3;
```

En el constructor procedemos a crear los tres objetos de la clase Dado:

```
public JuegoDeDados() {
    dado1=new Dado();
    dado2=new Dado();
    dado3=new Dado();
}
```

En el método jugar llamamos al método tirar de cada dado, pedimos que se imprima el valor generado y finalmente procedemos a verificar si se ganó o no:

```
public void jugar() {
    dado1.tirar();
    dado1.imprimir();
    dado2.tirar();
    dado2.imprimir();
    dado3.tirar();
    dado3.imprimir();
    if (dado1.retornarValor()==dado2.retornarValor() &&
        dado1.retornarValor()==dado3.retornarValor()) {
        System.out.println("Ganó");
    } else {
        System.out.println("Perdió");
    }
}
```

En la main creamos solo un objeto de la clase principal (en este caso la clase principal es el JuegoDeDados):

```
public static void main(String[] ar){
    JuegoDeDados j=new JuegoDeDados();
    j.jugar();
```

}

## Problemas propuestos

1. Plantear una clase Club y otra clase Socio.

La clase Socio debe tener los siguientes atributos privados: nombre y la antigüedad en el club (en años). En el constructor pedir la carga del nombre y su antigüedad. La clase Club debe tener como atributos 3 objetos de la clase Socio. Definir una responsabilidad para imprimir el nombre del socio con mayor antigüedad en el club.

[Solución](#)

[Retornar](#)

Vimos en el concepto anterior que dos clases pueden estar relacionadas por la colaboración. Ahora veremos otro tipo de relaciones entre clases que es la Herencia.

La herencia significa que se pueden crear nuevas clases partiendo de clases existentes, que tendrá todas los atributos y los métodos de su 'superclase' o 'clase padre' y además se le podrán añadir otros atributos y métodos propios.

## clase padre

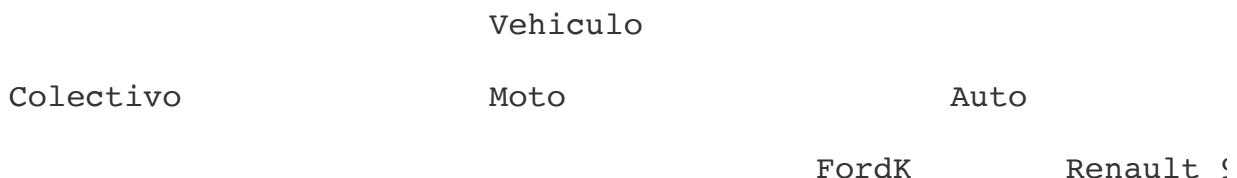
Clase de la que desciende o deriva una clase. Las clases hijas (descendientes) heredan (incorporan) automáticamente los atributos y métodos de la clase padre.

## Subclase

Clase descendiente de otra. Hereda automáticamente los atributos y métodos de su superclase. Es una especialización de otra clase. Admiten la definición de nuevos atributos y métodos para aumentar la especialización de la clase.

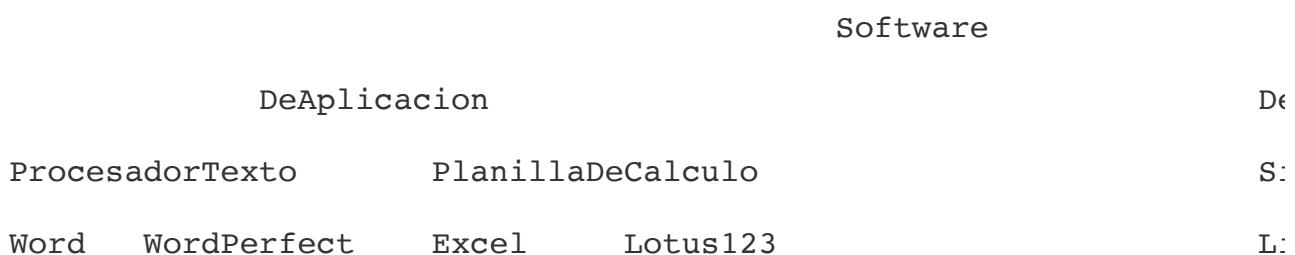
Veamos algunos ejemplos teóricos de herencia:

1) Imaginemos la clase Vehículo. Qué clases podrían derivar de ella?



Siempre hacia abajo en la jerarquía hay una especialización (las subclases añaden nuevos atributos y métodos).

2) Imaginemos la clase Software. Qué clases podrían derivar de ella?



El primer tipo de relación que habíamos visto entre dos clases, es la de colaboración. Recordemos que es cuando una clase contiene un objeto de otra clase como atributo. Cuando la relación entre dos clases es del tipo "...tiene un..." o "...es parte de...", no debemos implementar herencia. Estamos frente a una relación de colaboración de clases no de herencia.

Si tenemos una ClaseA y otra ClaseB y notamos que entre ellas existe una relación de tipo "... tiene un...", no debe implementarse herencia sino declarar en la clase ClaseA un atributo de la clase ClaseB.

Por ejemplo: tenemos una clase Auto, una clase Rueda y una clase Volante. Vemos que la relación entre ellas es: Auto "...tiene 4..." Rueda, Volante "...es parte de..." Auto; pero la clase Auto no debe derivar de Rueda ni Volante de Auto porque la relación no es de tipo-subtipo sino de colaboración. Debemos declarar en la clase Auto 4 atributos de tipo Rueda y 1 de tipo Volante.

Luego si vemos que dos clases responden a la pregunta ClaseA "...es un..." ClaseB es posible que haya una relación de herencia.

Por ejemplo:

```
Auto "es un" Vehiculo
Circulo "es una" Figura
Mouse "es un" DispositivoEntrada
Suma "es una" Operacion
```

## Problema 1:

Ahora plantearemos el primer problema utilizando herencia. Supongamos que necesitamos implementar dos clases que llamaremos Suma y Resta. Cada clase tiene como atributo valor1, valor2 y resultado. Los métodos a definir son cargar1 (que inicializa el atributo valor1), cargar2 (que inicializa el atributo valor2), operar (que en el caso de la clase "Suma" suma los dos atributos y en el caso de la clase "Resta" hace la diferencia entre valor1 y valor2, y otro método mostrarResultado.

Si analizamos ambas clases encontramos que muchos atributos y métodos son idénticos. En estos casos es bueno definir una clase padre que agrupe dichos atributos y responsabilidades comunes.

La relación de herencia que podemos disponer para este problema es:

Operacion

Suma

Resta

Solamente el método operar es distinto para las clases Suma y Resta (esto hace que no lo podamos disponer en la clase Operacion), luego los métodos cargar1, cargar2 y mostrarResultado son idénticos a las dos clases, esto hace que podamos disponerlos en la clase Operacion. Lo mismo los atributos valor1, valor2 y resultado se definirán en la clase padre Operacion.

Crear un proyecto y luego crear cuatro clases llamadas: Operacion, Suma, Resta y Prueba

### Programa:

```
import java.util.Scanner;
public class Operacion {
    protected Scanner teclado;
    protected int valor1;
    protected int valor2;
    protected int resultado;
    public Operacion() {
        teclado=new Scanner(System.in);
    }

    public void cargar1() {
        System.out.print("Ingrese el primer valor:");
        valor1=teclado.nextInt();
    }

    public void cargar2() {
        System.out.print("Ingrese el segundo valor:");
        valor2=teclado.nextInt();
    }

    public void mostrarResultado() {
        System.out.println(resultado);
    }
}

public class Suma extends Operacion{
    void operar() {
        resultado=valor1+valor2;
    }
}
```

```
}
```

```
public class Resta extends Operacion {  
    public void operar(){  
        resultado=valor1-valor2;  
    }  
}  
  
public class Prueba {  
    public static void main(String[] ar) {  
        Suma sumal=new Suma();  
        sumal.cargar1();  
        sumal.cargar2();  
        sumal.operar();  
        System.out.print("El resultado de la suma es:")  
        sumal.mostrarResultado();  
        Resta restal=new Resta();  
        restal.cargar1();  
        restal.cargar2();  
        restal.operar();  
        System.out.print("El resultado de la resta es:")  
        restal.mostrarResultado();  
    }  
}
```

La clase Operación define los cuatro atributos:

```
import java.util.Scanner;  
public class Operacion {  
    protected Scanner teclado;  
    protected int valor1;  
    protected int valor2;  
    protected int resultado;
```

Ya veremos que definimos los atributos con este nuevo modificador de acceso (protected) para que la subclase tenga acceso a dichos atributos. Si los definimos

private las subclases no pueden acceder a dichos atributos.

Los métodos de la clase Operacion son:

```
public Operacion() {
    teclado=new Scanner(System.in);
}

public void cargar1() {
    System.out.print("Ingrese el primer valor:");
    valor1=teclado.nextInt();
}

public void cargar2() {
    System.out.print("Ingrese el segundo valor:");
    valor2=teclado.nextInt();
}

public void mostrarResultado() {
    System.out.println(resultado);
}
```

Ahora veamos como es la sintaxis para indicar que una clase hereda de otra:

```
public class Suma extends Operacion{
```

Utilizamos la palabra clave extends y seguidamente el nombre de la clase padre (con esto estamos indicando que todos los métodos y atributos de la clase Operación son también métodos de la clase Suma).

Luego la característica que añade la clase Suma es el siguiente método:

```
void operar() {
    resultado=valor1+valor2;
}
```

El método operar puede acceder a los atributos heredados (siempre y cuando los mismos se declaren protected, en caso que sean private si bien lo hereda de la clase padre solo los pueden modificar métodos de dicha clase padre).

Ahora podemos decir que la clase Suma tiene cinco métodos (cuatro heredados y uno propio) y 3 atributos (todos heredados)

Luego en otra clase creamos un objeto de la clase Suma:

```
public class Prueba {
    public static void main(String[] ar) {
```

```

        Suma sumal=new Suma();
        sumal.cargar1();
        sumal.cargar2();
        sumal.operar();
        System.out.print("El resultado de la suma es:");
        sumal.mostrarResultado();
        Resta restal=new Resta();

        restal.cargar1();
        restal.cargar2();
        restal.operar();
        System.out.print("El resultado de la resta es:");
        restal.mostrarResultado();
    }
}

```

Podemos llamar tanto al método propio de la clase Suma "operar()" como a los métodos heredados. Quien utilice la clase Suma solo debe conocer que métodos públicos tiene (independientemente que pertenezcan a la clase Suma o a una clase superior)

La lógica es similar para declarar la clase Resta.

La clase Operación agrupa en este caso un conjunto de atributos y métodos comunes a un conjunto de subclases (Suma, Resta). No tiene sentido definir objetos de la clase Operacion.

El planteo de jerarquías de clases es una tarea compleja que requiere un perfecto entendimiento de todas las clases que intervienen en un problema, cuales son sus atributos y responsabilidades.

## Problemas propuestos

1. Confeccionar una clase Persona que tenga como atributos el nombre y la edad. Definir como responsabilidades un método que cargue los datos personales y otro que los imprima.  
Plantear una segunda clase Empleado que herede de la clase Persona. Añadir un atributo sueldo y los métodos de cargar el sueldo e imprimir su sueldo.  
Definir un objeto de la clase Persona y llamar a sus métodos. También crear un objeto de la clase Empleado y llamar a sus métodos.

[Solución](#)

[Retornar](#)

# 30 - Interfaces visuales (componentes Swing)

[Listado completo de tutoriales](#)

Hasta ahora hemos resuelto todos los algoritmos haciendo las salidas a través de una consola en modo texto. La realidad que es muy común la necesidad de hacer la entrada y salida de datos mediante una interfaz más amigables con el usuario.

En Java existen varias librerías de clase para implementar interfaces visuales. Utilizaremos las componentes Swing.

## Problema 1:

Confeccionar el programa "Hola Mundo" utilizando una interfaz gráfica de usuario.

## Programa:

```
import javax.swing.*;
public class Formulario extends JFrame{
    private JLabel label1;
    public Formulario() {
        setLayout(null);
        label1=new JLabel("Hola Mundo.");
        label1.setBounds(10,20,200,30);
        add(label1);
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(10,10,400,300);
        formulario1.setVisible(true);
    }
}
```

Hasta ahora habíamos utilizado la clase Scanner para hacer la entrada de datos por teclado. Dicha clase debemos importarla en nuestro programa con la sintaxis:

```
import java.util.Scanner;
```

Otra sintaxis para importarla es:

```
import java.util.*;
```

Si disponemos un \* indicamos que importe todas las clases del paquete java.util.

Ahora bien las componentes Swing hay que importarlas del paquete javax.swing. Cuando debemos importar varias componentes de un paquete es más conveniente utilizar el asterisco que indicar cada clase a importar:

```
import javax.swing.JFrame;
import javax.swing.JLabel;
```

En lugar de las dos líneas anteriores es mejor utilizar la sintaxis:

```
import javax.swing.*;
```

La clase JFrame encapsula el concepto de una ventana. Luego para implementar una aplicación que muestre una ventana debemos plantear una clase que herede de la clase JFrame:

```
public class Formulario extends JFrame{
```

Con la sintaxis anterior estamos indicando que la clase Formulario hereda todos los métodos y propiedades de la clase JFrame.

Para mostrar un texto dentro de una ventana necesitamos requerir la colaboración de la clase JLabel (que tiene por objetivo mostrar un texto dentro de un JFrame)

Definimos luego como atributo de la clase un objeto de la clase JLabel:

```
private JLabel label1;
```

En el constructor de la clase llamamos al método heredado de la clase JFrame llamado setLayout y le pasamos como parámetro un valor null, con esto estamos informándole a la clase JFrame que utilizaremos posicionamiento absoluto para los controles visuales dentro del JFrame.

```
public Formulario() {
    setLayout(null);
```

Luego tenemos que crear el objeto de la clase JLabel y pasarle como parámetro al constructor el texto a mostrar:

```
label1=new JLabel("Hola Mundo.");
```

Ubicamos al objeto de la clase JLabel llamando al método setBounds, este requiere como parámetros la columna, fila, ancho y alto del JLabel. Finalmente llamamos al método add (método heredado de la clase JFrame) que tiene como objetivo añadir el control JLabel al control JFrame.

```
label1=new JLabel("Hola Mundo.");
label1.setBounds(10,20,200,30);
add(label1);
}
```

Finalmente debemos codificar la main donde creamos un objeto de la clase Formulario, llamamos al método setBounds para ubicar y dar tamaño al control y mediante el método setVisible hacemos visible el JFrame:

```
public static void main(String[] ar) {
    Formulario formulario1=new Formulario();
    formulario1.setBounds(10,10,400,300);
    formulario1.setVisible(true);
}
```

Cuando ejecutamos nuestro proyecto tenemos como resultado una ventana similar a esta:



[Retornar](#)

# 31 - Swing - JFrame

[Listado completo de tutoriales](#)

La componente básica que requerimos cada vez que implementamos una interfaz visual con la librería Swing es la clase JFrame. Esta clase encapsula una Ventana clásica de cualquier sistema operativo con entorno gráfico (Windows, OS X, Linux etc.)

Hemos dicho que esta clase se encuentra en el paquete javax.swing y como generalmente utilizamos varias clases de este paquete luego para importarla utilizamos la sintaxis:

```
import javax.swing.*;
```

Podemos hacer una aplicación mínima con la clase JFrame:

```
import javax.swing.JFrame;
public class Formulario {
    public static void main(String[] ar) {
        JFrame f=new JFrame();
        f.setBounds(10,10,300,200);
        f.setVisible(true);
    }
}
```

Como vemos importamos la clase JFrame del paquete javax.swing:

```
import javax.swing.JFrame;
```

y luego en la main definimos y creamos un objeto de la clase JFrame (llamando luego a los métodos setBounds y setVisible):

```
public static void main(String[] ar) {
    JFrame f=new JFrame();
    f.setBounds(10,10,300,200);
    f.setVisible(true);
}
```

Pero esta forma de trabajar con la clase JFrame es de poca utilidad ya que rara vez necesitemos implementar una aplicación que muestre una ventana vacía.

Lo más correcto es plantear una clase que herede de la clase JFrame y extienda sus responsabilidades agregando botones, etiquetas, editores de línea etc.

Entonces la estructura básica que emplearemos para crear una interfaz visual será:

**Programa:**

```
import javax.swing.*;
public class Formulario extends JFrame{
    public Formulario() {
        setLayout(null);
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(10,20,400,300);
        formulario1.setVisible(true);
    }
}
```

Importamos el paquete donde se encuentra la clase JFrame:

```
import javax.swing.*;
```

Planteamos una clase que herede de la clase JFrame:

```
public class Formulario extends JFrame{
```

En el constructor indicamos que ubicaremos los controles visuales con coordenadas absolutas mediante la desactivación del Layout heredado (más adelante veremos otras formas de ubicar los controles visuales dentro del JFrame):

```
public Formulario() {
    setLayout(null);
}
```

En la main creamos un objeto de la clase y llamamos a los métodos setBounds y setVisible:

```
public static void main(String[] ar) {
    Formulario formulario1=new Formulario();
    formulario1.setBounds(10,20,400,300);
```

```
formulario1.setVisible(true);  
}
```

El método `setBounds` ubica el `JFrame` (la ventana) en la columna 10, fila 20 con un ancho de 400 píxeles y un alto de 300.

Debemos llamar al método `setVisible` y pasarle el valor `true` para que se haga visible la ventana.

## Problemas propuestos

1. Crear una ventana de 1024 píxeles por 800 píxeles. Luego no permitir que el operador modifique el tamaño de la ventana. Sabiendo que hacemos visible al `JFrame` llamando la método `setVisible` pasando el valor `true`, existe otro método llamado `setResizable` que también requiere como parámetro un valor `true` o `false`.

[Solución](#)

[Retornar](#)

# 32 - Swing - JLabel

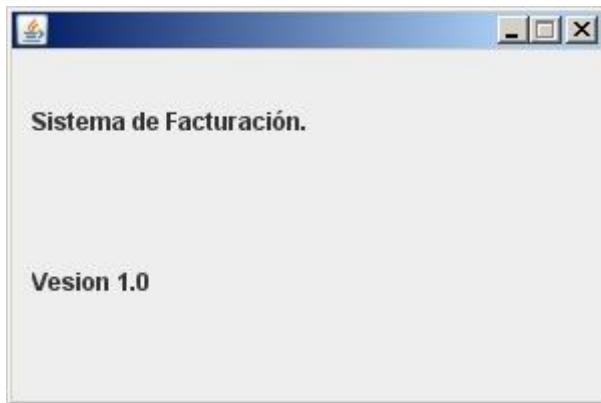
[Listado completo de tutoriales](#)

Veamos la segunda componente de la librería swing llamada JLabel.

La clase JLabel nos permite mostrar básicamente un texto.

## Problema 1:

Confeccionar una ventana que muestre el nombre de un programa en la parte superior y su número de versión en la parte inferior. No permitir modificar el tamaño de la ventana en tiempo de ejecución.



## Programa:

```
import javax.swing.*;
public class Formulario extends JFrame {
    private JLabel label1,label2;
    public Formulario() {
        setLayout(null);
        label1=new JLabel("Sistema de Facturación.");
        label1.setBounds(10,20,300,30);
        add(label1);
        label2=new JLabel("Version 1.0");
        label2.setBounds(10,100,100,30);
        add(label2);
    }
    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,300,200);
    }
}
```

```
        formulario1.setResizable(false);
        formulario1.setVisible(true);
    }
}
```

Importamos el paquete javax.swing donde se encuentran definidas las clase JFrame y JLabel:

```
import javax.swing.*;
```

Heredamos de la clase JFrame:

```
public class Formulario extends JFrame {
```

Definimos dos atributos de la clase JLabel:

```
    private JLabel label1,label2;
```

En el constructor creamos las dos JLabel y las ubicamos llamando al método setBounds, no hay que olvidar de llamar al método add que añade la JLabel al JFrame:

```
public Formulario() {
    setLayout(null);
    label1=new JLabel("Sistema de Facturación.");
    label1.setBounds(10,20,300,30);
    add(label1);
    label2=new JLabel("Vesion 1.0");
    label2.setBounds(10,100,100,30);
    add(label2);
}
```

Por último en la main creamos un objeto de la clase que acabamos de codificar, llamamos al setBounds para ubicar y dar tamaño al JFrame, llamamos al método setResizable pasando el valor false para no permitir modificar el tamaño del JFrame en tiempo de ejecución y finalmente llamamos al método setVisible para que se visualice el JFrame:

```
public static void main(String[] ar) {
    Formulario formulario1=new Formulario();
    formulario1.setBounds(0,0,300,200);
    formulario1.setResizable(false);
```

```
    formulario1.setVisible(true);  
}
```

## Problemas propuestos

1. Crear tres objetos de la clase JLabel, ubicarlos uno debajo de otro y mostrar nombres de colores.

[Solución](#)

[Retornar](#)

El tercer control visual de uso muy común es el que provee la clase JButton. Este control visual muestra un botón.

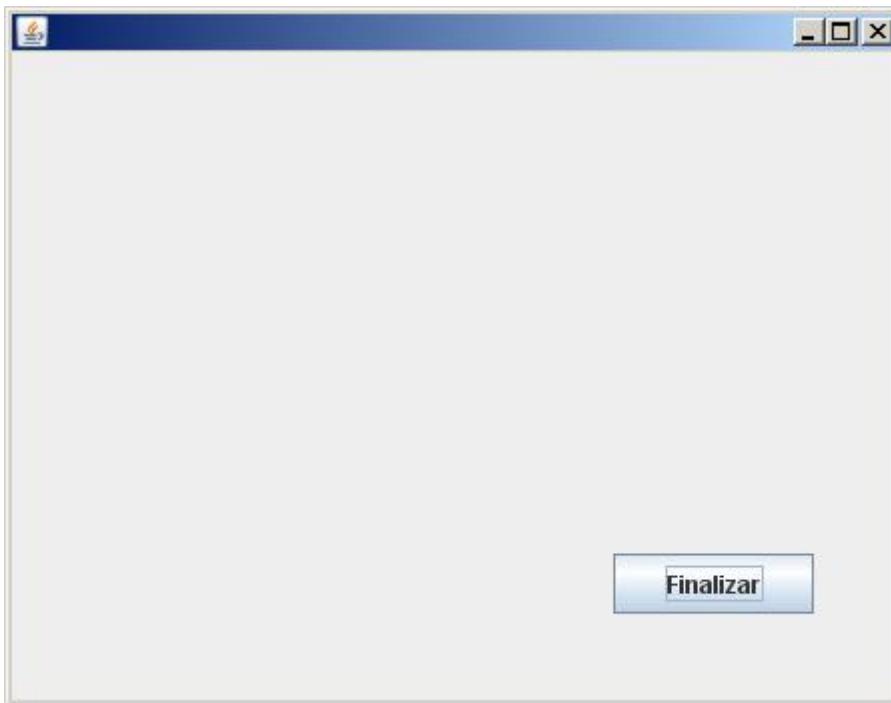
El proceso para añadir botones a un control JFrame es similar a añadir controles de tipo JLabel.

Ahora veremos la captura de eventos con los controles visuales. Uno de los eventos más comunes es cuando hacemos clic sobre un botón.

Java implementa el concepto de interfaces para poder llamar a métodos de una clase existente a una clase desarrollada por nosotros.

## Problema 1:

Confeccionar una ventana que muestre un botón. Cuando se presione finalizar la ejecución del programa Java.



## Programa:

```
import javax.swing.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener {
    JButton boton1;
    public Formulario() {
```

```

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);
        boton1=new JButton("Finalizar");
        boton1.setBounds(300,250,100,30);
        add(boton1);
        boton1.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==boton1) {
            System.exit(0);
        }
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,450,350);
        formulario1.setVisible(true);
    }
}

```

La mecánica para atrapar el clic del objeto de la clase JButton se hace mediante la implementación de una interface. Una interface es un protocolo que permite la comunicación entre dos clases. Una interface contiene uno o más cabecera de métodos, pero no su implementación. Por ejemplo la interface ActionListener tiene la siguiente estructura:

```

interface ActionListener {
    public void actionPerformed(ActionEvent e) {
}

```

Luego las clases que implementen la interface ActionListener deberán especificar el algoritmo del método actionPerformed.

Mediante el concepto de interfaces podemos hacer que desde la clase JButton se puede llamar a un método que implementamos en nuestra clase.

Para indicar que una clase implementará una interface lo hacemos en la declaración de la clase con la sintaxis:

```
public class Formulario extends JFrame implements ActionListener {
```

Con esto estamos diciendo que nuestra clase implementa la interface ActionListener, luego estamos obligados a codificar el método actionPerformed.

Definimos un objeto de la clase JButton:

```
JButton boton1;
```

En el constructor creamos el objeto de la clase JButton y mediante la llamada del método addActionListener le pasamos la referencia del objeto de la clase JButton utilizando la palabra clave this (this almacena la dirección de memoria donde se almacena el objeto de la clase JFrame, luego mediante dicha dirección podemos llamar al método actionPerformed):

```
public Formulario() {  
    setLayout(null);  
    boton1=new JButton("Finalizar");  
    boton1.setBounds(300,250,100,30);  
    add(boton1);  
    boton1.addActionListener(this);  
}
```

El método actionPerformed (este método de la interface ActionListener debe implementarse obligatoriamente, en caso de no codificarlo o equivocarnos en su nombre aparecerá un mensaje de error en tiempo de compilación de nuestro programa).

El método actionPerformed se ejecutará cada vez que hagamos clic sobre el objeto de la clase JButton.

La interface ActionListener y el objeto de la clase ActionEvent que llega como parámetro están definidos en el paquete:

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

Es decir que cada vez que se presiona el botón desde la clase JButton se llama al método actionPerformed y recibe como parámetro un objeto de la clase ActionEvent.

En el método actionPerformed mediante el acceso al método getSource() del objeto que llega como parámetro podemos analizar que botón se presionó:

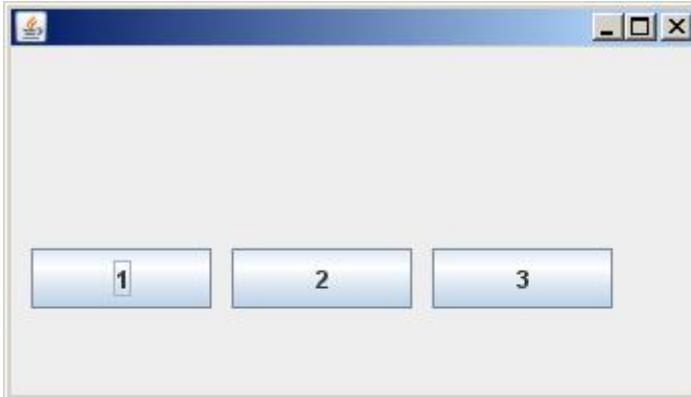
```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==boton1) {  
        System.exit(0);  
    }  
}
```

Si se presionó el botón1 luego el if se verifica verdadero y por lo tanto llamando al método exit de la clase System se finaliza la ejecución del programa.

La main no varía en nada con respecto a problemas anteriores.

## Problema 2:

Confeccionar una ventana que contenga tres objetos de la clase JButton con las etiquetas "1", "2" y "3". Al presionarse cambiar el título del JFrame indicando cuál botón se presionó.



### Programa:

```
import javax.swing.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JButton boton1,boton2,boton3;
    public Formulario() {
        setLayout(null);
        boton1=new JButton("1");
        boton1.setBounds(10,100,90,30);
        add(boton1);
        boton1.addActionListener(this);
        boton2=new JButton("2");
        boton2.setBounds(110,100,90,30);
        add(boton2);
        boton2.addActionListener(this);
        boton3=new JButton("3");
        boton3.setBounds(210,100,90,30);
        add(boton3);
        boton3.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==boton1) {
            setTitle("boton 1");
        }
        if (e.getSource()==boton2) {
            setTitle("boton 2");
        }
        if (e.getSource()==boton3) {
            setTitle("boton 3");
        }
    }
    public static void main(String[] ar){
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,350,200);
        formulario1.setVisible(true);
    }
}
```

Debemos declarar 3 objetos de la clase JButton:

```
private JButton boton1,boton2,boton3;
```

En el constructor creamos los tres objetos de la clase JButton y los ubicamos dentro del control JFrame (también llamamos al método addActionListener para enviarle la dirección del objeto de la clase Formulario):

```
public Formulario() {  
    setLayout(null);  
    boton1=new JButton("1");  
    boton1.setBounds(10,100,90,30);  
    add(boton1);  
    boton1.addActionListener(this);  
    boton2=new JButton("2");  
    boton2.setBounds(110,100,90,30);  
    add(boton2);  
    boton2.addActionListener(this);  
    boton3=new JButton("3");  
    boton3.setBounds(210,100,90,30);  
    add(boton3);  
    boton3.addActionListener(this);  
}
```

Cuando se presiona alguno de los tres botones se ejecuta el método actionPerformed y mediante tres if verificamos cual de los botones se presionó:

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==boton1) {  
        setTitle("boton 1");  
    }  
    if (e.getSource()==boton2) {  
        setTitle("boton 2");  
    }  
    if (e.getSource()==boton3) {  
        setTitle("boton 3");  
    }  
}
```

Según el botón presionado llamamos al método setTitle que se trata de un método heredado de la clase JFrame y que tiene por objetivo mostrar un String en el título de la ventana.

## Problemas propuestos

1. Disponer dos objetos de la clase JButton con las etiquetas: "varón" y "mujer", al presionarse mostrar en la barra de títulos del JFrame la etiqueta del botón presionado.

[Solución](#)

[\*\*Retornar\*\*](#)

Así como podríamos decir que el control JLabel remplaza a la salida estándar System.out.print, el control JTextField cumple la función de la clase Scanner para la entrada de datos.

El control JTextField permite al operador del programa ingresar una cadena de caracteres por teclado.

## Problema 1:

Confeccionar un programa que permita ingresar el nombre de usuario y cuando se presione un botón mostrar el valor ingresado en la barra de títulos del JFrame.



## Programa:

```
import javax.swing.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JTextField textfield1;
    private JLabel label1;
    private JButton boton1;
    public Formulario() {
        setLayout(null);
        label1=new JLabel("Usuario:");
        label1.setBounds(10,10,100,30);
        add(label1);
        textfield1=new JTextField();
        textfield1.setBounds(120,10,150,20);
        add(textfield1);
        boton1=new JButton("Aceptar");
        boton1.setBounds(10,80,100,30);
        add(boton1);
        boton1.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==boton1) {
            String cad=textfield1.getText();
```

```

        setTitle(cad);
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,300,150);
        formulario1.setVisible(true);
    }
}

```

Definimos los tres objetos que colaboran con nuestra aplicación:

```

public class Formulario extends JFrame implements ActionListener{
    private JTextField textfield1;
    private JLabel label1;
    private JButton boton1;

```

En el constructor creamos los tres objetos y los ubicamos:

```

public Formulario() {
    setLayout(null);
    label1=new JLabel("Usuario:");
    label1.setBounds(10,10,100,30);
    add(label1);
    textfield1=new JTextField();
    textfield1.setBounds(120,10,150,20);
    add(textfield1);
    boton1=new JButton("Aceptar");
    boton1.setBounds(10,80,100,30);
    add(boton1);
    boton1.addActionListener(this);
}

```

En el método actionPerformed se verifica si se presionó el objeto JButton, en caso afirmativo extraemos el contenido del control JTextField mediante el método getText:

```

public void actionPerformed(ActionEvent e) {
    if (e.getSource()==boton1) {
        String cad=textfield1.getText();
        setTitle(cad);
    }
}

```

En la variable auxiliar cad almacenamos temporalmente el contenido del JTextField y seguidamente actualizamos el título del control JFrame.

## Problema 2:

Confeccionar un programa que permita ingresar dos números en controles de tipo JTextField, luego sumar los dos valores ingresados y mostrar la suma en la barra del

título del control JFrame.



### Programa:

```
import javax.swing.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JTextField textfield1,textfield2;
    private JButton boton1;
    public Formulario() {
        setLayout(null);
        textfield1=new JTextField();
        textfield1.setBounds(10,10,100,30);
        add(textfield1);
        textfield2=new JTextField();
        textfield2.setBounds(10,50,100,30);
        add(textfield2);
        boton1=new JButton("Sumar");
        boton1.setBounds(10,90,100,30);
        add(boton1);
        boton1.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==boton1) {
            String cad1=textfield1.getText();
            String cad2=textfield2.getText();
            int x1=Integer.parseInt(cad1);
            int x2=Integer.parseInt(cad2);
            int suma=x1+x2;
            String total=String.valueOf(suma);
            setTitle(total);
        }
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,140,150);
        formulario1.setVisible(true);
    }
}
```

Definimos los tres objetos:

```
public class Formulario extends JFrame implements ActionListener{
    private JTextField textfield1,textfield2;
```

```
private JButton boton1;
```

En el método actionPerformed es donde debemos sumar los valores ingresados en los controles de tipo JTextField. Para extraer el contenido de los controles debemos extraerlos con el método getText:

```
String cad1=textfield1.getText();
String cad2=textfield2.getText();
```

Como debemos sumar numéricamente los valores ingresados debemos convertir el contenido de las variables cad1 y cad2 a tipo de dato int:

```
int x1=Integer.parseInt(cad1);
int x2=Integer.parseInt(cad2);
```

El método parseInt de la clase Integer retorna el contenido de cad1 convertido a int (provoca un error si ingresamos caracteres en el control JTextField en lugar de números)

Una vez que tenemos los dos valores en formato numérico procedemos a sumarlos y almacenar el resultado en otra variable auxiliar:

```
int suma=x1+x2;
```

Ahora tenemos que mostrar el valor almacenado en suma en la barra de títulos del control JFrame, pero como el método setTitle requiere un String como parámetro debemos convertirlo a tipo String:

```
String total=String.valueOf(suma);
setTitle(total);
```

Como veremos de acá en adelante es muy común la necesidad de convertir String a enteros y enteros a String:

De String a int:

```
int x1=Integer.parseInt(cad1);
```

De int a String:

```
String total=String.valueOf(suma);
```

## Problemas propuestos

1. Ingresar el nombre de usuario y clave en controles de tipo JTextField. Si se ingresa las cadena (usuario: juan, clave=abc123) luego mostrar en el título del JFrame el mensaje "Correcto" en caso contrario mostrar el mensaje "Incorrecto".

[Solución](#)

[\*\*Retornar\*\*](#)

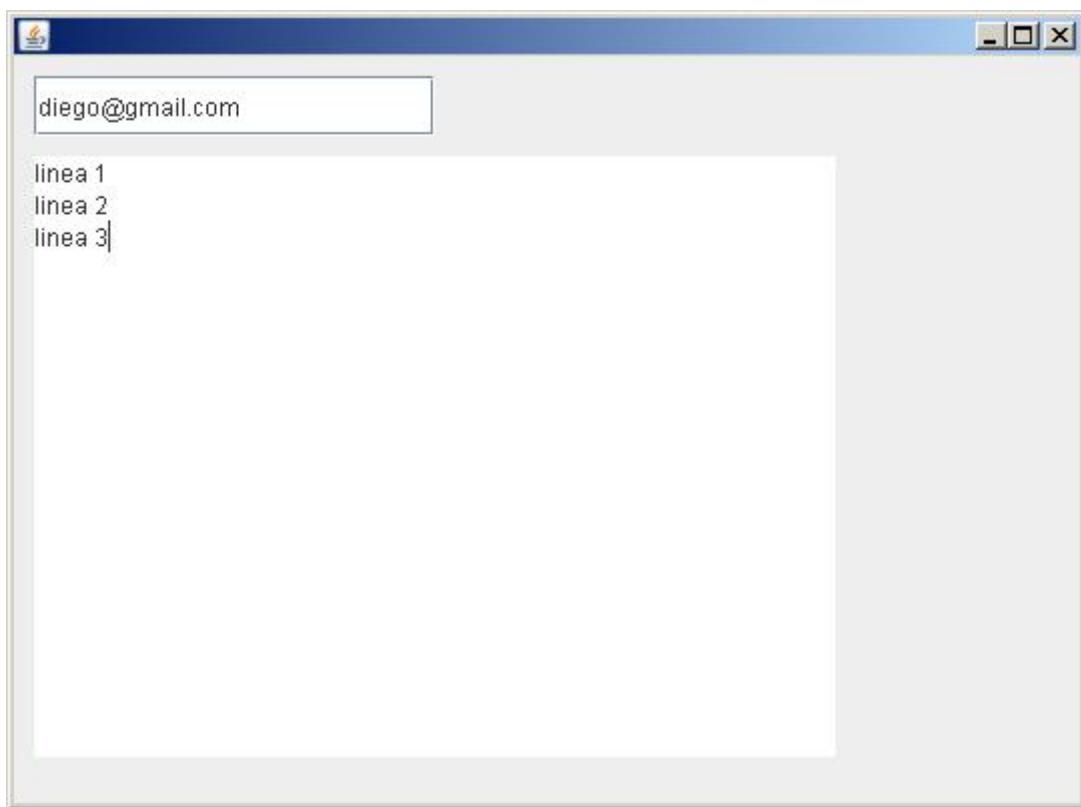
# 35 - Swing - JTextArea

[Listado completo de tutoriales](#)

El control de tipo JTextArea permite ingresar múltiples líneas, a diferencia del control de tipo JTextField.

## Problema 1:

Confeccionar un programa que permita ingresar un mail en un control de tipo JTextField y el cuerpo del mail en un control de tipo JTextArea.



## Programa:

```
import javax.swing.*;
public class Formulario extends JFrame{
    private JTextField textfield1;
    private JTextArea textareal;
    public Formulario() {
        setLayout(null);
        textfield1=new JTextField();
        textfield1.setBounds(10,10,200,30);
        add(textfield1);
        textareal=new JTextArea();
        textareal.setBounds(10,50,400,300);
        add(textareal);
```

```

    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,540,400);
        formulario1.setVisible(true);
    }
}

```

Como vemos crear un control de tipo JTextArea es similar a la creación de controles de tipo JTextField:

```

textareal=new JTextArea();
textareal.setBounds(10,50,400,300);
add(textareal);

```

El inconveniente que tiene este control es que si ingresamos más texto que el que puede visualizar no aparecen las barras de scroll y no podemos ver los caracteres tipeados.

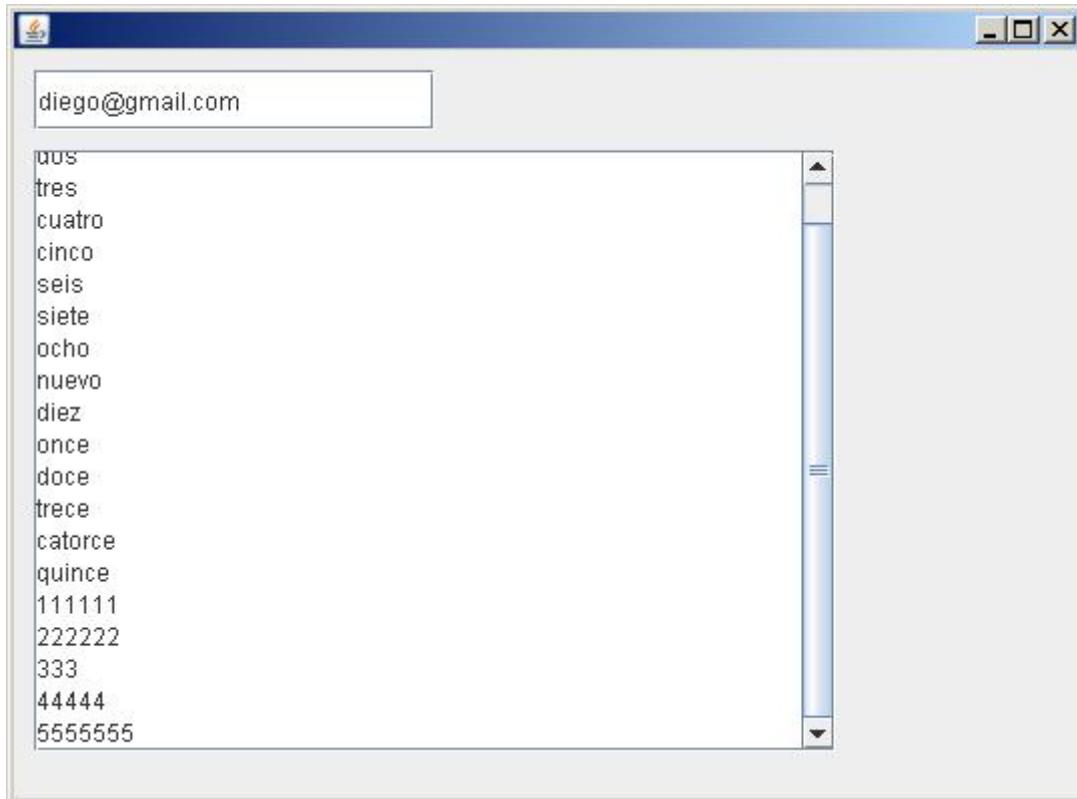
Para salvar el problema anterior debemos crear un objeto de la clase JScrollPane y añadir en su interior el objeto de la clase JTextArea, luego el programa definitivo es el siguiente:

```

import javax.swing.*;
public class Formulario extends JFrame{
    private JTextField textfield1;
    private JScrollPane scrollpanel;
    private JTextArea textareal;
    public Formulario() {
        setLayout(null);
        textfield1=new JTextField();
        textfield1.setBounds(10,10,200,30);
        add(textfield1);
        textareal=new JTextArea();
        scrollpanel=new JScrollPane(textareal);
        scrollpanel.setBounds(10,50,400,300);
        add(scrollpanel);
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,540,400);
        formulario1.setVisible(true);
    }
}

```



Declaramos los dos objetos:

```
private JScrollPane scrollpanel;  
private JTextArea textareal;
```

Primero creamos el objeto de la clase JTextArea:

```
textareal=new JTextArea();
```

Seguidamente creamos el objeto de la clase JScrollPane y le pasamos como parámetro el objeto de la clase JTextArea:

```
scrollpanel=new JScrollPane(textareal);
```

Definimos la posición y tamaño del control de tipo JScrollPane (y no del control JTextArea):

```
scrollpanel.setBounds(10,50,400,300);
```

Finalmente añadimos el control de tipo JScrollPane al JFrame:

```
add(scrollpanel);
```

**Problema 2:**

Confeccionar un programa que permita ingresar en un control de tipo JTextArea una carta. Luego al presionar un botón mostrar un mensaje si la carta contiene el String "argentina".

### Programa:

```
import javax.swing.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JScrollPane scrollpanel;
    private JTextArea textareal;
    private JButton boton1;
    public Formulario() {
        setLayout(null);
        textareal=new JTextArea();
        scrollpanel=new JScrollPane(textareal);
        scrollpanel.setBounds(10,10,300,200);
        add(scrollpanel);
        boton1=new JButton("Verificar");
        boton1.setBounds(10,260,100,30);
        add(boton1);
        boton1.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==boton1) {
            String texto=textareal.getText();
            if (texto.indexOf("argentina")!=-1) {
                setTitle("Si contiene el texto \"argentina\"");
            } else {
                setTitle("No contiene el texto \"argentina\"");
            }
        }
    }
    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,400,380);
        formulario1.setVisible(true);
    }
}
```

Cuando se presiona el botón se ejecuta el método actionPerformed y procedemos a extraer el contenido del control TextArea a través del método getText:

```
String texto=textareal.getText();
```

Luego mediante el método indexOf de la clase String verificamos si el String "argentina" está contenido en la variable texto:

```
if (texto.indexOf("argentina")!=-1) {
    setTitle("Si contiene el texto \"argentina\"");
} else {
```

```
        setTitle("No contiene el texto \"argentina\"");  
    }
```

Si queremos introducir una comilla doble dentro de un String de Java debemos antecederle la barra invertida (luego dicho caracter no se lo considera parte del String):

```
setTitle("Si contiene el texto \"argentina\"");
```

## Problemas propuestos

1. Disponer dos controles de tipo JTextArea, luego al presionar un botón verificar si tienen exactamente el mismo contenido.

[Solución](#)

[Retornar](#)

# 36 - Swing - JComboBox

[Listado completo de tutoriales](#)

El control JComboBox permite seleccionar un String de una lista.

Para inicializar los String que contendrá el JComboBox debemos llamar al método addItem tantas veces como elementos queremos cargar.

Un evento muy útil con este control es cuando el operador selecciona un Item de la lista. Para capturar la selección de un item debemos implementar la interface ItemListener que contiene un método llamada itemStateChanged.

## Problema 1:

Cargar en un JComboBox los nombres de varios colores. Al seleccionar alguno mostrar en la barra de título del JFrame el String seleccionado.



## Programa:

```
import javax.swing.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ItemListener{
    private JComboBox combo1;
    public Formulario() {
        setLayout(null);
        combo1=new JComboBox();
        combo1.setBounds(10,10,80,20);
        add(combo1);
        combo1.addItem("rojo");
        combo1.addItem("verde");
        combo1.addItem("azul");
        combo1.addItem("amarillo");
        combo1.addItem("negro");
        combo1.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent e) {
        if (e.getSource()==combo1) {
            String seleccionado=(String)combo1.getSelectedItem();
        }
    }
}
```

```
        setTitle(seleccionado);
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,200,150);
        formulario1.setVisible(true);
    }
}
```

Indicamos a la clase que implementaremos la interface ItemListener:

```
public class Formulario extends JFrame implements ItemListener{
```

Declaramos un objeto de la clase ComboBox:

```
private JComboBox combo1;
```

En el constructor creamos el objeto de la clase JComboBox:

```
combo1=new JComboBox();
```

Posicionamos el control:

```
combo1.setBounds(10,10,80,20);
```

Añadimos el control al JFrame:

```
add(combo1);
```

Añadimos los String al JComboBox:

```
combo1.addItem("rojo");
combo1.addItem("verde");
combo1.addItem("azul");
combo1.addItem("amarillo");
combo1.addItem("negro");
```

Asociamos la clase que capturará el evento de cambio de item (con this indicamos que esta misma clase capturará el evento):

```
combo1.addItemListener(this);
```

El método itemStateChanged que debemos implementar de la interface ItemListener tiene la siguiente sintaxis:

```

public void itemStateChanged(ItemEvent e) {
    if (e.getSource()==combo1) {
        String seleccionado=(String)combo1.getSelectedItem();
        setTitle(seleccionado);
    }
}

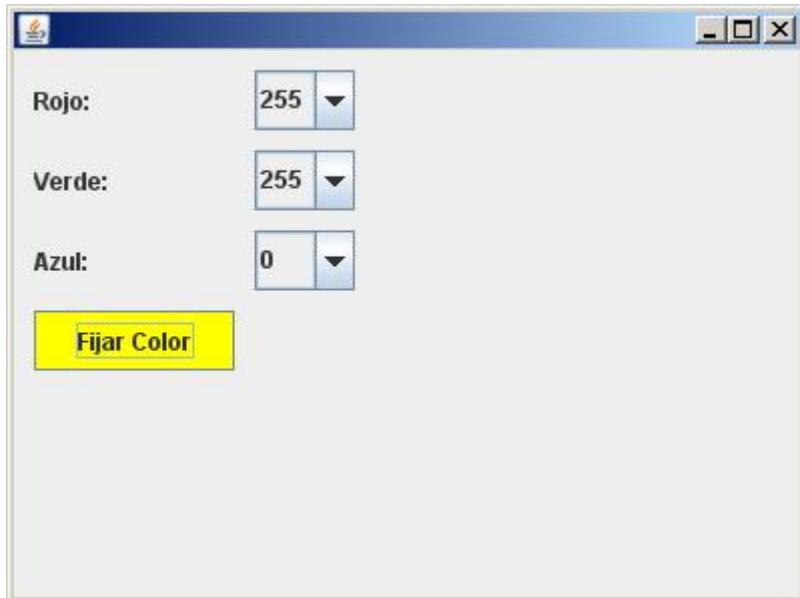
```

Para extraer el contenido del item seleccionado llamamos al método `getSelectedItem()` el cual retorna un objeto de la clase `Object` por lo que debemos indicarle que lo transforme en `String`:

```
String seleccionado=(String)combo1.getSelectedItem();
```

## Problema 2:

Disponer tres controles de tipo `JComboBox` con valores entre 0 y 255 (cada uno representa la cantidad de rojo, verde y azul). Luego al presionar un botón pintar el mismo con el color que se genera combinando los valores de los `JComboBox`.



## Programa:

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JLabel label1,label2,label3;
    private JComboBox combo1,combo2,combo3;
    private JButton boton1;
    public Formulario() {
        setLayout(null);
        label1=new JLabel("Rojo:");

```

```

        label1.setBounds(10,10,100,30);
        add(label1);
        combo1=new JComboBox();
        combo1.setBounds(120,10,50,30);
        for(int f=0;f<=255;f++) {
            combo1.addItem(String.valueOf(f));
        }
        add(combo1);
        label2=new JLabel("Verde:");
        label2.setBounds(10,50,100,30);
        add(label2);
        combo2=new JComboBox();
        combo2.setBounds(120,50,50,30);
        for(int f=0;f<=255;f++) {
            combo2.addItem(String.valueOf(f));
        }
        add(combo2);
        label3=new JLabel("Azul:");
        label3.setBounds(10,90,100,30);
        add(label3);
        combo3=new JComboBox();
        combo3.setBounds(120,90,50,30);
        for(int f=0;f<=255;f++) {
            combo3.addItem(String.valueOf(f));
        }
        add(combo3);
        boton1=new JButton("Fijar Color");
        boton1.setBounds(10,130,100,30);
        add(boton1);
        boton1.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==boton1) {
            String cad1=(String)combo1.getSelectedItem();
            String cad2=(String)combo2.getSelectedItem();
            String cad3=(String)combo3.getSelectedItem();
            int rojo=Integer.parseInt(cad1);
            int verde=Integer.parseInt(cad2);
            int azul=Integer.parseInt(cad3);
            Color color1=new Color(rojo,verde,azul);
            boton1.setBackground(color1);
        }
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,400,300);
        formulario1.setVisible(true);
    }
}

```

Importamos el paquete `java.awt` ya que el mismo contiene la clase `Color`:

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

Implementaremos la interface `ActionListener` ya que tenemos que cambiar el color del botón cuando se lo presione y no haremos actividades cuando cambiemos items de los

controles JComboBox:

```
public class Formulario extends JFrame implements ActionListener{
```

Definimos los siete objetos requeridos en esta aplicación:

```
private JLabel label1,label2,label3;
private JComboBox combo1,combo2,combo3;
private JButton boton1;
```

En el constructor creamos los objetos, primero el control label1 de la clase JLabel:

```
label1=new JLabel("Rojo:");
label1.setBounds(10,10,100,30);
add(label1);
```

Lo mismo hacemos con el objeto combo1:

```
combo1=new JComboBox();
combo1.setBounds(120,10,50,30);
```

Para añadir los 256 elementos del JComboBox disponemos un for y previa a llamar al método addItem convertimos el entero a String:

```
for(int f=0;f<=255;f++) {
    combo1.addItem(String.valueOf(f));
}
add(combo1);
```

En el método actionPerformed cuando detectamos que se presionó el botón procedemos a extraer los tres item seleccionados:

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource()==boton1) {
        String cad1=(String)combo1.getSelectedItem();
        String cad2=(String)combo2.getSelectedItem();
        String cad3=(String)combo3.getSelectedItem();
```

Los convertimos a entero:

```
int rojo=Integer.parseInt(cad1);
int verde=Integer.parseInt(cad2);
int azul=Integer.parseInt(cad3);
```

y creamos finalmente un objeto de la clase Color, el constructor de la clase Color requiere que le pasemos tres valores de tipo int:

```
Color color1=new Color(rojo,verde,azul);
```

Para cambiar el color de fondo del control JButton debemos llamar al método setBackground y pasarle el objeto de la clase Color:

```
boton1.setBackground(color1);
```

## Problemas propuestos

1. Solicitar el ingreso del nombre de una persona y seleccionar de un control JComboBox un país. Al presionar un botón mostrar en la barra del título del JFrame el nombre ingresado y el país seleccionado.

[Solución](#)

[Retornar](#)

# 37 - Swing - JMenuBar, JMenu, JMenuItem

[Listado completo de tutoriales](#)

Ahora veremos como crear un menú de opciones y la captura de eventos de los mismos.

Cuando necesitamos implementar un menú horizontal en la parte superior de un JFrame requerimos de un objeto de la clase JMenuBar, uno o más objetos de la clase JMenu y por último objetos de la clase JMenuItem.

Par la captura de eventos debemos implementar la interface ActionListener y asociarlo a los controles de tipo JMenuItem, el mismo se dispara al presionar con el mouse el JMenuItem.

## Problema 1:

Confeccionaremos un menú de opciones que contenga tres opciones que permita cambiar el color de fondo del JFrame a los colores: rojo, verde y azul.



## Programa:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JMenuBar mb;
    private JMenu menu1;
    private JMenuItem mi1,mi2,mi3;
    public Formulario() {
        setLayout(null);
        mb=new JMenuBar();
        setJMenuBar(mb);
        menu1=new JMenu("Opciones");
        mb.add(menu1);
```

```

        mi1=new JMenuItem("Rojo");
        mi1.addActionListener(this);
        menu1.add(mi1);
        mi2=new JMenuItem("Verde");
        mi2.addActionListener(this);
        menu1.add(mi2);
        mi3=new JMenuItem("Azul");
        mi3.addActionListener(this);
        menu1.add(mi3);
    }

    public void actionPerformed(ActionEvent e) {
        Container f=this.getContentPane();
        if (e.getSource()==mi1) {
            f.setBackground(new Color(255,0,0));
        }
        if (e.getSource()==mi2) {
            f.setBackground(new Color(0,255,0));
        }
        if (e.getSource()==mi3) {
            f.setBackground(new Color(0,0,255));
        }
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(10,20,300,200);
        formulario1.setVisible(true);
    }
}

```

Importamos el paquete javax.swing ya que en el mismo se encuentran las tres clases JMenuBar, JMenu y JMenuItem:

```
import javax.swing.*;
```

Importamos java.awt donde se encuentra la clase Color:

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

Para la captura de eventos mediante la interface ActionListener debemos importar el paquete java.awt.event:

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

Declaramos la clase Formulario, heredamos de la clase JFrame e indicamos que implementaremos la interface ActionListener:

```
public class Formulario extends JFrame implements ActionListener{
```

Definimos un objeto de la clase JMenuBar (no importa que tan grande sea un menú de opciones solo se necesitará un solo objeto de esta clase):

```
private JMenuBar mb;
```

Definimos un objeto de la clase JMenu (esta clase tiene por objeto desplegar un conjunto de objetos de tipo JMenuItem u otros objetos de tipo JMenu:

```
private JMenu menu1;
```

Definimos tres objetos de la clase JMenuItem (estos son los que disparan eventos cuando el operador los selecciona:

```
private JMenuItem mi1,mi2,mi3;
```

En el constructor creamos primero el objeto de la clase JMenuBar y lo asociamos al JFrame llamando al método setJMenuBar:

```
mb=new JMenuBar();
setJMenuBar(mb);
```

Seguidamente creamos un objeto de la clase JMenu, en el constructor pasamos el String que debe mostrar y asociamos dicho JMenu con el JMenuBar llamando al método add de objeto de tipo JMenuBar (Es decir el objeto de la clase JMenu colabora con la clase JMenuBar):

```
menu1=new JMenu("Opciones");
mb.add(menu1);
```

Ahora comenzamos a crear los objetos de la clase JMenuItem y los añadimos al objeto de la clase JMenu (también mediante la llamada al método addActionListener indicamos al JMenuItem que objeto procesará el clic):

```
mi1=new JMenuItem("Rojo");
mi1.addActionListener(this);
menu1.add(mi1);
```

Lo mismo hacemos para los otros dos JMenuItem:

```
mi2=new JMenuItem("Verde");
mi2.addActionListener(this);
menu1.add(mi2);
mi3=new JMenuItem("Azul");
```

```
mi3.addActionListener(this);
menu1.add(mi3);
```

En el método actionPerformed primero obtenemos la referencia al panel asociado con el JFrame:

```
public void actionPerformed(ActionEvent e) {
    Container f=this.getContentPane();
```

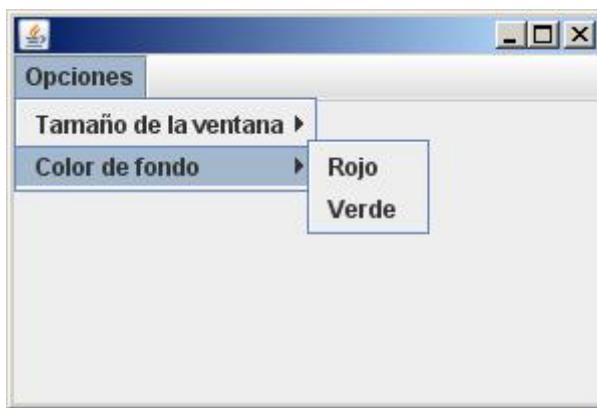
Luego mediante if verificamos cual de los tres JMenuItem fue seleccionado y a partir de esto llamamos al método setBackground del objeto de la clase Container):

```
if (e.getSource()==mi1) {
    f.setBackground(new Color(255,0,0));
}
if (e.getSource()==mi2) {
    f.setBackground(new Color(0,255,0));
}
if (e.getSource()==mi3) {
    f.setBackground(new Color(0,0,255));
}
}
```

## Problema 2:

Confeccionaremos un menú de opciones que contenga además del JMenu de la barra otros dos objetos de la clase JMenu que dependan del primero.

Uno debe mostrar dos JMenuItem que permitan modificar el tamaño del JFrame y el segundo también debe mostrar dos JMenuItem que permitan cambiar el color de fondo.



## Programa:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JMenuBar mb;
```

```

private JMenu menu1,menu2,menu3;
private JMenuItem mi1,mi2,mi3,mi4;
public Formulario() {
    setLayout(null);
    mb=new JMenuBar();
    setJMenuBar(mb);
    menu1=new JMenu("Opciones");
    mb.add(menu1);
    menu2=new JMenu("Tamaño de la ventana");
    menu1.add(menu2);
    menu3=new JMenu("Color de fondo");
    menu1.add(menu3);
    mi1=new JMenuItem("640*480");
    menu2.add(mi1);
    mi1.addActionListener(this);
    mi2=new JMenuItem("1024*768");
    menu2.add(mi2);
    mi2.addActionListener(this);
    mi3=new JMenuItem("Rojo");
    menu3.add(mi3);
    mi3.addActionListener(this);
    mi4=new JMenuItem("Verde");
    menu3.add(mi4);
    mi4.addActionListener(this);
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource()==mi1) {
        setSize(640,480);
    }
    if (e.getSource()==mi2) {
        setSize(1024,768);
    }
    if (e.getSource()==mi3) {
        getContentPane().setBackground(new Color(255,0,0));
    }
    if (e.getSource()==mi4) {
        getContentPane().setBackground(new Color(0,255,0));
    }
}
public static void main(String[] ar) {
    Formulario formulario1=new Formulario();
    formulario1.setBounds(0,0,300,200);
    formulario1.setVisible(true);
}
}

```

Definimos un objeto de la clase JMenuBar, 3 objetos de la clase JMenu y finalmente 4 objetos de la clase JMenuItem:

```

private JMenuBar mb;
private JMenu menu1,menu2,menu3;
private JMenuItem mi1,mi2,mi3,mi4;

```

Es importante notar el orden de creación de los objetos y como los relacionamos unos con otros.

Primero creamos el JMenuBar y lo asociamos con el JFrame:

```
mb=new JMenuBar();
setJMenuBar(mb);
```

Creamos el primer JMenu y lo pasamos como parámetro al JMenuBar mediante el método add:

```
menu1=new JMenu("Opciones");
mb.add(menu1);
```

Ahora creamos el segundo objeto de la clase JMenu y lo asociamos con el primer JMenu creado:

```
menu2=new JMenu("Tamaño de la ventana");
menu1.add(menu2);
```

En forma similar creamos el tercer objeto de la clase JMenu y lo asociamos con el primer JMenu creado:

```
menu3=new JMenu("Color de fondo");
menu1.add(menu3);
```

Finalmente comenzamos a crear los objetos de la clase JMenuItem y los dos primeros los asociamos con el segundo JMenu:

```
mi1=new JMenuItem("640*480");
menu2.add(mi1);
mi1.addActionListener(this);
mi2=new JMenuItem("1024*768");
menu2.add(mi2);
mi2.addActionListener(this);
```

También hacemos lo mismo con los otros dos objetos de tipo JMenuItem pero ahora los asociamos con el tercer JMenu:

```
mi3=new JMenuItem("Rojo");
menu3.add(mi3);
mi3.addActionListener(this);
mi4=new JMenuItem("Verde");
menu3.add(mi4);
mi4.addActionListener(this);
```

En el método actionPerformed si se presiona el mi1 procedemos a redimensionar el JFrame llamando al método setSize y le pasamos dos parámetros que representan el nuevo ancho y alto de la ventana:

```
if (e.getSource()==mi1) {  
    setSize(640,480);  
}
```

De forma similar si se presiona el segundo JMenuItem cambiamos el tamaño de la ventana a 1024 píxeles por 768:

```
if (e.getSource()==mi2) {  
    setSize(1024,768);  
}
```

Para cambiar de color de forma similar al problema anterior mediante el método getContentPane obtenemos la referencia al objeto de la clase Container y llamamos al método setBackground para fijar un nuevo color de fondo:

```
if (e.getSource()==mi3) {  
    getContentPane().setBackground(new Color(255,0,0));  
}  
if (e.getSource()==mi4) {  
    getContentPane().setBackground(new Color(0,255,0));  
}
```

## Problemas propuestos

1. Mediante dos controles de tipo JTextField permitir el ingreso de dos números. Crear un menú que contenga una opción que redimensione el JFrame con los valores ingresados por teclado. Finalmente disponer otra opción que finalice el programa (Finalizamos un programa java llamando al método exit de la clase System: System.exit(0))

[Solución](#)

[Retornar](#)

# 38 - Swing - JCheckBox

[Listado completo de tutoriales](#)

El control JCheckBox permite implementar un cuadro de selección (básicamente un botón de dos estados)

## Problema 1:

Confeccionar un programa que muestre 3 objetos de la clase JCheckBox con etiquetas de tres idiomas. Cuando se lo selecciona mostrar en el título del JFrame todos los JCheckBox seleccionados hasta el momento.



## Programa:

```
import javax.swing.*;
import javax.swing.event.*;
public class Formulario extends JFrame implements ChangeListener{
    private JCheckBox check1,check2,check3;
    public Formulario() {
        setLayout(null);
        check1=new JCheckBox("Inglés");
        check1.setBounds(10,10,150,30);
        check1.addChangeListener(this);
        add(check1);
        check2=new JCheckBox("Francés");
        check2.setBounds(10,50,150,30);
        check2.addChangeListener(this);
        add(check2);
        check3=new JCheckBox("Alemán");
        check3.setBounds(10,90,150,30);
        check3.addChangeListener(this);
        add(check3);
    }
    public void stateChanged(ChangeEvent e){
        String cad="";
        if (check1.isSelected()==true) {
```

```

        cad=cad+"Inglés-";
    }
    if (check2.isSelected()==true) {
        cad=cad+"Francés-";
    }
    if (check3.isSelected()==true) {
        cad=cad+"Alemán-";
    }
    setTitle(cad);
}

public static void main(String[] ar) {
    Formulario formulario1=new Formulario();
    formulario1.setBounds(0,0,300,200);
    formulario1.setVisible(true);
}
}

```

Lo primero y más importante que tenemos que notar que para capturar el cambio de estado del JCheckBox hay que implementar la interface ChangeListener que se encuentra en el paquete:

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

y no en el paquete:

```
import java.awt.event.*
```

Cuando declaramos la clase JFrame indicamos que implementaremos la interface ChangeListener:

```
public class Formulario extends JFrame implements ChangeListener{
```

Definimos tres objetos de la clase JCheckBox:

```
private JCheckBox check1,check2,check3;
```

En el constructor creamos cada uno de los objetos de la clase JCheckBox y llamamos al método addChangeListener indicando quien procesará el evento de cambio de estado:

```
check1=new JCheckBox("Inglés");
check1.setBounds(10,10,150,30);
check1.addChangeListener(this);
add(check1);
```

El método que debemos implementar de la interface ChangeListener es:

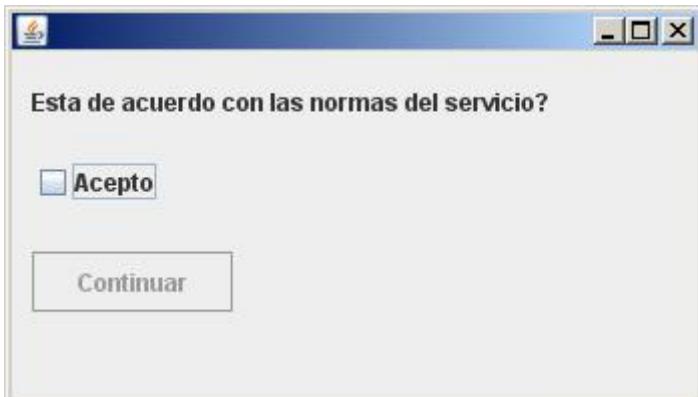
```
public void stateChanged(ChangeEvent e){
```

En este mediante tres if verificamos el estado de cada JCheckBox y concatenamos los String con los idiomas seleccionados:

```
String cad="";
if (check1.isSelected()==true) {
    cad=cad+"Inglés-";
}
if (check2.isSelected()==true) {
    cad=cad+"Francés-";
}
if (check3.isSelected()==true) {
    cad=cad+"Alemán-";
}
setTitle(cad);
```

## Problema 2:

Disponer un control JLabel que muestre el siguiente mensaje: "Esta de acuerdo con las normas del servicio?", luego un JCheckBox y finalmente un objeto de tipo JButton desactivo. Cuando se tilde el JCheckBox debemos activar el botón.



## Programa:

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener,
ChangeListener{
    private JLabel label1;
    private JCheckBox check1;
    private JButton boton1;
    public Formulario() {
        setLayout(null);
        label1=new JLabel("Esta de acuerdo con las normas del servicio?");
        label1.setBounds(10,10,400,30);
        add(label1);
        check1=new JCheckBox("Acepto");
        check1.setBounds(10,50,100,30);
```

```

        check1.addChangeListener(this);
        add(check1);
        boton1=new JButton("Continuar");
        boton1.setBounds(10,100,100,30);
        add(boton1);
        boton1.addActionListener(this);
        boton1.setEnabled(false);
    }

    public void stateChanged(ChangeEvent e) {
        if (check1.isSelected()==true) {
            boton1.setEnabled(true);
        } else {
            boton1.setEnabled(false);
        }
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==boton1) {
            System.exit(0);
        }
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,350,200);
        formulario1.setVisible(true);
    }
}

```

Importamos los paquetes donde se encuentran las interfaces para captura de eventos de objetos de tipo JButton y JCheckBox:

```

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

```

También importamos el paquete donde están definidas las clase JFrame, JButton y JCheckBox:

```

import javax.swing.*;

```

Como debemos implementar dos interfaces las debemos enumerar después de la palabra implements separadas por coma:

```

public class Formulario extends JFrame implements ActionListener, ChangeListener{

```

Definimos los tres objetos:

```

private JLabel label1;
private JCheckBox check1;
private JButton boton1;

```

En el constructor creamos el objeto de tipo JLabel:

```
public Formulario() {  
    setLayout(null);  
    label1=new JLabel("Esta de acuerdo con las normas del servicio?");  
    label1.setBounds(10,10,400,30);  
    add(label1);
```

El objeto de tipo JCheckBox:

```
check1=new JCheckBox("Acepto");  
check1.setBounds(10,50,100,30);  
check1.addChangeListener(this);  
add(check1);
```

y también creamos el objeto de tipo JButton y llamando al método setEnabled con un valor false luego el botón aparece desactivo:

```
boton1=new JButton("Continuar");  
boton1.setBounds(10,100,100,30);  
add(boton1);  
boton1.addActionListener(this);  
boton1.setEnabled(false);
```

Cuando se cambia el estado del control JCheckBox se ejecuta el método stateChanged donde verificamos si está seleccionado procediendo a activar el botón en caso negativo lo desactivamos:

```
public void stateChanged(ChangeEvent e) {  
    if (check1.isSelected()==true) {  
        boton1.setEnabled(true);  
    } else {  
        boton1.setEnabled(false);  
    }  
}
```

El método actionPerformed se ejecuta cuando se presiona el objeto de tipo JButton (debe estar activo para poder presionarlo):

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==boton1) {  
        System.exit(0);  
    }  
}
```

## Problemas propuestos

1. Disponer tres objetos de la clase JCheckBox con nombres de navegadores web. Cuando se presione un botón mostrar en el título del JFrame los programas seleccionados.

[Solución](#)

[\*\*Retornar\*\*](#)

Otro control visual muy común es el JRadioButton que normalmente se muestran un conjunto de JRadioButton y permiten la selección de solo uno de ellos. Se los debe agrupar para que actúen en conjunto, es decir cuando se selecciona uno automáticamente se deben deseleccionar los otros.

## Problema 1:

Confeccionar un programa que muestre 3 objetos de la clase JRadioButton que permitan configurar el ancho y alto del JFrame.



## Programa:

```
import javax.swing.*;
import javax.swing.event.*;
public class Formulario extends JFrame implements ChangeListener{
    private JRadioButton radio1,radio2,radio3;
    private ButtonGroup bg;
    public Formulario() {
        setLayout(null);
        bg=new ButtonGroup();
        radio1=new JRadioButton("640*480");
        radio1.setBounds(10,20,100,30);
        radio1.addChangeListener(this);
        add(radio1);
        bg.add(radio1);
        radio2=new JRadioButton("800*600");
        radio2.setBounds(10,70,100,30);
        radio2.addChangeListener(this);
        add(radio2);
        bg.add(radio2);
        radio3=new JRadioButton("1024*768");
        radio3.setBounds(10,120,100,30);
        add(radio3);
        bg.add(radio3);
    }
    public void stateChanged(ChangeEvent e) {
        if(radio1.isSelected())
            this.setSize(640,480);
        if(radio2.isSelected())
            this.setSize(800,600);
        if(radio3.isSelected())
            this.setSize(1024,768);
    }
}
```

```

        radio3.addChangeListener(this);
        add(radio3);
        bg.add(radio3);
    }

    public void stateChanged(ChangeEvent e) {
        if (radio1.isSelected()) {
            setSize(640,480);
        }
        if (radio2.isSelected()) {
            setSize(800,600);
        }
        if (radio3.isSelected()) {
            setSize(1024,768);
        }
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,350,230);
        formulario1.setVisible(true);
    }
}

```

Importamos los dos paquetes donde están definidas las clases e interfaces para la captura de eventos:

```

import javax.swing.*;
import javax.swing.event.*;

```

Heredamos de la clase JFrame e implementamos la interface ChangeListener para capturar el cambio de selección de objeto de tipo JRadioButton:

```
public class Formulario extends JFrame implements ChangeListener{
```

Definimos tres objetos de la clase JRadioButton y uno de tipo ButtonGroup:

```

private JRadioButton radio1,radio2,radio3;
private ButtonGroup bg;

```

En el constructor creamos primero el objeto de la clase ButtonGroup:

```
bg=new ButtonGroup();
```

Creamos seguidamente el objeto de la clase JRadioButton, definimos su ubicación, llamamos al método addChangeListener para informar que objeto capturará el evento y finalmente añadimos el objeto JRadioButton al JFrame y al ButtonGroup:

```

radio1=new JRadioButton("640*480");
radio1.setBounds(10,20,100,30);

```

```
radio1.addChangeListener(this);
add(radio1);
bg.add(radio1);
```

Exactamente hacemos lo mismo con los otros dos JRadioButton:

```
radio2=new JRadioButton("800*600");
radio2.setBounds(10,70,100,30);
radio2.addChangeListener(this);
add(radio2);
bg.add(radio2);
radio3=new JRadioButton("1024*768");
radio3.setBounds(10,120,100,30);
radio3.addChangeListener(this);
add(radio3);
bg.add(radio3);
```

En el método stateChanged verificamos cual de los tres JRadioButton está seleccionado y procedemos a redimensionar el JFrame:

```
public void stateChanged(ChangeEvent e) {
    if (radio1.isSelected()) {
        setSize(640,480);
    }
    if (radio2.isSelected()) {
        setSize(800,600);
    }
    if (radio3.isSelected()) {
        setSize(1024,768);
    }
}
```

## Problemas propuestos

1. Permitir el ingreso de dos números en controles de tipo JTextField y mediante dos controles de tipo JRadioButton permitir seleccionar si queremos sumarlos o restarlos. Al presionar un botón mostrar en el título del JFrame el resultado de la operación.

[Solución](#)

[Retornar](#)

# 40 - Estructuras dinámicas

[Listado completo de tutoriales](#)

Conocemos algunas estructuras de datos como son los vectores y matrices. No son las únicas. Hay muchas situaciones donde utilizar alguna de estas estructuras nos proporcionará una solución muy ineficiente (cantidad de espacio que ocupa en memoria, velocidad de acceso a la información, etc.)

Ejemplo 1. Imaginemos que debemos realizar un procesador de texto, debemos elegir la estructura de datos para almacenar en memoria las distintas líneas que el operador irá tipeando. Una solución factible es utilizar una matriz de caracteres. Pero como sabemos debemos especificar la cantidad de filas y columnas que ocupará de antemano. Podría ser por ejemplo 2000 filas y 200 columnas. Con esta definición estamos reservando de antemano 800000 bytes de la memoria, no importa si el operador después carga una línea con 20 caracteres, igualmente ya se ha reservado una cantidad de espacio que permanecerá ociosa.

Tiene que existir alguna estructura de datos que pueda hacer más eficiente la solución del problema anterior.

Ejemplo 2. ¿Cómo estarán codificadas las planillas de cálculo? ¿Reservarán espacio para cada casilla de la planilla al principio? Si no la lleno, ¿lo mismo se habrá reservado espacio?

Utilizar una matriz para almacenar todas las casillas de una planilla de cálculo seguro será ineficiente.

Bien, todos estos problemas y muchos más podrán ser resueltos en forma eficiente cuando conozcamos estas nuevas estructuras de datos (Listas, árboles)

[\*\*Retornar\*\*](#)

# 41 - Estructuras dinámicas: Listas

[Listado completo de tutoriales](#)

Una lista es un conjunto de nodos, cada uno de los cuales tiene dos campos: uno de información y un apuntador al siguiente nodo de la lista. Además un apuntador externo señala el primer nodo de la lista.

Representación gráfica de un nodo:

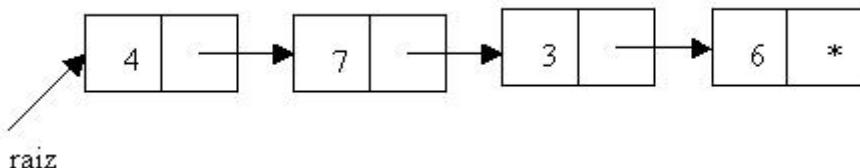
Información   Dirección al siguiente nodo.



La información puede ser cualquier tipo de dato simple, estructura de datos o inclusive uno o más objetos.

La dirección al siguiente nodo es un puntero.

Representación gráfica de una lista:



Como decíamos, una lista es una secuencia de nodos (en este caso cuatro nodos). La información de los nodos en este caso es un entero y siempre contiene un puntero que guarda la dirección del siguiente nodo.

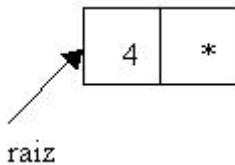
raiz es otro puntero externo a la lista que contiene la dirección del primer nodo.

El estado de una lista varía durante la ejecución del programa:

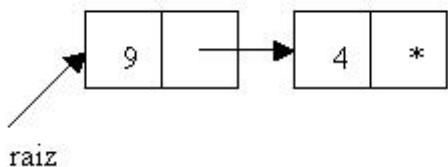


De esta forma representamos gráficamente una lista vacía.

Si insertamos un nodo en la lista quedaría luego:



Si insertamos otro nodo al principio con el valor 9 tenemos:



Lo mismo podemos borrar nodos de cualquier parte de la lista.

Esto nos trae a la mente el primer problema planteado: el desarrollo del procesador de texto. Podríamos utilizar una lista que inicialmente estuviera vacía e introdujéramos un nuevo nodo con cada línea que tipea el operador. Con esta estructura haremos un uso muy eficiente de la memoria.

## Tipos de listas.

Según el mecanismo de inserción y extracción de nodos en la lista tenemos los siguientes tipos:

- Listas tipo pila.
- Listas tipo cola.
- Listas genéricas.

Una lista se comporta como una pila si las inserciones y extracciones las hacemos por un mismo lado de la lista. También se las llama listas LIFO (Last In First Out - último en entrar primero en salir)

Una lista se comporta como una cola si las inserciones las hacemos al final y las extracciones las hacemos por el frente de la lista. También se las llama listas FIFO (First In First Out - primero en entrar primero en salir)

Una lista se comporta como genérica cuando las inserciones y extracciones se realizan en cualquier parte de la lista.

Podemos en algún momento insertar un nodo en medio de la lista, en otro momento al final, borrar uno del frente, borrar uno del fondo o uno interior, etc.

[Retornar](#)

# 42 - Estructuras dinámicas: Listas tipo Pila

[Listado completo de tutoriales](#)

Una lista se comporta como una pila si las inserciones y extracciones las hacemos por un mismo lado de la lista. También se las llama listas LIFO (Last In First Out - último en entrar primero en salir)

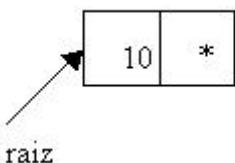
**Importante:** Una pila al ser una lista puede almacenar en el campo de información cualquier tipo de valor (int, char, float, vector de caracteres, un objeto, etc)

Para estudiar el mecanismo de utilización de una pila supondremos que en el campo de información almacena un entero (para una fácil interpretación y codificación)

Inicialmente la PILA está vacía y decimos que el puntero raiz apunta a null (Si apunta a null decimos que no tiene una dirección de memoria):

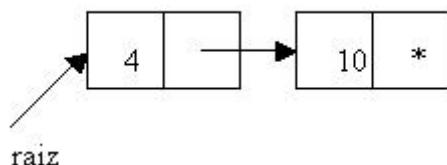


Insertamos un valor entero en la pila: insertar(10)



Luego de realizar la inserción la lista tipo pila queda de esta manera: un nodo con el valor 10 y raiz apunta a dicho nodo. El puntero del nodo apunta a null ya que no hay otro nodo después de este.

Insertamos luego el valor 4: insertar(4)

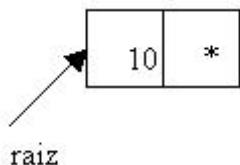


Ahora el primer nodo de la pila es el que almacena el valor cuatro. raiz apunta a dicho nodo. Recordemos que raiz es el puntero externo a la lista que almacena la dirección

del primer nodo. El nodo que acabamos de insertar en el campo puntero guarda la dirección del nodo que almacena el valor 10.

Ahora qué sucede si extraemos un nodo de la pila. ¿Cuál se extrae? Como sabemos en una pila se extrae el último en entrar.

Al extraer de la pila tenemos: extraer()



La pila ha quedado con un nodo.

Hay que tener cuidado que si se extrae un nuevo nodo la pila quedará vacía y no se podrá extraer otros valores (avisar que la pila está vacía)

## Problema 1:

Confeccionar una clase que administre una lista tipo pila (se debe poder insertar, extraer e imprimir los datos de la pila)

## Programa:

```
public class Pila {  
  
    class Nodo {  
        int info;  
        Nodo sig;  
    }  
  
    private Nodo raiz;  
  
    public Pila () {  
        raiz=null;  
    }  
  
    public void insertar(int x) {  
        Nodo nuevo;  
        nuevo = new Nodo();  
        nuevo.info = x;  
        if (raiz==null)  
        {  
            nuevo.sig = null;  
            raiz = nuevo;  
        }  
        else  
        {  
            nuevo.sig = raiz;  
            raiz = nuevo;  
        }  
    }  
}
```

```

    }

    public int extraer () {
        if (raiz!=null) {
            int informacion = raiz.info;
            raiz = raiz.sig;
            return informacion;
        }
        else
        {
            return Integer.MAX_VALUE;
        }
    }

    public void imprimir() {
        Nodo reco=raiz;
        System.out.println("Listado de todos los elementos de la pila.");
        while (reco!=null) {
            System.out.print(reco.info+"-");
            reco=reco.sig;
        }
        System.out.println();
    }

    public static void main(String[] ar) {
        Pila pilal=new Pila();
        pilal.insertar(10);
        pilal.insertar(40);
        pilal.insertar(3);
        pilal.imprimir();
        System.out.println("Extraemos de la pila:"+pilal.extraer());
        pilal.imprimir();
    }
}
}

```

Analicemos las distintas partes de este programa:

```

class Nodo {
    int info;
    Nodo sig;
}

private Nodo raiz;

```

Para declarar un nodo debemos utilizar una clase. En este caso la información del nodo (info) es un entero y siempre el nodo tendrá una referencia de tipo Nodo, que le llamamos sig.

El puntero sig apunta al siguiente nodo o a null en caso que no exista otro nodo. Este puntero es interno a la lista.

También definimos un puntero de tipo Nodo llamado raiz. Este puntero tiene la dirección del primer nodo de la lista. En caso de estar vacía la lista, raiz apunta a null (es decir no tiene dirección)

El puntero raiz es fundamental porque al tener la dirección del primer nodo de la lista nos permite acceder a los demás nodos.

```
public Pila () {  
    raiz=null;  
}
```

En el constructor de la clase hacemos que raiz guarde el valor null. Tengamos en cuenta que si raiz tiene almacenado null la lista está vacía, en caso contrario tiene la dirección del primer nodo de la lista.

```
public void insertar(int x) {  
    Nodo nuevo;  
    nuevo = new Nodo();  
    nuevo.info = x;  
    if (raiz==null)  
    {  
        nuevo.sig = null;  
        raiz = nuevo;  
    }  
    else  
    {  
        nuevo.sig = raiz;  
        raiz = nuevo;  
    }  
}
```

Uno de los métodos más importantes que debemos entender en una pila es el de insertar un elemento en la pila.

Al método llega la información a insertar, en este caso en particular es un valor entero.

La creación de un nodo requiere dos pasos:

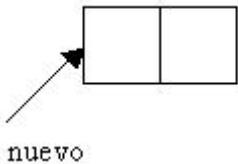
- Definición de un puntero o referencia a un tipo de dato Nodo:

```
Nodo nuevo;
```

- Creación del nodo (creación de un objeto):

```
nuevo = new Nodo();
```

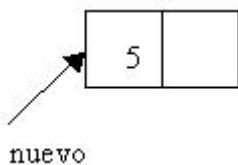
Cuando se ejecuta el operador new se reserva espacio para el nodo. Realmente se crea el nodo cuando se ejecuta el new.



Paso seguido debemos guardar la información del nodo:

```
nuevo.info = x;
```

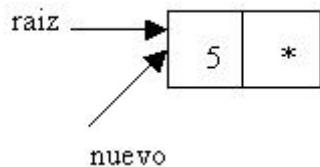
En el campo info almacenamos lo que llega en el parámetro x. Por ejemplo si llega un 5 el nodo queda:



Por último queda enlazar el nodo que acabamos de crear al principio de la lista.

Si la lista está vacía debemos guardar en el campo sig del nodo el valor null para indicar que no hay otro nodo después de este, y hacer que raiz apunte al nodo creado (sabemos si una lista está vacía si raiz almacena un null)

```
if (raiz==null)
{
    nuevo.sig = null;
    raiz = nuevo;
}
```



Gráficamente podemos observar que cuando indicamos `raiz=nuevo`, el puntero `raiz` guarda la dirección del nodo apuntado por `nuevo`.

Tener en cuenta que cuando finaliza la ejecución del método el puntero `nuevo` desaparece, pero no el nodo creado con el operador `new`.

En caso que la lista no esté vacía, el puntero `sig` del nodo que acabamos de crear debe apuntar al que es hasta este momento el primer nodo, es decir al nodo que apunta `raiz` actualmente.

```
else
{
```

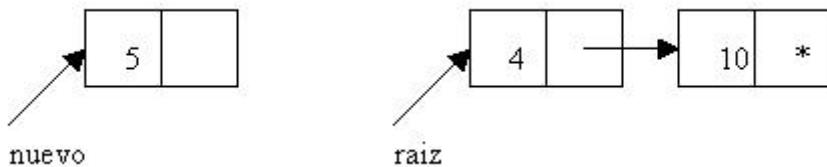
```

        nuevo.sig = raiz;
        raiz = nuevo;
    }
}

```

Como primera actividad cargamos en el puntero sig del nodo apuntado por nuevo la dirección de raiz, y posteriormente raiz apunta al nodo que acabamos de crear, que será ahora el primero de la lista.

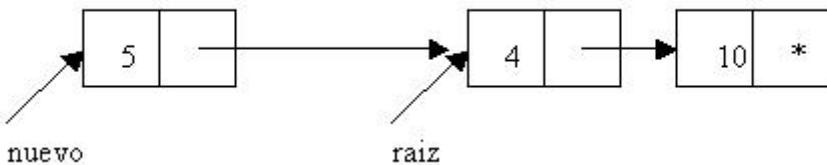
Antes de los enlaces tenemos:



Luego de ejecutar la línea:

```
nuevo.sig = raiz;
```

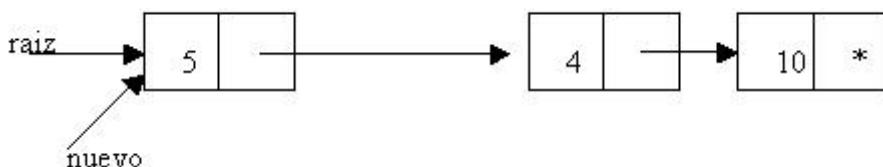
Ahora tenemos:



Por último asignamos a raiz la dirección que almacena el puntero nuevo.

```
raiz = nuevo;
```

La lista queda:



El método extraer:

```

public int extraer ()
{
    if (raiz!=null)
    {
        int informacion = raiz.info;

```

```
        raiz = raiz.sig;
        return informacion;
    }
else
{
    return Integer.MAX_VALUE;
}
}
```

El objetivo del método extraer es retornar la información del primer nodo y además borrarlo de la lista.

Si la lista no está vacía guardamos en una variable local la información del primer nodo:

```
int informacion = raiz.info;
```

Avanzamos raiz al segundo nodo de la lista, ya que borraremos el primero:

```
raiz = raiz.sig;
```

el nodo que previamente estaba apuntado por raiz es eliminado automáticamente por la máquina virtual de Java, al no tener ninguna referencia.

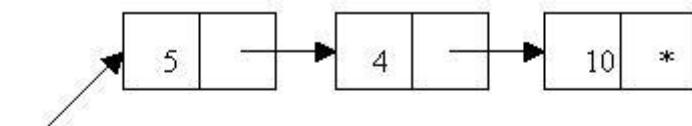
Retornamos la información:

```
return informacion;
```

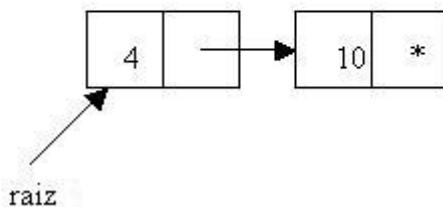
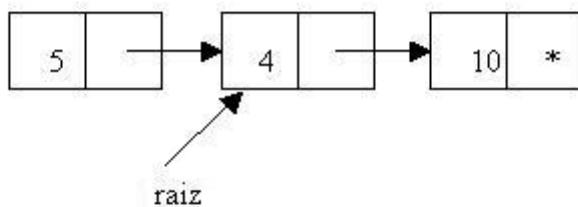
En caso de estar vacía la pila retornamos el número entero máximo y lo tomamos como código de error (es decir nunca debemos guardar el entero mayor en la pila)

```
return Integer.MAX_VALUE;
```

Es muy importante entender gráficamente el manejo de las listas. La interpretación gráfica nos permitirá plantear inicialmente las soluciones para el manejo de listas.



```
raiz=raiz.sig;
```



Por último expliquemos el método para recorrer una lista en forma completa e imprimir la información de cada nodo:

```
public void imprimir() {
    Nodo reco=raiz;
    System.out.println("Listado de todos los elementos de la pila.");
    while (reco!=null) {
        System.out.print(reco.info+"-");
        reco=reco.sig;
    }
    System.out.println();
}
```

Definimos un puntero auxiliar reco y hacemos que apunte al primer nodo de la lista:

```
Nodo reco=raiz;
```

Disponemos una estructura repetitiva que se repetirá mientras reco sea distinto a null. Dentro de la estructura repetitiva hacemos que reco avance al siguiente nodo:

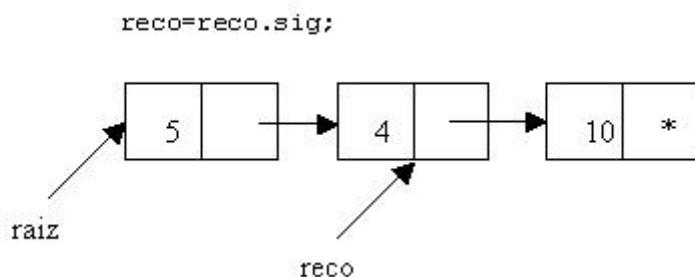
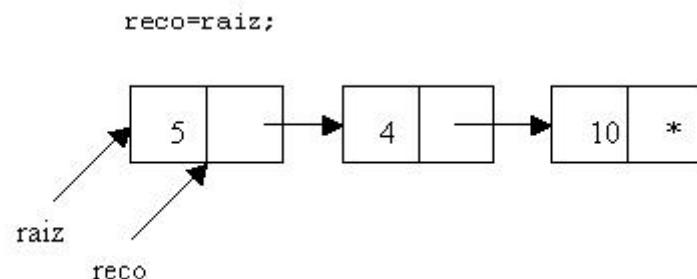
```
while (reco!=null) {
    System.out.print(reco.info+"-");
    reco=reco.sig;
}
```

Es muy importante entender la línea:

```
reco=reco.sig;
```

Estamos diciendo que reco almacena la dirección que tiene el puntero sig del nodo apuntado actualmente por reco.

Gráficamente:



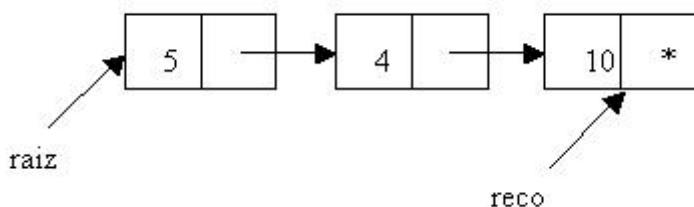
Al analizarse la condición:

```
while (reco!=null) {
```

se valúa en verdadero ya que reco apunta a un nodo y se vuelve a ejecutar la línea:

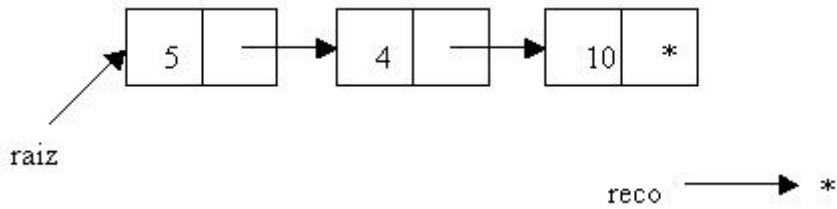
```
reco=reco.sig;
```

Ahora reco apunta al siguiente nodo:



La condición del while nuevamente se valúa en verdadera y avanza el puntero reco al siguiente nodo:

```
reco=reco.sig;
```



Ahora sí reco apunta a null y ha llegado el final de la lista (Recordar que el último nodo de la lista tiene almacenado en el puntero sig el valor null, con el objetivo de saber que es el último nodo)

Para poder probar esta clase recordemos que debemos definir un objeto de la misma y llamar a sus métodos:

```

public static void main(String[] ar) {
    Pila pilal=new Pila();
    pilal.insertar(10);
    pilal.insertar(40);
    pilal.insertar(3);
    pilal.imprimir();
    System.out.println("Extraemos de la pila:"+pilal.extraer());
    pilal.imprimir();
}
  
```

Insertamos 3 enteros, luego imprimimos la pila, extraemos uno de la pila y finalmente imprimimos nuevamente la pila.

## Problema 2:

Agregar a la clase Pila un método que retorne la cantidad de nodos y otro que indique si esta vacía.

### Programa:

```

public class Pila {

    class Nodo {
        int info;
        Nodo sig;
    }

    private Nodo raiz;

    Pila () {
        raiz=null;
    }

    public void insertar(int x) {
        Nodo nuevo;
        nuevo = new Nodo();
  
```

```
nuevo.info = x;
if (raiz==null)
{
    nuevo.sig = null;
    raiz = nuevo;
}
else
{
    nuevo.sig = raiz;
    raiz = nuevo;
}
}

public int extraer ()
{
    if (raiz!=null)
    {
        int informacion = raiz.info;
        raiz = raiz.sig;
        return informacion;
    }
    else
    {
        return Integer.MAX_VALUE;
    }
}

public void imprimir() {
    Nodo reco=raiz;
    System.out.println("Listado de todos los elementos de la pila.");
    while (reco!=null) {
        System.out.print(reco.info+"-");
        reco=reco.sig;
    }
    System.out.println();
}

public boolean vacia() {
    if (raiz==null) {
        return true;
    } else {
        return false;
    }
}

public int cantidad() {
    int cant=0;
    Nodo reco=raiz;
    while (reco!=null) {
        cant++;
        reco=reco.sig;
    }
    return cant;
}

public static void main(String[] ar) {
    Pila pilal=new Pila();
    pilal.insertar(10);
    pilal.insertar(40);
    pilal.insertar(3);
    pilal.imprimir();
```

```

        System.out.println("La cantidad de nodos de la lista
es:"+pila1.cantidad());
        while (pila1.vacia()==false) {
            System.out.println(pila1.extraer());
        }
    }
}

```

Para verificar si la pila esta vacía verificamos el contenido de la variable raiz, si tiene null luego la lista esta vacía y por lo tanto retornamos un true:

```

public boolean vacia() {
    if (raiz==null) {
        return true;
    } else {
        return false;
    }
}

```

El algoritmo para saber la cantidad de nodos es similar al imprimir, pero en lugar de mostrar la información del nodo procedemos a incrementar un contador:

```

public int cantidad() {
    int cant=0;
    Nodo reco=raiz;
    while (reco!=null) {
        cant++;
        reco=reco.sig;
    }
    return cant;
}

```

Para probar esta clase en la main creamos un objeto de la clase Pila insertamos tres enteros:

```

Pila pila1=new Pila();
pila1.insertar(10);
pila1.insertar(40);
pila1.insertar(3);

```

Imprimimos la pila (nos muestra los tres datos):

```

pila1.imprimir();

```

Llamamos al método cantidad (nos retorna un 3):

```

System.out.println("La cantidad de nodos de la lista
es:"+pila1.cantidad());

```

Luego mientras el método vacía nos retorne un false (lista no vacía) procedemos a llamar al método extraer:

```
while (pila1.vacia()==false) {  
    System.out.println(pila1.extraer());  
}
```

## Problemas propuestos

1. Agregar un método a la clase Pila que retorne la información del primer nodo de la Pila sin borrarlo.

[Solución](#)

## Retornar

# 44 - Estructuras dinámicas: Listas tipo Cola

[Listado completo de tutoriales](#)

Una lista se comporta como una cola si las inserciones las hacemos al final y las extracciones las hacemos por el frente de la lista. También se las llama listas FIFO (First In First Out - primero en entrar primero en salir)

Confeccionaremos un programa que permita administrar una lista tipo cola. Desarrollaremos los métodos de insertar, extraer, vacía e imprimir.

## Programa:

```
public class Cola {  
  
    class Nodo {  
        int info;  
        Nodo sig;  
    }  
  
    private Nodo raiz,fondo;  
  
    Cola() {  
        raiz=null;  
        fondo=null;  
    }  
  
    boolean vacia (){  
        if (raiz == null)  
            return true;  
        else  
            return false;  
    }  
  
    void insertar (int info)  
    {  
        Nodo nuevo;  
        nuevo = new Nodo ();  
        nuevo.info = info;  
        nuevo.sig = null;  
        if (vacia ()) {  
            raiz = nuevo;  
            fondo = nuevo;  
        } else {  
            fondo.sig = nuevo;  
            fondo = nuevo;  
        }  
    }  
  
    int extraer ()  
    {  
        if (!vacia ())
```

```

    {
        int informacion = raiz.info;
        if (raiz == fondo){
            raiz = null;
            fondo = null;
        } else {
            raiz = raiz.sig;
        }
        return informacion;
    } else
        return Integer.MAX_VALUE;
    }

    public void imprimir() {
        Nodo reco=raiz;
        System.out.println("Listado de todos los elementos de la cola.");
        while (reco!=null) {
            System.out.print(reco.info+"-");
            reco=reco.sig;
        }
        System.out.println();
    }

    public static void main(String[] ar) {
        Cola colal=new Cola();
        colal.insertar(5);
        colal.insertar(10);
        colal.insertar(50);
        colal.imprimir();
        System.out.println("Extraemos uno de la cola:"+colal.extraer());
        colal.imprimir();
    }
}

```

La declaración del nodo es igual a la clase Pila. Luego definimos dos punteros externos:

```
private Nodo raiz,fondo;
```

raíz apunta al principio de la lista y fondo al final de la lista. Utilizar dos punteros tiene como ventaja que cada vez que tengamos que insertar un nodo al final de la lista no tengamos que recorrerla. Por supuesto que es perfectamente válido implementar una cola con un único puntero externo a la lista.

En el constructor inicializamos a los dos punteros en null (Realmente esto es opcional ya que los atributos de una clase en java se inicializan automáticamente con null):

```
Cola() {
    raiz=null;
    fondo=null;
}
```

El método vacía retorna true si la lista no tiene nodos y false en caso contrario:

```

boolean vacia (){
    if (raiz == null)
        return true;
    else
        return false;
}

```

En la inserción luego de crear el nodo tenemos dos posibilidades: que la cola esté vacía, en cuyo caso los dos punteros externos a la lista deben apuntar al nodo creado, o que haya nodos en la lista.

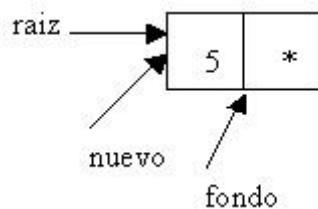
```

Nodo nuevo;
nuevo = new Nodo ();
nuevo.info = info;
nuevo.sig = null;
if (vacia ()) {
    raiz = nuevo;
    fondo = nuevo;
} else {
    fondo.sig = nuevo;
    fondo = nuevo;
}

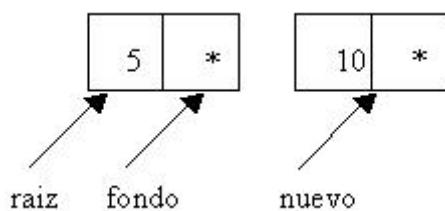
```

Recordemos que definimos un puntero llamado nuevo, luego creamos el nodo con el operador new y cargamos los dos campos, el de información con lo que llega en el parámetro y el puntero con null ya que se insertará al final de la lista, es decir no hay otro después de este.

Si la lista está vacía:



En caso de no estar vacía:

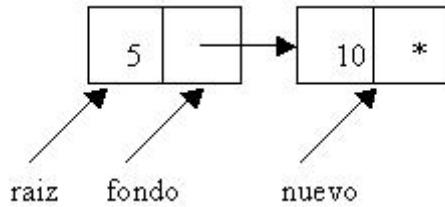


Debemos enlazar el puntero sig del último nodo con el nodo recién creado:

```

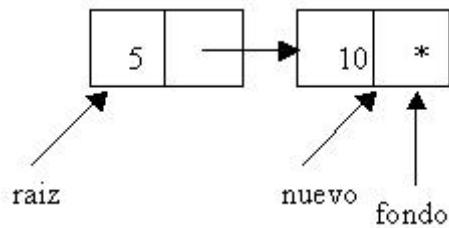
fondo.sig = nuevo;

```



Y por último el puntero externo fondo debe apuntar al nodo apuntado por nuevo:

```
fondo = nuevo;
```



Con esto ya tenemos correctamente enlazados los nodos en la lista tipo cola. Recordar que el puntero nuevo desaparece cuando se sale del método insertar, pero el nodo creado no se pierde porque queda enlazado en la lista.

El funcionamiento del método extraer es similar al de la pila:

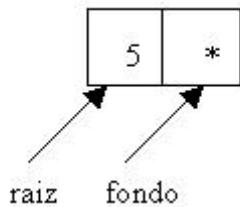
```
int extraer ()
{
    if (!vacia ())
    {
        int informacion = raiz.info;
        if (raiz == fondo){
            raiz = null;
            fondo = null;
        } else {
            raiz = raiz.sig;
        }
        return informacion;
    } else
        return Integer.MAX_VALUE;
}
```

Si la lista no está vacía guardamos en una variable local la información del primer nodo:

```
int informacion = raiz.info;
```

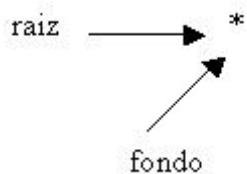
Para saber si hay un solo nodo verificamos si los dos punteros raiz y fondo apuntan a la misma dirección de memoria:

```
if (raiz == fondo){
```

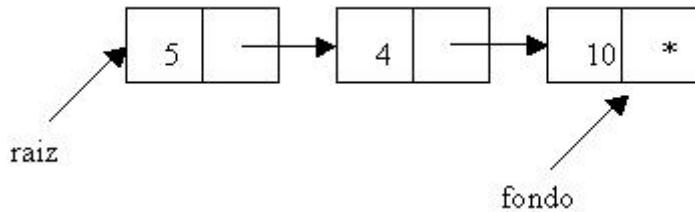


Luego hacemos:

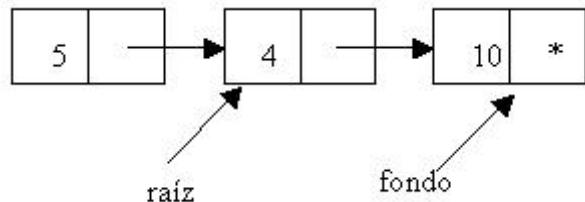
```
raiz = null;  
fondo = null;
```



En caso de haber 2 o más nodos debemos avanzar el puntero raiz al siguiente nodo:



```
raiz = raiz.sig;
```



Ya tenemos la lista correctamente enlazada (raiz apunta al primer nodo y fondo continúa apuntando al último nodo)

**Retornar**

# 43 - Estructuras dinámicas: Listas tipo Pila - Problema de aplicación

[Listado completo de tutoriales](#)

Hasta ahora hemos visto como desarrollar los algoritmos para administrar una lista tipo Pila, hemos visto que hay bastante complejidad en el manejo de punteros pero todo esto acarrea ventajas en la solución de problemas que requieren una estructura de tipo Pila.

## Planteo del problema:

Este práctico tiene por objetivo mostrar la importancia de las pilas en las Ciencias de la Computación y más precisamente en la programación de software de bajo nivel.

Todo compilador o intérprete de un lenguaje tiene un módulo dedicado a analizar si una expresión está correctamente codificada, es decir que los paréntesis estén abiertos y cerrados en un orden lógico y bien balanceados.

Se debe desarrollar una clase que tenga las siguientes responsabilidades (clase Formula):

- Ingresar una fórmula que contenga paréntesis, corchetes y llaves.
- Validar que los ( ) [] y {} estén correctamente balanceados.

Para la solución de este problema la clase formula tendrá un atributo de la clase Pila.

Veamos como nos puede ayudar el empleo de una pila para solucionar este problema. Primero cargaremos la fórmula en un JTextField.

Ejemplo de fórmula:  $(2+[3-12]*\{8/3\})$

El algoritmo de validación es el siguiente:

Analizamos carácter a carácter la presencia de los paréntesis, corchetes y llaves.

Si vienen símbolos de apertura los almacenamos en la pila.

Si vienen símbolos de cerrado extraemos de la pila y verificamos si está el mismo símbolo pero de apertura: en caso negativo podemos inferir que la fórmula no está correctamente balanceada.

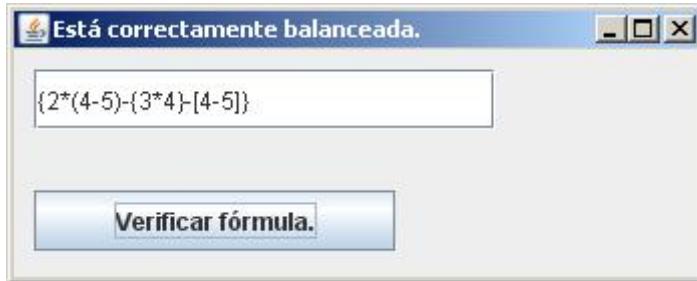
Si al finalizar el análisis del último carácter de la fórmula la pila está vacía podemos concluir que está correctamente balanceada.

Ejemplos de fórmulas no balanceadas:

```

}(2+[3-12]*{8/3})
Incorrecta: llega una } de cerrado y la pila está vacía.
{[2+4]}
Incorrecta: llega una llave } y en el tope de la pila hay un corchete [ .
{[2+4]}
Incorrecta: al finalizar el análisis del último carácter en la pila queda
pendiente una llave {.

```



## Programa:

```

public class Pila {

    class Nodo {
        char simbolo;
        Nodo sig;
    }

    private Nodo raiz;

    Pila () {
        raiz=null;
    }

    public void insertar(char x) {
        Nodo nuevo;
        nuevo = new Nodo();
        nuevo.simbolo = x;
        if (raiz==null)
        {
            nuevo.sig = null;
            raiz = nuevo;
        }
        else
        {
            nuevo.sig = raiz;
            raiz = nuevo;
        }
    }

    public char extraer ()
    {
        if (raiz!=null)
        {
            char informacion = raiz.simbolo;
            raiz = raiz.sig;
            return informacion;
        }
        else
    }
}

```

```

        {
            return Character.MAX_VALUE;
        }
    }

    public boolean vacia() {
        if (raiz==null) {
            return true;
        } else {
            return false;
        }
    }
}

import javax.swing.*;
import java.awt.event.*;
public class Formula extends JFrame implements ActionListener {
    private JTextField tf1;
    private JButton boton1;
    public Formula() {
        setLayout(null);
        tf1=new JTextField("{2*(4-5)-{3*4}-[4-5]}");
        tf1.setBounds(10,10,230,30);
        add(tf1);
        boton1=new JButton("Verificar fórmula.");
        boton1.setBounds(10,70,180,30);
        add(boton1);
        boton1.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==boton1) {
            if (balanceada()) {
                setTitle("Está correctamente balanceada.");
            } else {
                setTitle("No está correctamente balanceada.");
            }
        }
    }

    public boolean balanceada() {
        Pila pilal;
        pilal = new Pila ();
        String cadena=tf1.getText();
        for (int f = 0 ; f < cadena.length() ; f++)
        {
            if (cadena.charAt(f) == '(' || cadena.charAt(f) == '[' || cadena.charAt(f) == '{') {
                pilal.insertar(cadena.charAt(f));
            } else {
                if (cadena.charAt(f)==')') {
                    if (pilal.extraer()!='(') {
                        return false;
                    }
                } else {
                    if (cadena.charAt(f)=='[') {
                        if (pilal.extraer()!='[') {

```

```

        return false;
    }
} else {
    if (cadena.charAt(f)=='}') {
        if (pila1.extraer()!='{') {
            return false;
        }
    }
}
if (pila1.vacia()) {
    return true;
} else {
    return false;
}
}

public static void main(String[] ar) {
    Formula formula1=new Formula();
    formula1.setBounds(0,0,350,140);
    formula1.setVisible(true);
}
}

```

Primero declaramos y definimos la clase Pila. Almacenamos en cada nodo un carácter y llamamos al campo de información símbolo.

No es necesario implementar los métodos imprimir, cantidad, etc. Porque no se requieren para este problema.

La clase Formula tiene como atributos:

```

private JTextField tf1;
private JButton boton1;

```

En el constructor creamos los dos objetos y los ubicamos:

```

setLayout(null);
tf1=new JTextField("{2*(4-5)-{3*4}-[4-5]}");
tf1.setBounds(10,10,230,30);
add(tf1);
boton1=new JButton("Verificar fórmula.");
boton1.setBounds(10,70,180,30);
add(boton1);
boton1.addActionListener(this);

```

En el método actionPerformed llamamos al método balanceada que debe retornar si la fórmula están correctos los parentesis, corchetes y llaves:

```

if (e.getSource() == boton1) {
    if (balanceada()) {
        setTitle("Está correctamente balanceada.");
    } else {
        setTitle("No está correctamente balanceada.");
    }
}

```

El método más importante es el balanceada.

En este analizamos la fórmula para verificar si está correctamente balanceada.

En este método es donde está gran parte del algoritmo de este problema. Retorna true en caso de ser correcta y false en caso contrario.

Definimos una pila y extraemos el contenido del JTextField:

```

Pila pilal;
pilal = new Pila ();
String cadena = tf1.getText();

```

El for se repite tantas veces como caracteres tenga el JTextField.

Se deben procesar sólo los símbolos ( [ { y } ].

Si el símbolo es un ( [ { de apertura procedemos a cargarlo en la pila:

```

if (cadena.charAt(f) == '(' || cadena.charAt(f) == '[' ||
cadena.charAt(f) == '{') {
    pilal.insertar(cadena.charAt(f));
}

```

En caso de ser un ) cerrado debemos extraer un carácter de la pila y verificar si no coincide con el paréntesis de apertura ( la fórmula está incorrecta:

```

if (cadena.charAt(f) == ')') {
    if (pilal.extraer() != '(') {
        return false;
    }
}

```

El mismo proceso es para los símbolos ] }.

Al finalizar el análisis de toda la cadena si la pila está vacía podemos afirmar que la fórmula está correctamente balanceada, en caso contrario quiere decir que faltan símbolos de cerrado y es incorrecta:

```

if (pilal.vacia()) {
    return true;
} else {
    return false;
}

```

Es importante entender que la clase Formula utiliza un objeto de la clase Pila para resolver el algoritmo de verificar el balanceo de la fórmula, pero no accede directamente a los nodos de la lista.

[\*\*Retornar\*\*](#)

# 45 - Estructuras dinámicas: Listas tipo Cola - Problemas de aplicación

[Listado completo de tutoriales](#)

## Planteo del problema:

Este práctico tiene por objetivo mostrar la importancia de las colas en las Ciencias de la Computación y más precisamente en las simulaciones.

Las simulaciones permiten analizar situaciones de la realidad sin la necesidad de ejecutarlas realmente. Tiene el beneficio que su costo es muy inferior a hacer pruebas en la realidad.

Desarrollar un programa para la simulación de un cajero automático.

Se cuenta con la siguiente información:

- Llegan clientes a la puerta del cajero cada 2 ó 3 minutos.
- Cada cliente tarda entre 2 y 4 minutos para ser atendido.

Obtener la siguiente información:

- 1 - Cantidad de clientes que se atienden en 10 horas.
- 2 - Cantidad de clientes que hay en cola después de 10 horas.
- 3 - Hora de llegada del primer cliente que no es atendido luego de 10 horas (es decir la persona que está primera en la cola cuando se cumplen 10 horas)

## Programa:

```
public class Cola {  
  
    class Nodo {  
        int info;  
        Nodo sig;  
    }  
  
    Nodo raiz,fondo;  
  
    Cola() {  
        raiz=null;  
        fondo=null;  
    }  
  
    boolean vacia (){  
        if (raiz == null)  
            return true;  
        else  
            return false;  
    }  
  
    void insertar (int info)  
    {  
        Nodo n = new Nodo();  
        n.info = info;  
        if (raiz == null)  
            raiz = n;  
        else  
            fondo.sig = n;  
        fondo = n;  
    }  
  
    void imprimir () {  
        Nodo aux = raiz;  
        while (aux != null) {  
            System.out.print(aux.info + " ");  
            aux = aux.sig;  
        }  
    }  
}
```

```

    }
    Nodo nuevo;
    nuevo = new Nodo ();
    nuevo.info = info;
    nuevo.sig = null;
    if (vacia ()) {
        raiz = nuevo;
        fondo = nuevo;
    } else {
        fondo.sig = nuevo;
        fondo = nuevo;
    }
}

int extraer ()
{
    if (!vacia ())
    {
        int informacion = raiz.info;
        if (raiz == fondo){
            raiz = null;
            fondo = null;
        } else {
            raiz = raiz.sig;
        }
        return informacion;
    } else
        return Integer.MAX_VALUE;
}

public void imprimir() {
    Nodo reco=raiz;
    System.out.println("Listado de todos los elementos de la cola.");
    while (reco!=null) {
        System.out.print(reco.info+"-");
        reco=reco.sig;
    }
    System.out.println();
}

public int cantidad() {
    int cant=0;
    Nodo reco=raiz;
    while (reco!=null) {
        cant++;
        reco=reco.sig;
    }
    return cant;
}
}
}

```

```

import javax.swing.*;
import java.awt.event.*;
public class Cajero extends JFrame implements ActionListener{
    private JLabel l1,l2,l3;
    private JButton boton1;
    public Cajero() {
        ...
    }
}

```

```

setLayout(null);
boton1=new JButton("Activar Simulación");
boton1.setBounds(10,10,180,30);
add(boton1);
boton1.addActionListener(this);
l1=new JLabel("Atendidos:");
l1.setBounds(10,50,300,30);
add(l1);
l2=new JLabel("En cola:");
l2.setBounds(10,90,300,30);
add(l2);
l3=new JLabel("Minuto de llegada:");
l3.setBounds(10,130,400,30);
add(l3);
}

public void actionPerformed(ActionEvent e) {
    if (e.getSource()==boton1) {
        simulacion();
    }
}

public void simulacion () {
    int estado = 0;
    int llegada = 2 + (int) (Math.random () * 2);
    int salida = -1;
    int cantAtendidas = 0;
    Cola cola = new Cola ();
    for (int minuto = 0 ; minuto < 600 ; minuto++) {
        if (llegada == minuto)
        {
            if (estado==0) {
                estado=1;
                salida=minuto+2+(int)(Math.random()*3);
            } else {
                cola.insertar(minuto);
            }
            llegada=minuto+2+(int)(Math.random()*2);
        }
        if (salida == minuto)
        {
            estado=0;
            cantAtendidas++;
            if (!cola.vacia()) {
                cola.extraer();
                estado=1;
                salida=minuto+2+(int)(Math.random()*3);
            }
        }
    }
    l1.setText("Atendidos:"+String.valueOf(cantAtendidas));
    l2.setText("En cola"+String.valueOf(cola.cantidad()));
    l3.setText("Minuto llegada:"+String.valueOf(cola.extraer()));
}

public static void main(String[] ar) {
    Cajero cajero1=new Cajero();
    cajero1.setBounds(0,0,340,250);
    cajero1.setVisible(true);
}
}

```

La clase Cola colabora con la clase Cajero. En la clase Cola debemos definir como mínimo los métodos de insertar, extraer, vacía y cantidad.

La clase Cajero define tres objetos de la clase JLabel para mostrar los resultados de la simulación.

El método más importante es el de simulacion, veamos las distintas partes de dicho método:

```
int estado = 0;
int llegada = 2 + (int) (Math.random () * 2);
int salida = -1;
int cantAtendidas = 0;
Cola cola = new Cola ();
```

La variable estado almacena un cero si el cajero está libre y un uno cuando está ocupado.

La variable llegada almacena en que minuto llegará el próximo cliente (debemos generar un valor entre 2 y 3)

La variable salida almacenará en que minuto terminará el cliente de ser atendido (como al principio el cajero está vacío inicializamos esta variable con -1).

Llevamos un contador para saber la cantidad de personas atendidas (cantAtendidas)

Luego definimos un objeto de la clase Cola para poder almacenar las personas que llegan al cajero y se lo encuentran ocupado.

Disponemos un for que se repita 600 veces (600 minutos o lo que es lo mismo 10 horas)

```
for (int minuto = 0 ; minuto < 600 ; minuto++) {
```

Dentro del for hay dos if fundamentales que verifican que sucede cuando llega una persona o cuando una persona se retira:

```
if (llegada == minuto)
{
    .....
}
if (salida == minuto)
{
    .....
}
```

Cuando llega una persona al cajero primero verificamos si el cajero está desocupado:

```

if (llegada == minuto)
{
    if (estado==0) {

```

Si está desocupado lo ocupamos cambiando el valor de la variable estado y generando en que minuto esta persona dejará el cajero (un valor aleatorio entre 2 y 4 minutos):

```

        estado=1;
        salida=minuto+2+(int)(Math.random()*3);
    }
}
```

Si el cajero está ocupado procedemos a cargar dicha persona en la cola (insertamos el minuto que llega):

```

    } else {
        cola.insertar(minuto);
    }
}
```

Luego generamos el próximo minuto que llegará otra persona:

```

llegada=minuto+2+(int)(Math.random()*2);

```

El otro if importante es ver que sucede cuando sale la persona del cajero:

```
if (salida == minuto) {
```

Si sale una persona del cajero cambiamos el valor de la variable estado, incrementamos en uno el contador cantAtendidos y si la cola no está vacía extraemos una persona, cambiamos a uno la variable estado y generamos en que minuto dejará esta persona el cajero:

```

        estado=0;
        cantAtendidas++;
        if (!cola.vacia()) {
            cola.extraer();
            estado=1;
            salida=minuto+2+(int)(Math.random()*3);
        }
    }
}
```

Fuera del for actualizamos las tres JLabel:

```

11.setText("Atendidos:"+String.valueOf(cantAtendidas));
12.setText("En cola"+String.valueOf(cola.cantidad()));
13.setText("Minuto llegada:"+String.valueOf(cola.extraer()));

```

## Problemas propuestos

1. Un supermercado tiene tres cajas para la atención de los clientes.  
Las cajeras tardan entre 7 y 11 minutos para la atención de cada cliente.  
Los clientes llegan a la zona de cajas cada 2 ó 3 minutos. (Cuando el cliente llega, si todas las cajas tienen 6 personas, el cliente se marcha del supermercado)  
Cuando el cliente llega a la zona de cajas elige la caja con una cola menor.  
Realizar una simulación durante 8 horas y obtener la siguiente información:  
a - Cantidad de clientes atendidos por cada caja.  
b - Cantidad de clientes que se marcharon sin hacer compras.  
c - Tiempo promedio en cola.

[Solución](#)

[\*\*Retornar\*\*](#)

# 46 - Estructuras dinámicas: Listas genéricas

[Listado completo de tutoriales](#)

Continuando con el tema de listas trabajaremos con las listas genéricas. Una lista se comporta como genérica cuando las inserciones y extracciones se realizan en cualquier parte de la lista.

Codificaremos una serie de métodos para administrar listas genéricas.

## Métodos a desarrollar:

Inserta un nodo en la posición (pos) y con la información que hay en el parámetro x.

```
void insertar(int pos, int x)
```

Extrae la información del nodo de la posición indicada (pos). Se debe eliminar el nodo.

```
int extraer(int pos)
```

Borra el nodo de la posición (pos).

```
void borrar(int pos)
```

Intercambia las informaciones de los nodos de las posiciones pos1 y pos2.

```
void intercambiar(int pos1,int pos2)
```

Retorna el valor del nodo con mayor información.

```
int mayor()
```

Retorna la posición del nodo con mayor información.

```
int posMayor()
```

Retorna la cantidad de nodos de la lista.

```
int cantidad()
```

Debe retornar true si la lista está ordenada de menor a mayor, false en caso contrario.

```
boolean ordenada()
```

Debe retornar true si existe la información que llega en el parámetro, false en caso contrario.

```
boolean existe(int info)
```

El método vacía debe retornar true si está vacía y false si no lo está.

```
boolean vacia()
```

## Programa:

```
public class ListaGenerica {  
  
    class Nodo {  
        int info;  
        Nodo sig;  
    }  
  
    private Nodo raiz;  
  
    public ListaGenerica () {  
        raiz=null;  
    }  
  
    void insertar (int pos, int x)  
    {  
        if (pos <= cantidad () + 1) {  
            Nodo nuevo = new Nodo ();  
            nuevo.info = x;  
            if (pos == 1){  
                nuevo.sig = raiz;  
                raiz = nuevo;  
            } else  
                if (pos == cantidad () + 1) {  
                    Nodo reco = raiz;  
                    while (reco.sig != null) {  
                        reco = reco.sig;  
                    }  
                    reco.sig = nuevo;  
                    nuevo.sig = null;  
                } else {  
                    Nodo reco = raiz;  
                    for (int f = 1 ; f <= pos - 2 ; f++)  
                        reco = reco.sig;  
                    Nodo siguiente = reco.sig;  
                    reco.sig = nuevo;  
                    nuevo.sig = siguiente;  
                }  
        }  
    }
```

```

}

public int extraer (int pos) {
    if (pos <= cantidad ()) {
        int informacion;
        if (pos == 1) {
            informacion = raiz.info;
            raiz = raiz.sig;
        } else {
            Nodo reco;
            reco = raiz;
            for (int f = 1 ; f <= pos - 2 ; f++)
                reco = reco.sig;
            Nodo prox = reco.sig;
            reco.sig = prox.sig;
            informacion = prox.info;
        }
        return informacion;
    }
    else
        return Integer.MAX_VALUE;
}

public void borrar (int pos)
{
    if (pos <= cantidad ()) {
        if (pos == 1) {
            raiz = raiz.sig;
        } else {
            Nodo reco;
            reco = raiz;
            for (int f = 1 ; f <= pos - 2 ; f++)
                reco = reco.sig;
            Nodo prox = reco.sig;
            reco.sig = prox.sig;
        }
    }
}

public void intercambiar (int pos1, int pos2) {
    if (pos1 <= cantidad () && pos2 <= cantidad ()) {
        Nodo recol = raiz;
        for (int f = 1 ; f < pos1 ; f++)
            recol = recol.sig;
        Nodo reco2 = raiz;
        for (int f = 1 ; f < pos2 ; f++)
            reco2 = reco2.sig;
        int aux = recol.info;
        recol.info = reco2.info;
        reco2.info = aux;
    }
}

public int mayor () {
    if (!vacia ()) {
        int may = raiz.info;
        Nodo reco = raiz.sig;
        while (reco != null) {
            if (reco.info > may)
                may = reco.info;
            reco = reco.sig;
        }
    }
}

```

```

        }
        return may;
    }
    else
        return Integer.MAX_VALUE;
}

public int posMayor() {
    if (!vacia ())
        int may = raiz.info;
        int x=1;
        int pos=x;
        Nodo reco = raiz.sig;
        while (reco != null){
            if (reco.info > may) {
                may = reco.info;
                pos=x;
            }
            reco = reco.sig;
            x++;
        }
        return pos;
    }
    else
        return Integer.MAX_VALUE;
}

public int cantidad ()
{
    int cant = 0;
    Nodo reco = raiz;
    while (reco != null) {
        reco = reco.sig;
        cant++;
    }
    return cant;
}

public boolean ordenada() {
    if (cantidad()>1) {
        Nodo reco1=raiz;
        Nodo reco2=raiz.sig;
        while (reco2!=null) {
            if (reco2.info<reco1.info) {
                return false;
            }
            reco2=reco2.sig;
            reco1=reco1.sig;
        }
    }
    return true;
}

public boolean existe(int x) {
    Nodo reco=raiz;
    while (reco!=null) {
        if (reco.info==x)
            return true;
        reco=reco.sig;
    }
    return false;
}

```

```

}

public boolean vacia ()
{
    if (raiz == null)
        return true;
    else
        return false;
}

public void imprimir ()
{
    Nodo reco = raiz;
    while (reco != null) {
        System.out.print (reco.info + "-");
        reco = reco.sig;
    }
    System.out.println();
}

public static void main(String[] ar) {
    ListaGenerica lg=new ListaGenerica();
    lg.insertar (1, 10);
    lg.insertar (2, 20);
    lg.insertar (3, 30);
    lg.insertar (2, 15);
    lg.insertar (1, 115);
    lg.imprimir ();
    System.out.println ("Luego de Borrar el primero");
    lg.borrar (1);
    lg.imprimir ();
    System.out.println ("Luego de Extraer el segundo");
    lg.extraer (2);
    lg.imprimir ();
    System.out.println ("Luego de Intercambiar el primero con el
tercero");
    lg.intercambiar (1, 3);
    lg.imprimir ();
    if (lg.existe(20))
        System.out.println("Se encuentra el 20 en la lista");
    else
        System.out.println("No se encuentra el 20 en la lista");
    System.out.println("La posición del mayor es:"+lg.posMayor());
    if (lg.ordenada())
        System.out.println("La lista está ordenada de menor a mayor");
    else
        System.out.println("La lista no está ordenada de menor a
mayor");
}
}

```

Para insertar en una determinada posición dentro de la lista:

```
void insertar (int pos, int x)
```

Primero con un if verificamos que exista esa posición en la lista (por ejemplo si la lista tiene 4 nodos podemos insertar hasta la posición 5, es decir uno más allá del último):

```
if (pos <= cantidad () + 1) {
```

Si ingresa al if ya podemos crear el nodo:

```
Nodo nuevo = new Nodo ();
nuevo.info = x;
```

Ahora debemos analizar si la inserción es al principio de la lista, al final o en medio ya que los enlaces varían según donde se lo inserta.

Para saber si se inserta al principio de la lista preguntamos si en pos llega un 1:

```
if (pos == 1){
```

Si llega un 1 luego enlazamos el puntero sig del nodo que creamos con la dirección del primer nodo de la lista (raiz apunta siempre al primer nodo de la lista) y luego desplazamos raiz al nodo que acabamos de crear:

```
nuevo.sig = raiz;
raiz = nuevo;
```

Si no se inserta al principio de la lista preguntamos si se inserta al final:

```
if (pos == cantidad () + 1) {
```

En caso de insertarse al final recorremos la lista hasta el último nodo:

```
Nodo reco = raiz;
while (reco.sig != null) {
    reco = reco.sig;
}
```

y enlazamos el puntero sig del último nodo de la lista con la dirección del nodo que acabamos de crear (disponemos en sig del nodo creado el valor null ya que no hay otro nodo más adelante)

```
reco.sig = nuevo;
nuevo.sig = null;
```

Si no se inserta al principio o al final significa que tenemos que insertar en medio de la lista.

Disponemos un for donde avanzamos un puntero auxiliar y nos detenemos una posición antes a donde tenemos que insertarlo:

```
for (int f = 1 ; f <= pos - 2 ; f++)
    reco = reco.sig;
```

Disponemos otro puntero auxiliar que apunte al nodo próximo a donde está apuntando reco. Ahora enlazamos el puntero sig del nodo apuntado por reco con la dirección del nodo creado y el puntero sig del nodo creado con la dirección del nodo siguiente:

```
Nodo siguiente = reco.sig;
reco.sig = nuevo;
nuevo.sig = siguiente;
```

El método extraer recibe como parámetro la posición del nodo a extraer:

```
public int extraer (int pos) {
```

Primero verificamos que la posición exista en la lista:

```
if (pos <= cantidad ()) {
```

En caso que exista verificamos si el nodo a extraer es el primero de la lista (este análisis debe hacerse ya que si es el primero de la lista se modifica el puntero raiz):

```
if (pos == 1) {
```

Si es el primero guardamos en una variable auxiliar la información del nodo y avanzamos el puntero raiz:

```
informacion = raiz.info;
raiz = raiz.sig;
```

Si el nodo a extraer no está al principio de la lista avanzamos con una estructura repetitiva hasta el nodo anterior a extraer:

```
for (int f = 1 ; f <= pos - 2 ; f++)
    reco = reco.sig;
```

Luego definimos otro puntero auxiliar y lo disponemos en el siguiente nodo a donde está apuntando reco:

```
Nodo prox = reco.sig;
```

Ahora enlazamos el puntero sig del nodo apuntado por reco al nodo siguiente del nodo apuntado por prox (es decir el nodo apuntado por prox queda fuera de la lista):

```
reco.sig = prox.sig;
```

El método borrar es muy similar al método extraer con la diferencia de que no retorna valor:

```
public void borrar (int pos)
{
    if (pos <= cantidad ())
        if (pos == 1) {
            raiz = raiz.sig;
        } else {
            Nodo reco;
            reco = raiz;
            for (int f = 1 ; f <= pos - 2 ; f++)
                reco = reco.sig;
            Nodo prox = reco.sig;
            reco.sig = prox.sig;
        }
    }
}
```

El método intercambiar recibe dos enteros que representan las posiciones de los nodos que queremos intercambiar sus informaciones:

```
public void intercambiar (int pos1, int pos2) {
```

Mediante un if verificamos que las dos posiciones existan en la lista:

```
if (pos1 <= cantidad () && pos2 <= cantidad ()) {
```

Definimos un puntero auxiliar llamado reco1, lo inicializamos con la dirección del primer nodo y mediante un for avanzamos hasta la posición almacenada en pos1:

```
Nodo reco1 = raiz;
for (int f = 1 ; f < pos1 ; f++)
    reco1 = reco1.sig;
```

De forma similar con un segundo puntero auxiliar avanzamos hasta la posición indicada por pos2:

```
Nodo reco2 = raiz;
for (int f = 1 ; f < pos2 ; f++)
    reco2 = reco2.sig;
```

Por último intercambiamos las informaciones que almacenan cada nodo:

```
int aux = reco1.info;
reco1.info = reco2.info;
reco2.info = aux;
```

El método que retorna el mayor de la lista:

```
public int mayor () {
```

Verificamos que la lista no esté vacía:

```
if (!vacia ()) {
```

Suponemos que el mayor es el primero de la lista e inicializamos un puntero auxiliar con la dirección del segundo nodo de la lista:

```
int may = raiz.info;
Nodo reco = raiz.sig;
```

Mediante una estructura repetitiva recorremos toda la lista:

```
while (reco != null) {
```

Cada vez que encontramos un nodo con información mayor que la variable may la actualizamos con este nuevo valor y avanzamos el puntero reco para visitar el siguiente nodo:

```
if (reco.info > may)
    may = reco.info;
reco = reco.sig;
```

Fuera de la estructura repetitiva retornamos el mayor:

```
return may;
```

El método que retorna la posición del mayor es similar al anterior con la salvedad que debemos almacenar en otro auxiliar la posición donde se almacena el mayor:

```

public int posMayor() {
    if (!vacia ()) {
        int may = raiz.info;
        int x=1;
        int pos=x;
        Nodo reco = raiz.sig;
        while (reco != null){
            if (reco.info > may) {
                may = reco.info;
                pos=x;
            }
            reco = reco.sig;
            x++;
        }
        return pos;
    }
    else
        return Integer.MAX_VALUE;
}

```

El método que debe retornar si está ordenada la lista de menor a mayor es:

```
public boolean ordenada() {
```

Lo primero que verificamos si la lista tiene más de un nodo significa que debemos controlarla:

```
if (cantidad()>1) {
```

Disponemos dos punteros auxiliares con las direcciones del primer y segundo nodo de la lista:

```

Nodo reco1=raiz;
Nodo reco2=raiz.sig;
```

Mediante un while mientras no se finaliza la lista:

```
while (reco2!=null) {
```

controlamos si la información del segundo nodo es menor al nodo anterior significa que la lista no está ordenada y podemos parar el análisis retornando un false

```

if (reco2.info<reco1.info) {
    return false;
```

Dentro del while avanzamos los dos punteros a sus nodos siguientes respectivamente.

```
reco2=reco2.sig;
reco1=reco1.sig;
```

Fuera del while retornamos true indicando que la lista está ordenada de menor a mayor

```
return true;
```

El método existe:

```
public boolean existe(int x) {
```

Mediante un while recorremos la la lista:

```
Nodo reco=raiz;
while (reco!=null) {
```

y en cada nodo que visitamos controlamos si el parámetro x es igual a la información del nodo, en caso afirmativo salimos del método retornando true:

```
if (reco.info==x)
    return true;
reco=reco.sig;
```

Fuera del while retornamos false indicando que ningún nodo coincide con el parámetro x:

```
return false;
```

## Problemas propuestos

1. Plantear una clase para administrar una lista genérica implementando los siguientes métodos:
  - a) Insertar un nodo al principio de la lista.
  - b) Insertar un nodo al final de la lista.
  - c) Insertar un nodo en la segunda posición. Si la lista está vacía no se inserta el nodo.
  - d) Insertar un nodo en la ante última posición.
  - e) Borrar el primer nodo.

- f) Borrar el segundo nodo.
- g) Borrar el último nodo.
- h) Borrar el nodo con información mayor.

[Solución](#)

[\*\*Retornar\*\*](#)

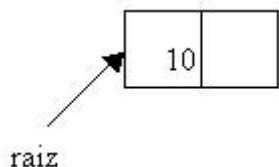
# 47 - Estructuras dinámicas: Listas genéricas ordenadas

[Listado completo de tutoriales](#)

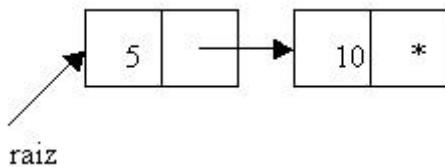
Una lista genérica es ordenada si cuando insertamos información en la lista queda ordenada respecto al campo info (sea de menor a mayor o a la inversa)

Ejemplo:

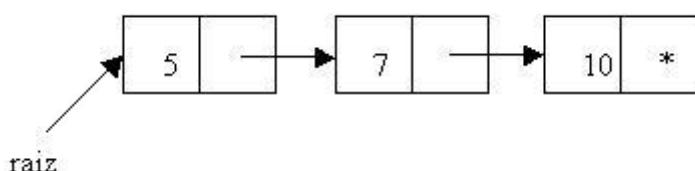
```
listaOrdenada.insertar(10)
```



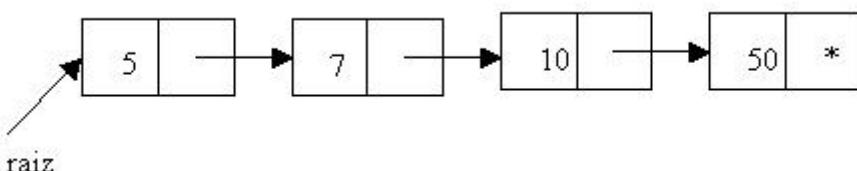
```
listaOrdenada.insertar(5)
```



```
listaOrdenada.insertar(7)
```



```
listaOrdenada.insertar(50)
```



Podemos observar que si recorremos la lista podemos acceder a la información de menor a mayor.

No se requiere un método para ordenar la lista, sino que siempre permanece ordenada, ya que se inserta ordenada.

### Programa:

```
public class ListaOrdenada {  
  
    class Nodo {  
        int info;  
        Nodo sig;  
    }  
  
    private Nodo raiz;  
  
    public ListaOrdenada() {  
        raiz=null;  
    }  
  
    void insertar(int x)  
    {  
        Nodo nuevo = new Nodo ();  
        nuevo.info = x;  
        if (raiz==null) {  
            raiz=nuevo;  
        } else {  
            if (x<raiz.info) {  
                nuevo.sig=raiz;  
                raiz=nuevo;  
            } else {  
                Nodo reco=raiz;  
                Nodo atras=raiz;  
                while (x>=reco.info && reco.sig!=null)  
                    atras=reco;  
                    reco=reco.sig;  
                }  
                if (x>=reco.info) {  
                    reco.sig=nuevo;  
                } else {  
                    nuevo.sig=reco;  
                    atras.sig=nuevo;  
                }  
            }  
        }  
    }  
}
```

```

        }
    }

}

public void imprimir () {
    Nodo reco = raiz;
    while (reco != null) {
        System.out.print (reco.info + "-");
        reco = reco.sig;
    }
    System.out.println();
}

public static void main(String[] ar) {
    ListaOrdenada lo=new ListaOrdenada();
    lo.insertar(10);
    lo.insertar(5);
    lo.insertar(7);
    lo.insertar(50);
    lo.imprimir();
}
}

```

El método insertar lo resolvemos de la siguiente forma:

Creamos primeramente el nodo, ya que siempre se insertará la información en la lista:

```

Nodo nuevo = new Nodo ();
nuevo.info = x;

```

Se puede presentar las siguientes situaciones, si está vacía, lo insertamos inmediatamente:

```

if (raiz==null) {
    raiz=nuevo;
} else {

```

Si no está vacía la lista, verificamos si lo debemos insertar en la primera posición de la lista (analizamos si la información a insertar es menor a lo apuntado por raiz en el campo info):

```
if (x<raiz.info) {  
    nuevo.sig=raiz;  
    raiz=nuevo;  
} else {
```

Sino analizamos si lo debemos insertar en medio o al final de la lista.

Mientras la información a insertar sea mayor o igual a la información del nodo que visitamos ( $x \geq \text{reco.info}$ ) y no lleguemos al final de la lista ( $\text{reco.sig} \neq \text{null}$ ) avanzamos `reco` al siguiente nodo y fijamos un puntero en el nodo anterior (`atras`)

```
Nodo reco=raiz;  
Nodo atras=raiz;  
while (x>=reco.info && reco.sig!=null) {  
    atras=reco;  
    reco=reco.sig;  
}
```

Cuando salimos del `while` si la condición ( $x \geq \text{reco.info}$ ) continua siendo verdadera significa que se inserta al final de la lista, en caso contrario se inserta en medio de la lista:

```
if (x>=reco.info) {  
    reco.sig=nuevo;  
} else {  
    nuevo.sig=reco;  
    atras.sig=nuevo;  
}
```

[Retornar](#)

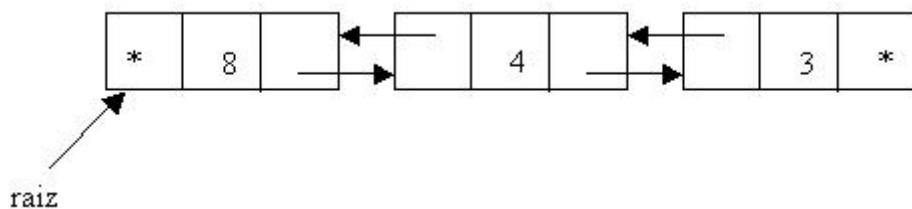
# 48 - Estructuras dinámicas: Listas genéricas doblemente encadenadas

[Listado completo de tutoriales](#)

A las listas vistas hasta el momento podemos recorrerlas solamente en una dirección (Listas simplemente encadenadas). Hay problemas donde se requiere recorrer la lista en ambas direcciones, en estos casos el empleo de listas doblemente encadenadas es recomendable.

Como ejemplo pensemos que debemos almacenar un menú de opciones en una lista, la opción a seleccionar puede ser la siguiente o la anterior, podemos desplazarnos en ambas direcciones.

Representación gráfica de una lista doblemente encadenada:



Observemos que una lista doblemente encadenada tiene dos punteros por cada nodo, uno apunta al nodo siguiente y otro al nodo anterior.

Seguimos teniendo un puntero (raiz) que tiene la dirección del primer nodo.

El puntero sig del último nodo igual que las listas simplemente encadenadas apunta a null, y el puntero ant del primer nodo apunta a null.

Se pueden plantear Listas tipo pila, cola y genéricas con enlace doble.

Hay que tener en cuenta que el requerimiento de memoria es mayor en las listas doblemente encadenadas ya que tenemos dos punteros por nodo.

La estructura del nodo es:

```
class Nodo {  
    int info;  
    Nodo sig, ant;  
}
```

Resolveremos algunos métodos para administrar listas genéricas empleando listas doblemente encadenadas para analizar la mecánica de enlace de nodos.

Muchos de los métodos, para listas simple y doblemente encadenadas no varía, como por ejemplo: el constructor, vacia, cantidad, etc.

## Programa:

```
public class ListaGenerica {  
  
    class Nodo {  
        int info;  
        Nodo ant,sig;  
    }  
  
    private Nodo raiz;  
  
    public ListaGenerica () {  
        raiz=null;  
    }  
  
    void insertar (int pos, int x)  
{  
        if (pos <= cantidad () + 1) {  
            Nodo nuevo = new Nodo ();  
            nuevo.info = x;  
            if (pos == 1){  
                nuevo.sig = raiz;  
                if (raiz!=null)  
                    raiz.ant=nuevo;  
                raiz = nuevo;  
            } else {  
                if (pos == cantidad () + 1) {  
                    Nodo reco = raiz;  
                    while (reco.sig != null) {  
                        reco = reco.sig;  
                    }  
                    reco.sig = nuevo;  
                    nuevo.ant=reco;  
                    nuevo.sig = null;  
                } else {  
                    Nodo reco = raiz;  
                    for (int f = 1 ; f <= pos - 2 ; f++)  
                        reco = reco.sig;  
                    Nodo siguiente = reco.sig;  
                    reco.sig = nuevo;  
                    nuevo.ant=reco;  
                    nuevo.sig = siguiente;  
                    siguiente.ant=nuevo;  
                }  
            }  
        }  
    }  
  
    public int extraer (int pos) {  
        if (pos <= cantidad ()) {  
            int informacion;  
            if (pos == 1) {  
                informacion = raiz.info;  
                raiz = raiz.sig;  
                if (raiz!=null)  
                    raiz.ant=null;  
            } else {  
                Nodo reco;  
                reco = raiz;  
                for (int f = 1 ; f <= pos - 2 ; f++)  
                    reco = reco.sig;  
            }  
        }  
    }  
}
```

```

        Nodo prox = reco.sig;
        reco.sig = prox.sig;
        Nodo siguiente=prox.sig;
        if (siguiente!=null)
            siguiente.ant=reco;
        informacion = prox.info;
    }
    return informacion;
}
else
    return Integer.MAX_VALUE;
}

public void borrar (int pos)
{
    if (pos <= cantidad ())
        if (pos == 1) {
            raiz = raiz.sig;
            if (raiz!=null)
                raiz.ant=null;
        } else {
            Nodo reco;
            reco = raiz;
            for (int f = 1 ; f <= pos - 2 ; f++)
                reco = reco.sig;
            Nodo prox = reco.sig;
            prox=prox.sig;
            reco.sig = prox;
            if (prox!=null)
                prox.ant=reco;
        }
    }
}

public void intercambiar (int pos1, int pos2) {
    if (pos1 <= cantidad () && pos2 <= cantidad ())
        Nodo recol = raiz;
        for (int f = 1 ; f < pos1 ; f++)
            recol = recol.sig;
        Nodo reco2 = raiz;
        for (int f = 1 ; f < pos2 ; f++)
            reco2 = reco2.sig;
        int aux = recol.info;
        recol.info = reco2.info;
        reco2.info = aux;
    }
}

public int mayor () {
    if (!vacia ()) {
        int may = raiz.info;
        Nodo reco = raiz.sig;
        while (reco != null) {
            if (reco.info > may)
                may = reco.info;
            reco = reco.sig;
        }
        return may;
    }
    else
        return Integer.MAX_VALUE;
}

```

```

    }

    public int posMayor() {
        if (!vacia ())
            {
                int may = raiz.info;
                int x=1;
                int pos=x;
                Nodo reco = raiz.sig;
                while (reco != null){
                    if (reco.info > may) {
                        may = reco.info;
                        pos=x;
                    }
                    reco = reco.sig;
                    x++;
                }
                return pos;
            }
        else
            return Integer.MAX_VALUE;
    }

    public int cantidad ()
    {
        int cant = 0;
        Nodo reco = raiz;
        while (reco != null) {
            reco = reco.sig;
            cant++;
        }
        return cant;
    }

    public boolean ordenada() {
        if (cantidad()>1) {
            Nodo reco1=raiz;
            Nodo reco2=raiz.sig;
            while (reco2!=null) {
                if (reco2.info<reco1.info) {
                    return false;
                }
                reco2=reco2.sig;
                reco1=reco1.sig;
            }
        }
        return true;
    }

    public boolean existe(int x) {
        Nodo reco=raiz;
        while (reco!=null) {
            if (reco.info==x)
                return true;
            reco=reco.sig;
        }
        return false;
    }

    public boolean vacia ()
    {
        if (raiz == null)

```

```

        return true;
    else
        return false;
}

public void imprimir ()
{
    Nodo reco = raiz;
    while (reco != null) {
        System.out.print (reco.info + "-");
        reco = reco.sig;
    }
    System.out.println();
}

public static void main(String[] ar) {
    ListaGenerica lg=new ListaGenerica();
    lg.insertar (1, 10);
    lg.insertar (2, 20);
    lg.insertar (3, 30);
    lg.insertar (2, 15);
    lg.insertar (1, 115);
    lg.imprimir ();
    System.out.println ("Luego de Borrar el primero");
    lg.borrar (1);
    lg.imprimir ();
    System.out.println ("Luego de Extraer el segundo");
    lg.extraer (2);
    lg.imprimir ();
    System.out.println ("Luego de Intercambiar el primero con el
tercero");
    lg.intercambiar (1, 3);
    lg.imprimir ();
    if (lg.existe(10))
        System.out.println("Se encuentra el 20 en la lista");
    else
        System.out.println("No se encuentra el 20 en la lista");
    System.out.println("La posición del mayor es:"+lg.posMayor());
    if (lg.ordenada())
        System.out.println("La lista está ordenada de menor a mayor");
    else
        System.out.println("La lista no está ordenada de menor a
mayor");
}
}

```

Para insertar en una determinada posición dentro de la lista:

```
void insertar (int pos, int x)
```

Primero con un if verificamos que exista esa posición en la lista (por ejemplo si la lista tiene 4 nodos podemos insertar hasta la posición 5, es decir uno más allá del último):

```
if (pos <= cantidad () + 1) {
```

Si ingresa al if ya podemos crear el nodo:

```
Nodo nuevo = new Nodo ();
nuevo.info = x;
```

Ahora debemos analizar si la inserción es al principio de la lista, al final o en medio ya que los enlaces varían según donde se lo inserta.

Para saber si se inserta al principio de la lista preguntamos si en pos llega un 1:

```
if (pos == 1){
```

Si llega un 1 luego enlazamos el puntero sig del nodo que creamos con la dirección del primer nodo de la lista (raiz apunta siempre al primer nodo de la lista)

Verificamos si raiz está apuntando actualmente a un nodo, en caso afirmativo enlazamos el puntero ant con el nodo que acabamos de crear y luego desplazamos raiz al nodo creado:

```
nuevo.sig = raiz;
if (raiz!=null)
    raiz.ant=nuevo;
raiz = nuevo;
```

Si no se inserta al principio de la lista preguntamos si se inserta al final:

```
if (pos == cantidad () + 1) {
```

En caso de insertarse al final recorremos la lista hasta el último nodo:

```
Nodo reco = raiz;
while (reco.sig != null) {
    reco = reco.sig;
}
```

y enlazamos el puntero sig del último nodo de la lista con la dirección del nodo que acabamos de crear (disponemos en sig del nodo creado el valor null ya que no hay otro nodo más adelante) El puntero ant del nodo que creamos lo enlazamos con el nodo que era último hasta este momento y está siendo apuntado por reco:

```
reco.sig = nuevo;
nuevo.ant=reco;
nuevo.sig = null;
```

Si no se inserta al principio o al final significa que tenemos que insertar en medio de la lista.

Disponemos un for donde avanzamos un puntero auxiliar y nos detenemos una posición antes a donde tenemos que insertarlo:

```
for (int f = 1 ; f <= pos - 2 ; f++)
    reco = reco.sig;
```

Disponemos otro puntero auxiliar que apunte al nodo próximo a donde está apuntando reco. Ahora enlazamos el puntero sig del nodo apuntado por reco con la dirección del nodo creado y el puntero sig del nodo creado con la dirección del nodo siguiente. El puntero ant del nodo apuntado por nuevo lo enlazamos con el nodo apuntado por raiz y el puntero ant del nodo apuntado por siguiente lo apuntamos a nuevo (con esto tenemos actualizados los cuatro punteros internos a la lista):

```
Nodo siguiente = reco.sig;
reco.sig = nuevo;
nuevo.ant=reco;
nuevo.sig = siguiente;
siguiente.ant=nuevo;
```

El método extraer recibe como parámetro la posición del nodo a extraer:

```
public int extraer (int pos) {
```

Primero verificamos que la posición exista en la lista:

```
if (pos <= cantidad ()) {
```

En caso que exista verificamos si el nodo a extraer es el primero de la lista (este análisis debe hacerse ya que si es el primero de la lista se modifica el puntero raiz):

```
if (pos == 1) {
```

Si es el primero guardamos en una variable auxiliar la información del nodo y avanzamos el puntero raiz, luego si raiz apunta a un nodo disponemos el puntero ant de dicho nodo a null:

```
informacion = raiz.info;
raiz = raiz.sig;
if (raiz!=null)
    raiz.ant=null;
```

Si el nodo a extraer no está al principio de la lista avanzamos con una estructura repetitiva hasta el nodo anterior a extraer:

```
for (int f = 1 ; f <= pos - 2 ; f++)
    reco = reco.sig;
```

Luego definimos otro puntero auxiliar y lo disponemos en el siguiente nodo a donde está apuntando reco:

```
Nodo prox = reco.sig;
```

Ahora enlazamos el puntero sig del nodo apuntado por reco al nodo siguiente del nodo apuntado por prox (es decir el nodo apuntado por prox queda fuera de la lista) disponemos finalmente otro puntero llamado siguiente que apunte al nodo que se encuentra una posición más adelante del nodo apuntado por prox, si dicho puntero apunta a un nodo actualizamos el puntero ant de dicho nodo con la dirección del nodo apuntado por reco:

```
reco.sig = prox.sig;
Nodo siguiente=prox.sig;
if (siguiente!=null)
    siguiente.ant=reco;
informacion = prox.info;
```

El método borrar es muy similar al método extraer con la diferencia de que no retorna valor:

```
public void borrar (int pos)
{
    if (pos <= cantidad ())
        {
            if (pos == 1) {
                raiz = raiz.sig;
                if (raiz!=null)
                    raiz.ant=null;
            } else {
                Nodo reco;
                reco = raiz;
                for (int f = 1 ; f <= pos - 2 ; f++)
                    reco = reco.sig;
                Nodo prox = reco.sig;
                prox=prox.sig;
                reco.sig = prox;
                if (prox!=null)
                    prox.ant=reco;
            }
        }
}
```

```
    }  
}
```

El método intercambiar recibe dos enteros que representan las posiciones de los nodos que queremos intercambiar sus informaciones:

```
public void intercambiar (int pos1, int pos2) {
```

Mediante un if verificamos que las dos posiciones existan en la lista:

```
    if (pos1 <= cantidad () && pos2 <= cantidad ()) {
```

Definimos un puntero auxiliar llamado reco1, lo inicializamos con la dirección del primer nodo y mediante un for avanzamos hasta la posición almacenada en pos1:

```
    Nodo reco1 = raiz;  
    for (int f = 1 ; f < pos1 ; f++)  
        reco1 = reco1.sig;
```

De forma similar con un segundo puntero auxiliar avanzamos hasta la posición indicada por pos2:

```
    Nodo reco2 = raiz;  
    for (int f = 1 ; f < pos2 ; f++)  
        reco2 = reco2.sig;
```

Por último intercambiamos las informaciones que almacenan cada nodo:

```
    int aux = reco1.info;  
    reco1.info = reco2.info;  
    reco2.info = aux;
```

El método que retorna el mayor de la lista:

```
public int mayor () {
```

Verificamos que la lista no esté vacía:

```
    if (!vacia ()) {
```

Suponemos que el mayor es el primero de la lista e inicializamos un puntero auxiliar con la dirección del segundo nodo de la lista:

```
int may = raiz.info;
Nodo reco = raiz.sig;
```

Mediante una estructura repetitiva recorremos toda la lista:

```
while (reco != null) {
```

Cada vez que encontramos un nodo con información mayor que la variable may la actualizamos con este nuevo valor y avanzamos el puntero reco para visitar el siguiente nodo:

```
if (reco.info > may)
    may = reco.info;
reco = reco.sig;
```

Fuera de la estructura repetitiva retornamos el mayor:

```
return may;
```

El método que retorna la posición del mayor es similar al anterior con la salvedad que debemos almacenar en otro auxiliar la posición donde se almacena el mayor:

```
public int posMayor() {
    if (!vacia ()) {
        int may = raiz.info;
        int x=1;
        int pos=x;
        Nodo reco = raiz.sig;
        while (reco != null){
            if (reco.info > may) {
                may = reco.info;
                pos=x;
            }
            reco = reco.sig;
            x++;
        }
        return pos;
    }
    else
        return Integer.MAX_VALUE;
}
```

El método que debe retornar si está ordenada la lista de menor a mayor es:

```
public boolean ordenada() {
```

Lo primero que verificamos si la lista tiene más de un nodo significa que debemos controlarla:

```
if (cantidad(>1) {
```

Disponemos dos punteros auxiliares con las direcciones del primer y segundo nodo de la lista:

```
Nodo reco1=raiz;  
Nodo reco2=raiz.sig;
```

Mediante un while mientras no se finaliza la lista:

```
while (reco2!=null) {
```

controlamos si la información del segundo nodo es menor al nodo anterior significa que la lista no está ordenada y podemos parar el análisis retornando un false

```
if (reco2.info<reco1.info) {  
    return false;
```

Dentro del while avanzamos los dos punteros a sus nodos siguientes respectivamente.

```
reco2=reco2.sig;  
reco1=reco1.sig;
```

Fuera del while retornamos true indicando que la lista está ordenada de menor a mayor

```
return true;
```

El método existe:

```
public boolean existe(int x) {
```

Mediante un while recorremos la la lista:

```
Nodo reco=raiz;  
while (reco!=null) {
```

y en cada nodo que visitamos controlamos si el parámetro x es igual a la información del nodo, en caso afirmativo salimos del método retornando true:

```
if (reco.info==x)
    return true;
reco=reco.sig;
```

Fuera del while retornamos false indicando que ningún nodo coincide con el parámetro x:

```
return false;
```

## Problemas propuestos

1. Plantear una clase para administrar una lista genérica doblemente encadenada implementando los siguientes métodos:
  - a) Insertar un nodo al principio de la lista.
  - b) Insertar un nodo al final de la lista.
  - c) Insertar un nodo en la segunda posición. Si la lista está vacía no se inserta el nodo.
  - d) Insertar un nodo en la ante última posición.
  - e) Borrar el primer nodo.
  - f) Borrar el segundo nodo.
  - g) Borrar el último nodo.
  - h) Borrar el nodo con información mayor.

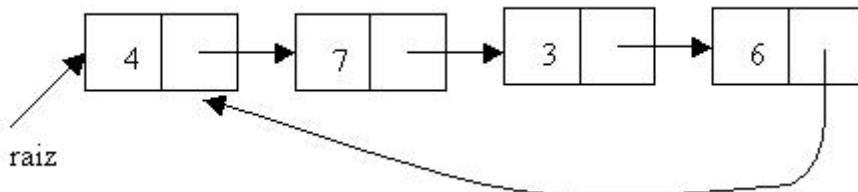
[Solución](#)

[Retornar](#)

# 49 - Estructuras dinámicas: Listas genéricas circulares

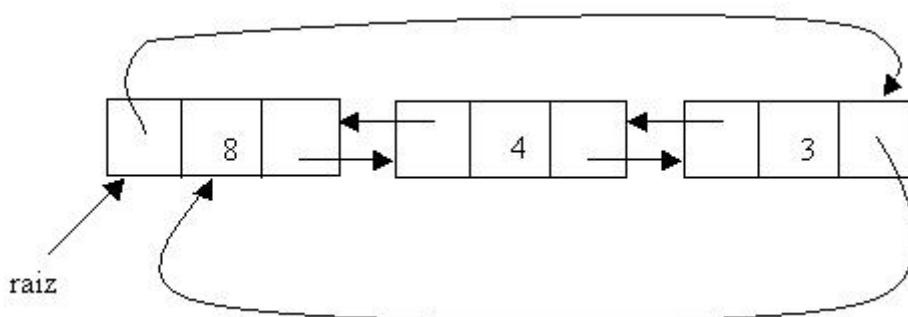
[Listado completo de tutoriales](#)

Una lista circular simplemente encadenada la podemos representar gráficamente:



Observemos que el puntero sig del último nodo apunta al primer nodo. En este tipo de listas si avanzamos raiz no perdemos la referencia al nodo anterior ya que es un círculo.

Una lista circular puede también ser doblemente encadenada:



El puntero ant del primer nodo apunta al último nodo de la lista y el puntero sig del último nodo de la lista apunta al primero.

Resolveremos algunos métodos para administrar listas genéricas circulares doblemente encadenadas para analizar la mecánica de enlace de nodos.

## Programa:

```
public class ListaCircular {  
  
    class Nodo {  
        int info;  
        Nodo ant,sig;  
    }  
  
    private Nodo raiz;
```

```
public ListaCircular () {
    raiz=null;
}

public void insertarPrimero(int x) {
    Nodo nuevo=new Nodo();
    nuevo.info=x;
    if (raiz==null) {
        nuevo.sig=nuevo;
        nuevo.ant=nuevo;
        raiz=nuevo;
    } else {
        Nodo ultimo=raiz.ant;
        nuevo.sig=raiz;
        nuevo.ant=ultimo;
        raiz.ant=nuevo;
        ultimo.sig=nuevo;
        raiz=nuevo;
    }
}

public void insertarUltimo(int x) {
    Nodo nuevo=new Nodo();
    nuevo.info=x;
    if (raiz==null) {
        nuevo.sig=nuevo;
        nuevo.ant=nuevo;
        raiz=nuevo;
    } else {
        Nodo ultimo=raiz.ant;
        nuevo.sig=raiz;
        nuevo.ant=ultimo;
        raiz.ant=nuevo;
        ultimo.sig=nuevo;
    }
}

public boolean vacia ()
{
    if (raiz == null)
        return true;
    else
        return false;
}

public void imprimir ()
{
    if (!vacia()) {
        Nodo reco=raiz;
        do {
            System.out.print (reco.info + "-");
            reco = reco.sig;
        } while (reco!=raiz);
        System.out.println();
    }
}

public int cantidad ()
{
    int cant = 0;
    if (!vacia()) {
```

```

        Nodo reco=raiz;
        do {
            cant++;
            reco = reco.sig;
        } while (reco!=raiz);
    }
    return cant;
}

public void borrar (int pos)
{
    if (pos <= cantidad ()) {
        if (pos == 1) {
            if (cantidad()==1) {
                raiz=null;
            } else {
                Nodo ultimo=raiz.ant;
                raiz = raiz.sig;
                ultimo.sig=raiz;
                raiz.ant=ultimo;
            }
        } else {
            Nodo reco = raiz;
            for (int f = 1 ; f <= pos - 1 ; f++)
                reco = reco.sig;
            Nodo anterior = reco.ant;
            reco=reco.sig;
            anterior.sig=reco;
            reco.ant=anterior;
        }
    }
}

public static void main(String[] ar) {
    ListaCircular lc=new ListaCircular();
    lc.insertarPrimero(100);
    lc.insertarPrimero(45);
    lc.insertarPrimero(12);
    lc.insertarPrimero(4);
    System.out.println("Luego de insertar 4 nodos al principio");
    lc.imprimir();
    lc.insertarUltimo(250);
    lc.insertarUltimo(7);
    System.out.println("Luego de insertar 2 nodos al final");
    lc.imprimir();
    System.out.println("Cantidad de nodos:"+lc.cantidad());
    System.out.println("Luego de borrar el de la primera posición:");
    lc.borrar(1);
    lc.imprimir();
    System.out.println("Luego de borrar el de la cuarta posición:");
    lc.borrar(4);
    lc.imprimir();
}
}
}

```

Para insertar al principio de una lista circular doblemente encadenada:

```
public void insertarPrimero(int x) {
```

Creamos un nodo y guardamos la información:

```
Nodo nuevo=new Nodo();
nuevo.info=x;
```

Si la lista está vacía luego tanto el puntero sig y ant apuntan a si mismo ya que debe ser circular (y raiz apunta al nodo creado):

```
if (raiz==null) {
    nuevo.sig=nuevo;
    nuevo.ant=nuevo;
    raiz=nuevo;
```

En caso que la lista no esté vacía disponemos un puntero al final de la lista (el puntero ant del primer nodo tiene dicha dirección):

```
} else {
    Nodo ultimo=raiz.ant;
```

El nodo a insertar lo enlazamos previo al nodo apuntado por raiz:

```
nuevo.sig=raiz;
nuevo.ant=ultimo;
raiz.ant=nuevo;
ultimo.sig=nuevo;
```

Finalmente hacemos que raiz apunte al nodo creado luego de haber hecho todos los enlaces:

```
raiz=nuevo;
```

Para insertar un nodo al final de la lista:

```
public void insertarUltimo(int x) {
```

El algoritmo es idéntico al método que inserta al principio con la salvedad que no desplazamos raiz con la dirección del nodo creado (es decir al insertar en la posición anterior del primer nodo lo que estamos haciendo realmente es insertar al final de la lista):

```

Nodo nuevo=new Nodo();
nuevo.info=x;
if (raiz==null) {
    nuevo.sig=nuevo;
    nuevo.ant=nuevo;
    raiz=nuevo;
} else {
    Nodo ultimo=raiz.ant;
    nuevo.sig=raiz;
    nuevo.ant=ultimo;
    raiz.ant=nuevo;
    ultimo.sig=nuevo;
}
}
}

```

Para imprimir la lista ya no podemos disponer un puntero reco que apunte al primer nodo y que se detenga cuando encuentre un nodo que el atributo sig almacene null.

```
public void imprimir ()
```

Si la lista no está vacía disponemos un puntero en el primer nodo y utilizamos un do/while para recorrer la lista. La condición del do/while es que se repita mientras el puntero reco sea distinto a raiz (es decir que no haya dado toda la vuelta a la lista):

```

if (!vacia()) {
    Nodo reco=raiz;
    do {
        System.out.print (reco.info + "-");
        reco = reco.sig;
    } while (reco!=raiz);
    System.out.println();
}
}

```

Para borrar el nodo de una determinada posición:

```
public void borrar (int pos)
```

Debemos primero identificar si es el primero de la lista (ya que en este caso se modifica el puntero externo raiz):

```

if (pos <= cantidad ()) {
    if (pos == 1) {

```

Si es el primero y el único de la lista hacemos que raiz apunte a null:

```
if (cantidad()==1) {  
    raiz=null;
```

Si no disponemos un puntero al final de la lista, avanzamos raiz y enlazamos el último nodo con el segundo de la lista:

```
} else {  
    Nodo ultimo=raiz.ant;  
    raiz = raiz.sig;  
    ultimo.sig=raiz;  
    raiz.ant=ultimo;  
}
```

En caso que queremos borrar un nodo que se encuentra en medio de la lista o inclusive al final debemos recorrer con un for hasta el nodo que queremos borrar y luego disponemos un puntero en el nodo anterior y otro puntero en el nodo siguiente. Seguidamente procedemos a enlazar los nodos:

```
Nodo reco = raiz;  
for (int f = 1 ; f <= pos - 1 ; f++)  
    reco = reco.sig;  
Nodo anterior = reco.ant;  
reco=reco.sig;  
anterior.sig=reco;  
reco.ant=anterior;
```

[Retornar](#)

# 50 - Recursividad: Conceptos básicos

[Listado completo de tutoriales](#)

Primero debemos decir que la recursividad no es una estructura de datos, sino que es una técnica de programación que nos permite que un bloque de instrucciones se ejecute n veces. Reemplaza en ocasiones a estructuras repetitivas.

Este concepto será de gran utilidad para el capítulo de la estructura de datos tipo árbol.

La recursividad es un concepto difícil de entender en principio, pero luego de analizar diferentes problemas aparecen puntos comunes.

En Java los métodos pueden llamarse a sí mismos. Si dentro de un método existe la llamada a sí mismo decimos que el método es recursivo.

Cuando un método se llama a sí mismo, se asigna espacio en la pila para las nuevas variables locales y parámetros.

Al volver de una llamada recursiva, se recuperan de la pila las variables locales y los parámetros antiguos y la ejecución se reanuda en el punto de la llamada al método.

## Problema 1:

Implementación de un método recursivo.

## Programa:

```
public class Recursividad {  
  
    void repetir() {  
        repetir();  
    }  
  
    public static void main(String[] ar) {  
        Recursividad re=new Recursividad();  
        re.repetir();  
    }  
}
```

La función repetir es recursiva porque dentro de la función se llama a sí misma. Cuando ejecuta este programa se bloqueará y generará una excepción: "Exception in thread "main" java.lang.StackOverflowError"

Analicemos como funciona:

Primero se ejecuta la función main, luego de crear un objeto llamamos a la función repetir.

Hay que tener en cuenta que cada vez que se llama a una función se reservan 4 bytes de la memoria que se liberarán cuando finalice su ejecución.

La primera línea de la función llama a la función repetir, es decir que se reservan 4 bytes nuevamente. Se ejecuta nuevamente una instancia de la función repetir y así sucesivamente hasta que la pila estática se colme y se cuelgue el programa.

## Problema 2:

Implementación de un método recursivo que reciba un parámetro de tipo entero y luego llame en forma recursiva con el valor del parámetro menos 1.

### Programa:

```
public class Recursividad {  
  
    void imprimir(int x) {  
        System.out.println(x);  
        imprimir(x-1);  
    }  
  
    public static void main(String[] ar) {  
        Recursividad re=new Recursividad();  
        re.imprimir(5);  
    }  
}
```

Desde la main se llama a la función imprimir y se le envía el valor 5. El parámetro x recibe el valor 5. Se ejecuta el algoritmo de la función, imprime el contenido del parámetro (5) y seguidamente se llama a una función, en este caso a sí misma (por eso decimos que es una función recursiva), enviándole el valor 4.

El parámetro x recibe el valor 4 y se imprime en pantalla el cuatro, llamando nuevamente a la función imprimir enviándole el valor 3.

Si continuamos este algoritmo podremos observar que en pantalla se imprime:

5 4 3 2 1 0 ?1 ?2 ?3 . . . . . . . .

hasta que se bloquee el programa.

Tener en cuenta que cada llamada a una función consume 4 bytes por la llamada y en este caso 4 bytes por el parámetro x. Como nunca finaliza la ejecución completa de las funciones se desborda la pila estática por las sucesivas llamadas.

### Problema 3:

Implementar un método recursivo que imprima en forma descendente de 5 a 1 de uno en uno.

#### Programa:

```
public class Recursividad {  
  
    void imprimir(int x) {  
        if (x>0) {  
            System.out.println(x);  
            imprimir(x-1);  
        }  
    }  
  
    public static void main(String[] ar) {  
        Recursividad re=new Recursividad();  
        re.imprimir(5);  
    }  
}
```

Ahora si podemos ejecutar este programa y observar los resultados en pantalla. Se imprimen los números 5 4 3 2 1 y no se bloquea el programa.

Analice qué sucede cada vez que el if ( $x>0$ ) se evalúa como falso, ¿a qué línea del programa retorna?

### Problema 4:

Imprimir los números de 1 a 5 en pantalla utilizando recursividad.

#### Programa:

```
public class Recursividad {
```

```

void imprimir(int x) {
    if (x>0) {
        imprimir(x-1);
        System.out.println(x);
    }
}

public static void main(String[] ar) {
    Recursividad re=new Recursividad();
    re.imprimir(5);
}
}

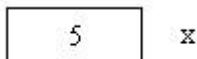
```

Con este ejemplo se presenta una situación donde debe analizarse línea a línea la ejecución del programa y el porque de estos resultados.

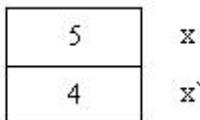
¿Por qué se imprime en pantalla 1 2 3 4 5 ?

Veamos como se apilan las llamadas recursivas:

En la primera llamada desde la función main el parámetro x recibe el valor 5.

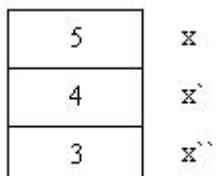


Cuando llamamos desde la misma función le enviamos el valor de x menos 1 y la memoria queda de la siguiente forma:



Debemos entender que el parámetro x en la nueva llamada está en otra parte de la memoria y que almacena un 4, nosotros le llamaremos x prima.

Comienza a ejecutarse la función, la condición del if se valúa como verdadero por lo que entra al bloque y llama recursivamente a la función imprimir pasándole el valor 3 al parámetro.



Nuevamente la condición se valúa como verdadero y llama a la función enviándole un 2, lo mismo ocurre cuando le envía un 1 y un 0.

|   |        |
|---|--------|
| 5 | x      |
| 4 | x`     |
| 3 | x``    |
| 2 | x```   |
| 1 | x````  |
| 0 | x````` |

```
void imprimir(int x) {
    if (x>0) {
        imprimir(x-1);
        System.out.println(x);
    }
}
```

Cuando x vale 0 la condición del if se valúa como falsa y sale de la función imprimir.

¿Qué línea ahora se ejecuta ?

Vuelve a la función main ? NO.

Recordemos que la última llamada de la función imprimir se había hecho desde la misma función imprimir por lo que vuelve a la línea:

```
System.out.println(x);
```

Ahora si analicemos que valor tiene el parámetro x. Observemos la pila de llamadas del gráfico:

|   |       |
|---|-------|
| 5 | x     |
| 4 | x`    |
| 3 | x``   |
| 2 | x```  |
| 1 | x```` |

x cuarta tiene el valor 1. Por lo que se imprime dicho valor en pantalla.

Luego de imprimir el 1 finaliza la ejecución de la función, se libera el espacio ocupado por el parámetro x y pasa a ejecutarse la siguiente línea donde se había llamado la función:

```
System.out.println(x);
```

Ahora x en esta instancia de la función tiene el valor 2.

Así sucesivamente hasta liberar todas las llamadas recursivas.

Es importante tener en cuenta que siempre en una función recursiva debe haber un if para finalizar la recursividad ( en caso contrario la función recursiva será infinita y provocará que el programa se bloquee)

## Problema 5:

Otro problema típico que se presenta para analizar la recursividad es el obtener el factorial de un número.

Recordar que el factorial de un número es el resultado que se obtiene de multiplicar dicho número por el anterior y así sucesivamente hasta llegar a uno.

Ej. el factorial de 4 es  $4 * 3 * 2 * 1$  es decir 24.

### Programa:

```
public class Recursividad {  
  
    int factorial(int fact) {  
        if (fact>0) {  
            int valor=fact * factorial(fact-1);  
            return valor;  
        } else  
            return 1;  
    }  
  
    public static void main(String[] ar) {  
        Recursividad re=new Recursividad();  
        int f=re.factorial(4);  
        System.out.println("El factorial de 4 es "+f);  
    }  
}
```

La función factorial es recursiva porque desde la misma función llamamos a la función factorial.

Debemos hacer el seguimiento del problema para analizar como se calcula.

La memoria en la primera llamada:

|   |
|---|
| 4 |
|   |

fact  
valor

fact recibe el valor 4 y valor se cargará con el valor que se obtenga con el producto de fact por el valor devuelto por la función factorial (llamada recursiva)

|   |
|---|
| 4 |
|   |
| 3 |
|   |

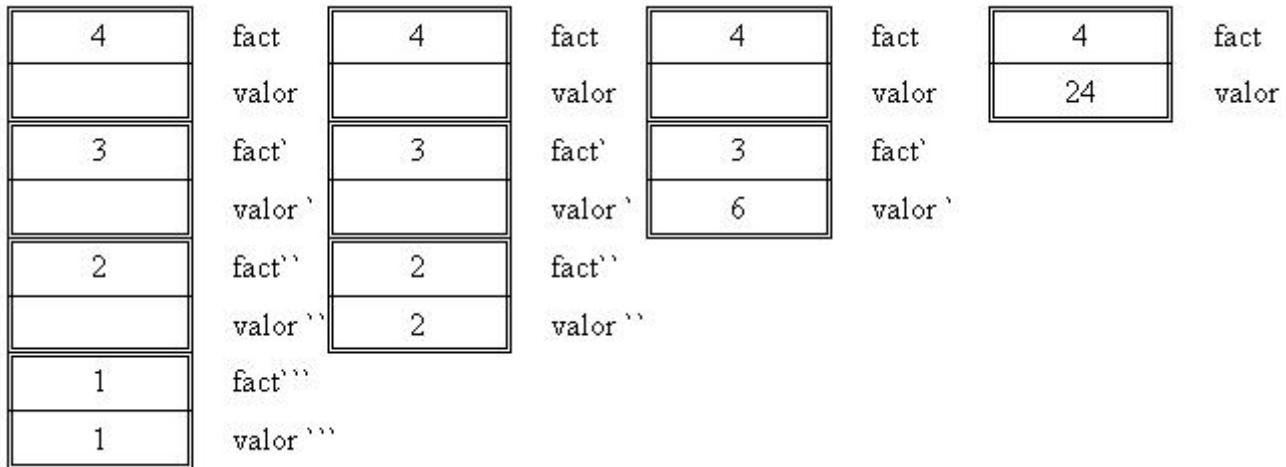
fact  
valor  
fact'  
valor'

Nuevamente se llama recursivamente hasta que el parámetro fact reciba el valor 0.

|   |
|---|
| 4 |
|   |
| 3 |
|   |
| 2 |
|   |
| 1 |
|   |
| 0 |

fact  
valor  
fact'  
valor'  
fact''  
valor''  
fact'''  
valor'''  
fact''''

Cuando fact recibe un cero la condición del if se valúa como falsa y ejecuta el else retornando un 1, la variable local de la llamada anterior a la función queda de la siguiente manera:



Es importantísimo entender la liberación del espacio de las variables locales y los parámetros en las sucesivas llamadas recursivas.

Por último la función main recibe "valor", en este caso el valor 24.

## Problema 6:

Implementar un método recursivo para ordenar los elementos de un vector.

### Programa:

```
class Recursividad {
    static int [] vec = {312, 614, 88, 22, 54};

    void ordenar (int [] v, int cant) {
        if (cant > 1) {
            for (int f = 0 ; f < cant - 1 ; f++)
                if (v [f] > v [f + 1]) {
                    int aux = v [f];
                    v [f] = v [f + 1];
                    v [f + 1] = aux;
                }
            ordenar (v, cant - 1);
        }
    }

    void imprimir () {
        for (int f = 0 ; f < vec.length ; f++)
            System.out.print (vec [f] + " ");
        System.out.println ("\n");
    }
}
```

```
}
```

```
public static void main (String [] ar)  {  
    Recursividad r = new Recursividad();  
    r.imprimir ();  
    r.ordenar (vec, vec.length);  
    r.imprimir ();  
}  
}
```

Hasta ahora hemos visto problemas que se pueden resolver tanto con recursividad como con estructuras repetitivas.

Es muy importante tener en cuenta que siempre que podamos emplear un algoritmo no recursivo será mejor (ocupa menos memoria de ram y se ejecuta más rápidamente)

Pero hay casos donde el empleo de recursividad hace mucho más sencillo el algoritmo (tener en cuenta que no es el caso de los tres problemas vistos previamente)

[Retornar](#)

# 51 - Recursividad: Problemas donde conviene aplicar la recursividad

[Listado completo de tutoriales](#)

En el concepto anterior se vieron pequeños problemas para entender como funciona la recursividad, pero no se desarrollaron problemas donde conviene utilizar la recursividad.

## Problema 1:

Imprimir la información de una lista simplemente encadenada de atrás para adelante. El empleo de estructuras repetitivas para resolver este problema es bastante engorroso y lento (debemos avanzar hasta el último nodo e imprimir, luego avanzar desde el principio hasta el anteúltimo nodo y así sucesivamente) El empleo de la recursividad para este problema hace más sencillo su solución.

## Programa:

```
public class Recursividad {

    class Nodo {
        int info;
        Nodo sig;
    }

    private Nodo raiz;

    void insertarPrimero(int x)
    {
        Nodo nuevo = new Nodo ();
        nuevo.info = x;
        nuevo.sig=raiz;
        raiz=nuevo;
    }

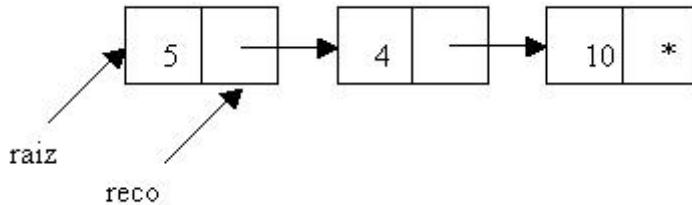
    public void imprimirInversa(Nodo reco) {
        if (reco!=null) {
            imprimirInversa(reco.sig);
            System.out.print(reco.info+"-");
        }
    }

    public void imprimirInversa () {
        imprimirInversa(raiz);
    }
}
```

```

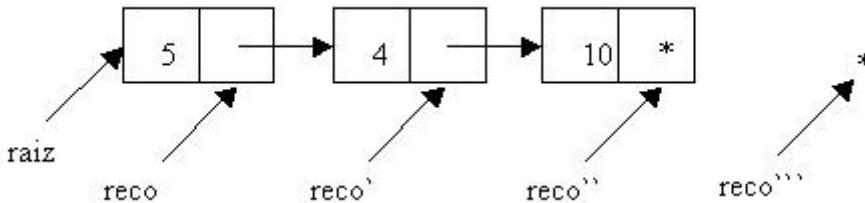
public static void main(String[] ar) {
    Recursividad r=new Recursividad();
    r.insertarPrimero (10);
    r.insertarPrimero(4);
    r.insertarPrimero(5);
    r.imprimirInversa();
}
}

```



Cuando llamamos al método recursivo le enviamos raiz y el parámetro reco recibe esta dirección. Si reco es distinto a null llamamos recursivamente al método enviándole la dirección del puntero sig del nodo.

Por lo que el parámetro reco recibe la dirección del segundo nodo.



Podemos observar como en las distintas llamadas recursivas el parámetro reco apunta a un nodo. Cuando se van desapilando las llamadas recursivas se imprime primeramente el 10 luego el 4 y por último el 5.

## Problema 2:

Recorrer un árbol de directorios en forma recursiva.

### Programa:

```

import java.io.File;
public class Recursividad{

    public void leer(String inicio,String altura)
    {
        File ar=new File(inicio);
        String[] dir=ar.list();
        for(int f=0;f<dir.length;f++){
            File ar2=new File(inicio+dir[f]);
            if (ar2.isFile())
                System.out.println(altura+dir[f]);
    }
}
}

```

```

        if (ar2.isDirectory()) {
            System.out.println(altura +
"Directorio:"+dir[f].toUpperCase());
            leer(inicio+dir[f]+"\\",altura+" ");
        }
    }

    public static void main(String[] arguments)
{
    Recursividad rec=new Recursividad();
    rec.leer("d:\\windows\\","");
}
}

```

Para recorrer y visitar todos los directorios y archivos de un directorio debemos implementar un algoritmo recursivo que reciba como parámetro el directorio inicial donde comenzaremos a recorrer:

```
public void leer(String inicio, String altura)
```

Creamos un objeto de la clase File con el directorio que llega como parámetro y mediante el método list obtenemos todos los archivos y directorios de dicho directorio:

```
File ar=new File(inicio);
String[] dir=ar.list();
```

Mediante un for recorremos todo el vector que contiene la lista de archivos y directorios:

```
for(int f=0;f<dir.length;f++){
```

Creamos un objeto de la clase File para cada directorio y archivo:

```
File ar2=new File(inicio+dir[f]);
```

Luego de crear un objeto de la clase file podemos verificar si se trata de un archivo o directorio:

```

if (ar2.isFile())
    System.out.println(altura+dir[f]);
if (ar2.isDirectory())
    System.out.println(altura +
"Directorio:"+dir[f].toUpperCase());
    leer(inicio+dir[f]+"\\",altura+" ");
}

```

Si es un archivo lo mostramos y si es un directorio además de mostrarlo llamamos recursivamente al método leer con el directorios nuevo a procesar.

### Problema 3:

Desarrollar un programa que permita recorrer un laberinto e indique si tiene salida o no. Para resolver este problema al laberinto lo representaremos con una matriz de 10 x 10 JLabel.

El valor:

|     |                    |
|-----|--------------------|
| "0" | Representa pasillo |
| "1" | Representa pared   |
| "9" | Persona            |
| "s" | Salida             |

A la salida ubicarla en la componente de la fila 9 y columna 9 de la matriz. La persona comienza a recorrer el laberinto en la fila 0 y columna 0. Los ceros y unos disponerlos en forma aleatoria (con la función random)

### Programa:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class Laberinto extends JFrame implements ActionListener {
    JLabel[][] l;
    JButton b1;
    JButton b2;
    boolean salida;
    Laberinto()
    {
        setLayout(null);
        l=new JLabel[10][10];
        for(int f=0;f<10;f++) {
            for(int c=0;c<10;c++) {
                l[f][c]=new JLabel();
                l[f][c].setBounds(20+c*20,50+f*20,20,20);
                add(l[f][c]);
            }
        }
        b1=new JButton("Recorrer");
        b1.setBounds(10,300,100,25);
        add(b1);
        b1.addActionListener(this);
        b2=new JButton("Crear");
        b2.setBounds(120,300,100,25);
        add(b2);
        b2.addActionListener(this);
        crear();
    }
}
```

```

public void crear()
{
    for(int f=0;f<10;f++) {
        for(int c=0;c<10;c++) {
            int a=(int)(Math.random()*4);
            l[f][c].setForeground(Color.black);
            if (a==0)
                l[f][c].setText("1");
            else
                l[f][c].setText("0");
        }
    }
    l[9][9].setText("s");
    l[0][0].setText("0");
}

public void recorrer(int fil,int col)
{
    if (fil>=0 && fil<10 && col>=0 && col<10 && salida==false) {
        if (l[fil][col].getText().equals("s"))
            salida=true;
        else
            if (l[fil][col].getText().equals("0")) {
                l[fil][col].setText("9");
                l[fil][col].setForeground(Color.red);
                recorrer(fil,col+1);
                recorrer(fil+1,col);
                recorrer(fil-1,col);
                recorrer(fil,col-1);
            }
    }
}

public void actionPerformed(ActionEvent e)
{
    if (e.getSource()==b1) {
        salida=false;
        recorrer(0,0);
        if (salida)
            setTitle("tiene salida");
        else
            setTitle("no tiene salida");
    }
    if (e.getSource()==b2)
        crear();
}

public static void main(String[] ar)
{
    Laberinto l=new Laberinto();
    l.setBounds(0,0,300,400);
    l.setVisible(true);
}
}

```

El método más importante es el recorrer:

```
public void recorrer(int fil,int col)
```

Primero verificamos si la coordenada a procesar del laberinto se encuentra dentro de los límites correctos y además no hayamos encontrado la salida hasta el momento:

```
if (fil>=0 && fil<10 && col>=0 && col<10 && salida==false)
```

Si entra al if anterior verificamos si estamos en la salida:

```
if (l[fil][col].getText().equals("s"))
    salida=true;
```

En el caso que no estemos en la salida verificamos si estamos en pasillo:

```
if (l[fil][col].getText().equals("0")) {
```

En caso de estar en el pasillo procedemos a fijar dicha JLabel con el carácter "9" e intentamos desplazarnos en las cuatro direcciones (arriba, abajo, derecha e izquierda), este desplazamiento lo logramos llamando recursivamente:

```
l[fil][col].setText("9");
l[fil][col].setForeground(Color.red);
recorrer(fil,col+1);
recorrer(fil+1,col);
recorrer(fil-1,col);
recorrer(fil,col-1);
```

## Problemas propuestos

1. Desarrollar el juego del Buscaminas. Definir una matriz de 10\*10 de JButton y disponer una 'b' para las bombas (10 diez) un cero en los botones que no tienen bombas en su perímetro, un 1 si tiene una bomba en su perímetro y así sucesivamente. Cuando se presiona un botón si hay un cero proceder en forma recursiva a destapar los botones que se encuentran a sus lados. Disponer el mismo color de frente y fondo de los botones para que el jugador no pueda ver si hay bombas o no.

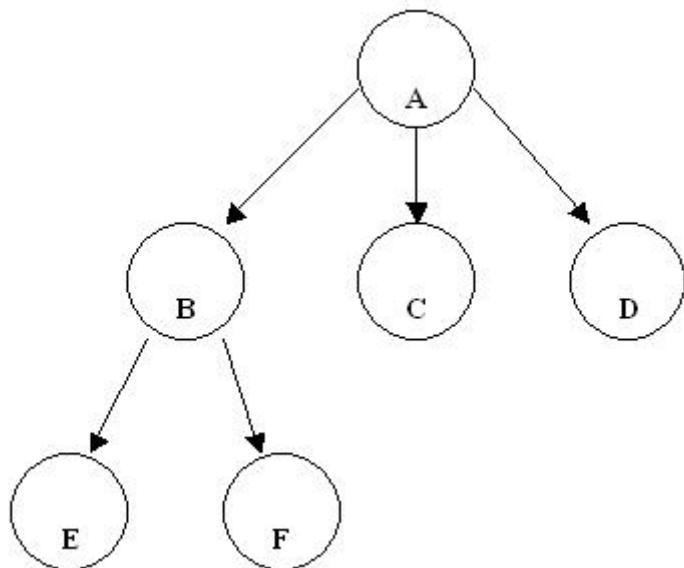
[Solución](#)

[Retornar](#)

# 52 - Estructuras dinámicas: Conceptos de árboles

[Listado completo de tutoriales](#)

Igual que la lista, el árbol es una estructura de datos. Son muy eficientes para la búsqueda de información. Los árboles soportan estructuras no lineales.



Algunos conceptos de la estructura de datos tipo árbol:

**Nodo hoja:** Es un nodo sin descendientes (Nodo terminal)

Ej. Nodos E ? F ? C y D.

**Nodo interior:** Es un nodo que no es hoja.

Ej. Nodos A y B.

**Nivel de un árbol:** El nodo A está en el nivel 1 sus descendientes directos están en el nivel 2 y así sucesivamente.

El nivel del árbol está dado por el nodo de máximo nivel.

Ej. Este árbol es de nivel 3.

**Grado de un nodo:** es el número de nodos hijos que tiene dicho nodo (solo se tiene en cuenta los nodos interiores)

Ej. El nodo A tiene grado 3.

El nodo B tiene grado 2.

Los otros nodos no tienen grado porque no tienen descendientes.

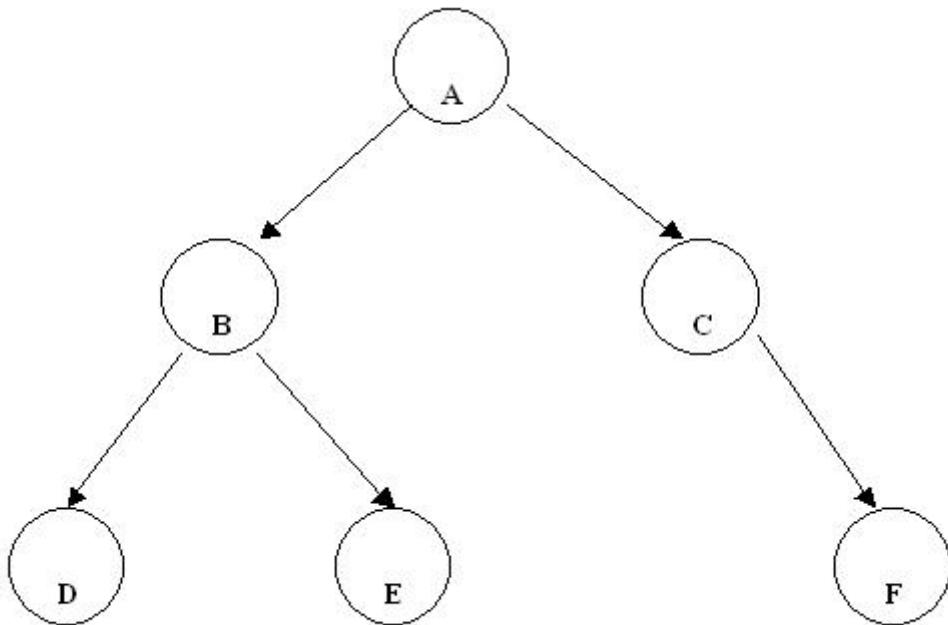
**Grado de un árbol:** Es el máximo de los grados de todos los nodos de un árbol.

Ej. El grado del árbol es 3.

**Longitud de camino del nodo x:** Al número de arcos que deben ser recorridos para llegar a un nodo x, partiendo de la raíz.

La raíz tiene longitud de camino 1, sus descendientes directos tienen longitud de camino 2, etc. En forma general un nodo en el nivel i tiene longitud de camino i.

**Árbol binario:** Un árbol es binario si cada nodo tiene como máximo 2 descendientes.



Para cada nodo está definido el subárbol izquierdo y el derecho.

Para el nodo A el subárbol izquierdo está constituido por los nodos B, D y E. Y el subárbol derecho está formado por los nodos C y F.

Lo mismo para el nodo B tiene el subárbol izquierdo con un nodo (D) y un nodo en el subárbol derecho (E).

El nodo D tiene ambos subárboles vacíos.

El nodo C tiene el subárbol izquierdo vacío y el subárbol derecho con un nodo (F).

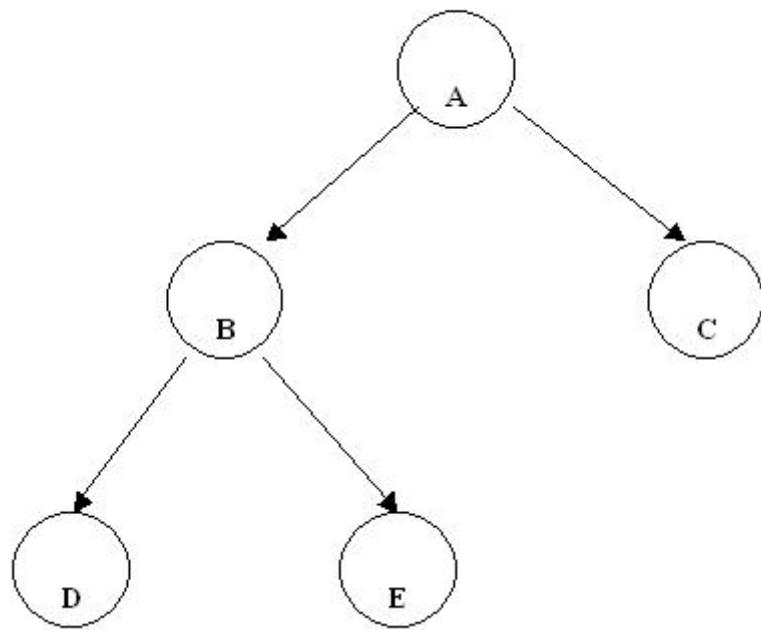
**Árbol binario perfectamente equilibrado:** Si para cada nodo el número de nodos en el subárbol izquierdo y el número de nodos en el subárbol derecho, difiere como mucho en una unidad.

Hay que tener en cuenta todos los nodos del árbol.

El árbol de más arriba es perfectamente equilibrado.

Ej. árbol que no es perfectamente equilibrado:

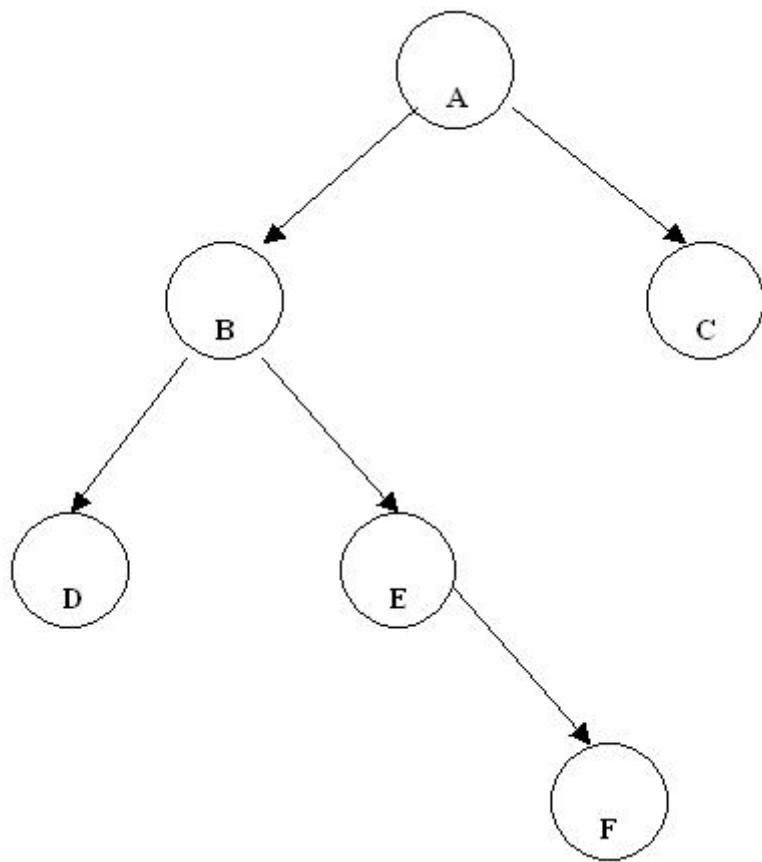
El nodo A tiene 3 nodos en el subárbol izquierdo y solo uno en el subárbol derecho, por lo que no es perfectamente equilibrado.



**Árbol binario completo:** Es un árbol binario con hojas como máximo en los niveles  $n-1$  y  $n$  (Siendo  $n$  el nivel del árbol)

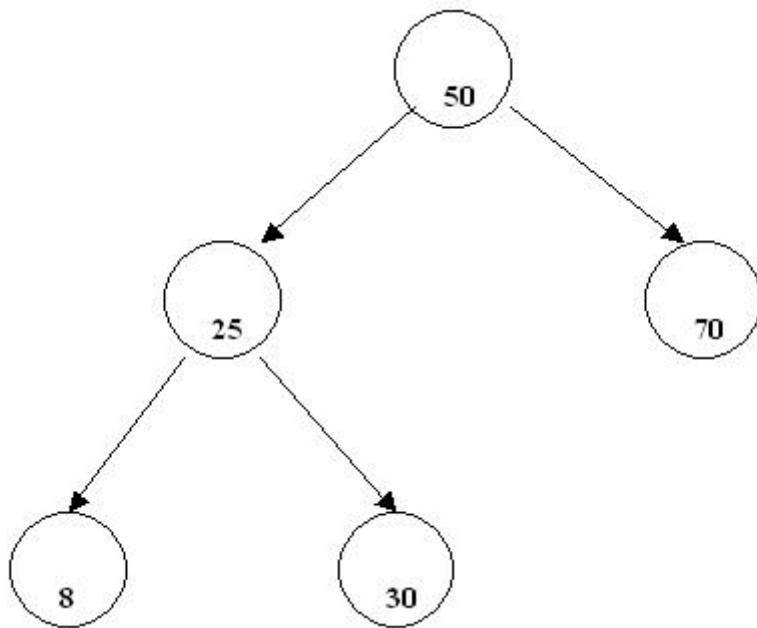
Los dos árboles graficados son completos porque son árboles de nivel 3 y hay nodos hoja en el nivel 3 en el primer caso, y hay nodos hoja en los niveles 3 y 2 en el segundo caso.

Ej. Árbol binario no completo:



Hay nodos hoja en los niveles 4, 3 y 2. No debería haber nodos hojas en el nivel 2.

**Árbol binario ordenado:** Si para cada nodo del árbol, los nodos ubicados a la izquierda son inferiores al que consideramos raíz para ese momento y los nodos ubicados a la derecha son mayores que la raíz.



Ej. Analicemos si se trata de un árbol binario ordenado:

Para el nodo que tiene el 50:

Los nodos del subárbol izquierdo son todos menores a 50? 8, 25, 30 Si

Los nodos del subárbol derecho son todos mayores a 50? 70 Si.

Para el nodo que tiene el 25:

Los nodos del subárbol izquierdo son todos menores a 25? 8 Si

Los nodos del subárbol derecho son todos mayores a 25? 30 Si.

No hace falta analizar los nodos hoja. Si todas las respuestas son afirmativas podemos luego decir que se trata de un árbol binario ordenado.

[Retornar](#)

## 53 - Estructuras dinámicas:

[Listado completo de tutoriales](#)

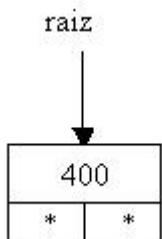
## Inserción de nodos y recorrido de un árbol binario

Para administrar un árbol binario ordenado debemos tener especial cuidado en la inserción.

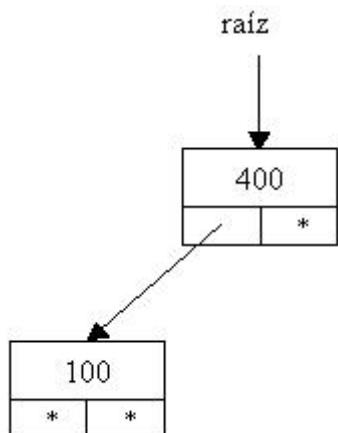
Inicialmente el árbol está vacío, es decir raíz apunta a null:



Insertamos el 400

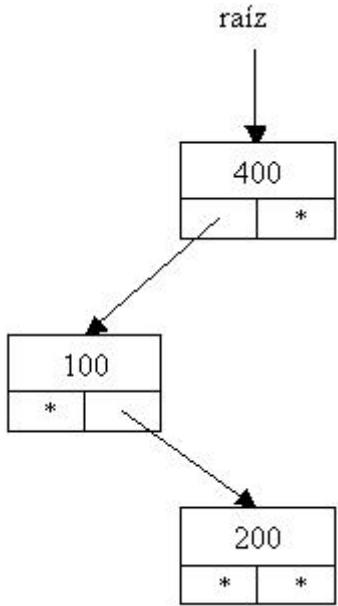


Insertamos el valor 100. Debemos analizar si raíz es distinto a null verificamos si 100 es mayor o menor a la información del nodo apuntado por raíz, en este caso es menor y como el subárbol izquierdo es null debemos insertarlo allí.

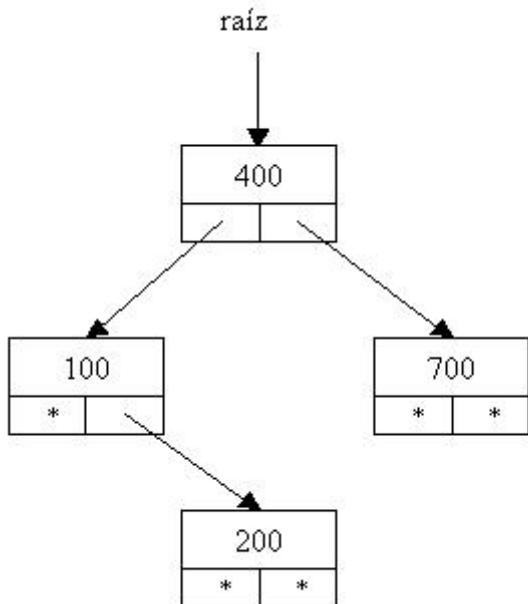


Insertamos el 200. Hay que tener en cuenta que siempre comenzamos las comparaciones a partir de raíz. El 200 es menor que 400, descendemos por el subárbol

izquierdo. Luego analizamos y vemos que el 200 es mayor a 100, debemos avanzar por derecha. Como el subárbol derecho es null lo insertamos en dicha posición.



Insertamos el 700 y el árbol será:



Como podemos observar si cada vez que insertamos un nodo respetamos este algoritmo siempre estaremos en presencia de un árbol binario ordenado. Posteriormente veremos el algoritmo en java para la inserción de información en el árbol.

## Búsqueda de información en un árbol binario ordenado.

Este es una de los principales usos de los árboles binarios.

Para realizar una búsqueda debemos ir comparando la información a buscar y descender por el subárbol izquierdo o derecho según corresponda.

Ej. Si en el árbol anterior necesitamos verificar si está almacenado el 700, primero verificamos si la información del nodo apuntado por raíz es 700, en caso negativo verificamos si la información a buscar (700) es mayor a la información de dicho nodo (400) en caso afirmativo descendemos por el subárbol derecho en caso contrario descendemos por el subárbol izquierdo.

Este proceso lo repetimos hasta encontrar la información buscada o encontrar un subárbol vacío.

## Recorridos de árboles binarios.

Recorrer: Pasar a través del árbol enumerando cada uno de sus nodos una vez.

Visitar: Realizar algún procesamiento del nodo.

Los árboles pueden ser recorridos en varios órdenes:

Pre-orden:

- Visitar la raíz.
- Recorrer el subárbol izquierdo en pre-orden.
- Recorrer el subárbol derecho en pre-orden.

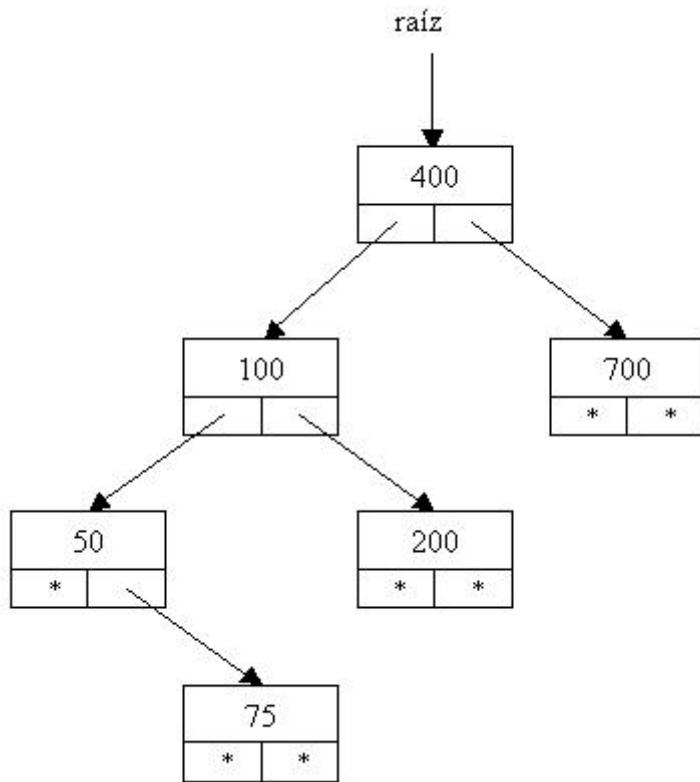
Entre-orden

- Recorrer el subárbol izquierdo en entre-orden.
- Visitar la raíz.
- Recorrer el subárbol derecho en entre-orden.

Post-orden

- Recorrer el subárbol izquierdo en post-orden.
- Recorrer el subárbol derecho en post-orden.
- Visitar la raíz.

Ejemplo:



Veamos como se imprimen las informaciones de los nodos según su recorrido:

#### Recorrido preorder:

Visitar la raiz: 400

Recorrer el subárbol izquierdo en preorder.

Visitar la raiz: 100

Recorrer el subárbol izquierdo en preorder.

Visitar la raiz: 50

Recorrer el subárbol izquierdo en preorder.

Vacio

Recorrer el subárbol derecho en preorder.

Visitar la raiz: 75

Recorrer el subárbol izquierdo en preorder.

Vacio

Recorrer el subárbol derecho en preorder.

Vacio

Recorrer el subárbol derecho en preorder.

Visitar la raiz: 200

Recorrer el subárbol izquierdo en preorder.

Vacio

Recorrer el subárbol derecho en preorder.

Vacio

Recorrer el subárbol derecho en preorder.

Visitar la raiz: 700

Recorrer el subárbol izquierdo en preorder.

Vacio

Recorrer el subárbol derecho en preorder.

Vacio

Es decir que el orden de impresión de la información es:

400 ? 100 ?50 ? 75 ?200 ? 700

Es importante analizar que el recorrido de árboles es recursivo. Recorrer un subárbol es semejante a recorrer un árbol.

Es buena práctica dibujar el árbol en un papel y hacer el seguimiento del recorrido y las visitas a cada nodo.

### **Recorrido entreorden:**

Recorrer el subárbol izquierdo en entreorden.

Recorrer el subárbol izquierdo en entreorden.

Recorrer el subárbol izquierdo en entreorden.

Vacio

Visitar la raiz: 50

Recorrer el subárbol derecho en entreorden.

Recorrer el subárbol izquierdo en entreorden.

Vacio

Visitar la raiz: 75

Recorrer el subárbol derecho en entreorden.

Vacio

Visitar la raiz: 100

Recorrer el subárbol derecho en entreorden.

Recorrer el subárbol izquierdo en entreorden.

Vacio

Visitar la raiz: 200

Recorrer el subárbol derecho en entreorden.

Vacio

Visitar la raiz: 400

Recorrer el subárbol derecho en entreorden.

Recorrer el subárbol izquierdo en entreorden.

Vacio

Visitar la raiz: 700

Recorrer el subárbol derecho en entreorden.

Vacio

Es decir que el orden de impresión de la información es:

50 ?75 ? 100 ?200 ? 400 ? 700

Si observamos podemos ver que la información aparece ordenada.

Este tipo de recorrido es muy útil cuando queremos procesar la información del árbol en orden.

### **Recorrido postorden:**

Recorrer el subárbol izquierdo en postorden.

Recorrer el subárbol izquierdo en postorden.

Recorrer el subárbol izquierdo en postorden.

Vacio

Recorrer el subárbol derecho en postorden.

Recorrer el subárbol izquierdo en postorden.

Vacio

Recorrer el subárbol derecho en postorden.

Vacio

Visitar la raiz: 75

Visitar la raiz: 50

Recorrer el subárbol derecho en postorden.

Recorrer el subárbol izquierdo en postorden.

Vacio

Recorrer el subárbol derecho en postorden.

Vacio

Visitar la raiz: 200

Visitar la raiz: 100

Recorrer el subárbol derecho en postorden.

Recorrer el subárbol izquierdo en postorden.

Vacio

Recorrer el subárbol derecho en postorden.

Visitar la raiz: 700

Visitar la raiz: 400

Es decir que el orden de impresión de la información es:

75 ? 50 ? 200 ? 100 ? 700 ? 400

**Retornar**

# 54 - Estructuras dinámicas: Implementación en Java de un árbol binario ordenado

[Listado completo de tutoriales](#)

## Problema 1:

A continuación desarrollamos una clase para la administración de un árbol binario ordenado.

## Programa:

```
public class ArbolBinarioOrdenado {  
    class Nodo  
    {  
        int info;  
        Nodo izq, der;  
    }  
    Nodo raiz;  
  
    public ArbolBinarioOrdenado() {  
        raiz=null;  
    }  
  
    public void insertar (int info)  
    {  
        Nodo nuevo;  
        nuevo = new Nodo ();  
        nuevo.info = info;  
        nuevo.izq = null;  
        nuevo.der = null;  
        if (raiz == null)  
            raiz = nuevo;  
        else  
        {  
            Nodo anterior = null, reco;  
            reco = raiz;  
            while (reco != null)  
            {  
                anterior = reco;  
                if (info < reco.info)  
                    reco = reco.izq;  
                else  
                    reco = reco.der;  
            }  
            if (info < anterior.info)  
                anterior.izq = nuevo;  
            else  
                anterior.der = nuevo;  
        }  
    }  
}
```

```
}

private void imprimirPre (Nodo reco)
{
    if (reco != null)
    {
        System.out.print(reco.info + " ");
        imprimirPre (reco.izq);
        imprimirPre (reco.der);
    }
}

public void imprimirPre ()
{
    imprimirPre (raiz);
    System.out.println();
}

private void imprimirEntre (Nodo reco)
{
    if (reco != null)
    {
        imprimirEntre (reco.izq);
        System.out.print(reco.info + " ");
        imprimirEntre (reco.der);
    }
}

public void imprimirEntre ()
{
    imprimirEntre (raiz);
    System.out.println();
}

private void imprimirPost (Nodo reco)
{
    if (reco != null)
    {
        imprimirPost (reco.izq);
        imprimirPost (reco.der);
        System.out.print(reco.info + " ");
    }
}

public void imprimirPost ()
{
    imprimirPost (raiz);
    System.out.println();
}

public static void main (String [] ar)
{
    ArbolBinarioOrdenado abo = new ArbolBinarioOrdenado ();
    abo.insertar (100);
    abo.insertar (50);
    abo.insertar (25);
    abo.insertar (75);
    abo.insertar (150);
```

```

        System.out.println ("Impresion preorden: ");
        abo.imprimirPre ();
        System.out.println ("Impresion entreorden: ");
        abo.imprimirEntre ();
        System.out.println ("Impresion postorden: ");
        abo.imprimirPost ();
    }
}

public void insertar (int info)
{
    Nodo nuevo;
    nuevo = new Nodo ();
    nuevo.info = info;
    nuevo.izq = null;
    nuevo.der = null;
    if (raiz == null)
        raiz = nuevo;
    else
    {
        Nodo anterior = null, reco;
        reco = raiz;
        while (reco != null)
        {
            anterior = reco;
            if (info < reco.info)
                reco = reco.izq;
            else
                reco = reco.der;
        }
        if (info < anterior.info)
            anterior.izq = nuevo;
        else
            anterior.der = nuevo;
    }
}

```

Creamos un nodo y disponemos los punteros izq y der a null, guardamos la información que llega al método en el nodo.

Si el árbol está vacío, apuntamos raíz al nodo creado; en caso de no estar vacío, dentro de una estructura repetitiva vamos comparando info con la información del nodo, si info es mayor a la del nodo descendemos por el subárbol derecho en caso contrario descendemos por el subárbol izquierdo.

Cuando se encuentra un subárbol vacío insertar el nodo en dicho subárbol. Para esto llevamos un puntero anterior dentro del while.

```

private void imprimirPre (Nodo reco)
{
    if (reco != null)
    {
        System.out.print(reco.info + " ");
        imprimirPre (reco.izq);
        imprimirPre (reco.der);
    }
}

```

```

public void imprimirPre ()
{
    imprimirPre (raiz);
    System.out.println();
}

```

El método imprimirPre(), es decir el no recursivo se encarga de llamar al método recursivo pasando la dirección del nodo raiz.

El método recursivo void imprimirPre (Nodo reco) lo primero que verifica con un if si reco está apuntando a un nodo (esto es verdad si reco es distinto a null), en caso afirmativo ingresa al bloque del if y realiza:

- Visitar la raiz.
- Recorrer el subárbol izquierdo en pre-orden.
- Recorrer el subárbol derecho en pre-orden.

La visita en este caso es la impresión de la información del nodo y los recorridos son las llamadas recursivas pasando las direcciones de los subárboles izquierdo y derecho.

Los algoritmos de los recorridos en entreorden y postorden son similares. La diferencia es que la visita la realizamos entre las llamadas recursivas en el recorrido en entre orden:

```

private void imprimirEntre (Nodo reco)
{
    if (reco != null)
    {
        imprimirEntre (reco.izq);
        System.out.print(reco.info + " ");
        imprimirEntre (reco.der);
    }
}

```

y por último en el recorrido en postorden la visita la realizamos luego de las dos llamadas recursivas:

```

private void imprimirPost (Nodo reco)
{
    if (reco != null)
    {
        imprimirPost (reco.izq);
        imprimirPost (reco.der);
        System.out.print(reco.info + " ");
    }
}

```

## Problema 2:

Confeccionar una clase que permita insertar un entero en un árbol binario ordenado verificando que no se encuentre previamente dicho número.

Desarrollar los siguientes métodos:

- 1 - Retornar la cantidad de nodos del árbol.
- 2 - Retornar la cantidad de nodos hoja del árbol.
- 3 - Imprimir en entre orden.
- 4 - Imprimir en entre orden junto al nivel donde se encuentra dicho nodo.
- 5 - Retornar la altura del árbol.
- 6 - Imprimir el mayor valor del árbol.
- 7 - Borrar el nodo menor del árbol.

```
public class ArbolBinarioOrdenado {  
    class Nodo  
    {  
        int info;  
        Nodo izq, der;  
    }  
    Nodo raiz;  
    int cant;  
    int altura;  
  
    public ArbolBinarioOrdenado() {  
        raiz=null;  
    }  
  
    public void insertar (int info) {  
        if (!existe(info)) {  
            Nodo nuevo;  
            nuevo = new Nodo ();  
            nuevo.info = info;  
            nuevo.izq = null;  
            nuevo.der = null;  
            if (raiz == null)  
                raiz = nuevo;  
            else {  
                Nodo anterior = null, reco;  
                reco = raiz;  
                while (reco != null) {  
                    anterior = reco;  
                    if (info < reco.info)  
                        reco = reco.izq;  
                    else  
                        reco = reco.der;  
                }  
                if (info < anterior.info)  
                    anterior.izq = nuevo;  
                else  
                    anterior.der = nuevo;  
            }  
        }  
    }  
  
    public boolean existe(int info) {  
        Nodo reco=raiz;  
        while (reco!=null) {
```

```

        while (reco!=null) {
            if (info==reco.info)
                return true;
            else
                if (info>reco.info)
                    reco=reco.der;
                else
                    reco=reco.izq;
        }
        return false;
    }

private void imprimirEntre (Nodo reco) {
    if (reco != null) {
        imprimirEntre (reco.izq);
        System.out.print(reco.info + " ");
        imprimirEntre (reco.der);
    }
}

public void imprimirEntre () {
    imprimirEntre (raiz);
    System.out.println();
}

private void cantidad(Nodo reco) {
    if (reco!=null) {
        cant++;
        cantidad(reco.izq);
        cantidad(reco.der);
    }
}

public int cantidad() {
    cant=0;
    cantidad(raiz);
    return cant;
}

private void cantidadNodosHoja(Nodo reco) {
    if (reco!=null) {
        if (reco.izq==null && reco.der==null)
            cant++;
        cantidadNodosHoja(reco.izq);
        cantidadNodosHoja(reco.der);
    }
}

public int cantidadNodosHoja() {
    cant=0;
    cantidadNodosHoja(raiz);
    return cant;
}

private void imprimirEntreConNivel (Nodo reco,int nivel) {
    if (reco != null) {
        imprimirEntreConNivel (reco.izq,nivel+1);
        System.out.print(reco.info + " ("+nivel+") - ");
        imprimirEntreConNivel (reco.der,nivel+1);
    }
}

```

```

}

public void imprimirEntreConNivel () {
    imprimirEntreConNivel (raiz,1);
    System.out.println();
}

private void retornarAltura (Nodo reco,int nivel)      {
    if (reco != null) {
        retornarAltura (reco.izq,nivel+1);
        if (nivel>altura)
            altura=nivel;
        retornarAltura (reco.der,nivel+1);
    }
}

public  int retornarAltura () {
    altura=0;
    retornarAltura (raiz,1);
    return altura;
}

public void mayorValorl() {
    if (raiz!=null) {
        Nodo reco=raiz;
        while (reco.der!=null)
            reco=reco.der;
        System.out.println("Mayor valor del árbol:"+reco.info);
    }
}

public void borrarMenor() {
    if (raiz!=null) {
        if (raiz.izq==null)
            raiz=raiz.der;
        else {
            Nodo atras=raiz;
            Nodo reco=raiz.izq;
            while (reco.izq!=null) {
                atras=reco;
                reco=reco.izq;
            }
            atras.izq=reco.der;
        }
    }
}

public static void main (String [] ar)
{
    ArbolBinarioOrdenado abo = new ArbolBinarioOrdenado ();
    abo.insertar (100);
    abo.insertar (50);
    abo.insertar (25);
    abo.insertar (75);
    abo.insertar (150);
    System.out.println ("Impresion entreorden: ");
    abo.imprimirEntre ();
    System.out.println ("Cantidad de nodos del árbol:"+abo.cantidad());
    System.out.println ("Cantidad de nodos
hoja:"+abo.cantidadNodosHoja());
    System.out.println ("Impresion en entre orden junto al nivel del
nodo `..");
}

```

```

        nroos. );
        abo.imprimirEntreConNivel();
        System.out.print ("Altura del arbol:");
        System.out.println(abo.retornarAltura());
        abo.mayorValor();
        abo.borrarMenor();
        System.out.println("Luego de borrar el menor:");
        abo.imprimirEntre ();
    }
}

```

Para verificar si existe un elemento de información en el árbol disponemos un puntero reco en el nodo apuntado por raiz. Dentro de un while verificamos si la información del parámetro coincide con la información del nodo apuntado por reco, en caso afirmativo salimos del método retornando true, en caso contrario si la información a buscar es mayor a la del nodo procedemos a avanzar reco con la dirección del subárbol derecho:

```

public boolean existe(int info) {
    Nodo reco=raiz;
    while (reco!=null) {
        if (info==reco.info)
            return true;
        else
            if (info>reco.info)
                reco=reco.der;
            else
                reco=reco.izq;
    }
    return false;
}

```

Para retornar la cantidad de nodos del árbol procedemos a inicializar un atributo de la clase llamado cant con cero. Llamamos al método recursivo y en cada visita al nodo incrementamos el atributo cant en uno:

```

private void cantidad(Nodo reco) {
    if (reco!=null) {
        cant++;
        cantidad(reco.izq);
        cantidad(reco.der);
    }
}

public int cantidad() {
    cant=0;
    cantidad(raiz);
    return cant;
}

```

Para imprimir todos los nodos en orden entre junto al nivel donde se encuentra planteamos un método recursivo que llegue la referencia del nodo a imprimir junto al nivel de dicho nodo. Desde el método no recursivo pasamos la referencia a raiz y un

uno (ya que raiz se encuentra en el primer nivel)

Cada vez que descendemos un nivel le pasamos la referencia del subárbol respectivo junto al nivel que se encuentra dicho nodo:

```
private void imprimirEntreConNivel (Nodo reco,int nivel)  {
    if (reco != null) {
        imprimirEntreConNivel (reco.izq,nivel+1);
        System.out.print(reco.info + " ("+nivel+") - ");
        imprimirEntreConNivel (reco.der,nivel+1);
    }
}

public void imprimirEntreConNivel () {
    imprimirEntreConNivel (raiz,1);
    System.out.println();
}
```

Para obtener la altura del árbol procedemos en el método no recursivo a inicializar el atributo altura con el valor cero. Luego llamamos al método recursivo con la referencia a raiz que se encuentra en el nivel uno. Cada vez que visitamos un nodo procedemos a verificar si el parámetro nivel supera al atributo altura, en dicho caso actualizamos el atributo altura con dicho nivel.

```
private void retornarAltura (Nodo reco,int nivel)      {
    if (reco != null) {
        retornarAltura (reco.izq,nivel+1);
        if (nivel>altura)
            altura=nivel;
        retornarAltura (reco.der,nivel+1);
    }
}

public  int retornarAltura () {
    altura=0;
    retornarAltura (raiz,1);
    return altura;
}
```

Para imprimir el mayor valor del árbol debemos recorrer siempre por derecha hasta encontrar un nodo que almacene null en der:

```
public void mayorValor1() {
    if (raiz!=null) {
        Nodo reco=raiz;
        while (reco.der!=null)
            reco=reco.der;
        System.out.println("Mayor valor del árbol:"+reco.info);
    }
}
```

Para borrar el menor valor del árbol lo primero que comprobamos es si el subárbol izquierdo es nulo luego el menor del árbol es el nodo apuntado por raiz. Luego si el subárbol izquierdo no está vacío procedemos a descender siempre por la izquierda llevando un puntero en el nodo anterior. Cuando llegamos al nodo que debemos borrar procedemos a enlazar el puntero izq del nodo que se encuentra en el nivel anterior con la referencia del subárbol derecho del nodo a borrar:

```
public void borrarMenor() {  
    if (raiz!=null) {  
        if (raiz.izq==null)  
            raiz=raiz.der;  
        else {  
            Nodo atras=raiz;  
            Nodo reco=raiz.izq;  
            while (reco.izq!=null) {  
                atras=reco;  
                reco=reco.izq;  
            }  
            atras.izq=reco.der;  
        }  
    }  
}
```

[Retornar](#)

# 56 - Plug-in WindowBuilder para crear interfaces visuales.

[Listado completo de tutoriales](#)

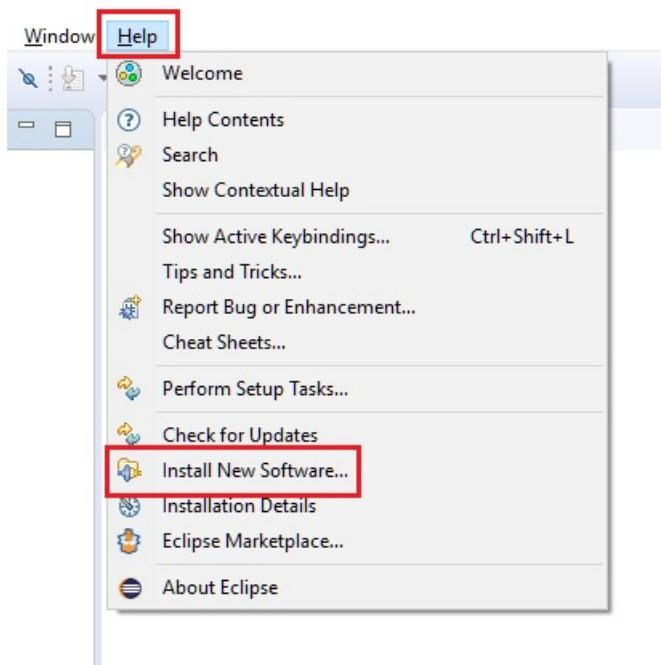
El objetivo de este concepto es conocer el empleo del plug-in WindowBuilder para el desarrollo de interfaces visuales arrastrando componentes.

Desde la versión 3.7 de Eclipse llamada Indigo(2011) hasta la versión Mars(2015) se incorpora por defecto el plug-in WindowBuilder para la implementación de interfaces visuales.

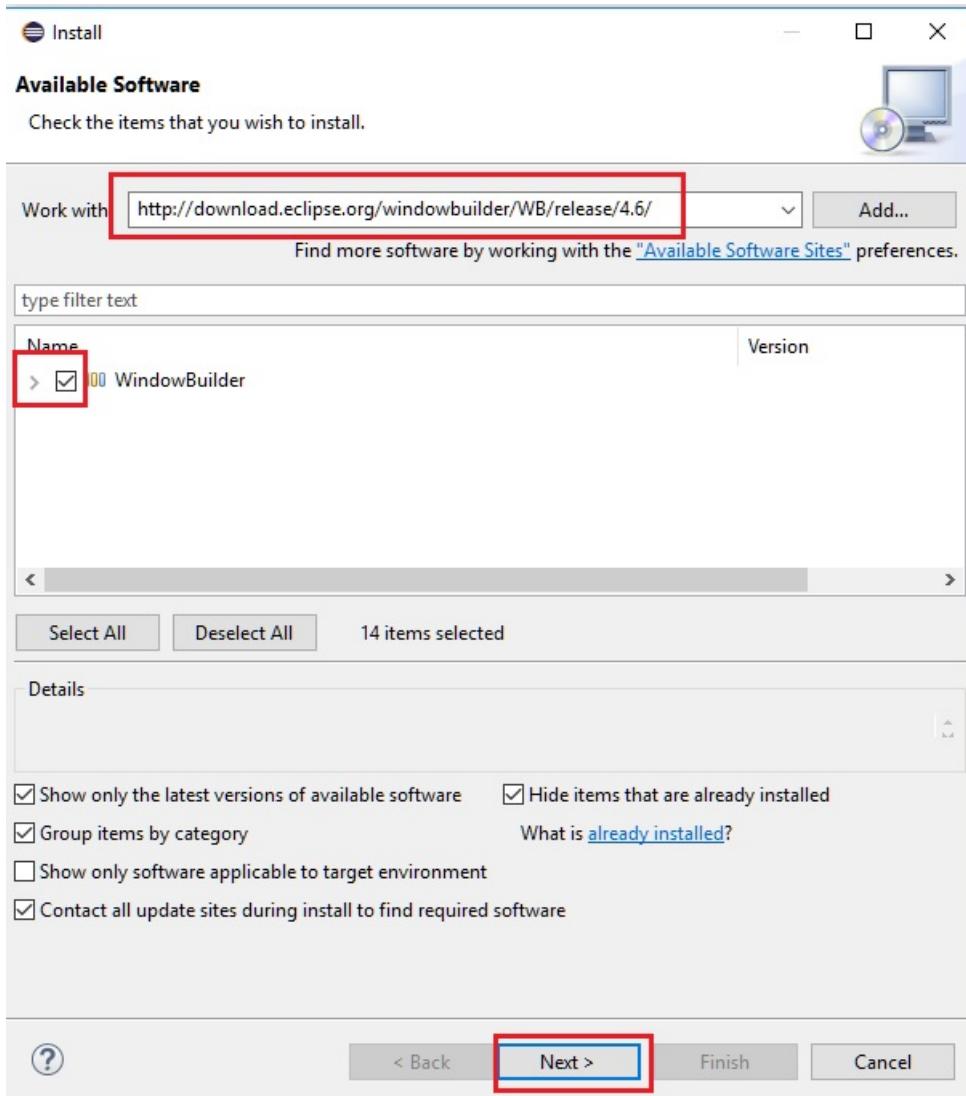
La versión Neon y Oxygen no trae instalado el plug-in WindowBuilder por defecto.

## Instalación del plug-in WindowBuilder en la versión Neon u Oxygen.

Debemos seleccionar desde el menú de opciones de Eclipse Help->Install New Software..:



En el diálogo siguiente procedemos a copiar la siguiente URL  
(<http://download.eclipse.org/windowbuilder/WB/release/4.6/>) de donde se descargará el WindowBuilder:



Presionamos "Next" y aceptamos los términos y condiciones en los siguientes diálogos.

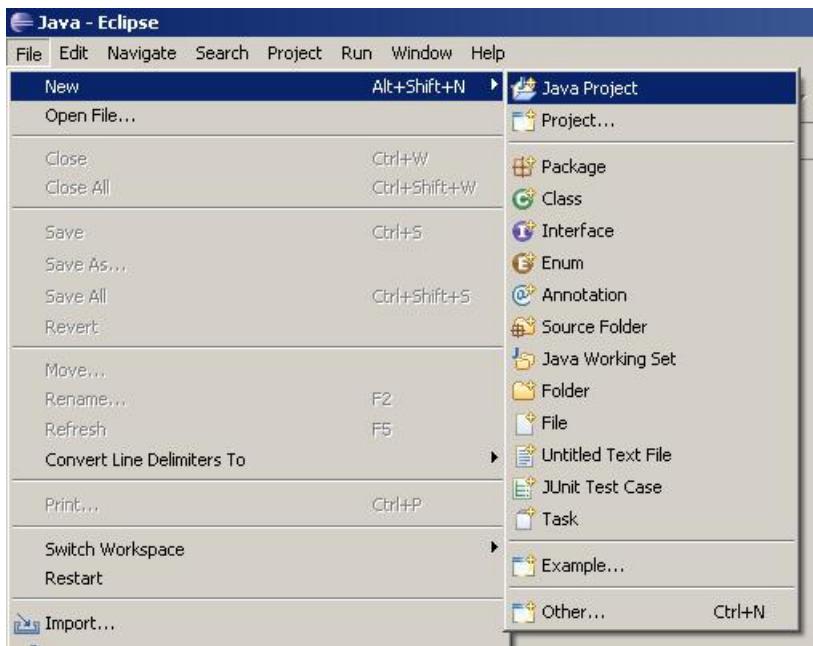
Con esto ya tenemos instalado el WindowBuilder para trabajar en los siguientes conceptos.

## Uso del WindowBuilder

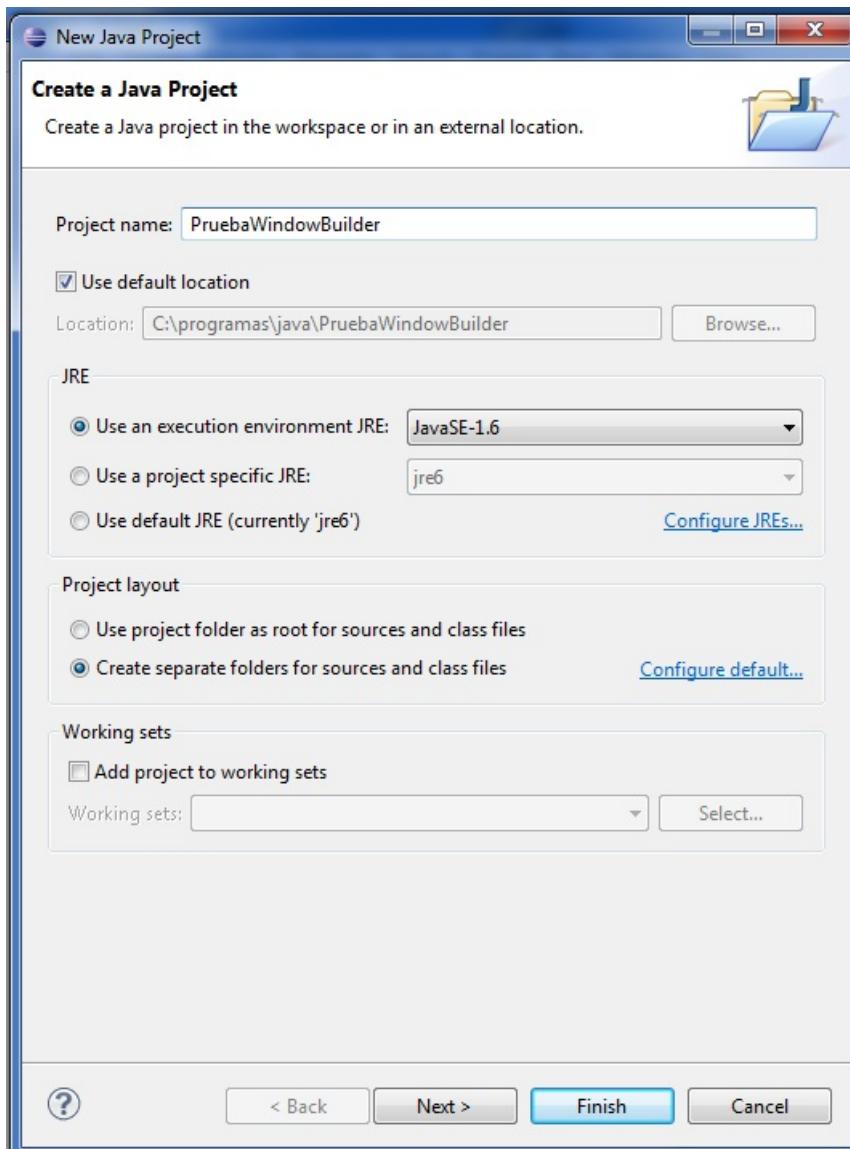
A medida que uno arrastra componentes visuales sobre un formulario se genera en forma automática el código Java, esto nos permite ser más productivos en el desarrollo de la interfaz de nuestra aplicación y nos ayuda a concentrarnos en la lógica de nuestro problema.

### Pasos para crear un JFrame con el WindowBuilder

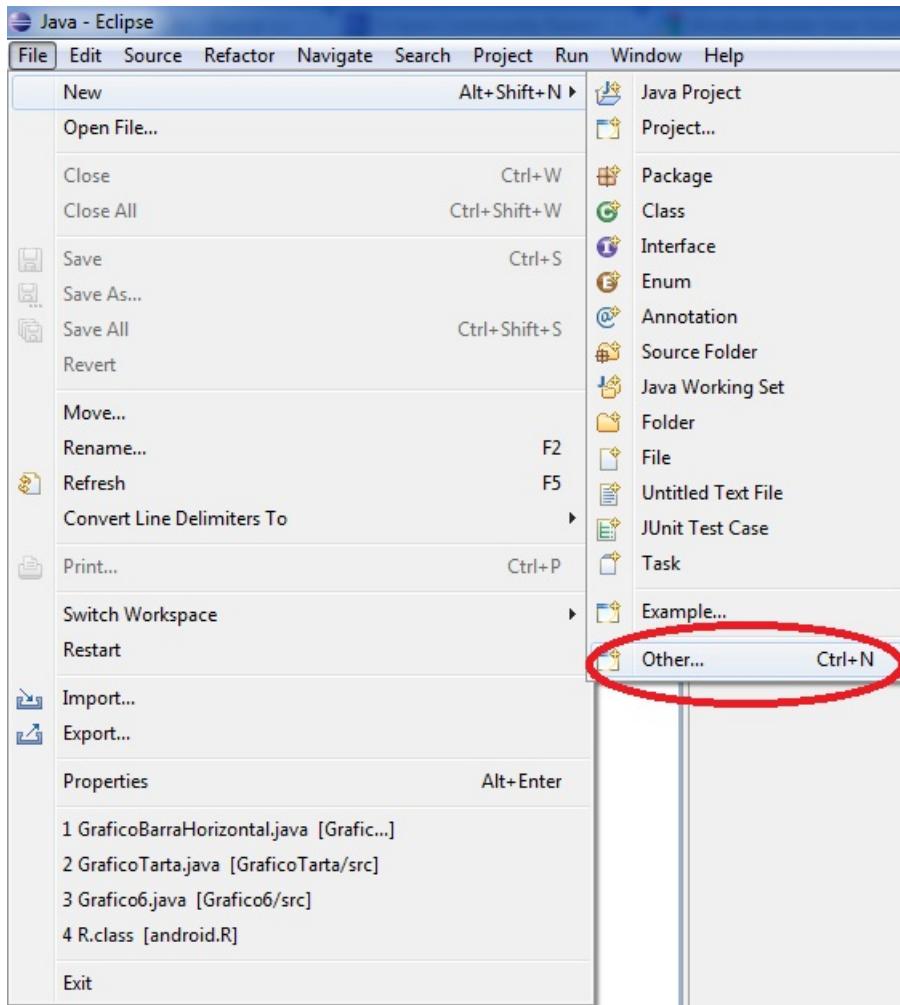
1 - Creación de un proyecto.



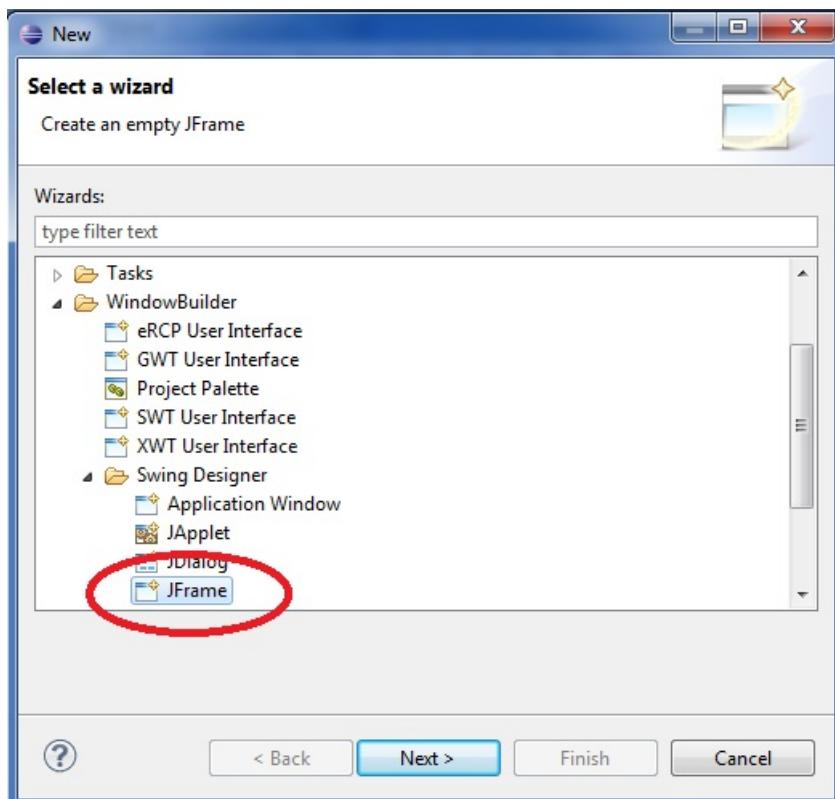
2 - Seleccionamos el nombre de nuestro proyecto (lo llamaremos PruebaWindowBuilder):



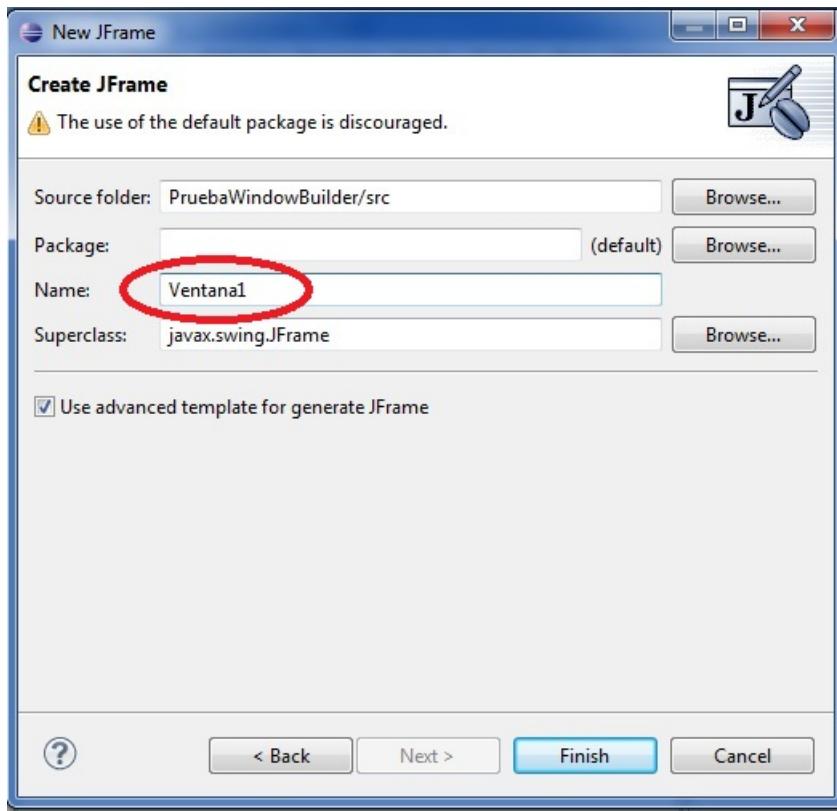
3 - Ahora seleccionamos la opción del menú File -> New -> Other ...



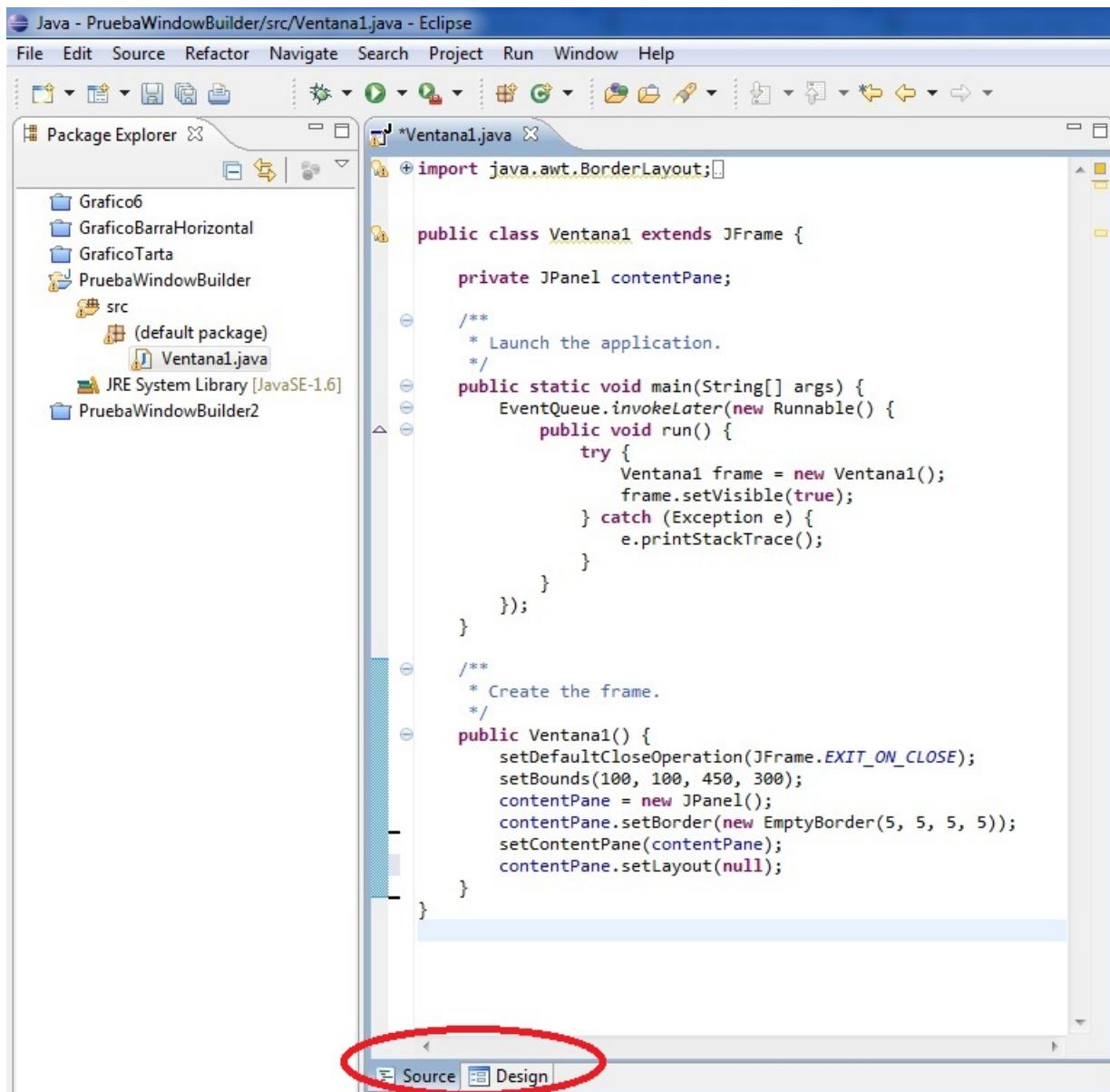
4 - Seleccionamos la opción la opción JFrame:



5 - Seguidamente presionamos el botón Next > y definimos el nombre de la clase a crear (Ventana1):



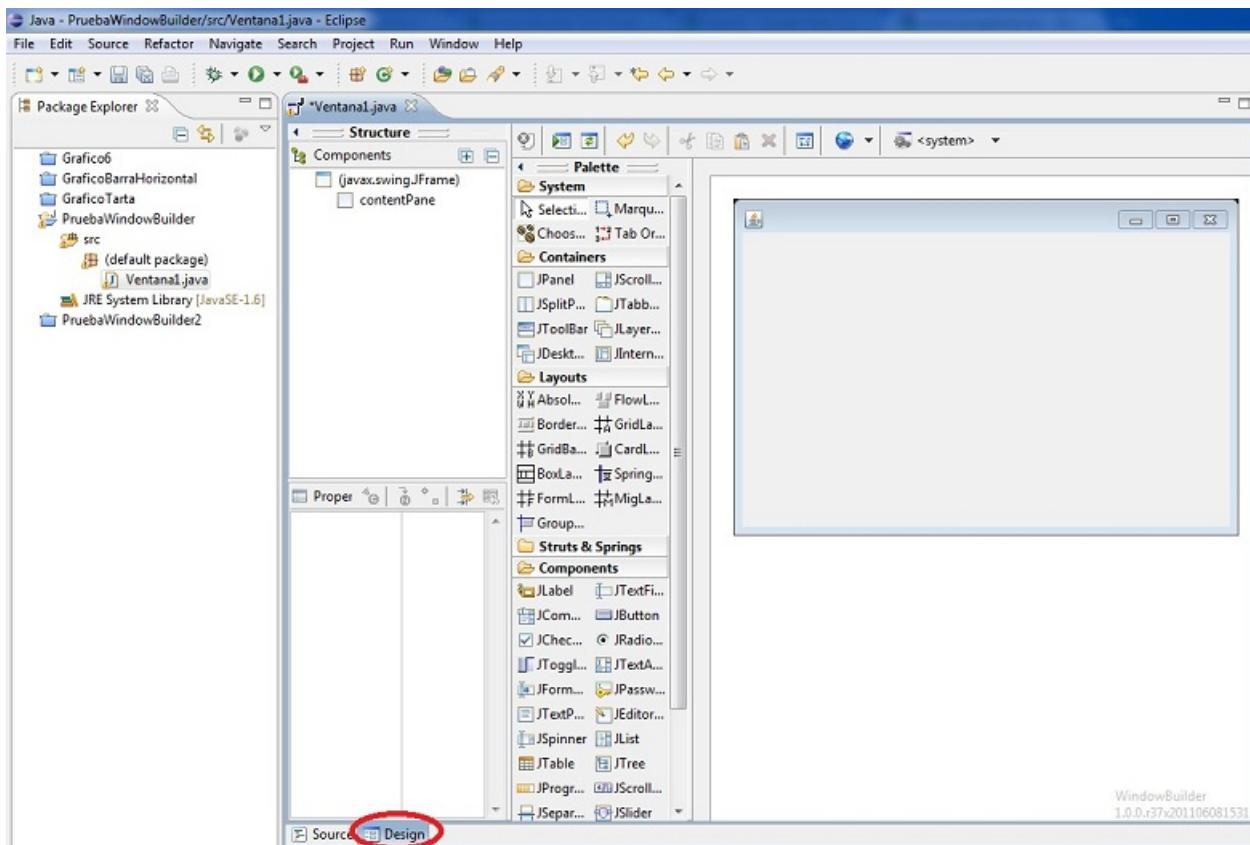
Tenemos en este momento nuestra aplicación mínima generada por el WindowBuilder. Podemos observar que en la parte inferior de la ventana central aparecen dos pestañas (Source y Design) estas dos pestañas nos permiten ver el código fuente de nuestro JFrame en vista de diseño o en vista de código Java:



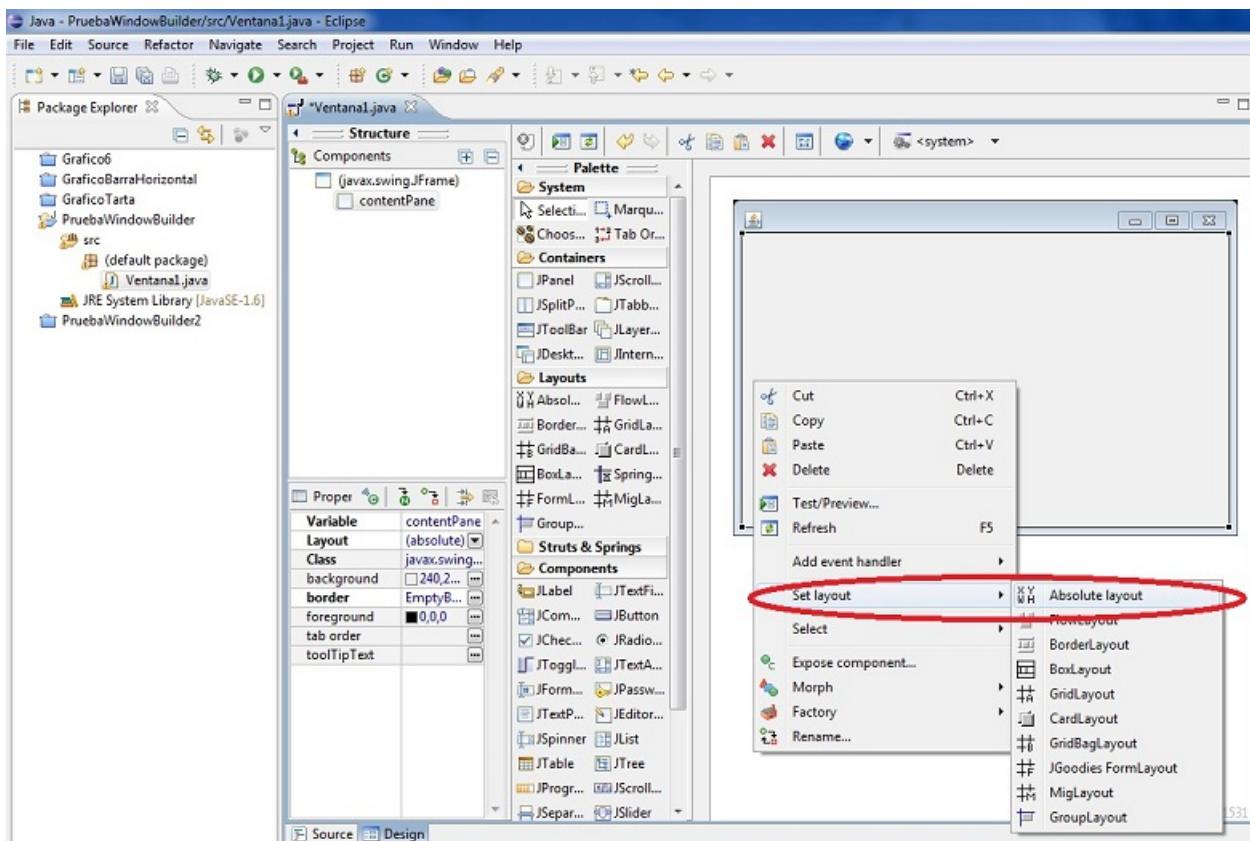
The screenshot shows the Eclipse IDE interface with the title bar "Java - PruebaWindowBuilder/src/Ventana1.java - Eclipse". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations. The "Package Explorer" view on the left shows a project structure with packages like "Grafico6", "GraficoBarraHorizontal", "GraficoTarta", "PruebaWindowBuilder" (which contains a "src" folder with "Ventana1.java"), and "PruebaWindowBuilder2". The "JRE System Library [JavaSE-1.6]" is also listed. The main editor view on the right displays the Java code for "Ventana1.java". The code defines a class "Ventana1" that extends "JFrame". It includes a main method that creates and displays the frame. The constructor "Ventana1()" initializes the frame with a border layout, sets its bounds, and configures the content pane. The code ends with a closing brace for the class definition. At the bottom of the editor, there are tabs for "Source" and "Design", with "Source" currently selected. A red circle highlights the "Design" tab.

```
import java.awt.BorderLayout;
public class Ventana1 extends JFrame {
    private JPanel contentPane;
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Ventana1 frame = new Ventana1();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    public Ventana1() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);
    }
}
```

Luego en vista de "Design":

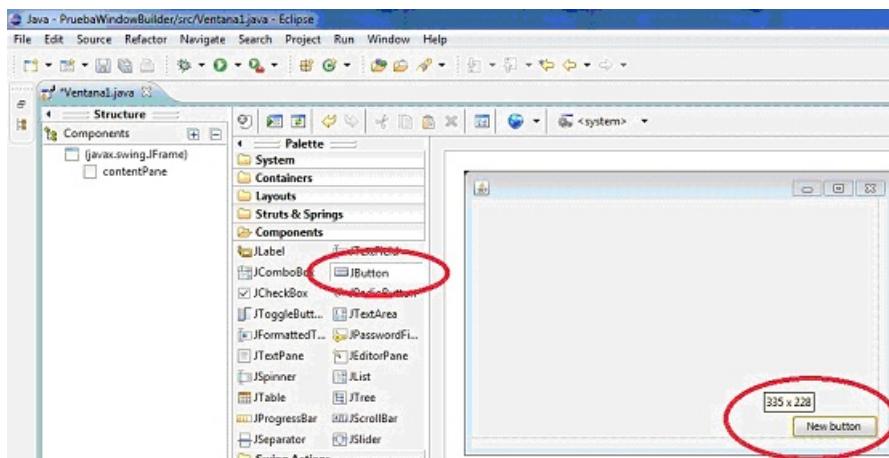


6 - Configuramos el Layout de JFrame presionando el botón derecho del mouse sobre el formulario generado y seleccionamos la opción SetLayout > Absolute layout (esto nos permite luego disponer controles visuales como JButton, JLabel etc. en posiciones fijas dentro del JFrame):



7 - De la ventana Palette seleccionamos con el mouse un objeto de la clase JButton (presionamos con el mouse dicha componente, deberá aparecer seleccionada) y luego nos desplazamos con el mouse sobre el JFrame y

presionamos el botón del mouse nuevamente ( en este momento aparece el botón dentro del JFrame):



En todo momento podemos cambiar la pestaña de "Source" y "Design" para ver el código generado. Por ejemplo cuando agregamos el botón podemos ver que se agregó un objeto de la clase JButton al constructor:

```
import java.awt.BorderLayout;
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;

public class Ventana1 extends JFrame {

    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Ventana1 frame = new Ventana1();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

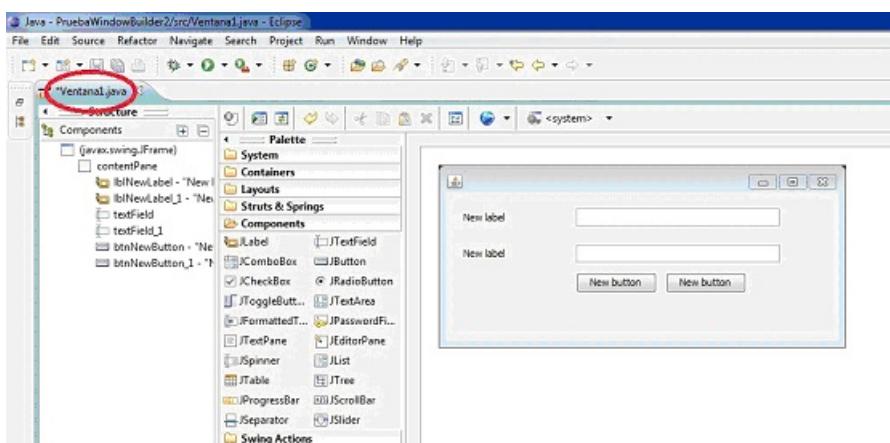
    /**
     * Create the frame.
     */
    public Ventana1() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JButton btnNewButton = new JButton("New button");
        btnNewButton.setBounds(335, 228, 89, 23);
        contentPane.add(btnNewButton);
    }
}
```

**Iniciarizar propiedades de los objetos.**

Crear un proyecto y luego un JFrame con las siguientes componentes visuales :

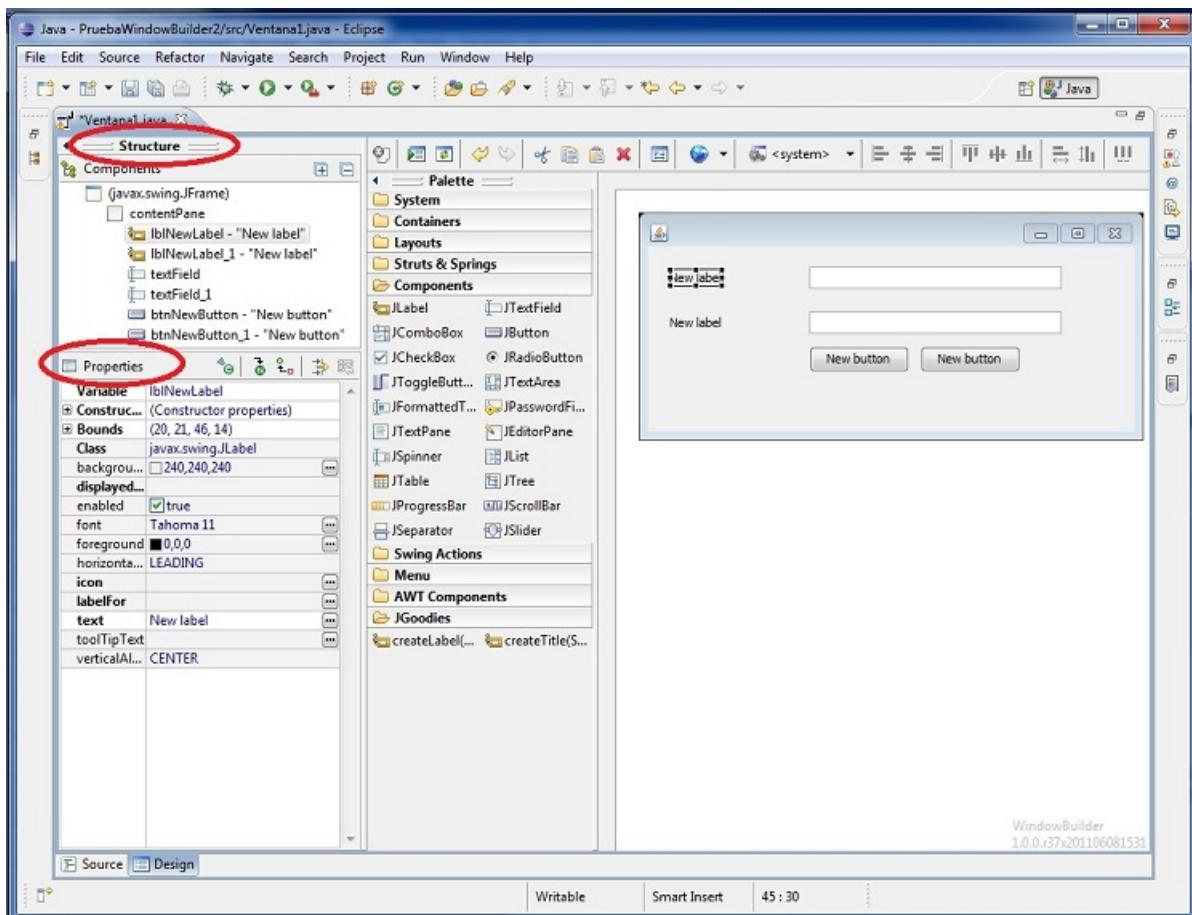
Dos controles de tipo JLabel, dos JTextField y dos JButton (previamente definir el layout para el panel contenido en el JFrame de tipo Absolute Layout)



Si hacemos doble clic con el mouse en la pestaña Ventana1.java podemos maximizar el espacio de nuestro editor visual (haciendo nuevamente doble clic vuelve al tamaño anterior)

Seleccionemos el primer JLabel de nuestro formulario y observemos las distintas partes que componen el plug-in WindowBuilder. En la parte superior izquierda se encuentra la sección "Structure" donde se muestran las componentes visuales agregadas al formulario. Aparece resaltada la que se encuentra actualmente seleccionada.

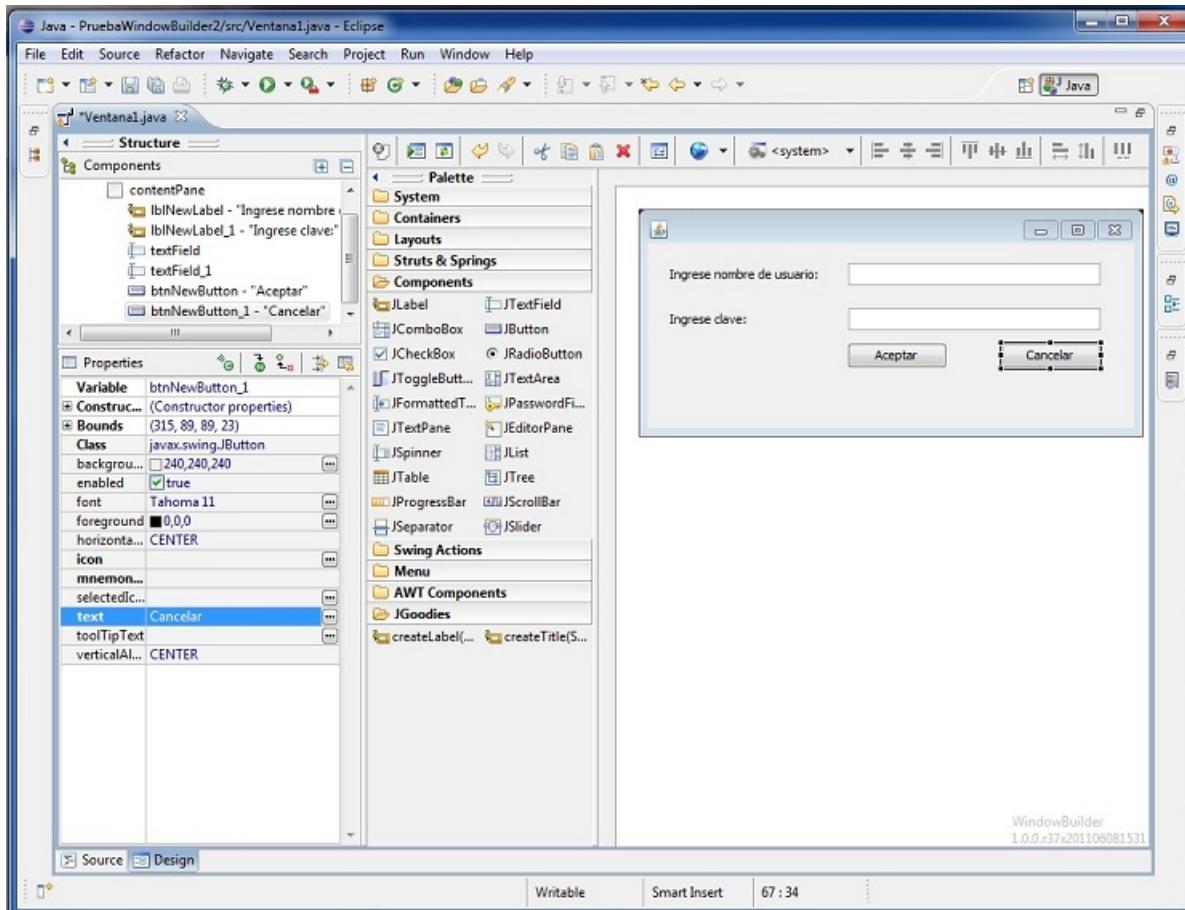
En la parte inferior aparece la ventana de "Properties" o propiedades del control visual



Veamos algunas propiedades que podemos modificar desde esta ventana y los cambios que se producen en el código fuente en java.

La propiedad text cambia el texto que muestra el objeto JLabel. Probemos de disponer el texto "Ingrese nombre

de usuario:". De forma similar hagamos los cambios en la propiedad text de los otros controles visuales de nuestro JFrame:



Si ahora seleccionamos la pestaña inferior para ver la vista de código java: "Source" podemos ver que el WindowBuilder nos generó automáticamente el código para inicializar los textos de los controles JLabel y JButton:

```
import java.awt.BorderLayout;
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;

public class Ventanal extends JFrame {

    private JPanel contentPane;
    private JTextField textField;
    private JTextField textField_1;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Ventanal frame = new Ventanal();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```

        }
    });

/** 
 * Create the frame.
 */
public Ventana1() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 203);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JLabel lblNewLabel = new JLabel("Ingrese nombre de usuario:");
    lblNewLabel.setBounds(20, 21, 149, 14);
    contentPane.add(lblNewLabel);

    JLabel lblNewLabel_1 = new JLabel("Ingrese clave:");
    lblNewLabel_1.setBounds(20, 61, 125, 14);
    contentPane.add(lblNewLabel_1);

    textField = new JTextField();
    textField.setBounds(179, 18, 225, 20);
    contentPane.add(textField);
    textField.setColumns(10);

    textField_1 = new JTextField();
    textField_1.setBounds(179, 58, 225, 20);
    contentPane.add(textField_1);
    textField_1.setColumns(10);

    JButton btnNewButton = new JButton("Aceptar");
    btnNewButton.setBounds(178, 89, 89, 23);
    contentPane.add(btnNewButton);

    JButton btnNewButton_1 = new JButton("Cancelar");
    btnNewButton_1.setBounds(315, 89, 89, 23);
    contentPane.add(btnNewButton_1);
}
}
}

```

Como podemos observar ahora cuando se crean los objetos de la clase JLabel en el constructor se inicializan con los valores cargados en la propiedad text:

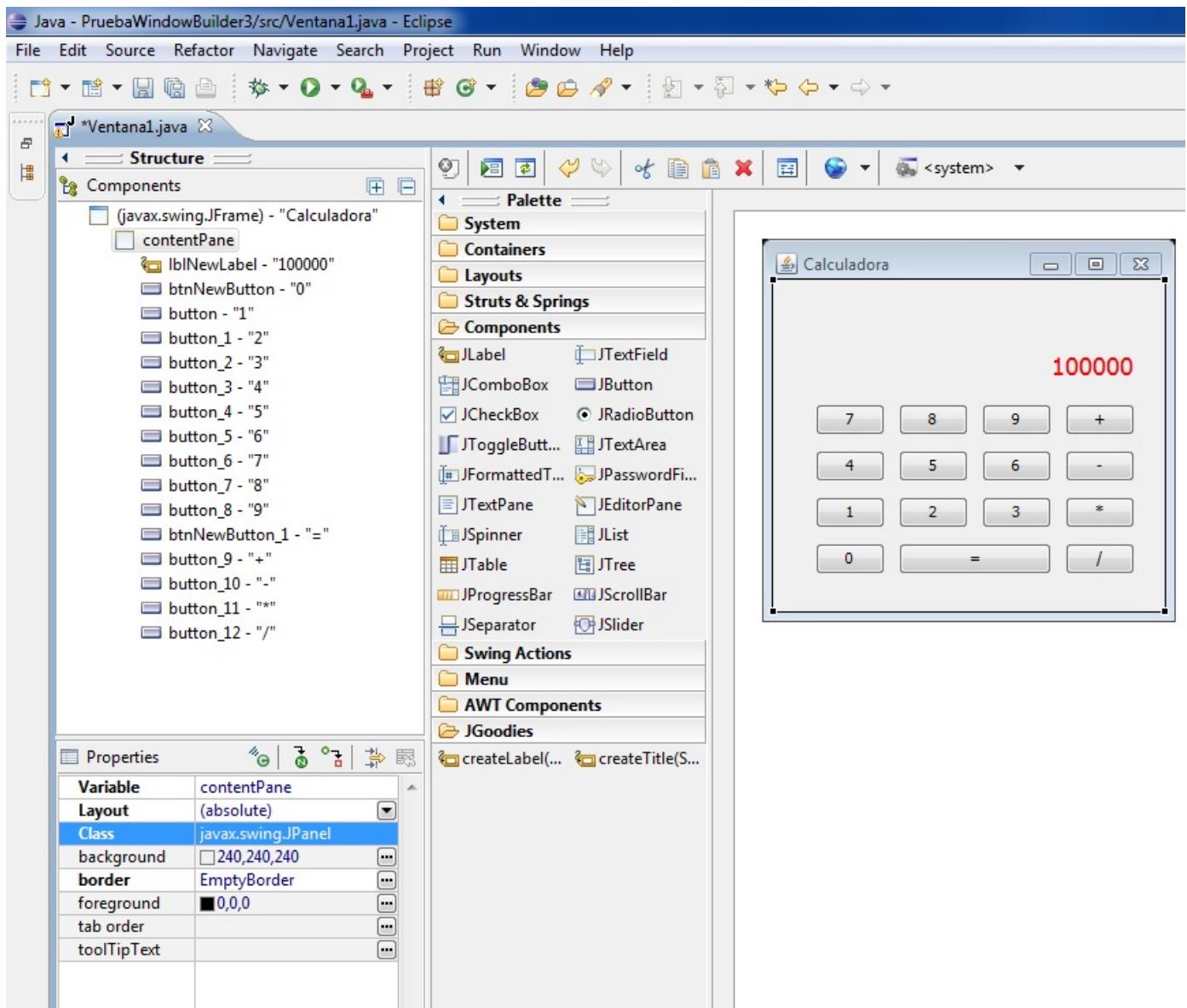
```

JLabel lblNewLabel = new JLabel("Ingrese nombre de usuario");
.....
JLabel lblNewLabel_1 = new JLabel("Ingrese clave:");
.....
JButton btnNewButton = new JButton("Aceptar");
.....
JButton btnNewButton_1 = new JButton("Cancelar");
.....

```

## Problema propuesto 1

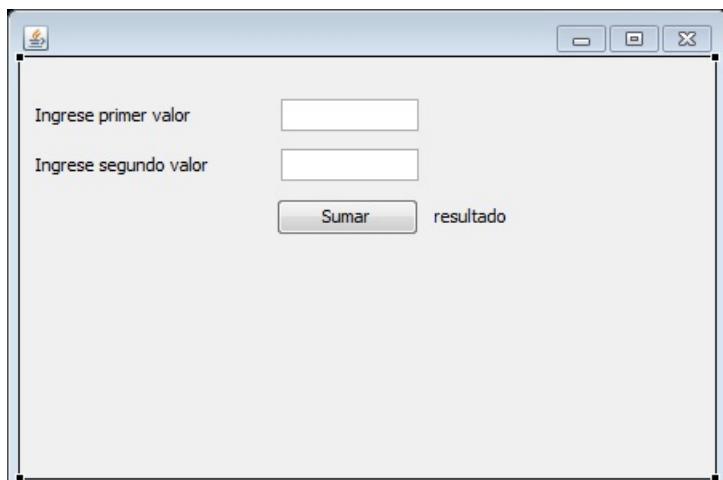
Crear la interfaz visual que aparece abajo. Inicializar las propiedades que corresponde.



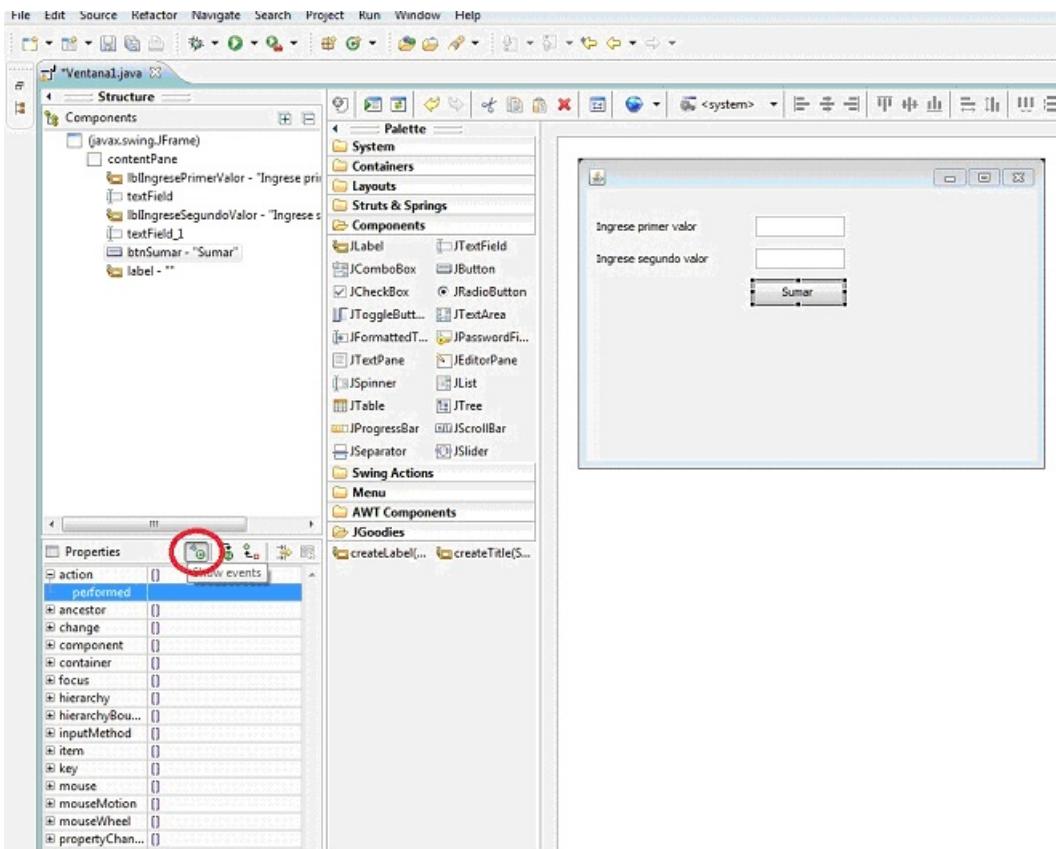
## Eventos

Para asociar eventos el plug-in WindowBuilder nos proporciona una mecánica para automatizar la generación de las interfaces que capturan los eventos de los objetos JButton, JMenuItem, JList etc.

Crearemos una interfaz visual similar a esta (tres controles de tipo JLabel, dos JTextField y un JButton):



Ahora seleccionamos el control JButton y en la ventana de propiedades presionamos el icono de la parte superior:



Hacemos doble clic sobre la palabra performed y vemos que se abre el editor de texto y aparece el siguiente código generado automáticamente:

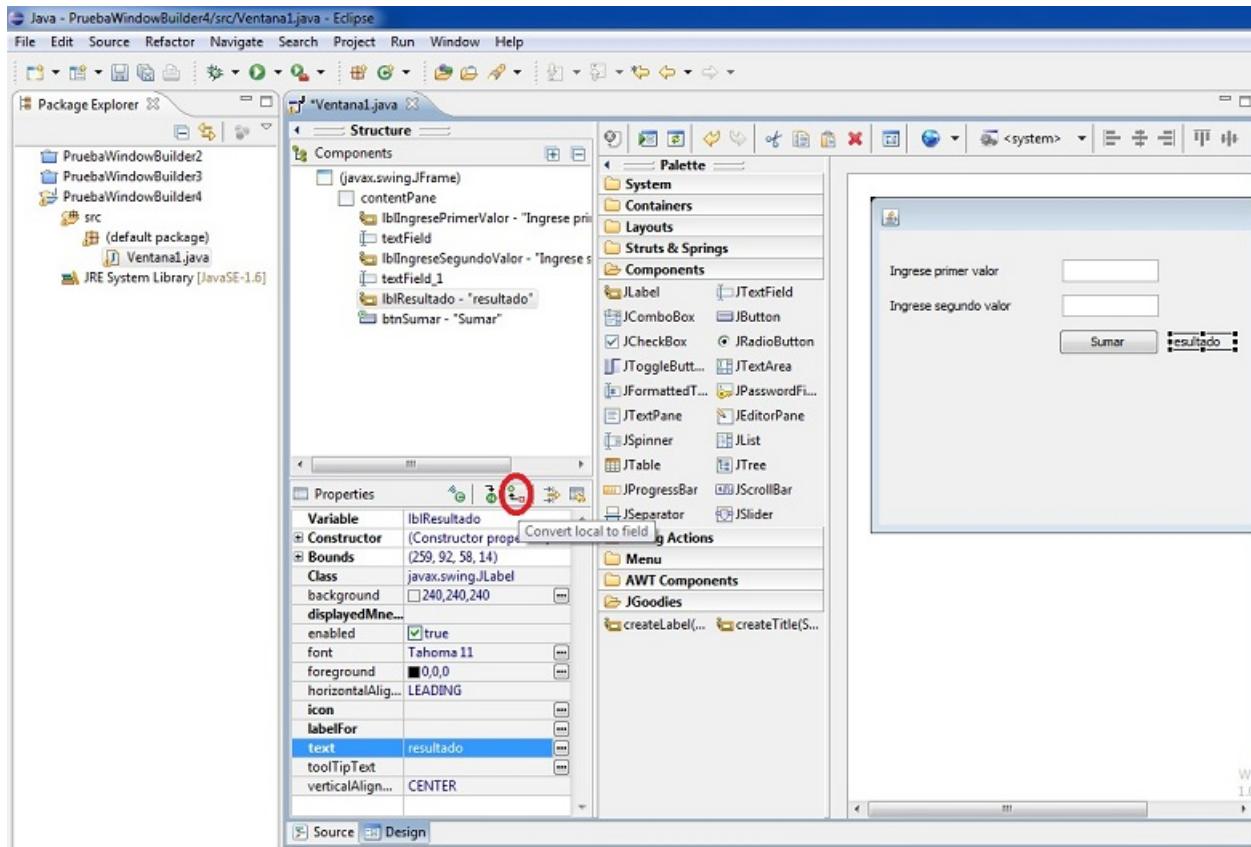
```
btnSumar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
    }
});
```

En el parámetro del método addActionListener del botón que suma se le pasa la referencia a una interface que se crea de tipo ActionListener e implementa el método actionPerformed donde agregaremos el código necesario para responder el evento.

Para este problema debemos rescatar los valores almacenados en los controles de tipo JTextField, convertirlos a entero, sumarlos y enviar dicho resultado a una JLabel.

```
btnSumar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int v1=Integer.parseInt(textField.getText());
        int v2=Integer.parseInt(textField_1.getText());
        int suma=v1+v2;
        lblResultado.setText(String.valueOf(suma));
    }
});
```

Cuando compilamos vemos que no tenemos acceso al objeto lblResultado ya que está definido como una variable local al constructor. Si queremos que se definan como atributos de la clase debemos seleccionar la JLabel y presionar "convert Local to Field" (convertir de variable local a atributo de la clase):

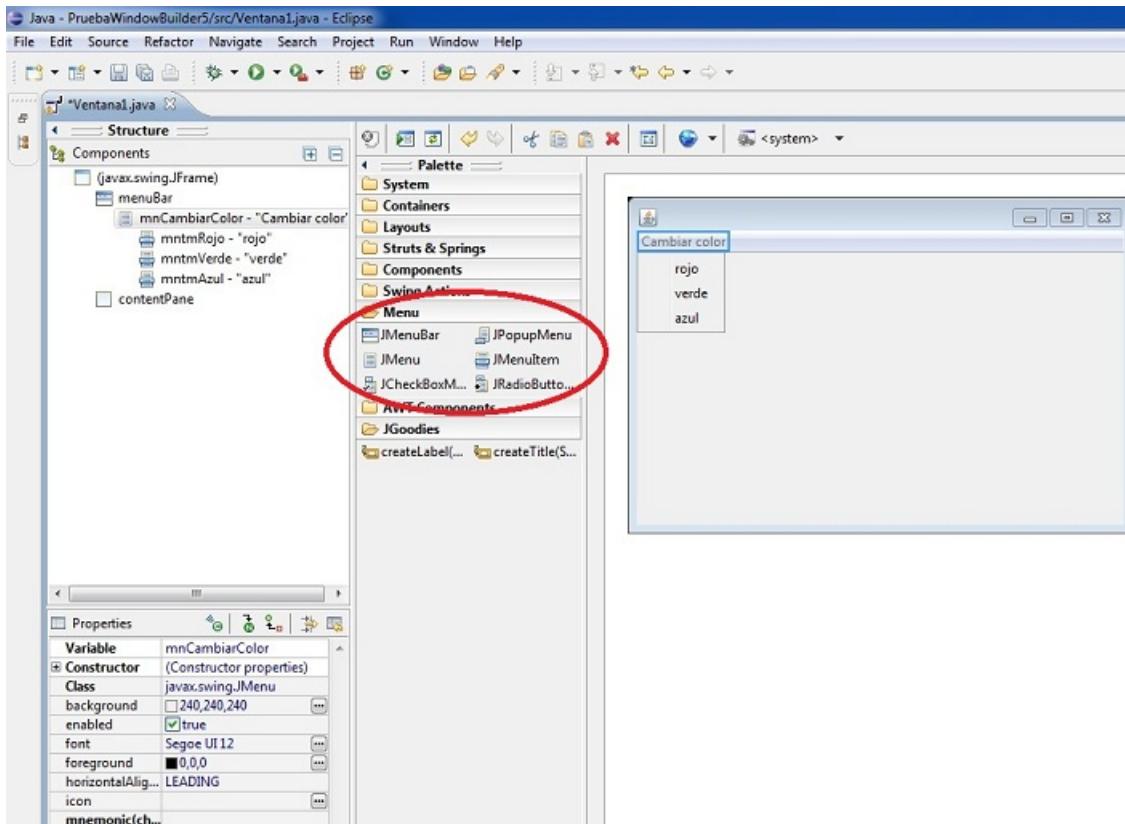


Después de esto podemos acceder desde el método actionPerformed a la label.

## Problema

Crear un menú de opciones que permita cambiar el color de fondo. Disponer un JMenuBar, un JMenu y 3 objetos de la clase JMenuItem. Asociar los eventos respectivos para cada control de tipo JMenuItem.

La interfaz visual debe quedar similar a la siguiente:



Para crear esta interface debemos primero seleccionar la pestaña "Menu" donde se encuentran las componentes relacionadas a la creación de menús.

Debemos agregar (en este orden las siguientes componentes):

1 - Un JMenuBar en la parte superior.

2 - Un objeto de la clase JMenu en la barra del JMenuBar (podemos disponer el texto que queremos que se muestre)

3 - Agregamos un objeto de la clase JMenuItem en el sector donde aparece el texto: "Add items here". Los mismos pasos hacemos para agregar los otros dos JMenuItem.

Ahora debemos asociar el evento clic para cada JMenuItem. Seleccionamos primero el control de tipo JMenuItem y en la ventana de "Properties" presionamos el botón "Show events" y generamos el actionPerformed para el JMenuItem seleccionado.

Luego codificamos:

```
mntmRojo.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        contentPane.setBackground(Color.red);
    }
});
```

Para cambiar el color del JFrame en realidad debemos modificar el color del JPanel que cubre el JFrame. El objeto de la clase JPanel llamado contentPane tiene un método llamado setBackground que nos permite fijar el color de fondo.

De forma similar asociamos los eventos para los otros dos objetos de la clase JMenuItem:

```
JMenuItem mntmVerde = new JMenuItem("Verde");
mntmVerde.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        contentPane.setBackground(Color.green);
    }
});
```

```
mnNewMenu.add(mntmVerde);

JMenuItem mntmAzul = new JMenuItem("Azul");
mntmAzul.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        contentPane.setBackground(Color.blue);
    }
});
```

[Retornar](#)

# 56 - Plug-in WindowBuilder problemas resueltos

[Listado completo de tutoriales](#)

Desarrollaremos una serie de aplicaciones que requieran componentes visuales y utilizaremos el WindowBuilder para agilizar su desarrollo.

## Problema 1

Desarrollar un programa que muestre el tablero de un ascensor:



El funcionamiento es el siguiente:

Inicialmente el ascensor está en el piso 1.

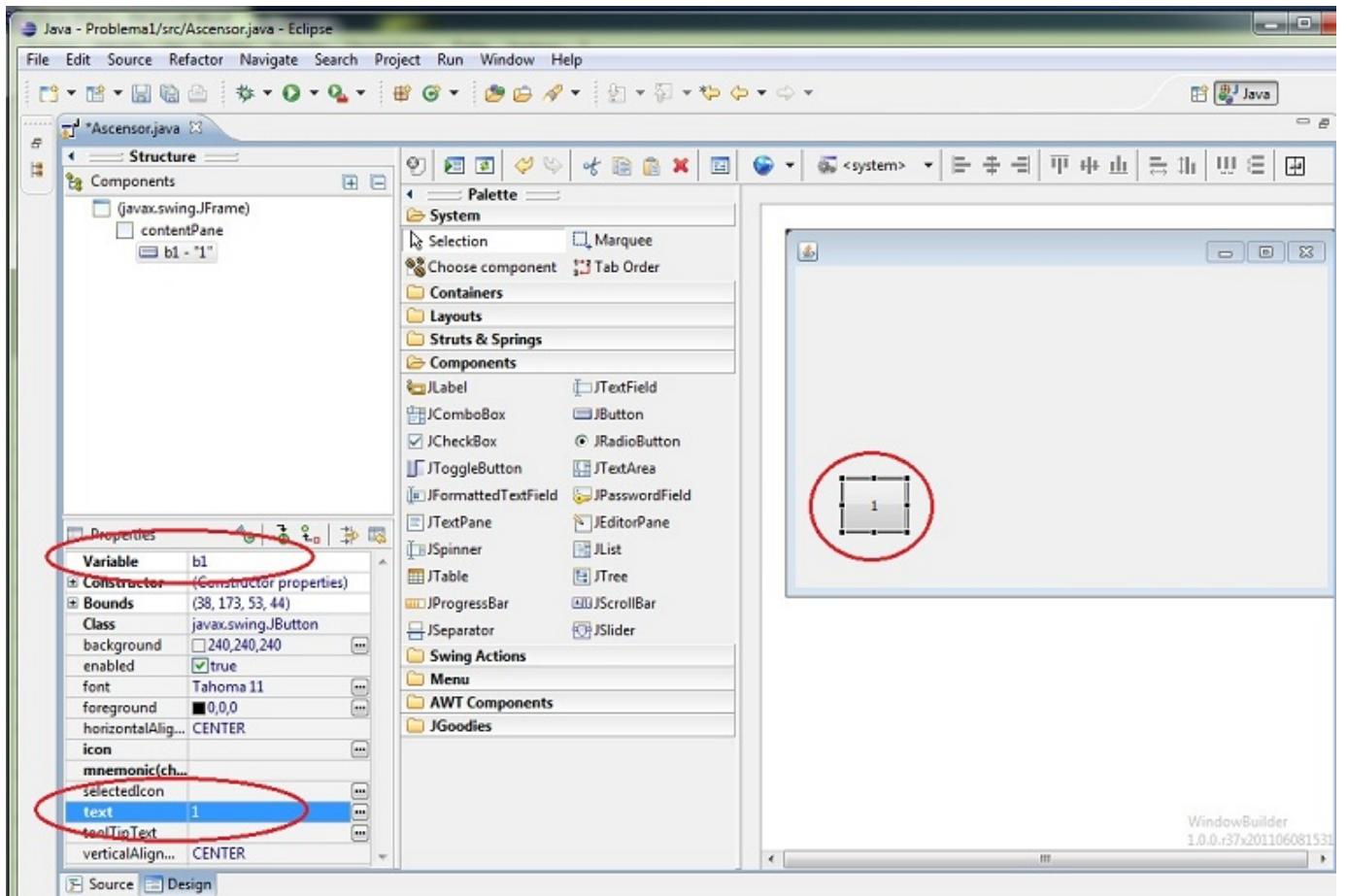
Por ejemplo: si se presiona el botón 3 se muestra en un JLabel el piso número 3 y en otra JLabel la dirección. La cadena "Sube", en caso de presionar un piso superior al actual. Mostramos la cadena "Baja" en el JLabel si se presiona un piso inferior. y si el piso donde se encuentra actualmente coincide con el presionado luego mostrar el mensaje "Piso actual".

Algunos consejos para crear la interfaz visual:

1 - Lo primero que debemos hacer cada vez que creamos un JFrame es definir el Layout a utilizar (normalmente utilizaremos "Absolute Layout", esto lo hacemos presionando el botón derecho del mouse dentro del JFrame y seleccionando la opción "Set Layout").

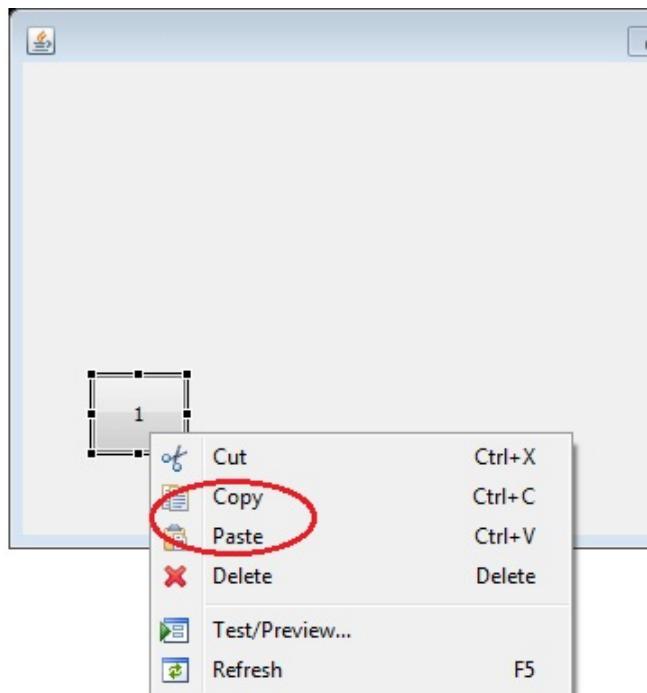
El tipo de layout a utilizar también se lo puede fijar seleccionando el objeto "contentPane"(este objeto es de la clase JPanel y todo JFrame lo contiene como fondo principal) y luego en la ventana de propiedades cambiamos la propiedad "Layout"

2 - Cuando creamos el primer JButton definimos el nombre del objeto cambiando la propiedad "Variable" y mediante la propiedad Text definimos el texto a mostrar (con el mouse dimensionamos el JButton):



3 - Los otros botones los podemos crear de la misma manera seleccionando un objeto de la clase JButton de la "Palette" o cuando tenemos que crear otros objetos semejantes podemos presionar el botón derecho del mouse sobre el objeto a duplicar y seguidamente en el menú contextual seleccionar la opción "Copy" y seguidamente la opción "Paste" con lo que tendremos otro objeto semejante.

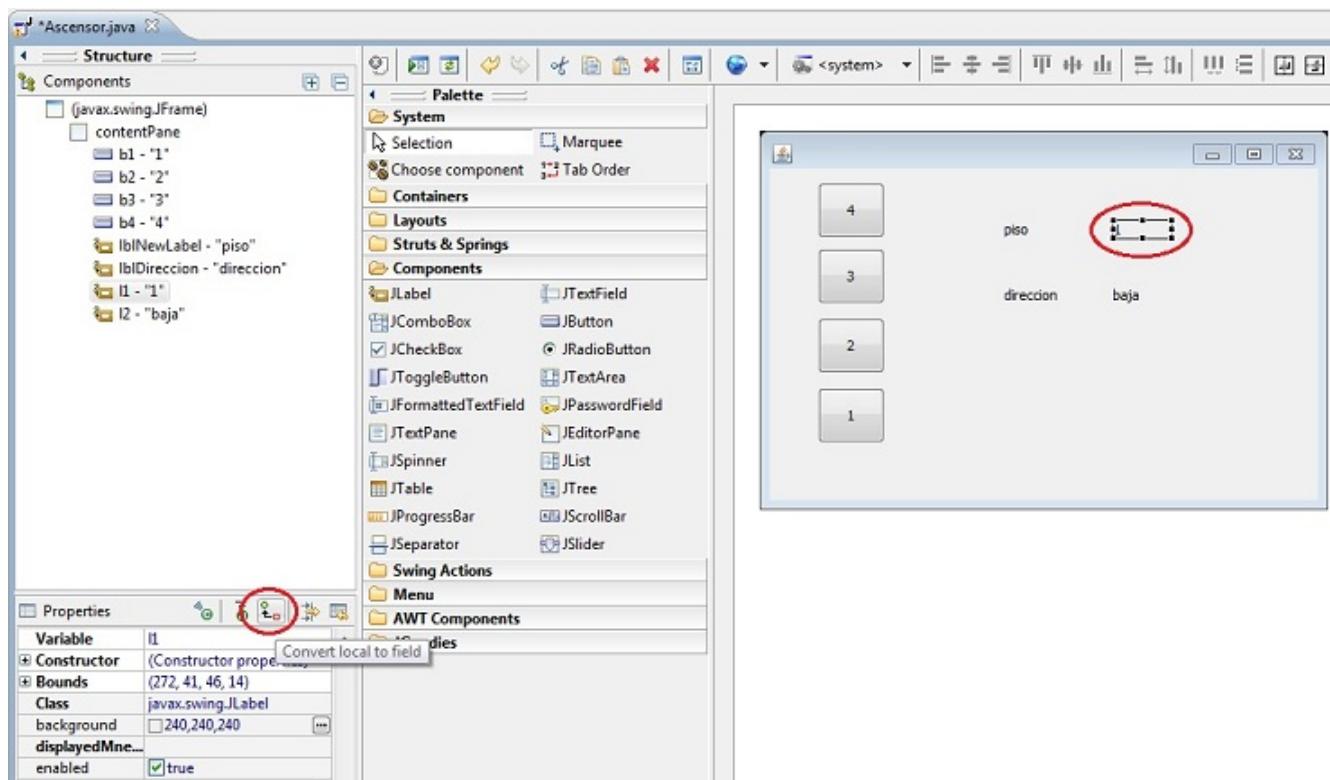
Luego si deberemos definir un nombre para el objeto (propiedad "Variable") y la propiedad "text" para la etiqueta a mostrar:



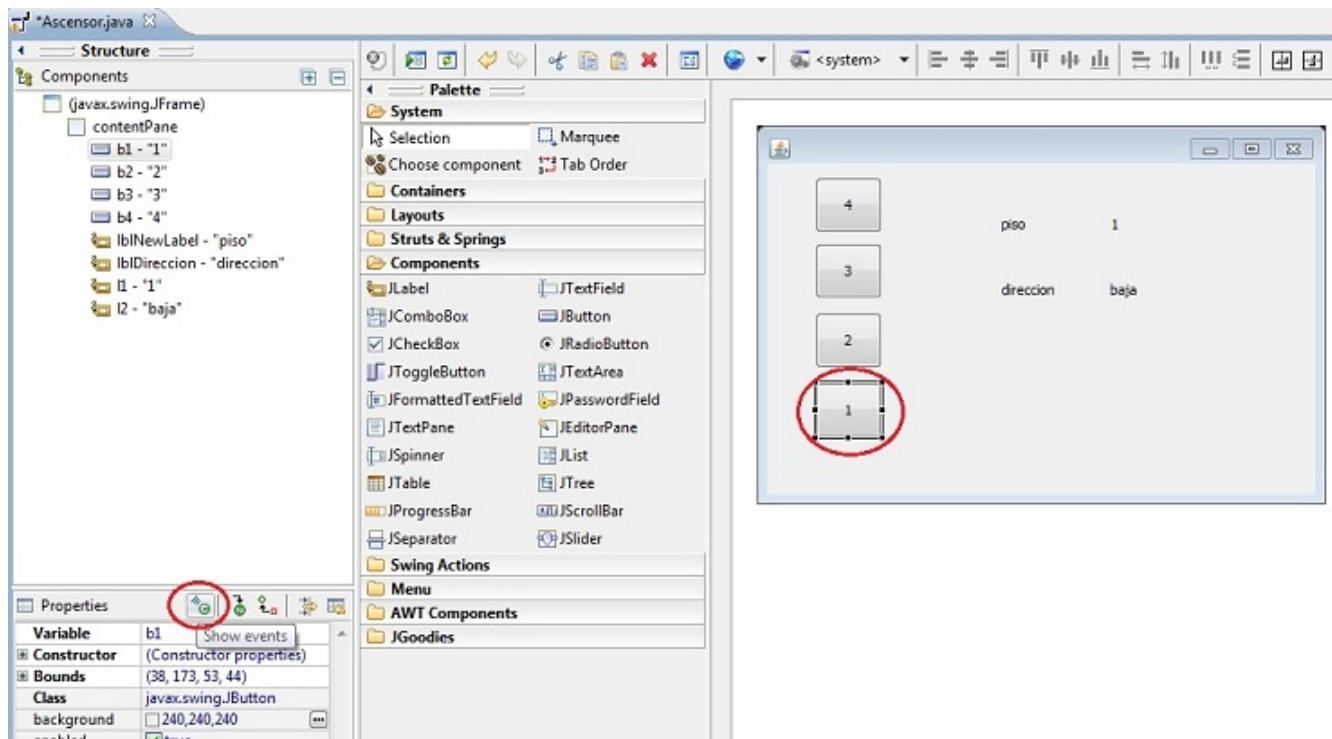
4 - Los objetos que necesitemos consultar o modificar en tiempo de ejecución debemos definirlos como atributos de clase (también llamados campos de clase)

En este problema cuando se presione alguno de los cuatro botones debemos consultar el contenido de la label que indica el piso actual y la label que muestra la dirección será modificada por otro String.

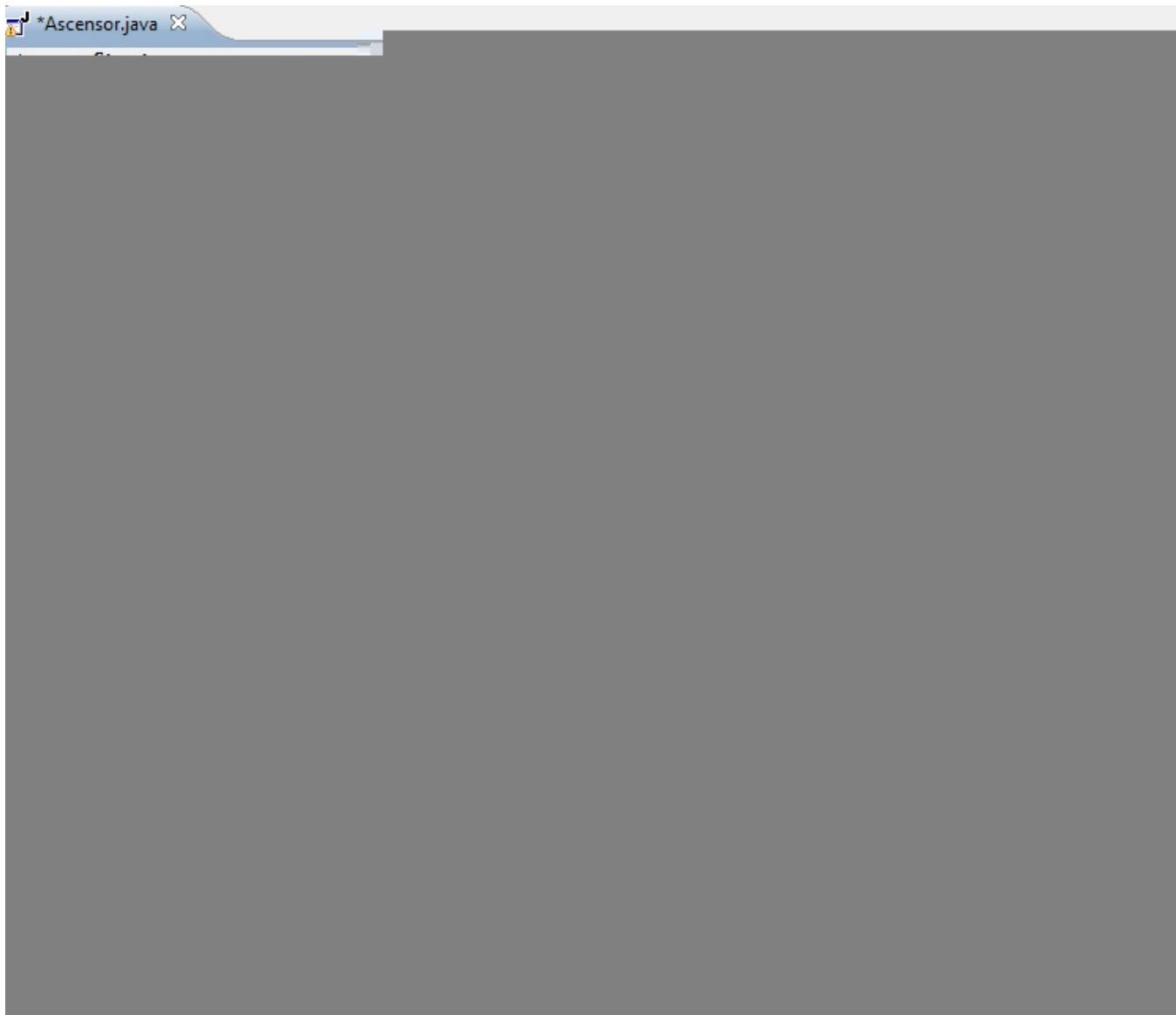
Para definir un control visual como atributo de clase debemos seleccionarlo y presionar en la ventana de propiedades el botón "Convert local to field" (en nuestro problema definamos a estos dos objetos de la clase JLabel con el nombre l1 y l2):



5 - Para capturar el evento clic de un objeto de la clase JButton debemos seleccionarlo y presionar el botón "Show Events":



y seguidamente hacer doble-clic sobre el evento a implementar:



La solución a este problema es el siguiente:

```
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Ascensor extends JFrame {

    private JPanel contentPane;
    private JLabel l1;
    private JLabel l2;
```

```
/**  
 * Launch the application.  
 */  
public static void main(String[] args) {  
    EventQueue.invokeLater(new Runnable() {  
        public void run() {  
            try {  
                Ascensor frame = new Ascensor();  
                frame.setVisible(true);  
            } catch (Exception e) {  
                e.printStackTrace();  
            }  
        }  
    });  
}  
  
/**  
 * Create the frame.  
 */  
public Ascensor() {  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    setBounds(100, 100, 450, 300);  
    contentPane = new JPanel();  
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
    setContentPane(contentPane);  
    contentPane.setLayout(null);  
  
    JButton b1 = new JButton("1");  
    b1.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent e) {  
            int pisoactual=Integer.parseInt(l1.getText());  
            if (1<pisoactual)  
                l2.setText("Baja");  
            else  
                l2.setText("Piso actual");  
            l1.setText("1");  
        }  
    });  
    b1.setBounds(38, 173, 53, 44);  
    contentPane.add(b1);  
  
    JButton b2 = new JButton("2");  
    b2.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent e) {  
            int pisoactual=Integer.parseInt(l1.getText());  
            if (2<pisoactual)  
                l2.setText("Baja");  
            else  
                l2.setText("Piso actual");  
            l1.setText("2");  
        }  
    });  
    b2.setBounds(38, 218, 53, 44);  
    contentPane.add(b2);  
}  
}
```

```

        l2.setText("Baja");
    else
        if (2>pisoactual)
            l2.setText("Sube");
        else
            l2.setText("Piso actual");
    l1.setText("2");
}
});

b2.setBounds(38, 118, 53, 44);
contentPane.add(b2);

JButton b3 = new JButton("3");
b3.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int pisoactual=Integer.parseInt(l1.getText());
        if (3<pisoactual)
            l2.setText("Baja");
        else
            if (3>pisoactual)
                l2.setText("Sube");
            else
                l2.setText("Piso actual");
        l1.setText("3");
    }
});
b3.setBounds(38, 63, 53, 44);
contentPane.add(b3);

JButton b4 = new JButton("4");
b4.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int pisoactual=Integer.parseInt(l1.getText());
        if (4>pisoactual)
            l2.setText("Sube");
        else
            l2.setText("Piso actual");
        l1.setText("4");
    }
});
b4.setBounds(38, 11, 53, 44);
contentPane.add(b4);

JLabel lblNewLabel = new JLabel("piso");
lblNewLabel.setBounds(186, 41, 46, 14);
contentPane.add(lblNewLabel);

```

```

JLabel lblDireccion = new JLabel("direccion");
lblDireccion.setBounds(186, 93, 61, 14);
contentPane.add(lblDireccion);

l1 = new JLabel("1");
l1.setBounds(272, 41, 46, 14);
contentPane.add(l1);

l2 = new JLabel("baja");
l2.setBounds(272, 93, 92, 14);
contentPane.add(l2);
}

}
}

```

Cuando se presiona el botón 1 procedemos a extraer el contenido de la label 1 que almacena el valor del piso actual, como se presionó el primer botón preguntamos si 1 es menor al piso actual, en dicho caso mostramos en la label 2 el texto "Baja" en caso contrario significa que estamos actualmente en el piso 1 (cuando se presiona el botón 1 nunca puede decir el texto sube) Luego cambiamos la etiqueta de la label 1 con el valor "1" que es el nuevo piso:

```

b1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int pisoactual=Integer.parseInt(l1.getText());
        if (1<pisoactual)
            l2.setText("Baja");
        else
            l2.setText("Piso actual");
        l1.setText("1");
    }
});
```

El botón 4 es similar al botón 1 ya que mostraremos la etiqueta "Sube" o "Piso actual":

```

b4.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int pisoactual=Integer.parseInt(l1.getText());
        if (4>pisoactual)
            l2.setText("Sube");
        else
            l2.setText("Piso actual");
        l1.setText("4");
    }
});
```

Si se presiona el botón del segundo piso debemos verificar si 2 es menor, mayor o igual al piso actual (igual para el botón del tercer piso):

```

b2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int pisoactual=Integer.parseInt(l1.getText());
        if (2<pisoactual)
```

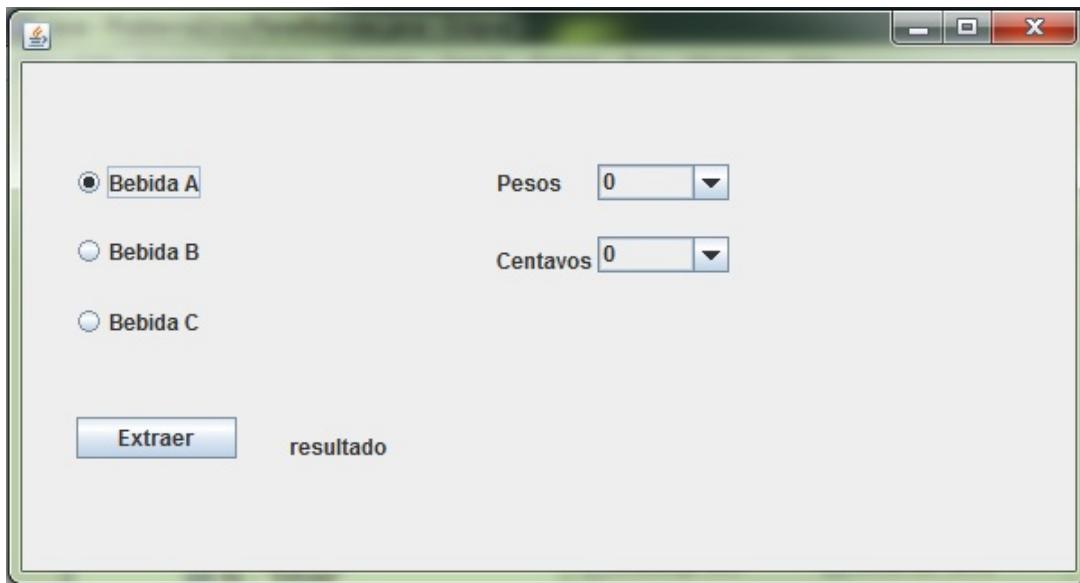
```

++ 12.setText("Baja");
else
    if (2>pisoactual)
        12.setText("Sube");
    else
        12.setText("Piso actual");
11.setText("2");
}
);
}

```

## Problema 2

Desarrollar un programa que muestre un panel para extracción de una bebida:



Por un lado disponer tres objetos de la clase JRadioButton (llamarlos radio1, radio2 y radio 3), configurar el primero para que aparezca seleccionado (propiedad "selected")

Disponer dos objetos de la clase JComboBox (llamarlos comboPesos y comboCentavos)

En el JComboBox pesos inicializar la propiedad model con los valores del 0 al 5 (hay que cargar un valor por cada línea en el diálogo que aparece)

En forma similar el segundo JComboBox cargamos los valores: 0,10,20,30 etc. hasta 90.

Se sabe que :

Bebida A tiene un costo de 0 pesos 80 centavos.

Bebida B tiene un costo de 1 peso 20 centavos.

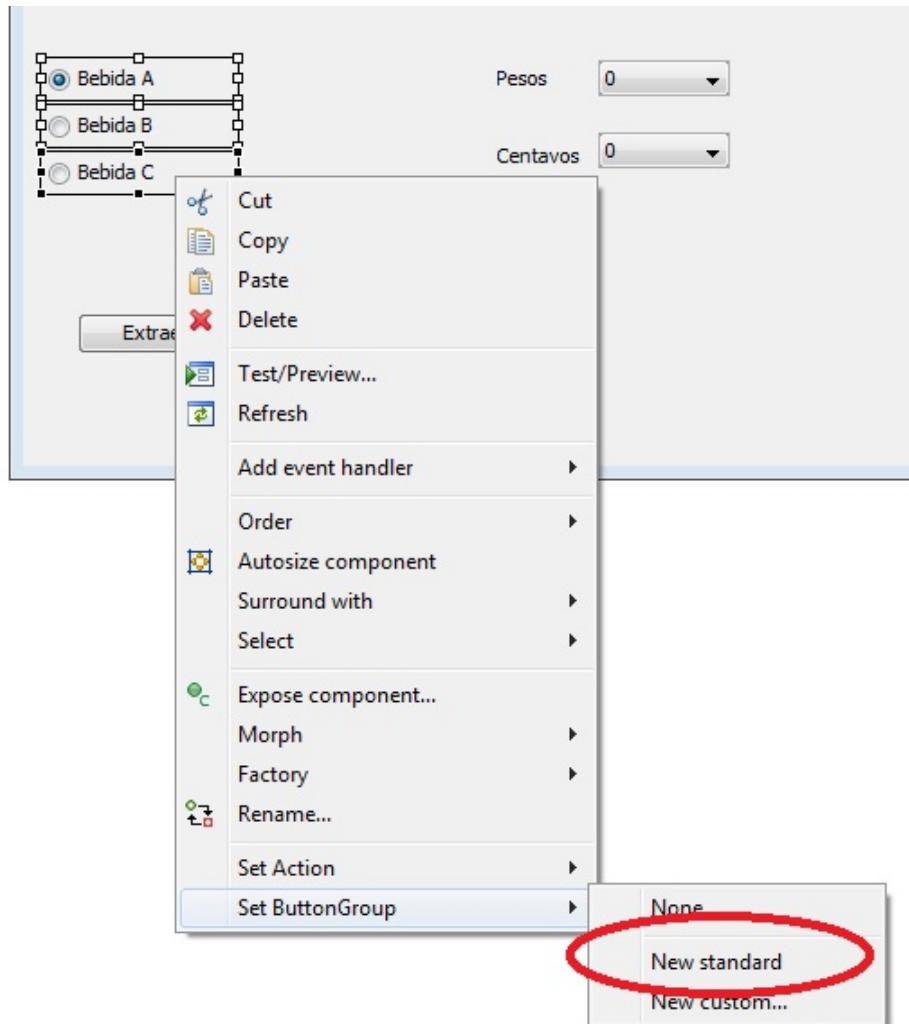
Bebida C tiene un costo de 3 pesos 10 centavos.

Cuando se presiona el botón extraer mostrar en la label de resultado el texto "Correcto" o "Incorrecto" dependiendo la bebida seleccionada y la cantidad de pesos y centavos seleccionados.

Solución:

Para que todos los JRadioButton estén asociados (es decir que cuando se seleccione uno se deseleccione el actual lo debemos hacer en forma visual), primero seleccionamos con el mouse todos los JRadioButton (para seleccionar varios controles presionamos la tecla "Ctrl" del teclado y

con el botón izquierdo del mouse seleccionamos los tres JRadioButton ) y seguidamente presionamos el botón derecho del mouse y seleccionamos "New standard":



Ahora ya tenemos los tres controles de tipo JRadioButton agrupados.

El código fuente del problema es:

```
import java.awt.EventQueue;  
  
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;  
import javax.swing.JRadioButton;  
import javax.swing.JLabel;  
import javax.swing.JComboBox;  
import javax.swing.DefaultComboBoxModel;  
import javax.swing.JButton;  
import java.awt.event.ActionListener;  
import java.awt.event.ActionEvent;  
import javax.swing.ButtonGroup;
```

```
public class PanelBebidas extends JFrame {

    private JPanel contentPane;
    private JComboBox comboPesos;
    private JComboBox comboCentavos;
    private JRadioButton radio1;
    private JRadioButton radio2;
    private JRadioButton radio3;
    private JLabel l1;
    private final ButtonGroup buttonGroup = new ButtonGroup();

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    PanelBebidas frame = new PanelBebidas();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public PanelBebidas() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 600, 319);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblNewLabel = new JLabel("Pesos");
        lblNewLabel.setBounds(263, 59, 46, 14);
        contentPane.add(lblNewLabel);

        comboPesos = new JComboBox();
        comboPesos.setModel(new DefaultComboBoxModel(new String[] {"", "50", "100", "200", "500", "1000", "2000", "5000", "10000"}));
        comboPesos.setBounds(319, 56, 73, 20);
        contentPane.add(comboPesos);
    }
}
```

```
JLabel lblNewLabel_1 = new JLabel("Centavos");
lblNewLabel_1.setBounds(263, 102, 58, 14);
contentPane.add(lblNewLabel_1);

comboCentavos = new JComboBox();
comboCentavos.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
    }
});
comboCentavos.setModel(new DefaultComboBoxModel(new String[] {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47", "48", "49", "50", "51", "52", "53", "54", "55", "56", "57", "58", "59", "60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "70", "71", "72", "73", "74", "75", "76", "77", "78", "79", "80", "81", "82", "83", "84", "85", "86", "87", "88", "89", "90", "91", "92", "93", "94", "95", "96", "97", "98", "99", "100"}));
comboCentavos.setBounds(319, 96, 73, 20);
contentPane.add(comboCentavos);

JButton b1 = new JButton("Extraer");
b1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        int pesos=Integer.parseInt((String)comboPesos.getSelectedItem());
        int centavos=Integer.parseInt((String)comboCentavos.getSelectedItem());
        if (radio1.isSelected() && pesos==0 && centavos==0)
            l1.setText("Correcto");
        else
            if (radio2.isSelected() && pesos==1 && centavos==1)
                l1.setText("Correcto");
            else
                if (radio3.isSelected() && pesos==3 && centavos==1)
                    l1.setText("Correcto");
                else
                    l1.setText("Incorrecto");
    }
});
b1.setBounds(30, 196, 89, 23);
contentPane.add(b1);

l1 = new JLabel("resultado");
l1.setBounds(148, 205, 73, 14);
contentPane.add(l1);

radio1 = new JRadioButton("Bebida A");
buttonGroup.add(radio1);
radio1.setSelected(true);
radio1.setBounds(10, 55, 109, 23);
contentPane.add(radio1);

radio2 = new JRadioButton("Bebida B");
buttonGroup.add(radio2);
```

```

        radio2.setBounds(10, 81, 109, 23);
        contentPane.add(radio2);

        radio3 = new JRadioButton("Bebida C");
        buttonGroup.add(radio3);
        radio3.setBounds(10, 107, 109, 23);
        contentPane.add(radio3);
    }

}

```

La lógica del problema se encuentra cuando se presiona el botón "Extraer":

```

b1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        int pesos=Integer.parseInt((String)comboPesos.getSelectedItem());
        int centavos=Integer.parseInt((String)comboCentavos.getSelectedItem());
        if (radio1.isSelected() && pesos==0 && centavos==80)
            l1.setText("Correcto");
        else
            if (radio2.isSelected() && pesos==1 && centavos==20)
                l1.setText("Correcto");
            else
                if (radio3.isSelected() && pesos==3 && centavos==10)
                    l1.setText("Correcto");
                else
                    l1.setText("Incorrecto");
    }
});

```

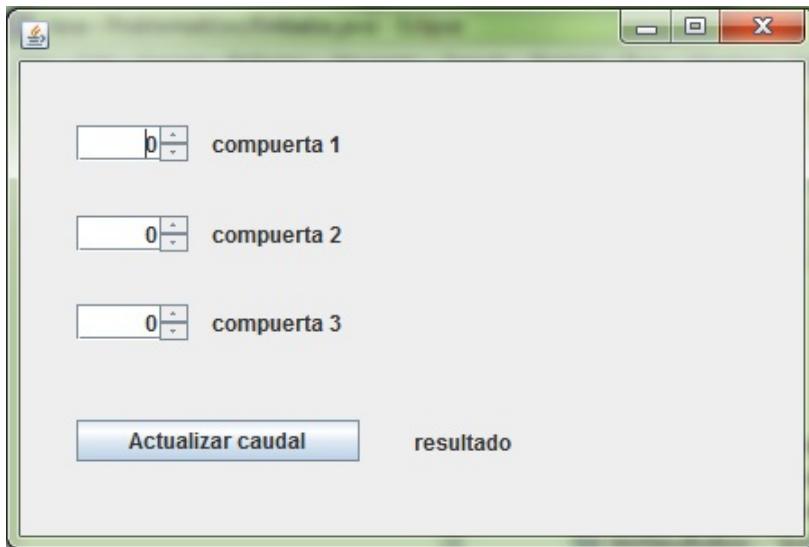
Extraemos los contenidos de los dos controles de tipo JComboBox y los convertimos a entero. Luego mediante tres if verificamos si el primer JRadioButton está seleccionado y el dinero seleccionado corresponde a exactamente 0 pesos y 80 centavos, en tal caso mostramos en la label el mensaje "Correcto". La lógica es similar para las otras dos bebidas.

### Problema 3

Un embalse debe manejar la cantidad de mts3 de agua que pasa por cada compuerta. Por cada compuerta puede pasar un caudal de 100 mts3 x seg.

Cuando presionamos el botón "Actualizar caudal" mostramos el nivel de caudal actual y un mensaje que indica si el caudal es Bajo (0 a 100 mts3 x seg.) , Medio (> 100 -200 mts3. x seg.) o Alto (>200 mts3 x seg.)

Para la selección del caudal de cada compuerta utilizar componentes de tipo JSpinner.



El código fuente es:

```
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JSpinner;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.SpinnerNumberModel;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Embalse extends JFrame {

    private JPanel contentPane;
    private JSpinner spinner1;
    private JSpinner spinner2;
    private JSpinner spinner3;
    private JLabel l1;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Embalse frame = new Embalse();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    });
}

/**
 * Create the frame.
 */
public Embalse() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    spinner1 = new JSpinner();
    spinner1.setModel(new SpinnerNumberModel(0, 0, 100, 1));
    spinner1.setBounds(31, 35, 62, 20);
    contentPane.add(spinner1);

    spinner2 = new JSpinner();
    spinner2.setModel(new SpinnerNumberModel(0, 0, 100, 1));
    spinner2.setBounds(31, 85, 62, 20);
    contentPane.add(spinner2);

    spinner3 = new JSpinner();
    spinner3.setModel(new SpinnerNumberModel(0, 0, 100, 1));
    spinner3.setBounds(31, 134, 62, 20);
    contentPane.add(spinner3);

    JLabel lblCompuerta = new JLabel("compuerta 1");
    lblCompuerta.setBounds(106, 38, 82, 14);
    contentPane.add(lblCompuerta);

    JLabel lblCompuerta_1 = new JLabel("compuerta 2");
    lblCompuerta_1.setBounds(106, 88, 82, 14);
    contentPane.add(lblCompuerta_1);

    JLabel lblCompuerta_2 = new JLabel("compuerta 3");
    lblCompuerta_2.setBounds(106, 137, 82, 14);
    contentPane.add(lblCompuerta_2);

    JButton btnNewButton = new JButton("Actualizar caudal");
    btnNewButton.addActionListener(new ActionListener() {
```

```

        public void actionPerformed(ActionEvent arg0) {
            int v1=Integer.parseInt(spinner1.getValue());
            int v2=Integer.parseInt(spinner2.getValue());
            int v3=Integer.parseInt(spinner3.getValue());
            int suma=v1+v2+v3;
            if (suma<=100)
                l1.setText("Bajo");
            else
                if (suma<=200)
                    l1.setText("Medio");
                else
                    l1.setText("Alto");
            }
        });
btnNewButton.setBounds(31, 198, 157, 23);
contentPane.add(btnNewButton);

l1 = new JLabel("resultado");
l1.setBounds(218, 203, 149, 14);
contentPane.add(l1);
}
}
}

```

En el evento clic del JButton extraemos los tres valores almacenados en los JSpinner:

```

int v1=Integer.parseInt(spinner1.getValue().toString());
int v2=Integer.parseInt(spinner2.getValue().toString());
int v3=Integer.parseInt(spinner3.getValue().toString());

```

y mediante tres if valuamos si la suma es menor o igual a 100 o menor o igual a 200 o en su defecto es mayor a 200:

```

int suma=v1+v2+v3;
if (suma<=100)
    l1.setText("Bajo");
else
    if (suma<=200)
        l1.setText("Medio");
    else
        l1.setText("Alto");

```

## Problema propuesto

1. Implementar un programa para la extracción de dinero de un cajero automático.  
Se debe poder fijar la cantidad de dinero a extraer:  
Disponer un control de tipo JComboBox (disponer los valores: 0,50,150 etc. hasta 500)  
Por otro lado poder seleccionar el tipo de cuenta (almacenar en otro JComboBox los

textos "Caja de Ahorro" y "Cuenta Corriente".

Se debe tener en cuenta que:

De Caja de Ahorro se puede extraer hasta 200.

De Cuenta Corriente se puede extraer hasta 400.

Al presionar el botón extraer mostrar en una label el texto "correcto" si para el tipo de cuenta el importe está permitido.

Inicialmente el cajero tiene almacenado un monto de 3000 pesos. Restar en cada extracción el monto respectivo y mostrar el mensaje "fuera de servicio" cuando se intenta extraer más del dinero que hay en el cajero.



[Retornar](#)

# 57 - Clase Graphics y sus métodos

[Listado completo de tutoriales](#)

Java proporciona la clase `Graphics`, que permite dibujar elipses, cuadrados, líneas, mostrar texto y también tiene muchos otros métodos de dibujo. Para cualquier programador, es esencial el entendimiento de la clase `Graphics`, antes de adentrarse en el dibujo en Java.

La clase `Graphics` proporciona el entorno de trabajo para cualquier operación gráfica que se realice dentro del AWT.

Para poder pintar, un programa necesita un contexto gráfico válido, representado por una instancia de la clase `Graphics`. Pero esta clase no se puede instanciar directamente; así que debemos crear un componente y pasarlo al programa como un argumento al método `paint()`.

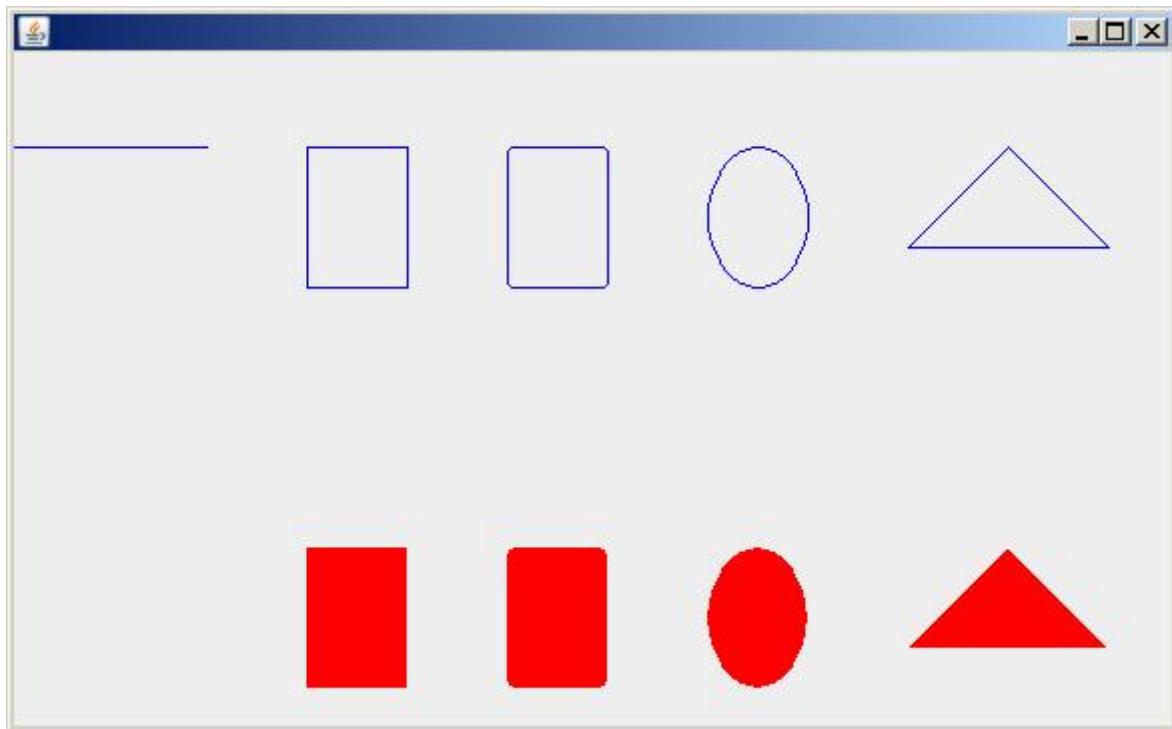
El único argumento del método `paint()` es un objeto de esta clase. La clase `Graphics` dispone de métodos para soportar tres categorías de operaciones gráficas:

- 1) Dibujo de primitivas gráficas,
- 2) Dibujo de texto,
- 3) Presentación de imágenes en formatos `*.gif` y `*.jpeg`.

Además, la clase `Graphics` mantiene un contexto gráfico: un área de dibujo actual, un color de dibujo del `Background` y otro del `Foreground`, un `Font` con todas sus propiedades, etc. Los ejes están situados en la esquina superior izquierda. Las coordenadas se miden siempre en pixels.

## Problema 1

Crear una aplicación que utilice las primitivas gráficas principales que provee la clase `Graphics`:



```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Graphics;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

public class Grafico1 extends JFrame {

    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Grafico1 frame = new Grafico1();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Grafico1() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
```

```

contentPane = new JPanel();
contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
setContentPane(contentPane);
contentPane.setLayout(null);
setBounds(0,0,800,600);
}

public void paint (Graphics g)
{
    super.paint(g);

    g.setColor (Color.blue);
    g.drawLine (0, 70, 100, 70);
    g.drawRect (150, 70, 50, 70);
    g.drawRoundRect (250, 70, 50, 70, 6, 6);
    g.drawOval (350, 70, 50, 70);
    int [] vx1 = {500, 550, 450};
    int [] vy1 = {70, 120, 120};
    g.drawPolygon (vx1, vy1, 3);

    g.setColor (Color.red);
    g.fillRect (150, 270, 50, 70);
    g.fillRoundRect (250, 270, 50, 70, 6, 6);
    g.fillOval (350, 270, 50, 70);
    int [] vx2 = {500, 550, 450};
    int [] vy2 = {270, 320, 320};
    g.fillPolygon (vx2, vy2, 3);
}
}

```

Sobreescrivimos el método paint heredado de la clase JFrame:

```

public void paint (Graphics g)
{

```

El método paint se ejecuta cada vez que el JFrame debe ser redibujado y llega como parámetro un objeto de la clase Graphics. Este objeto nos permite acceder al fondo del JFrame y utilizando las primitivas gráficas dibujar líneas, rectángulos, elipses etc.

Lo primero que hacemos dentro de este método es llamar al método paint de la clase superior para que se pinte el fondo del JFrame y otras componentes contenidas dentro (para llamar al método paint de la clase JFrame debemos anteceder la palabra clave super y pasar el parámetro respectivo):

```

super.paint(g);

```

Mediante el método setColor activamos un color:

```

g.setColor (Color.blue);

```

Dibuja una línea desde la coordenada (0,70) es decir columna 0 y fila 70 en píxeles, hasta la coordenada (100,70). La línea es de color azul:

```
g.drawLine (0, 70, 100, 70);
```

Dibujamos un rectángulo desde la coordenada (150,70) con un ancho de 50 píxeles y un alto de 70, solo se pinta el perímetro del rectángulo de color azul):

```
g.drawRect (150, 70, 50, 70);
```

Similar a drawRect más un valor de redondeo de los vértices que le indicamos en el quinto y sexto parámetro:

```
g.drawRoundRect (250, 70, 50, 70, 6, 6);
```

Dibujamos un óvalo:

```
g.drawOval (350, 70, 50, 70);
```

Dibujamos un triángulo (debemos indicar mediante dos vectores los vértices de cada punto del triángulo), el primer punto es el (500,70) el segundo punto es el (550,120) y por último el punto (450,120):

```
int [] vx1 = {500, 550, 450};  
int [] vy1 = {70, 120, 120};  
g.drawPolygon (vx1, vy1, 3);
```

De forma similar los métodos fillRect, fillRoundRect, fillOval y fillPolygon son similares a los anteriores con la diferencia que pinta su interior con el color activo de la última llamada al método setColor:

```
g.setColor (Color.red);  
g.fillRect (150, 270, 50, 70);  
g.fillRoundRect (250, 270, 50, 70, 6, 6);  
g.fillOval (350, 270, 50, 70);  
int [] vx2 = {500, 550, 450};  
int [] vy2 = {270, 320, 320};  
g.fillPolygon (vx2, vy2, 3);
```

## Dibujar texto

La clase Graphics permite ?dibujar? texto, como alternativa al texto mostrado en los componentes JLabel, JTextField y JTextArea. El método que permite graficar texto sobre el JFrame es:

```
drawString(String str, int x, int y);
```

## Problema 2

Crear una aplicación que utilice las primitiva drawString de Java:

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Graphics;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

public class Grafico1 extends JFrame {

    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Grafico1 frame = new Grafico1();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Grafico1() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new BorderLayout(0, 0));
        setContentPane(contentPane);
        setBounds(0,0,800,600);
    }

    public void paint (Graphics g)
    {
        super.paint(g);
        g.setColor (Color.blue);
        g.drawString("Primer linea",10,200);
        g.drawString("Segunda linea",10,300);
    }
}
```

## 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`, `magenta`, `cyan`, `orange`, `pink`, `gray`, `darkGray`, `lightGray`.

## Problema 3

Crear una aplicación que dibuje 255 líneas creando un color distinto para cada una de ellas:



```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Graphics;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

public class Grafico1 extends JFrame {

    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Grafico1 frame = new Grafico1();
                    frame.setVisible(true);
                }
                catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```

        } catch (Exception e) {
            e.printStackTrace();
        }
    });
}

/**
 * Create the frame.
 */
public Grafico1() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    setBounds(0,0,800,255);
}

public void paint (Graphics g)
{
    super.paint(g);
    int fila = 0;
    for (int rojo = 0 ; rojo <= 255 ; rojo++)
    {
        Color col = new Color (rojo, 0, 0);
        g.setColor (col);
        g.drawLine (0, fila, 800, fila);
        fila++;
    }
}
}

```

Dentro de un for creamos objetos de la clase Color y fijamos el color de la línea seguidamente (con esto logramos un degradé del negro al rojo):

```

int fila = 0;
for (int rojo = 0 ; rojo <= 255 ; rojo++)
{
    Color col = new Color (rojo, 0, 0);
    g.setColor (col);
    g.drawLine (0, fila, 800, fila);
    fila++;
}

```

## Presentación de imágenes

Java permite incorporar imágenes de tipo GIF y JPEG definidas en ficheros. Se dispone para ello de la clase `java.awt.Image`. Para cargar una imagen hay que indicar la localización del archivo y cargarlo mediante el método `getImage()`. Este método existe en las clases `java.awt.Toolkit`.

Entonces, para cargar una imagen hay que comenzar creando un objeto (o una referencia) Image y llamar al método `getImage()` (de Toolkit); Una vez cargada la imagen, hay que representarla, para lo cual se redefine el método `paint()` para llamar al método `drawImage()` de la clase `Graphics`. Los objetos `Graphics` pueden mostrar imágenes a través del método: `drawImage()`. Dicho método admite varias formas, aunque casi siempre hay que incluir el nombre del objeto imagen creado.

## Clase Image

Una imagen es un objeto gráfico rectangular compuesto por pixels coloreados. Cada pixel en una imagen describe un color de una particular localización de la imagen.

A continuación, algunos métodos de la clase `Image`:

La clase `Graphics` provee el método `drawImage()` para dibujar imágenes; este método admite varias formas:

- `drawImage (Image i, int x, int y, ImageObserver o)`
- `drawImage (Image i,int x,int y,int width,int height,ImageObserver o)`

## Problema 4

Crear una aplicación que muestre un archivo jpg dentro de un `JFrame`.



Luego de crear el proyecto debemos disponer un archivo en la carpeta raíz del proyecto (el archivo debe llamarse `imagen1.jpg`)

```
import java.awt.BorderLayout;
import java.awt.EventQueue;
```

```

import java.awt.Graphics;
import java.awt.Image;
import java.awt.Toolkit;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

public class Graficol extends JFrame {

    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Graficol frame = new Graficol();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Graficol() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new BorderLayout(0, 0));
        setContentPane(contentPane);
        setBounds(0, 0, 800, 600);
    }

    public void paint (Graphics g)
    {
        super.paint(g);
        Toolkit t = Toolkit.getDefaultToolkit ();
        Image imagen = t.getImage ("imagen1.jpg");
        g.drawImage (imagen, 0, 0, this);
    }
}

```

Creamos un objeto de la clase Toolkit llamando al método estático de la misma clase:

```
Toolkit t = Toolkit.getDefaultToolkit ();
```

Creamos un objeto de la clase Image llamando al método getImage de la clase Toolkit pasando como parámetro el archivo con la imagen:

```
Image imagen = t.getImage ("imagen1.jpg");
```

Por último llamamos al método drawImage con la referencia al objeto de tipo Image, la columna, la fila y la referencia al JFrame donde debe dibujarse:

```
g.drawImage (imagen, 0, 0, this);
```

### Método repaint()

Este es el método que con más frecuencia es llamado por el programador. El método repaint() llama ?lo antes posible? al método paint() del componente.

El método repaint() puede ser:

```
repaint()  
repaint(int x, int y, int w, int h)
```

Las segunda forma permiten definir una zona rectangular de la ventana a la que aplicar el método.

## Problema 5

Crear una aplicación que muestre un círculo en medio de la pantalla y mediante dos botones permitir que se desplace a izquierda o derecha.

```
import java.awt.Color;  
import java.awt.EventQueue;  
import java.awt.Graphics;  
  
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;  
import javax.swing.JButton;  
import java.awt.event.ActionListener;  
import java.awt.event.ActionEvent;  
  
public class Grafico1 extends JFrame {  
  
    private JPanel contentPane;  
  
    /**  
     * Launch the application.  
     */  
  
    private int columna;  
  
    public static void main(String[] args) {
```

```

        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Grafico1 frame = new Grafico1();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Grafico1() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JButton bi = new JButton("Izquierda");
        bi.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                columnna=columnna-10;
                repaint();
            }
        });
        bi.setBounds(105, 482, 89, 23);
        contentPane.add(bi);

        JButton bd = new JButton("Derecha");
        bd.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                columnna=columnna+10;
                repaint();
            }
        });
        bd.setBounds(556, 482, 89, 23);
        contentPane.add(bd);
        setBounds(0,0,800,600);
        columnna=400;
    }

    public void paint (Graphics g)
    {
        super.paint(g);
        g.setColor (Color.red);
        g.fillOval (columnna, 300, 100, 100);
    }
}

```

Definimos un atributo columnna:

```
private int columna;
```

Cuando se presiona el botón (bi) restamos 10 al atributo columna y pedimos que se ejecute el método paint (esto último llamando al método repaint()), el método repaint borra todo lo dibujado dentro del JFrame y llama al paint:

```
bi.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        columna=columna-10;
        repaint();
    }
});
```

El método paint dibuja un círculo utilizando como posición el valor del atributo columna:

```
public void paint (Graphics g)
{
    super.paint(g);
    g.setColor (Color.red);
    g.fillOval (columna, 300, 100, 100);
}
```

## Problema 6

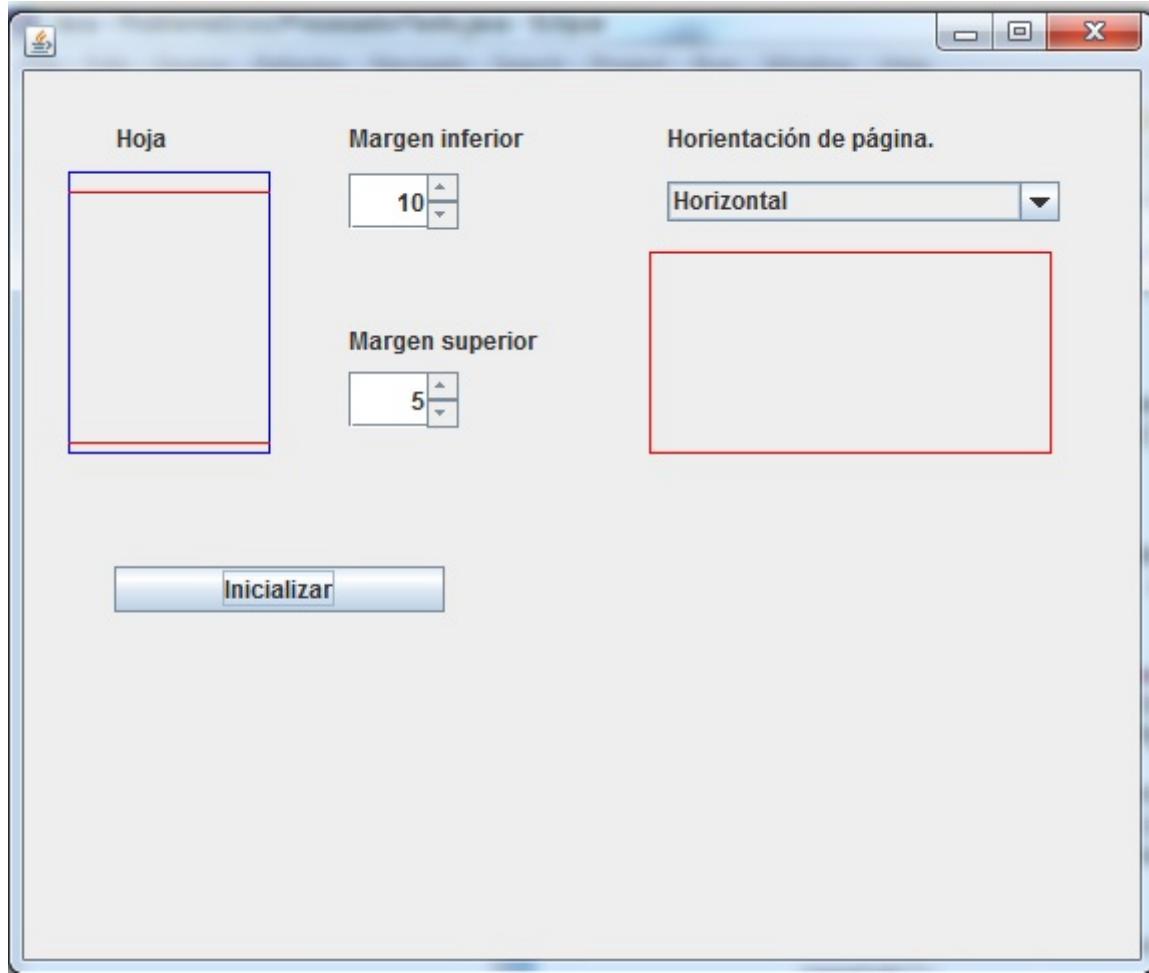
Se debe desarrollar una pantalla para configurar ciertas características de un procesador de texto.

Debe aparecer y poder seleccionarse los márgenes superior e inferior de la página.

Los márgenes pueden ir en el rango de 0 a 10. Desplazar las líneas a medida que modificamos los márgenes.

Por otro lado tenemos la orientación de página. La misma se administra a través de un JComboBox que tiene dos valores posibles (Horizontal y Vertical). Cuando está seleccionado en el JComboBox el String Horizontal dibujar un rectángulo con base mayor a la altura, y cuando está seleccionado el String Vertical dibujar un rectángulo con una base menor.

Cuando se presiona el botón inicializar la configuración de márgenes se inicializan con 0 y se selecciona orientación horizontal.



Para implementar esta aplicación con el WindowBuilder creamos la interfaz visual, disponemos 4 objetos de la clase JLabel, dos JSpinner, un JButton y un objeto de la clase JComboBox. El dibulo de la hoja con las líneas de márgenes superior e inferior como el gráfico de orientación de la hoja se hacen en el método paint.

El código fuente que resuelve esta aplicación es:

```
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Graphics;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JSpinner;
import javax.swing.JLabel;
import javax.swing.JComboBox;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.SpinnerNumberModel;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
```

```
public class ProcesadorTexto extends JFrame {

    private JPanel contentPane;
    private JSpinner sp1;
    private JSpinner sp2;
    private JComboBox comboBox;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    ProcesadorTexto frame = new ProcesadorTexto();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public ProcesadorTexto() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 573, 481);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        sp1 = new JSpinner();
        sp1.addChangeListener(new ChangeListener() {
            public void stateChanged(ChangeEvent arg0) {
                repaint();
            }
        });
        sp1.setModel(new SpinnerNumberModel(0, 0, 10, 1));
        sp1.setBounds(162, 51, 55, 28);
        contentPane.add(sp1);

        sp2 = new JSpinner();
        sp2.addChangeListener(new ChangeListener() {
            public void stateChanged(ChangeEvent e) {
                repaint();
            }
        });
        sp2.setModel(new SpinnerNumberModel(0, 0, 10, 1));
        sp2.setBounds(162, 150, 55, 28);
        contentPane.add(sp2);

        JLabel lblMargenInferior = new JLabel("Margen inferior");
        lblMargenInferior.setBounds(162, 26, 109, 14);
        contentPane.add(lblMargenInferior);

        JLabel lblMargenSuperior = new JLabel("Margen superior");
        lblMargenSuperior.setBounds(162, 127, 109, 14);
    }
}
```

```

        lblMargenSuperior.setBounds(104, 121, 107, 14);
        contentPane.add(lblMargenSuperior);

        JLabel lblHoja = new JLabel("Hoja");
        lblHoja.setBounds(46, 26, 46, 14);
        contentPane.add(lblHoja);

        comboBox = new JComboBox();
        comboBox.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent arg0) {
                repaint();
            }
        });
        comboBox.setModel(new DefaultComboBoxModel(new String[]
        {"Horizontal", "Vertical"}));
        comboBox.setBounds(321, 55, 196, 20);
        contentPane.add(comboBox);

        JLabel lblHorientacionDePgina = new JLabel("Horientaci00F3n de
        pu00E1gina.");
        lblHorientacionDePgina.setBounds(321, 26, 203, 14);
        contentPane.add(lblHorientacionDePgina);

        JButton btnInicializar = new JButton("Inicializar");
        btnInicializar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                sp1.setValue(0);
                sp2.setValue(0);
                comboBox.setSelectedIndex(0);
                repaint();
            }
        });
        btnInicializar.setBounds(45, 247, 165, 23);
        contentPane.add(btnInicializar);
    }

    public void paint(Graphics g)
    {
        super.paint(g);
        g.setColor(Color.blue);
        g.drawRect(30,80,100,140);
        int ms=Integer.parseInt(sp1.getValue().toString());
        int mi=Integer.parseInt(sp2.getValue().toString());
        g.setColor(Color.red);
        g.drawLine(30,80+ms,130,80+ms);
        g.drawLine(30,220-mi,130,220-mi);
        String direccion=(String)comboBox.getSelectedItem();
        if (direccion.equals("Horizontal"))
            g.drawRect(320,120,200,100);
        else
            g.drawRect(320,120,100,200);
    }
}

```

## Explicación del código.

Para el evento stateChanged de los controles JSpinner se debe llamar al método repaint() para que se grafique nuevamente las líneas de márgenes:

```

sp1.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent arg0) {
        repaint();
    }
});

sp2.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent e) {
        repaint();
    }
});

```

En el método paint dibujamos primero un rectángulo de color azul que representa la hoja:

```

g.setColor(Color.blue);
g.drawRect(30,80,100,140);

```

Extraemos los valores seleccionados de cada control JSpinner y los convertimos a tipo entero:

```

int ms=Integer.parseInt(sp1.getValue().toString());
int mi=Integer.parseInt(sp2.getValue().toString());

```

Activamos el color rojo y dibujamos las dos líneas, la superior coincide con el comienzo del rectángulo (sumamos tantos pixeles en la fila como lo indica el primer JSpinner):

```

g.setColor(Color.red);
g.drawLine(30,80+ms,130,80+ms);

```

La segunda línea le restamos el valor del JSpinner:

```

g.drawLine(30,220-mi,130,220-mi);

```

Para saber la orientación de la hoja debemos extraer el valor seleccionado del JComboBox y mediante un if verificar si el String seleccionado es "Horizontal":

```

String direccion=(String)comboBox.getSelectedItem();
if (direccion.equals("Horizontal"))
    g.drawRect(320,120,200,100 );
else
    g.drawRect(320,120,100,200 );

```

Por último cuando se presiona el botón inicializar procedemos a fijar nuevos valores a los JSpinner y al JComboBox (luego redibujamos):

```
btnInicializar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        sp1.setValue(0);
        sp2.setValue(0);
        comboBox.setSelectedIndex(0);
        repaint();
    }
});
```

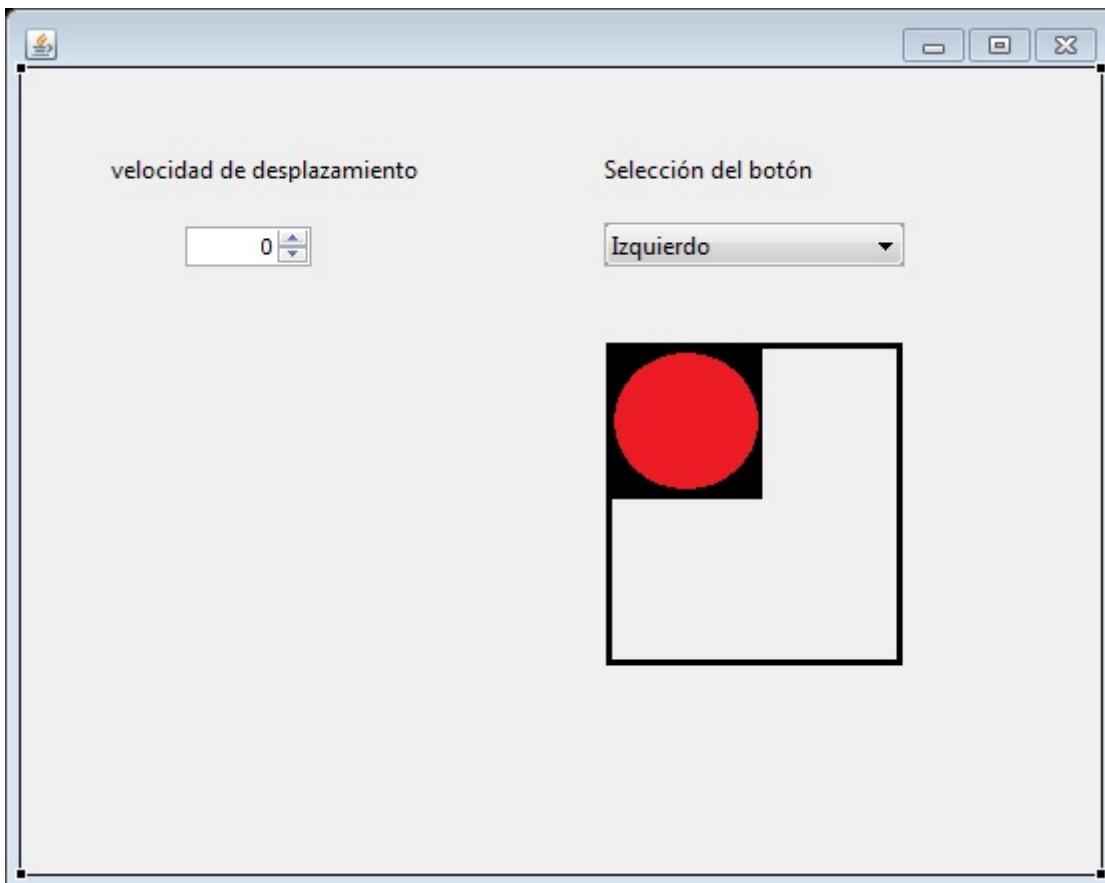
## Problemas propuestos

1. Confeccionar un programa que permita configurar las características del mouse.

Por un lado debemos seleccionar la velocidad de desplazamiento de la flecha del mouse. Disponer un JSpinner para poder seleccionarse los valores 0,25,50,75 y 100.

Por otro lado debemos poder seleccionar cual de los dos botones del mouse será el principal, tenemos para esta función un JComboBox con dos opciones: izquierdo o derecho.

Cuando se selecciona el botón (cambio en el JComboBox) actualizar el gráfico mostrando el botón del mouse seleccionado (graficar en el método paint el mouse en pantalla)



2. En una aduana hay una máquina que sortea las personas cuyo equipaje serán revisados.

La persona selecciona si viene del Interior del país o del Exterior (a través de un JComboBox), y por otro lado selecciona la cantidad de bultos (JSpinner).

Luego presiona el botón sortear y aparece al lado de este botón un círculo rojo o verde. (En caso de ser rojo se revisa su equipaje, en caso de ser verde, no se revisa)

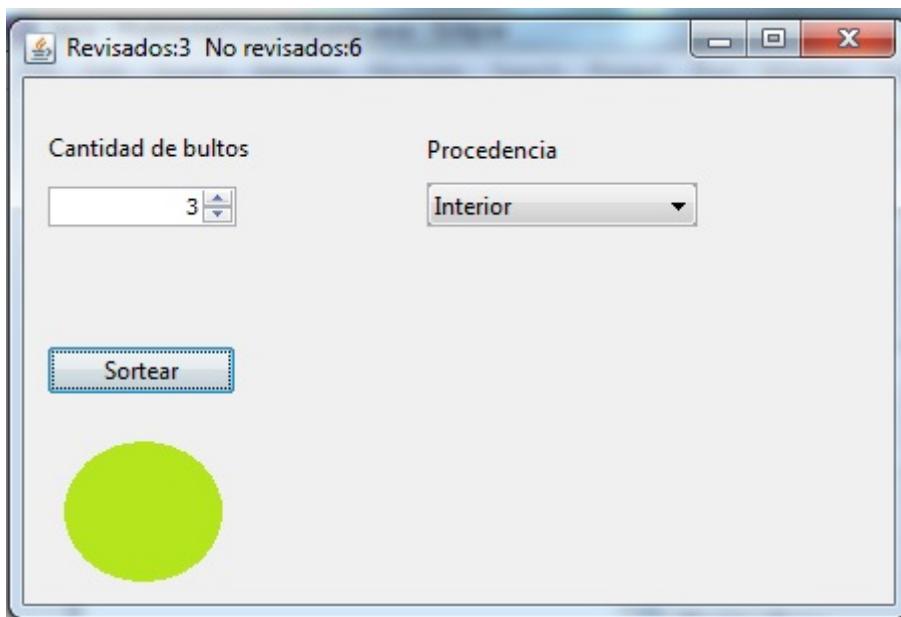
Para el sorteo generar un valor aleatorio entre 1 y 3. Si se genera un 1 se revisa, si se genera un 2 o 3 no se revisa.

Validar que también este seleccionado un valor distinto a cero en bultos (los valores pueden ir de 0 a 10).

Si la cantidad de bultos supera a 5 se revisa siempre sus bultos (es decir que aparece un círculo rojo).

Luego de sortear fijar en cero cantidad de bultos.

Mostrar en el título del JFrame la cantidad de bultos revisados y no revisados hasta el momento.



[Retornar](#)

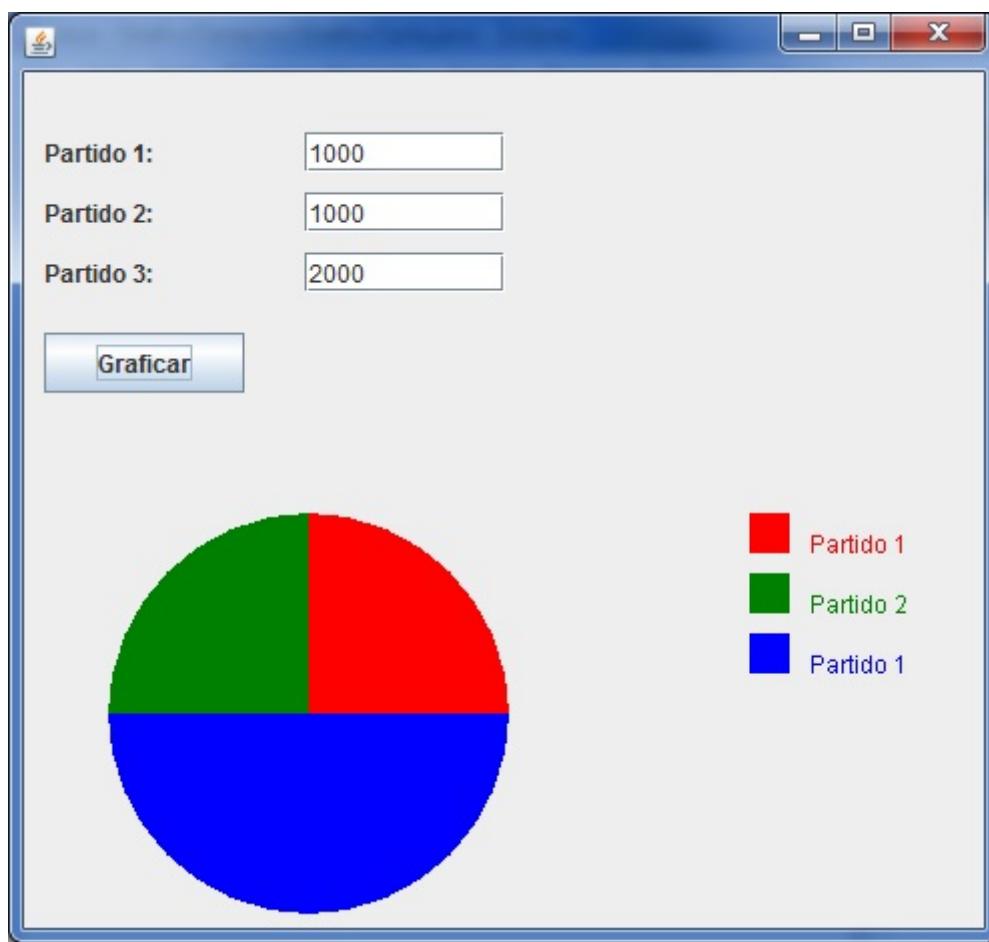
# 58 - Gráficos estadísticos

[Listado completo de tutoriales](#)

El objetivo de este concepto es la implementación de algoritmos para mostrar gráficos estadísticos.

## Problema 1

Crear una aplicación que solicite el ingreso de tres valores por teclado que representan las cantidades de votos obtenidas por tres partidos políticos. Luego mostrar un gráfico de tartas:



El algoritmo es:

```
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Graphics;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel.
```

```
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class GraficoTarta extends JFrame {

    private JPanel contentPane;
    private JTextField tf1;
    private JTextField tf2;
    private JTextField tf3;
    private boolean bandera=false;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    GraficoTarta frame = new GraficoTarta();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public GraficoTarta() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 800, 600);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblPartido = new JLabel("Partido 1:");
        lblPartido.setBounds(46, 39, 61, 14);
        contentPane.add(lblPartido);

        JLabel lblPartido_1 = new JLabel("Partido 2:");
        lblPartido_1.setBounds(46, 69, 61, 14);
        contentPane.add(lblPartido_1);

        JLabel lblPartido_2 = new JLabel("Partido 3:");
        lblPartido_2.setBounds(46, 103, 61, 14);
        contentPane.add(lblPartido_2);

        tf1 = new JTextField();
        tf1.setBounds(117, 36, 86, 20);
        contentPane.add(tf1);
        tf1.setColumns(10);

        tf2 = new JTextField();
        tf2.setBounds(117, 66, 86, 20);
        contentPane.add(tf2);
    }
}
```

```

contentPane.setLayout(null);
tf2.setColumns(10);

tf3 = new JTextField();
tf3.setBounds(117, 97, 86, 20);
contentPane.add(tf3);
tf3.setColumns(10);

JButton btnGraficar = new JButton("Graficar");
btnGraficar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        bandera=true;
        repaint();
    }
});
btnGraficar.setBounds(45, 138, 107, 37);
contentPane.add(btnGraficar);
}

public void paint(Graphics g)
{
    super.paint(g);
    if (bandera==true)
    {
        String s1=tf1.getText();
        String s2=tf2.getText();
        String s3=tf3.getText();
        int v1=Integer.parseInt(s1);
        int v2=Integer.parseInt(s2);
        int v3=Integer.parseInt(s3);
        int suma=v1+v2+v3;
        int grados1=v1*360/suma;
        int grados2=v2*360/suma;
        int grados3=v3*360/suma;

        g.setColor(new Color(255,0,0));
        g.fillArc(50,250,200,200,0,grados1);
        g.fillRect(370,250,20,20);
        g.drawString("Partido 1", 400, 270);

        g.setColor(new Color(0,128,0));
        g.fillArc(50,250,200,200,grados1,grados2);
        g.fillRect(370,280,20,20);
        g.drawString("Partido 2", 400, 300);

        g.setColor(new Color(0,0,255));
        g.fillArc(50,250,200,200,grados1+grados2,grados3);
        g.fillRect(370,310,20,20);
        g.drawString("Partido 1", 400, 330);
    }
}
}

```

Disponemos un if en el método paint para controlar que se haya presionado el botón graficar:

```
public void paint(Graphics g)
{
    super.paint(g);
    if (bandera==true)
    {
```

El atributo bandera se inicializa cuando se define con el valor false, esto hace que cuando se ejecute por primera vez el método paint no ingrese al if:

```
private boolean bandera=false;
```

Se definen tres objetos de la clase JTextField para ingresar los tres valores por teclado:

```
private JTextField tf1;
private JTextField tf2;
private JTextField tf3;
```

Cuando se presiona el botón se cambia el estado del atributo bandera por el valor true y se llama al método repaint (recordemos que este método borra el JFrame y llama al método paint para que ahora dibuje el gráfico de tarta):

```
btnGraficar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        bandera=true;
        repaint();
    }
});
```

Lo primero que hacemos para graficar la tarta es rescatar los tres valores ingresados en los controles JTextField:

```
if (bandera==true)
{
    String s1=tf1.getText();
    String s2=tf2.getText();
    String s3=tf3.getText();
```

Los convertimos a tipo de dato entero:

```
int v1=Integer.parseInt(s1);
int v2=Integer.parseInt(s2);
int v3=Integer.parseInt(s3);
```

Sumamos los tres valores:

```
int suma=v1+v2+v3;
```

Seguidamente calculamos los grados que le corresponde a cada trozo de tarta (teniendo en cuenta que tenemos 360 grados para repartir):

Cada trozo de tarta lo obtenemos mediante la ecuación:

tamaño trozo= cantidad de votos del partido/ total de votos \* 360

si observamos la ecuación podemos imaginar que la división:

cantidad de votos del partido / total de votos

generará un valor menor a uno (salvo que el partido haya obtenido todos los votos de la elección en cuyo caso la división genera el valor uno)

Esta división nos genera el porcentaje de votos que le corresponde al partido y luego dicho porcentaje lo multiplicamos por la cantidad de grados a repartir (que son 360 grados)

Luego como la división generará un valor menor a uno y al tratarse de dos variables enteras el resultado será cero. Para evitar este problema procedemos primero a multiplicar por 360 y luego dividir por la variable suma:

```
int grados1=v1*360/suma;
int grados2=v2*360/suma;
int grados3=v3*360/suma;
```

Si quisiéramos primero dividir y luego multiplicar por 360 debemos proceder a anteceder a cada operación el tipo de resultado a obtener:

```
int grados1=(int)((float)v1/suma*360);
int grados2=(int)((float)v2/suma*360);
int grados3=(int)((float)v3/suma*360);
```

Como podemos ver es más complicado si queremos primero efectuar la división y luego el producto.

Procedemos ahora a graficar los trozos de tarta y leyenda con el valor ingresado (activamos el color rojo y mediante el método fillArc creamos un trozo de tarta que se inicia en el grado 0 y avanza tantos grados como indica la variable grados1. Luego mediante el método drawString mostramos la leyenda del partido respectivo con un cuadradito también de color rojo):

```
g.setColor(new Color(255,0,0));
g.fillArc(50,250,200,200,0,grados1);
g.fillRect(370,250,20,20);
g.drawString("Partido 1", 400, 270);
```

El segundo trozo comienza en grados1 y avanza tantos grados como lo indica la variable grados2. Activamos el color verde para diferenciar del trozo anterior:

```

g.setColor(new Color(0,128,0));
g.fillArc(50,250,200,200,grados1,grados2);
g.fillRect(370,280,20,20);
g.drawString("Partido 2", 400, 300);

```

El último trozo lo graficamos a partir de la suma de grados1+grados2 y avanzamos tantos grados como lo indica la variable grados3:

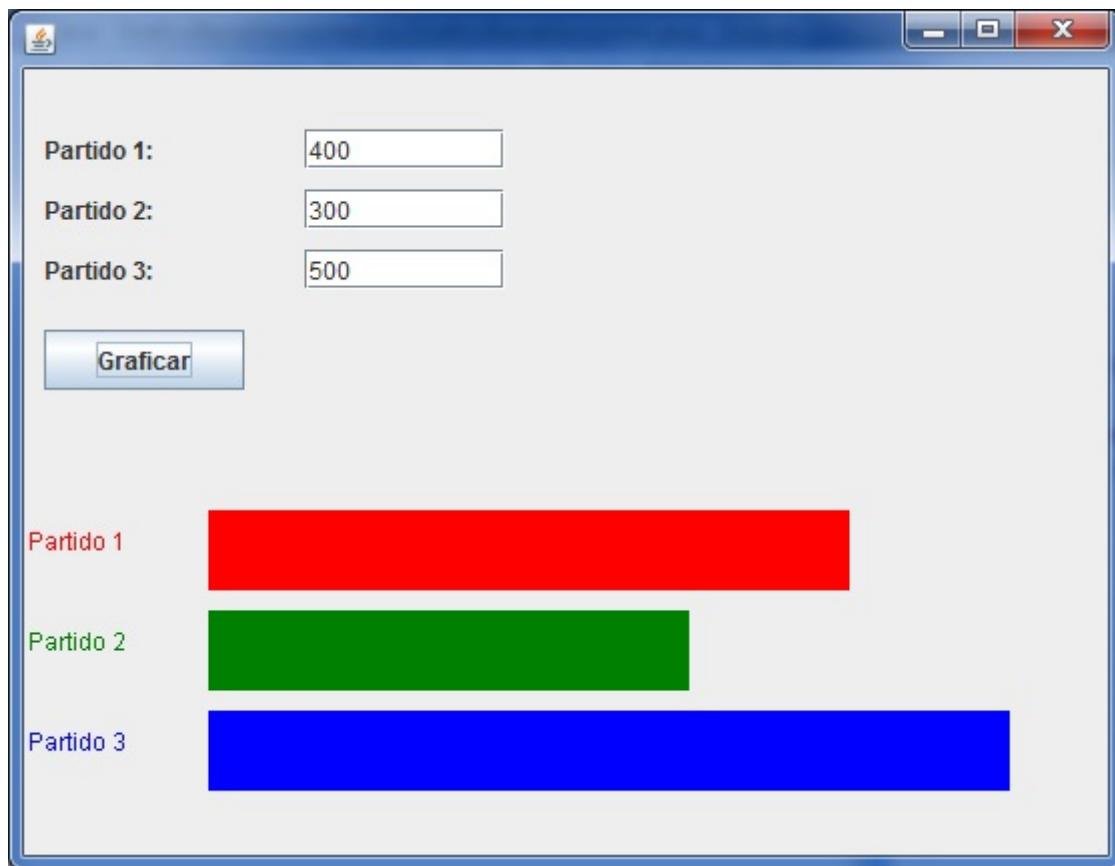
```

g.setColor(new Color(0,0,255));
g.fillArc(50,250,200,200,grados1+grados2,grados3);
g.fillRect(370,310,20,20);
g.drawString("Partido 1", 400, 330);

```

## Problema 2

Crear una aplicación que solicite el ingreso de tres valores por teclado que representan las cantidades de votos obtenidas por tres partidos políticos. Luego mostrar un gráfico de barras horizontales:



El algoritmo es:

```

import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Graphics;

import javax.swing.JFrame;
import javax.swing.JPanel;

```

```
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class GraficoBarraHorizontal extends JFrame {

    private JPanel contentPane;
    private JTextField tf3;

    private JTextField tf1;
    private JTextField tf2;
    private boolean bandera=false;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    GraficoBarraHorizontal frame = new
GraficoBarraHorizontal();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public GraficoBarraHorizontal() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 800, 600);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblPartido = new JLabel("Partido 1:");
        lblPartido.setBounds(46, 39, 61, 14);
        contentPane.add(lblPartido);

        JLabel lblPartido_1 = new JLabel("Partido 2:");
        lblPartido_1.setBounds(46, 69, 61, 14);
        contentPane.add(lblPartido_1);

        JLabel lblPartido_2 = new JLabel("Partido 3:");
        lblPartido_2.setBounds(46, 103, 61, 14);
        contentPane.add(lblPartido_2);

        tf1 = new JTextField();
        tf1.setBounds(117, 36, 86, 20);
        contentPane.add(tf1);
        tf1.setColumns(10);
    }
}
```

```

        tf2 = new JTextField();
        tf2.setBounds(117, 66, 86, 20);
        contentPane.add(tf2);
        tf2.setColumns(10);

        tf3 = new JTextField();
        tf3.setBounds(117, 97, 86, 20);
        contentPane.add(tf3);
        tf3.setColumns(10);

        JButton btnGraficar = new JButton("Graficar");
        btnGraficar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                bandera=true;
                repaint();
            }
        });
        btnGraficar.setBounds(45, 138, 107, 37);
        contentPane.add(btnGraficar);
    }

    public void paint(Graphics g)
    {
        super.paint(g);
        if (bandera==true)
        {
            String s1=tf1.getText();
            String s2=tf2.getText();
            String s3=tf3.getText();
            int v1=Integer.parseInt(s1);
            int v2=Integer.parseInt(s2);
            int v3=Integer.parseInt(s3);
            int mayor=retornarMayor(v1,v2,v3);

            int largo1=v1*400/mayor;
            int largo2=v2*400/mayor;
            int largo3=v3*400/mayor;

            g.setColor(new Color(255,0,0));
            g.fillRect(100,250,largo1,40);
            g.drawString("Partido 1", 10, 270);

            g.setColor(new Color(0,128,0));
            g.fillRect(100,300,largo2,40);
            g.drawString("Partido 2", 10, 320);

            g.setColor(new Color(0,0,255));
            g.fillRect(100,350,largo3,40);
            g.drawString("Partido 3", 10, 370);
        }
    }

    private int retornarMayor(int v1,int v2,int v3)
    {
        if (v1>v2 && v1>v3)
            return v1;
        else
            if (v2>v3)
                return v2;
    }
}

```

```

        else
            return v3;
    }

}

```

La metodología es similar al problema anterior. Ahora no tenemos grados para repartir, por lo que implementaremos el siguiente algoritmo:

Consideraremos que el partido que obtuvo más votos le corresponde una barra de 400 píxeles de largo y los otros dos partidos se les entregará en forma proporcional.

Lo primero que hacemos es obtener el mayor de los tres valores ingresados por teclado, para esto implementamos un método privado que retorne el mayor de tres valores enteros:

```

private int retornarMayor(int v1,int v2,int v3)
{
    if (v1>v2 && v1>v3)
        return v1;
    else
        if (v2>v3)
            return v2;
        else
            return v3;
}

```

El el método paint llamamos al método retornarMayor:

```
int mayor=retornarMayor(v1,v2,v3);
```

La ecuación para obtener el largo de la barra será:

largo=votos del partido/votos del partido con mas votos \* 400 píxeles.

Como podemos ver esta división generará un valor menor a uno salvo para el partido que tiene más votos (en este caso la división genera el valor 1) luego multiplicamos por 400.

Comenzamos a calcular el largo de cada rectángulo:

```
int largo1=v1*400/mayor;
```

Como podemos ver primero multiplicamos por 400 y lo dividimos por el mayor de los tres valores ingresados.

Nuevamente primero multiplicamos y luego dividimos con el objetivo que el resultado de la división no nos redondee a cero.

El primer trozo lo graficamos a partir de la columna 100, fila 250 y con un ancho indicado en la variable largo1 (el alto de la barra es de 40 píxeles):

```
g.setColor(new Color(255,0,0));
```

```
g.fillRect(100,250,largo1,40);
g.drawString("Partido 1", 10, 270);
```

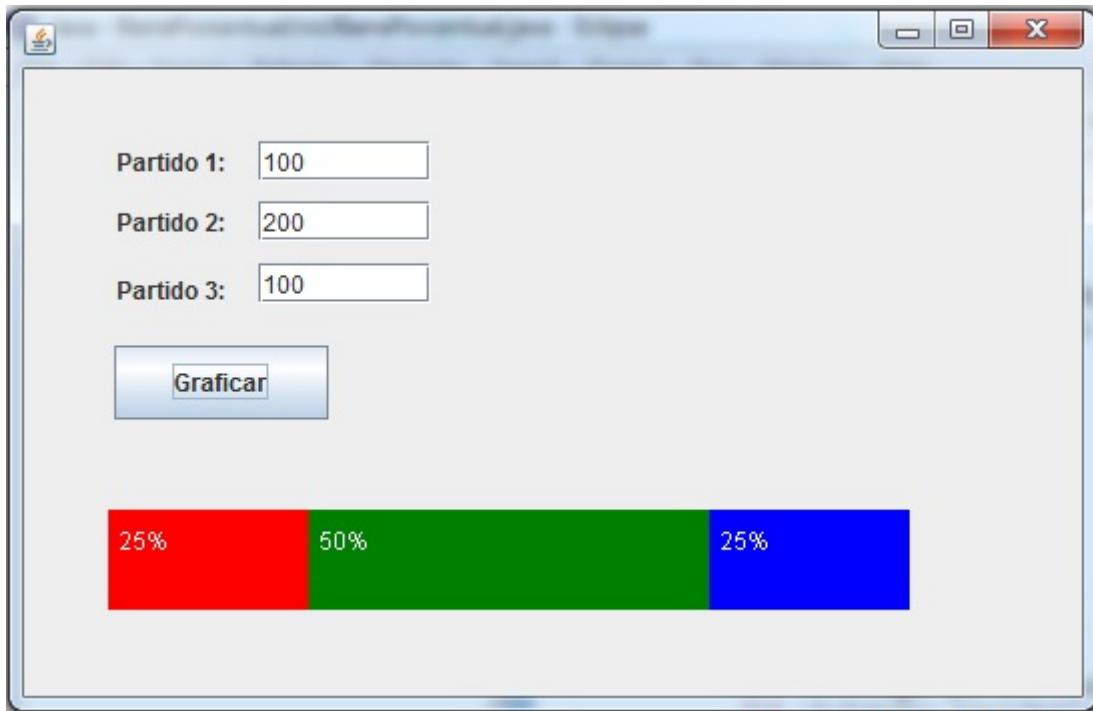
En forma similar graficamos los otros dos trozos de barra:

```
g.setColor(new Color(0,128,0));
g.fillRect(100,300,largo2,40);
g.drawString("Partido 2", 10, 320);

g.setColor(new Color(0,0,255));
g.fillRect(100,350,largo3,40);
g.drawString("Partido 3", 10, 370);
```

## Problema propuesto

1. Implementar un gráfico estadístico de tipo "Barra Porcentual":



[Solución](#)

[Retornar](#)

JDBC son las siglas en inglés de Java Database Connectivity. Es un conjunto de clases que nos permite acceder a diversos gestores de bases de datos de forma transparente.

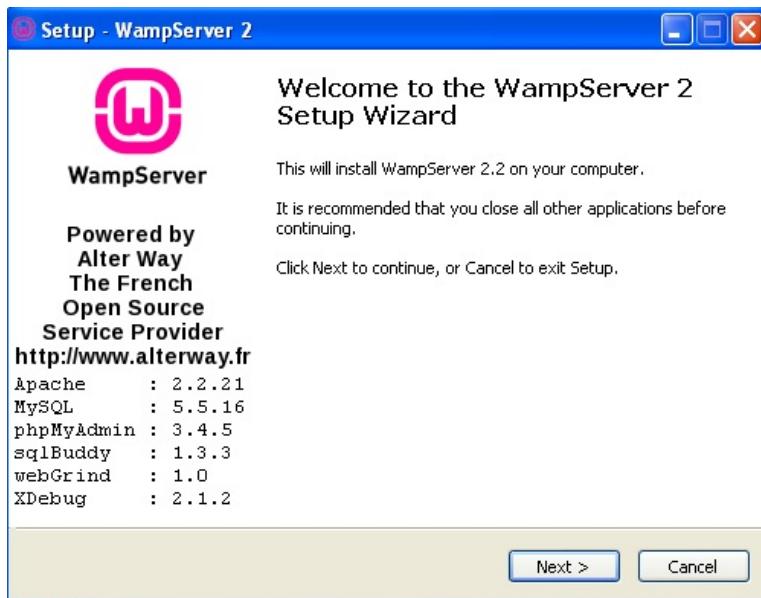
Veremos como conectarnos con el motor de base de datos MySQL.

## Instalación de MySQL integrado en el WampServer

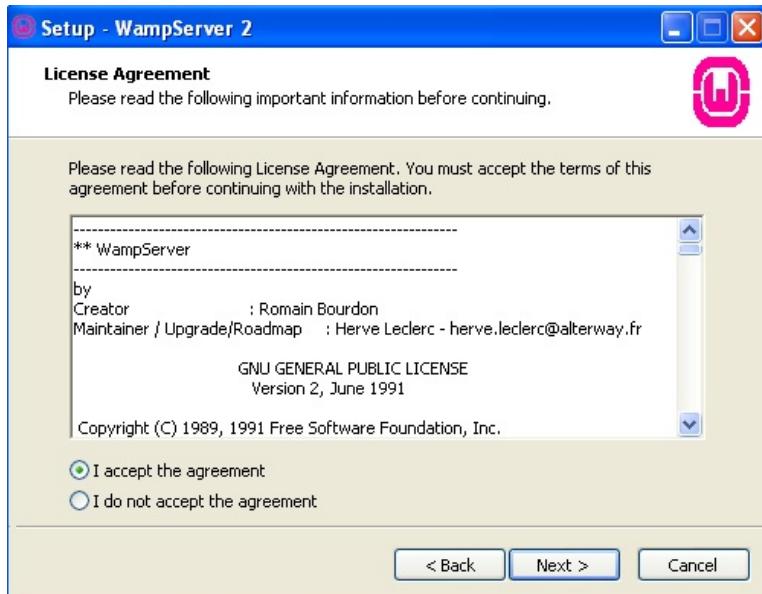
Utilizaremos esta herramienta en lugar de descargar solo el MySQL con la finalizar de facilitar la instalación y configuración del motor de base de datos (la instalación de esta herramienta es sumamente sencilla), además utilizaremos otra software que provee el WampServer que es el PhpMyAdmin que nos facilitará la creación de la base de datos.

Procedemos a descargar el WampServer de la siguiente página: [aquí](#).

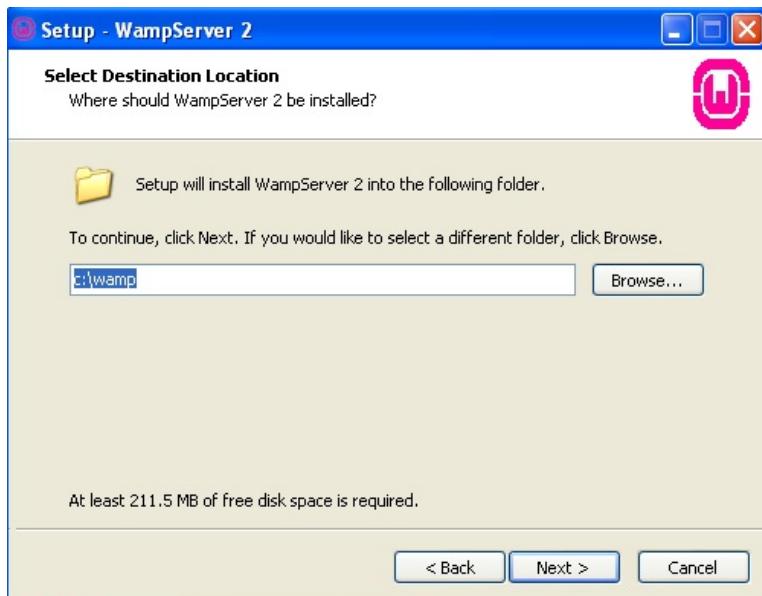
Luego de descargarlo procedemos a ejecutar el instalador:



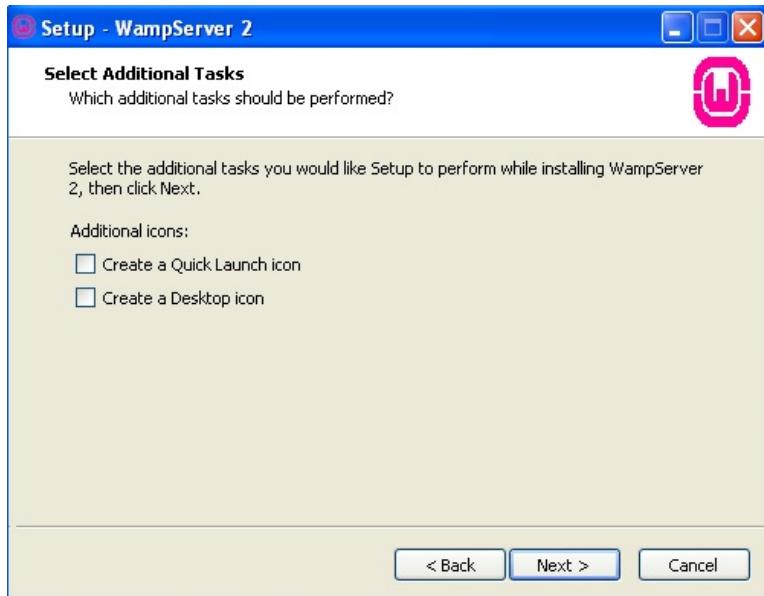
Aceptamos los términos y condiciones:



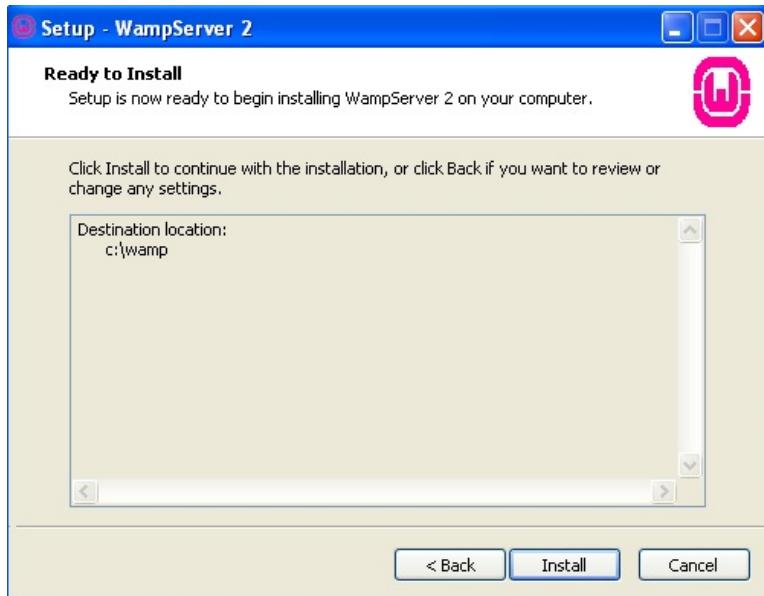
Elegimos el directorio donde se instalará":



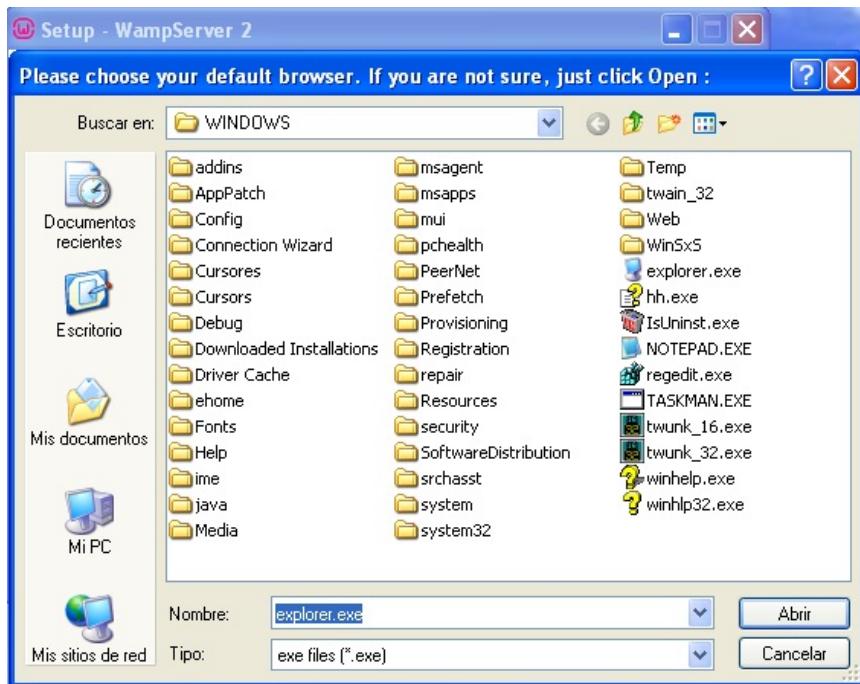
Indicamos si queremos que se cree un ícono en el escritorio:



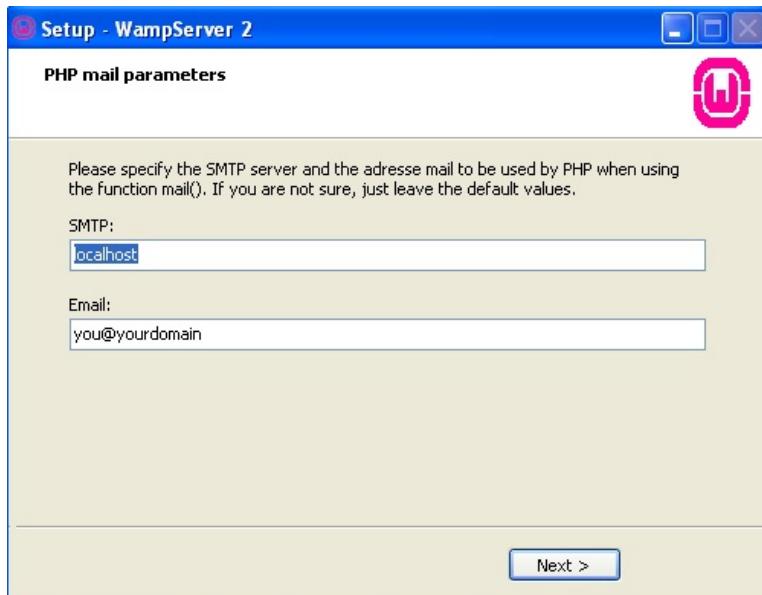
Procedemos a presionar el botón "Install":



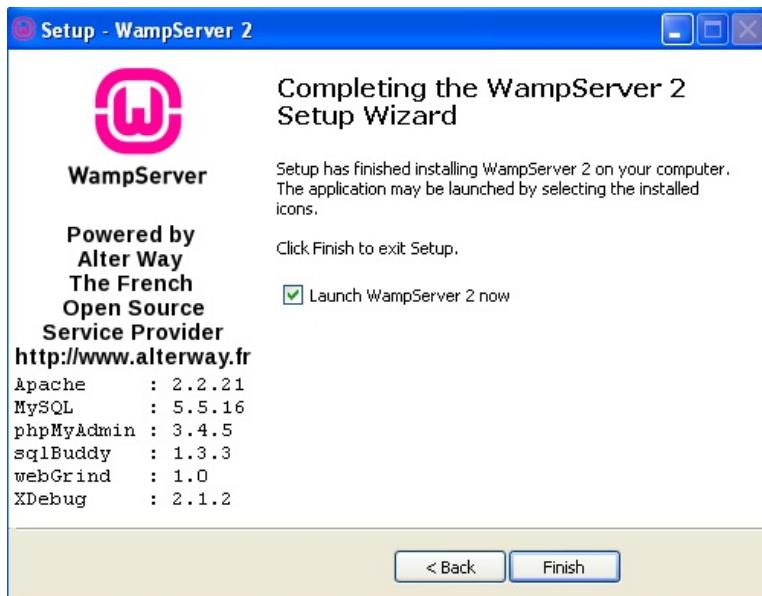
Luego de instalarse nos solicita que navegador abrirá por defecto cuando ejecutemos el PhpMyAdmin (para la creación de la base de datos de MySQL):



En el siguiente diálogo dejamos los datos por defecto:



Finalmente aparece el diálogo final donde se nos informa que se iniciará el WampServer (es decir que se cargará en memoria entre otras cosas el MySQL) :



Ahora podemos ver el iconos del WampServer en la bandeja del sistema de Windows (si se encuentra en color verde significa que el MySQL está ejecutándose correctamente):



### Ejecución del PhpMyAdmin para la creación de la base de datos.

Haciendo clic sobre el ícono de la bandeja del sistema aparece un menú que nos permite lanzar el PhpMyAdmin para crear la base de datos de MySQL:



El PhpMyAdmin es un programa web que nos permite administrar las bases de datos del MySQL:

The screenshot shows the phpMyAdmin 3.4.5 configuration interface. The top navigation bar includes the PMA logo, the URL 'localhost / localhost | phpMyAdmin', and a search bar with the placeholder 'localhost/phpmyadmin/'. The main menu on the left lists databases: 'information\_schema', 'mysql', 'performance\_schema', and 'test'. The central configuration area is divided into several panels:

- Configuraciones generales**: Shows the character set for MySQL connections set to 'utf8\_general\_ci'.
- Configuraciones de apariencia**: Shows the language set to 'Español - Spanish', the theme set to 'pmahomme', and the font size set to '82%'. It also includes a 'Más configuraciones' link.
- MySQL**: Displays system information:
  - Servidor: localhost (localhost via TCP/IP)
  - Versión del servidor: 5.5.16-log
  - Versión del protocolo: 10
  - Usuario: root@localhost
  - Juegos de caracteres de MySQL: UTF-8 Unicode (utf8)
- Servidor web**: Displays information about the web server:
  - Apache/2.2.21 (Win32)
  - PHP/5.3.8
  - Versión del cliente: mysqlnd 5.0.8-dev - 20102224 - \$Revision: 310735 \$
  - extensión PHP: mysqli
- phpMyAdmin**: Shows links to the version (3.4.5), documentation, wiki, official page, contribute, support, and change log.

Seleccionamos la pestaña "Base de datos" y donde dice "Crear nueva base de datos" especificamos que nuestra base de datos se llamará "bd1":

The screenshot shows the phpMyAdmin interface for a MySQL database. The left sidebar lists existing databases: information\_schema, mysql, performance\_schema, and test. The main panel is titled 'Bases de datos' (Databases). A red circle highlights the 'Create nueva base de datos' (Create new database) input field, which contains 'bd1'. Below this, a dropdown menu shows 'Cotejamiento' (Comparison) is selected. A 'Crear' (Create) button is to the right of the dropdown. The database list table has two columns: 'Base de datos' (Database) and 'Replicación maestra' (Master replication). The table shows the following data:

| Base de datos      | Replicación maestra  |
|--------------------|--|
| information_schema | ✓ Replicado/a <input type="button" value="Comprobar los privilegios"/> |
| mysql              | ✓ Replicado/a <input type="button" value="Comprobar los privilegios"/> |
| performance_schema | ✓ Replicado/a <input type="button" value="Comprobar los privilegios"/> |
| test               | ✓ Replicado/a <input type="button" value="Comprobar los privilegios"/> |
| <b>Total: 4</b>    |  |

Below the table, there are buttons for 'Marcar todos / Desmarcar todos' (Select all / Deselect all) and 'Eliminar' (Delete). A 'Activar las estadísticas' (Enable statistics) button is also present with a warning message: 'Nota: Activar aquí las estadísticas de la base de datos podría causar tráfico pesado entre el servidor web y el servidor MySQL.' (Note: Enabling statistics here could cause heavy traffic between the web server and the MySQL server.)

Presionamos el botón "crear" y con esto ya tenemos nuestra base de datos creada:

localhost / localhost | phpMyAdmin

localhost

Bases de datos SQL Estado actual Registro binario Más

bd1

information\_schema

mysql

performance\_schema

test

Crear nueva base de datos

bd1 Cotejamiento

Base de datos Replicación maestra

| Base de datos      | Replicación maestra |
|--------------------|---------------------|
| bd1                | ✓ Replicado/a       |
| information_schema | ✓ Replicado/a       |
| mysql              | ✓ Replicado/a       |
| performance_schema | ✓ Replicado/a       |
| test               | ✓ Replicado/a       |

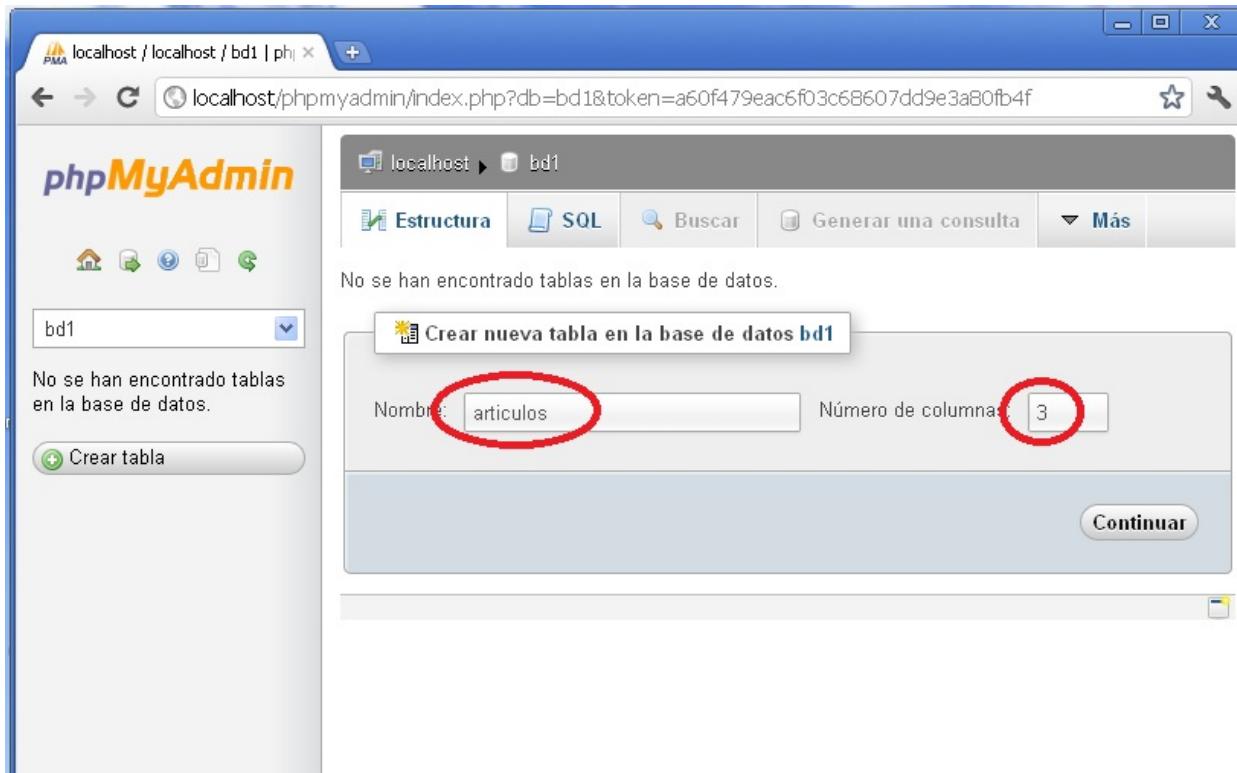
Total: 5

↑ Marcar todos / Desmarcar todos Para los elementos que están marcados: Eliminar

Activar las estadísticas

Nota: Activar aquí las estadísticas de la base de datos podría causar tráfico pesado entre el servidor web y el servidor MySQL.

Luego de seleccionar la base de datos "bd1" que figura a la izquierda procedemos a crear la primer tabla que contendrá (crearemos una tabla llamada "articulos" y que tendrá tres campos):



localhost / localhost / bd1 | ph| x

localhost/phpmyadmin/index.php?db=bd1&token=a60f479eac6f03c68607dd9e3a80fb4f

**phpMyAdmin**

Estructura SQL Buscar Generar una consulta Más

No se han encontrado tablas en la base de datos.

bd1

No se han encontrado tablas en la base de datos.

Crear tabla

Crear nueva tabla en la base de datos bd1

Nombre: **articulos** Número de columnas: **3**

Continuar

En la tabla "articulos" definimos el campo "codigo" de tipo int (este campo será el "primary key" y auto\_increment lo tildamos para que el código se genere automáticamente), el segundo campo es la descripción que es de tipo varchar con un máximo de 50 caracteres y por último el campo precio que es de tipo float.

Luego de especificar los tres campos en la parte inferior de la misma ventana aparece un botón llamada "Guardar" para confirmar la estructura de la tabla:



En el lado izquierdo del navegador podemos ver ahora que la base de datos "bd1" tiene una tabla llamada "articulos"

Hasta acá lo que nos ayuda el PhpMyAdmin (es decir creación de la base de datos y la tabla), de ahora en más todas las otras actividades las desarrollaremos desde nuestro programa en java (poblar o insertar datos, listar registros, consultar, modificar y borrar datos)

### **Descarga del Driver para permitir conectar nuestro programa Java con el MySQL**

Como última actividad de configuración previo a implementar los programas en Java para acceder a MySQL es la descarga del Driver que nos permita conectarnos con la base de datos.

El Driver lo podemos descargar de la página:[aquí](#)

Podemos descomprimir el archivo mysql-connector-java-5.1.18.zip que acabamos de descargar. Luego veremos que en nuestro programa en java haremos referencias al archivo mysql-connector-java-5.1.18-bin.jar (que es el Driver propiamente dicho)

**[Retornar](#)**

# 60 - Alta y Consulta de una tabla de MySQL

[Listado completo de tutoriales](#)

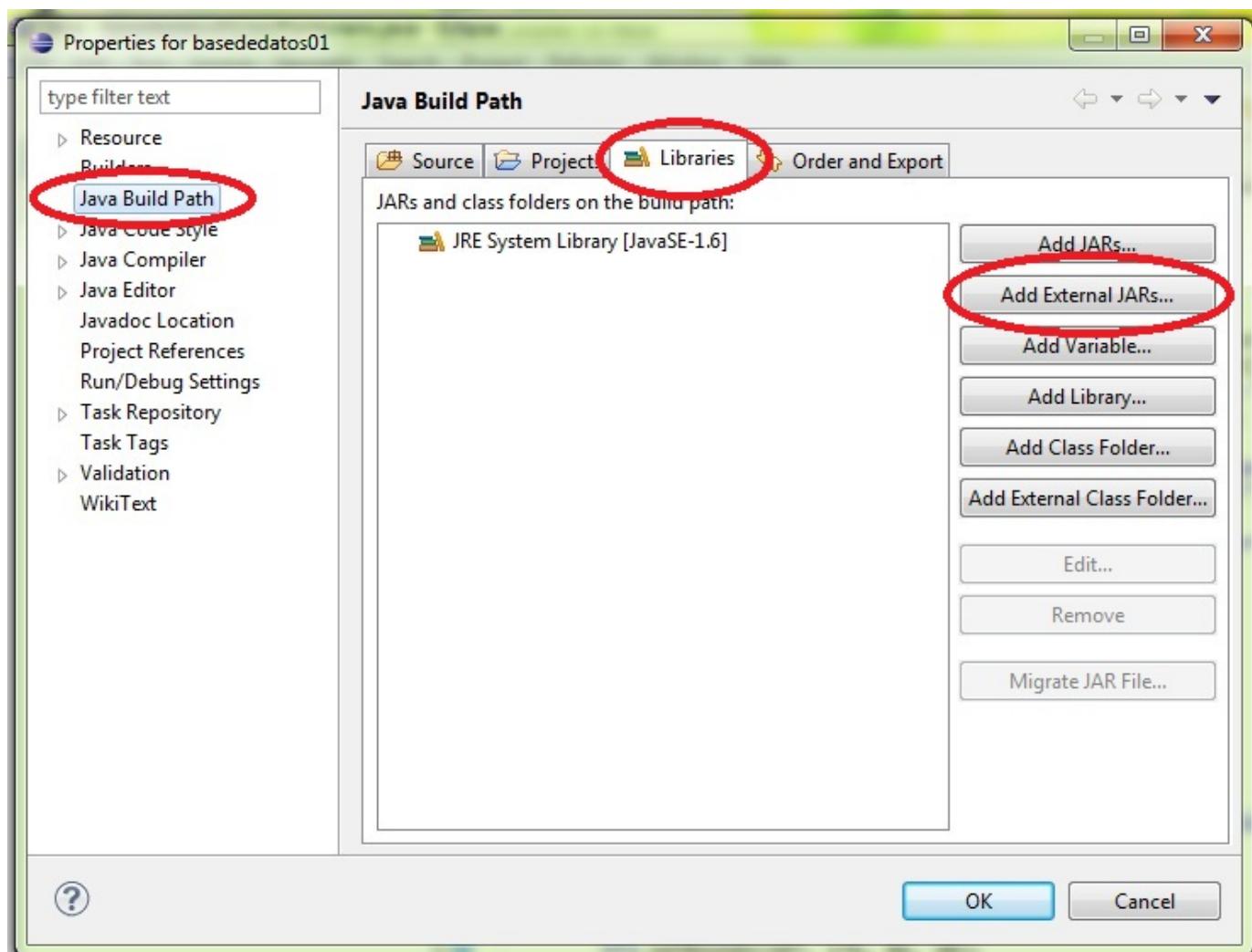
## Problema 1

Ya creamos en el concepto anterior una base de datos llamada bd1 y en la misma creamos una tabla llamada articulos.

Procederemos a implementar en Java un programa que nos permita comunicarnos con la base de datos "bd1" e insertar filas en la tabla "articulos" y posteriormente consultar su contenido.

1 - Creamos desde Eclipse un proyecto llamado "basededatos01" y seguidamente con el WindowBuilder creamos una clase llamada "Formulario".

2 - Primero debemos añadir el driver que descargamos (mysql-connector-java-5.1.18-bin.jar) presionamos el botón derecho del mouse sobre nuestro proyecto y seleccionamos la opción "Properties", aparece el siguiente diálogo:



Seleccionamos la opción "Java Build Path", de la parte central seleccionamos la pestaña "Libraries" y procedemos a presionar el botón "Add External JARs...", donde procedemos a buscar el archivo mysql-connector-java-5.1.18-bin.jar

El código fuente de la clase formulario es:

```
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class Formulario extends JFrame {

    private JPanel contentPane;
    private JTextField tf1;
    private JTextField tf2;
    private JLabel labelResultado;
    private JButton btnConsultaPorCódigo;
    private JLabel lblIngreseCódigoDe;
    private JTextField tf3;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Formulario frame = new Formulario();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Formulario() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 606, 405);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblDescripciónDelArtículo = new JLabel("Descripción del artículo:");
        lblDescripciónDelArtículo.setBounds(23, 38, 193, 14);
        contentPane.add(lblDescripciónDelArtículo);
    }
}
```

```

tf1 = new JTextField();
tf1.setBounds(247, 35, 193, 20);
contentPane.add(tf1);
tf1.setColumns(10);

JLabel lblPrecio = new JLabel("Precio:");
lblPrecio.setBounds(23, 74, 95, 14);
contentPane.add(lblPrecio);

tf2 = new JTextField();
tf2.setBounds(247, 71, 107, 20);
contentPane.add(tf2);
tf2.setColumns(10);

JButton btnAlta = new JButton("Alta");
btnAlta.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        labelResultado.setText("");
        try {
            Connection
conexion=DriverManager.getConnection("jdbc:mysql://localhost/bd1","root" , "");
            Statement comando=conexion.createStatement();
            comando.executeUpdate("insert into articulos(descripcion,precio) values
('"+tf1.getText()+'','"+tf2.getText()+"')");
            conexion.close();
            labelResultado.setText("se registraron los datos");
            tf1.setText("");
            tf2.setText("");
        } catch(SQLException ex){
            setTitle(ex.toString());
        }
    }
});
btnAlta.setBounds(247, 118, 89, 23);
contentPane.add(btnAlta);

labelResultado = new JLabel("resultado");
labelResultado.setBounds(361, 122, 229, 14);
contentPane.add(labelResultado);

btnConsultaPorCdig = new JButton("Consulta por código");
btnConsultaPorCdig.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        labelResultado.setText("");
        tf1.setText("");
        tf2.setText("");
        try {
            Connection
conexion=DriverManager.getConnection("jdbc:mysql://localhost/bd1","root" , "");
            Statement comando=conexion.createStatement();
            ResultSet registro = comando.executeQuery("select descripcion,precio
from articulos where codigo='"+tf3.getText()+"');
            if (registro.next()==true) {
                tf1.setText(registro.getString("descripcion"));
                tf2.setText(registro.getString("precio"));
            } else {
                labelResultado.setText("No existe un artículo con dicho código");
            }
            conexion.close();
        } catch(SQLException ex){
            setTitle(ex.toString());
        }
    }
});
btnConsultaPorCdig.setBounds(23, 212, 177, 23);

```

```

contentPane.add(btnConsultaPorCdigo);

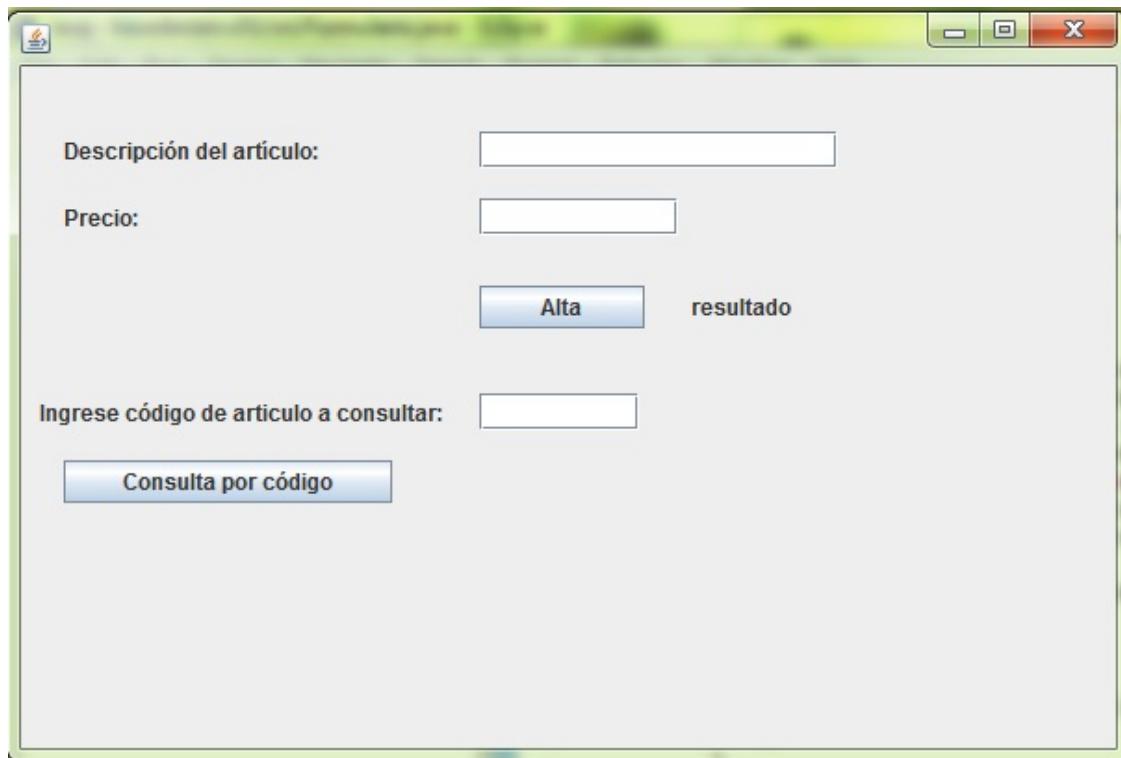
lblIngresarCdigoDe = new JLabel("Ingrese código de artículo a consultar:");
lblIngresarCdigoDe.setBounds(10, 179, 243, 14);
contentPane.add(lblIngresarCdigoDe);

tf3 = new JTextField();
tf3.setBounds(247, 176, 86, 20);
contentPane.add(tf3);
tf3.setColumns(10);
cargarDriver();
}

private void cargarDriver() {
    try {
        Class.forName("com.mysql.jdbc.Driver");
    } catch (Exception ex) {
        setTitle(ex.toString());
    }
}
}

```

La interfaz visual de la aplicación a implementar es la siguiente (se solicita el ingreso de la descripción del artículo y su precio, cuando se presiona el botón "Alta" se procede a insertar una fila en la tabla articulos de la base de datos bd1):



Expliquemos ahora el código fuente de la aplicación:

Primero debemos cargar en memoria el Driver, esto lo hacemos mediante el método cargarDriver que es llamado luego desde el constructor de la clase:

```

private void cargarDriver() {
    try {
        Class.forName("com.mysql.jdbc.Driver");
    }
}

```

```

        }catch(Exception ex) {
            setTitle(ex.toString());
        }
    }
}

```

Tenemos una clase llamada "Class" que tiene un método estático llamado forName, al mismo hay que pasar el nombre de la clase a importar:

```
Class.forName("com.mysql.jdbc.Driver");
```

com.mysql.jdbc es el nombre del paquete donde se encuentra la clase Driver. Esta es la forma que importamos los driver en Java.

El método forName de la clase Class genera excepciones de tipo Excepcion que deben ser capturadas obligatoriamente (luego por eso encerramos el código en un bloque try/catch).

Si no importamos el driver desde el diálogo Properties del proyecto o indicamos en forma incorrecta el nombre del paquete o clase luego aparece en el título del JFrame un mensaje del error sucedido.

Luego desde el constructor llamamos por única vez al método cargarDriver:

```

.....
tf3 = new JTextField();
tf3.setBounds(247, 176, 86, 20);
contentPane.add(tf3);
tf3.setColumns(10);
cargarDriver();
}

```

Veamos ahora cual es el código a implementar cuando se presiona el botón "Alta":

```

JButton btnAlta = new JButton("Alta");
btnAlta.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        labelResultado.setText("");
        try {
            Connection
conexion=DriverManager.getConnection("jdbc:mysql://localhost/bd1","root" , "");
            Statement comando=conexion.createStatement();
            comando.executeUpdate("insert into articulos(descripcion,precio) values
('"+tf1.getText()+"','"+tf2.getText()+"')");
            conexion.close();
            labelResultado.setText("se registraron los datos");
            tf1.setText("");
            tf2.setText("");
        } catch(SQLException ex){
            setTitle(ex.toString());
        }
    }
});

```

En el actionPerformed procedemos primero a limpiar la label que puede tener un mensaje de ejecuciones anteriores:

```
labelResultado.setText("");
```

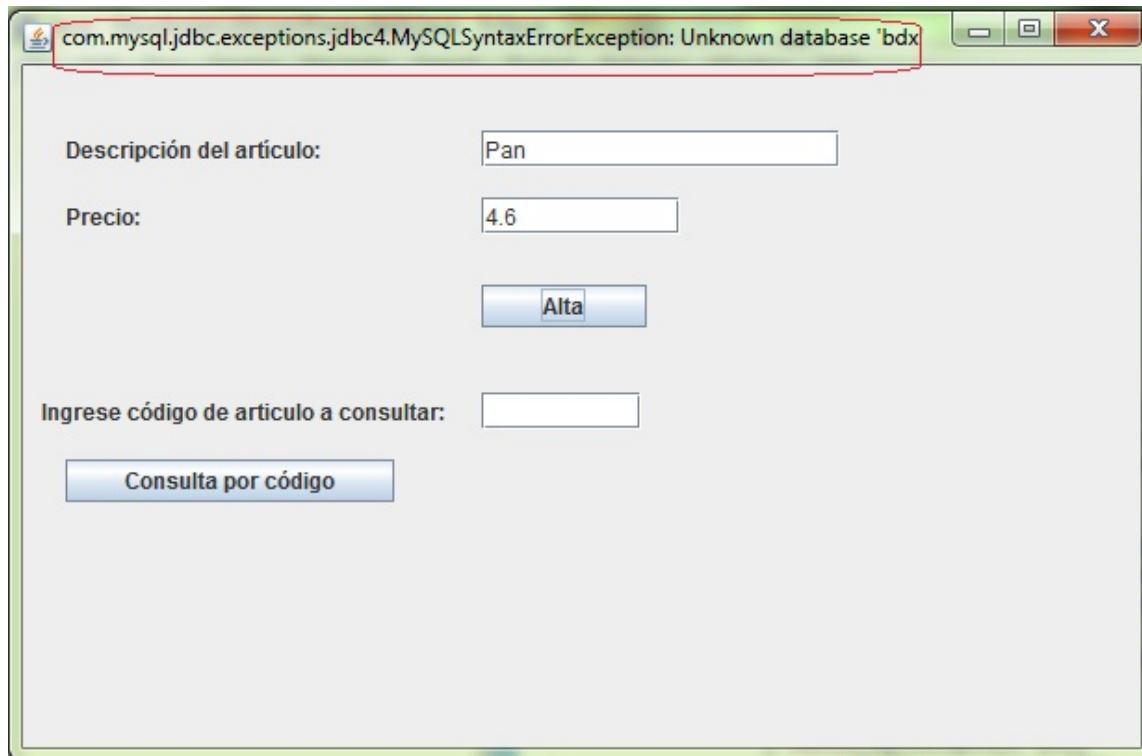
Todas las clases orientadas al acceso a base de datos generan excepciones de tipo SQLException y deben ser capturadas obligatoriamente. Lo primero que hacemos es crear un objeto de la clase Connection, para esto la clase DriverManager tiene un método llamado getConnection que retorna un objeto de la clase Connection:

```
Connection  
conexion=DriverManager.getConnection("jdbc:mysql://localhost/bd1","root" , "");
```

El método getConnection debemos pasarle tres String, el primero indica el nombre de la base de datos que queremos acceder (en este caso "bd1"), el segundo parámetro es el nombre de usuario (recordemos que cuando instalamos el MySQL se crea un usuario por defecto llamado "root") y el último parámetro es la clave del usuario "root", por defecto esta clave es un String vacío.

Como podemos ver también previo a la base de datos tenemos en la cadena de conexión el nombre de nuestro servidor (localhost)

Si nos equivocamos por ejemplo con el nombre de base de datos a comunicarnos (por ejemplo cambiar "bd1" por "bdx") veremos en el título del JFrame el mensaje de error que nos devuelve el MySQL:



Luego creamos un objeto de la clase Statement a partir del objeto de la clase Connection que acabamos de crear:

```
Statement comando=conexion.createStatement();
```

La clase Statement tiene un método llamado executeUpdate que le pasamos el comando SQL insert para agregar una fila a la tabla articulos:

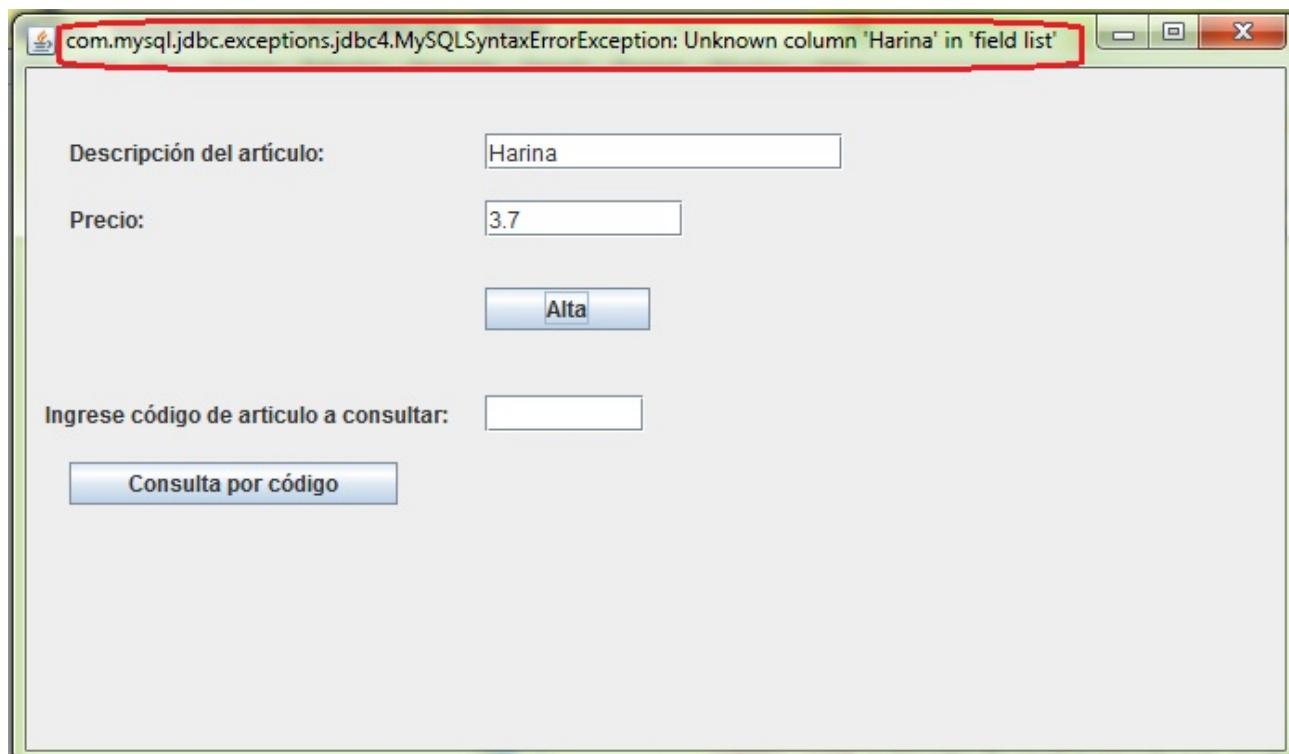
```
comando.executeUpdate("insert into articulos(descripcion,precio) values  
( '"+tf1.getText()+"','"+tf2.getText()+"')");
```

Como podemos ver generamos el String con el comando insert rescatando los datos de los dos controles de tipo JTextField. Es importante notar que en Java los String están encerrados entre comillas dobles y los contatenamos con el operador +. Las comillas simples son necesarias para los campos de tipo varchar de MySql (como podemos notar el lugar donde se dispondrá el texto de la descripción del artículo deben ir obligatoriamente las comillas simples):

```
... '"+tf1.getText()+"' ...
```

Si nos olvidamos las comillas simples al generar el String con el comando Insert el MySQL nos devolverá un error que será capturado por el try/catch, por ejemplo si lo ejecutamos con la siguiente sintaxis (sin las comillas simples envolviendo el valor de la descripción):

```
comando.executeUpdate("insert into articulos(descripcion,precio) values  
( "+tf1.getText()+"','"+tf2.getText()+"')");
```



Luego de solicitar la ejecución del comando Insert al MySQL procedemos a llamar al método close de la clase Connection:

```
conexion.close();
```

Con lo visto ya podemos agregar filas a la tabla articulos. Veamos ahora como consultar datos. El código a implementar cuando se presiona el botón "Consulta por código" es el siguiente:

```
btnConsultaPorCódigo = new JButton("Consulta por código");
btnConsultaPorCódigo.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        labelResultado.setText("");
        tf1.setText("");
        tf2.setText("");
        try {
            Connection
conexión=DriverManager.getConnection("jdbc:mysql://localhost/bd1","root" , "");
            Statement
comando=conexión.createStatement();
            ResultSet     registro     =
comando.executeQuery("select     descripción,precio     from     artículos     where
codigo='"+tf3.getText()+"');
            if (registro.next()==true) {
                tf1.setText(registro.getString("descripción"));
                tf2.setText(registro.getString("precio"));
            } else {
                labelResultado.setText("No existe
un artículo con dicho código");
            }
            conexión.close();
        } catch(SQLException ex){
            setTitle(ex.toString());
        }
    }
});
```

De forma similar al Insert procedemos a crear un objeto de la clase Connection y otro objeto de la clase Statement:

```
Connection
conexión=DriverManager.getConnection("jdbc:mysql://localhost/bd1","root" , "");
Statement
comando=conexión.createStatement();
```

Seguidamente definimos una variable de la clase ResultSet llamada registro y llamamos al método executeQuery de la clase Statement del objeto que acabamos de crear previamente:

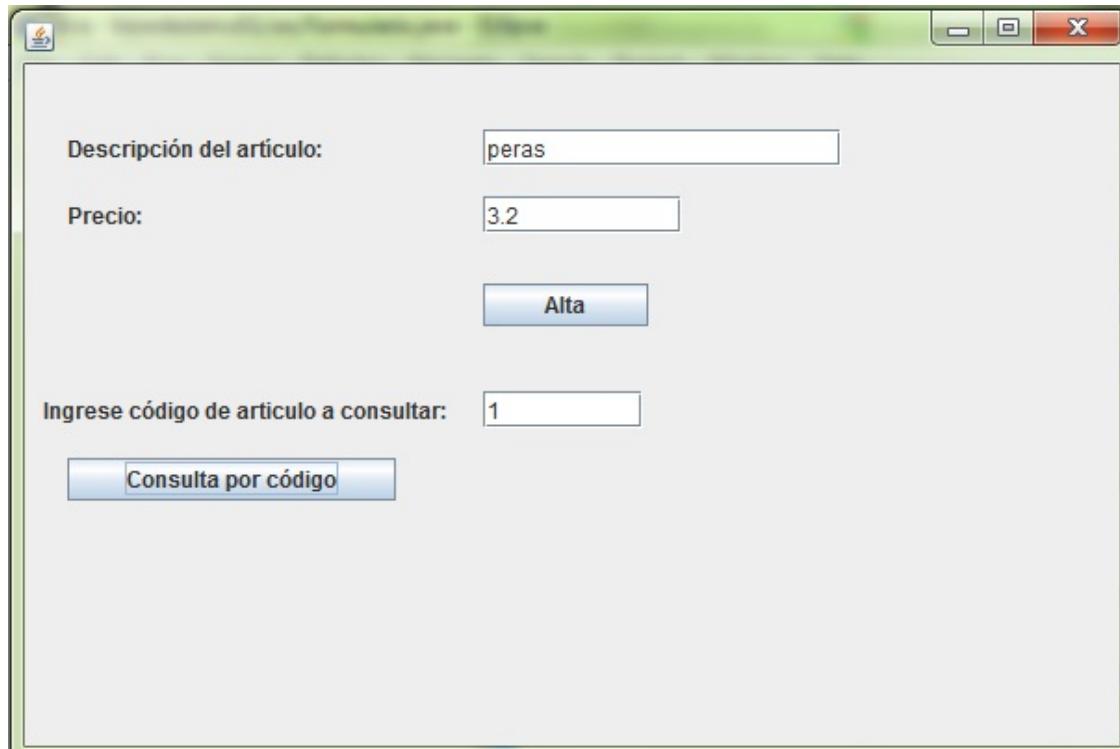
```
ResultSet     registro     =
comando.executeQuery("select     descripción,precio     from     artículos     where
codigo='"+tf3.getText()+"');
```

La clase ResultSet lo podemos imaginar como una tabla con todos los datos recuperados del comando SQL select que acaba de ejecutar el MySQL. En este ejemplo puede retornar una fila o ninguna ya que estamos utilizando la cláusula where y preguntando por el campo clave codigo.

Para acceder al registro devuelto debemos llamar al método next(), si retorna true es que si se recuperó una fila de la tabla artículos (es decir si existe el código de artículo ingresado), en

caso que retorne false el método next() significa que no hay un artículo con el código que ingresamos en el control JTextField:

```
if (registro.next()==true) {  
  
tf1.setText(registro.getString("descripcion"));  
  
tf2.setText(registro.getString("precio"));  
} else {  
    labelResultado.setText("No existe  
un artículo con dicho código");  
}
```



Este proyecto lo puede descargar en un zip desde este enlace: [basededatos01.zip](#)

[\*\*Retornar\*\*](#)

# 61 - Baja y modificación de datos de una tabla de MySQL

[Listado completo de tutoriales](#)

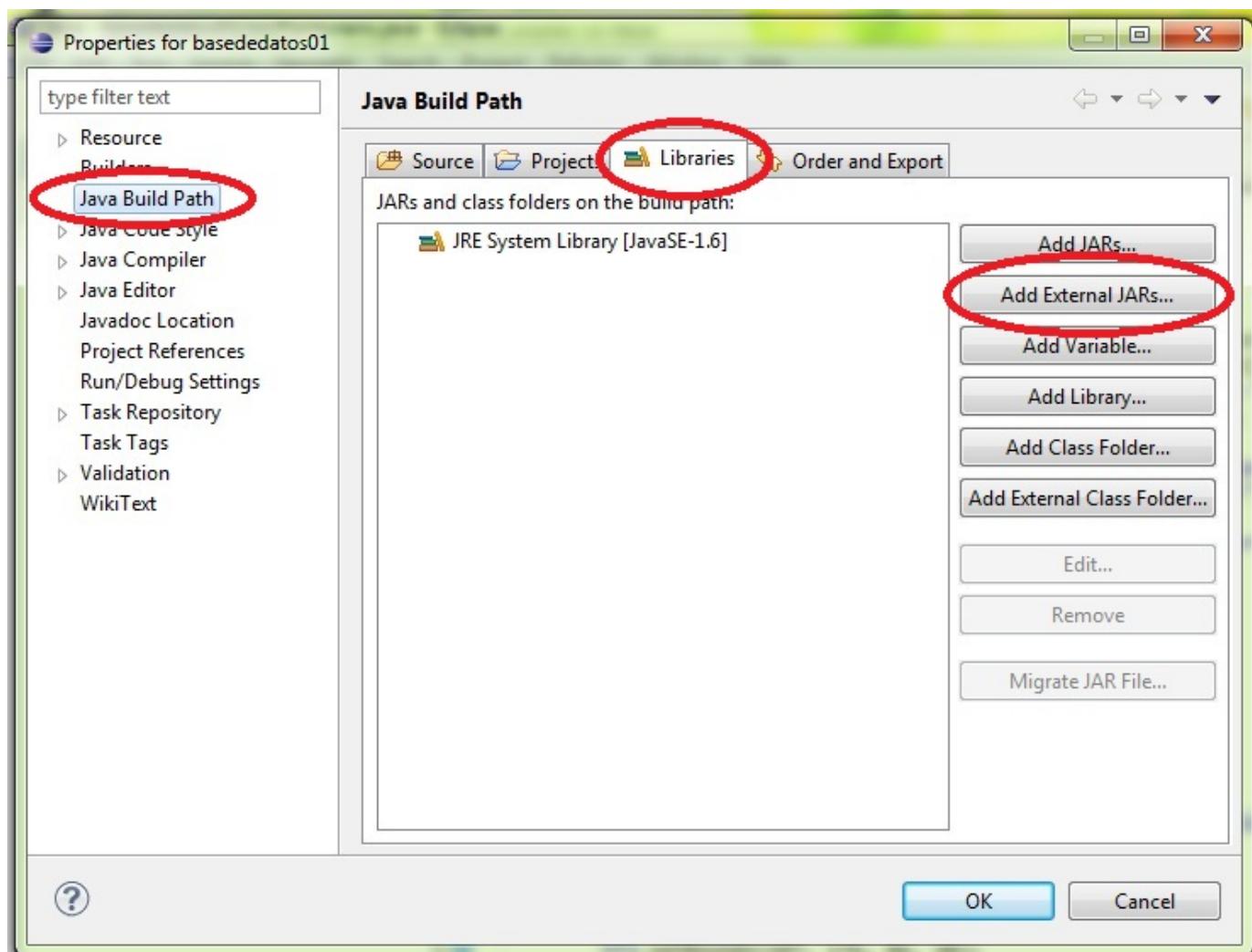
## Problema 1

Ya creamos anteriormente una base de datos llamada bd1 y en la misma creamos una tabla llamada articulos.

Procederemos a implementar en Java un programa que nos permita comunicarnos con la base de datos "bd1" y consultar, borrar y modificar filas en la tabla "articulos".

1 - Creamos desde Eclipse un proyecto llamado "basededatos02" y seguidamente con el WindowBuilder creamos una clase llamada "Formulario".

2 - Primero debemos añadir el driver que descargamos (mysql-connector-java-5.1.18-bin.jar) presionamos el botón derecho del mouse sobre nuestro proyecto y seleccionamos la opción "Properties", aparece el siguiente diálogo:



Seleccionamos la opción "Java Build Path", de la parte central seleccionamos la pestaña "Libraries" y procedemos a presionar el botón "Add External JARs...", donde procedemos a buscar el archivo mysql-connector-java-5.1.18-bin.jar

El código fuente completo que resuelve este problema es:

```
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class Formulario extends JFrame {

    private JPanel contentPane;
    private JTextField tf1;
    private JTextField tf2;
    private JLabel labelResultado;
    private JButton btnConsultaPorCdigo;
    private JTextField tf3;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Formulario frame = new Formulario();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public Formulario() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 606, 405);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblDescripcinDelArtculo = new JLabel("Descripción del artículo:");
        lblDescripcinDelArtculo.setBounds(23, 38, 193, 14);
        contentPane.add(lblDescripcinDelArtculo);
    }
}
```

```

tf1 = new JTextField();
tf1.setBounds(247, 35, 193, 20);
contentPane.add(tf1);
tf1.setColumns(10);

JLabel lblPrecio = new JLabel("Precio:");
lblPrecio.setBounds(23, 74, 95, 14);
contentPane.add(lblPrecio);

tf2 = new JTextField();
tf2.setBounds(247, 71, 107, 20);
contentPane.add(tf2);
tf2.setColumns(10);

labelResultado = new JLabel("resultado");
labelResultado.setBounds(361, 122, 229, 14);
contentPane.add(labelResultado);

btnConsultaPorCdigo = new JButton("Consulta por código");
btnConsultaPorCdigo.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        labelResultado.setText("");
        tf1.setText("");
        tf2.setText("");
        try {
            Connection
conexion=DriverManager.getConnection("jdbc:mysql://localhost/bd1","root" , "");
            Statement comando=conexion.createStatement();
            ResultSet registro = comando.executeQuery("select descripcion,precio
from articulos where codigo='"+tf3.getText());
            if (registro.next()==true) {
                tf1.setText(registro.getString("descripcion"));
                tf2.setText(registro.getString("precio"));
            } else {
                labelResultado.setText("No existe un artículo con dicho código");
            }
            conexion.close();
        } catch(SQLException ex){
            setTitle(ex.toString());
        }
    }
});
btnConsultaPorCdigo.setBounds(25, 122, 177, 23);
contentPane.add(btnConsultaPorCdigo);

tf3 = new JTextField();
tf3.setBounds(247, 123, 86, 20);
contentPane.add(tf3);
tf3.setColumns(10);

JButton btnNewButton = new JButton("Borrar");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        labelResultado.setText("");
        try {
            Connection
conexion=DriverManager.getConnection("jdbc:mysql://localhost/bd1","root" , "");
            Statement comando=conexion.createStatement();
            int cantidad = comando.executeUpdate("delete from articulos where
codigo='"+tf3.getText());
            if (cantidad==1) {
                tf1.setText("");
                tf2.setText("");
                labelResultado.setText("Se borro el artículo con dicho código");
            } else {
        }
    }
}
);

```

```

        labelResultado.setText("No existe un artículo con dicho código");
    }
    conexion.close();
} catch(SQLException ex){
    setTitle(ex.toString());
}
}
});
btnNewButton.setBounds(24, 156, 177, 23);
contentPane.add(btnNewButton);

JButton btnNewButton_1 = new JButton("Modificar");
btnNewButton_1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        labelResultado.setText("");
        try {
            Connection
conexion=DriverManager.getConnection("jdbc:mysql://localhost/bd1","root" , "");
            Statement comando=conexion.createStatement();
            int cantidad = comando.executeUpdate("update articulos set
descripcion='" + tf1.getText() + "','" +
                                         "precio=" + tf2.getText() + " where
codigo="+tf3.getText());
            if (cantidad==1) {
                labelResultado.setText("Se modifico la descripción y el precio del
artículo con dicho código");
            } else {
                labelResultado.setText("No existe un artículo con dicho código");
            }
            conexion.close();
        } catch(SQLException ex){
            setTitle(ex.toString());
        }
    }
});
btnNewButton_1.setBounds(21, 190, 179, 23);
contentPane.add(btnNewButton_1);
cargarDriver();
}

private void cargarDriver() {
    try {
        Class.forName("com.mysql.jdbc.Driver");
    }catch(Exception ex) {
        setTitle(ex.toString());
    }
}
}
}

```

El código a implementar cuando se presiona el botón "Consulta por código" es el visto en el concepto anterior:

```

btnConsultaPorCódigo = new JButton("Consulta por código");
btnConsultaPorCódigo.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        labelResultado.setText("");
        tf1.setText("");
        tf2.setText("");
        try {
            Connection
conexion=DriverManager.getConnection("jdbc:mysql://localhost/bd1","root" , "");
            Statement
comando=conexion.createStatement();

```

```

comando=conexion.createStatement();
    ResultSet registro = comando.executeQuery("select descripcion,precio from articulos where
codigo='"+tf3.getText());
        if (registro.next()==true) {
tf1.setText(registro.getString("descripcion"));
tf2.setText(registro.getString("precio"));
        } else {
labelResultado.setText("No existe
un artículo con dicho código");
        }
        conexion.close();
    } catch(SQLException ex){
        setTitle(ex.toString());
    }
}
});
```

Veamos el código para efectuar una baja en la tabla articulos:

```

btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        labelResultado.setText("");
        try {
            Connection
conexion=DriverManager.getConnection("jdbc:mysql://localhost/bd1","root" , "");
            Statement comando=conexion.createStatement();
            int cantidad = comando.executeUpdate("delete from articulos where
codigo='"+tf3.getText());
            if (cantidad==1) {
                tf1.setText("");
                tf2.setText("");
                labelResultado.setText("Se borro el artículo con dicho código");
            } else {
                labelResultado.setText("No existe un artículo con dicho código");
            }
            conexion.close();
        } catch(SQLException ex){
            setTitle(ex.toString());
        }
    }
});
```

Luego de crear un objeto de la clase Statement procedemos a llamar al método executeUpdate con un comando SQL válido (delete from articulos where codigo= código de artículo) El código de artículo lo extraemos del tercer JTextField.

El método executeUpdate retorna un entero que representa la cantidad de registros borrados de la tabla articulos. Luego en caso que retorne un uno procedemos a mostrar en un JLabel el mensaje "Se borro el artículo con dicho código", en caso contrario mostramos el mensaje "No existe un artículo con dicho código".

Para la modificación procedemos de forma muy similar al borrado:

```

btnNewButton_1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
```

```

labelResultado.setText("");
try {
    Connection
conexion=DriverManager.getConnection("jdbc:mysql://localhost/bd1","root" , "");
    Statement comando=conexion.createStatement();
    int cantidad = comando.executeUpdate("update articulos set
descripcion='" + tf1.getText() + "','" +
"precio=" + tf2.getText() + " where
codigo="+tf3.getText());
    if (cantidad==1) {
        labelResultado.setText("Se modifico la descripcion y el precio del
artículo con dicho código");
    } else {
        labelResultado.setText("No existe un artículo con dicho código");
    }
    conexion.close();
} catch(SQLException ex){
    setTitle(ex.toString());
}
});
});

```

Al método executeUpdate le pasamos un comando SQL de tipo update. Debemos concatenar los datos fijos del comando update con los valores que extraemos de los JTextField:

```

int cantidad = comando.executeUpdate("update articulos set
descripcion='" + tf1.getText() + "','" +
"precio=" + tf2.getText() + " where
codigo="+tf3.getText());

```

Es importante notar las comillas simples luego del carácter =, esto debido a que se trata de un campo de tipo varchar.

Nuevamente el método executeUpdate retorna la cantidad de registros modificados. En caso que retorne un 1 significa que se modificaron los datos correctamente.

Este proyecto lo puede descargar en un zip desde este enlace: [basededatos02.zip](#)

[\*\*Retornar\*\*](#)

## 62 - Instalación del "Eclipse IDE for Java EE Developers" y el servidor "Apache Tomcat"

[Listado completo de tutoriales](#)

### "Eclipse IDE for Java EE Developers"

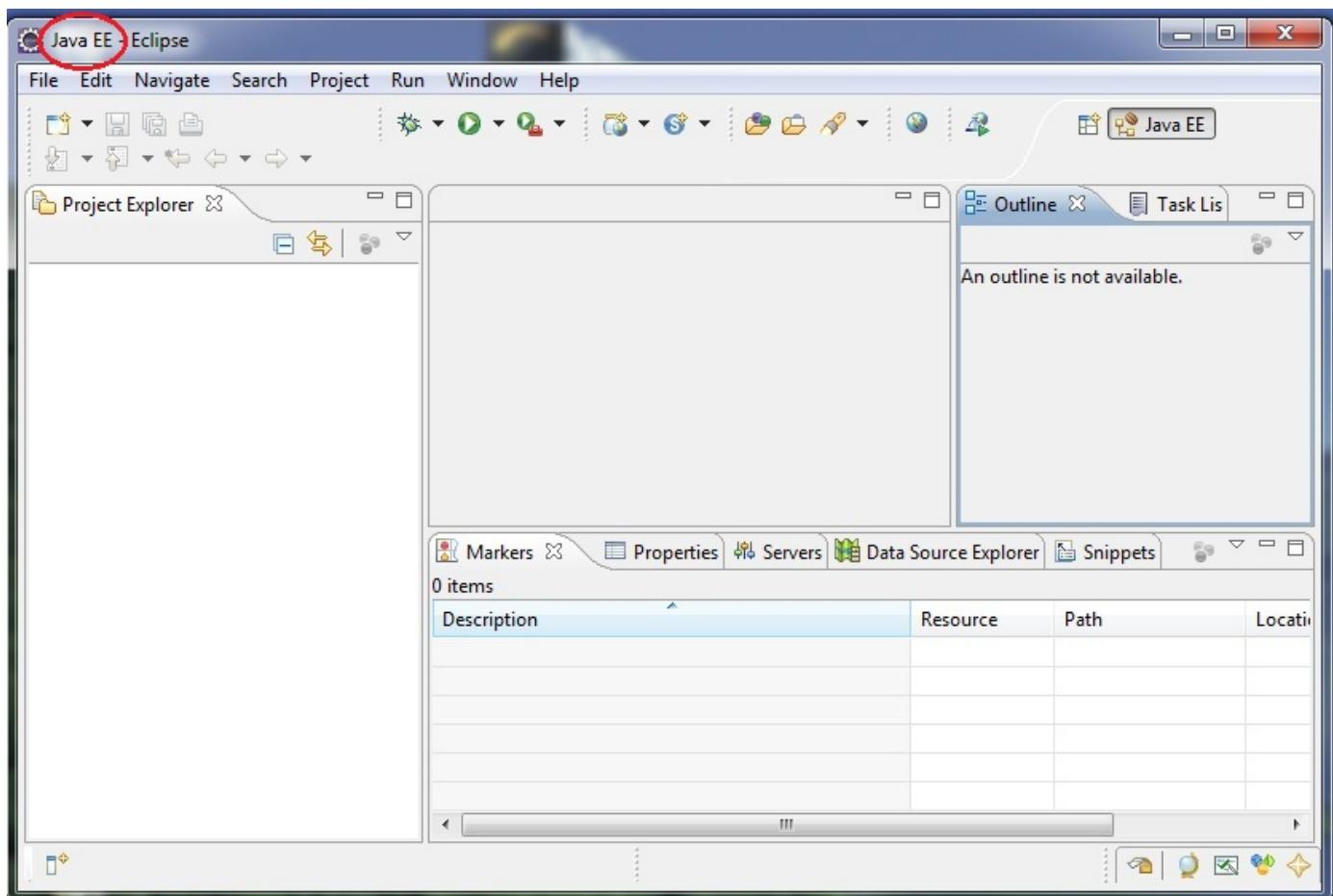
Para desarrollar aplicaciones que se ejecuten en un servidor web debemos utilizar la versión de Eclipse que viene con todos los complementos que facilitan el desarrollo.

La versión que debemos descargar es [Eclipse IDE for Java EE Developers](#), como podemos ver el tamaño es mayor que la versión que hemos utilizado hasta este momento (Eclipse IDE for Java Developers)

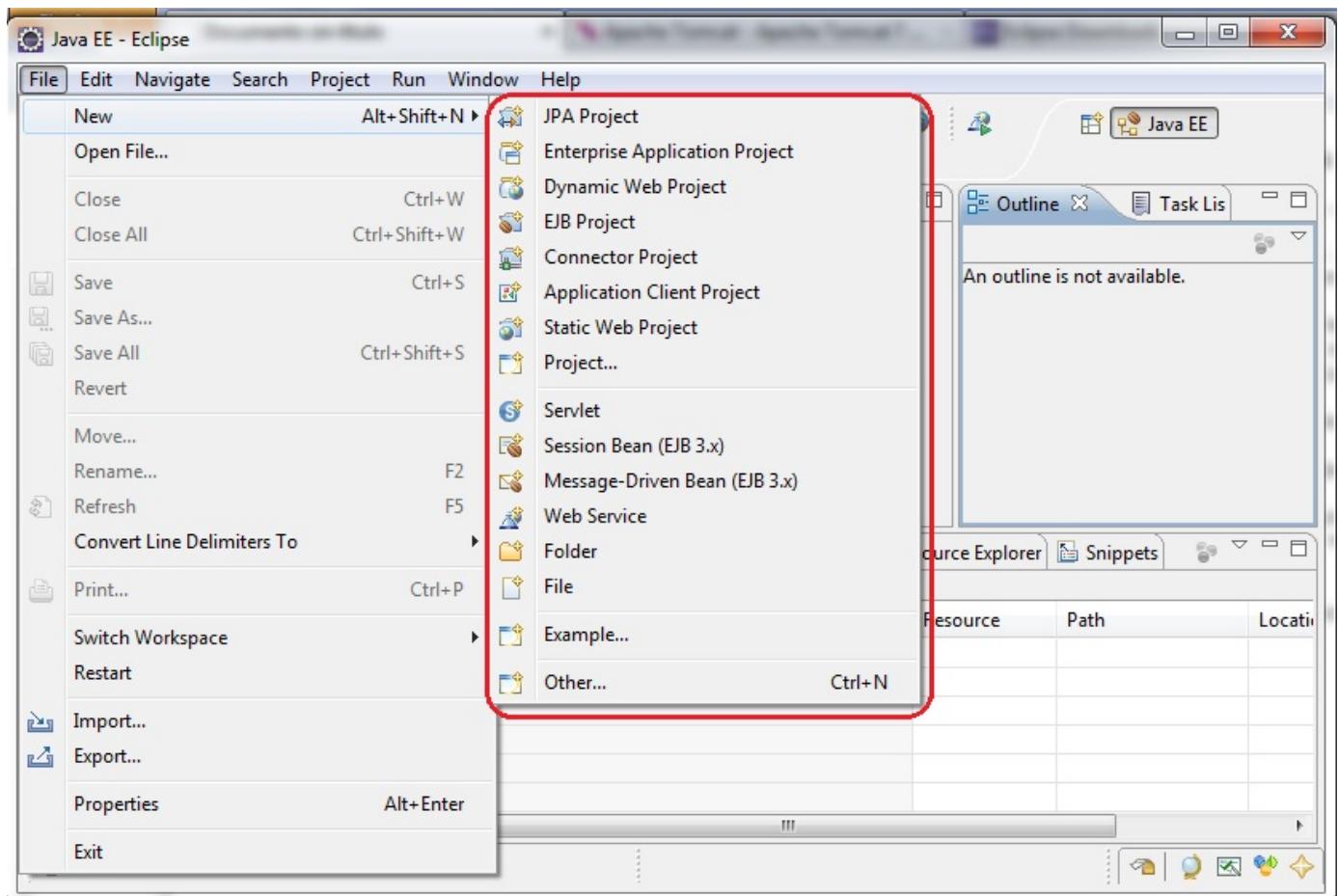
Podemos crear otra carpeta con otro nombre para no perder la versión de Eclipse que hemos utilizado para el desarrollo de aplicaciones de escritorio (swing)

Creemos la carpeta `eclipsej2ee` y dentro de la misma descomprimamos el entorno de Eclipse que acabamos de descargar "Eclipse IDE for Java EE Developers".

Cuando ejecutamos el Eclipse nos pide seleccionar la carpeta donde se almacenarán los proyectos que crearemos y aparece el siguiente entorno (como podemos ver prácticamente igual que la versión "Java Developers" con un título distinto):



Pero si ingresamos al menú de opciones `File -> New` veremos que nos permite crear una serie de proyectos muy distintos a la otra versión de Eclipse:

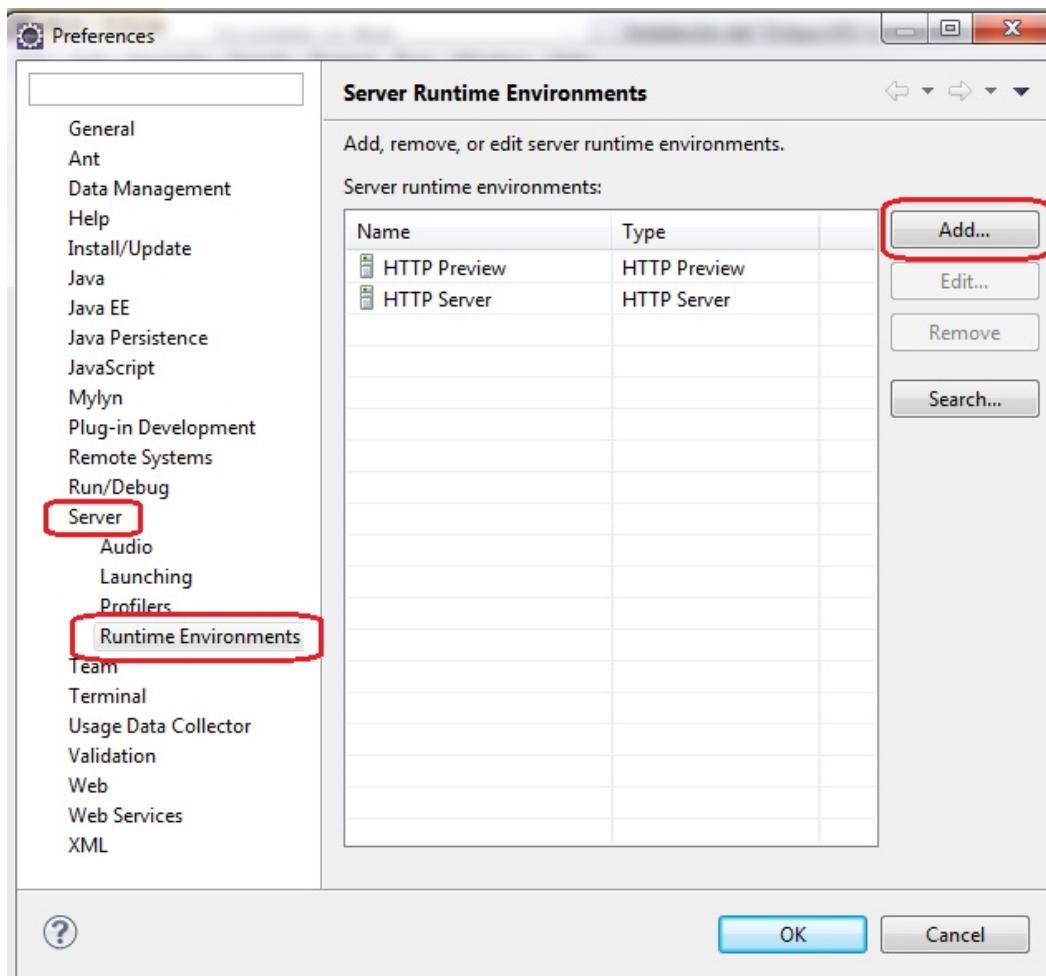


## "Apache Tomcat"

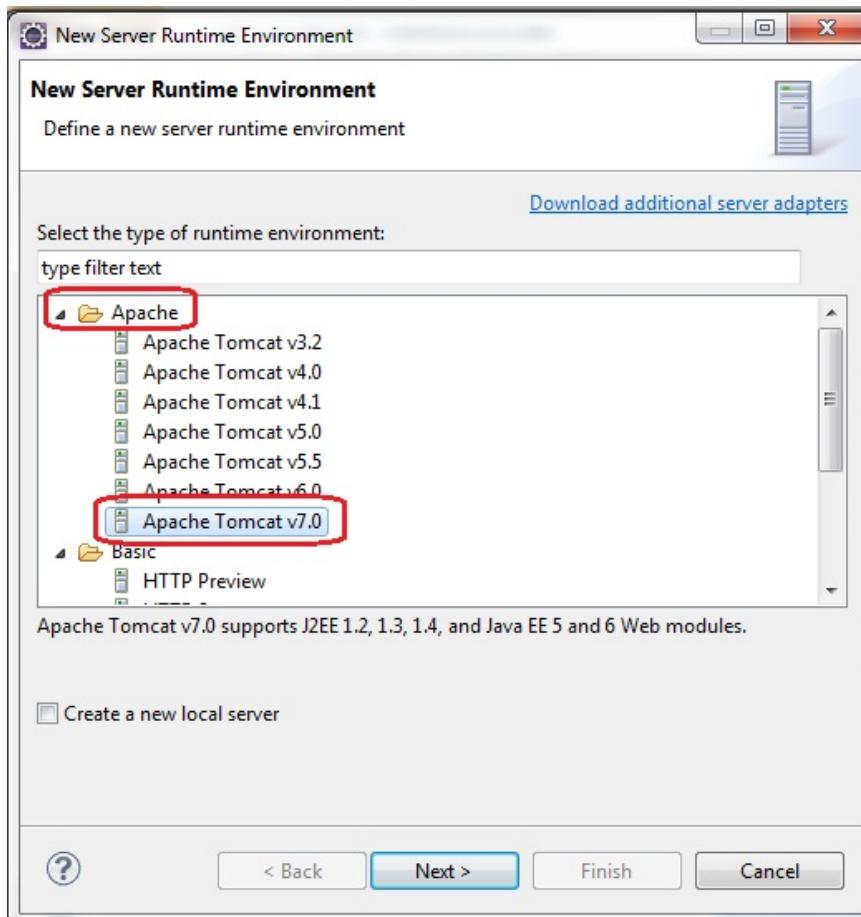
Ahora pasaremos a instalar un servidor web "Apache Tomcat" que nos permitirá ejecutar servlet y páginas dinámicas.

Podemos descargar el "Apache Tomcat" de [aquí](#) (descargar el archivo Binary Distributions Core 32-bit Windows zip) y descomprimirlo en una carpeta.

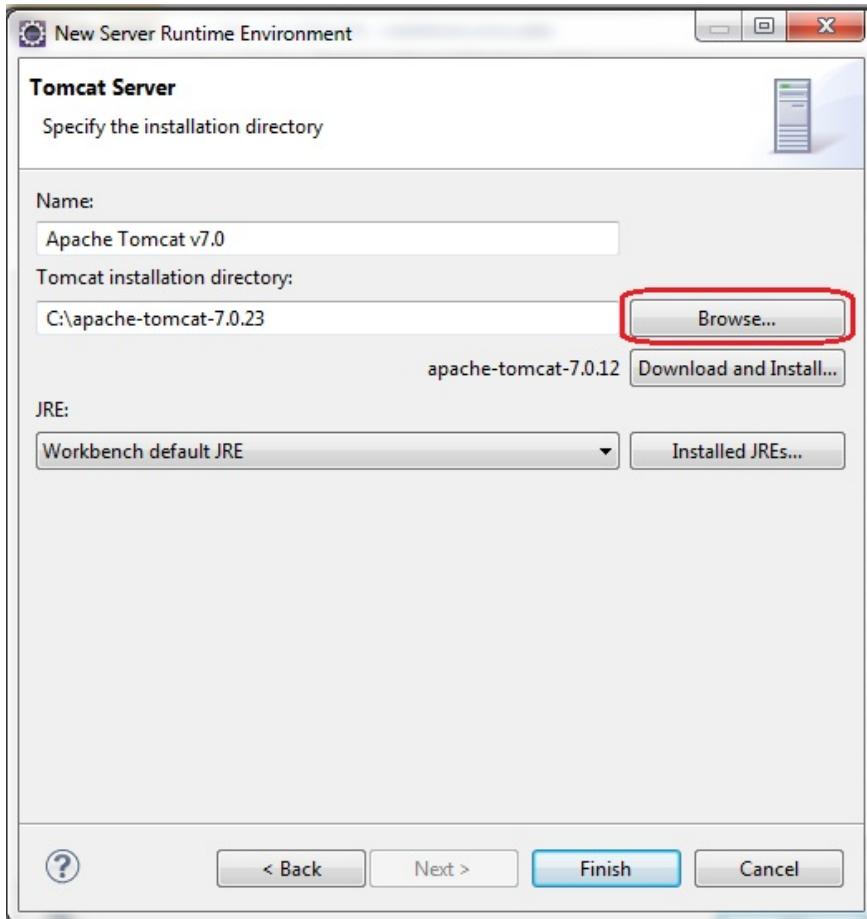
Una vez descomprimido procedemos a registrarlo en Eclipse. Desde el menú de opciones seleccionamos Window -> Preferences y en el diálogo que aparece debemos seleccionar Server -> Runtimes Environments y presionar el botón "Add...":



En el nuevo diálogo que aparece seleccionamos de la carpeta "Apache" la versión 7 que es la que acabamos de descargar y descomprimir en una carpeta de nuestro disco duro:

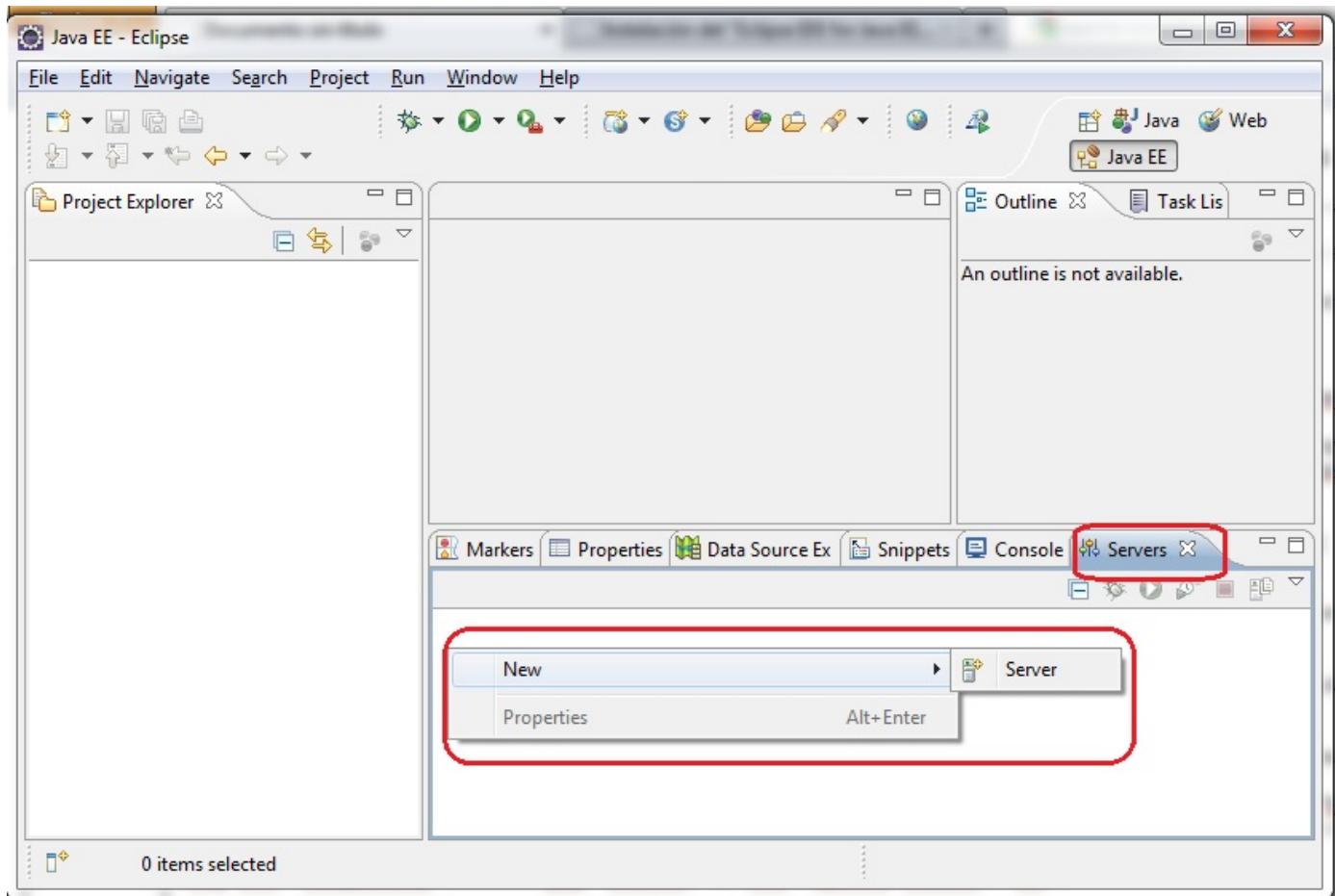


En el último diálogo que aparece debemos seleccionar la carpeta donde hemos descomprimido el "Apache Tomcat" y presionar el botón "Finish":

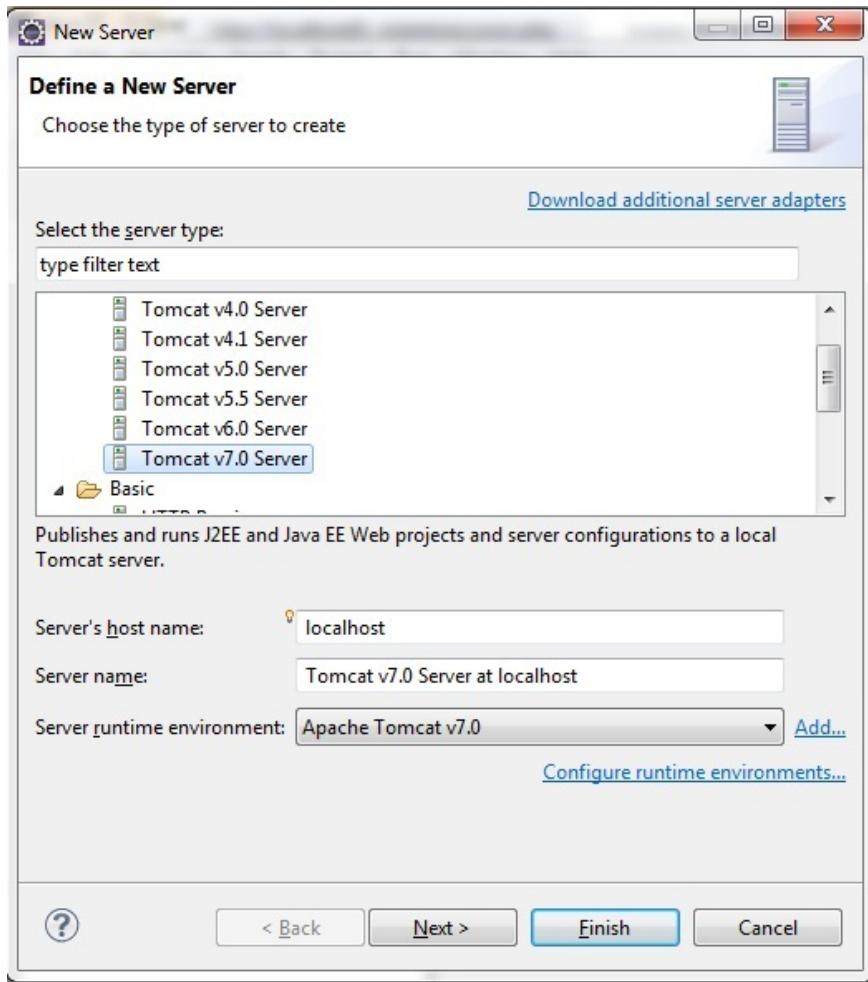


Ahora debemos iniciar los servicios del servido "Apache Tomcat" para podes hacer aplicaciones que hagan peticiones.

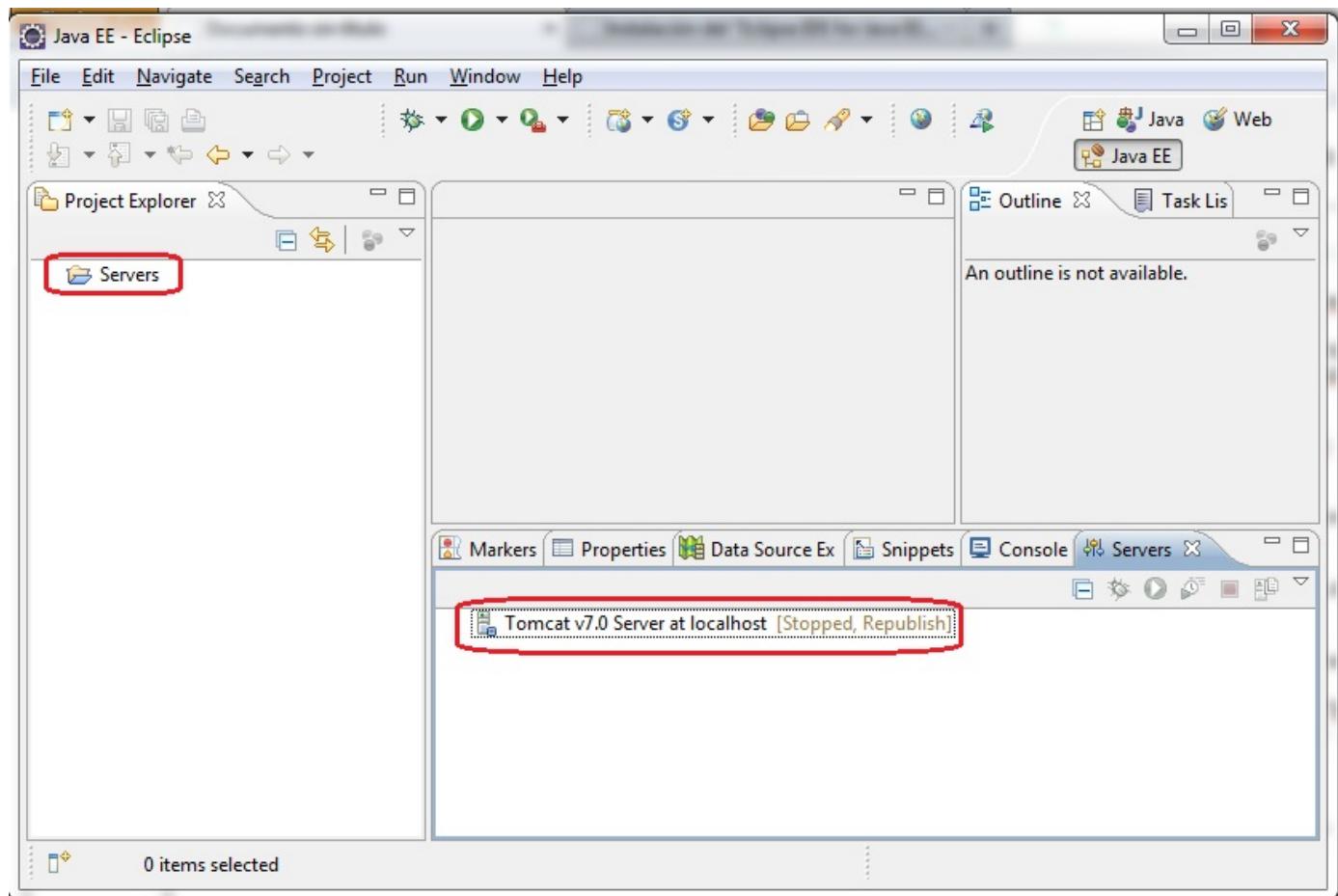
Para arrancar el Tomcat debemos presionar el botón derecho del mouse sobre la ventana "Server", si no parece esta ventana podemos activarla desde el menú (Window -> Show View -> Servers) y seguidamente seleccionar del menú contextual la opción New -> Server:



En este diálogo seleccionamos "Apache" Tomcat V7.0 y presionamos el botón "Finish":



Como podemos ver ya tenemos el "Tomcat" listo para poderlo utilizar en los distintos proyectos que implementaremos:

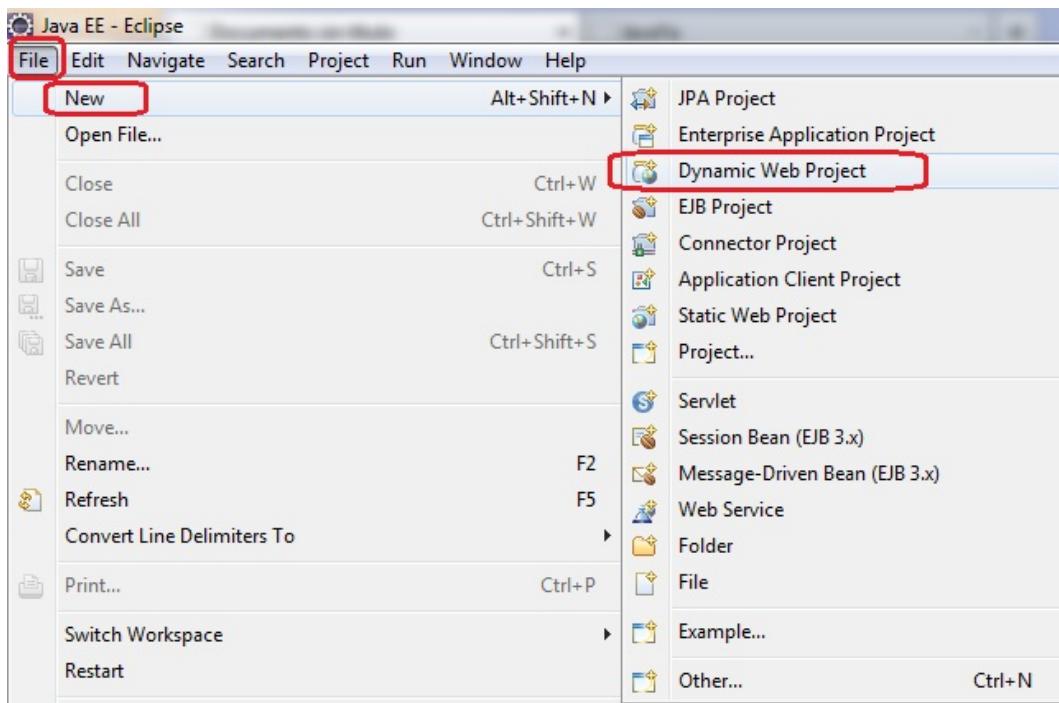


[Retornar](#)

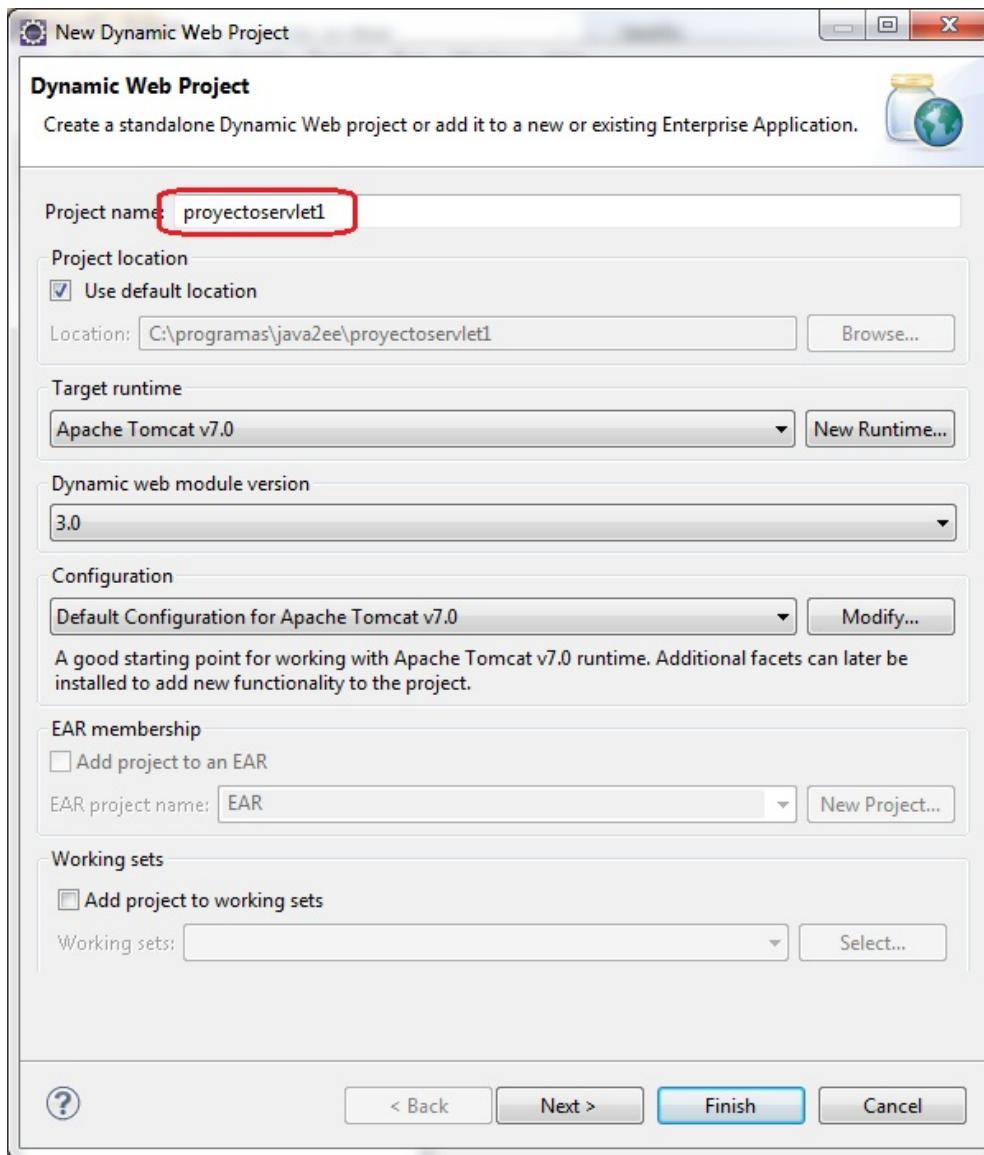
Un servlet es una clase que se ejecuta en el contexto de un servidor web (en nuestro caso el Apache Tomcat). Un servlet se ejecuta en un servidor web y el resultado de ejecución viaja por internet para ser visualizado en un navegador web (normalmente un servlet genera HTML, pero puede generar otros formatos de archivos).

Veremos los pasos en Eclipse para crear un servlet mínimo que nos muestre un mensaje y los números del 1 al 10000.

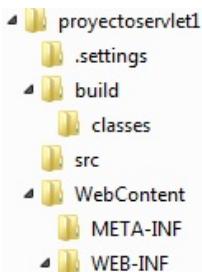
Desde el menú de opciones seleccionamos File -> New -> Dynamic Web Project:



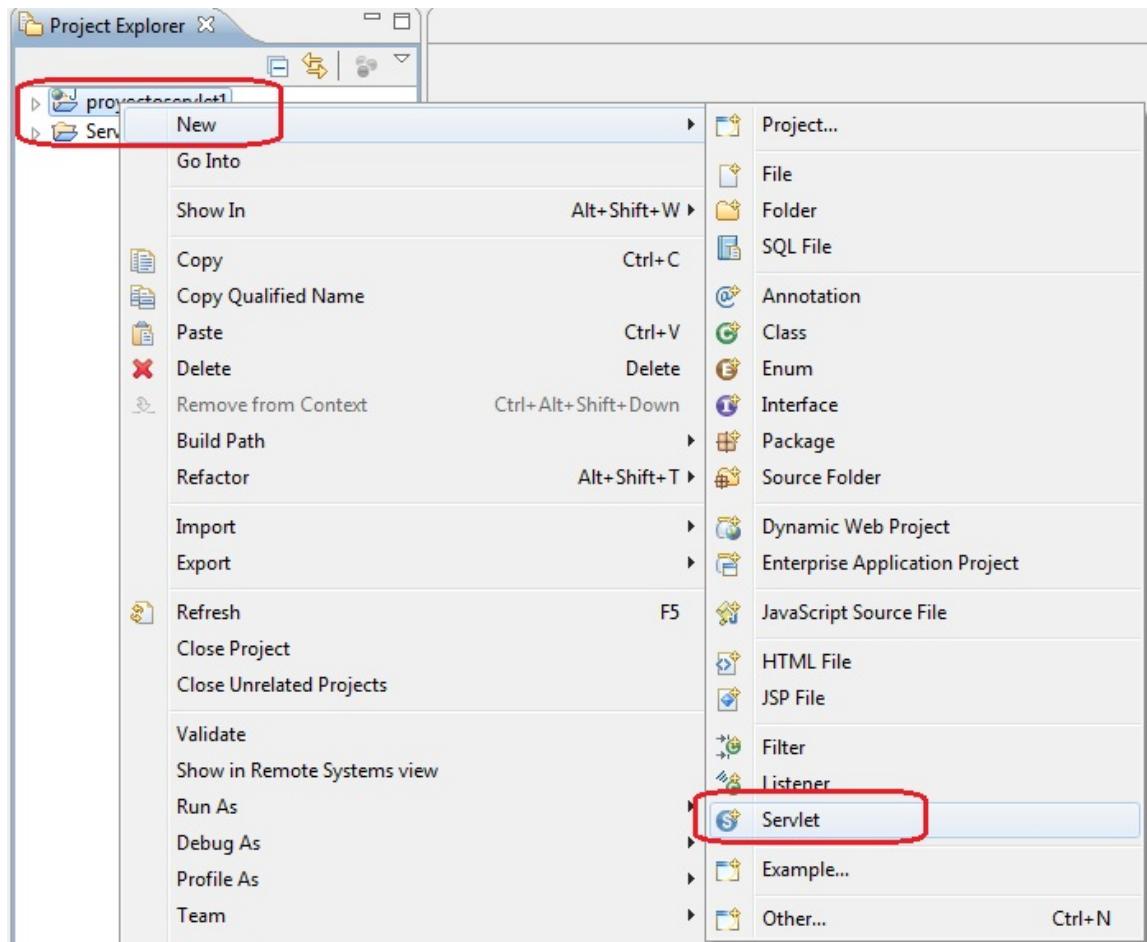
En el diálogo siguiente especificamos el nombre del proyecto (en nuestro caso le llamaremos `proyectoservlet1`) y presionamos el botón "Finish":



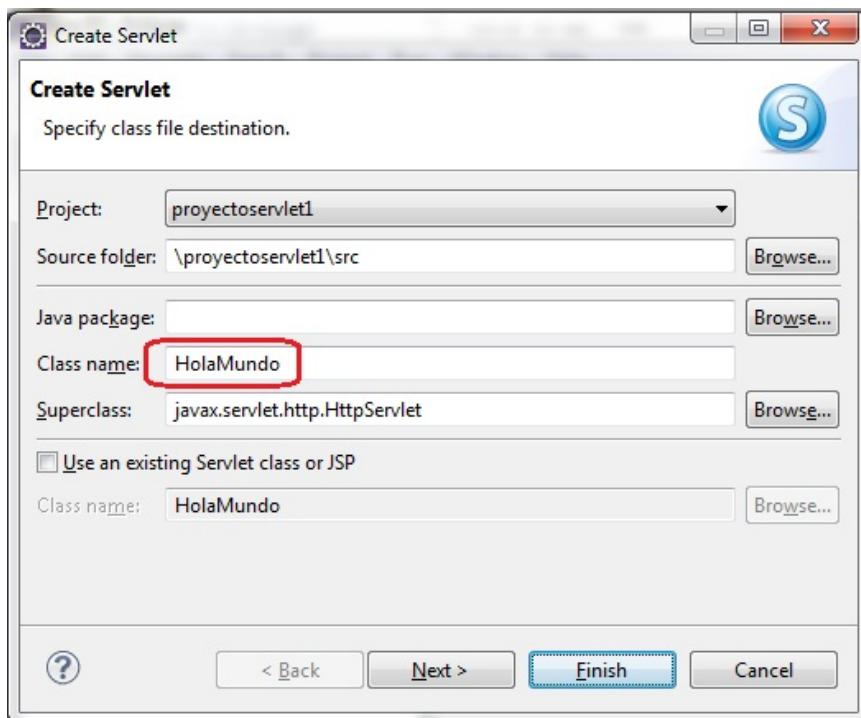
El Eclipse nos crea una serie de carpetas y archivos donde alojaremos los servlet:



Ahora presionamos el botón derecho sobre el nombre del proyecto y seleccionamos la opción New -> Servlet:



En el diálogo siguiente especificamos el nombre de nuestro servlet (en nuestro ejemplo le llamaremos HolaMundo), presionamos el botón "Finish" y ya tenemos el esqueleto básico de un servlet:



El código fuente generado es el siguiente:

```

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class HolaMundo
 */
@WebServlet("/HolaMundo")
public class HolaMundo extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public HolaMundo() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // TODO Auto-generated method stub
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // TODO Auto-generated method stub
    }
}

```

Todo servlet debe heredar de la clase HttpServlet que se encuentra en el paquete javax.servlet.http

Esta clase debe sobreescribir el método doGet o doPost (o ambos) En el protocolo HTTP las peticiones pueden ser de tipo post (cuando llamamos a una página desde un formulario HTML) y de tipo get (páginas sin formulario)

Nuestro problema es mostrar un mensaje e imprimir los números del 1 al 10000, esta actividad la haremos en el método doGet.

El algoritmo a implementar en el método doGet para dicha salida es:

```

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class HolaMundo
 */
@WebServlet("/HolaMundo")
public class HolaMundo extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**

```

```

* @see HttpServlet#HttpServlet()
*/
public HolaMundo() {
    super();
    // TODO Auto-generated constructor stub
}

/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    // TODO Auto-generated method stub
    PrintWriter out = response.getWriter();

    out.println("<html>");
    out.println("<head></head>");
    out.println("<body>");
    out.println("<h1>Hola Mundo</h1>");
    for(int f=1;f<=10000;f++) {
        out.println(f);
        out.println(" - ");
    }
    out.println("</body>");
    out.println("</html>");
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    // TODO Auto-generated method stub
}
}

```

Una parte importante de la declaración del servlet que nos genera automáticamente el Eclipse es la anotación `@WebServlet` (esta línea registra el servlet para todas las peticiones al servidor con la sintaxis `http://localhost:8080/proyectoservlet1/HolaMundo`):

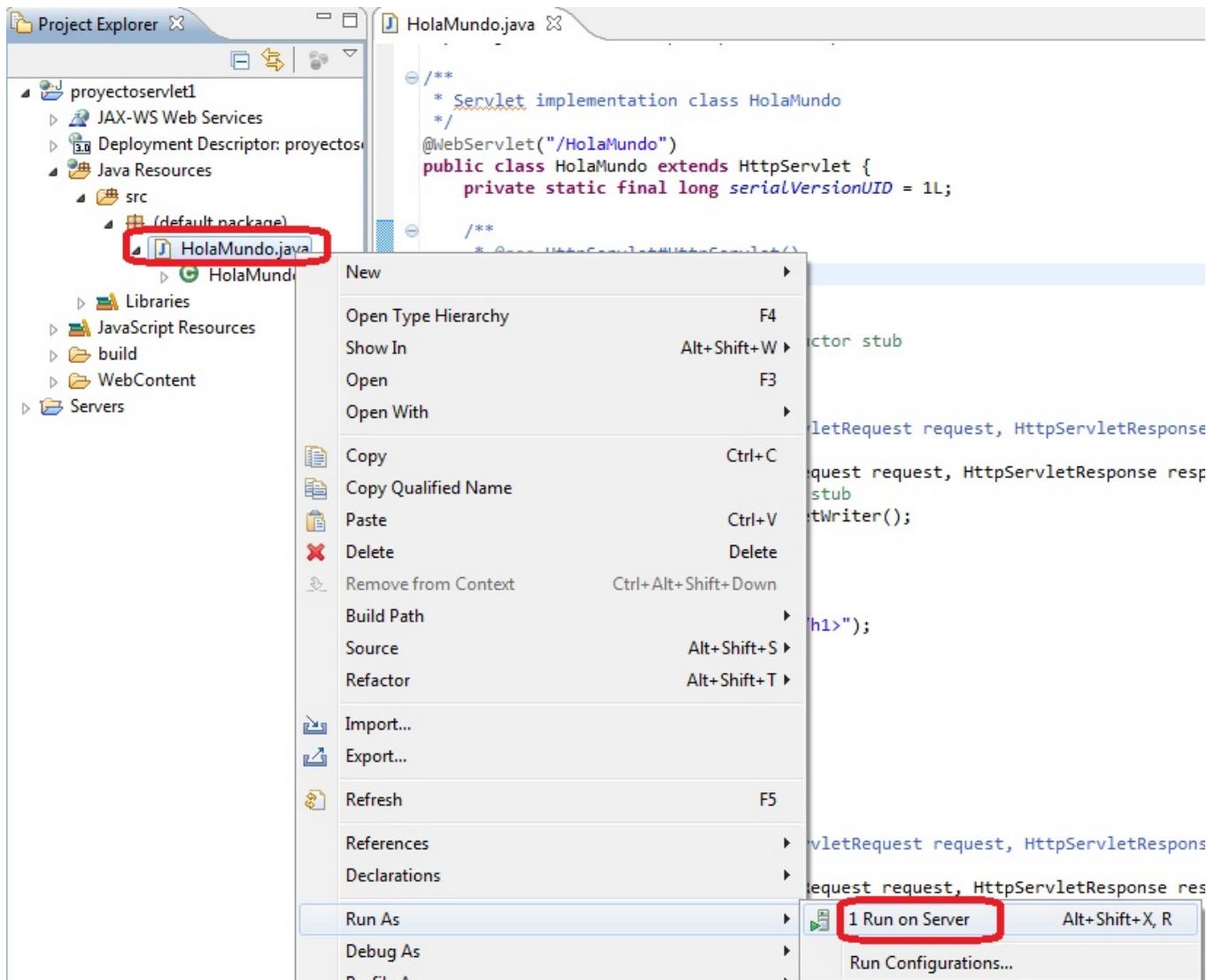
```
@WebServlet("/HolaMundo")
```

Obtenemos una referencia de un objeto de la clase `PrintWriter` (debemos importar la clase `PrintWriter`) mediante la llamada al método `getWriter` del objeto `response` que llega como parámetro al método `doGet`:

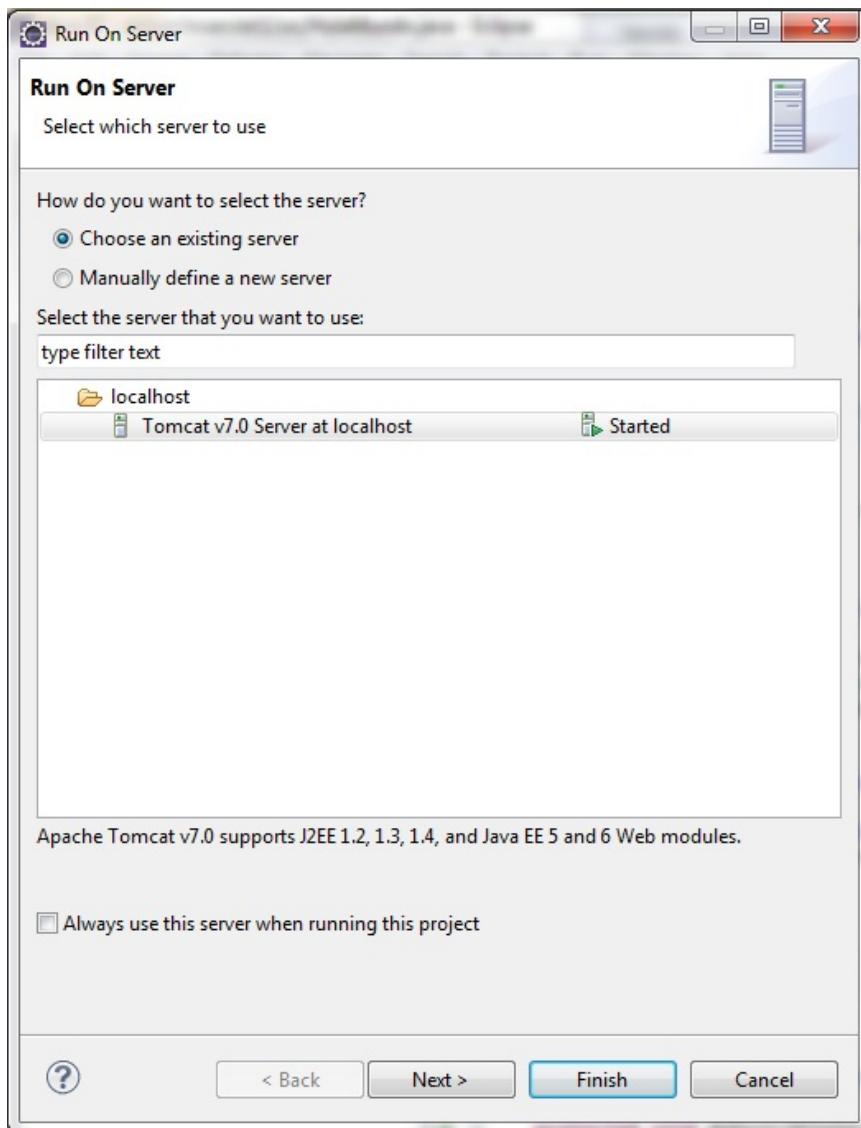
```
PrintWriter out = response.getWriter();
```

Todas las salidas son llamando al método `println` del objeto `out` de la clase `PrintWriter`. Como vemos generamos como salida HTML, para mostrar los números del 1 al 10000 es más conveniente utilizar una estructura repetitiva que hacer una salida secuencial.

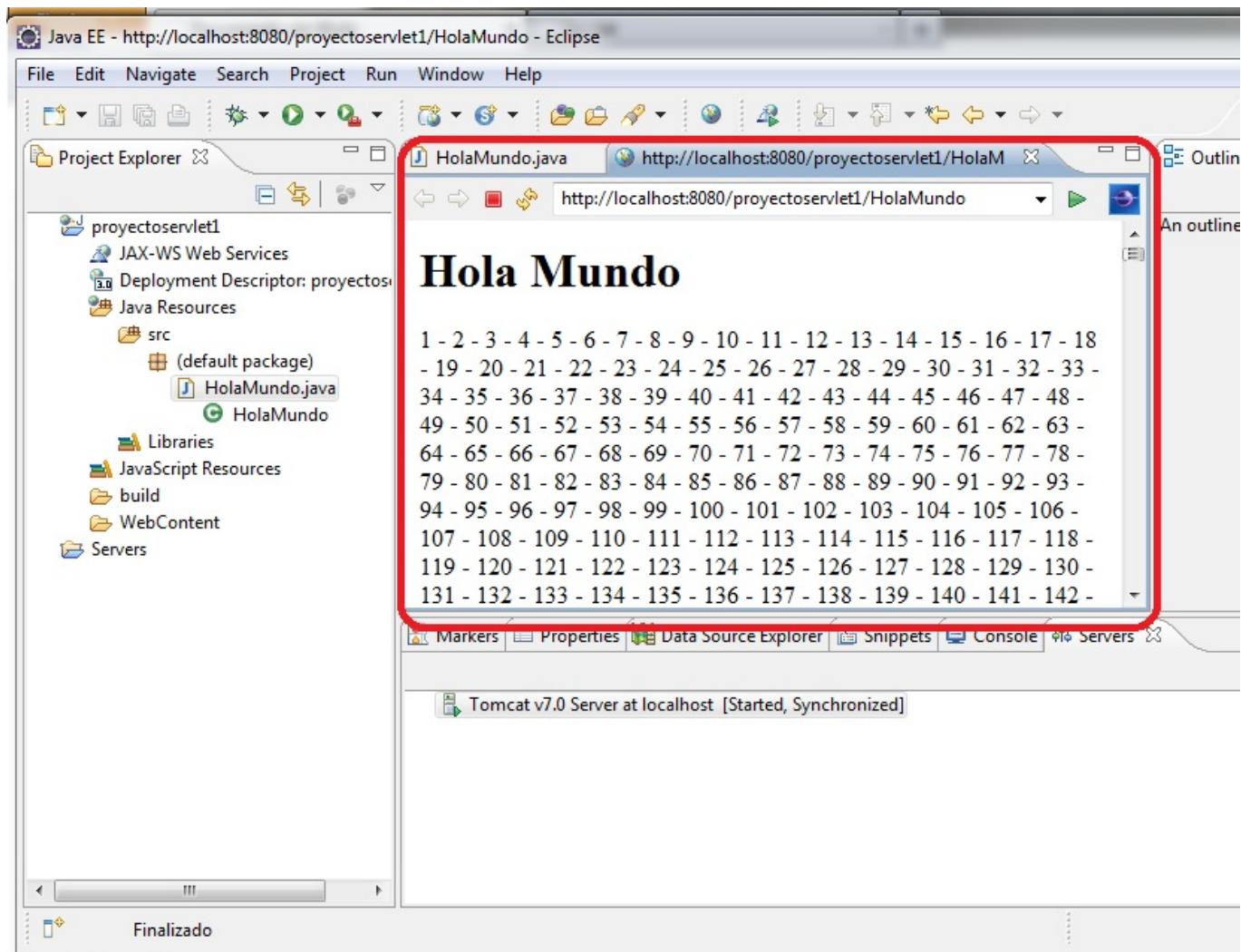
Para probar el servlet que acabamos de codificar debemos presionar el botón derecho del mouse sobre el nombre de la clase y seleccionar "Run on Server":



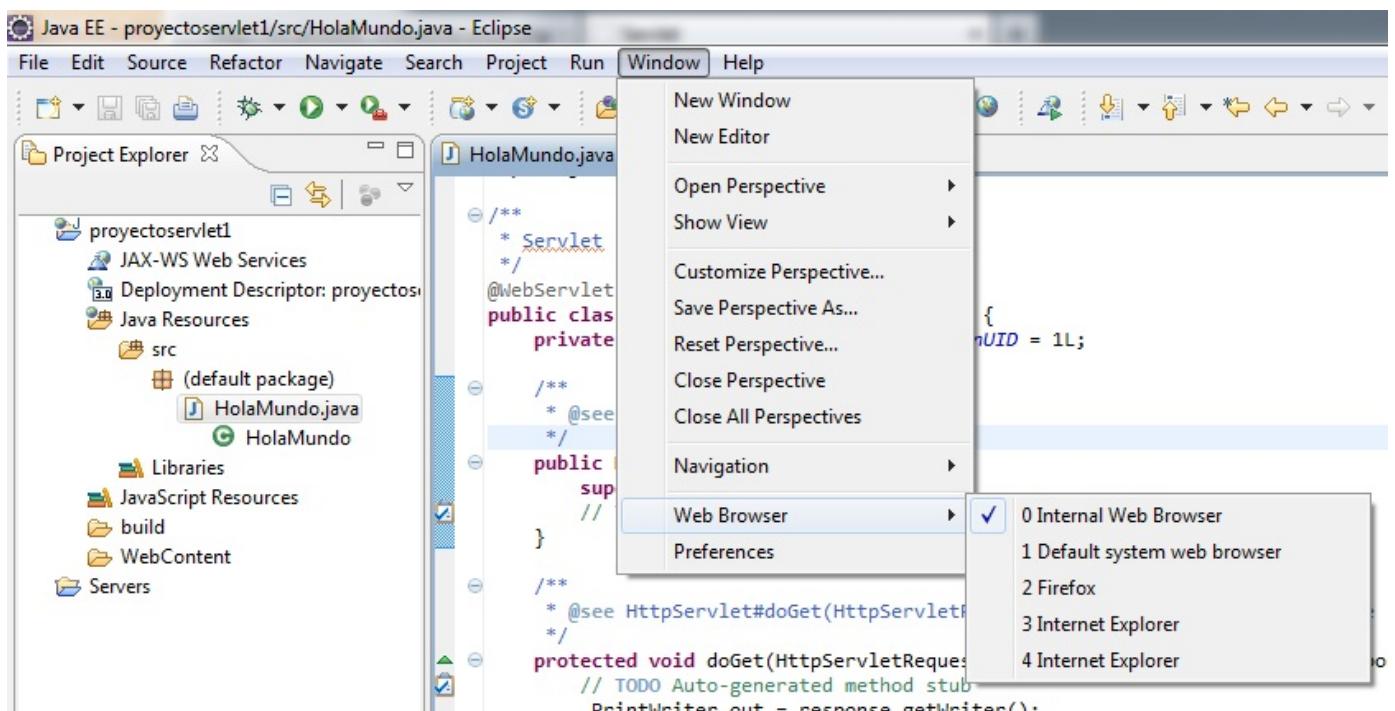
Aparece un diálogo que debemos seleccionar el botón "Finish" ya que está seleccionado el servidor "Tomcat" para ejecutar el servlet:



El resultado de la ejecución del servlet lo podemos ver dentro de una ventana dentro del mismo Eclipse:



Si queremos que el resultado aparezca en otro navegador podemos configurar desde el menú de Eclipse el navegador que muestra el resultado que devuelve Tomcat:



[Retornar](#)

## 64 - Recuperación de los datos de un formulario HTML en un servlet

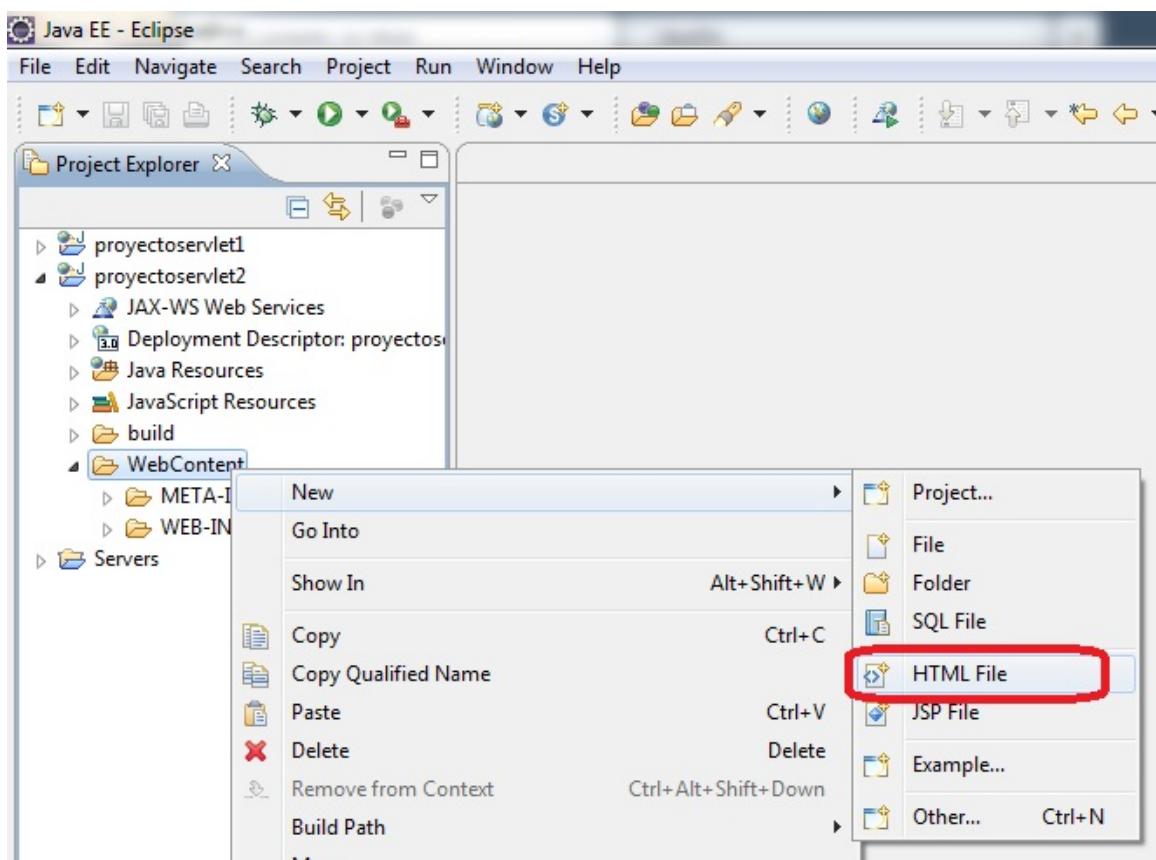
[Listado completo de tutoriales](#)

Veremos ahora que un servlet puede recibir datos de un formulario HTML.

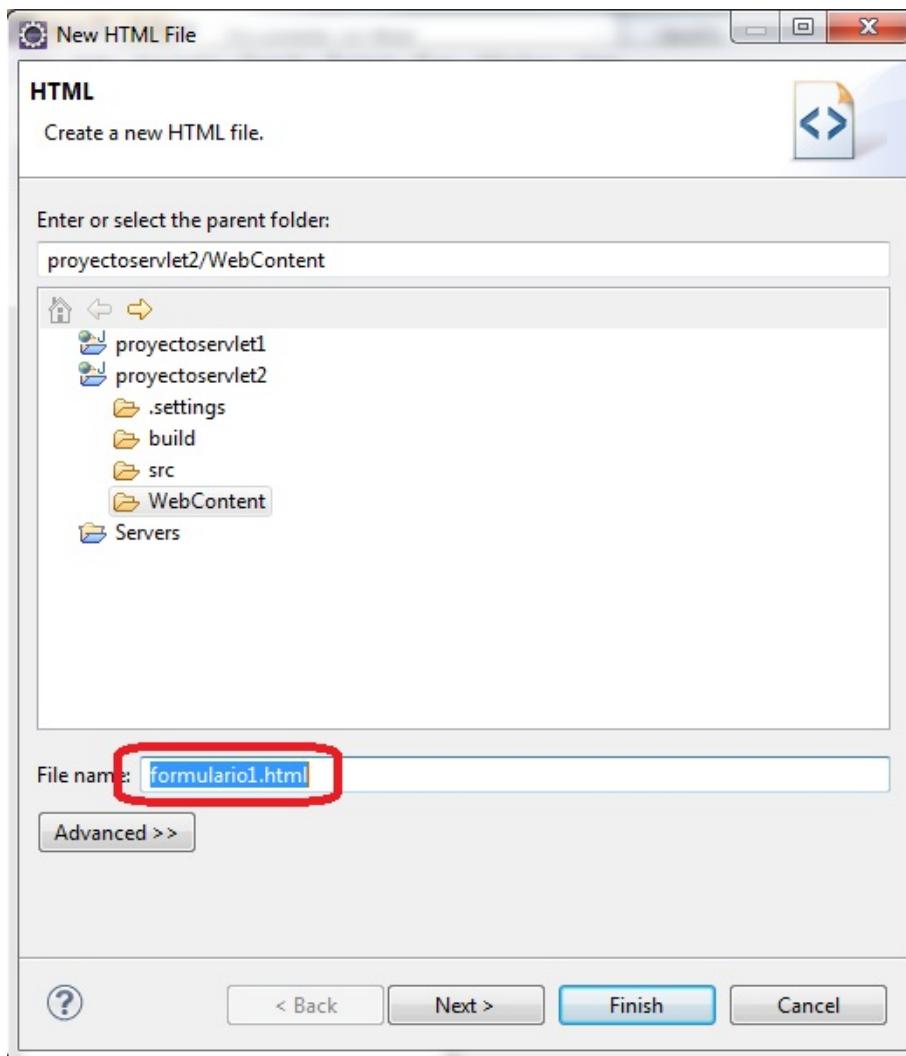
Implementaremos un formulario HTML que solicite el ingreso del nombre y clave de un usuario y posteriormente recuperaremos los dos datos en un servlet y los mostraremos en otra página generada por el servlet.

Primero crearemos un proyecto llamado proyectoservlet2.

Para crear el archivo html con el formulario presionamos el botón derecho del mouse sobre la carpeta WebContent:



En el diálogo siguiente especificamos el nombre del archivo html, en nuestro caso lo llamaremos formulario1.html:



Codificamos la página html con el formulario web que solicita el ingreso del nombre de usuario y su clave:

```
<!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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form method="post" action="RecuperacionUsuario">
Ingresar nombre de usuario:
<input type="text" name="usuario" size="20"><br>
Ingresar clave:
<input type="password" name="clave" size="20"><br>
<input type="submit" value="confirmar">
</form>
</body>
</html>
```

Lo más importante cuando creamos el formulario web es la especificación de la propiedad action de la marca form con el nombre el servlet que recuperará los datos del formulario:

```
<form method="post" action="RecuperacionUsuario">
```

Ahora creamos el servlet que como vemos deberá llamarse "RecuperacionUsuario". El código fuente del servlet es el siguiente:

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class RecuperacionUsuario
 */
@WebServlet("/RecuperacionUsuario")
public class RecuperacionUsuario extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public RecuperacionUsuario() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // TODO Auto-generated method stub
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // TODO Auto-generated method stub
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head></head>");
        out.println("<body>");

        out.println("Usuario:");
        String usu=request.getParameter("usuario");
        out.println(usu);
        out.println("<br>");
        out.println("Clave:");
        String cla=request.getParameter("clave");
        out.println(cla);

        out.println("</body>");
        out.println("</html>");
    }
}
```

Como podemos ver en el código fuente de la clase RecuperacionUsuario debemos implementar todo el código en el método doPost, ya que este se ejecuta cuando se tiene un formulario HTML y se especificó en el HTML en la propiedad method el valor post:

```
<form method="post" action="RecuperacionUsuario">
```

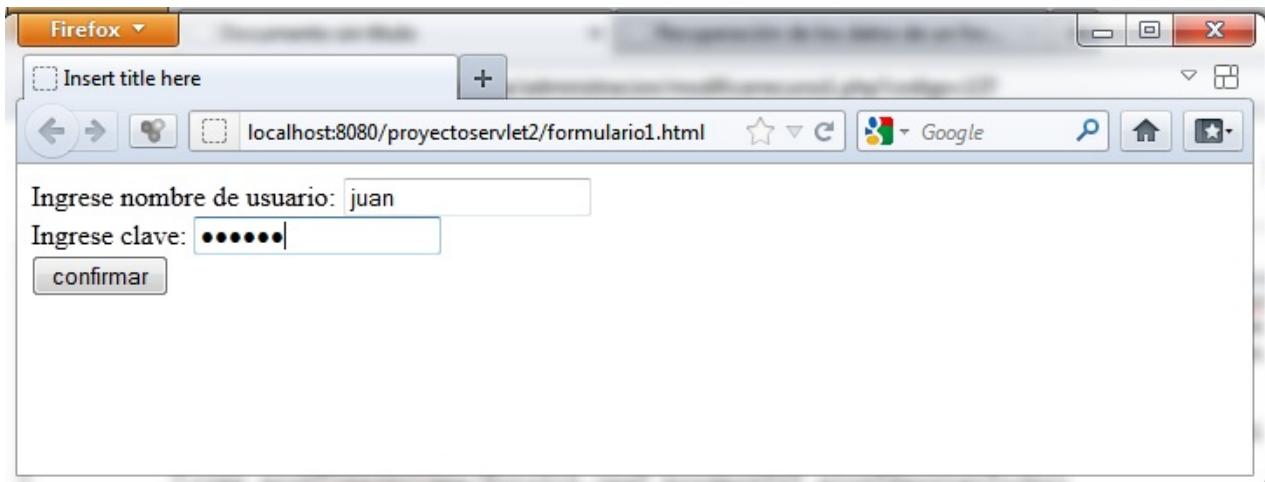
Para recuperar los datos de los controles text y password del formulario HTML el objeto request de la clase HttpServletRequest dispone de un método llamado getParameter que le indicamos el nombre del control a recuperar:

```
String usu=request.getParameter("usuario");
```

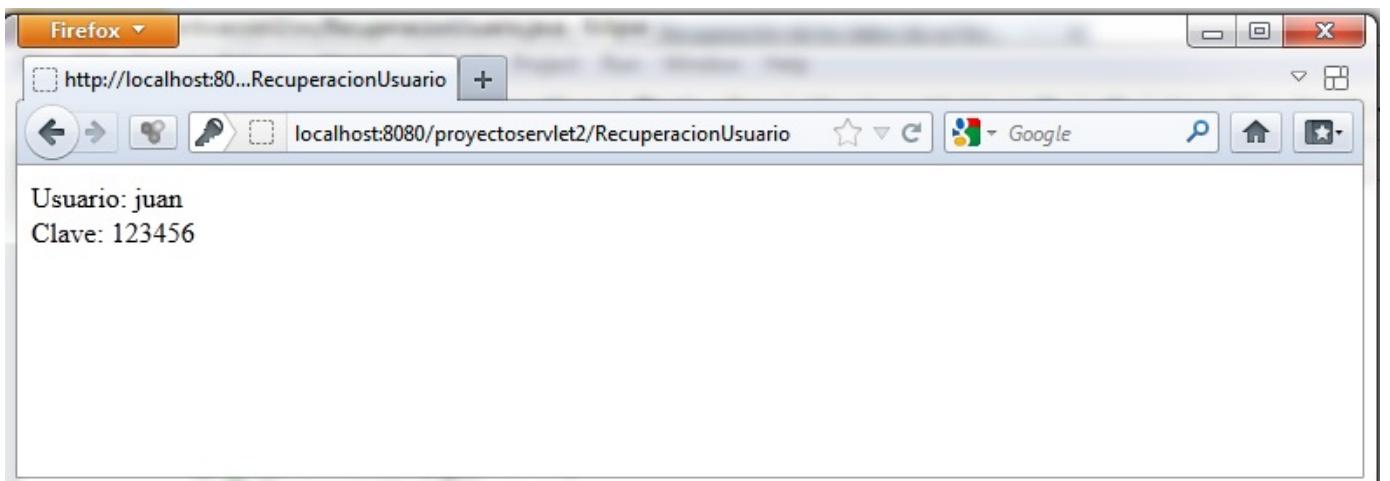
Luego de recuperarlo procedemos a mostrarlo dentro de la página generada:

```
out.println(usu);
```

Para probar nuestra aplicación debemos presionar el botón derecho sobre el formulario1.html y seleccionar "Run As" -> "Run on Server", luego el resultado de la ejecución en el navegador:



y



## Problema propuesto

1. Crear un proyecto llamado proyectoservlet3. Confeccionar un formulario HTML que solicite la carga de 2 valores por teclado. Cuando se presione el botón submit llamar a un servlet que recupere los dos valores ingresados y muestre su suma.

[Solución](#)

[\*\*Retornar\*\*](#)

# 65 - Llamada a servlet desde un hipervínculo (con y sin parámetros)

[Listado completo de tutoriales](#)

## Problema:

Confeccionaremos una página HTML con dos hipervínculos a dos servlet. El primero tiene por objetivo mostrar la tabla de multiplicar del 2, el segundo servlet llega como parámetro el número del cual queremos mostrar la tabla de multiplicar.

Primero creamos un proyecto llamado `proyectoservlet4`. Seguidamente creamos el archivo HTML (presionamos el botón derecho del mouse sobre la carpeta `WebContent` y creamos un archivo HTML llamado `menu.html`) y codificamos lo siguiente:

```
<!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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<a href="TablaDel2"> Mostrar tabla del 2</a><br>
<a href="Tabla?num=5">Mostrar tabla del 5</a>
</body>
</html>
```

El primer hipervínculo en la propiedad `href` indicamos el nombre del servlet a ejecutar:

```
<a href="TablaDel2"> Mostrar tabla del 2</a><br>
```

El segundo hipervínculo llama al servlet `Tabla` y pasa un parámetro `num` con el valor 5 (este valor se rescatará posteriormente desde el servlet):

```
<a href="Tabla?num=5">Mostrar tabla del 5</a>
```

Ahora crearemos un servlet llamado `TablaDel2`:

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
```

```

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class TablaDel2
 */
@WebServlet("/TablaDel2")
public class TablaDel2 extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public TablaDel2() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
     * response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head></head>");
        out.println("<body>");
        out.println("<h1>Tabla del 2</h1>");
        for(int f=2;f<=20;f=f+2) {
            out.println(f);
            out.println(" - ");
        }
        out.println("</body>");
        out.println("</html>");
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
     * response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        // TODO Auto-generated method stub
    }
}

```

Es importante hacer notar que la anotación del servlet coincide con la propiedad href del hipervínculo de la página HTML:

```

@WebServlet("/TablaDel2")
public class TablaDel2 extends HttpServlet {

```

En el método doGet procesamos la petición mediante la generación de la página dinámica:

```

protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    // TODO Auto-generated method stub
    PrintWriter out = response.getWriter();

    out.println("<html>");
    out.println("<head></head>");
    out.println("<body>");
    out.println("<h1>Tabla del 2</h1>");
    for(int f=2;f<=20;f=f+2) {
        out.println(f);
        out.println(" - ");
    }
    out.println("</body>");
    out.println("</html>");
}

```

Ahora crearemos el segundo servlet llamado Tabla:

```

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Tabla
 */
@WebServlet("/Tabla")
public class Tabla extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Tabla() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
     * response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head></head>");
        out.println("<body>");
        String cad=request.getParameter("num");
        int valor=Integer.parseInt(cad);
        out.println("<h1>Tabla del "+cad+"</h1>");
        for(int f=valor;f<=valor*10;f=f+valor) {

```

```
        out.println(f);
        out.println(" - ");
    }
    out.println("</body>");
    out.println("</html>");
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    // TODO Auto-generated method stub
}

}
```

Para recuperar el parámetro del hipervínculo llamamos al método `getParameter` del objeto `request`. Debemos pasar un `String` con el nombre del parámetro y el mismo nos retorna el valor del parámetro:

```
String cad=request.getParameter("num");
```

## Retornar

# 66 - Redireccionamiento a otro sitio o página desde un servlet

[Listado completo de tutoriales](#)

## Problema:

Confeccionar un formulario html que solicite el ingreso de un sitio web. Cuando se presione un botón redireccione a dicha web.

Crearemos un proyecto llamado proyectoservlet5 y dentro del mismo un archivo HTML:

```
<!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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form method="get" action="Redireccionamiento">
Ingrrese la dirección de la página que quiere visitar(Ej. www.google.com):
<input type="text" name="direccion" size="60">
<br>
<input type="submit" value="Ir">
</form>
</body>
</html>
```

Como vemos cuando se presiona el botón submit se envían los datos al servlet llamado Redireccionamiento:

```
<form method="get" action="Redireccionamiento">
```

Creamos ahora un servlet llamado Redireccionamiento:

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Redireccionamiento
 */
@WebServlet("/Redireccionamiento")
public class Redireccionamiento extends HttpServlet {
    private static final long serialVersionUID = 1L;
```

```

/**
 * @see HttpServlet#HttpServlet()
 */
public Redireccionamiento() {
    super();
    // TODO Auto-generated constructor stub
}

/**
 * @see HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // TODO Auto-generated method stub
    String dire=request.getParameter("direccion");
    response.sendRedirect("http://"+dire);
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request,
HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // TODO Auto-generated method stub
}

}

```

Para redireccionar el objeto request de la clase HttpServletRequest tiene un método llamado sendRedirect que le pasamos como parámetro un String con la dirección del sitio que debe devolver el servlet (en lugar del servlet propiamente dicho)

La dirección del sitio a redireccionar la extraemos con el método getParameter como hemos visto:

```

protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // TODO Auto-generated method stub
    String dire=request.getParameter("direccion");
    response.sendRedirect("http://"+dire);
}

```

[\*\*Retornar\*\*](#)