Introducción a la Programación en **JAVA** *Oracle ®*

Introducción.

Un programa hecho en Java, no es de por si ejecutable. Es decir, el programa compilado no hace nada. Para que funcione, necesitaremos lo que se conoce como **Java Virtual Machine** o ***Máquina Virtual de Java***.
Esto hace que los programas Java sean independientes de la plataforma y puedan funcionar en sistemas operativos distintos y en varios dispositivos. Es un lenguaje PO de Objetos.

Lo que necesitamos para empezar a programar Java.

Descarga la máquina virtual *JRE* y el *JDK* o mejor el ***Java SE*** con su documentación. Desde la página:
*http://www.**oracle**.com/technetwork/java/javase/downloads/index.html*
Elegir la versión correcta según Plataforma (Windows 32, 64 etc…)
Las Ediciones JAVA son principalmente tres:

1. ***JSE*** *(Java Standar Edition*): Es la base y provee los fundamentos para las restantes. Con ella se puede programar y ejecutar aplicaciones de escritorio y applets. Está compuesta por el *JRE* (Java Runtime Environment) y el ***JDK*** (Java Development Kit)
2. ***JEE*** *(Java Enterprise Edition):* Plataforma multiusuario para aplicaciones empresariales JAVA.
3. ***JME*** *(JAVA Micro Edition*): Para dispositivos de escasos recursos como los dispositivos móviles.

Introducción al código

Clase y proyecto: El código se escribe en un archivo de unidad o clase. Antes de crear la clase (unidad de código), necesitamos un proyecto donde alojarla.
Package: es una agrupación de clases comunes. Equivale al concepto de namespace o librería en C#.
Objeto: es una instancia particular de una clase. La clase es la definición general y el objeto es la materialización concreta de una clase. El fenómeno de crear objetos de una clase se llama instanciación.
Los Objetos se crean (se les asigna memoria) con el Operador *new*. Tiene *atributos* (propiedades) y *métodos* (eventos)
Método*: Equivale a Procedimiento o función:* bloque de código con nombre que puede recibir o devolver parámetros ( ).
Main(): Dentro de la clase empezamos con un método o función principal llamado main() (desde donde arranca la app.)
Normalmente: public static void main() donde:

- ***public*** significa que el método es visible y puede ser llamado desde otros objetos a diferencia de private
- ***static*** significa que el método está creado en la clase, no en el objeto, y se puede llamar sin crear ningún objeto.
- ***void*** significa que el método o función no tiene valor de retorno.

Sintaxis: Reglas básicas de ortografía:

- *Las variables*: No pueden empezar por números ni contener palabras reservadas. Empiezan con el tipo de variable, luego el nombre y opcionalmente podemos asignar un valor inicial. *Ejemplo*: string nombre = "Pedro";
- Las constantes: Empiezan con la palabra "final".

Todas las líneas de código acaban en **;** al igual que C, Java distingue mayúsculas (case sensitive)
Tipos de variables usadas: string – int – float y boolean
Tipos primitivos: boolean: valores true o false - byte : Enteros entre -128 y 127 - short : Enteros entre -32768 y 32767 - int : Enteros entre -2147483648 y 2147483647 - long : Enteros largos - float : Números coma flotante - double : Números coma flotante 64-bits - Char : Caracteres.
Secuencias de escape: \n: Salto de línea \t: Tabulador \b: Backspace \r: Retorno de carro \uxxxx: Unicode

Ejemplo y explicación del primer programa: Hola mundo

```
public class Hola                    // Declaro una clase de ámbito público llamada Hola, Hola.java
{                                    // Entre llaves se declaran los atributos y métodos de la clase
  public static void main(String[] args)   //Función principal. Public: indica que el método main() es público, accesible
                                     //void: indica que la función main() no devuelve ningún valor
                                     //El método main() debe aceptar siempre como parámetro un vector de strings
  {
    System.out.println("Hola, mundo!");  // Esta línea indica que se va a ejecutar el método println(), encargado de mostrar
                                     // un valor a través de la salida estándar (en nuestro caso, un String)
  } }                                //cierra las llaves del bloque principal y de la clase.
```

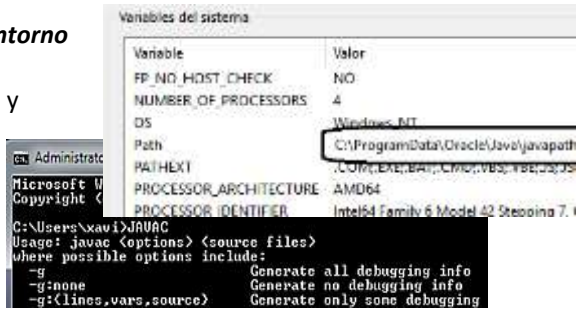
IDEs para Java

- Un **IDE** es una aplicación complementaria que nos proporciona un editor y atajos para ejecutar y depurar programas en Java.
- Los más conocidos son ***Eclipse***, ***NetBeans***, ***Visual Studio Code***, Borland ***JBuilder***, etc…
- Puedes descargar *Eclipse IDE for Java Developers* o *Netbeans* por ser gratuitos y añadir el kit JDK.

1.- Compilar y ejecutar el programa desde la consola (Sin IDE)

Poner las variables de entorno en Windows:

- En Equipo – Propiedades - Opciones Avanzadas - ***Variables de Entorno*** se debe añadir el directorio donde se instaló el JDK por defecto (C:\Archivos de Programa\Java). Añadir " ; " al final de la cadena y Salir con OK



Ejecutar la consola:

- En Inicio – Ejecutar: **CMD**
- En la consola, escribir **JAVAC** para compilar (si no aparece un error está bien configurado el path).
- Escribir JAVA (nombre_del_programa.java) para ejecutar.

<u>Ejercicio</u>: *Programa* ***Hola mundo sin IDE***

Crear un archivo de texto llamado ***Main.java***
con el texto del recuadro: →

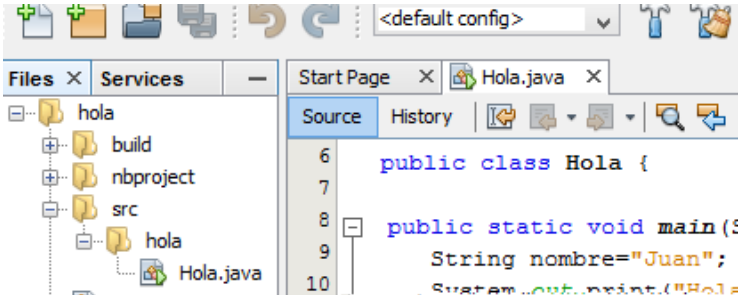
1. Guardar Main.java en la carpeta: C:\hola mundo
2. Abrir la consola (Inicio > Ejecutar > CMD)
3. Ir a la carpeta C:/hola mundo
   - Para Compilar: **javac** Main.java
   - Para Ejecutar: **java** Main.java

Ejercicio Hola Mundo

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

2.- Utilizar el IDE NETBEANS

Netbeans IDE permite utilizar este IDE para programar en otros lenguajes como PHP o C



<u>Primer proyecto en Java:</u>

- Primero creamos un proyecto nuevo escogiendo del menú:
  File – New File – ***Java Application*** (Archivo – Nuevo – Aplicación Java)
  Escribimos: ***Hola***, escogemos una carpeta y pulsamos en terminar.
  Nos mostrará un estructura prefabricada con paquetes, bibliotecas y el programa principal *main*.
- Netbeans suele guardar los proyectos como una subcarpeta dentro de la carpeta: NetBeansProjects y el archivo fuente queda guardado dentro de la carpeta SRC con la extensión .java

<u>Ejercicio:</u>

- En el proyecto nuevo ***Hola***, escribe el texto del recuadro →
  Vamos a incorporar una *<u>variable</u>* del tipo de texto (string) llamada *<u>nombre</u>* que contenga nuestro nombre.
- Pulsa al finalizar, en Run ▶ o F6 para comprobar su funcionamiento.

Ejemplo con variable

```
package hola;
public class Hola {
    public static void main(String[] args) {
       String nombre="Edu";
       System.out.print("Hola Mundo, mi nombre es   " + nombre);
    }
}
```

2.- Utilizar el IDE VSCODE

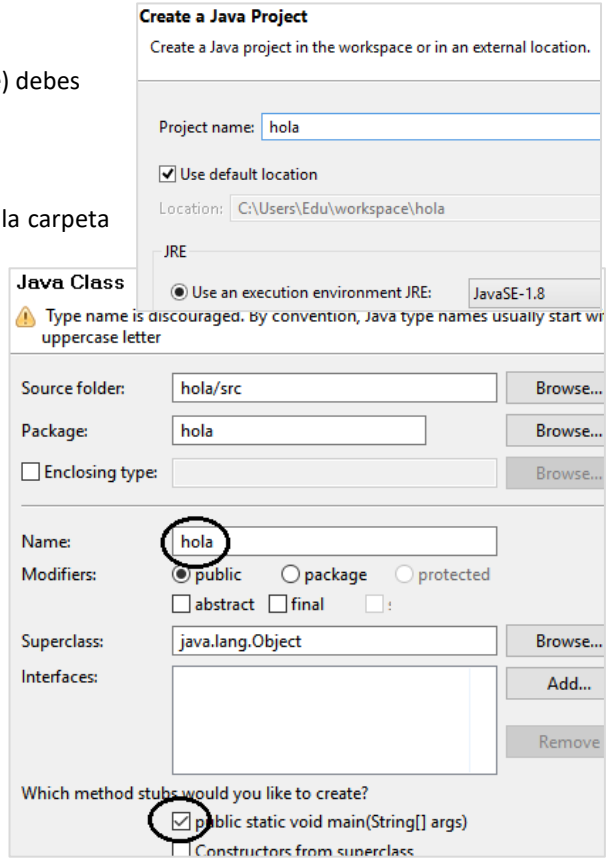
- Instala el IDE gratuito VSCode de Microsoft muy versátil, por ser multilenguaje: (code.visualstudio.com/download)
- En extensiones Busca la extensión: Java Extension Pack en el Marketplace. Es una colección de extensiones populares que pueden ayudar a escribir, probar y depurar aplicaciones Java. Remendado instalarlas todas.
- Crea la carpeta Java.
- Archivo – Nuevo Archivo… New Java Class
- En el explorador de VSCode pulsa en Create Java Project – No build Tools y selecciona la carpeta anterior Java
- Escribe el texto del recuadro →
- Pulsa en Guardar archivo
- Pulsa en ejecutar (▶Run)

```
public class Hola {
    public static void main(String[] args) {
       String nombre="Edu";
       System.out.print("Hola Mundo, mi nombre es   " + nombre);
    }
}
```

4.- Utilizar el IDE ECLIPSE

Eclipse es un entorno de desarrollo integrado (IDE) para programación, desarrollo y compilación programas en C++ o aplicaciones Java.
Descarga el programa desde *www.eclipse.org*. El clásico o estándar es suficiente.
Debes tener instalado también el kit de desarrollo java: JDK.
JRE Java Runtime Environment, no incluye las herramientas de desarrollo sólo ejecuta las aplicaciones Java.

Ejercicio 1



1. Ejecuta *Eclipse*. Si te propone un área de trabajo (workspace) debes indicar la carpeta donde quieres guardar los proyectos.
2. Pulsa en File: ***New – Java Project***.
   Pon el nombre al proyecto: *hola*
   Al iniciar el IDE verás en el panel izquierdo el proyecto con la carpeta ***src***.
   - hola
     - src
     - JRE System Library
3. Pulsa en File: ***New – Class***
   Pon el nombre a la clase: ***Hola***.
4. Activa la casilla: ☑ *public static void main(..)*
   esto te escribirá la función principal *main* dentro de la clase.
5. Dentro del documento, añade entre las llaves la línea que se muestra:

```
*hola.java
1  package hola;
2
3  public class hola {
4
5      public static void main(String[] args) {
6        // TODO Auto-generated method stub
7        System.out.print("Hola, mundo soy Edu");
8      }
9  }
```

6. Pulsa el botón Run y comprueba en el panel de la ***Consola*** cómo se muestra el resultado:

```
Console   doc   Declaration
<terminated> hola [Java Application] C:\Program Files\
Hola, mundo soy Edu
```

Ejercicio 2: Incorporar una biblioteca *(librería)* para ejecutar en ventana de Windows

Una libería es un programa anexo que aprovechará nuestro programa. Para ello se debe hacer constar al principio de nuestro programa con la instrucción ***import***

Para mostrar el mensaje en una ventana de Windows, podemos importar la librería para Windows: **javax.swing**
Luego utilizar su función *showMessageDialog* para mostrar la ventana de mensaje

Otra librería muy utilizada es ***util.Scanner*** para la introducción de datos.

```
import javax.swing.*;  //librería para Windows

public class Hola2 {

   /**
    * @param args the command line arguments
    */
   public static void main(String[] args) {
      JOptionPane.showMessageDialog(null,"hola en ventana");
   }
}
```

Véase Referencias: http://www.edu4java.com/es/progbasica/progbasica4.html

Algoritmo. Proceso y secuencia de un programa

Un algoritmo es el proceso para obtener un resultado.

Ejercicio 3. Planteamiento y ejecución de un algoritmo.

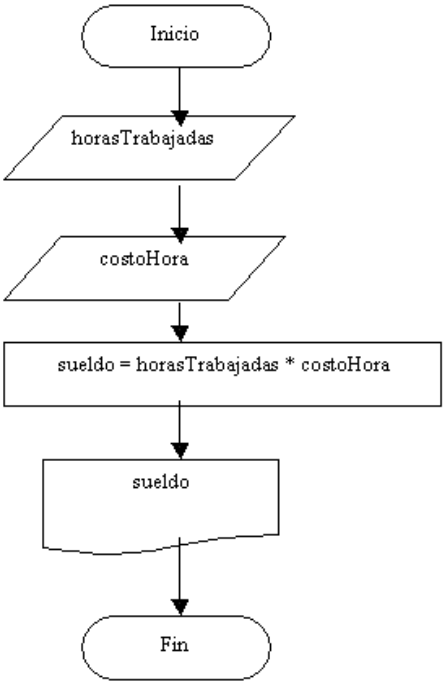
En nuestro ejemplo, queremos calcular el sueldo de un trabajador. Seguiremos estos 4 pasos:

1. *Primero Planteamiento*

Hacemos un boceto de la secuencia o diagrama de flujo →

o escribimos en sucio (pseudocódigo) el planteamiento del programa.

```
main

horasTrabajadas=teclado.nextInt();
(Carga por teclado)

costoHora=teclado.nextFloat();
(Carga por teclado)

sueldo=horasTrabajadas*costoHora;
(Operación)

System.out.print(sueldo);
(Salida por pantalla)
```



2. *Creación del Proyecto*:
   Creamos un nuevo proyecto escribiendo el "Name: ***SueldoOperario***
3. *Creación de la Clase*:
   Creamos la clase con el mismo nombre que le asignamos al proyecto, es decir: *SueldoOperario*. (esto no es obligatorio, un proyecto puede contener varias clases). Escribimos el código del recuadro.

```
import java.util.Scanner;                                   // librería externa para entrada por teclado

public class SueldoOperario {                               //Declaración de la clase

    public static void main(String[ ] ar) {                 //Declaración del método-función main
        Scanner teclado = new Scanner(System.in);           //objeto teclado de la clase Scanner
        int horasTrabajadas;                                //Declaración variable numérica entera
        float costeHora;                                    //Declaración variable numérica decimal
        float sueldo;                                       //Declaración variable numérica decimal
        System.out.print("Ingrese la cantidad de horas trabajadas:");
        horasTrabajadas=teclado.nextInt();                  //Entrada de valor entero por teclado
        System.out.print("Ingrese el coste de la hora:");
        costeHora = teclado.nextFloat();                    //Entrada de valor decimal por teclado
        sueldo = horasTrabajadas * costeHora;
        System.out.print("El empleado debe cobrar:");
        System.out.print(sueldo);                           //Salida por consola de la variable sueldo
    }
}
```

4. *Ejecutamos el programa*:
   Si no hay errores sintácticos, comprobamos en la ventana de la "Console" su funcionamiento, introduciendo los dos datos que nos solicitan (las horas trabajadas y el precio de la hora)

<u>Características de las clases:</u>

- ***Encapsulación***: Las clases pueden ser declaradas como public, private, pritected y package
- ***Herencia***: Cuando una clase se crea a partir de otra, hereda sus variables y métodos.
- ***Polimorfismo***: Objetos de la misma clase o tipo se pueden agrupar y tratar al mismo tiempo.

Explicación y conceptos básicos

Concepto de una clase. La programación orientada a objetos está basada en clases; que es una agrupación de datos y funciones (como un objeto). 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. Toda clase debe tener llaves de apertura y cierre.

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 qué 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.

Importar una clase: Cuando se requieren utilizar otras clases externas debemos importarlas al inicio de nuestro programa. En este caso 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 (ya se verá).

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

De ahora en adelante se sobreentiende el proceso general de declarar un proyecto, declarar una clase y definir una función main.

Variables:

Hemos definido tres variables: (horasTrabajadas, costoHora,sueldo), y el tipo de datos que almacenarán. La cantidad de horas normalmente será un valor entero (ej. 100 - 150 - 230 etc.), pero el costo de la hora y el sueldo serán decimales. La definición de las variables la hacemos dentro de la main:

```
int horasTrabajadas;      //-> variable numérica entera
float costoHora;          //-> variable decimal flotante
float sueldo;             //-> variable decimal flotante
```

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 caracter 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 qué almacenan (no es conveniente llamar a nombres de variables con letras individuales)

Ámbito de las variables: Las variables pueden ser *public* o *private* según su alcance.

Tipos de variables:

Tipo de variable	Descripción
Boolean	1 byte. Valores true y false
Char	2 bytes. Unicode. Comprende el código ASCII
Byte	1 byte. Valor entero entre -128 y 127
Short	2 bytes. Valor entero entre -32768 y 32767
Int	4 bytes. Valor entero entre -2.147.483.648 y 2.147.483.647
Long	8 bytes. Valor entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807
Float	4 bytes (entre 6 y 7 cifras decimales equivalentes). De -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38
Double	8 bytes (unas 15 cifras decimales equivalentes). De -1.79769313486232E308 a -4.94065645841247E-324 y de 4.94065645841247E-324 a 1.79769313486232E308

Asignar un valor a una variable:

Normalmente se utiliza el operador de asignación **=**

Pero también existen los operadores *unarios* (+ y -) o los *incrementales* (++ y --) que suben o bajan el valor una unidad.

Operadores relacionales. Sirven para comparar valores o variables.

Operador	Utilización	El resultado es true
>	op1 > op2	si op1 es mayor que op2
>=	op1 >= op2	si op1 es mayor o igual que op2
<	op1 < op2	si op1 es menor que op2
<=	op1 <= op2	si op1 es menor o igual que op2
==	op1 == op2	si op1 y op2 son iguales
!=	op1 != op2	si op1 y op2 son diferentes

Operadores lógicos:

Operador	Nombre	Utilización	Resultado
&&	AND	op1 && op2	true si op1 y op2 son true. Si op1 es false ya no se evalúa op2
||	OR	op1 || op2	true si op1 u op2 son true. Si op1 es true ya no se evalúa op2
!	negación	! op	true si op es false y false si op es true
&	AND	op1 & op2	true si op1 y op2 son true. Siempre se evalúa op2
|	OR	op1 | op2	true si op1 u op2 son true. Siempre se evalúa op2

Mostrar mensajes:

En la "Console" utilizamos la sintaxis: ***System.out.print*** con estos complementos:

```
System.out.print("Cantidad de horas trabajadas:");
```

Todo lo que se encuentra contenido entre comillas aparecerá exactamente en la ventana de la "Console".

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

Aparecerá el valor almacenado en la variable sueldo y no el mensaje "sueldo".

Entrada de datos:

Por teclado en Java se complica. Importaremos una clase llamada ***Scanner*** que nos facilita el ingreso de datos, que se encuentra en el paquete *java.util* en la primer línea de nuestro programa.

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

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

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

La clase Scanner, que tiene un método **next**() que nos da la siguiente entrada por consola y que guardaremos en la variable de entrada:

- `String entrada = scanner.next();` -> Si la variable de entrada es de texto
- `horasTrabajadas=teclado.nextInt();` -> Si pedimos un número entero
- `costoHora=teclado.nextFloat();` -> Si pedimos un valor decimal *float*

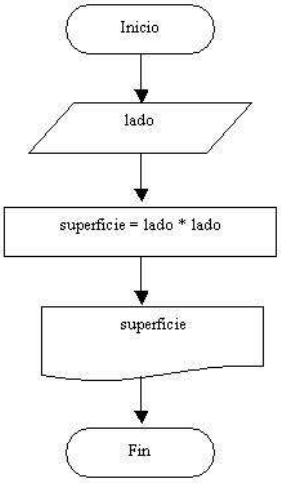
Las operaciones que indicamos en el diagrama de flujo mediante la figura rectángulo la codificamos tal cual:

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

Ejercicio 4. Errores. *Hallar la superficie de un cuadrado conociendo el valor de un lado.*

Creemos un proyecto llamado: ***SuperficieCuadrado*** y una clase llamada: ***SuperficieCuadrado***.

```
import java.util.Scanner;  //importamos la clase scanner

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



*Ejercicio de **error de sintaxis***: cambia la palabra superficie de la última línea por:

System.out.print(Superficie);

Esto provocará un error de sintaxis ya que los nombres de las variables distinguen las mayúsculas.

*Ejercicio de **error lógico***: cambia la línea: superficie=lado * lado; por: superficie=lado * lado*lado

Esto provocará un error lógico o de cálculo en tu programa.

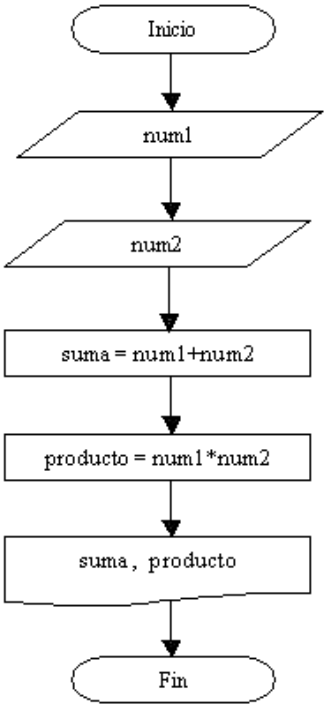
Estructura secuencial

El método ejecuta las órdenes por orden de arriba abajo. El *flujorama* fluye en una única dirección.

Ejercicio 5: *Leer dos números enteros por teclado e imprimir su suma y su producto.*

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.

```
import java.util.Scanner;

public class SumaProductoNumeros {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int num1,num2,suma,producto;         //variables int
        System.out.print("Primer valor:");
        num1=teclado.nextInt();             //entrada entera
        System.out.print("Segundo valor");
        num2=teclado.nextInt();
        suma = num1 + num2;
        producto = num1 * num2;
        System.out.print("La suma es:");
        System.out.println(suma);
        System.out.print("El producto es:");
        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);

Operadores

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

Operadores Relacionales:	*Operadores Matemáticos*
> (mayor)	+ (más)
< (menor)	- (menos)
>= (mayor o igual)	* (producto)
<= (menor o igual)	/ (división)
== (igual)	% (resto de una división)
!= (distinto)	Ej.: x=13%5; {se guarda 3}

Ejemplo:

- Pedir un número e indicar si es positivo o negativo

```
import java.util.Scanner;
public class pepe {
 public static void main(String[] args) {
    Scanner teclado=new Scanner(System.in);
    int num;
    System.out.print("Introduce un número: ");
    num=teclado.nextInt();
    if( num < 0)
    System.out.println("Es Negativo");
    Else             // suponemos que el 0 es positivo.
    System.out.println("Es Positivo");
  }
}
```

Problemas propuestos estructura secuencial

1. *Realizar un programa que lea cuatro valores numéricos e informar su suma y promedio.*
2. *Leer el lado de un cuadrado y mostrar por pantalla su perímetro (multiplicando el valor del lado por cuatro)*
3. *Se debe desarrollar un programa que pida el precio de un artículo y la cantidad. Mostrar el importe a abonar.*
4. *Pedir el radio de un círculo y calcular su área y perímetro. a=PI*r^2 ; p=2*PI*r*

   Nota: Usamos la librería Math para el número PI: `a=Math.PI*(r*r);` `Potencia: Math.pow (r, 2)`

```
// *** 1 suma y promedio de 4 valores***
import java.util.Scanner;

public class SumaPromedio {
    public static void main(String[] ar) {
    Scanner teclado=new Scanner(System.in);
    int num1,num2,num3,num4,suma,promedio;
    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();
    System.out.print("Ingrese cuarto valor:");
    num4=teclado.nextInt();
    suma=num1 + num2 + num3 + num4;
    promedio=suma/4;
    System.out.print("La suma de los cuatro valores
    es:");
    System.out.println(suma);
    System.out.print("El promedio es:");
    System.out.print(promedio);
    }
}
```

```
//***3 Cálculo del importe ***

import java.util.Scanner;

public class Importe {
    public static void main(String[] ar) {
    Scanner teclado=new Scanner(System.in);
    int cantidad;
    float precio;
    float importe;
    System.out.print("Cantidad de productos:");
    cantidad=teclado.nextInt();
    System.out.print("Precio unitario del
    producto:");
    precio=teclado.nextFloat();
    importe=precio * cantidad;
    System.out.print("El importe a pagar es:");
    System.out.print(importe);
    }
}
```

```
// *** 2 Cálculo del perímetro de un cuadrado***
import java.util.Scanner;

public class PerimetroCuadrado {
    public static void main(String[] ar) {
    Scanner teclado=new Scanner(System.in);
    int lado,perimetro;
    System.out.print("Lado del cuadrado:");
    lado=teclado.nextInt();
    perimetro=lado * 4;
    System.out.print("El perímetro es:");
    System.out.print(perimetro);
    }
}
```

```
//*** 4 Área y perímetro del círculo ***

package circulo;
import java.util.Scanner;
public class Circulo {

  public static void main(String[] args) {
    Scanner entrada = new Scanner(System.in);
    double a,r,p; // variables área, radio y
      perimetro

    System.out.print("Radio del círculo: ");
    r = entrada.nextFloat();
    a=Math.PI*(r*r);   //o usar libreria math *
    // para elevar al cuadrado mejor: Math.pow (r, 2)
    System.out.println("El área de un círculo de
      radio " + r+ " es: " + a);
    p=2*Math.PI*r;
    System.out.println("El perímetro de una
      circunferencia de radio " + r+ " es: " + p);
  }
}
```

**** Funciones incorporadas en la librería Math***

abs(x)	valor absoluto
acos(x)	arco coseno
asin(x)	arco seno
atan(x)	arco tangente
atan2(x,y)	representación polar de x,y
ceil(x)	menor entero mayor que x
cos(x)	coseno
exp(x)	e^x
floor(x)	mayor entero menor que x
IEEEremainder(x,y)	resto de la división x/y según IEEE
log(x)	logaritmo natural
max(x,y)	el mayor de x e y
min(x,y)	el menor de x e y
pow(x,y)	x^y
random()	número aleatorio entre 0 y 1
rint(x)	entero más cercano a x (devuelve doble)
round(x)	entero más cercano a x
sin(x)	seno
sqrt(x)	raíz cuadrada
tan(x)	tangente

Estructuras condicionales (simples, compuestas y anidadas)

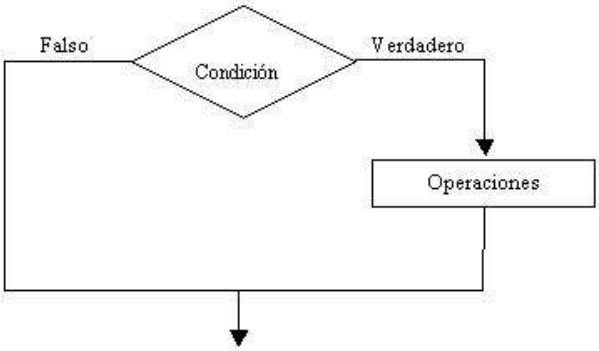
Cuando hay que tomar una decisión. Cuando se presenta la elección tenemos la opción de realizar una actividad o no.

a) Estructura condicional simple.

En la representación gráfica: →

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.



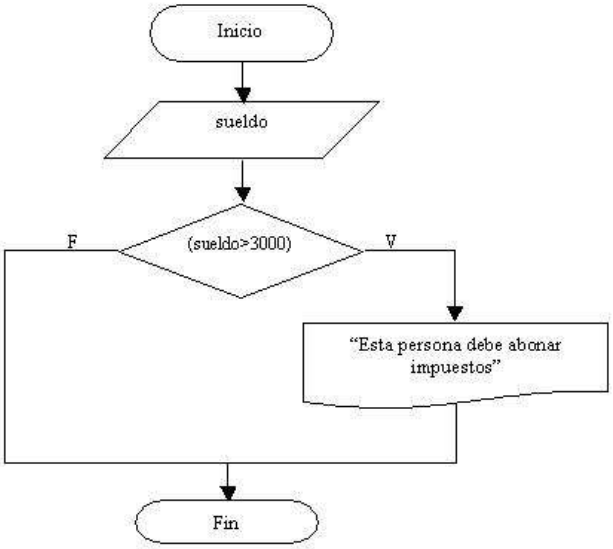
Ejercicio 6: Problema estructura condicional simple

Archivo – Nuevo proyecto –Java application: ***condicionalsimple***

Introducir el sueldo de una persona. Si supera los 3000 € mostrar un mensaje en pantalla indicando que debe abonar impuestos.

Diagrama de flujo:

```
package condicionalsimple;
import java.util.Scanner;
public class Condicionalsimple {
public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        float sueldo;               //variable decimal
        System.out.print("Ingrese el sueldo:");
        sueldo=teclado.nextFloat();
        if (sueldo>3000) {
            System.out.println("Abonar impuestos");
        }
    }
}
```



La palabra clave "***if***" indica inicio del condicional; seguidamente disponemos la condición entre paréntesis. Por último encerrada entre llaves { }las instrucciones de la rama del verdadero.

Operadores

En una condición pueden compararse variables y valores constantes con los operadores relacionales.

Relacionales:	Matemáticos	Lógicos
> (mayor) < (menor) >= (mayor o igual) <= (menor o igual) == (igual) != (distinto)	+ (más) - (menos) * (producto) / (división) % (resto de una división)	&& (Y) || (O)

Ejemplo operador matemático: x=13%5; {se guarda 3}

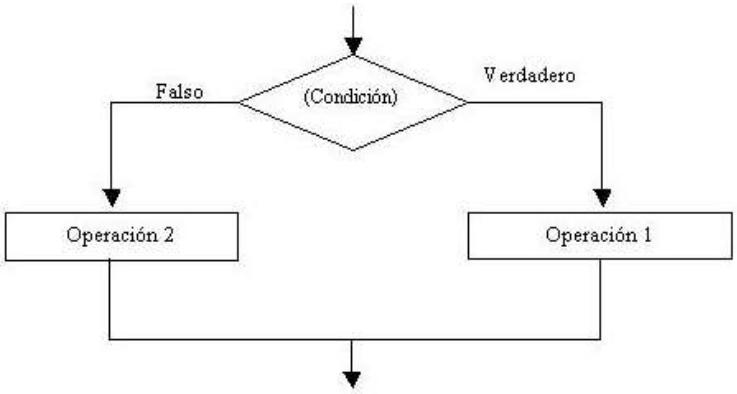
Ejemplos operador relacional:

- Al pedir un número, comprobar que es distinto a 0. (!=) -> `if(num!=0) System.out.print("Es cero");`
- Comprobar si son iguales. (==) `if(num1==num2) System.out.print("Ambos súmeros son iguales");`

Ejemplo operador lógico: Comprueba si es 1 ó 2: `if(num1==1 || num1==2) System.out.print("es uno o cero");`

b) Estructura condicional compuesta.

En una estructura condicional compuesta tenemos acciones, tanto por la rama del verdadero como por la rama del falso, pero NUNCA se ejecutan las dos ramas a la vez.



Ejercicio 7.

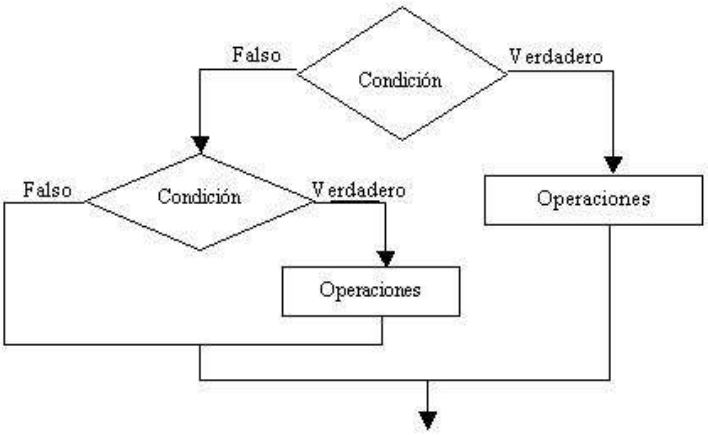
Realizar un programa que solicite dos números distintos y muestre por pantalla el mayor de ellos.

Explicación: Se pide la entrada de num1 y num2 por teclado. Comparamos si el valor de num1 es mayor (>) que el valor de num2, si la respuesta es verdadera vamos por la rama de la derecha y mostramos num1, en caso contrario mostramos num2.

```
package numeromayor;
import java.util.Scanner;
public class condicionalCompuesto {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int num1,num2;
        System.out.print("Primer valor:");
        num1=teclado.nextInt();
        System.out.print("Segundo valor:");
        num2=teclado.nextInt();
        if (num1>num2) {
                System.out.print(num1);
        } else {
                System.out.print(num2);
        }
    }
}
```

c) Estructuras condicionales anidadas

Cuando por la rama del verdadero o el falso de una estructura condicional hay otra estructura condicional.



Ejercicio 7b: *variante*
Pedir dos números y decir cuál es el mayor o si son iguales.

```
package numeromayor2;
public static void main(String[] args) {
Scanner teclado=new Scanner(System.in);
int num1,num2;
System.out.print("Primer valor: ");
num1=teclado.nextInt();
System.out.print("Segundo valor: ");
num2= teclado.nextInt();
if(num1==num2)  System.out.println("Son iguales");
else
  {
  if(num1>num2)
  System.out.println(num1 + " es mayor que " + num2);
    else
    System.out.println(num2 + " es mayor que " + num1);
```

Problemas propuestos de condicionales

1. CondicionCompuesta1: Pedir 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. Aprobado: Pedir tres notas de un alumno, si el promedio es mayor o igual a siete mostrar un mensaje "Aprobado".
3. Digitos: Pedir un número positivo de uno o dos dígitos (1..99) e informar si el número tiene uno o dos dígitos.
4. Capicua: Pedir un número entre 0 y 9.999, decir si es capicúa
5. Notas5: 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 >=7 mostrar "Promocionado". Si el promedio es >=5 y <7 mostrar "Regular". Si el promedio es <5 mostrar "Suspendido"

```
//** 1. CondicionCompuesta1 ***
import java.util.Scanner;
public class EstructuraCondicionalCompuesta {
    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) {
       int suma,diferencia;
       suma=num1 + num2;
       diferencia=num1 - num2;
       System.out.print("La suma de los dos valores es:");
       System.out.println(suma);
       System.out.print("y su diferencia es:");
       System.out.println(diferencia);
     } else {
       int producto,division;
       producto=num1 * num2;
       division=num1 / num2;
       System.out.print("El producto de los dos valores
es:");
       System.out.println(producto);
       System.out.print("La división de los dos valores
es:");
       System.out.println(division);
     }
    }
}

//*** 4.- Capicua *** con operador Y ( &&  )
import java.util.Scanner;
public class Capicua {
public static void main(String[] args) {
Scanner teclado=new Scanner(System.in);
int num;
int dm, um, c, d, u;
//dm um c d u: dm (decenas de millar), um:(unidades de
millar) c: (centenas), d: (decenas), u: (unidades)
System.out.print("Introduzca un número entre 0 y 99.999: ");
num=teclado.nextInt();
u = num % 10; // unidad
num = num / 10; // decenas
d = num % 10;
num = num / 10; // centenas
c = num % 10;
num = num / 10; // unidades de millar
um = num % 10;
num = num / 10; // decenas de millar
dm = num;
// capicúa si las cifras son iguales dos a dos por los
extremos , las centenas no las tenemos en cuenta
if (dm == u && um == d)  // si … Y …
System.out.println ("el número es capicúa");
else
System.out.println ("el número NO es capicúa");
//el número 121 es similar al 00121 y 121 lo identifica como
NO capicúa. No tener en cuenta este pequeño error.
}
}
```

```
//** 2. Aprobado**
import java.util.Scanner;

public class Aprobado {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int nota1,nota2,nota3;
        System.out.print("Primera nota:");
        nota1=teclado.nextInt();
        System.out.print("Segunda nota:");
        nota2=teclado.nextInt();
        System.out.print("Tercera nota:");
        nota3=teclado.nextInt();
        int promedio;
        promedio=(nota1 + nota2 + nota3) / 3;
        if (promedio>=7) {
            System.out.print("Promocionado");
        }
    }
}


//3. ** digitos **
import java.util.Scanner;

public class digitos {
    public static void main(String[] ar) {
     Scanner teclado=new Scanner(System.in);
     int num;
     System.out.print("Ingrese un valor entero
de 1 o 2 dígitos:");
     num=teclado.nextInt();
     if (num<10) {
         System.out.print("Tiene un dígito");
     } else {
         System.out.print("Tiene dos dígitos");
     }
    }
}
//*** 5.- Notas5 ***
import java.util.Scanner;
public class Notas5 {
    public static void main(String[] ar) {
      Scanner teclado=new Scanner(System.in);
      int nota1,nota2,nota3;
      System.out.print("Primer nota:");
      nota1=teclado.nextInt()
      System.out.print("Segunda nota:");
      nota2=teclado.nextInt();
      System.out.print("Tercera nota:");
      nota3=teclado.nextInt();
      int promedio=(nota1 + nota2 + nota3)/3;
      if (promedio>=7) {
          System.out.print("Promocionado");
      } else {
          if (promedio>=5) {
              System.out.print("Regular");
          } else {
              System.out.print("Suspendido");
          }
        }}}
```

Problemas propuestos

1. ElMayordeTres : Se cargan por teclado tres números distintos. Mostrar por pantalla el mayor de ellos.
2. PositivoNegativo: Pedir por teclado un valor entero, mostrar si el número es positivo, nulo o negativo.
3. Digitos: 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. TestCandidato: Un candidato, realiza un test y 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>=90%. Nivel medio: Porcentaje>=75% y <90%.
   Nivel regular: Porcentaje>=50% y <75%. Fuera de nivel: Porcentaje<50%

```java
//** 1.- el mayor de tres numeros***
import java.util.Scanner;
public class ElMayordeTres {
    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 segunda valor:");
        num2=teclado.nextInt();
        System.out.print("Ingrese tercer valor:");
        num3=teclado.nextInt();
        if (num1>num2) {
            if (num1>num3) {
                System.out.print(num1);
            } else {
                System.out.println(num3);
            }
        } else {
            if (num2>num3) {
                System.out.print(num2);
            } else {
                System.out.print(num3);
            }
        }
    }
}
```

```java
//** 2.- Positivo o nevativo ***
import java.util.Scanner;

public class PositivoNegativo {
    public static void main(String[] ar) {
     Scanner teclado=new Scanner(System.in);
     int num;
     System.out.print("Ingrese un valor:");
     num=teclado.nextInt();
     if (num==0) {
     System.out.print("Se ingresó el cero");
      } else {
        if (num>0) {
        System.out.print("Se ingresó un valor
        positivo");
        } else {
    System.out.print("Se ingresó un valor negativo");
            }
        }
    }
}
```

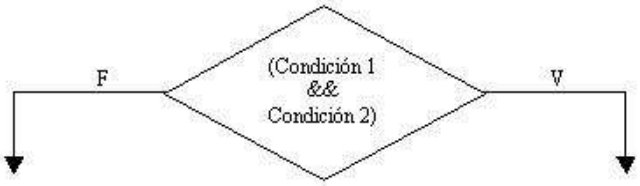
```java
//** 3.- digitos ***
import java.util.Scanner;
public class digitos {
    public static void main(String[] ar) {
    Scanner teclado=new Scanner(System.in);
    int num;
    System.out.print("Introduzca un valor de hasta
tres dígitos positivo:");
    num=teclado.nextInt();
        if (num<10) {
         System.out.print("Tiene un dígito");
        }
        else {
         if (num<100) {
          System.out.print("Tiene dos dígitos");
          } else {
          if (num<1000) {
          System.out.print("Tiene tres dígitos");
          } else {
           System.out.print("Error en la entrada de
           datos.");
          }
         }
        }
    }
}
```

```java
// ** 4 Calificaciones **
import java.util.Scanner;

public class TestCandidato {
  public static void main(String[] ar) {
   Scanner teclado=new Scanner(System.in);
   int totalPreguntas,totalCorrectas;
   System.out.print("Cantidad total de preguntas del
   examen:");
   totalPreguntas=teclado.nextInt();
   System.out.print("Cantidad preguntas contestadas
   correctamente:");
   totalCorrectas=teclado.nextInt();
   int porcentaje=totalCorrectas * 100 /
   totalPreguntas;
    if (porcentaje>=90) {
          System.out.print("Notable");
      } else {
          if (porcentaje>=75) {
          System.out.print("Bien");
          } else {
          if (porcentaje>=50) {
          System.out.print("Suficiente");
          } else {
          System.out.print("Insuficiente");
          }
          }
      }
   }
}
```

Condiciones compuestas con operadores lógicos

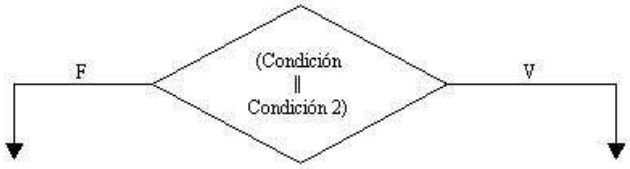
Operador && (Y)

Traducido se lo lee como "**Y**".
Debe cumplir ambas condiciones a la vez.



Operador ||

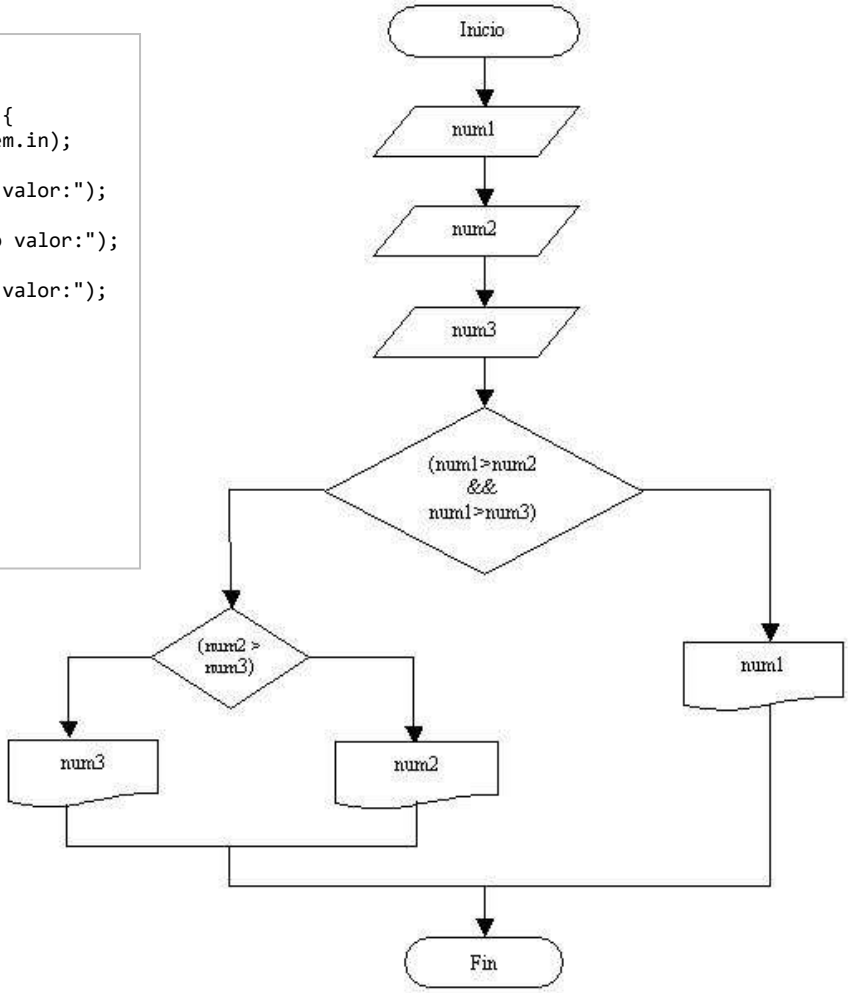
Traducido se lo lee como "**O**".
Con que una de las dos condiciones sea Verdadera, el resultado de la condición compuesta sea Verdadero.



Ejercicio 9:

Leer por teclado tres números distintos y nos muestre el mayor.

```
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);
            }}
    }}
```



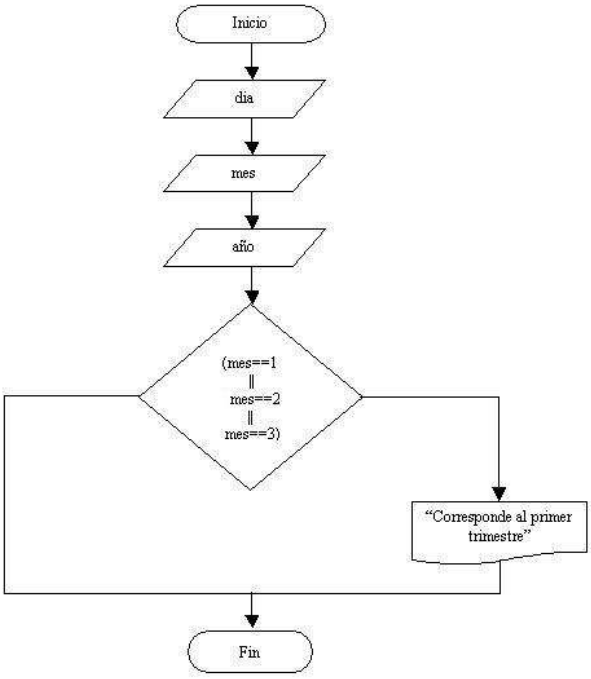
Explicación.

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.
Se mostrará el contenido de num1 si y sólo si num1>num2 y num1>num3 y en caso de ser Falsa la condición, analizamos el contenido de num2 y num3 para ver cual tiene un valor mayor.

Ejercicio 10:

a) Cargar 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.

```
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("Núm. de día:");
  dia=teclado.nextInt();
  System.out.print("Núm. de mes:");
  mes=teclado.nextInt();
  System.out.print("Núm. de año:");
  año=teclado.nextInt();
  if (mes==1 || mes==2 || mes==3) {
  System.out.print("Corresponde a primer trimestre");
   }
```



b) Cargar tres notas por teclado y mostrar si debe recuperar o no. Dos variantes utilizando || ó &&

```
 public static void main(String[] args) {
     Scanner teclado=new Scanner(System.in);
     float nota1,nota2,nota3;
     System.out.print("Nota 1:");
     nota1=teclado.nextFloat();
     System.out.print("Nota 2:");
     nota2=teclado.nextFloat();
     System.out.print("Nota 3:");
     nota3=teclado.nextFloat();
     if (nota1<5 || nota2<5 || nota3<5) {
         System.out.print("Debes recuperar");
     }
```

```
public static void main(String[] args) {
    Scanner teclado=new Scanner(System.in);
    float nota1,nota2,nota3;
    System.out.print("Nota 1:");
    nota1=teclado.nextFloat();
    System.out.print("Nota 2:");
    nota2=teclado.nextFloat();
    System.out.print("Nota 3:");
    nota3=teclado.nextFloat();
    if (nota1>=5 && nota2>=5 && nota3>=5) {
    System.out.print("No debes recuperar");
    }
```

c) Pedir el día, mes y año de una fecha e indicar si la fecha es correcta. Con meses de 28, 30 y 31 días. Sin años bisiestos.

```
public static void main(String[] args) {
Scanner teclado=new Scanner(System.in);
int dia,mes,año;
System.out.print("Introduzca día: ");
dia= teclado.nextInt();
System.out.print("Introduzca mes: ");
mes= teclado.nextInt();
System.out.print("Introduzca año: ");
año= teclado.nextInt();
// el único año que no existe es el 0
if(año==0)
  System.out.println("Fecha incorrecta");
  else{
    if(mes==2 && (dia>=1 && dia<=28))
    System.out.println(dia + "/" + mes + "/" + año+": Fecha correcta");
    else{
    if((mes==4 || mes==6 || mes==9 || mes==11) &&
    (dia>=1 && dia<=30))
    System.out.println(dia + "/" + mes + "/" + año+": Fecha correcta");
    else{
    if( (mes==1 || mes==3 || mes==5 || mes==7 || mes==8 || mes==10 || mes==12) &&
    (dia>=1 && dia<=31))
    System.out.println(dia + "/" + mes + "/" + año+": Fecha correcta");
    else
    System.out.println("Fecha incorrecta");
}}
}}
}
```

Problemas propuestos

1. Navidad: Realizar un programa que pida una fecha y verifique si dicha fecha corresponde a Navidad.
2. TodosIguales: 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. MenoresaDiezTodos: 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. MenoresaDiezAlguno: 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. Cuadrante: 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. Antigüedad: 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. Rango: 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)
8. CursoOfimegaJava: ¿Necesitas un curso de Java? Si la respuesta en NO o N mostrará: “Necesitas un curso de Java” Nota: la comparación con variables de texto se explica más adelante pero de momento usaremos la función: `equalsIgnoreCase`

```
import java.util.Scanner;
public class Navidad {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int dia,mes,año;
        System.out.print("Nº de día:");
        dia=teclado.nextInt();
        System.out.print("Nº de mes:");
        mes=teclado.nextInt();
        System.out.print("Nº de año:");
        año=teclado.nextInt();
        if (mes==12 && dia==25) {
            System.out.print("Es Navidad.");
        }
    }
}
```

```
import java.util.Scanner;
public class TodosIguales {
    public static void main(String[] ar) {
     Scanner teclado=new Scanner(System.in);
     int num1,num2,num3;
     System.out.print("Primer valor:");
     num1=teclado.nextInt();
     System.out.print("Segundo valor:");
     num2=teclado.nextInt();
     System.out.print("Tercer valor:");
     num3=teclado.nextInt();
     if (num1==num2 && num1==num3) {
       int suma=num1 + num2;
       System.out.print("La suma del primero y
       segundo:");
       System.out.println(suma);
       int producto=suma * num3;
       System.out.print("La suma del primero y
       segundo multiplicado por el tercero:");
       System.out.print(producto);
```

```
import java.util.Scanner;
public class MenoresaDiezTodos {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int num1,num2,num3;
        System.out.print("Primer valor:");
        num1=teclado.nextInt();
        System.out.print("Segundo valor:");
        num2=teclado.nextInt();
        System.out.print("Tercer valor:");
        num3=teclado.nextInt();
        if (num1<10 && num2<10 && num3<10) {
            System.out.print("Todos    los    números    son
    menores a diez");
        }
    }
}
```

```
import java.util.Scanner;
public class MenoresaDiezAlguno {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int num1,num2,num3;
        System.out.print("Primer valor:");
        num1=teclado.nextInt();
        System.out.print("Segundo valor:");
        num2=teclado.nextInt();
        System.out.print("Tercer valor:");
        num3=teclado.nextInt();
        if (num1<10 || num2<10 || num3<10) {
           System.out.print("Alguno de los números es
           menor a diez");
        }
    }
}
```

```java
import java.util.Scanner;

public class Cuadrante {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int x,y;
        System.out.print("Ingrese coordenada x:");
        x=teclado.nextInt();
        System.out.print("Ingrese coordenada y:");
        y=teclado.nextInt();
        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 2º cuadrante");
     } else {
      if (x<0 && y<0) {
      System.out.print("Se encuentra en el 3er cuadrante");
      }
        else {
      System.out.print("Se encuentra en el 4º cuadrante");
        }
       }
     }
  }
}
```

```java
import java.util.Scanner;
public class Antiguedad {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        float sueldo;
        int antiguedad;
        System.out.print("Ingrese sueldo del empleado:");
        sueldo=teclado.nextFloat();
        System.out.print("Antiguedad en años:");
        antiguedad=teclado.nextInt();
        if (sueldo<500 && antiguedad>10) {
            float aumento=sueldo * 0.20f;
            float sueldoTotal=sueldo+aumento;
            System.out.print("Sueldo a pagar:");
            System.out.print(sueldoTotal);
        } else {
            if (sueldo<500) {
                float aumento=sueldo * 0.05f;
                float sueldoTotal=sueldo+aumento;
                System.out.print("Sueldo a pagar:");
                System.out.print(sueldoTotal);
            } else {
                System.out.print("Sueldo a pagar:");
                System.out.print(sueldo);
            }
        }
    }
}
```

```java
import java.util.Scanner;

public class Rango {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int num1,num2,num3;
        System.out.print("Primer valor:");
        num1=teclado.nextInt();
        System.out.print("Segundo valor:");
        num2=teclado.nextInt();
        System.out.print("Tercer valor:");
        num3=teclado.nextInt();
        System.out.print("Rango de valores:");
        if (num1<num2 && num1<num3) {
            System.out.print(num1);
        } else {
            if (num2<num3) {
                System.out.print(num2);
            } else {
                System.out.print(num3);
            }
        }
        System.out.print("-");
        if (num1>num2 && num1>num3) {
            System.out.print(num1);
        } else {
            if (num2>num3) {
                System.out.print(num2);
            } else {
                System.out.print(num3);
            }
        }
    }
}
```

```java
package cursoofimegajava;
/**
 * Práctica CURSO DE JAVA programación
 * @author Ofimega acadèmies Salou
 */
import java.util.Scanner;
public class CursoOfimegaJava {
     public static void main(String[] args) {
    Scanner lector = new Scanner(System.in);
    String respuesta;
    System.out.println("¿Sabes programar con
    Java?");
    respuesta=lector.next();
    if (respuesta.equalsIgnoreCase("No")==true
    || respuesta.equalsIgnoreCase("N")==true){
            System.out.println("Necesitas un
    curso de Java");
        }else{System.out.println("OK");
    }
}}
```

Instrucción switch – case

La sentencia switch te da la opción de testear un rango de valores de variables. Se pueden utilizar en caso de sentencias if-else complejas o excesivamente largas. El valor a comparar debe ser numeral y cada caso debe cerrarse con break

Ejemplo 1: Crear un menú con 4 opciones para escoger entre sumar, restar, multiplicar o dividir dos números.
Ejemplo 2: *Pedir una nota numérica entera entre 0 y 10, y mostrar dicha nota de la forma: cero, uno, dos, tres...*
Ejemplo 3: Pedir un número de 0 a 99 y mostrarlo escrito. Por ejemplo, para 56 mostrar: cincuenta y seis.

```
public class NotaTexto {
public static void main(String[] args) {
int num;
System.out.print("Introduzca nota (0-10):");
num=Entrada.entero();
switch(num){
  case 0:
    System.out.println("CERO");
    break;
  case 1:
    System.out.println("UNO");
    break;
  case 2:
    System.out.println("DOS");
    break;
  case 3:
    System.out.println("TRES");
    break;
  case 4:
    System.out.println("CUATRO");
    break;
  case 5:

  ...

  case 10:
    System.out.println("DIEZ");
    break;}}
```

```
import java.util.Scanner;
public class menu {
  public static void main(String[] args) {
  Scanner scanner = new Scanner(System.in);
  System.out.println("Menú de opciones");
  System.out.println("1. Sumar dos números (x+y)");
  System.out.println("2. Restar dos números (x-y)");
  System.out.println("3. Multiplicar dos números (x*y)");
  System.out.println("4. Dividir dos números (x/y)");
  System.out.println();
  System.out.println("Ingresar opción:");
  int opcion = scanner.nextInt();

  System.out.println("Ingresar el primer número");
  int x = scanner.nextInt();
  System.out.println("Ingresar el segundo número");
  int y = scanner.nextInt();
  int resultado = 0;
        switch (opcion) {
        case 1:
                resultado = x + y;
                break;
        case 2:
                resultado = x - y;
                break;
        case 3:
                resultado = x * y;
                break;
        case 4:
                resultado = x / y;
                break;
        default:
                System.out.println("opción incorrecta");
                break;
        }
  System.out.print("mostrar el resultado:");
  System.out.println(resultado);
  }}
```

```
public static void main(String[] args) {
//versión burda, muestra 11 como diez y uno.
int num;
int unidades, decenas;
num=Entrada.entero();
unidades = num % 10;
decenas = num / 10;
switch(decenas){
case 0:
System.out.print("");
break;
case 1:
System.out.print("diez");
break;
case 2:
System.out.print("veinte");
break;
case 3:
System.out.print("treinta");
break;
case 4:
System.out.print("cuarenta");
break;
case 5:
System.out.print("cincuenta");
break;
case 6:
System.out.print("sesenta");
break;
```

```
case 7:
System.out.print("setenta");
break;case 8:
System.out.print("ochenta");
break;
case 9:
System.out.print("noventa");
break;
}
System.out.print (" y ");
switch(unidades){
case 0:
System.out.println("");
break;
case 1:
System.out.println("uno");
break;
case 2:
System.out.println("dos");
break;
case 3:
System.out.println("tres");
break;
case 4:
 .  .   .
case 8:
System.out.println("ocho");
break;
case 9:
System.out.println("nueve");
break;
}}}
```

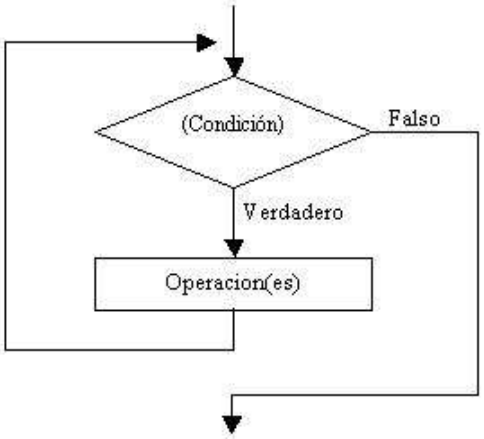
Estructura repetitivas: bucles *while y for*

Permite ejecutar una instrucción o un conjunto de instrucciones varias veces en bucle finito o infinito.

Estructura repetitiva while.

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.

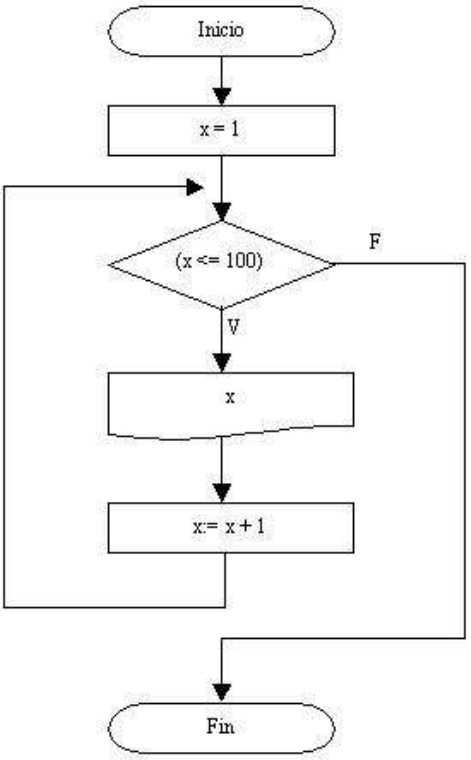


Ejercicio 11:

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

Inicializamos una variable con el valor 1, luego imprimimos la variable, incrementamos nuevamente la variable y así sucesivamente. Ver diagramas de flujo:

```
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;
        }
    }
}
```



La primera operación inicializa la variable x en 1, seguidamente comienza la estructura repetitiva while y disponemos la siguiente condición (x <= 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.

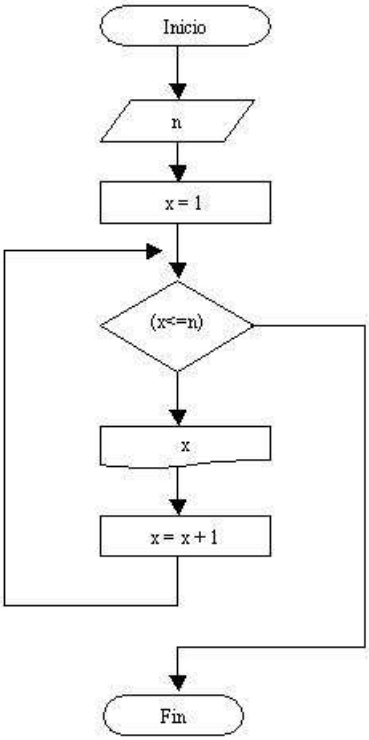
La variable x debe estar inicializada con algún valor antes que se ejecute la operación x=x + 1 en caso de no estar inicializada aparece un error de compilación.

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.

Ejercicio 12 while:

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

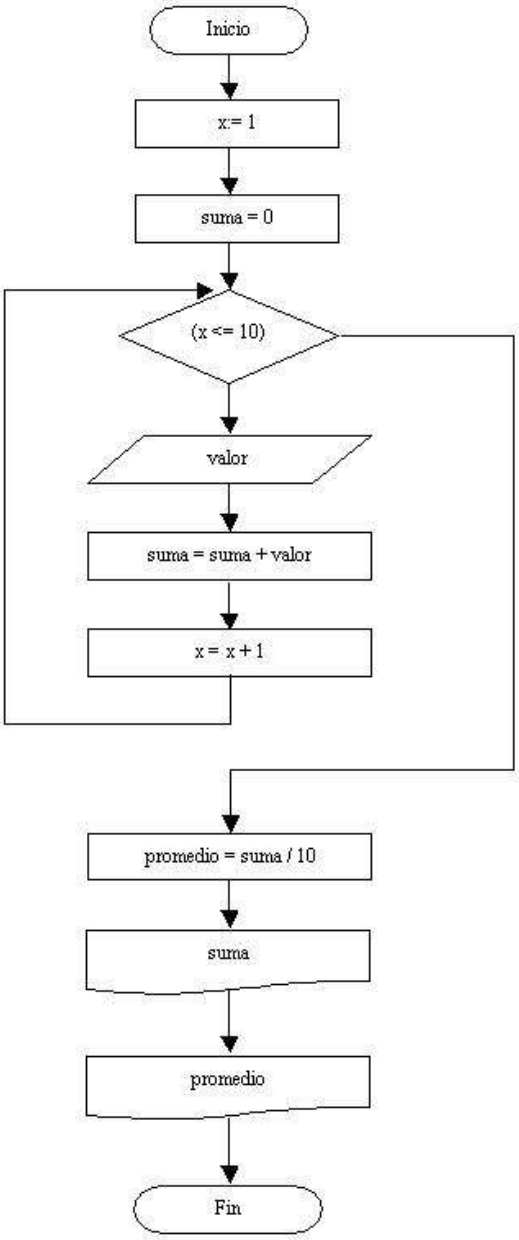


Podemos observar que se ingresa por teclado la variable n. El operador puede cargar cualquier valor.
Si el operador carga 10 el bloque repetitivo se ejecutará 10 veces, ya que la condición es ?Mientras x<=n ?, es decir ?mientras x sea menor o igual a 10?; pues x comienza en uno y se incrementa en uno cada vez que se ejecuta el bloque repetitivo. La variable x recibe el nombre de CONTADOR.

```
import java.util.Scanner;

public class EstructuraRepetitivaWhile2 {
    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;
        }
    }
}
```

Ejercicio 12 b:

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.

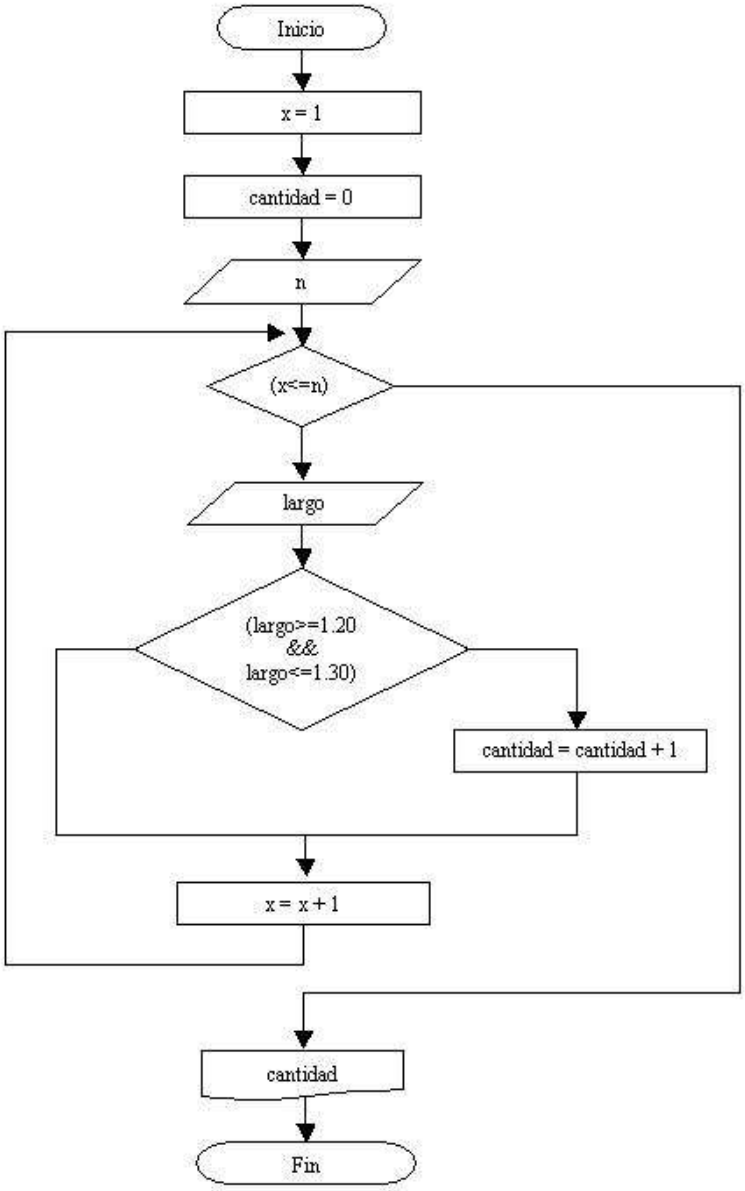


Llevamos un CONTADOR llamado x que nos sirve para contar las vueltas que debe repetir el while.
También aparece el concepto de ACUMULADOR (un acumulador es un tipo especial de variable que se incrementa o decrementa con valores variables durante la ejecución del programa)
Hemos dado el nombre de suma a nuestro acumulador.
Los números que toma la variable valor dependerá de qué cifras cargue el operador durante la ejecución del 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);
 }
}
```

Ejercicio 13 while:

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.



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

```
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 piezar 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 son:");
        System.out.print(cantidad);
    }
}
```

Problemas propuestos

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

1. AprobadoWhile: Escribir un programa que solicite 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 5 y cuántos menores.
2. AlturasWhile: Se pide un conjunto de n alturas de personas por teclado. Mostrar el promedio de alturas.
3. SueldosWhile: 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. ListasWhile: 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. ParesWhile: Desarrollar un programa que permita cargar n números enteros y luego nos informe cuántos valores fueron pares y cuántos impares.
   Nota: Emplear el operador ?%? en la condición de la estructura condicional:
   if (valor%2==0) es múltiplo de 2, luego es par.

```java
import java.util.Scanner;
public class AprobadoWhile {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int x,nota,conta1,conta2;
        x=1;
        conta1=0;
        conta2=0;
        while (x<=10) {
            System.out.print("Ingrese nota:");
            nota=teclado.nextInt();
            if (nota>=5) {
                conta1=conta1 + 1;
            }else {
                conta2=conta2 + 1;
            }
            x=x + 1;
        }
        System.out.print("Nº alumnos aprobados:");
        System.out.println(conta1);
        System.out.print("Nº alumnos suspendidos:");
        System.out.print(conta2);
    }
}
```

```java
import java.util.Scanner;
public class AlturasWhile {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int n,x;
        float altura,suma,promedio;
        System.out.print("Cuantas personas hay:");
        n=teclado.nextInt();
        x=1;
        suma=0;
        while (x<=n) {
            System.out.print("Ingrese la altura:");
            altura=teclado.nextFloat();
            suma=suma + altura;
            x=x + 1;
        }
        promedio=suma/n;
        System.out.print("Altura promedio:");
        System.out.print(promedio);
    }
}
```

```java
import java.util.Scanner;
public class SueldosWhile {
   public static void main(String[] ar) {
      Scanner teclado=new Scanner(System.in);
      int n,x,conta1,conta2;
      float sueldo,gastos;
      System.out.print("Cuantos empleados tiene la empresa:");
      n=teclado.nextInt();
      x=1;
      conta1=0;
      conta2=0;
      gastos=0;
      while (x<=n) {
         System.out.print("Ingrese el sueldo del empleado:");
         sueldo=teclado.nextFloat();
         if (sueldo<=300) {
            conta1=conta1 + 1;
         } else {
            conta2=conta2 + 1;
         }
         gastos=gastos+sueldo;
         x=x + 1;
      }
      System.out.print("Cantidad de empleados con sueldos entre 100 y 300:");
      System.out.println(conta1);
      System.out.print("Cantidad de empleados con sueldos mayor a 300:");
      System.out.println(conta2);
      System.out.print("Gastos total de la empresa en sueldos:");
      System.out.println(gastos);
   }
}
```

```java
public class EstructuraRepetitivaWhile8 {
   public static void main(String[] ar) {
      int x,termino;
      x=1;
      termino=11;
      while (x<=25) {
         System.out.print(termino);
         System.out.print(" - ");
         x=x + 1;
         termino=termino +11;
      }
   }
}
```

```java
public class EstructuraRepetitivaWhile9 {
   public static void main(String[] ar) {
      int mult8;
      mult8=8;
      while (mult8<=500) {
         System.out.print(mult8);
         System.out.print(" - ");
         mult8=mult8 + 8;
      }
   }
}
```

```java
import java.util.Scanner;
public class ListasWhile {
   public static void main(String[] ar) {
      Scanner teclado=new Scanner(System.in);
      int valor,x,suma1,suma2;
      x=1;
      suma1=0;
      suma2=0;
      System.out.println("Primer lista");
      while (x<=15) {
         System.out.print("Ingrese valor:");
         valor=teclado.nextInt();
         suma1=suma1 + valor;
         x=x + 1;
      }
      System.out.println("Segunda lista");
      x=1;
      while (x<=15) {
         System.out.print("Ingrese valor:");
         valor=teclado.nextInt();
         suma2=suma2 + valor;
         x=x + 1;
      }
      if (suma1>suma2) {
         System.out.print("Lista 1 mayor.");
      } else {
         if (suma2>suma1) {
            System.out.print("Lista2 mayor.");
         } else {
            System.out.print("Listas iguales.");
         }
      }
   }
}
```

```java
import java.util.Scanner;

public class ParesWhile {
   public static void main(String[] ar) {
      Scanner teclado=new Scanner(System.in);
      int n,x,valor,pares,impares;
      x=1;
      pares=0;
      impares=0;
      System.out.print("Cuantos números ingresará:");
      n=teclado.nextInt();
      while (x<=n) {
         System.out.print("Ingrese el valor:");
         valor=teclado.nextInt();
         if (valor%2==0) {
            pares=pares + 1;
         } else {
            impares=impares + 1;
         }
         x=x + 1;
      }
      System.out.print("Cantidad de pares:");
      System.out.println(pares);
      System.out.print("Cantidad de impares:");
      System.out.print(impares);
   }
```

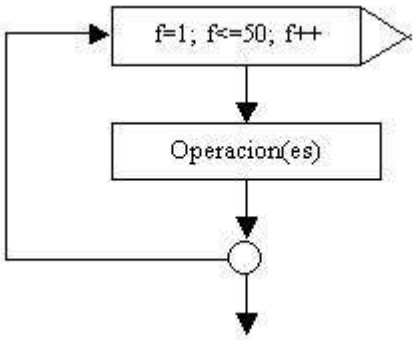
Juego piedra-papel-tijeras

```java
package hola;
import java.util.Scanner;
```

```
import java.util.Random;

public class hola {

   public static void main(String[] args)
   {
         Scanner teclado = new Scanner(System.in);
         System.out.println("Bienvenido al juego de piedra, papel y tijeras");
         System.out.print("Escoge una opción (1=piedra, 2=papel, 3=tijeras): ");
         int valor;
         valor=teclado.nextInt();
         if (valor==1) System.out.println("Has escogido piedra");
         else if (valor==2) System.out.println("Has escogido papel");
         else System.out.println("Has escogido tijeras");
         int valorc = (int)(Math.random()*3);
         if (valorc==1) System.out.println("Mi elección es piedra");
         else if (valorc==2) System.out.println("Mi elección es papel");
         else System.out.println("Mi elección es tijeras");
         System.out.print("Ganador: ");
         if (valor==valorc) System.out.print(" empatados");
         else if (valor==1) //si ha elegido piedra
         {
               if (valorc==2)System.out.print(" yo");
               else System.out.print(" tú");
         }
         else if (valor==2) //si ha elegido papel
         {
               if (valorc==1)System.out.print(" tú");
               else System.out.print(" yo");
         }
         else if (valor==3) //si ha elegido tijeras
         {
               if (valorc==1)System.out.print(" yo");
               else System.out.print(" tú");
         }
   }
}
```

Estructura repetitiva for

En general, la estructura *for* se usa cuando sabemos el número de veces que queremos que se ejecute el bloque de instrucciones. (**Bucle finito**)



`for( contador ; condición ; increm.)`

Necesita tres datos:

- El *primero* es una variable que hará de ***contador*** de vueltas, asignándole a dicha variable un valor inicial.
- En el *segundo* se pone la ***condición*** que deberá ser verdadera para que el ciclo continúe.
- El *tercero*, es la sección para ***incrementar*** el contador.

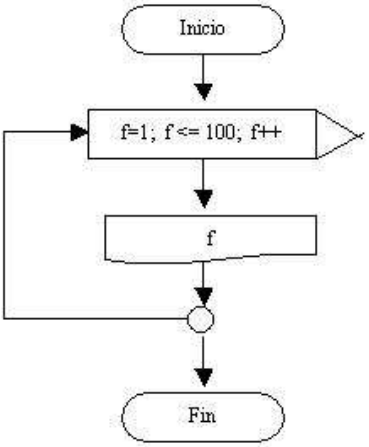
Cuando el ciclo comienza, se verifica si la condición es verdadera. En caso de serlo se ejecuta el bloque interno del ciclo, se incrementa el contador y se vuelve a comprobra la condición para repetir el bloque. Así sucesivamente.

Ejercicio 14:

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

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.

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

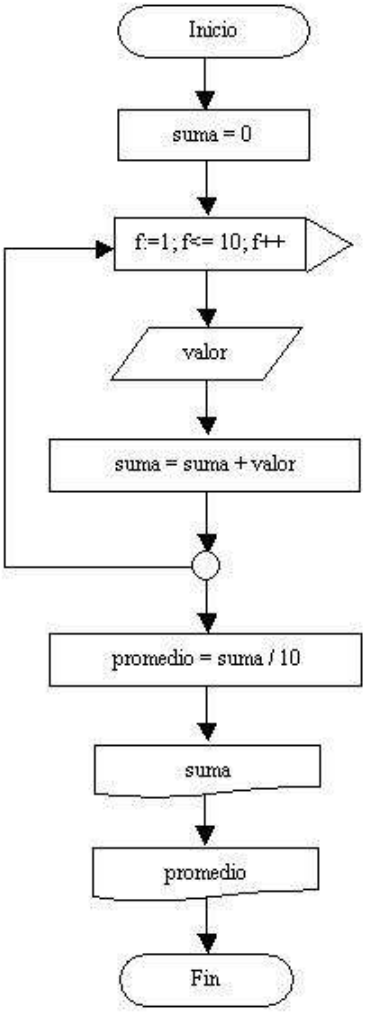


Ejercicio 15:

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 desarrollamo , lo resolveremos empleando la estructura for.

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

```
repita el bloque de instrucciones 10 veces.
import java.util.Scanner;

public class EstructuraRepetitivaFor2 {
    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.

Ejercicio 16:

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

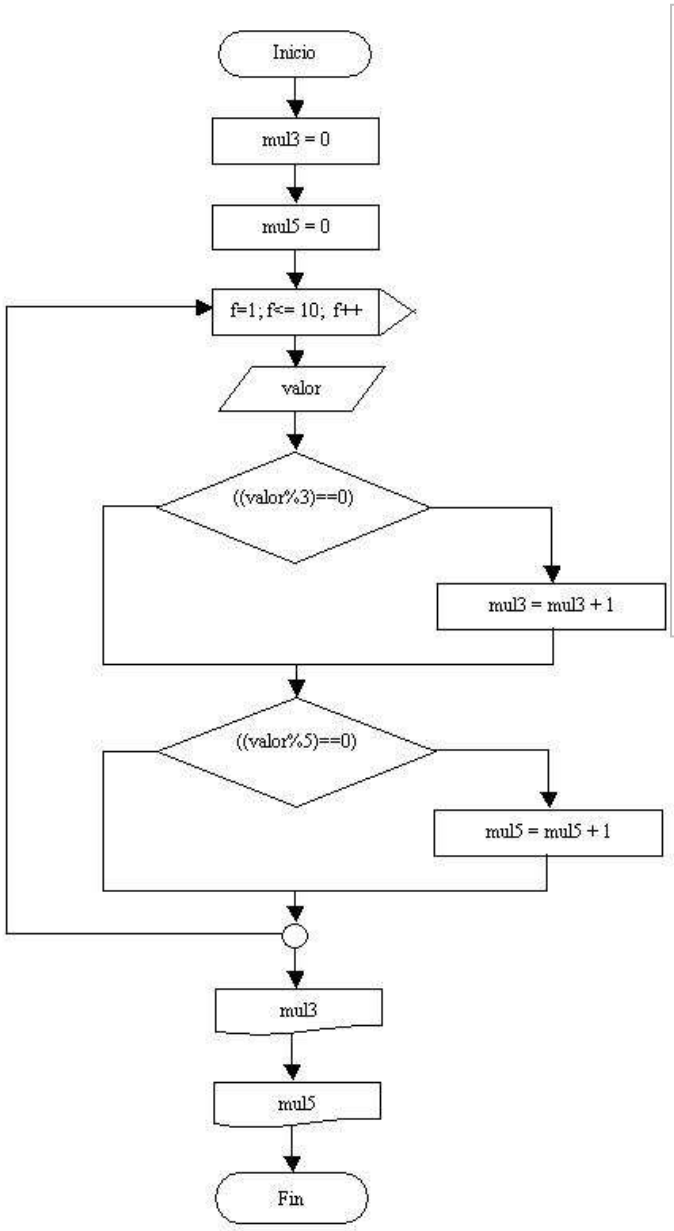
Para resolver este problema se requieren tres contadores:
aprobados (Cuenta la cantidad de alumnos aprobados)
suspendidos (Cuenta la cantidad de suspendidos)
f (es el contador del for)
Dentro de la estructura repetitiva debemos hacer la carga de la variable nota y verificar con una estructura condicional 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 suspendidos.

Los contadores aprobados y suspendidos deben iniciarse e imprimirse FUERA de la estructura repetitiva.
Es fundamental inicializar los contadores aprobados y suspendidos en cero antes de entrar a la estructura for.

```
import java.util.Scanner;

public class EstructuraRepetitivaFor3 {
    public static void main(String[] ar) {
        Scanner teclado=new Scanner(System.in);
        int aprobados, suspendidos,f,nota;
        aprobados=0;
        suspendidos =0;
        for(f=1;f<=10;f++) {
            System.out.print("Ingrese la nota:");
            nota=teclado.nextInt();
            if (nota>=7) {
                aprobados=aprobados+1;
            } else {
               suspendidos =suspendidos+1;
            }
        }
        System.out.print("Cantidad de aprobados:");
        System.out.println(aprobados);
        System.out.print("Cantidad de suspendidos:");
        System.out.print(suspendidos);
    }
}
```

Ejercicio 17:

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.




```
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 múltiplos de 3:");
  System.out.println(mul3);
  System.out.print("Cantidad múltiplos de 5:");
  System.out.print(mul5);
  }
}
```

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

Ejercicio 18:
Escribir un programa que lea n números enteros y calcule la cantidad de valores mayores o iguales a 1000.

```
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
         mayores o iguales a 1000 son:");
        System.out.print(cantidad);
    }
}
```

Problemas propuestos

Estos son ejercicios 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 cuál de los tres turnos tiene un promedio de edades mayor.

```
import java.util.Scanner;

public class EstructuraRepetitivaFor6 {
   public static void main(String[] ar) {
      Scanner teclado=new Scanner(System.in);
      int base,altura,superficie,cantidad,f,n;
      cantidad=0;
      System.out.print("Cuantos triángulos procesará:");
      n=teclado.nextInt();
      for(f=1;f<=n;f++) {
         System.out.print("Ingrese el valor de la base:");
         base=teclado.nextInt();
         System.out.print("Ingrese el valor de la altura:");
         altura=teclado.nextInt();
         superficie=base*altura/2;
         System.out.print("La superficie es:");
         System.out.println(superficie);
         if (superficie>12) {
            cantidad=cantidad+1;
         }
      }
      System.out.print("La cantidad de triángulos con
      superficie superior a 12 son:");
      System.out.print(cantidad);
   }
}
```

```
import java.util.Scanner;

public class EstructuraRepetitivaFor7 {
   public static void main(String[] ar) {
      Scanner teclado=new Scanner(System.in);
      int f,valor,suma;
      suma=0;
      for(f=1;f<=10;f++) {
         System.out.print("Ingrese un valor:");
         valor=teclado.nextInt();
         if (f>5) {
            suma=suma+valor;
         }
      }
      System.out.print("La suma de los últimos 5 valores
      es:");
      System.out.print(suma);
   }
}
```

```
public class EstructuraRepetitivaFor8 {
      public static void main(String[] ar) {
         int f;
         for(f=5;f<=50;f=f+5) {
             System.out.print(f);
             System.out.print("-");
         }
   }
}
```

```
import java.util.Scanner;

public class EstructuraRepetitivaFor8 {
   public static void main(String[] ar) {
      Scanner teclado=new Scanner(System.in);
      int f,valor;
      System.out.print("Ingrese un valor entre 1 y 10:");
      valor=teclado.nextInt();
      for(f=valor;f<=valor*12;f=f+valor) {
         System.out.print(f);
         System.out.print("-");
      }
   }
}
```

```
import java.util.Scanner;

public class EstructuraRepetitivaFor10 {
   public static void main(String[] ar) {
      Scanner teclado=new Scanner(System.in);
      int n,f,x,y,cant1,cant2,cant3,cant4;
      cant1=0;
      cant2=0;
      cant3=0;
      cant4=0;
      System.out.print("Cantidad de puntos:");
      n=teclado.nextInt();
      for(f=1;f<=n;f++) {
         System.out.print("Ingrese coordenada x:");
         x=teclado.nextInt();
         System.out.print("Ingrese coordenada y:");
         y=teclado.nextInt();
         if (x>0 && y>0) {
            cant1++;
         } else {
            if (x<0 && y>0) {
               cant2++;
            } else {
               if (x<0 && y<0) {
                  cant3++;
               } else {
                  if (x>0 && y<0) {
                        cant4++;
                  }
               }
            }
         }
      }
```

```
import java.util.Scanner;

public class EstructuraRepetitivaFor9 {
   public static void main(String[] ar) {
      Scanner teclado=new Scanner(System.in);
      int f,lado1,lado2,lado3,cant1,cant2,cant3,n;
      cant1=0;
      cant2=0;
      cant3=0;
      System.out.print("Ingrese la cantidad de triángulos:");
      n=teclado.nextInt();
      for(f=1;f<=n;f++) {
        System.out.print("Ingrese lado 1:");
        lado1=teclado.nextInt();
        System.out.print("Ingrese lado 2:");
        lado2=teclado.nextInt();
        System.out.print("Ingrese lado 3:");
        lado3=teclado.nextInt();
        if (lado1==lado2 && lado1==lado3) {
           System.out.println("Es un triángulo equilatero.");
           cant1++;
        } else {
           if (lado1==lado2 || lado1==lado3 || lado2==lado3) {
              System.out.println("Es un triángulo isósceles.");
              cant2++;
           }  else {
              cant3++;
              System.out.println("Es un triángulo escaleno.");
           }
        }
      }
      System.out.print("Cantidad de triángulos equilateros:");
      System.out.println(cant1);
      System.out.print("Cantidad de triángulos isósceles:");
      System.out.println(cant2);
      System.out.print("Cantidad de triángulos escalenos:");
      System.out.println(cant3);
      if (cant1<cant2 && cant1<cant3) {
         System.out.print("Hay menor cantidad de triángulos
      equilateros.");
      } else {
         if (cant2<cant3) {
            System.out.print("Han menor cantidad de triángulos
      isósceles");
         } else {
            System.out.print("Han menor cantidad de triángulos
      escalenos");
         }
      }
   }
}  System.out.print("Cantidad de puntos en el primer
      cuadrante:");
      System.out.println(cant1);
      System.out.print("Cantidad de puntos en el segundo
      cuadrante:");
      System.out.println(cant2);
      System.out.print("Cantidad de puntos en el tercer
      cuadrante:");
      System.out.println(cant3);
      System.out.print("Cantidad de puntos en el cuarto
      cuadrante:");
      System.out.println(cant4);
```

```
import java.util.Scanner;

public class EstructuraRepetitivaFor11 {
   public static void main(String[] ar) {
      Scanner teclado=new Scanner(System.in);
      int f,valor,negativos,positivos,mult15,sumapares;
      negativos=0;
      positivos=0;
      mult15=0;
      sumapares=0;
      for(f=1;f<=10;f++) {
         System.out.print("Ingrese valor:");
         valor=teclado.nextInt();
         if (valor<0) {
            negativos++;
         } else {
             if (valor>0) {
                        positivos++;
             }
         }
         if (valor%15==0) {
             mult15++;
         }
         if (valor%2==0) {
             sumapares=sumapares+valor;
         }
      }
      System.out.print("Cantidad de valores negativos:");
      System.out.println(negativos);
      System.out.print("Cantidad de valores positivos:");
      System.out.println(positivos);
      System.out.print("Cantidad de valores múltiplos de
       15:");
      System.out.println(mult15);
      System.out.print("Suma de los valores pares:");
      System.out.println(sumapares);
   }
}
```

```
import java.util.Scanner;

public class EstructuraRepetitivaFor12 {
   public static void main(String[] ar) {
      Scanner teclado=new Scanner(System.in);
      int f,edad,suma1,suma2,suma3,pro1,pro2,pro3;
      suma1=0;
      suma2=0;
      suma3=0;
      for(f=1;f<=50;f++) {
         System.out.print("Ingrese edad:");
         edad=teclado.nextInt();
         suma1=suma1+edad;
      }
      pro1=suma1/50;
      System.out.print("Promedio de edades del turno mañana:");
      System.out.println(pro1);
      for(f=1;f<=60;f++) {
         System.out.print("Ingrese edad:");
         edad=teclado.nextInt();
         suma2=suma2+edad;
      }
      pro2=suma2/60;
      System.out.print("Promedio de edades del turno tarde:");
      System.out.println(pro2);
      for(f=1;f<=110;f++) {
         System.out.print("Ingrese edad:");
         edad=teclado.nextInt();
         suma3=suma3+edad;
      }
      pro3=suma3/110;
      System.out.print("Promedio de edades del turno noche:");
      System.out.println(pro3);
      if (pro1<pro2 && pro1<pro3) {
         System.out.print("El turno mañana tiene un promedio menor
       de edades.");
      } else {
         if (pro2<pro3) {
            System.out.print("El turno tarde tiene un promedio menor
       de edades.");
         } else {
            System.out.print("El turno noche tiene un promedio menor
       de edades.");
         }
      }
   }
}
```

Cadenas de texto

- Para una cadena de caracteres, definir un objeto de la clase *String*.
- Para pedir una cadena: `nombre1=teclado.next();`
- Para pedir una cadena con caracteres en blanco: `nombre1=teclado.nextLine();`
- Para comparar dos String no podemos utilizar el operador == Debemos utilizar el método **equals** y pasar como parámetro el String con el que queremos compararlo: `if (apellido1.equals(apellido2))`
- Para comparar ignorando las mayúsculas utilizar equalsIgnoreCase: `apellido1.equalsIgnoreCase(apellido2)`
- parámetro el String con el que queremos compararlo: `if (apellido1.equals(apellido2))`
- Para comparar ignorando las mayúsculas utilizar equalsIgnoreCase: `apellido1.equalsIgnoreCase(apellido2)`
- Otros: **tochararray** (de cadena a array) – **Split** (separar) – **substring** (subcadena)

Ejemplos:

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

      String nom1="Marta",nom2="Ana";
      //equals, equalsIgnoreCase
      if (nom1.equals(nom2)==true){
          System.out.println("Iguasl");
      }
      if (nom1.equalsIgnoreCase(nom2)==true){
          System.out.println("Iguals");
      }
      //toCharArray
      char[] llista=nom1.toCharArray();
      for (int i = 0; i < llista.length; i++) {
          System.out.println(llista[i]);
      }
      //split
      String cadena2="dilluns,dimarts,dimecres";
      String[] cadena=cadena2.split(",");
      for (int i = 0; i < cadena.length; i++) {
          System.out.println(cadena[i]);
      }
      // compareTo
      if (nom1.compareTo(nom2)==0){
        System.out.println("Iguals");
      }
      if (nom1.compareTo(nom2)>0){
          System.out.println("nom1 > nom2");
      }
      if (nom1.compareTo(nom2)<0){
          System.out.println("nom1 < nom2");
      }
      //substring
      String cadena3=cadena2.substring(8,15);
      System.out.println("cadena3="+cadena3);
  }
```

```java
//** Ordenar 5 paises alfabéticamente **

import java.util.Scanner;
public class PruebaPaises {
  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("Nombre del pais:");
         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
alfabéticamente:");
     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();
    }
}
```

Arrays tablas, vectores o matrices

Un array guarda múltiples valores en una especie de rejilla o tabla. Es como una hoja de cálculo, en donde cada celda tiene un número de posición. Las posiciones en un array empiezan en 0 y ascienden secuencialmente.
Puedes ser *unidimensionales*, de una sola fila y varias columnas o *multidimensionales*, con varias filas y columnas.
Los arrays hay que declararlos y luego definir sus dimensiones y tipo.

Array unidimensional: Tiene una sola fila o columna

Sintaxis de una array unidimensional o vector:
Declaración del array del tipo integer: `int[] matriz;`
Definición del tamaño del array: `matriz = new int[6];`
Retorno del tamaño o ancho del array: `matriz.length;`
Vectores paralelos: Si tiene el mismo tamaño

```
public class arrays {
   public static void main(String[] args) {
   int[] matriz = { 1, 2, 3, 4, 5, 6 };

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

Podemos declarar y crear la matriz al mismo tiempo: `int[] matriz=new int[6]`
Podemos crear una matriz de tipo objetos:

```
Cliente[] arr = new Cliente[7]; //Cliente es una clase definida por el usuario
```

Atrays bidimensionales: Si tiene filas y columas

Declaración: `int matriz[][];`
`matriz = new int[2][2];`
Acceso: `int x = matriz[1][1];` // Para leer el contenido de un elemento
`matriz[1][1] = x;` // Para asignar un valor.
Nº columnas: matriz.length;
Nº filas de la col 1: matrix[1].lenght;

```
int matriz[][];
matriz = new int[4][4];
for (int x=0; x < matrix.length; x++) {
  for (int y=0; y < matriz[x].length; y++) {
      System.out.println (matriz[x][y]);
  }
}
```

Ejercicio: Arrays bidimensionales.

Ejercicio que introduzca en un array de 5 filas por 5 columnas la suma de sus filas y columnas y muestre el resultado:

```
public class Main {
    public static void main(String[] args) {
        int cf=0;
        int suma[][]= new int[5][5];                    //crea el array suma
        for (int c = 0; c < suma.length; c++) {
            System.out.println("\t");
            for (int f = 0; f < suma.length; f++) {
                cf=c+f;
                suma[c][f]=cf;
                System.out.print(c+","+f+" = "+cf+" ");
            }
        }  }  }
```

Ejercicio *que introduzca en un array de 5 filas por 4 columnas un valor aleatorio entre 1 y 100 únicamente en sus columnas impares.*

```
import java.util.Random;
    public static void main(String[] args) {
        Random aleat = new Random();                    //creamos un objeto aleatorio
         int t[][]= new int[4][5];                      //creamos array de 4x5
         for (int c = 0; c < t.length; c++) {
             if (c%2!=0){                               //si no es múltiplo de 2
                for (int f = 0; f < t.length; f++) {
                   t[c][f]=(aleat.nextInt()*100+1);
                   System.out.print(t[c][f] +"   ");
             }
        }
    }
}
```

Ejemplo tres en raya

Rellenamos una tabla de 3x3 de valores aleatorios entre 0 y 1 y mostramos su resultado ordenado. Luego evalúa e informa si los "0" o los "1" están en raya.

```
int tabla[][];
tabla = new int[3][3];
for (int i=0; i < tabla.length; i++) {
  for (int j=0; j < tabla[i].length; j++) {
    tabla[i][j] = (int) (Math.random()*2);
    System.out.print(tabla[i][j]+"  ");
    if (j==2){System.out.println(""); }
   }
}
   /* comprobar columnas */
   boolean enraya=false;
   for (int i = 0; i < 3; i++)
   if (tabla[i][0] == tabla[i][1] && tabla[i][1] == tabla[i][2])
   enraya=true;

   /* comprobar filas */
   for (int j = 0; j < 3; j++)
   if (tabla[0][j] == tabla[1][j] && tabla[1][j] == tabla[2][j])
   enraya=true;

   /* comprobar diagonal principal */
   if (tabla[0][0] == tabla[1][1] && tabla[1][1] == tabla[2][2])
   enraya=true;

   /* comprobar diagonal inversa */
   if (tabla[0][2] == tabla[1][1] && tabla[1][1] == tabla[2][0])
   enraya=true;
   // muetra si estan en linea
   if (enraya==true){System.out.println("En raya");}
   else {System.out.println("No están en raya");}
                }
```

Ejercicio: Boleto de lotería.

Comprueba los aciertos que tiene el boleto introducido por el jugador con una combinación generada aleatoriamente por el ordenador.

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

public class Loteria {

    public static void main(String[] args) {
    Scanner lector = new Scanner(System.in);
    Random random = new Random();

    int[] numero = {0,0,0,0,0,0};
    int[] sorteig = new int[6];
    int encert = 0 ;


//Demanem els numeros al jugador
 System.out.println(" \n ");
 for (int c = 0; c < 6; c++) {
  System.out.println(" Escriu un numero del 1 al
    49: ");
        sorteig[c] = lector.nextInt();
       if(sorteig[c]>49){
      System.out.println(" Error , només numeros
    del 1 al 49");
     c--;
         }
    }

//Es mostren els números introduits per teclat
  System.out.println(" \n ");
   System.out.print("La teva combinació es: ");
        for (int d = 0; d < 6; d++) {
             System.out.print(sorteig[d]+"  ");
        }

//Generació de la combinació
   for (int i = 0; i < 6; i++) {
       numero[i]=random.nextInt(49)+1;
      //En cas de que estiguin repetits
       for (int a = 0; a < 6; a++) {
         if(numero[a]==numero[i]){
//Comparació
            numero[i]=random.nextInt(49)+1;
               }
             }
        }

//Mostra els números aleatoris
   System.out.println(" ");
   System.out.println(" \n ");
   System.out.print("La combinació ganadora es: ");
    for (int b = 0; b < numero.length; b++) {
        System.out.print(numero[b]+"  ");
    }


  //fem la compraració
       for (int e = 0; e < 6; e++) {
         if(sorteig[e]==numero[e]){
            encert++;
          }
        }

  //Mostrem els encerts
     System.out.println(" ");
     System.out.println(" \n ");
     System.out.println("Has encertat "+encert+"
    vegades ");
     }
}
```

Ejercicio: Agenda de personas.

Crear app con un menú pasa escoger altas, bajas y listados de personas en un array variable.

```
    public static void main(String[] args) {
        Scanner lector = new Scanner(System.in);
        int op=0,llarg=3,posencontrado=0;
        String llista[];                  //declaramos array del tipo String
        String busca;
        boolean encontrado=false;
        llista= new String [llarg];       //creamos el array
        while (op!=4) {
         System.out.println("Escull una opció.");
         System.out.println("1 - Donar altes persones.");
         System.out.println("2 - Baixes de persones.");
         System.out.println("3 - Llistats de persones.");
         System.out.println("4 - Sortir.");
         op = lector.nextInt();
         switch (op) {
             case 1: //Altas
                 System.out.println("ALTAS DE PERSONAS");
                 for (int i = 0; i < llista.length; i++) {
                 System.out.println("Introdueix el nom "+i+" de " + llarg + ": ");
                 llista[i]=lector.next();
                 }
                 break;
             case 2: //Bajas
                 encontrado=false;
                 System.out.println("BAJAS DE PERSONAS");
                 System.out.println("Introdueix el nom de la persona a donar de baixa:");
                 busca=lector.next();
                 for (int i = 0; i < llista.length; i++) {
                     if (llista[i].compareTo(busca)==0){
                     posencontrado=i;
                     encontrado=true;
                     }
                 }
                 if (encontrado=true)
                 {
                 for (int j = posencontrado; j < llista.length-1; j++) {
                 llista[j]=llista[j+1];}
                 llista[llista.length]="";
                 }
                 else
                 {
                     System.out.println("Persona no encontrada");
                 }

                 System.out.println("*******************\n");
                 break;
             case 3: //Llistats
                 System.out.println("LLISTAT DE PERSONAS");
                 for (int i = 0; i < llista.length; i++) {
                 System.out.println(llista[i]);}
                 break;
             case 4: System.out.println("Fi del programa.");//Sortida.
                 break;
             default: System.out.println("Opció incorrecta. Torna a provar.");
        }
        }
    }
```

```
// ** El sueldo mayor de 5 empleados **
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("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();
    }
}
```

Funciones

Una función es aquella acción que recoge un argumento y retorna un valor

Parámetros y argumentos

El término parámetro, se usa a menudo para referirse a la variable en la declaración del método, mientras que argumento, se refiere al valor que se envia. **Ejemplo:**

```
private static void insertarDatosArray(int[] array, Scanner scanner) {
     for (int i = 0; i < array.length; i++) {
     System.out.print("insertar array[" + i + "]:");
     array[i] = scanner.nextInt();
     }
     }
```

array y scanner son los argumentos y en la declaración del método, array y scanner son los parámetros

Métodos con retorno

Un método vuelve al código del que se llamó en el momento en el que alguna de estas circunstancias se de:
- se completan todas las sentencias del método, - llega a una sentencia retorno o - lanza una excepción,
Ejemplo; return valorRetorno;

Ejercicio de funciones:

```
import java.util.Scanner;

public class Funciones2 {
   public static void main(String[] args) {
   int[] array = new int[5];
   Scanner scanner = new Scanner(System.in);

   insertarDatosArray(array, scanner);
   sumarArray(array);
   int max = maxArray(array);
   System.out.println("Max= " + max);

   double promedio = promedioArray(array);
   System.out.println("Promedio= " + promedio);
   }

   private static void insertarDatosArray(int[] array,
   Scanner scanner) {
   for (int i = 0; i < array.length; i++) {
   System.out.print("insertar array[" + i + "]:");
   array[i] = scanner.nextInt();
   }
   }
   private static void sumarArray(int[] array) {
   System.out.print("Suma: ");
   int acumulador = 0;
   for (int i = 0; i < array.length; i++) {
   acumulador = acumulador + array[i];
   System.out.print("+" + array[i]);
   }
   System.out.println("= " + acumulador);
   }

   private static int maxArray(int[] array) {
        int max = 0;
        for (int i = 0; i < array.length; i++) {
                if (array[i] > max) {
                        max = array[i];
   }
   }
   return max;
   }

   private static double promedioArray(int[] array) {
        double promedio = 0;
        for (int i = 0; i < array.length; i++) {
        promedio = promedio + array[i];
        }
        promedio = promedio / array.length;
        return promedio;
   }
    }    }}
```

```
public class Funciones1 {

  public static void main(String[] args) {

      int resultado=0;

      //llamamos la funcion suma con dos numeros
      resultado = suma(5,5);

      System.out.println("El resultado de la suma fue: " +
resultado);

      sumaSinRetorno(20,30);
  }

  /**
   * Funcion que suma dos numeros
   * Toma dos valores o parametros
   * Recordar poner public static
   * int se pone porque devuelve un valor entero
   * **/
  public static int suma(int numero1, int numero2){

   int resultado = 0;

   //Sumamos los dos numeros
   resultado = numero1 + numero2;

   //Retornamos el valor
   return resultado;
  }

  //funcion que no retorna nigun dato
  //La manera mas facil de saber si retorna o no es por la palabra
return

  public static void sumaSinRetorno(int numero1, int numero2){

   int resultado = numero1 + numero2;

   System.out.println("El resultado de la suma fue: " + resultado);

  }
}
```

Creación de objetos y clases

Creación de la clase

Sintaxis:

public class Nombre_clase o

[public] [final | abstract] class Clase [extends ClaseMadre] [implements Interfase1 [, Interfase2]…]

public: significa que puede ser usada por cualquier clase en cualquier paquete
abstract: puede tener herederas, pero no puede ser instanciada.
final: no puede tener clases que la hereden, normalmente por razones de seguridad.
extends: indica de qué clase desciende la nuestra
Interface: declara sus métodos, pero no los implementa
Atributos: [private |protected|public] [static] [final] [transient] [volatile] Tipo NombreVariable [= Valor];
Private, ***protected*** o ***public***: tipos de acceso diferente a las variables o métodos
Final: no pueda ser sobreescrito o redefinido. O sea, como una constante.
Volatile asegura que se vuelva a leer la variable (por si fue modificada) cada vez que se la va a usar
Los métodos internos de la clase pueden ser **Setter** (definidores) o **Getter** (Captadores)

Definición de la clase Contador (Implementación de un contador sencillo)

```
                                    //GRABAR EN UN ARCHIVO "Contador.java" (OJO CON LAS MAYUSCULAS!)
                                    // COMPILAR CON: "javac Contador.java" (NO OLVIDAR EL .java!)
                                    // ESTA CLASE NO ES UNA APLICACION, pero nos va a servir enseguida
public class Contador {                                 // Se define la clase Contador
    // Atributos
    int cnt;                                            // Un entero para guardar el valor actual
    // Constructor                                      // Un método constructor…
    public Contador() {                                 // …lleva el mismo nombre que la clase
    cnt = 0;                                            // Simplemente, inicializa (1)
    }
    // Métodos
    public int incCuenta() {                            // Un método para incrementar el contador
    cnt++;                                              // incrementa cnt
    return cnt;                                         // y de paso devuelve el nuevo valor
    }
    public int getCuenta() {                            // Este sólo devuelve el valor actual
    return cnt;                                         // del contador
    }
}
```

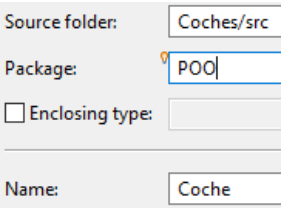
Creación del objeto

```
// Usemos nuestra clase contador en una mini-aplicación
    // GRABAR EN UN ARCHIVO "Ejemplo1.java" (OJO CON LAS MAYUSCULAS!)
import java.io.*;                               // Uso la biblioteca de entradas/salidas
public class Ejemplo1 {                         // Nombre de la clase  igual al nombre del archivo!
static int n;       // entero para asignarle el valor del contador e imprimirlo// aunque en realidad no me hace falta.

static Contador laCuenta;                       // y una variable tipo Contador para instanciar el objeto…

public static void main ( String args[] ) {     // metodo main, para que se comporte como una aplicacion
    System.out.println ("Cuenta… ");            // Imprimo el título
    laCuenta = new Contador();                  // Creo una instancia del Contador
    System.out.println (laCuenta.getCuenta());  // 0 - Imprimo el valor actual (cero)
    n = laCuenta.incCuenta();                   // 1 - Asignación e incremento
    System.out.println (n);                     // Ahora imprimo n
    laCuenta.incCuenta();                       // 2 - Lo incremento (no uso el valor de retorno)
    System.out.println (laCuenta.getCuenta());  // y lo imprimo
    System.out.println (laCuenta.incCuenta());  // 3 - Ahora todo en un paso!
 }
}
```

Ejercicio de creación de una clase

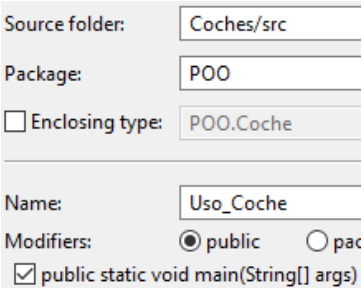
(Extracto Curso Java Youtube)

1º parte: Crera un Nuevo proyecto Java en Eclipse llamado **Coches** y una Clase **Coche** con el Pakage POO como en la imagen y el siguiente código:




```
package POO;
public class Coche {
        //atributos de la clase (propiedades)
        int ruedas;
        int largo;
        int ancho;
        int motor;
        int peso;
        //Método constructor, tiene el mismo nombre que la clase
        //inicializa los valores por defecto (valor de las propiedades por defecto)
        public Coche(){
                ruedas=4;
                largo=2000;  //en cm
                ancho=300;   //en cm
                motor=1600;  //en cc
                peso=700;       //en Kg
}       }
```

2º parte: Añadimos una segunda clase, para utilizar la primera, como en la imagen y el siguiente código:




```
package POO;
public class Uso_Coche {
        public static void main(String[] args) {
        Coche Renault = new Coche(); //creamos un objeto como instancia de la clase Coche
        System.out.println("Este coche tiene "+ Renault.ruedas+" ruedas");
}       }
```

Comprobamos y guardamos.

Ampliación de código: Modificamos con las siguientes mejoras.

```
public class Coche {
        //atributos comunes de la clase para todos los objetos
        private int ruedas;  //se encapsulan o cierran como privadas
        private int largo;   //para que no se puedan cambiar desde el exterior
        private int ancho;   //sólo se podrán pasar con los métodos getter o setter
        private int peso_base;
        int peso_total;
        String color;
        private String marca;
        boolean asientos_cuero, climatizador;
        //Método constructor, tiene el mismo nombre que la clase
        //inicializa los valores por defecto
        public Coche(){
                ruedas=4;
                largo=2000;  //en cm
                ancho=300;   //en cm
                motor=1600;  //en cc
                peso_base=700;  //en Kg
        }
        public String dime_largo() {  //método getter
                return "el largo del coche es" + largo;
        }
        public void establece_color() {//método setter
                color="azul";
        }
        public String dime_color(){ //método getter
                return "el color del coche es " + color;
        }
        public String marca(){ //método getter
                marca="Seat";
                return marca;
}       }

public class Uso_Coche {

        public static void main(String[] args) {
        Coche micoche = new Coche(); //creamos un objeto como instancia de la clase Coche
        System.out.println("mi coche es de la marca " + micoche.marca());
        micoche.establece_color();
        System.out.println(micoche.dime_color());
}     }
```

Ampliación y paso de parámetros: Para poder elegir el color o la marca

Cambia la función en la clase principal establece_color:

```
package POO;
public class Coche {
   //atributos comunes de la clase para todos los objetos
   private int ruedas; //se encapsulan o cierran como privadas
   private int largo;
   private int ancho;
   private int motor;
   private int peso_base;
   private int peso_total;
   private String color;
   String marca;  //no aconsejado sin private
   boolean asientos_cuero, climatizador;
   //Método constructor, tiene el mismo nombre que la clase
   //inicializa los valores por defecto
   public Coche(){
       ruedas=4;
       largo=2000;  //en cm
       ancho=300;   //en cm
       motor=1600;  //en cc
       peso_base=700; //en Kg
   }
   public String dime_datos_generales() {  //método getter
       return "vehículo de " + ruedas + " ruedas. Longitud " + largo/1000 + " metros. Ancho " +
       ancho + " cm y peso "+peso_base+" Kg";
   }
   public void establece_color(String color_coche) { //método setter color
       color=color_coche;
   }
   public String dime_color(){ //método getter color
       return "el color del coche es " + color;
   }
   public String dime_marca(){ //método getter y setter a la vez. No muy recomendado
       marca="Seat";
       return marca;
   }
   public void configura_asientos(String asientos_cuero) { //método setter asientos
       if (asientos_cuero=="si") {//mejor asientos_cuero.equals("si");
       this.asientos_cuero=true;  //this: para no confundir con el parámetro string
       }else {this.asientos_cuero=false;
       }
   }
   public String dime_asientos() { //método getter asientos
       if (asientos_cuero==true) {
          return "tiene asientos de cuero";
       }else {
          return "tiene asientos de serie";
       }
   }
   public void configura_climatizador(String climatizador) { //setter establece clima
       if (climatizador=="si") {//mejor climatizador.equals("si");
          this.climatizador=true;
       }else {
          this.climatizador=false;
       }
   }

   public String dime_climatizador() {//getter devuelve clima
       if (climatizador==true) {
          return "Vehiculo con climatizador";
          }else {
          return "Vehiculo con AC";
          }
   }
   public String dime_peso_coche() {//método getter y setter a la vez. No muy recomendado
       int peso_carroceria=500;
       peso_total=peso_base+peso_carroceria;  //aquí establece valor
       if(asientos_cuero==true) {
          peso_total=peso_total+500;
       }
       if(climatizador==true) {
          peso_total=peso_total+20;
```

```
        }
        return "Peso total del coche: "+peso_total; //aquí devuelve valor
    }

    public int precio_coche() {//método getter
        int precio_final=10000;
        if (asientos_cuero==true) {
           precio_final+=2000;
        }
        if (climatizador==true) {
           precio_final+=1500;
        }
        return precio_final;
    }
}
```

2º parte

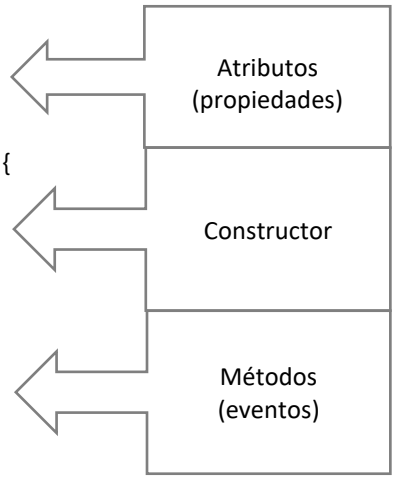
```
package POO;

public class Uso_Coche {

    public static void main(String[] args) {
    Coche micoche = new Coche(); //creamos un objeto como instancia de la clase Coche
    micoche.marca="Renault";    //definición no aconsejada
    System.out.println("mi coche es de la marca " + micoche.marca);  //--> dice Renault
    System.out.println("mi coche es de la marca " + micoche.dime_marca()); //--> dice Seat
    micoche.establece_color("rojo");  //definición aconsejada
    System.out.println(micoche.dime_color());
    System.out.println(micoche.dime_datos_generales());
    micoche.configura_asientos("si"); //o pedir con JoptionPane.showInputDialog("Pregunta: ")
    System.out.println(micoche.dime_asientos());
    micoche.configura_climatizador("si"); //o pedir con JoptionPane.showInputDialog("Pregunta: ")
    System.out.println(micoche.dime_climatizador());
    System.out.println(micoche.dime_peso_coche());
    System.out.println("Precio final: "+ micoche.precio_coche());
    }
}
```

Salida:

```
mi coche es de la marca Renault
mi coche es de la marca Seat
el color del coche es rojo
vehículo de 4 ruedas. Longitud 2 metros. Ancho 300 cm y peso 700 Kg
tiene asientos de cuero
Vehiculo con climatizador
Peso total del coche: 1720
Precio final: 13500
```

Ejemplo de creación de una clase (URV)

```
public class Cotxe {
  private int capacitat;
  private int velocitat;
  private int potencia;
  public Cotxe (int cap, int vel, int pot) {
      capacitat = cap;
      velocitat = vel;
      potencia = pot;
  }
  public void accelerar() {
      velocitat = velocitat+10;
  }
  public void frenar() {
      velocitat = velocitat-10;
  } ...
}
```



Creación de objetos de la clase *Cotxe*:

```
public class UsaCotxe {
public static void main() {
    Cotxe sport = new Cotxe(5, 0, 105);
    Cotxe batmovil = new Cotxe (1, 0, 280);
    sport.accelerar();
    batmovil.frenar();
}
```

Encapsulació de la información

En Java se puede guardar la información en

- *Variables primitivas*: Se guarda el valor directamente en la variable (int, float…)
- *Variables de tipo abstracto*: Se guarda como una referencia a objetos: `Punto P1 = new Punto();`
- En *tablas o arrays*: Utilizan tambien una referencia para acceder a su contenido. No son objetos, no tienen métodos, pero utilizan algún atributo como .length. ejemplo: `int nota[] = new int[ 10 ]`
- *Referències a tipos enumerados*: Conjunto de valores en orden y fijo. ejemplo:
  `public` ***enum*** `Dias {Lunes, Martes, Miercoles, Jueves, Sabado, Dodimgo}` se declara: `Dias d;`
  `para acceder a un valor: d = Dias.Martes;` Usa los métodos: `valueOf()` y `ordinal()`
- En estructuras de datos como Listas, pilas y colas. Que se ven en el siguiente apartado.

Estructuras de datos en Java (Tipo abstractos de datos TAD)

Los arrays tiene un tamaño fijo y es difícil reordenar o insertar elementos.
Por eso estudiaremos 3 algoritmos de estructures de datos habituales: Listas, Pilas y Colas.

Listas: Permite almacenar elementos uno detrás de otro.

Pilas: Conjunto ordenado de elementos a los que sólo podemos acceder por un único lugar o extremo.
Tienen comportamiento LIFO (last-in, first-out).

Colas: Conjunto ordenado de elementos en los que añadimos información por un extremo y lo sacamos por el otro.
Tienen comportamiento FIFO (first-in, firstout)

Lista:
Acceso a los elementos:

```
// creamos un puntero al primer nodo de la lista
Nodo puntero=lista.primero;

// lo hacemos avanzar hasta el 4 nodo
int i=1;
while(i<4){
    puntero=puntero.siguiente;
    i++;
}

// usamos el contenido del nodo para algo útil
System.out.println("El 4to nombre almacenado es "+puntero.contenido);
```

Diseño de la clase ListaEnlazada

```
public class
Cliente{

  - String rut;
  - String nombre;
  - String apellido;
  - boolean esClienteFrecuente;

  public
  Cliente(String rut, String nombre, String apellido, boolean esClienteFrecuente){
    this.rut=rut;
    this.nombre=nombre;
    this.apellido=apellido;
    this.esClienteFrecuente=esClienteFrecuente;
  }

}
```

Métodos principales: añadir(), buscar() y remover().

Método para buscar:

```
while(puntero!=null){ // mientras el puntero no apunte al vacío

  if (puntero.contenido.rut.compareTo(rut)==0) // si los RUTs son iguales...
    return puntero.contenido; // ...entonces retorna el contenido del nodo apuntado

  puntero=puntero.siguiente; // en otro caso, se avanza al siguiente hasta encontrar
                             // una coincidencia
}
```

Sobrecarga del constructor

Un constructor se utiliza para inicializar un objeto recién creado.

Sobrecarga de constructor: una clase puede tener varios constructores con el mismo nombre, pero se difierencian en el número de parámetros.

Ej. Con tres constructores:

Cuenta (int a);
Cuenta (int a, int b);
Cuenta (String a, int b);

Herencia y Polimorfismo.

La Herencia: se utiliza para crear sublaceses aprovechando atributos de una clase padre o superclase.

- Se indica la extensión de la clase:
  `public class Classe2 extends Classe1.`
- `super` – hace referència a la parte de la madre/padre. (similar al concepto del this pero refiriéndose al padre)
- Acceso a la madre: Si esta tiene el modificador ***protected*** podemos acceder a las variables de les classe madre. Pero si son privadas, no.

```
public class Classe2 extends Classe1 {
    private int atribut2;
    public Classe2() {
      super();
      atribut2 = 0;
    }
    public void setAtribut2(int valor) {
      atribut2 = valor;
    }

    public int getAtribut2() {
       return(atribut2);
    }
    public String toString() {
         return(super.toString()+"At_2: "+atribut2);
     }
}
```

Polimorfismo.

- Una misma función puede ejecutar operaciones diferentes según el tipo de objecto que le ha llamado.

Tipos estatic y dinamic:

Tipo estático es el tipo en el que se declara una variable referencia y el Tipo dinámico es el tipo de objeto al que apunta la referencia.

```
Pare obj1 = new Pare(); // obj1 té el mateix tipus estàtic i dinàmic
Pare obj2 = new Filla(); // obj2 té el tipus estàtic Pare i el dinàmic
Filla Filla obj3 = new Pare(); // ERROR
Filla obj4 = new Filla(); // obj4 té el mateix tipus estàtic i dinàmic
```

En la referencia hacia arriba, un tipo estático limita la visibilidad del objeto al que referencia.

Ahora bien, el tipo dinámico determina qué método se ejecuta, en caso de sobreescritura.

Instance off permite distingir el método a ejecutar: `if (obj2 instanceof Filla) ((Filla)obj2).metode03();`

Jerarquia de classes

La clase ***Object*** es la raíz de la jerarquía de clases. Todas las clases que definimos son extensiones suyas.

clone(): Creates and returns a copy of this object.

equals (Object obj): Indicates whether some other object is "equal to" this one.

Clases y métodos abstractos:

Abstract, es una clase incompleta, porque hay métodos sin implementar, que ya serán implementados por cada subclase. `public abstract class FiguraGeometrica { . . . }`

Clases y métodos final:

Final, es una clase que no se puede extender. Lo indicaremos en las clases que no queremos bajo ningún concepto que puedan tener subclases. `public final class A { . . . }`

Interfaces:

Una clase sólo puede heredar información de una sola clase, sea abstracta o no.

Para heredar información de diferentes sitios tenemos las interfaces.

Entre interfaces puede haber herencia múltiple, ejemplo: `interface FiguraGrafica { . . . }`

Paquetes e importación:

Un package es una colección de clases con algo en común.

Las clases que están dentro de un paquete deben incluir al inicio: `package packagename;`

Si desde una clase del paquete A quiero acceder a una clase del paquete B, añadir al inicio: `import paqueteB.*;`

(el asterisco significa que doy acceso a todas las clases del paquete).

También se puede usar otro paquete así: `paquetB.Classe1 = new paquetB.Classe1();`

Multihilos:

Multithreading permite varios subprocesos simultáneamente. También llamado concurrencia. Al comprtir recursos comunes y reducir memoria. Los cambios son más rápidos:

```
Thread thread1 = new Thread();        thread1.start();
Thread thread2 = new Thread();        thread2.start();
```

Juegos en Java — Curso de Java Nivel 2

- Si usas Eclipse:
  - Pulsa en File: New – *Java Project*. Pon el nombre al proyecto: **Juego**
  - En el panel izquierdo se mostrará el proyecto con la carpeta ***src***.
  - Pulsa en File: *New – Class*. Pon el nombre a la clase: **Game**.
  - Activa la casilla, ☑ *Public static main void* te escribirá la función principal dentro de la clase.
- Si usas VSCode: (Recuerda que en VSCode mejor tener instalada la extensión : *Java Extension Pack)*
  - Crear la carpeta Juego y abrir la carpeta.
  - Archivo – Nuevo Archivo… New Java Class
  - En el explorador de VSCode pulsa en Create Java Project

JFrame: La Ventana

Para dibujar algo necesitamos una superficie donde pintar. Esta superficie o lienzo (Canvas en inglés) donde pintaremos nuestro primer ejemplo es un objeto ***JPanel*** que a su vez estará enmarcado en una ventana generada por la clase ***JFrame***.

El siguiente código crea una ventana con titulo "Mini Tennis" de 300 pixels por 300 pixels. La ventana no será visible hasta que llamemos setVisible(true). Si no incluimos la última línea "frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)", cuando cerremos la ventana el programa no terminará y seguirá ejecutándose. Guarda la unidad con el nombre **Juego**.java

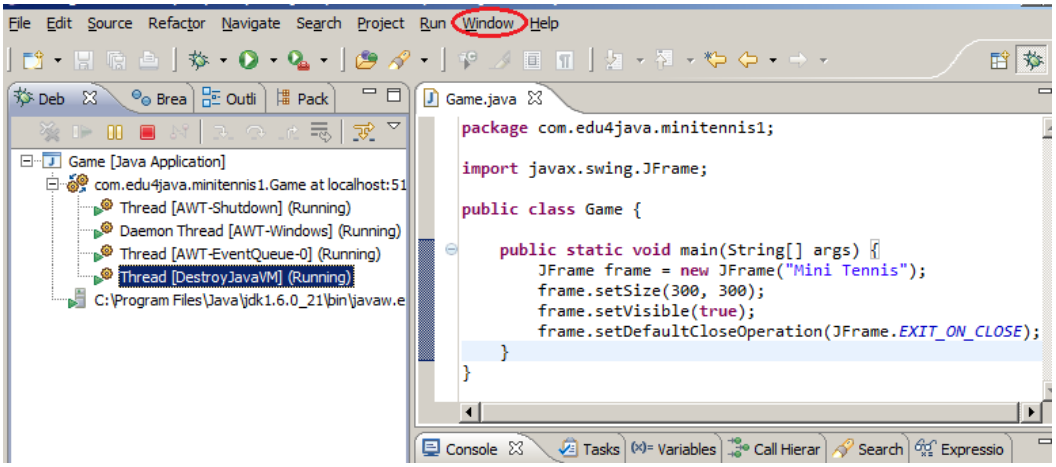
```
//package Juego;
import javax.swing.JFrame;    //o import java.awt.*;  importa la librería para dibujar la ventana
public class Juego {                                                    //clase Game
     public static void main(String[] args) {                           //función principal
          JFrame frame = new JFrame("Mini Tennis");                     //crea ventana
          frame.setSize(300, 300);                                      //pone ancho a 300x300
          frame.setVisible(true);                                       //pone en visible
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);         //sale al cerrar
        }
}
```

Al ejecutar (Run) obtenemos una ventana que se puede maximizar, minimizar, cambiar de tamaño con el ratón, etc. En realidad, cuando creamos una ventana, iniciamos un motor que se comunica con el sistema operativo. Este motor se llama "***Motor AWT***" o "***Motor Swing***". En las primeras versiones de java solo existía la librería AWT y luego se agregó Swing, que permite utilizar varios hilos de ejecución.

¿Qué es un hilo o thread en java?

Normalmente un programa se ejecuta por una línea de flujo. El hilo (Thread) permite a un programa iniciar varias ejecuciones a la vez, como si existieran varios programas ejecutándose al mismo tiempo. Aunque los hilos son herramientas muy potentes puede traer problemas al acceder a las mismas variables a la vez.

El Motor AWT inicia varios Hilos (Threads) que podemos ver en la vista *Debug* de *Eclipse* si escoges del menú: ***Window - Open perspective - Debug*** donde cada hilo es como si fuera un programa independiente ejecutándose al mismo tiempo que los otros hilos. El tercer hilo que vemos en la vista Debug llamado "Thread [AWT-EventQueue-0]" es el encargado de pintar la pantalla y recibir los eventos del teclado y el ratón. Vuelve a la vista Java: ***Window - Open perspective - Java***

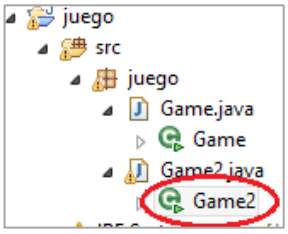


JPanel: El lienzo (Canvas en inglés)

Para poder dibujar necesitamos un objeto JPanel que incluiremos en la ventana. Extenderemos la clase JPanel para poder sobrescribir el método paint que es el método que llamará el Motor AWT para pintar.

Para ello crea una clase nueva llamada: Game2.

En esta clase nueva añade el código del recuadro.



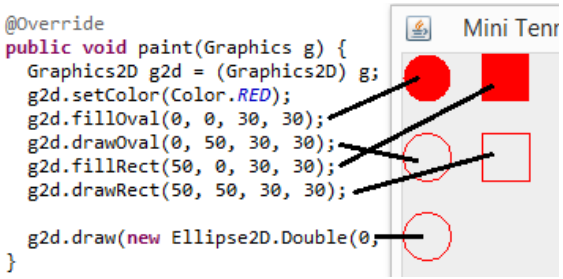
Al importar awt color, graphics y graphics2D los usamos en los parámetros del método Paint

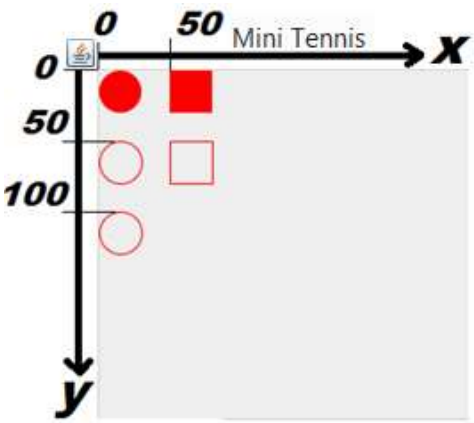
Graphics es la vieja clase usada por AWT que ha sido reemplazada por Graphics2D que tiene más funcionalidad.

El parámetro sigue siendo de tipo Graphics por compatibilidad pero nosotros siempre utilizaremos Graphics2D por lo que es necesario crear una variable g2d: "Graphics2D g2d = (Graphics2D) g;". Una vez que tenemos g2d podemos utilizar todos los métodos de Graphics2D para dibujar.

```
package juego;
  import java.awt.Color;
  import java.awt.Graphics;
  import java.awt.Graphics2D;
  import java.awt.RenderingHints;
  import java.awt.geom.Ellipse2D;
  import javax.swing.JFrame;
  import javax.swing.JPanel;

  @SuppressWarnings("serial")
public class Game2 extends JPanel {

    @Override
    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.setColor(Color.RED);   //poner color rojo
        g2d.fillOval(0, 0, 30, 30); // elipse relleno
        g2d.drawOval(0, 50, 30, 30); //elipse contorno
        g2d.fillRect(50, 0, 30, 30); //cuadrado relleno
        g2d.drawRect(50, 50, 30, 30); //cuadrado controno
        g2d.draw(new Ellipse2D.Double(0, 100, 30, 30)); //elipse
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Mini Tennis");
        frame.add(new Game2());
        frame.setSize(300, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Lo primero que hacemos es elegir el color que utilizamos para dibujar: "g2d.setColor(Color.RED);". Luego dibujamos unos círculos y cuadrados.





Coordenadas del lienzo: Cada punto del lienzo tiene una posición (x,y) asociada siendo (0,0) el punto de la esquina superior izquierda.

El primer circulo rojo se pinta con "g2d.fillOval(0, 0, 30, 30)": los primeros dos parámetros son la posición (x,y) y luego se indica el ancho y alto. Como resultado tenemos un circulo de 30 pixeles de diámetro en la posición (0,0).

El circulo vacío se pinta con "g2d.drawOval(0, 50, 30, 30)": el la posición x=0 (pegado al margen izquierdo) y la posición y=50 (50 pixeles más abajo del margen superior) pinta un circulo de 30 pixeles de alto y 30 de ancho.

Los rectángulos se pintan con "g2d.fillRect(50, 0, 30, 30)" y "g2d.drawRect(50, 50, 30, 30)" de forma similar a los círculos.

Por último "g2d.draw(new Ellipse2D.Double(0, 100, 30, 30))" pinta el ultimo circulo usando un objeto Ellipse2D.Double.

¿Cuándo el motor AWT llama al método paint?

El motor AWT llama al método paint cada vez que el sistema operativo le informa que es necesario pintar el lienzo. Cuando se carga por primera vez la ventana se llama a paint, si minimizamos y luego recuperamos la ventana se llama a paint, si modificamos el tamaño de la ventana con el ratón se llama a paint.

Animación de un objeto en movimiento

Este efecto de animación se consigue dibujando el círculo en posiciones x,y sucesivas cada cada cierto tiempo.

Creamos un método moveBall() que incrementará en 1 tanto a "x" como "y" cada vez que es llamado. En el método paint dibuja un circulo de 30 pixeles de diámetro en la posición (x,y) dada por las propiedades antes mencionadas "g2d.fillOval(x, y, 30, 30);".

Game loop - bucle

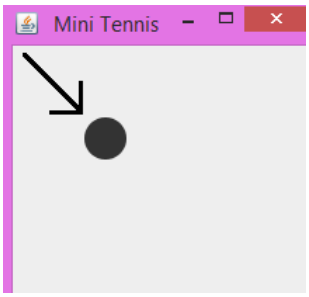
Al final del método main iniciamos un ciclo infinito "while (true)" donde repetidamente llamamos a moveBall() para cambiar la posición del circulo y luego llamamos a repaint() que fuerza al motor AWT a llamar al método paint para repintar el lienzo.

Este ciclo o repetición se conoce como "Game loop" o bucle y se caracteriza por realizar dos operaciones:

- Actualización (Update): dada tan solo por el método *moveBall*() que incrementa las propiedades "x" e "y" en 1.
- Renderizado (Render): dado por la llamada a repaint() y la subsecuente llamada a paint realizada por el motor AWT

Vuelve a reescribir la clase Game con el código de la derecha: →

Al ejecutar (Run Game) la bola se desplaza.




```
package juego;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game extends JPanel {
    int x = 0;      //creamos la variable entera x
    int y = 0;      //creamos la variable entera y
    private void moveBall() {  //functon que mueve x e y
        x = x + 1;
        y = y + 1;
    }
    @Override
    public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.fillOval(x, y, 30, 30);        //-> pinta la bola en x,y
    }
    public static void main(String[] args) throws InterruptedException {
        JFrame frame = new JFrame("Mini Tennis");
        Game game = new Game();
        frame.add(game);
        frame.setSize(300, 400);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        while (true) {  // bucle donde se mueve y dibuja la bola
            game.moveBall();
            game.repaint();
            Thread.sleep(10);
        }
    }
}
```

Analizando nuestro método paint

El método ***paint*** se ejecuta al iniciar el lienzo. Con el método ***repaint***() pedimos al Motor AWT que ejecute el método paint tan pronto como pueda y reflejar el cambio en la posición del circulo.

Creando archivo jar ejecutable y qué es la máquina virtual de java

```
@Override
public void paint(Graphics g) {
super.paint(g);
Graphics2D g2d = (Graphics2D) g;
g2d.setRenderingHint(
RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
g2d.fillOval(x, y, 30, 30);
}
```

La instrucción: RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON)" suaviza los bordes de las figuras como se puede ver en la imagen.

Analizando la concurrencia y el comportamiento de los hilos. Véase: http://www.edu4java.com/es/game/game2.html

Cuando se inicia la ejecución del método main sólo existe un hilo en ejecución. Luego se crean cuatro hilos.

"***Thread***.sleep(10)" le dice al procesador que el thread que se está ejecutando descanse por 10 milisegundos lo que permite que el procesador ejecute otros threads y en particular el thread AWT-EventQueue que llama al método paint.

Control de la dirección de movimiento

En el ejercicio anterior logramos que la pelota (el círculo) se moviera hacia abajo y a la derecha a un píxel por vuelta en el Game Loop. Cuando llegaba al límite de la pantalla la pelota seguía su curso desapareciendo del lienzo. Lo que haremos a continuación es que la pelota rebote en los límites del lienzo cambiando su dirección.

En el siguiente recuadro, se agregan dos propiedades "xa" y "ya" que representan la dirección en que se mueve la pelota. Si xa=1, la pelota se mueve hacia la derecha a un píxel por vuelta del Game Loop y si xa=-1, la pelota se mueve hacia la izquierda. Similarmente ya=1 mueve hacia abajo y ya=-1 mueve hacia arriba. Esto lo logramos con las líneas "x = x + xa" e "y = y + ya" del método moveBall().

Comprobamos con : (x + xa > getWidth() - 30) que la pelota no salga de del margen y cambiamos la dirección del movimiento sobre el eje x o lo que es lo mismo asignar menos uno a xa "xa = -1".

```
package juego;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game extends JPanel {

    int x = 0;      //-> posición coordenada x
    int y = 0;      //-> posición coordenada x
    int xa = 1;     //->dirección horizontal
    int ya = 1;     //->dirección vertical

    private void moveBall() {          //-> función para mover bola, se declara y crea antes que la función main
        if (x + xa < 0) xa = 1;
        if (x + xa > getWidth() - 30)  xa = -1;
        if (y + ya < 0) ya = 1;
        if (y + ya > getHeight() - 30) ya = -1;

        x = x + xa;            //->reasigna nuevo valor a x
        y = y + ya;            //->reasigna nuevo valor a y
    }

    @Override
    public void paint(Graphics g) {    //-> función para dibujar bola, se declara y crea antes que la función main
        super.paint(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        g.fillOval(x, y, 30, 30);          //-> dibuja bola en la coordenada x, y de 30x30

    }

    public static void main(String[] args) throws InterruptedException {
        JFrame frame = new JFrame("Mini Tennis");
        Game game = new Game();
        frame.add(game);
        frame.setSize(300, 400);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        while (true) {                     // -> este bucle llama repetitivamente a las funciones mover y pintar
            game.moveBall();
            game.repaint();
            Thread.sleep(10);              //-> pausa de 10 milisegundos en el hilo
        }
    }
}
```

Sprites

Cada objeto que se mueve en la pantalla tiene características propias como la posición (x,y), la velocidad y la dirección en que se mueve, etc. Todas estas características se pueden agrupar en un objeto con clase que llamaremos Sprite.

Crear el Sprite "*ball*" (pelota en inglés)

Crea una clase nueva llamada ***Ball*** (File-New-Class) para que agrupe todo lo referente a la pelota. →

El Sprite *Ball* necesita un dato del objeto Game2 para obtener los límites del lienzo y así saber cuándo debe cambiar de dirección.

En el método move() de la clase Ball se llama a game.getWidth() y game.getHeight().

```
package juego;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game2 extends JPanel {

    Ball ball = new Ball(this);

    private void move() {
        ball.move();
    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
        ball.paint(g2d);
    }

    public static void main(String[] args) throws InterruptedException {
        JFrame frame = new JFrame("Mini Tennis");
        Game2 game = new Game2();
        frame.add(game);
        frame.setSize(300, 400);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        while (true) {
            game.move();
            game.repaint();
            Thread.sleep(10);
        }
    }
}
```

```
package juego;
import java.awt.Graphics2D;

public class Ball {
    int x = 0;
    int y = 0;
    int xa = 1;
    int ya = 1;
    private Game2 game;

  public Ball(Game2 game) {
      this.game= game;
  }

    void move() {
    if (x + xa < 0)  xa = 1;
    if (x + xa > game.getWidth() - 30) xa = -1;
    if (y + ya < 0)  ya = 1;
    if (y + ya > game.getHeight() - 30) ya = -1;

    x = x + xa;
    y = y + ya;
}
```

En la clase Game2 vamos a reescribir el código adjunto para compararlo con el que tenemos en Game.

En el código adjunto, podemos ver como extraemos todo el código referente a la pelota de la clase Game2 y lo incorporamos a nuestra nueva clase *Ball*.

Si ejecutamos Game2 obtendremos el mismo resultado que si ejecutamos la versión anterior Game. La conveniencia de esta separación del código referente a la pelota en una clase de tipo *Sprite* se vuelve más obvia cuando incluimos la raqueta mediante un nuevo Sprite en un próximo ejercicio.

Eventos. Capturando las entrada por teclado

En este tutorial veremos como funcionan los eventos y en particular como obtener la información acerca de los eventos producidos en el teclado desde un programa java. Además explicaremos el concepto y uso de clases anónimas que es el método más comúnmente usado para manejar eventos en java. Abandonaremos nuestro juego momentáneamente y haremos un simple ejemplo de captura de eventos

Ejemplo de lectura del teclado

Para leer del teclado es necesario registrar un objeto que se encargue de "escuchar si una tecla es presionada". Este objeto conocido como "Listener" u "oyente" y tendrá métodos que serán llamados cuando alguien presione una tecla. En nuestro ejemplo el Listener se registra en el JPanel (o KeyboardExample) usando el método addKeyListener(KeyListener listener).

En el constructor de la clase KeyboardExample creamos el listener y lo registramos. Para que un objeto JPanel reciba las notificaciones del teclado es necesario incluir la instrucción setFocusable(true) que permite que KeyboardExample reciba el foco.

```
public KeyboardExample() {
        KeyListener listener = new MyKeyListener();
        addKeyListener(listener);
        setFocusable(true);
    }
```

La clase MyKeyListener es la que uso para crear el objeto Listener. Este Listener imprimirá en la consola el nombre del método y la tecla afectada por el evento.

Una vez registrado, cuando KeyboardExample (nuestro JPanel) tenga el foco y alguien oprima una tecla KeyboardExample informará al objeto listener registrado. El objeto Listener de nuestro ejemplo implementa la interfaz KeyListener que tiene los métodos keyTyped(), keyPressed() y keyReleased(). El método keyPressed será llamado cada vez que una tecla sea oprimida (y varias veces si se mantiene oprimida). El método keyReleased será llamado cuando solemos una tecla.

Los métodos antes mencionados reciben como parámetro un objeto KeyEvent que contiene información sobre que tecla se ha oprimido o soltado. Usando e.getKeyCode() podemos obtener el código de la tecla y si le pasamos un código de tecla a la función estatica KeyEvent.getKeyText(...) podemos obtener el texto asociado a la tecla.

```
package com.edu4java.minitennis4;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class KeyboardExample extends JPanel {

    public KeyboardExample() {
        KeyListener listener = new MyKeyListener();
        addKeyListener(listener);
        setFocusable(true);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Mini Tennis");
        KeyboardExample keyboardExample = new KeyboardExample();
        frame.add(keyboardExample);
        frame.setSize(200, 200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public class MyKeyListener implements KeyListener {
        @Override
        public void keyTyped(KeyEvent e) {
        }

        @Override
        public void keyPressed(KeyEvent e) {

    System.out.println("keyPressed="+KeyEvent.getKeyText(e.getKeyC
ode()));
        }

        @Override
        public void keyReleased(KeyEvent e) {

    System.out.println("keyReleased="+KeyEvent.getKeyText(e.getKey
Code()));
        }}}
```

```
public class MyKeyListener implements KeyListener {
        @Override
        public void keyTyped(KeyEvent e) {
        }

        @Override
        public void keyPressed(KeyEvent e) {
            System.out.println("keyPressed="+KeyEvent.getKeyText(e.getKeyCode()));
        }

        @Override
        public void keyReleased(KeyEvent e) {
            System.out.println("keyReleased="+KeyEvent.getKeyText(e.getKeyCode()));
    }   }
```

Clase anónima

En el ejemplo anterior la clase MyKeyListener será solo usada una vez por lo que podríamos reemplazarla por una clase anónima. KeyboardExample2 muestra como sería:

```
package minitennis;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class KeyboardExample2 extends JPanel {

  public KeyboardExample2() {
      KeyListener listener = new KeyListener() {
          @Override
          public void keyTyped(KeyEvent e) {
          }

          @Override
          public void keyPressed(KeyEvent e) {
              System.out.println("keyPressed="+KeyEvent.getKeyText(e.getKeyCode()));
          }

          @Override
          public void keyReleased(KeyEvent e) {
              System.out.println("keyReleased="+KeyEvent.getKeyText(e.getKeyCode()));
          }
      };
      addKeyListener(listener);
      setFocusable(true);
  }

  public static void main(String[] args) {
      JFrame frame = new JFrame("Mini Tennis");
      KeyboardExample2 keyboardExample = new KeyboardExample2();
      frame.add(keyboardExample);
      frame.setSize(200, 200);
      frame.setVisible(true);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

En el constructor de la clase KeyboardExample2 podemos ver como se reemplaza:
KeyListener listener = new MyKeyListener();

Por:

Esta instrucción tiene el mismo efecto que la anterior. Reemplaza la definición de la clase MyKeyListener por una clase anónima que hace exactamente lo mismo.

```
KeyListener listener = new KeyListener() {
        @Override
        public void keyTyped(KeyEvent e) {
        }

        @Override
        public void keyPressed(KeyEvent e) {
            System.out.println("keyPressed="+KeyEvent.getKeyText(e.getKeyCode()));
        }

        @Override
        public void keyReleased(KeyEvent e) {

    System.out.println("keyReleased="+KeyEvent.getKeyText(e.getKeyCode()));
        }
      };
```

La forma de crear un objeto desde una clase anónima es reemplazar el nombre de la clase a crear por una definición que empieza por la interfaz a implementar seguida por () y luego dentro de {} la definición de la clase como hacemos normalmente.

Aunque parezca un poco extraño esta es la forma más cómoda de implementar Listeners de eventos y es la forma que más encontrarán en código java avanzado.

Ahora sigamos con el desarrollo de nuestro juego en el próximo ejercicio.

Agregando el sprite raqueta

Agregaremos la raqueta mediante un Sprite llamado Racquet. La raqueta se moverá hacia la izquierda o derecha cuando oprimamos las teclas del cursor por lo que nuestro programa necesita leer del teclado.

Nuevo Sprite Racquet

Lo primero que hacemos es agregar en la clase Game una nueva propiedad llamada racquet donde mantendremos el Sprite que maneja la raqueta. En el método move() añadimos una llamada a racquet.move() y en paint() una llamada a racquet.paint(). Hasta ahora todo es similar al sprite Ball pero como la posición de la raqueta responde al teclado tenemos que hacer algo más.

En el constructor de la clase game se puede ver como se registra un listener para capturar los eventos del teclado. En el método *keyPressed*() del listener informamos a la raqueta que una tecla ha sido oprimida llamando a racquet.keyPressed(e). Lo mismo hacemos para *keyReleased*(). Con esto el Sprite racquet se enterará cuando se ha pulsado una tecla. Veamos ahora las clases *Ball* y *Racquet* que implementan los sprites.

```
package minitennis;

import java.awt.Graphics2D;

public class Ball {
    int x = 0;
    int y = 0;
    int xa = 1;
    int ya = 1;
    private Game game;

    public Ball(Game game) {
        this.game= game;
    }

    void move() {
        if (x + xa < 0)
            xa = 1;
        if (x + xa > game.getWidth() - 30)
            xa = -1;
        if (y + ya < 0)
            ya = 1;
        if (y + ya > game.getHeight() - 30)
            ya = -1;

        x = x + xa;
        y = y + ya;
    }

    public void paint(Graphics2D g) {
        g.fillOval(x, y, 30, 30);
    }
}
```

```
package minitennis;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game extends JPanel {

    Ball ball = new Ball(this);
    Racquet racquet = new Racquet(this);

    public Game() {
        addKeyListener(new KeyListener() {
            @Override
            public void keyTyped(KeyEvent e) {
            }

            @Override
            public void keyReleased(KeyEvent e) {
                racquet.keyReleased(e);
            }

            @Override
            public void keyPressed(KeyEvent e) {
                racquet.keyPressed(e);
            }
        });
        setFocusable(true);
    }

    private void move() {
        ball.move();
        racquet.move();
    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
        ball.paint(g2d);
        racquet.paint(g2d);
    }

    public static void main(String[] args) throws InterruptedException {
        JFrame frame = new JFrame("Mini Tennis");
        Game game = new Game();
        frame.add(game);
        frame.setSize(300, 400);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        while (true) {
            game.move();
            game.repaint();
            Thread.sleep(10);
     }}}
```

A diferencia de Ball, Racquet no tiene propiedades para la posición "y". Esto es debido a que la raqueta no variará su posición vertical, solo se moverá hacia la izquierda o derecha, nunca hacia arriba o abajo. En el método paint la instrucción g.fillRect(x, 330, 60, 10) define un rectángulo de 60 por 10 pixeles en la posición (x,y)=(x,330). Como vemos "x" puede variar pero "y" está fijada a 330 pixeles del limite superior del lienzo.

El método move() es similar al de Ball en el sentido de incrementar en "xa" la posición "x" y controlar que el sprite no se salga de los limites.

```
public void move() {
        if (x + xa > 0 && x + xa < game.getWidth()-60)
                x = x + xa;
}
```

Inicialmente el valor de "x" es cero lo que indica que la raqueta estará en el limite izquierdo del lienzo. "xa" también está inicializado a cero, lo que hace que en principio la raqueta aparezca estática ya que x = x + xa no modificará "x" mientras "xa" sea cero.

Cuando alguien presione una tecla el método keyPressed de Racquet será llamado y este pondrá "xa" en 1 si la tecla presionada es la de dirección derecha (KeyEvent.VK_RIGHT) lo que a su vez hará que la raqueta se mueva a la derecha la próxima vez que se llame al método move (recordar x = x + xa). De la misma forma si se presiona la tecla KeyEvent.VK_LEFT se moverá a la izquierda.

```
public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_LEFT)
                xa = -1;
        if (e.getKeyCode() == KeyEvent.VK_RIGHT)
                xa = 1;
}
```

Cuando una tecla deja de ser presionada el método keyReleased es llamado y "xa" pasa a valer cero lo que hace que el movimiento de la raqueta se detenga.

```
public void keyReleased(KeyEvent e) {
        xa = 0;
}
```

Si ejecutamos el ejemplo podemos ver como la pelota se mueve rebotando contra los límites y la raqueta se mueve cuando presionamos las teclas de dirección correspondientes. Pero cuando la pelota choca con la raqueta la atraviesa pareciendo como si esta no existiese. En el próximo tutorial veremos como hacer que la pelota rebote sobre la raqueta.

Detección de colisiones

Aprenderemos como detectar cuando un sprite choca con otro. En nuestro juego haremos que la pelota rebote contra la raqueta. Además haremos que el juego termine si la pelota alcanza el límite inferior del lienzo mostrando una ventana popup con el clásico mensaje: "Game Over".
A continuación vemos esta nueva clase Game que es idéntica a la anterior con la sola diferencia de que se ha agregado el método: gameOver();

```
package minitennis;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game extends JPanel {

        Ball ball = new Ball(this);
        Racquet racquet = new Racquet(this);

        public Game() {
                addKeyListener(new KeyListener() {
                        @Override
                public void keyTyped(KeyEvent e) {
        }

        @Override
        public void keyReleased(KeyEvent e) {
        racquet.keyReleased(e);
        }

        @Override
        public void keyPressed(KeyEvent e) {
        racquet.keyPressed(e);
        }
        });
        setFocusable(true);
        }

        private void move() {
        ball.move();
        racquet.move();
        }

        @Override
        public void paint(Graphics g) {
                super.paint(g);
                Graphics2D g2d = (Graphics2D) g;

        g2d.setRenderingHint(RenderingHints.KEY_ANTIA
LIASING,

        RenderingHints.VALUE_ANTIALIAS_ON);
```

```
                ball.paint(g2d);
                racquet.paint(g2d);
        }

        public void gameOver() {
        JOptionPane.showMessageDialog(this, "Game
Over", "Game Over", JOptionPane.YES_NO_OPTION);
        System.exit(ABORT);
        }

        public static void main(String[] args) throws
InterruptedException {
        JFrame frame = new JFrame("Mini Tennis");
        Game game = new Game();
        frame.add(game);
        frame.setSize(300, 400);
        frame.setVisible(true);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON
_CLOSE);

        while (true) {
                game.move();
                game.repaint();
                Thread.sleep(10);
                }
        }
}
```

El método gameOver() muestra un mensaje en una ventan del tipo cuadro de diálogo: JOptionPane.showMessageDialog con el mensaje "Game Over" y un botón de "Aceptar". Después llamamos a : System.exit(ABORT) para que se termine el programa. El método gameOver() es público ya que será llamado desde el sprite Ball cuando detecte que ha llegado al límite inferior del lienzo.

Colisión de Sprites

Para detectar la colisión entre la pelota y la raqueta usaremos rectángulos.
En el caso de la pelota crearemos un cuadrado alrededor de la pelota.

La clase java.awt.Rectangle tiene un método ***intersects***(Rectangle r) que retorna *true* cuando dos rectángulos ocupan el mismo espacio. Este método no es la intersección exacta con la bola pero para nuestro ejemplo es suficiente.

En la clase Racquet se ha agregado el método getBounds() que retorna un objeto de tipo rectángulo indicando la posición de la raqueta. Este método será usado por el sprite *Ball* para saber la posición de la raqueta y así detectar la colisión:

```
package minitennis;  //racquet.java

import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.event.KeyEvent;

public class Racquet {
        private static final int Y = 330;  //->constante Y
        private static final int WITH = 60; //->constante
        private static final int HEIGHT = 10;  //->constante
        int x = 0;      //-> variable  x
        int xa = 0;     //-> variable  y
        private Game game;

public Racquet(Game game) {
        this.game = game;
        }

public void move() {
        if (x + xa > 0 && x + xa < game.getWidth() - WITH)
        x = x + xa;
        }

public void paint(Graphics2D g) {
        g.fillRect(x, Y, WITH, HEIGHT);
        }

public void keyReleased(KeyEvent e) {
        xa = 0;
        }

public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_LEFT)
        xa = -1;
        if (e.getKeyCode() == KeyEvent.VK_RIGHT)
        xa = 1;
        }

public Rectangle getBounds() {
        return new Rectangle(x, Y, WITH, HEIGHT);
        }

public int getTopY() {
        return Y - HEIGHT;
        }
}
```

Otro cambio que funcionalmente no afecta pero que es una buena práctica es la inclusión de constantes:

```
        private static final int Y = 330;
        private static final int WITH = 60;
        private static final int HEIGH = 20;
```

La forma de definir una constante en java es declarando una propiedad "static final" y en mayúsculas. El compilador permite usar minúsculas pero el estándar dice que se deben usar mayúsculas para los nombres de las constantes.

Por último la clase Ball:

```
package com.edu4java.minitennis;  // Ball.java

import java.awt.Graphics2D;
import java.awt.Rectangle;

public class Ball {
	private static final int DIAMETER = 30;
	int x = 0;
	int y = 0;
	int xa = 1;
	int ya = 1;
	private Game game;

  public Ball(Game game) {
		this.game= game;
	}

  void move() {
		if (x + xa < 0)
			xa = 1;
		if (x + xa > game.getWidth() - DIAMETER)
			xa = -1;
		if (y + ya < 0)
			ya = 1;
		if (y + ya > game.getHeight() - DIAMETER)
			game.gameOver();
		if (collision()){
			ya = -1;
			y = game.racquet.getTopY() - DIAMETER;
		}
		x = x + xa;
		y = y + ya;
	}

private boolean collision() {
	return
	game.racquet.getBounds().intersects(getBounds());
	}

public void paint(Graphics2D g) {
		g.fillOval(x, y, DIAMETER, DIAMETER);
	}

public Rectangle getBounds() {
		return new Rectangle(x, y, DIAMETER,
DIAMETER);
	}
}
```

De forma similar a la clase Racquet se ha incluido el método getBounds() y la constante DIAMETER.

Más interesante es la aparición de un nuevo método llamado ***collision***() que retorna true (verdadero) si el rectángulo ocupado por la raqueta "game.racquet.getBounds()" intersecta al rectángulo que encierra a la pelota "getBounds()".

```
	private boolean collision() {
		return game.racquet.getBounds().intersects(getBounds());
	}
```

Si la colisión se produce, además de cambiar la dirección ajustaremos la posición de la pelota. Si la colisión es por el lado, la pelota podría estar varios pixeles por debajo de la cara superior de la raqueta. En el siguiente game loop aunque la pelota se movería hacia arriba (figura 2) podría todavía estar en colisión con la raqueta.

Para evitar esto colocamos a la pelota sobre la raqueta mediante: `y = game.racquet.getTopY() - DIAMETER;`
El método getTopY() de Racquet nos da la posición en el eje y de la parte superior de la raqueta y restando DIAMETER conseguimos la posición y exacta donde colocar la pelota para que esté sobre la raqueta.

Por último es el método move() de la clase Ball el que usa los nuevos métodos collision() y gameOver() de la clase Game. El rebote al alcanzar el lÍmite inferior ha sido reemplazado por una llamada a game.gameOver().

```
		if (y + ya > game.getHeight() - DIAMETER)
			game.gameOver();
```

Y poniendo un nuevo condicional usando el método collision() logramos que la pelota rebote hacia arriba si esta colisiona con la raqueta:

```
	if (collision())  ya = -1;
```

Agregando sonido a nuestro juego

Un juego sin sonido no está completo. En este tutorial agregaremos música de fondo, el ruido del rebote de la pelota y un "Game Over" con voz graciosa al terminar el juego. Para evitar problemas de copyright vamos a crear nosotros mismos los sonidos.

Creando sonidos

Para crear los sonidos me tomé la libertad de buscar en Google "free audio editor" y como respuesta encontré http://free-audio-editor.com/. Tengo que decir que la versión gratis de este producto es potente y fácil de manejar.

Con este editor he creado los archivos: back.wav, gameover.wav y ball.wav. En el video de youtube pueden ver como lo hice y crearlos ustedes mismos. También pueden descargar y usar estos tres que en esta misma línea los declaro libres de copyright. Lo que tienen que hacer es copiar estos archivos al paquete com.edu4java.minitennis7.

Reproducir sonidos usando AudioClip

Para reproducir los archivos de sonido usaremos la clase AudioClip. Crearemos objetos AudioClip usando el método estático de la clase Applet: Applet.newAudioClip(URL url). Este método necesita un objeto URL que le indique donde está el archivo de audio que queremos cargar para luego reproducir. La siguiente instrucción crea un objeto URL utilizando una ubicación en Internet:

URL url = new URL("http://www.edu4java.com/es/game/sound/back.wav");

La siguiente utiliza un directorio dentro del sistema de archivos local:

URL url = new URL("file:/C:/eclipseClasic/workspace/minitennis/src/com/edu4java/minitennis7/back.wav");

Nosotros buscaremos nuestro archivo utilizando el classpath. Este es el sistema que usa java para cargar las clases o mejor dicho los archivos *.class que definen las clases del programa. Para obtener un URL desde el classpath se utiliza el método getResource(String name) de la clase Class donde name es el nombre del archivo que queremos obtener.

A continuación vemos dos formas de como conseguir el URL del archivo "back.wav" que está en el mismo paquete que la clase SoundTest o lo que es lo mismo en el mismo directorio donde esta el archivo SoundTest.class.

URL url = SoundTest.class.getResource("back.wav");
URL url = new SoundTest().getClass().getResource("back.wav");

Tanto "SoundTest.class" como "new SoundTest().getClass()" nos dan un objeto class que tiene el método getResource que queremos usar.

He creado la clase SoundTest con el sólo propósito de mostrarles como trabaja AudioClip y no es necesaria para nuestro juego. A continuación se muestra el código fuente de SoundTest completo:

```
package com.edu4java.minitennis;

import java.applet.Applet;
import java.applet.AudioClip;
import java.net.URL;

public class SoundTest {
	public static void main(String[] args) throws Exception {
//		System.out.println("1");
//		URL url = new URL("http://www.edu4java.com/es/game/sound/back.wav");
//		System.out.println("2");
//		AudioClip clip = Applet.newAudioClip(url);
//		System.out.println("3");
//		clip.play();
//		System.out.println("4");
//		Thread.sleep(1000);
//		URL url = new URL(
//		"file:/C:/eclipseClasic/workspace/minitennis/src/com/edu4java/minitennis7/back.wav");
		URL url = SoundTest.class.getResource("back.wav");
		AudioClip clip = Applet.newAudioClip(url);
		AudioClip clip2 = Applet.newAudioClip(url);
		clip.play();
		Thread.sleep(1000);
		clip2.loop();
		Thread.sleep(20000);
		clip2.stop();
		System.out.println("end");
	}
}
```

De esta forma el archivo back.wav se obtienen desde el classpath. El classpath es el conjunto de directorios y archivos *.jar desde donde nuestro programa puede leer las clases (archivos *.class).

Una ventaja de esta metodología es que sólo tenemos que indicar la posición del archivo con respecto a la clase que lo usa. En nuestro caso como está en el mismo paquete basta con el nombre "back.wav". Otra ventaja es que los archivos de sonido se pueden incluir en un archivo *.jar. Veremos más sobre archivos *.jar más adelante. Una ves que tenemos el objeto URL podemos crear objetos AudioClip usando Applet.newAudioClip(url).

AudioClip clip = Applet.newAudioClip(url);

```
AudioClip clip2 = Applet.newAudioClip(url);
```

El objeto AudioClip tiene un método play() que inicia un thread independiente que reproduce sólo una vez el audio contenido en el archivo. Para reproducir el audio en forma repetitiva podemos usar el método loop() de AudioClip que reproducirá el sonido una y otra vez hasta que se llame al método stop sobre el mismo objeto AudioClip.

Dos audioClips pueden reproducirse al mismo tiempo. En el ejemplo creo dos audioClips con el mismo audio: clip y clip2. Reproduzco clip con play, espero un segundo Thread.sleep(1000) y reproduzco clip2 con loop. El resultado es una mezcla de los dos audios. Por ultimo después de 20 segundos Thread.sleep(20000) llamo a clip2.stop() y detengo la repetición de clip2.

Creando una clase Sound para nuestro juego

Para guardar los audioclips de nuestro juego creamos una clase Sound que tendrá una constante con un audioclip por cada sonido que usemos. Estas constantes son públicas para que cualquier objeto que tenga acceso a ellas pueda reproducirlas. Por ejemplo en la clase Ball podemos reproducir el sonido del rebote de la pelota usando Sound.BALL.play() en el momento que detectamos que la pelota cambia de dirección

```
package com.edu4java.minitennis;
import java.applet.Applet;
import java.applet.AudioClip;

public class Sound {
    public static final AudioClip BALL =
Applet.newAudioClip(Sound.class.getResource("ball.wav"));
    public static final AudioClip GAMEOVER =
Applet.newAudioClip(Sound.class.getResource("gameover.wa
v"));
    public static final AudioClip BACK =
Applet.newAudioClip(Sound.class.getResource("back.wav"));
}
```

Los objetos audioclips se crearán al cargarse la clase Sound la primera vez que alguien use la clase Sound. A partir de este momento serán reutilizados una y otra vez. Ahora veamos las modificaciones en la clase Game:

```
package com.edu4java.minitennis;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game extends JPanel {
    Ball ball = new Ball(this);
    Racquet racquet = new Racquet(this);

    public Game() {
        addKeyListener(new KeyListener() {
        @Override
        public void keyTyped(KeyEvent e) {
        }
        @Override
        public void keyReleased(KeyEvent e) {
        racquet.keyReleased(e); }


        @Override
        public void keyPressed(KeyEvent e) {
        racquet.keyPressed(e); }
        });
        setFocusable(true);
        Sound.BACK.loop();
    }
    private void move() {
        ball.move();
        racquet.move();}
    @Override
    public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2d = (Graphics2D) g;

    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING
,   RenderingHints.VALUE_ANTIALIAS_ON);
        ball.paint(g2d);
        racquet.paint(g2d);
    }
    public void gameOver() {
        Sound.BACK.stop();
        Sound.GAMEOVER.play();
        JOptionPane.showMessageDialog(this, "Game
Over", "Game Over", JOptionPane.YES_NO_OPTION);
        System.exit(ABORT);
    }
    public static void main(String[] args) throws
InterruptedException {
        JFrame frame = new JFrame("Mini Tennis");
        Game game = new Game();
        frame.add(game);
        frame.setSize(300, 400);
        frame.setVisible(true);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
;
            while (true) {
                game.move();
                game.repaint();
                Thread.sleep(10);
            }
        }
}
```

En la última línea del constructor de la clase Game añadimos Sound.BACK.loop(), lo que iniciará la reproducción de nuestra música de fondo que se repetirá hasta que se alcance el método gameOver(), donde detenemos la música de fondo con Sound.BACK.stop(). A continuación de Sound.BACK.stop() y antes del popup informamos que se termino la partida reproduciendo "Game Over" Sound.GAMEOVER.play().

En la clase **Ball** modificamos el método **move**() para que se reproduzca **Sound.BALL** cuando la pelota rebote.

```
package com.edu4java.minitennis;

import java.awt.Graphics2D;
import java.awt.Rectangle;

public class Ball {
	private static final int DIAMETER = 30;

	int x = 0;
	int y = 0;
	int xa = 1;
	int ya = 1;
	private Game game;

	public Ball(Game game) {
		this.game = game;
	}

	void move() {
		boolean changeDirection = true;
		if (x + xa < 0)
			xa = 1;
		else if (x + xa > game.getWidth() - DIAMETER)
			xa = -1;
		else if (y + ya < 0)
			ya = 1;
		else if (y + ya > game.getHeight() - DIAMETER)
			game.gameOver();
		else if (collision()){
			ya = -1;
			y = game.racquet.getTopY() - DIAMETER;
		} else
			changeDirection = false;

		if (changeDirection)
			Sound.BALL.play();
		x = x + xa;
		y = y + ya;
	}

	private boolean collision() {
		return game.racquet.getBounds().intersects(getBounds());
	}

	public void paint(Graphics2D g) {
		g.fillOval(x, y, DIAMETER, DIAMETER);
	}

	public Rectangle getBounds() {
		return new Rectangle(x, y, DIAMETER, DIAMETER);
	}
}
```

En move() agregamos una variable *changeDirection* que inicializamos a true. Añadiendo un else a cada if y colocando un changeDirection = false que sólo se ejecutará si ninguna condición en los if es cumplida, conseguimos enterarnos si la bola ha rebotado. Si la pelota ha rebotado changeDirection será verdadero y Sound.BALL.play() será ejecutado.

Agregando puntuación y aumentando la velocidad

Todo juego necesita una medida de logro o éxito. En nuestro caso incluiremos en el rincón izquierdo de la pantalla nuestra puntuación que no será más que la cantidad de veces que logramos pegarle a la pelota con la raqueta. Por otro lado el juego debería ser cada vez más difícil para no matar de aburrimiento al jugador. Para esto aumentaremos la velocidad del juego cada vez que rebote la pelota en la raqueta.
Los objetos móviles del juego son la pelota y la raqueta. Modificando la velocidad de movimiento de estos dos objetos modificaremos la velocidad del juego. Vamos a incluir una propiedad llamada speed en la clase Game para mantener la velocidad del juego. La propiedad speed será inicialmente 1 e irá incrementándose cada vez que le demos a la pelota con la raqueta.

Para la puntuación necesitaríamos otra propiedad a incrementar cada vez que golpeemos la pelota. En vez de crear una nueva propiedad se me ocurrió reutilizar speed. El único inconveniente es que las puntuaciones suelen iniciarse en 0 y no en 1 como speed. La solución que se me ocurrió fue agregar un método getScore() que retorne el valor de speed menos uno.

```
private int getScore() {
        return speed - 1;
}
```

Veamos las modificaciones hechas en la clase **Game**:

package com.edu4java.minitennis8;

```
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Game extends JPanel {

    Ball ball = new Ball(this);
    Racquet racquet = new Racquet(this);
    int speed = 1;

    private int getScore() {
        return speed - 1;
    }

    public Game() {
        addKeyListener(new KeyListener() {
            @Override
            public void keyTyped(KeyEvent e) {
            }
            @Override
            public void keyReleased(KeyEvent e) {
                racquet.keyReleased(e);
            }
            @Override
            public void keyPressed(KeyEvent e) {
                racquet.keyPressed(e);
            }
        });
        setFocusable(true);
        Sound.BACK.loop();
    }

    private void move() {
        ball.move();
        racquet.move();
    }
    @Override
    public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2d = (Graphics2D) g;

    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING
    , RenderingHints.VALUE_ANTIALIAS_ON);
    ball.paint(g2d);
      racquet.paint(g2d);

      g2d.setColor(Color.GRAY);
      g2d.setFont(new Font("Verdana", Font.BOLD, 30));
      g2d.drawString(String.valueOf(getScore()), 10, 30);
      }

      public void gameOver() {
          Sound.BACK.stop();
          Sound.GAMEOVER.play();
          JOptionPane.showMessageDialog(this,
          "Puntuación: " + getScore(), "Game Over",
          JOptionPane.YES_NO_OPTION);
          System.exit(ABORT);
      }

      public static void main(String[] args) throws
      InterruptedException {
          JFrame frame = new JFrame("Mini Tennis");
          Game game = new Game();
          frame.add(game);
          frame.setSize(300, 400);
          frame.setVisible(true);
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

          while (true) {
              game.move();
              game.repaint();
              Thread.sleep(10);
          }
      }
}
```

Para pintar la puntuación en el rincón superior izquierdo al final del método paint he agregado:

```
g2d.setColor(Color.GRAY);
g2d.setFont(new Font("Verdana", Font.BOLD, 30));
g2d.drawString(String.valueOf(getScore()), 10, 30);
```

En la primera línea elegimos el color gris, en la segunda línea el tipo de letra Verdana, negrita de 30 pixeles y finalmente en la posición (x,y) igual a (10,30) donde dibujamos la puntuación.

En el método gameOver() modificamos el segundo parámetro para mostrar la puntuación alcanzada:

```
JOptionPane.showMessageDialog(this, "your score is: " + getScore(),
                "Game Over", JOptionPane.YES_NO_OPTION);
```

En la clase Ball el método move() ha sido modificado para considerar la nueva propiedad de velocidad "game.speed". Cuando la pelota cambiaba de dirección las propiedades de velocidad xa y ya eran modificadas a 1 o -1. Ahora

considerando la velocidad estas propiedades son cambiadas a game.speed o -game.speed. También se ha agregado en el condicional if(collision()) que la velocidad se incremente "game.speed++".

```
package minijuego;

import java.awt.Graphics2D;
import java.awt.Rectangle;

public class Ball {
    private static final int DIAMETER = 30;
    int x = 0;
    int y = 0;
    int xa = 1;
    int ya = 1;
    private Game game;

public Ball(Game game) {
        this.game= game;
    }

void move() {
        if (x + xa < 0)  //borde izdo
          xa = 1;
        if (x + xa > game.getWidth() - DIAMETER)
          xa = -1;
        if (y + ya < 0)
        ya = 1*game.speed;
         if (y + ya > game.getHeight() - DIAMETER)
        game.gameOver();
```

```
if (collision()){
    ya = -1*game.speed;  //saltos más rápidos
    y = game.racquet.getTopY() - DIAMETER;
    game.puntos = game.puntos + 1;  // suma 1 punto
    game.speed ++;
        }
        x = x + xa;
        y = y + ya;
    }

private boolean collision() {  //devuelve true si hay intersección
return game.racquet.getBounds().intersects(getBounds());
    }

public void paint(Graphics2D g) {
        g.fillOval(x, y, DIAMETER, DIAMETER);
    }

public Rectangle getBounds() {
        return new Rectangle(x, y, DIAMETER, DIAMETER);
    }
}
```

A continuación la clase **Racquet**:

```
package com.edu4java.minitennis8;

import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.event.KeyEvent;

public class Racquet {
    private static final int Y = 330;
    private static final int WITH = 60;
    private static final int HEIGHT = 10;
    int x = 0;
    int xa = 0;
    private Game game;

    public Racquet(Game game) {
        this.game = game;
    }
    public void move() {
        if (x + xa > 0 && x + xa < game.getWidth() -
WITH)
        x = x + xa;
```

```
        }
        public void paint(Graphics2D g) {
        g.fillRect(x, Y, WITH, HEIGHT);
    }
    public void keyReleased(KeyEvent e) {
        xa = 0;
    }
    public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_LEFT)
                xa = -game.speed;
        if (e.getKeyCode() == KeyEvent.VK_RIGHT)
                xa = game.speed;
    }
    public Rectangle getBounds() {
        return new Rectangle(x, Y, WITH, HEIGHT);
    }
    public int getTopY() {
        return Y - HEIGHT;
    }
}
```

Aquí la modificación es similar que en Ball. En el método keyPressed(KeyEvent e) la modificación de la velocidad xa pasa de -1 y 1 a -game.speed y game.speed.

Nota: según el estándar de "Java Beans" el acceso a la propiedad "game.speed" debería hacerse usando un método de la forma "game.getSpeed()". El accesos directo a una propiedad es considerado casi un pecado mortal en el ámbito de java empresarial. Curiosamente en el entorno de desarrollo de juegos es muy común y está justificado por eficiencia. Esto es especialmente importante en programación para móviles donde los recursos suelen ser más escasos.

Publicar la aplicación: Crear el archivo jar ejecutable y qué es la máquina virtual de java.

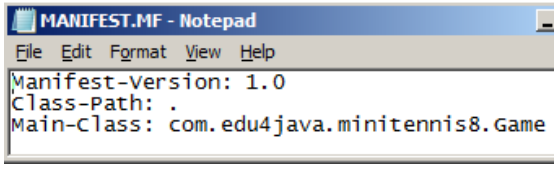
Un programa java necesita una máquina virtual para ser ejecutado. Java Virtual Machine (JVM).
En el caso de java cuando usamos el compilador no obtenemos código máquina. Lo que obtenemos es un código llamado *bytecode* que no se ejecuta directamente sobre una máquina real. Este bytecode solo se puede ejecutar en una máquina virtual. Una máquina virtual es un programa que se hace pasar por una máquina. Para cada sistema operativo diferente existirá un programa de máquina virtual especifico pero el bytecode que ejecutan será el mismo.

Compilación y ejecución en java: Existen dos versiones de instalación de java para cada sistema operativo: JRE y JDK. JRE Java Runtime Environment, es una versión reducida que contiene la JVM pero que no incluye el compilador java. JDK Java Development Kit contiene la JVM, el compilador java y muchas herramientas adicionales para el desarrollo de aplicaciones java. Si no tiene instalada la versión ***JDK*** tendrás que instalarla para poder continuar con este manual.

Archivo JAR

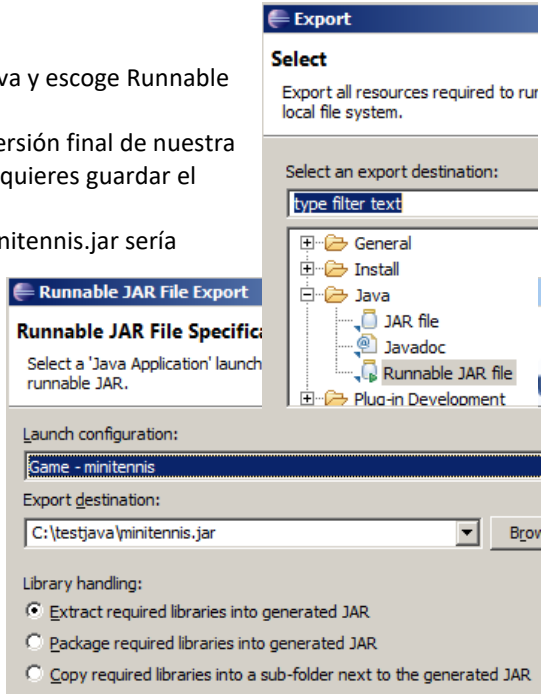
Un archivo jar no es más que un archivo comprimido con el algoritmo de compresión ZIP que puede contener:
Los archivos *.class que se generan a partir de compilar los archivos *.java que componen nuestra aplicación.
Los archivos de recursos que necesita nuestra aplicación (Por ejemplo los archivo de sonido *.wav)
Opcionalmente se puede incluir los archivos de código fuente *.java
Opcionalmente puede existir un archivo de configuración "META-INF/MANIFEST.MF".

Crear un archivo JAR ejecutable desde consola



Para que el archivo **jar** sea ejecutable hay que incluir en el archivo *MANIFEST.MF* una línea indicando la clase que contiene el método principal **main**() que se usará para iniciar la aplicación.
Es importante destacar que al final de la línea hay que agregar un retorno de carro para que funcione el archivo Manifest. Primero crear el archivo comprimido como *testjava.zip* que contenga el archivo .class, el directorio META-INF y dentro el archivo MANIFEST.MF. Una vez creado el archivo ***testjava.zip***, lo renombramos a ***testjava.jar*** y lo ejecutamos desde la línea de comandos:

Crear el archivo JAR ejecutable desde eclipse



Selecciona la opción "exportar" desde: File - ***Export***, Abre la carpeta Java y escoge Runnable JAR file como en la imagen.
En "Launch configuration" selecciona la que usamos para ejecutar la versión final de nuestra aplicación y en "Export destination" indica la carpeta y nombre donde quieres guardar el JAR.
Si java está bien instalado sobre Windows, con un doble click sobre minitennis.jar sería suficiente para ejecutar nuestra aplicación.

Si descomprimimos nuestro archivo minitennis.jar encontraremos los archivos *.class que componen nuestro juego. Estos archivos están dentro del árbol de directorios con los nombres de los paquetes java que contienen a las clases.

<u>Creación de un icono</u>
Trata de escoger una imagen fácil de recordar, o descriptiva. El tamaño de la imagen debe ser 256x256 para funcionar apropiadamente como icono.
Convierte la imagen a ICO (puedes utilizar convertico.com que es gratuito).

Generar archivo EXE:

Descarga ***launch4j***. Es un programa gratuito diseñado para compilar todas tus fuentes en un archivo ejecutable. Puedes descargar launch4j en http://sourceforge.net/projects/launch4j/files/launch4j-3/3.1.0-beta1
En el primer campo de texto, escribe o selecciona por medio del navegador la ubicación en la que deseas guardar tu archivo ejecutable. ¡Asegúrate de que el nombre del archivo tenga terminación ".exe"
En el segundo campo de texto, escribe o selecciona por medio del navegador el archivo .jar previamente exportado desde Eclipse. En el cuarto campo de texto, etiquetado como “icono:”, escribe o selecciona por medio del navegador el archivo “.ico” que convertimos previamente. Esto es opcional, y si lo dejas en blanco tu sistema operativo lo revertirá al icono de archivo ejecutable predeterminado. Bajo la pestaña “JRE” en la parte superior, selecciona la tab at the top, select the “versión Min JRE” e ingresa “1.4.0”. Esto asegura que los usuarios tengan una versión de Java suficiente como para utilizar tu programa. Puedes cambiar esta opción, pero 1.4.0 es una versión segura. Haz clic en el botón con forma de engranaje llamado "construir envoltorio" en la parte superior de la pantalla.

FIN

Aplicaciones con ventanas en Java

- Java **AWT** Es una librería para tener una interface gráfica tipo Windows.
- Java **Swing** Es una librería más evolucionada que AWT.

Ejemplo de ventana con la librería awt y con Swing:

```
import java.awt.*;

public class Ventana extends Frame
{
public Ventana()
{

super("Primera Ventana");
setLocation(100,100);
setSize(200,100);
show();
}

public static void main(String[] arg)
{
new Ventana();
System.out.println("he creado la ventana");
}
}
```

```
import javax.swing.*;

public class VentanaSwing extends JFrame
{
public VentanaSwing()
{

super("Primera Ventana");
setLocation(100,100);
setSize(200,100);
show();
}

public static void main(String[] arg)
{
new VentanaSwing();
System.out.println("He creado la ventana");
}
}
```

Aplicaciones Java para Android con el SDK Android en Eclipse

1. Instalar Eclipse con las SDK de Android.

Podemos instalar el ADT bundle de Google (android Developement) o el plugin kit de desarrollo de Android para Eclipse SDK_android.

Desde el sitio web: http://developer.android.com/sdk/

Pulsa en: *Download Eclipse ADT wirh SDK for Windows*.

Para descargar el programa completo Eclipse con el Kit de desarrollo SDK de Android.

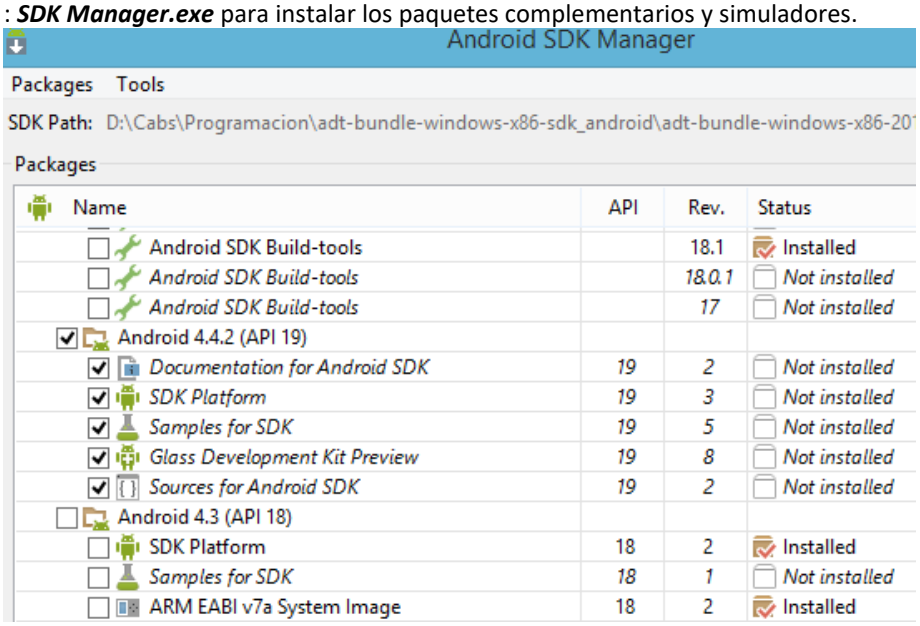
Nota: Si ya tienes *Eclipse* instalado en equipo, puedes bajarte solo el plugin SDK y añadirlo desde el propio programa de Eclipse en el menú: Help – *Install New Software*

2. Instalar y configurar las SDK en Windows:

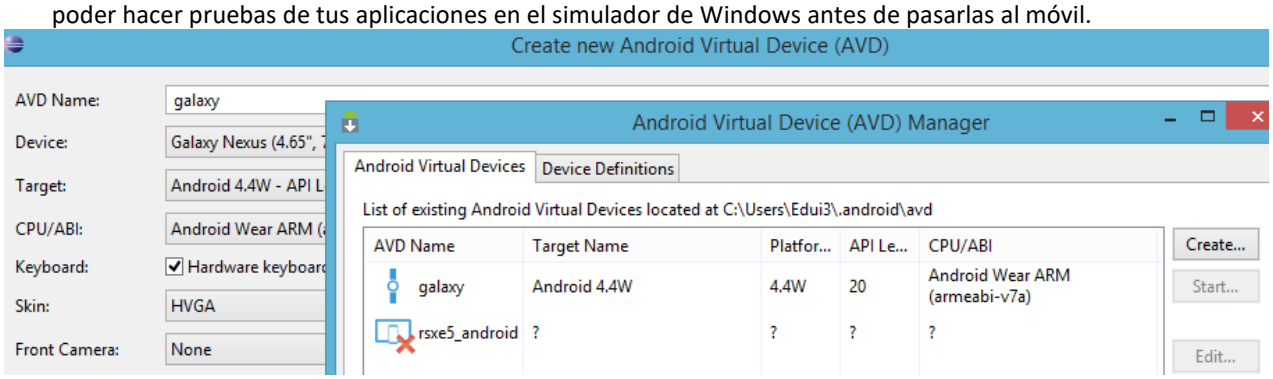
Una vez descargado el adt-bundle-windows, descomprime la carpeta mejor en una carpeta raíz de C.

En la carpeta: ***adt-bundle-windows-x86-sdk_android.***

Ejecuta el archivo : ***SDK Manager.exe*** para instalar los paquetes complementarios y simuladores.



Crea un Nuevo dispositivo virtual (Create new virtual device). Es decir, un simulador virtual de tu Smartphone para poder hacer pruebas de tus aplicaciones en el simulador de Windows antes de pasarlas al móvil.



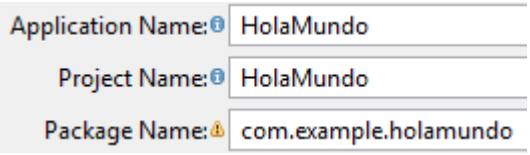
3. Primera aplicación en Eclipse

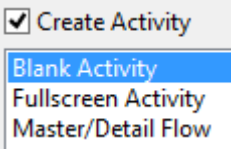
- Ejecuta Eclipse desde la carpeta de ***adt-bundle-windows-x86-sdk_android.***\Eclipse\Eclipse.exe.
- Para actualizar Eclipse con las SDK escoge del menú de Eclipse: Help – Check for updates.
- Para empezar la primera aplicación Android, escoge del menú: *File – New – Project – Android Project*.

O: *New - Android – Application Project* (según versión).

- Escribimos el nombre del proyecto: HolaMundo

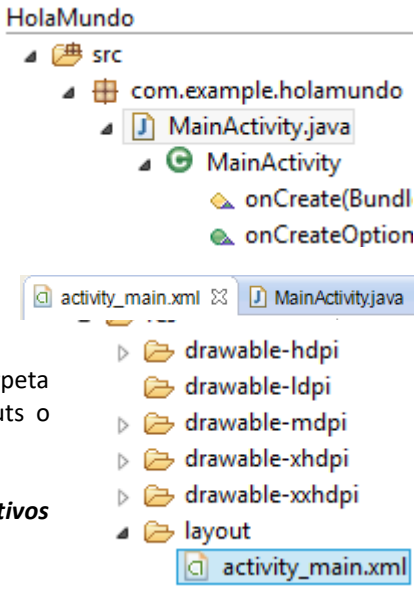
El nombre de la aplicación suele ser el mismo y el del paquete *Package* suele ponerlo auto.





- En las siguientes ventanas, escoge un logo y la actividad Blank.
- Pulsa para finalizar. Se creará el proyecto. Todo se hereda a partir de la clase Activity.
- En el panel izquierdo, en el explorador de paquetes, abre la carpeta ***src*** (resources) verás tu archivo fuente .java, donde dice que al crearse la aplicación (oncreate) se carga el layout o capa: activity_main. El aspecto visual se diseña en xml.
- Si pulsas en: ***actvity_main.xml*** verás el aspecto de tu aplicación

También puedes abrirla desde el explorador, en la carpeta: res – layout.



En resumen: La carpeta **rsc** suele contener la lógica y código fuente y la carpeta res el aspecto gráfico de la aplicación: imágenes, audio, capas layouts o pantallas, etc.

Exportar la aplicación a formato APK para ser ejecutada por dispositivos móviles.

- Limpiar primero la aplicación: Project – Clean
- Escoger del menú: Exportar - Aplicación Android

En la ventana, poner el nombre, luego generar una nueva clave de tienda Keystore (pasword) y escoger la carpeta para guardar. En la siguiente ventana introducir los años de validación, nombre del autor.

- Por último, una vez exportado y copiado a la tarjeta o por USB, utilizar un instalador de paquetes .apk

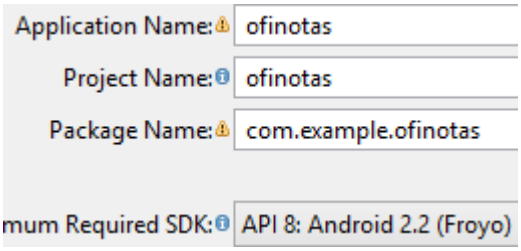
Se encuentra fácilmente en la app market de android. Algunos son programas instaladores de APK, que se instalan y ejecutan en el PC y los convierte desde cable usb o tarjeta de memoria, pero es más fácil instalar la aplicación en el Smartphone. Por ejemplo: APK instaler.

Actividades

Una aplicación estará formada por un conjunto de actividades independientes, es decir se trata de clases independientes que no comparten variables, aunque todas trabajan para un objetivo común. Otro aspecto importante es que toda actividad ha de ser una subclase de *Activity*.

Tema 2. Layouts - Proyecto base

- Prepara una imagen pequeña y una grande para el icono de la aplicación y el fondo splash
- Ejecuta el programa Eclipse con el pluging ***SDK_android***.
- Escoge del menú: File – New – Project – Android Project.
- Escribe el nombre del proyecto: **Ofinotas**

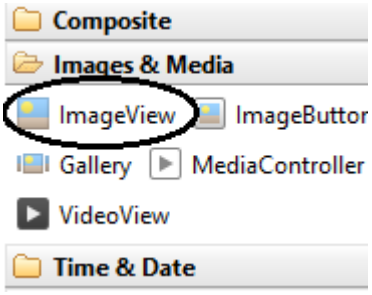


Mínimo SDK requerido: para Froyo

- Blank activity – Main activity
- Pulsa botón derecho sobre la carpeta: Ofinotas y escoge: Showinto para centrase en este proyecto.
- En la carpeta res – crea una carpeta general de dibujos (new – folder) llamada: drawable
- Arrastra a esa carpeta tu imagen de fondo. El nombre de la imagen debe ser toda en minúsculas sin números.
- Abre la carpeta: res - *layout* y pulsa doble clic sobre: *activity_main.xml* para trabajar con ella. Verás que está en formato xml
- Elimina las líneas de margin y cambia el texto como se muestra en el recuadro

Truco: Control + espacio muestra, al escribir, la ayuda contextual.

```
RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   tools:context=".MainActivity" >

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world" />
```



- Pulsa en la pestaña: *Graphical Layout* y en la sección *Images&Media* arrastra el componente *ImageView* a la pantalla.
- Escoge el nombre de la imagen que has añadido antes a la carpeta drawable.
- En la ventana gráfica, pulsa en sección: Texfields y arrastra el campo: ***multilinetext*** a la pantalla.
- En la pestaña: *activity_main.xml* cambia el alto del *edit text* por el valor: android:layout_height="150dp"

(150 dp no son pixels sino, puntos de densidad de pantalla).

- En la carpeta Src, pulsa doble clic en *Main_activity.java* para abrirla y ya puedes elegir : ***Run configurations*** … Ofinotas.

Pulsa en ***Run*** para comprobar la apariencia de la aplicación en el simulador. Recuerda pulsar en el botón *power* para encender primero el simulador y tener paciencia en la primera compilación.

Añadir un botón a un layout y añadir código en el evento onClick

Se puede añadir un botón desde *Graphical Layout* o manualmente desde fichero XML.

- En el grupo "Form Widgets" de la ventana "Palette" seleccionaremos "Button" y lo arrastraremos a la zona inferior del layout.
- En la parte derecha la ventana "Properties" indicaremos las dos propiedades más importantes:
  - Id: nombre que identificará el Button, por ejemplo "@+id/button1" (siempre indicaremos "@+id/nombre_identificativo":
  - El texto del botón, tendremos tres posibilidades:

1. Lo podemos escribir directamente en la propiedad "Text" (no es recomendable).
2. Lo podemos escribir en la actividad principal: android:text="Botón" />
3. Antes de asignarle el texto al Button crearemos un "Value" en "Resources". Para ello seguiremos los siguientes pasos; abriremos las siguientes carpetas en "Project Explorer": "res" - "values", haremos doble click en "strings.xml". En la parte derecha Eclipse nos mostrará los values actuales, pulsaremos "Add". Seleccionaremos “String” y pulsaremos “OK”. Introduciremos un nombre identificativo para el nuevo String en "Name", por ejemplo "button1" y un valor en "Value", por ejemplo "Guardar nota". Cerraremos el fichero "strings.xml" y guardaremos los cambios.

Asignar el código al evento onClick del botón para Salir:

La manera más rápida: añadimos a la sección del botón del archivo xml:

```
<Button
   android:id="@+id/button1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_alignParentBottom="true"
   android:layout_centerHorizontal="true"
   android:text="Guardar nota"
   android:onClick="salir"/>
   // -->aqui llamo a la accion salir//
```

Luego abriremos el fichero xxx.java de nuestra aplicación que estará en el "Project Explorer", en la carpeta "src".

Para trabajar con un Button deberemos añadir un import al principio del programa:

import android.widget.Button;

En la clase de la aplicación añadiremos las siguientes líneas:

```
package com.example.ofinotas;
import android.os.Bundle;
import android.app.Activity;
import android.view.*;
import android.widget.Button;

public class MainActivity extends Activity {

    protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main); }
    public boolean onCreateOptionsMenu(Menu menu) {
     // Inflate the menu; this adds items to the action bar if it is present.
     getMenuInflater().inflate(R.menu.main, menu);
     return true; }

     public void salir(View view) {finish();}
```

Hacer un block de notas

Diseño de la primera pantalla/layout

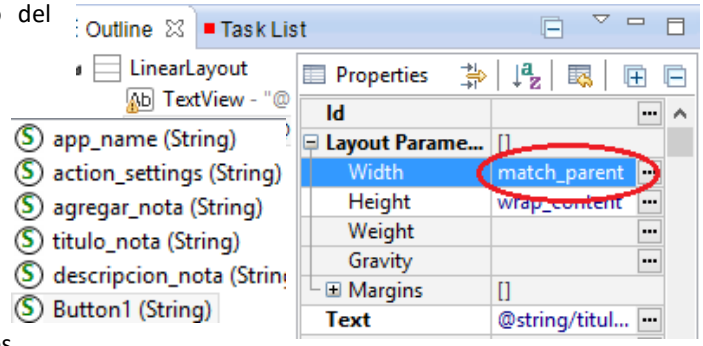
- Crear nuevo proyecto android con el nombre ***Notepad***.
- Abre la carpeta: res - *layout* y pulsa sobre doble clic sobre: *activity_main.xml* para trabajar con ella.
- Modifica y copia los TextView (labels o etiquetas de texto)

Puedes cambiar el ancho del texto para que ocupe todo su ancho de pantalla mediante el panel de **propiedades** del string o puedes cambiarlas desde el código del Layuot del Activity_main:

android:layout_width="*match_parent*"

Truco propiedades: también puedes pulsar el botón derecho sobre el Layout y escoger *Properties*. Entonces se muestran en el panel de abajo.

Al botón cambia la propiedad *gravity*=center_horizontal para centrarlo



- En la carpeta: ***Values*** -> ***strings.xml*** pulsa en ***Add*** para añadir las etiquetas que contendrán los textos de las 3 etiquetas TextView y del botón Button1:

agregar_nota, titulo_nota , descripcion_nota y button1 con sus correspondientes valores de texto.

Una vez terminado. Pulsa doble clic sobre la actividad principal de la carpeta src: *MainActivity.java* para seleccionarla y escoge de la barra de herramientas *Run* -> *Run configurations*.

En la ventana, selecciona el proyecto *Notepad* y pulsa en *Run* para ejecutar el simulador.

Como observarás, el botón no realiza ninguna acción y el texto no se guarda en ningún sitio.

Creación de la actividad para manipular la base de datos sqlite.

Escoge del menú: ***New –Other – Java***: Class (nueva clase) : Nombre: ControlDatos

Importamos algunas librerías, sobretodo las de database:

```
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
```

Funciones incorporadas en la librería Math

abs(x) valor absoluto
acos(x) arco coseno
asin(x) arco seno
atan(x) arco tangente
atan2(x,y) representación polar de x,y
ceil(x) menor entero mayor que x
cos(x) coseno
exp(x) e^x
floor(x) mayor entero menor que x
IEEEremainder(x,y) resto de la división x/y
log(x) logaritmo natural
max(x,y) el mayor de x e y
min(x,y) el menor de x e y
pow(x,y) x^y
random() número aleatorio entre 0 y 1
rint(x) entero más cercano a x (devuelve un doble)
round(x) entero más cercano a x
sin(x) seno
sqrt(x) raíz cuadrada
tan(x) tangente

CURS DE JAVA

CURS DE JAVA

OFIMEGA ACADÈMIES - SALOU

CURS DE JAVA

OFIMEGA

CURS DE JAVA