

viewnext.adrformacion.com © ADR Infor SL  
VICTOR TENA PALOMARES

## **Variables**

### **© ADR Infor SL**

viewnext.adrformacion.com © ADR Infor SL  
VICTOR TENA PALOMARES

viewnext.adrformacion.com © ADR Infor SL  
VICTOR TENA PALOMARES

# Indice

<b>Competencias y Resultados de Aprendizaje desarrollados en esta unidad</b>	<b>3</b>
<b>Variables.</b>	<b>4</b>
Objetivos.	4
Concepto de variable	4
Variables primitivas	5
Variables primitivas de tipo numérico	7
Variables primitivas de tipo lógico (booleanas)	8
Variables de tipo char (caracter)	8
Variables referenciadas	9
Proceso de creación de una variable	10
Nombre de variables	10
Declaración	10
Inicialización	11
Modificación del valor	11
Casting o transformaciones de tipo	12
Alcance de variables	15
Variables locales	15
Variables de instancia	16
Variables estáticas o de clase	18
Métodos estáticos o de clase	21
Estructuras básicas de programación	24
Expresiones	24
Operadores	25
Condicional if.. else	31
Condicional switch	36
Bucle For	38
Bucle For Extendido	40
Bucle while	41
Bucle do ... while	42
Sentencias break, return y continue	43
Bucles anidados	45
<b>Ejercicios</b>	<b>47</b>
Ejercicio 1. Ecuación de segundo grado	47
Lo necesario para comenzar.	47
Datos a mostrar por consola.	48
Ejercicio 2. Sintaxis.	48
Lo necesario para comenzar.	48
Ejercicio 3. Clase Math.	49
Lo necesario para comenzar.	49
Ejercicio 4. Enteros.	49
Lo necesario para comenzar	49
Datos a mostrar por consola.	50
<b>Recursos</b>	<b>51</b>
Enlaces de Interés	51
Glosario.	51

## Competencias y Resultados de Aprendizaje desarrollados en esta unidad

### Competencia:

Toma de contacto con el lenguaje Java

### Resultados de Aprendizaje:

- Escribir un programa en Java
- 

### Competencia:

Aprender a manejar las variables en Java.

### Resultados de Aprendizaje:

- Entender qué es una variable.
  - Utilizar una variable dentro de un programa. Declararla, inicializarla y modificar su valor.
  - Entender el alcance de las variables
- 

### Competencia:

Entender y usar correctamente las estructuras básicas que se utilizan en cualquier lenguaje de programación avanzado.

### Resultados de Aprendizaje:

- Entender las expresiones más comunes
- Entender los operadores más comunes
- Entender los bloques condicionales
- Entender los bucles

# Variables.

## Objetivos.

- Entender qué es una variable.
- Aprender a utilizar una variable, cómo declararla, inicializarla y modificar su valor.
- Entender las estructuras básicas que se utilizan en programación.

## Concepto de variable

Las variables son una de las características fundamentales de los lenguajes de programación, permiten **acceder a la memoria para almacenar y recuperar los datos** con los que nuestros programas van a trabajar. Son por tanto el **mecanismo** que los lenguajes de programación ponen a nuestra disposición **para acceder a la memoria**.

Se trata de un mecanismo de lo más sencillo, sólo tenemos que dar un nombre a nuestras variables, a partir de ese momento el compilador traducirá y almacenará de forma automática ese nombre en un acceso a memoria.

### Vocabulario: Variable

Una **variable** es un **contenedor de bits que representan a un valor**. Se emplean para **almacenar datos en memoria que pueden cambiar durante la ejecución de un programa**.

En función de los datos que almacenan se clasifican en dos tipos:

- **Variables primitivas:** almacenan datos numéricos, valores lógicos o caracteres.
- **Variables referenciadas:** asociadas a objetos o instancias de una clase. Por ejemplo, para almacenar cadenas de caracteres se empleará una variable referenciada asociada a la clase String, para almacenar información sobre la fecha actual, otra asociada a la clase Date, etc. Se estudiarán más adelante.

Además de estos dos tipos de variables se estudiarán los **arrays de variables primitivas y de variables referenciadas**.

**Vocabulario: Array**

Un **array** es un medio de guardar un conjunto de objetos de la misma clase.

```
int miVariableEntera=100;
```



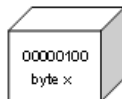
Creamos nuestras primeras variables

**Variables primitivas**

En el caso de variables primitivas, los bits **representan un número entero** que coincide con el valor de la variable, con lo que se va a trabajar a lo largo del programa.

Se tienen variables de tipo **byte** (utilizan 8 bits en memoria) que pueden almacenar números **enteros comprendidos entre -128 y 127**, de tipo **int** (utilizan 32 bits en memoria) para almacenar **enteros entre, aproximadamente, -2150 millones y 2150 millones**, de tipo **float** para **números decimales**, etc.

```
byte x=4;
```

**contenedor de bits**

Dichos bits representan el valor numérico 4. Este valor es el dato que se almacena en memoria y es con el que se trabajará durante la ejecución del programa.  
La x, en esta analogía, sería el nombre del contenedor, mientras que byte indicaría la forma del contenedor.

**Vocabulario: Variable primitiva**

Las variables primitivas representan valores de tipos primitivos como datos numéricos, valores lógicos o caracteres.

Hay varios tipos de datos que se deben tener en cuenta:

**Enteros**

Difieren en las precisiones y pueden ser positivos o negativos.

**Booleanos**

No son valores numéricos como tal, solo admite los valores true o false.

**Decimal simple**

Usa el código UNICODE, representa a cualquier carácter imprimible y ocupa 16 bits por cada carácter en lugar de los 8 habituales.

**Decimal doble**

Es la unidad de información de base utilizada en computación. Resulta equivalente a un conjunto ordenado de 8 bits.

En la siguiente tabla veremos un resumen de como se representan los diferentes tipos de datos en Java.

	Nombre	Tipo	Ocupa
<b>Tipos primitivos</b>	byte	Entero	1 byte
	short	Entero	2 bytes
	int	Entero	4 bytes
	long	Entero	8 bytes
	float	Decimal Simple	4 bytes
	double	Decimal Double	8 bytes
	char	Carácter simple	2 bytes
	boolean	Valor true o false	1 byte
<b>Tipos Referenciados</b>	Tipos de la biblioteca estándar Java	String (cadenas de texto) entre otros	
		Clases que Java tiene creadas para un uso común.	
	Tipos definidos por el usuario	Cualquier clase que se desarrolle	
	arrays	Elementos de tipo vector o matriz.	
		Se considera un objeto especial que no contiene métodos.	
	Tipos envoltorio	Byte	
		Short	
		Integer	
		Long	
		Float	
		Character	
		Boolean	

El lenguaje java es **case sensitive**, es decir, hace diferencia entre mayúsculas y minúsculas, las variables hola y Hola serían variables diferentes y ocuparían dos zonas diferentes en memoria. Si nos fijamos en la tabla podremos ver que los tipos primitivos (todos ellos empiezan con una letra minúscula) están 'duplicados' en los **tipos envoltorio**. Esto es porque Java tiene creadas clases que permiten utilizar los **valores primitivos** en **variables referenciadas**.

Vamos a ver como podemos manejar las variables de los tipos primitivos más utilizados:

## Variables primitivas de tipo numérico

### Vocabulario: Variable primitiva numérica

Son variables que **almacenan valores numéricos**. Pueden representar a números enteros o números reales.

Se pueden dividir en los siguientes tipos:

#### Enteros

##### Tipos de datos numéricos enteros

Tipo	Tamaño	Rango
byte	8 bits - complemento a 2	$-2^7$ a $2^7-1$
short	16 bits - complemento a 2	$-2^{15}$ a $2^{15}-1$
int	32 bits - complemento a 2	$-2^{31}$ a $2^{31}-1$
long	64 bits - complemento a 2	$-2^{63}$ a $2^{63}-1$

#### Reales

##### Tipos de datos numéricos reales

Tipo	Tamaño
float	32 bits - IEEE 754
double	64 bits - IEEE 754

El tipo de variable en que se almacena por defecto un número decimal es **double**.  
El valor por defecto asociado a cualquier variable real no inicializada es **0.0**.

## Variables primitivas de tipo lógico (booleanas)

### Vocabulario: Variable primitiva lógica

Son variables que almacenan dos posibles valores: **true** o **false**.

No se corresponden con ningún valor numérico aunque suele asociarse que el valor booleano del número **0** es **false** y el valor del número **1** es **true**.

```
boolean tienesCalor=true;
```

El valor por defecto asociado a cualquier variable booleana no inicializada es **false**

### Tipos de datos booleanos

Tipo	Tamaño	Rango
boolean	8 bits	true-false

## Variables de tipo char (caracter)

### Vocabulario: Variable primitiva char

Son variables que **almacenan caracteres individuales** (letra, número, signo ?, etc...).  
El carácter que se inicializa debe ir entre apóstrofes o comillas simples **'a'**.

El código de caracteres empleado por Java es **Unicode** y recoge los caracteres de prácticamente todos los idiomas importantes del mundo. Los caracteres Unicode del alfabeto occidental corresponden a los primeros 256 enteros; es decir van desde [0, 255].



A cada carácter le corresponde unívocamente un número entero perteneciente al intervalo [0, 65536] o a [0, 255] si se trabaja sólo con el alfabeto occidental. Por ejemplo, la letra ñ es el entero 164.

Más adelante se verá que el casting entre variables primitivas enteras y la variable char está permitido.

Asociadas a este tipo de variable se encuentran las **secuencias de escape**. Se emplean para representar caracteres especiales (por ejemplo, unas comillas dentro de una instrucción que exige una cadena entrecomillada) y caracteres no imprimibles (como el tabulador, salto de línea, etc).

Van precedidos de la contrabarra. Algunos de ellos se detallan en la tabla siguiente:

Carácter	Denominación
\n	salto de línea
\t	Tabulador
\"	Comillas dobles
\\	Contrabarra

El valor por defecto asociado a cualquier variable char no inicializada es `'\u0000'`

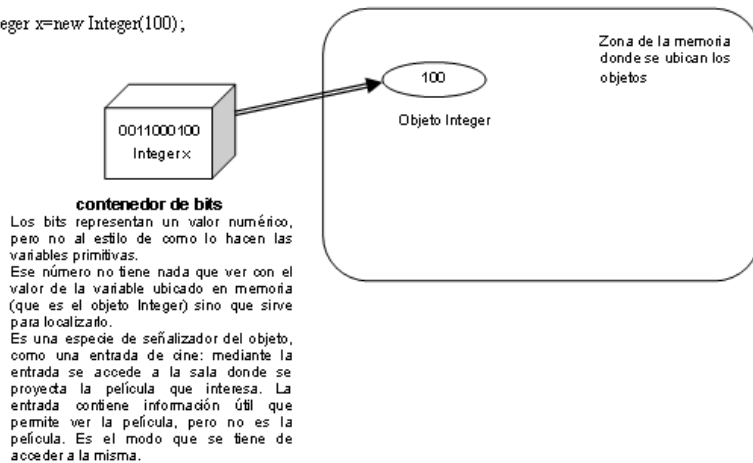
## Variables referenciadas

En el caso de variables referenciadas o asociadas a objetos, los bits **representan un puntero a una dirección de memoria** que permite acceder al valor de la variable, es decir, al objeto, pero **no es el valor u objeto en sí**.

### Vocabulario: Variable referenciada

Las variables referenciadas son aquellas que **almacenan la ubicación de variables primitivas, arrays de variables, o la ubicación de objetos definidos por clases**.

```
Integer x=new Integer(100);
```



El estudio y uso de estas variables se irá detallando a lo largo de todo el curso.

## Proceso de creación de una variable

Para crear una variable en Java hay que dar varios pasos:

### Nombre de variables

Un programa se refiere al valor de una variable por su nombre.

Todos los nombres empleados para hacer referencia a variables deben cumplir lo siguiente:

- Su **primer carácter** debe ser una **letra**, el símbolo del **subrayado** o el carácter dólar \$.
- No son válidos las palabras reservadas de Java.
- No se admiten espacios en blanco.
- Son **case-sensitive** (sensibles a mayúsculas).
- Si el nombre tiene más de dos palabras, **la primera letra de la primera palabra irá en minúscula, la primera letra de la segunda palabra en mayúscula**, igual con la tercera palabra (primera letra mayúscula) y así sucesivamente.
- Los nombres de variables deben ser **cortos y significativos**.
- La elección de un **nombre** de variable debe ser **nemotécnico**, es decir, pensado para que un lector casual al verla comprenda su uso.
- Se deben **evitar las variables de una sola letra**, excepto en variables temporales de corto uso. Nombres comunes para este tipo de variables son: i, j, k, m y n para enteros; c, d, y e para caracteres.

### Declaración

Para el lenguaje de programación Java es importante que **antes de usar una variable inicialmente se declare** la misma, es decir, que primero se debe definir cuál es su tipo (de los antes mencionados) para saber cuál es el espacio en la memoria que le debe asignar y cuál va a ser el significado de su contenido. Esto se hace en una sola sentencia colocando la palabra clave que designa el tipo de variable seguido del nombre de la variable y finalizando con un punto y coma.

```
public class EjemploDeclaracion {
    byte variableTipoByte; /* Las instrucciones y declaraciones finalizan siempre con ;*/
    short variableTipoShort;
    int variableTipoEntero;
    long variableTipoLong;
    float variableTipoFloat;
    double variableTipodouble;
    char variableTipoChar;
    boolean variableTipoBoolean;
} // Las declaraciones siempre finalizan con }
```

A este proceso se le llama **instanciar**. Crear una instancia de un tipo significa crear una variable de ese tipo de dato concreto.

## Inicialización

En Java es posible **declarar e inicializar** una variable **en un sólo paso**. Esto se hace en un renglón colocando la palabra clave que designa el tipo de variable, el nombre de la variable, un signo de igual, el valor que se le asignará a dicha variable recordando asignarle un valor válido y finalizando con un punto y coma.

```
public class EjemploInicializacion {
    byte variableTipoByte    = 10;
    short variableTipoShort  = 30000;
    int variableTipoEntero   = 225;
    long variableTipoLong    = 546831831321;
    float variableTipoFloat  = 4564565.4654874;
    double variableTipodouble = 564546.813227;
    char variableTipoChar    = 'A';
    boolean variableTipoBoolean = false;
}
```

## Modificación del valor

El valor de una variable puede ser modificado colocando el nombre de la variable seguido de un signo de igual, posteriormente el valor que se le desea asignar, recordando que debe ser un valor válido y finalizando con un punto y coma.

Cuando se modifica el valor de una variable se pueden presentar varios casos según el tipo de variable. Cuando la variable almacena un valor numérico, el valor a asignar **puede ser el resultado de una fórmula** que implique una operación entre literales y/o constantes o variables **respetando** que **los tipos de todas** las variables y las constantes **sean iguales**.

```
public class EjemploModificarValor {  
    variableTipoByte    = 100;  
    variableTipoShort   = 3500;  
    variableTipoEntero  = 223;  
    variableTipoLong    = 8500;  
    variableTipoFloat   = 86316546.54874;  
    variableTipodouble  = 212.364813227;  
    variableTipoChar    = '@';  
    variableTipoBoolean = true;  
}
```

Se puede comprobar que tenemos que usar las mismas sentencias para inicializar y para modificar el valor de una variable. La primera vez que se asigne el valor será la **inicialización** y las siguientes se tratará de la **modificación**.

## Casting o transformaciones de tipo

El casting es un procedimiento para **transformar una variable primitiva de un tipo a otro**, o **transformar un objeto de una clase a otra clase** siempre y cuando haya una relación de **herencia** entre ambas.

Dentro del casting de variables primitivas se distinguen dos clases

### Implícito

**No se necesita escribir código** para que se lleve a cabo. Ocurre cuando se realiza una conversión ancha (**widening casting**), es decir, cuando se coloca un valor pequeño en un contenedor grande.

```
int num1 = 100;  
long num2 = num1;
```

**Explícito**

**Sí es necesario escribir código.** Ocurre cuando se realiza una conversión estrecha (**narrowing casting**), es decir, cuando se coloca un valor grande en un contenedor pequeño. Son susceptibles de pérdida de datos.

```
int num1 = 100; // ocupa 4 bytes
short num2 = (short) num1;
/*num1 no encaja en un contenedor de 4 bytes. Hace falta casting explícito.*/
```

Se observa que antes de la variable a asignar se escribe el tipo al que se desea convertir dicha variable, esto se hace de manera explícita para tomar en cuenta las conversiones de tipos.

El manejo de este tipo de conversiones puede resultar un poco complicado si no se entiende bien por lo que vamos a explicar con una serie de ejemplos los casos más comunes:

**Ejemplo 1**

```
int num1 = 100; //ocupa 4 bytes
short num2 = (short)num1; /* num1 no encaja en un contenedor de 2 bytes. Hace falta casting explícito.*/
```

Si se sustituyera la primera línea `int num1=100` por `int num1=1000000`, el código compilaría bien, pero habría pérdida de datos, pues el 1000000 se sale del rango de `short [-32768, 32767]`. Al mostrar por consola el valor se obtendría un resultado incongruente.

**Ejemplo 2**

```
double num1 = 25.5; // ocupa 8 bytes
float num2 = (float)num1; /*num1 no encaja en un contenedor de 4 bytes. Hace falta casting explícito.*/
```

**Ejemplo 3**

Para que la línea

```
long num1 = 100000000000
```

compile debe hacerse un casting explícito a long

```
long num1 = 100000000000L ó long num1 = 100000000000L
```

pero no

```
long num1 = (long)100000000000 /* Provoca error de compilación*/
```

porque, en la línea anterior, 100000000000 es considerado int, mientras que en las de arriba, long.\*

**Ejemplo 4 (algo muy inusual)**

```
byte num1 = 10;
```

Dado que cualquier entero, por defecto, se almacena en un int (4 bytes), con la línea anterior se pretende colocar un valor grande (el int 10) en un contenedor pequeño (una primitiva de tipo byte con capacidad para 1 byte). Esto, según lo expuesto anteriormente, precisa de casting explícito.

```
byte num1 = (byte)10; //es lo que suele hacerse
```

Pero resulta que no hace falta ya que el compilador, cuando se trabaja con enteros, provoca un **casting implícito contranatura** y transforma automáticamente a byte el int 10. Ocurriría lo mismo si se trabaja con short o char.

Lo que pasa (y esto es lo que resulta extraño) es que no ocurre lo anterior con los números decimales, por eso, una línea como:

```
float num1 = 25.5;
```

provoca error de compilación. Recordar que cualquier decimal, por defecto, se almacena en un double (8 bytes) y que un tipo float tiene capacidad para 4 bytes. En los decimales el compilador no fuerza el casting implícito contranatura, de ahí que sea necesario un casting explícito a float para evitar errores de compilación.

Para finalizar con el casting entre primitivas, conviene tener en cuenta lo siguiente:

- No es posible realizar casting entre una variable primitiva booleana y cualquier otra variable primitiva.
- Sí es posible realizar casting entre una variable primitiva char y una variable primitiva que almacene enteros.

```
public class EjemploCastingChar{  
    public static void main(String args[]){  
        int num=164;  
        char letra=(char)num;  
        System.out.println(letra);  
        System.out.println((char)164);  
    }  
}
```



Vamos a realizar varios casting de variables primitivas

## Alcance de variables

El alcance de una variable indica la parte del programa donde puede utilizarse.

En base al alcance, las variables se clasifican en:

- Variables locales
- Variables de instancia

Durante el desarrollo del tema se irán viendo las características que definen a los diversos tipo de variables. Todos los códigos en c:\cursojava\tema2 o en el workspace correspondiente si se emplea Eclipse.

## Variables locales

Se declaran dentro de métodos o de instrucciones asociadas a bucles for, estructuras condicionales, etc.

Su alcance se restringe al código del método o de la instrucción.

No admiten modificadores de acceso salvo "**final**" y deben estar inicializadas antes de ser empleadas.

```

public class VariableLocal{
    public static void main(String args[]){
        VariableLocal vl=new VariableLocal();
        /**
         * La variable entera i es local pues está declarada e inicializada
         * dentro de un bucle for. Si se usa fuera del código del for se
         * producirá un error de compilación
         */
        for(int i=0;i<=5;i++){
            System.out.println(i);
            System.out.println("Repetición "+i);
        }
        vl.mostrarVariable();
    }
    void mostrarVariable(){
        //Línea que provoca error de compilación.
        //Acceso incorrecto a una variable local
        System.out.println(i+1);
    }
}

```

## Variables de instancia

Se declaran después de la clase y fuera de cualquier método.

Los valores que asumen para cada objeto constituyen el estado o **conjunto de atributos** del objeto.

Su **alcance** es **global**, es decir, las pueden utilizar directamente todos los métodos no estáticos de la clase.

Para acceder desde el método **main** o desde cualquier otro **método estático** a una variable de instancia es necesario crear un objeto de la clase.

Si no se inicializan explícitamente, asumen el valor nulo por defecto, una vez instanciada la clase.



```

public class SerHumano {
    //Declaración de tres variables de instancia
    String nombre;
    String colorOjos;
    int edad;
    void mostrarCaracteristicas(){
        System.out.println(nombre+" tiene "+edad+" años");
        System.out.println("Sus ojos son "+colorOjos);
    }
    void eresMayorEdad(){
        if(edad>=18){
            System.out.println(nombre+" es mayor de edad");
            System.out.println("Tiene "+edad+" años");
        }
        else{
            System.out.println(nombre+" es menor de edad");
            System.out.println("Tiene "+edad+" años");
        }
    }
}

public static void main(String args[]){
    /*Instanciar un objeto de la clase y asignar valor a sus variables
    de instancia*/
    SerHumano sh1=new SerHumano();
    /*
    * Las variables de instancia se inicializan mediante un objeto de
    * la clase. Sus valores constituyen el estado del objeto.
    * Si sólo se crea un objeto, resulta más cómodo inicializarlas
    * en la misma línea donde fueron declaradas:
    * String nombre="Jesus";
    * String colorOjos="azules";
    * int edad=28;
    */
    sh1.nombre="Jesus";
    sh1.colorOjos="azules";
    sh1.edad=28;
    sh1.mostrarCaracteristicas();
    sh1.eresMayorEdad();
    System.out.println("-----");
    //Otro objeto de la clase y asignación de otros valores
    SerHumano sh2=new SerHumano();
    sh2.nombre="Rebeca";
    sh2.colorOjos="verdes";
    sh2.edad=27;
    sh2.mostrarCaracteristicas();
    sh2.eresMayorEdad();
    System.out.println("-----");
    System.out.println("FIN DEL PROGRAMA");
}
}

```

```

<terminated> SerHumano [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (09/07/2016 13:07:58)
Jesus tiene 28 años
Sus ojos son azules
Jesus es mayor de edad
Tiene 28 años
-----
Rebeca tiene 27 años
Sus ojos son verdes
Rebeca es mayor de edad
Tiene 27 años
-----
FIN DEL PROGRAMA

```

Si las variables de instancia no se inicializan explícitamente, bien mediante un objeto de la clase, bien en la misma línea donde se declaran, qué ocurrirá?

Se mostrará el elemento nulo asociado a los valores iniciales de los diversos tipos de variables.

Así, en las variables primitivas enteras (byte, short, int y long) dicho elemento nulo es **0**, en las reales (float y double) es **0.0**, en las booleanas es **false** y en las referenciadas (asociadas a objetos de clases) es **null**.

```

<terminated> SerHumano [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (09/07/2016 13:15:55)
null tiene 0 años
Sus ojos son null
null es menor de edad
Tiene 0 años
-----
null tiene 0 años
Sus ojos son null
null es menor de edad
Tiene 0 años
-----
FIN DEL PROGRAMA

```

## Variables estáticas o de clase

Dentro de las variables de instancia hay otro tipo de variables que merecen ser estudiadas, estas son las **variables estáticas** o también llamadas **variables de clase**.

### Vocabulario: Variables estáticas

Son variables **propias** únicamente **de la clase** y no de los objetos que puedan crearse dentro de la misma, por lo tanto, **sus valores son compartidos** por todos los objetos de la clase.

Si van precedidas del modificador **static** se considera que es una **variable estática**.

Esta variable será única para todos los objetos de la clase y el compilador le asignará un único lugar en la memoria.

Si no van precedidas del modificador **static** el compilador asignará un lugar en la memoria cada vez que se instancie la variable, esto hará que pueda tener un valor diferente según el objeto de la clase que la esté utilizando.

Para invocar a una variable estática **no se necesita crear un objeto de la clase** en la que se define:

- Si se invoca desde la clase en la que se encuentra definido, basta con escribir su nombre.
- Si se le invoca desde una clase distinta, debe anteponerse a su nombre, el de la clase en la que se encuentra seguido del operador punto (.) **<NombreClase>.variableEstatica**

Suelen emplearse para definir variables comunes a todos los objetos de la clase.

También se les llama **atributos estáticos** de la clase.

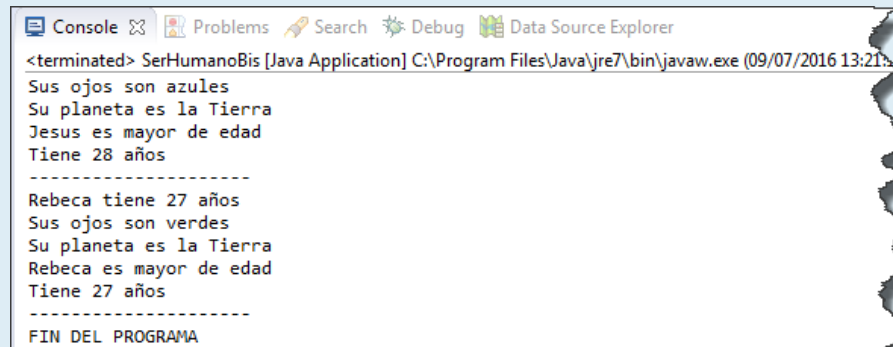
```

public class SerHumanoBis {
    String nombre;
    String colorOjos;
    int edad;
    /*
     * Declaración e inicialización de una variable de instancia estática
     * Tiene sentido declararla estática pues todos los objetos
     * de la clase, teniendo en cuenta que ésta modela a un ser humano,
     * habitan en el mismo planeta
     */
    static String planeta="Tierra";
    void mostrarCaracteristicas(){
        System.out.println(nombre+" tiene "+edad+" años");
        System.out.println("Sus ojos son "+colorOjos);
        System.out.println("Su planeta es la "+planeta);
    }
    void eresMayorEdad(){
        if(edad>=18){
            System.out.println(nombre+" es mayor de edad");
            System.out.println("Tiene "+edad+" años");
        }
        else{
            System.out.println(nombre+" es menor de edad");
            System.out.println("Tiene "+edad+" años");
        }
    }
}

public static void main(String args[]){
    SerHumanoBis shb1=new SerHumanoBis();
    shb1.nombre="Jesus";
    shb1.colorOjos="azules";
    shb1.edad=28;
    shb1.mostrarCaracteristicas();
    shb1.eresMayorEdad();

    System.out.println("-----");
    SerHumanoBis shb2=new SerHumanoBis();
    shb2.nombre="Rebeca";
    shb2.colorOjos="verdes";
    shb2.edad=27;
    shb2.mostrarCaracteristicas();
    shb2.eresMayorEdad();
    System.out.println("-----");
    System.out.println("FIN DEL PROGRAMA");
}
}

```



```

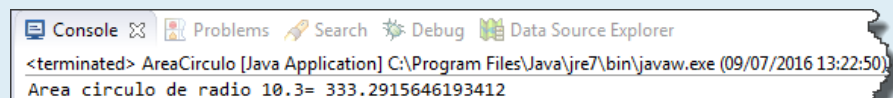
<terminated> SerHumanoBis [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (09/07/2016 13:21:...)
Sus ojos son azules
Su planeta es la Tierra
Jesus es mayor de edad
Tiene 28 años
-----
Rebeca tiene 27 años
Sus ojos son verdes
Su planeta es la Tierra
Rebeca es mayor de edad
Tiene 27 años
-----
FIN DEL PROGRAMA

```

```

public class AreaCirculo {
    /**
     * Declaración e inicialización de la variable de instancia estática
     * valorPi mediante la variable de campo estática PI de la clase
     * java.lang.Math de la API. Como PI no se encuentra en la clase
     * AreaCirculo, debe anteponerse el nombre de la clase en la que se
     * encuentra definida, es decir, Math.
     */
    static double valorPi=Math.PI;
    double radio=10.3;
    public static void main(String args[]){
        AreaCirculo ac=new AreaCirculo();
        /**
         * A radio se accede mediante un objeto de la clase ya que no es
         * estática, a valorPi directamente ya que es estática y está en la
         * clase desde la que se accede
         */
        double area=valorPi*ac.radio*ac.radio;
        System.out.println("Area circulo de radio "+ac.radio+"= "+area);
    }
}

```



```

<terminated> AreaCirculo [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (09/07/2016 13:22:50)
Area circulo de radio 10.3= 333.2915646193412

```

## Métodos estáticos o de clase

Igual que las variables, los métodos también pueden utilizar el modificado **static**. En ese caso se denominan **métodos estáticos** o **métodos de clase**.

Se cargan en memoria en tiempo de compilación y no a medida que se ejecutan las líneas de código del programa. Van precedidos del modificador static.

Para invocar a un método estático no se necesita crear un objeto de la clase en la que se define:

- Si se invoca desde la clase en la que se encuentra definido, basta con **escribir su nombre**.
- Si se le invoca desde una clase distinta, **debe anteponerse** a su nombre, **el de la clase** en la que se encuentra seguido del operador punto (.) <NombreClase>.metodoEstatico

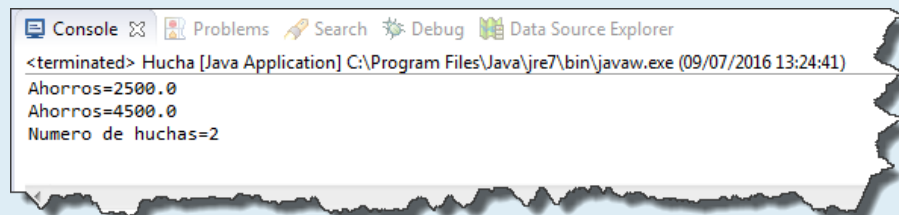
Suelen emplearse para realizar operaciones comunes a todos los objetos de la clase. No afectan a los estados de los mismos (a los valores de sus variables de instancia). Por ejemplo, si se necesita un método para contabilizar el número de objetos creados de una clase, tiene sentido que sea estático ya que su función (aumentar el valor de una variable entera) se realizaría independientemente del objeto empleado para invocarlo.

No conviene usar muchos métodos estáticos, pues si bien se **aumenta la rapidez** de ejecución, **se pierde flexibilidad**, no se hace un uso efectivo de la memoria y no se trabaja según los principios de la Programación Orientada a Objetos.

```

class Hucha{
    static int numHuchas=0;
    double ahorros=0.0;
    public static void main(String args[]){
        Hucha hucha1=new Hucha();
        contarHuchas();
        hucha1.ahorros=2500;
        hucha1.modificarAhorros();
        Hucha hucha2=new Hucha();
        contarHuchas();
        hucha2.ahorros=5000;
        hucha2.modificarAhorros();
        System.out.println("Numero de huchas="+numHuchas);
    }
    //La funcionalidad del método varía en función de si es invocado
    //por el objeto hucha1 o por hucha2.
    //No tendría sentido considerarlo estático.
    public void modificarAhorros(){
        if(ahorros>4000){
            ahorros=ahorros-0.1*ahorros;
        }
        System.out.println("Ahorros="+ahorros);
    }
    //La funcionalidad del método es la misma,
    //independientemente del objeto empleado para invocarlo.
    //Sí tiene sentido declararlo estático.
    public static void contarHuchas(){
        numHuchas++;
    }
}

```



Muchas clases de la API disponen de métodos estáticos.

Por ejemplo, la clase Math del paquete java.lang cuenta con multitud de ellos. Estos métodos se emplean para realizar operaciones matemáticas.

La clase Thread, del mismo paquete, cuenta con varios: uno que se emplea para retardar la ejecución de código es “void sleep(long retardo)”.

#### **Consultar la API.**

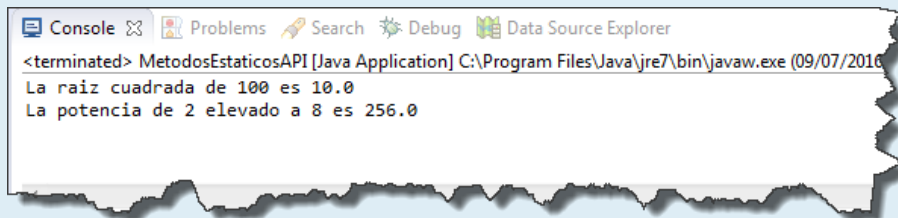
Lo importante de estos métodos es que para su utilización no es necesario instanciar un objeto de las clases en las que se encuentran ya que son estáticos.

```

public class MetodosEstaticosAPI{
    public static void main(String args[]){
        int num=100;
        System.out.println("La raiz cuadrada de "+num+" es "+Math.sqrt(num));

        //Bloque try ... catch. Se estudiará más adelante. A nivel de
        //ejecución no afecta.
        //Se introduce un retardo en la ejecución del código de 3 sg
        try{
            Thread.sleep(3000);
        }catch(InterruptedException e){}
        System.out.println("La potencia de 2 elevado a 8 es "+Math.pow(2,8));
    }
}

```



Variables locales y variables globales

## Estructuras básicas de programación

El estudio de esta sección del tema se recomienda a personas que no conocen las estructuras típicas de programación comunes a la mayoría de lenguajes, tales como los condicionales, los bucles for, los bucles while, las sentencias de ruptura break y continue, etc.

Primero se realiza una explicación de los operadores y expresiones que más frecuentemente se utilizan en cualquier lenguaje de programación y, a continuación, se explican las estructuras mencionadas anteriormente.

## Expresiones



Una expresión es una instrucción o sentencia que devuelve un valor llamado valor de retorno y que asigna a una variable un:

- Dato numérico
- Valor lógico
- Carácter
- Cadena de caracteres o dato textual

```
int x; // Expresión de definición de una variable.
x = 3; // Expresión de inicialización de una variable.
int x = 3; // Expresión que define e inicializa una variable.
```

```
int edad;
boolean mayorDeEdad;
mayorDeEdad=18>edad; // Expresión que devuelve un valor lógico.
```

## Operadores

Un operador permite relacionar **dos datos en una expresión** y evaluar el resultado de una operación.

Hay varios tipos de operadores:

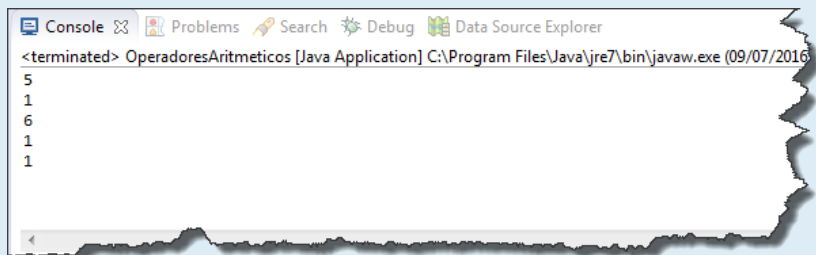
### Aritméticos

Manipulan datos numéricos. Los siguientes operadores forman parte de los operadores aritméticos:

- Suma +
- Resta −
- Producto \*
- División /
- Resto de la división entera o módulo %
- Incremento unitario ++
- Decremento unitario --
- Incremento de cien en cien +=100
- Decremento de cien en cien -=100

**Ejemplo: Operadores básicos**

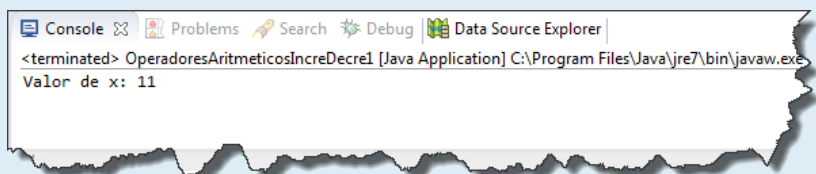
```
public class OperadoresAritmeticos {
    public static void main(String args[]){
        int x=3;
        int y=2;
        System.out.println(x+y);
        System.out.println(x-y);
        System.out.println(x*y);
        System.out.println(x/y);
        System.out.println(x%y);
    }
}
```



```
<terminated> OperadoresAritmeticos [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (09/07/2016)
5
1
6
1
1
```

**Ejemplo: Incrementar/Decrementar 1**

```
public class OperadoresAritmeticosIncreDecr1 {
    public static void main(String args[]){
        int x=10;
        x++; //++x; tendría el mismo efecto
        System.out.println("Valor de x: "+x);
    }
}
```



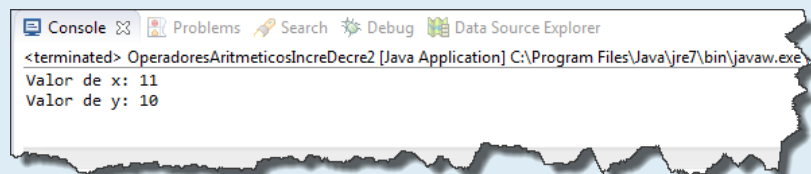
```
<terminated> OperadoresAritmeticosIncreDecr1 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe
Valor de x: 11
```

**Ejemplo: Incrementar/Decrementar 2**

```
public class OperadoresAritmeticosIncreDecre2{

public static void main(String args[]){
    int x=10;

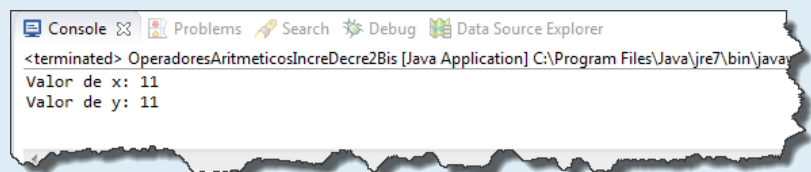
    //Primero se asigna el valor actual de la variable "x" a la
    //variable "y" y después se incrementa el valor de "x"
    int y=x++;
    System.out.println("Valor de x: "+x);
    System.out.println("Valor de y: "+y);
}
}
```



```
<terminated> OperadoresAritmeticosIncreDecre2 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe
Valor de x: 11
Valor de y: 10
```

```
public class OperadoresAritmeticosIncreDecre2Bis {
public static void main(String args[]){
    int x=10;

    //Primero se incrementa el valor actual de la variable "x" en una
    //unidad y después se asigna el valor incrementado a la variable "y"
    int y=++x;
    System.out.println("Valor de x: "+x);
    System.out.println("Valor de y: "+y);
}
}
```



```
<terminated> OperadoresAritmeticosIncreDecre2Bis [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe
Valor de x: 11
Valor de y: 11
```

**De asignación**

Transfieren datos de una variable a otra.

Hay también varios operadores que forman parte de este tipo:

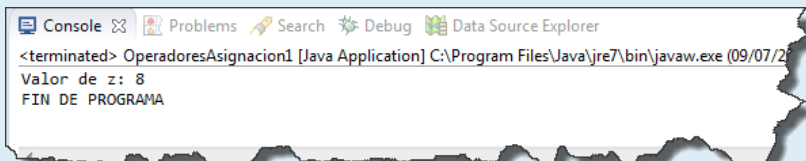
- **x+=y**       $x = x + y$
- **x-=y**       $x = x - y$

- $x*=y$        $x=x*y$
- $x/=y$        $x=x/y$
- $x\%=y$        $x=x\%y$  Es el módulo o resto de una división

### Ejemplo: Asignación 1

```
public class OperadoresAsignacion1 {
    public static void main(String args[]){
        int x=3;
        int y=5;

        //z se inicializa con la suma de los valores de las variables x e y
        int z=x+=y;
        System.out.println("Valor de z: "+z);
        System.out.println("FIN DE PROGRAMA");
    }
}
```

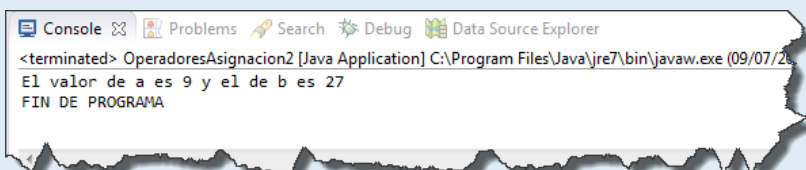


Console Problems Search Debug Data Source Explorer  
 <terminated> OperadoresAsignacion1 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (09/07/2016)  
 Valor de z: 8  
 FIN DE PROGRAMA

### Ejemplo: Asignación 2

```
public class OperadoresAsignacion2 {

    public static void main(String args[]){
        int a=6;
        int b=3;
        a+=b; //a=6+3=9 y b=3
        b*=a; // b=3*9=27 y a=9
        System.out.println("El valor de a es "+a+" y el de b es "+b);
        System.out.println("FIN DE PROGRAMA");
    }
}
```



Console Problems Search Debug Data Source Explorer  
 <terminated> OperadoresAsignacion2 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (09/07/2016)  
 El valor de a es 9 y el de b es 27  
 FIN DE PROGRAMA

## De comparación

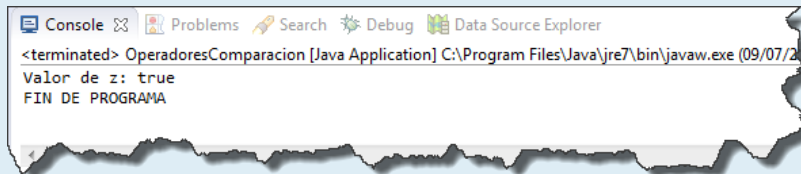
Comparan dos datos y devuelven un valor booleano.

Son los siguientes:

- == igual (no confundir con el = de asignación)
- != distinto
- < mayor que
- < menor que
- >= mayor o igual que
- <= menor o igual que

### Ejemplo: Operadores de comparación

```
public class OperadoresComparacion{
    public static void main(String args[]){
        int x=5;
        int y=8;
        boolean z=x!=y;
        System.out.println("Valor de z: "+z);
        System.out.println("FIN DE PROGRAMA");
    }
}
```



## Lógicos

Concatenan expresiones lógicas con objeto de evaluar si es cierto o falso el conjunto de las expresiones.

Son los siguientes:

- && (Y lógico): si todas las expresiones lógicas que se evalúan son verdaderas se devuelve un valor booleano true; si alguna es falsa, devuelve false.
- || (O lógico): si alguna de las expresiones lógicas que se evalúa es verdadera, se devuelve un valor booleano true; si todas son falsas, devuelve false.
- ! (NOT): invierte el valor de una expresión booleana.

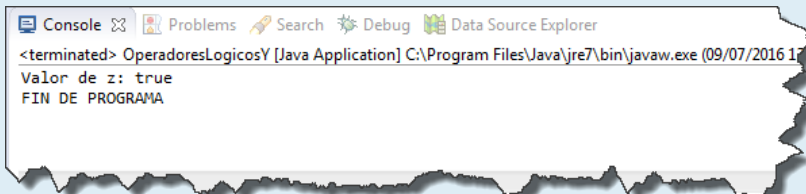
**Ejemplo: Operador lógico Y**

```

public class OperadoresLogicosY{
public static void main(String args[]){
    int x=10;
    int y=20;

    //x>5 devuelve true
    //y<50 devuelve true
    //Ambas expresiones unidas por && hacen que z almacene un valor true
    //No son necesarios los paréntesis aunque conviene ponerlos para
    //interpretar mejor el código
    boolean z=(x>5) && (y<50);
    System.out.println("Valor de z: "+z);
    System.out.println("FIN DE PROGRAMA");
}
}

```

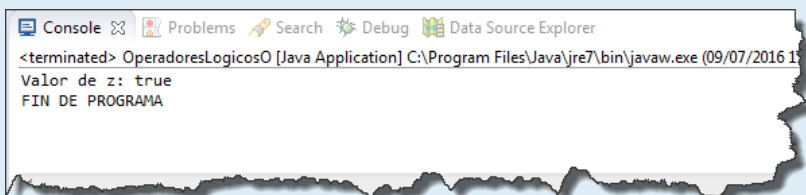
**Ejemplo: Operador lógico O**

```

public class OperadoresLogicosO{
public static void main(String args[]){
    int x=10;
    int y=20;

    //x>20 devuelve false
    //y<50 devuelve true
    //Ambas expresiones unidas por || hacen que z almacene true
    boolean z=(x>20) || (y<50);
    System.out.println("Valor de z: "+z);
    System.out.println("FIN DE PROGRAMA");
}
}

```



## Condicional if .. else

No todos los problemas pueden resolverse empleando estructuras secuenciales. Cuando hay que tomar una decisión aparecen las estructuras condicionales.

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

- ¿Elijo la carrera A o la carrera B?
- ¿Me pongo este pantalón?
- Para ir al trabajo, ¿elijo el camino A o el camino B?
- Al cursar una carrera, ¿elijo el turno mañana, tarde o noche?

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

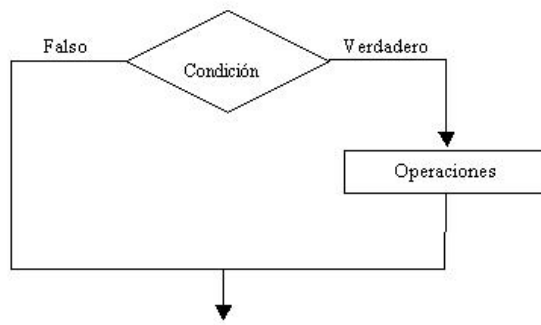
La palabra clave **if** indica que estamos en presencia de una estructura condicional; seguidamente disponemos la condición entre paréntesis. Por último encerrada entre llaves las instrucciones de la rama del verdadero. Es necesario que las instrucciones a ejecutar en caso que la condición sea verdadera estén encerradas entre llaves { }, con ellas marcamos el comienzo y el fin del bloque del verdadero.

Si el bloque de código del if tiene una sola línea de código no es necesario utilizar llave, aunque si se escribe, no da error. Lo mismo ocurre con el bloque de código alternativo. Si no se escribe el else opcional y no se cumple la condición del if, el programa continúa ejecutándose pasando a las siguientes líneas de código.

Las estructuras condicionales se dividen en dos tipos:

### Estructura condicional simple

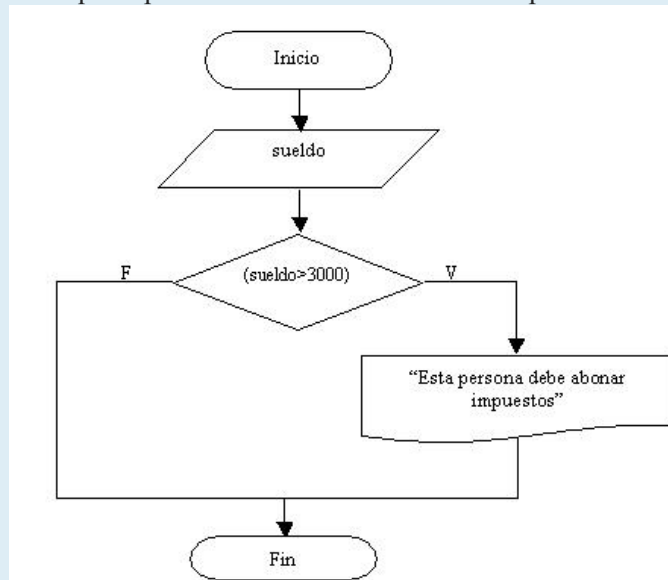
Cuando se presenta la elección tenemos la **opción de realizar una actividad o no realizar ninguna**. Se puede observar en la imagen de abajo que el rombo representa la condición, hay dos opciones que se pueden tomar. Si la condición da verdadera se sigue el camino del verdadero, o sea el de la derecha, si la condición da falsa se sigue el camino de la izquierda.



..com © ADR Infor SL  
TENA PALOMARES

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

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





```
import java.util.Scanner;

public class EstructuraCondicionalSimple1 {

    public static void main(String[] ar) {

        Scanner teclado=new Scanner(System.in);

        float sueldo;

        System.out.print("Ingrese el sueldo:");

        sueldo=teclado.nextFloat();

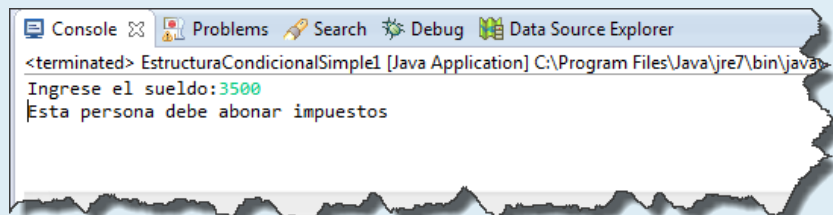
        if (sueldo>3000) {

            System.out.println("Esta persona debe abonar impuestos");

        }

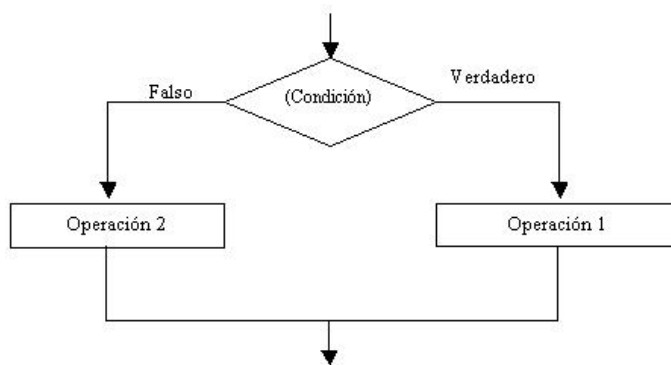
    }

}
```



### Estructura condicional compuesta

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

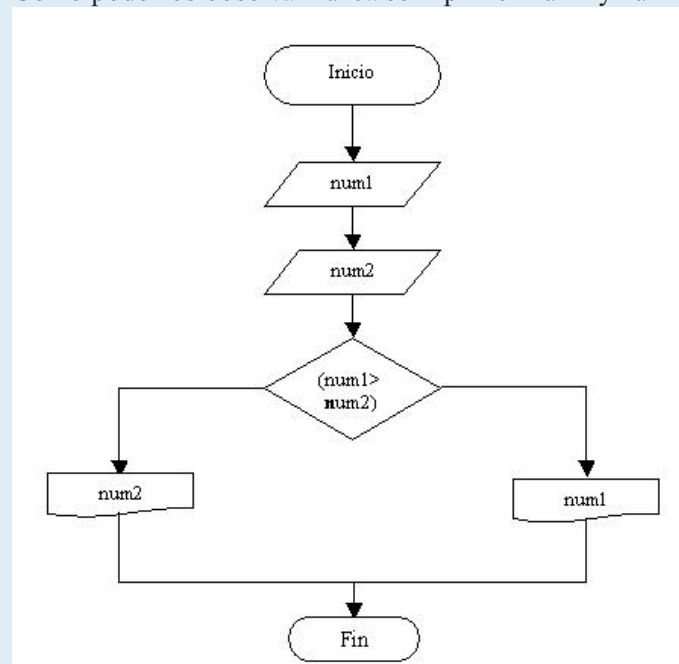


**Realizar un programa que solicite ingresar dos números distintos y muestre por pantalla el mayor de ellos.**

Se hace la entrada de num1 y num2 por teclado.

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

Como podemos observar nunca se imprimen num1 y num2 simultáneamente.



```

import java.util.Scanner;

public class EstructuraCondicionalCompuesta1 {

    public static void main(String[] ar) {

        Scanner teclado=new Scanner(System.in);

        int num1,num2;

        System.out.print("Ingrese primer valor:");

        num1=teclado.nextInt();

        System.out.print("Ingrese segundo valor:");

        num2=teclado.nextInt();

        if (num1>num2) {

            System.out.print(num1);

        } else {

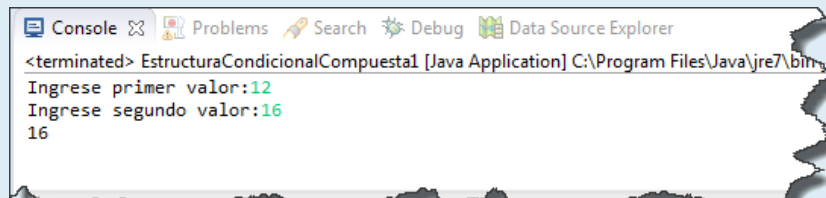
            System.out.print(num2);

        }

    }

}

```



â€œ

La estructura condicional compuesta también permite la concatenación de varias sentencias condicionales de modo que podemos tener un bloque como el siguiente:

```

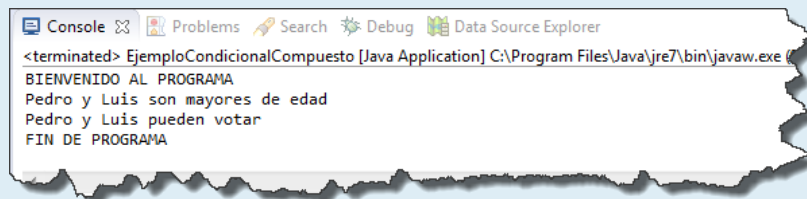
if (condición 1){
    Bloque de código 1
}else if (condición 2){
    Bloque de código 2
}else{
    Bloque de código alternativo
}

```

```

public class EjemploCondicionalCompuesto {
    public static void main(String args[]) {
        int edadPedro=25;
        int edadLuis=22;
        System.out.println("BIENVENIDO AL PROGRAMA");
        if (edadPedro>=18 && edadLuis>=18) {
            System.out.println("Pedro y Luis son mayores de edad");
            System.out.println("Pedro y Luis pueden votar");
        } else if (edadPedro>=18 && edadLuis<18) {
            System.out.println("Pedro es mayor de edad, Luis no");
            System.out.println("Pedro puede votar, Luis no");
        } else if (edadPedro<18 && edadLuis>=18) {
            System.out.println("Luis es mayor de edad, Pedro no");
            System.out.println("Luis puede votar, Pedro no");
        } else {
            System.out.println("Pedro y Luis no son mayores de edad");
            System.out.println("Ni Pedro ni Luis pueden votar");
        }
        System.out.println("FIN DE PROGRAMA");
    }
}

```



Se emplea cuando se quiere **ejecutar un bloque de código siempre y cuando se cumplan las condiciones**.

Estas condiciones **devuelven** un valor booleano, es decir, **true o false**.

## Condicional switch

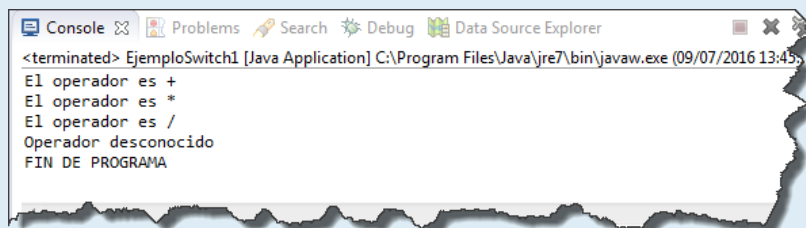
Es similar a la estructura if()...else if()...else if()...else, pero con ciertas **limitaciones**:

- Sólo admite una condición. No son válidos los operadores && y ||.
- En esa condición se presupone que el operador de relación es la igualdad (==).
- La variable asociada a la condición sólo puede ser de tipo byte, short, int o char, ninguna otra es válida.

Se ejecutan todas las instrucciones a partir del segundo case (es el que coincide con la variable operador). Si no hay ningún case coincidente, se ejecuta el código asociado a default.

**Ejemplo: Switch1**

```
public class EjemploSwitch1 {  
    public static void main(String args[]) {  
        char operador='+';  
        switch(operador) {  
            case '-':  
                System.out.println("El operador es -");  
            case '+':  
                System.out.println("El operador es +");  
            case '*':  
                System.out.println("El operador es *");  
            case '/':  
                System.out.println("El operador es /");  
            default:  
                System.out.println("Operador desconocido");  
        }  
        System.out.println("FIN DE PROGRAMA");  
    }  
}
```

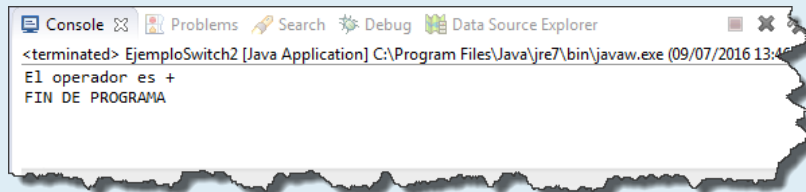


No se puede repetir ningún case con el mismo valor. Se produce error de compilación

Si se quiere que el programa, cuando encuentre el primer case coincidente con el valor de la variable evaluada, salga del switch, debe agregarse a cada case la instrucción break del siguiente modo:

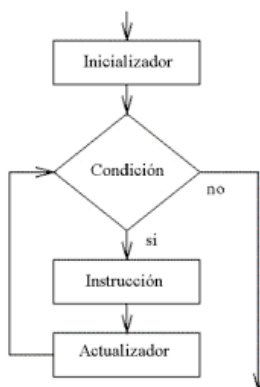
### Ejemplo: Switch 2

```
public class EjemploSwitch2{
    public static void main(String args[]){
        char operador='+';
        switch(operador){
            case '-':
                System.out.println("El operador es -");
                break;
            case '+': System.out.println("El operador es +");
                break;
            case '*':
                System.out.println("El operador es *");
                break;
            case '/':
                System.out.println("El operador es /");
                break;
            default:
                System.out.println("Operador desconocido");
        }
        System.out.println("FIN DE PROGRAMA");
    }
}
```



## Bucle For

El bucle For se utiliza para repetir una o más instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando sabemos seguro el número de veces que queremos que se ejecute.



El número de veces que se repetirá depende del cumplimiento de una condición fijada por el programador.

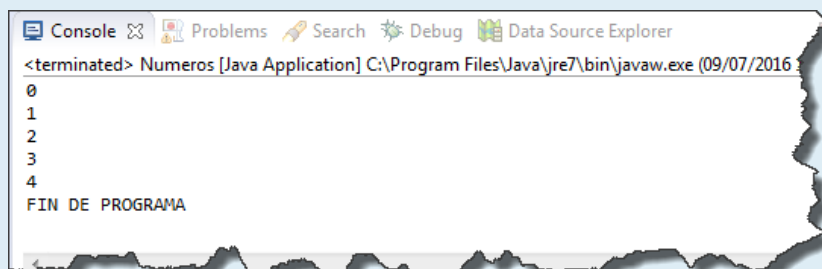
### Ortografía: Sintaxis

```
for(variable de control; condición; actualización de la variable de control){  
    Bloque de código que se ejecutará en cada iteración hasta que se  
    incumpla la condición.  
}
```

Se quiere desarrollar un programa que muestre por consola los números del 0 al 4. Con los conceptos aprendidos hasta ahora solamente podríamos hacer lo siguiente:

### Ejemplo: Sin bucle for

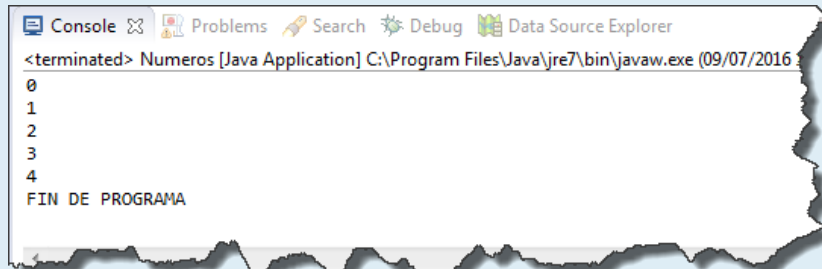
```
public class Numeros {  
    public static void main(String args[]) {  
        int x1=0;  
        int x2=1;  
        int x3=2;  
        int x4=3;  
        int x5=4;  
        System.out.println(x1);  
        System.out.println(x2);  
        System.out.println(x3);  
        System.out.println(x4);  
        System.out.println(x5);  
        System.out.println("FIN DE PROGRAMA");  
    }  
}
```



Ahora, con este nuevo bucle, podríamos hacer lo mismo de manera más sencilla:

**Ejemplo: Con bucle for**

```
public class BucleFor{
    public static void main(String args[]){
        for(int i=0;i<5;i++)
            System.out.println(i);
        System.out.println("FIN DE PROGRAMA");
    }
}
```

**Bucle For Extendido**

En las últimas versiones de Java se introdujo una nueva forma de uso del for, a la que se denomina **for extendido** o **for each**.

Esta forma de uso del for, que ya existía en otros lenguajes, facilita el recorrido de objetos existentes en una colección sin necesidad de definir el número de elementos a recorrer.

**Ortografía: Sintaxis**

```
for (TipoARecorrer nombreVariableTemporal : nombreDeLaColección
    Instrucciones
}
```

Para saber si un for es un for extendido o un for normal hemos de fijarnos en la sintaxis que se emplea.

En el siguiente ejemplo vamos a ver como podemos utilizar el bucle for extendido para realizar la tarea anterior.

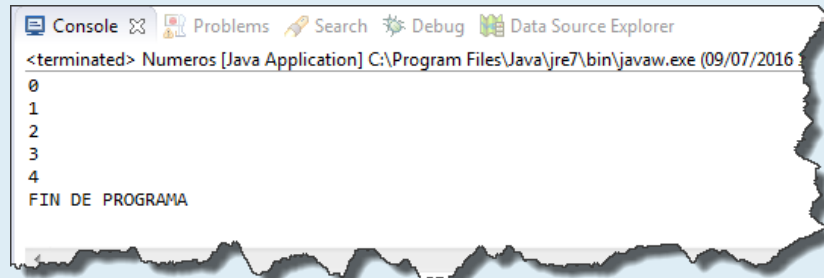
Para ello crearemos una variable de tipo array que contendrá los números que queremos mostrar y que se llamará **numeros**, después recorreremos la variable almacenando cada objeto (de tipo int) en la variable **numero** y por último mostraremos el valor de esa variable en pantalla.



```

public class ForExtendido {
    static int[] numeros = {0,1,2,3,4};
    public static void main (String args0[]){
        for(int numero:numeros){
            System.out.println(numero);
        }
    }
}

```



El for extendido tiene algunas ventajas y algunos inconvenientes. No siempre es aconsejable utilizarlo.

El for extendido es una herramienta muy útil cuando tenemos que realizar recorridos completos de colecciones, por lo que lo usaremos en numerosas ocasiones antes que bucles for o while que nos obligan a estar pendientes de más cuestiones (por ejemplo en este caso con el for, de llevar un contador, llamar en cada iteración a un método, etc.).

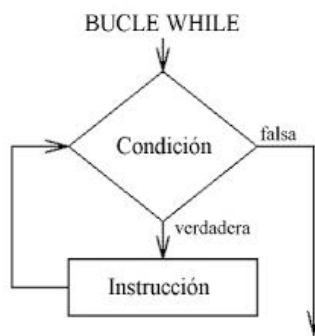
## Bucle while

La sentencia **while** es la más sencilla de las estructuras de iteración. La iteración continuará hasta que su condición sea falsa.

Sintaxis:

while ( condición ) sentencia ;

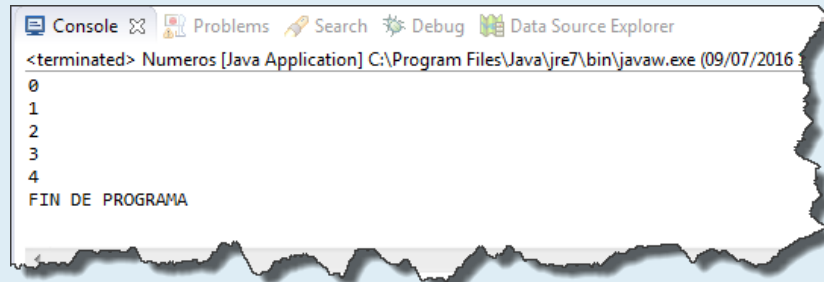
- La condición tiene que tomar un valor booleano (verdadero o falso).
- Si este valor es verdadero, se ejecutará la sentencia.
- Concluida esta acción se vuelve a evaluar la condición.
- Proseguirán los ciclos hasta que la condición no cambie a falso.



Esta es una estructura de iteración **preprueba**, es decir primero se evalúa la condición antes de realizar cualquier acción. Si de entrada la condición es falsa nunca ejecutará el conjunto de sentencias.

```

public class BucleWhile{
    public static void main(String args[]){
        int i = 0;
        while(i<5){
            System.out.println(i);
            i++;
        }
        System.out.println("FIN DE PROGRAMA");
    }
}
  
```



## Bucle do ... while

Es casi igual al anterior: la diferencia se basa en que un bucle do...while se ejecuta, al menos una vez, el bloque de código encerrado dentro del bucle, independientemente del valor booleano que devuelva la condición del while.

### Ortografía: Sintaxis

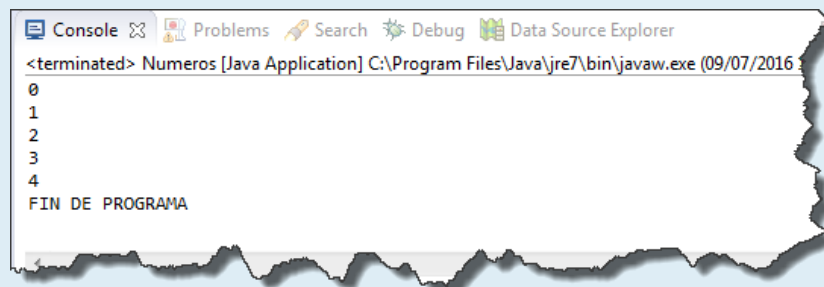
```

do{
    bloque de código a ejecutar en cada iteración
}while(condición booleana)
  
```

```

public class EjemploDoWhile{
    public static void main(String args[]){
        int i=0;
        do{
            System.out.println(i);
            i++;
        }
        while(i<=5);
        System.out.println("FIN DEL PROGRAMA");
    }
}

```



## Sentencias break, return y continue

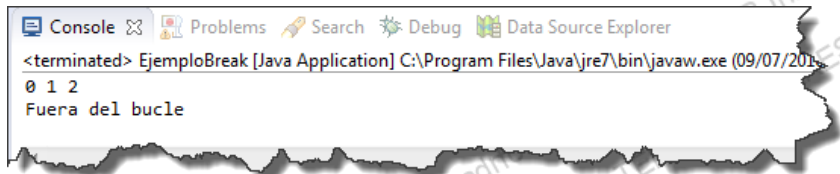
### Break

Aplicable dentro de bucles y estructuras condicionales de tipo switch. Su ejecución provoca que se salga del bucle o etiqueta que se esté ejecutando sin finalizar el resto de las sentencias asociadas al mismo.

```

public class EjemploBreak{
    public static void main(String args[]){
        for(int i=0;i<=10;i++){
            if(i==3) //Se sale del bucle en el momento en que i llega a 3
                break;
            System.out.print(i+" ");
        }
        System.out.println("Fuera del bucle");
    }
}

```



```

Console Problems Search Debug Data Source Explorer
<terminated> EjemploBreak [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (09/07/201
0 1 2
Fuera del bucle

```

Se sale del bucle en el momento en que *i* llega a 3. Si se sustituye `break` por `return`, se trunca la ejecución del método `main`

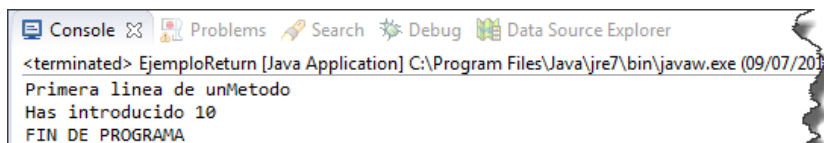
## Return

Se utiliza para truncar la ejecución de un método o para almacenar el dato que devuelven. Puede emplearse para truncar bucles cuando no haya más código después del bucle

```

public class EjemploReturn{
    public static void main(String args[]){
        EjemploReturn er=new EjemploReturn();
        er.unMetodo(10);
        System.out.println("FIN DE PROGRAMA");
    }
    public void unMetodo(int num){
        System.out.println("Primera linea de unMetodo");
        if(num==10){
            System.out.println("Has introducido 10"); //Implica que se sale del método
            return;
        }
        System.out.println("Fin de linea de unMetodo");
    }
}

```



```

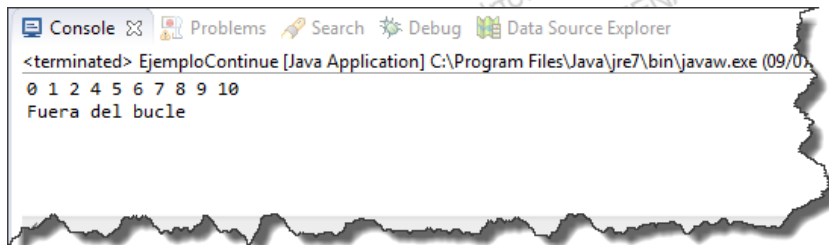
Console Problems Search Debug Data Source Explorer
<terminated> EjemploReturn [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (09/07/201
Primera linea de unMetodo
Has introducido 10
FIN DE PROGRAMA

```

## Continue

Aplicable sólo a bucles. Finaliza la iteración  $i$  que en ese momento se está ejecutando (significa que no ejecuta el resto de sentencias hasta el final del bucle) y vuelve a analizar la condición del bucle, pero con la siguiente iteración  $i+1$ .

```
public class EjemploContinue{
    public static void main(String args[]){
        for(int i=0;i<=10;i++){
            if(i==3)
                continue;
            System.out.print(i+" ");
        }
        System.out.println("\nFuera del bucle");
    }
}
```

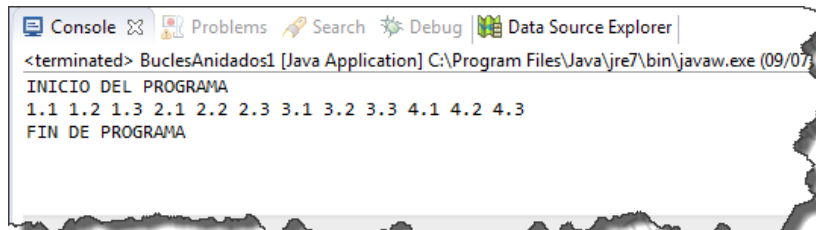


## Bucles anidados

Es uno o más bucles for dentro de otro. Se pueden poner los que se consideren necesarios.

```
public class BuclesAnidados1 {
    public static void main(String args[]){
        System.out.println("INICIO DEL PROGRAMA");
        for(int i=1;i<=4;i++){
            for(int j=1;j<=3;j++){
                System.out.print(i+"."+j+" ");
            }
            System.out.println("\nFIN DE PROGRAMA");
        }
    }
}
```

## Variables



The screenshot shows a Java IDE console window with the following text:

```
<terminated> BuclesAnidados1 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (09/07/2014)
INICIO DEL PROGRAMA
1.1 1.2 1.3 2.1 2.2 2.3 3.1 3.2 3.3 4.1 4.2 4.3
FIN DE PROGRAMA
```

# Ejercicios

## Ejercicio 1. Ecuación de segundo grado

20

Resolver una ecuación de segundo grado y realizar una serie de cálculos con sus soluciones reales.

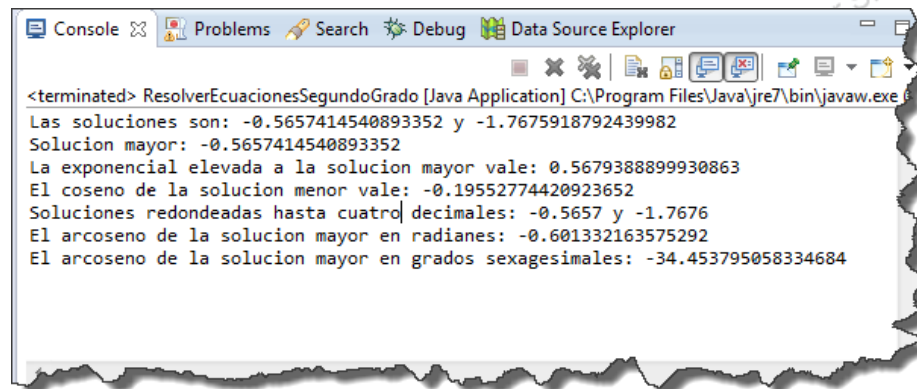
### Lo necesario para comenzar.

Se aporta el siguiente código comentado para que se tomado como punto de partida:

```
public class ResolverEcuacionesSegundoGrado {

    double a=1.5;
    double b=3.5;
    double c=1.5;
    double discriminante=Math.pow(b, 2)-4*a*c;
    double s1, s2;
    public static void main(String args[]){
        /* El método main se encargará de crear un objeto de la clase
        * y llamar al método resolucion()
        */
    }
    void resolucion(){
        /* Este método se encargará de mostrar por consola las soluciones
        * de la ecuación y de llamar al método calculitos() cuando la
        * ecuación tenga dos soluciones reales. Si la ecuación tiene una
        * sola solución real o soluciones complejas se mostrará un mensaje
        * por consola indicándolo no se llamará al método calculitos()
        */
    }
    void calculitos(double num1, double num2){
        double solMayor;
        double solMenor;
        /* Este método, mediante métodos estáticos de la clase Math,
        * inicializará las variables locales solMenor y solMayor en base
        * a los argumentos num1 y num2.
        * Luego mostrará por consola:
        * 1) La solución mayor.
        * 2) La exponencial elevada a la solución mayor
        * 3) El coseno de la solución menor
        * 4) Las soluciones redondeadas hasta cuatro decimales
        * 5) El arcoseno de la solución mayor en radianes
        * 6) El arcoseno de la solución mayor en grados sexagesimales
        */
    }
}
```

## Datos a mostrar por consola.



## Ejercicio 2. Sintaxis.

15

A) Dado el siguiente código, averiguar si va a producirse error de compilación.

Si consideráis que se produce dicho error, indicar la causa.

Si consideráis que la compilación se realiza con éxito, indicar qué se mostrará por consola al ejecutarlo.

### Lo necesario para comenzar.

```
public class MetodosEstaticosEjercicio1{
    static String nombre="Carlos";
    static void dimeNombre(){
        String mote="cucu";
        System.out.println("Tu nombre es "+nombre+" y tu mote es "+mote);
    }
    static int mostrarEdad(int anNacimiento){
        return 2003-anNacimiento;
    }
    public static void main(String args[]){
        dimeNombre();
        System.out.println(nombre+" tiene "+mostrarEdad(1960)+" años");
        System.out.println("FIN DEL PROGRAMA");
    }
}
```



B) Si se elimina el modificador **static** del método **mostrarEdad(..)**, ¿qué modificaciones deben hacerse en el código para que se muestre por consola lo mismo?

## Ejercicio 3. Clase Math.

15

¿Qué se mostrará por consola al compilar y ejecutar el siguiente código?

### Lo necesario para comenzar.

```
public class EjemplosMatemáticos {
    public static void main (String args[]){
        System.out.println(Math.ceil(10.8));
        System.out.println(Math.ceil(1.8956478));
        System.out.println(Math.ceil(-5.96));
        System.out.println(Math.ceil(-0.9));
        System.out.println(Math.floor(10.8));
        System.out.println(Math.floor(1.8956478));
        System.out.println(Math.floor(-5.96));
        System.out.println(Math.floor(-0.9));
        System.out.println(Math rint(10.2));
        System.out.println(Math rint(10.8));
        System.out.println(Math.round(10.2));
        System.out.println(Math.round(10.8));
        System.out.println(Math rint(3.891));
        System.out.println(Math rint(3.891*100)/100);
    }
}
```

## Ejercicio 4. Enteros.

20

Realizar un código que convierta un número entero positivo almacenado en una variable de tipo int en su correspondiente valor expresado en el sistema hexadecimal, octal y binario.

### Lo necesario para comenzar

```
public class SistemasNumeracion {
    public static void main (String args[]){
        int num=47;
    }
}
```

## Datos a mostrar por consola.



```
<terminated> SistemasNumeracion [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (1)
Numero 47 en sistema decimal: 47
Numero 47 en hexadecimal: 2f
Numero 47 en octal: 57
Numero 47 en binario: 101111
```

# Recursos

## Enlaces de Interés



[http://www.javahispano.org/antiguo\\_javahispano\\_org/2001/11/1/documentar-con-javadoc.html](http://www.javahispano.org/antiguo_javahispano_org/2001/11/1/documentar-con-javadoc.html)

[http://www.javahispano.org/antiguo\\_javahispano\\_org/2001/11/1/documentar-con-javadoc.html](http://www.javahispano.org/antiguo_javahispano_org/2001/11/1/documentar-con-javadoc.html)  
*Documentar con javadoc*



<http://www.eclipse.org/>

Descargar IDE

## Glosario.

- **Case-sensitive:** Es un término en inglés que significa que en la interpretación de los textos se distingue entre mayúsculas y minúsculas. Por ejemplo, en lenguaje Java la variable nombre es diferente a la variable Nombre, por lo que podrían existir las dos en el mismo ámbito. Lo contrario sería case insensitive, que indica que no se hace diferencia entre mayúsculas y minúsculas. Por ejemplo, si introduces en un buscador de internet la palabra Java obtendrás los mismos resultados que si introduces la palabra java o la palabra JAVA.
- **Crear un objeto de la clase:** También se denominará instanciar una clase o crear una instancia de la clase.
- **Instancia:** Una instancia (en inglés, instance) es la particularización, realización específica u ocurrencia de una determinada clase, entidad (modelo entidad-relación) o prototipo. En general, cuando se ejecuta un programa en un computador, se dice que éste se instancia.
- **Modificadores de acceso:** Restringen el acceso a los diferentes objetos (clases, atributos, métodos,...) que componen una aplicación. Esto permite manejar la seguridad de las aplicaciones a un nivel más alto.
- **Números enteros:** almacenan números que no tienen punto decimal. Pueden ser positivos, negativos o el cero.
- **Números reales:** almacenan números muy grandes que poseen parte entera y parte decimal.