



Java Básico

La sintaxis

Copyright

- Copyright (c) 2004
José M. Ordax
- Este documento puede ser distribuido solo bajo los términos y condiciones de la Licencia de Documentación de javaHispano v1.0 o posterior.
- La última versión se encuentra en
<http://www.javahispano.org/licencias/>

Comentarios

Existen tres formas distintas de escribir los comentarios:

// Comentario en una línea.

/* Comentario de una o más líneas */

/** Comentario de documentación, utilizado por la herramienta javadoc.exe */

Puntos y coma, bloques y espacios en blanco.

Una sentencia es una línea simple de código terminada en un punto y coma:

System.out.println("Hola");

Un bloque es un conjunto de sentencias agrupadas entre llaves ({ }):

```
while(true)
{
    x = y + 1;
    x = x + 1;
}
```

Puntos y coma, bloques y espacios en blanco.

Los bloques pueden estar anidados.

```
while(true)
{
    x = y + 1;
    if(x<0)
    {
        x = x + 1;
    }
}
```

Java permite los espacios en blanco entre elementos de código fuente.

Identificadores

Son los nombres unívocos que se le dan a las clases, métodos y variables.

Hay que tener presente las siguientes reglas:

Deben empezar por una letra, subrayado (_) o dólar (\$).

Después del primer carácter pueden usar números.

Distinguen las mayúsculas y minúsculas.

Nunca pueden coincidir con una 'keyword'.

Keywords

boolean	byte	char	double	float
int	long	short	public	private
protected	abstract	final	native	static
synchronized	transient	volatile	if	else
do	while	switch	case	default
for	break	continue	assert	class
extends	implements	import	instanceof	interface
new	package	super	this	catch
finally	try	throw	throws	return
void	null	enum		

Ejemplos de Identificadores



Estos identificadores serían válidos:



identificador



nombreUsuario



nombre_usuario



_sys_var2



\$cambio



if2

Variables

- Una variable es un contenedor de datos identificado mediante un nombre (identificador).
- Dicho identificador se utilizará para referenciar el dato que contiene.
- Toda variable debe llevar asociado un tipo que describe el tipo de dato que guarda.
- Por tanto, una variable tiene:
 - Un tipo.
 - Un identificador.
 - Un dato (o valor).

Declaración de variables

- Es la sentencia mediante la cual se define una variable, asignándole un tipo y un identificador:
 - tipo identificador;*
 - int** contador;
- Adicionalmente se le puede asignar un valor inicial mediante una asignación:
 - tipo identificador = valor;*
 - int** contador = 10;
- Si no se le asigna un valor, se inicializará con el valor por defecto para ese tipo.

Tipos de dato

- En Java existen dos tipos de datos genéricos:
 - Tipos primitivos.
 - Tipos complejos: clases.
- Existen ocho tipos de datos primitivos clasificados en cuatro grupos diferentes:
 - Lógico: boolean.
 - Carácter: char.
 - Números enteros: byte, short, int y long.
 - Números reales: double y float.


Tipo de dato lógico

- La 'keyword' es: boolean.
- Sus posibles valores son:
 - true
 - false
- Su valor por defecto es:
 - false
- Ejemplos:
 - boolean** switch1 = **true**;
 - boolean** switch2;

Tipo de dato carácter

- La 'keyword' es: char.
- Representa un carácter UNICODE.
- Su tamaño es: 16 bits (2 bytes).
- Sus posibles valores son:
 - Un carácter entre comillas simples: 'a'.
 - Un carácter especial con \ por delante: '\n', '\t', etc.
 - Un código UNICODE: '\uxxxx' (donde xxxx es un valor en hexadecimal).

Tipo de dato carácter

- Su valor por defecto es:
 - '\u0000' -> nul.
 - Ejemplos:
 - char** letra1 = 'a';
 - char** letra2 = '\n';
 - char** letra3 = '\u0041';
 - char** letra4;
 - Existe un tipo complejo para las cadenas de caracteres: la clase String.
-  Soporta UNICODE 4.0 que define algunos caracteres que no caben en 16 bits por lo que se necesita un int para representarlos (o dos char dentro de un String).

Tipos de datos enteros

- Las 'keywords' son: byte, short, int y long.
- Sus tamaños son:
 - byte: 8 bits (1 byte), por tanto: -128 a 127.
 - short: 16 bits (2 bytes), por tanto: -32768 a 32767
 - int: 32 bits (4 bytes), por tanto: -2147483648 a 2147483647
 - long: 64 bits (8 bytes), por tanto: *-enorme a enorme*
- Sus posibles valores son:
 - Un valor decimal entero: 2 (por defecto int) o 2L (long).
 - Un valor octal: 077.
 - Un valor hexadecimal: 0xBAAC

Tipos de datos enteros

- Su valor por defecto es:
 - 0 (cero)
- Ejemplos:
 - byte** unByte = 12;
 - short** unShort;
 - int** unInt = -199;
 - int** otroInt = 065;
 - long** unLong = 2; (o **long** unLong = 2L;)
 - long** otroLong = 0xABCD;

Tipos de datos reales

- Las 'keywords' son: float y double.
- Sus tamaños son:
 - float: 32 bits (4 bytes). Su precisión varía según plataforma.
 - double: 64 bits (8 bytes). Su precisión también varía.
- Sus posibles valores son:
 - Un valor decimal entero: 2 (por defecto int).
 - Un valor decimal real: 0.17 o 6.02E23 (por defecto double).
 - Un valor decimal real: 0.17F o 0.17D (redundante).

Tipos de datos reales

- Su valor por defecto es:
 - 0.0 (cero)
- Ejemplos:
 - float** unFloat = 0.17F;
 - double** unDouble;
 - double** otroDouble = -12.01E30;

Tipo de dato complejo

- La 'keyword' es el nombre de la clase del objeto que va a contener la variable.
- Posibles valores:
 - Referencias a objetos (o instancias) en memoria.
- Su valor por defecto es:
 - null
- Ejemplos:
 - `String unString = new String("Hola");`
 - `String otroString;`

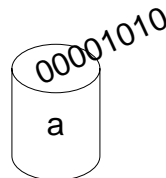
Tipo de dato enumeración

- La 'keyword' es: enum.
- Se trata de un tipo de dato complejo algo especial.
- Implementa una clase que tiene un atributo que puede tomar varios valores y solo esos.
- Ejemplo:
 - `enum Semaforo { VERDE, AMBAR, ROJO }`
- Veremos como trabajar con este nuevo tipo en el capítulo dedicado a J2SE 5.0

Variables primitivas vs. complejas

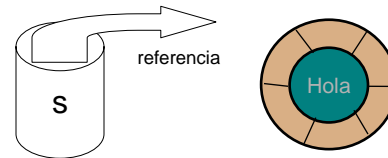
- Una variable de tipo primitivo contiene el dato directamente:

byte a = 10;



- Una variable de tipo complejo contiene una referencia (puntero) a la zona de memoria donde está el objeto:

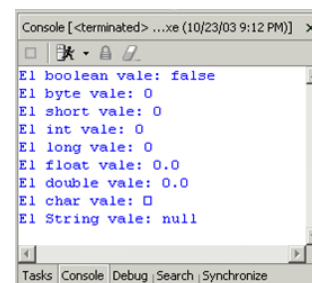
String s = **new** String("Hola");



```
public class VariablesTest1
{
    static boolean unBoolean;
    static byte unByte;
    static short unShort;
    static int unInt;
    static long unLong;
    static float unFloat;
    static double unDouble;
    static char unChar;
    static String unString;
```

```
    public static void main(String[] args)
    {
        System.out.println("El boolean vale: " + unBoolean);
        System.out.println("El byte vale: " + unByte);
        System.out.println("El short vale: " + unShort);
        System.out.println("El int vale: " + unInt);
        System.out.println("El long vale: " + unLong);
        System.out.println("El float vale: " + unFloat);
        System.out.println("El double vale: " + unDouble);
        System.out.println("El char vale: " + unChar);
        System.out.println("El String vale: " + unString);
    }
}
```

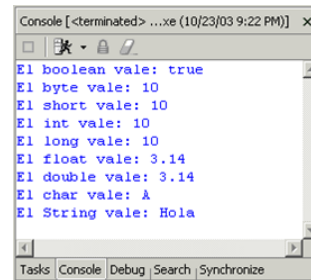
Ejemplo



Ejemplo

```
public class VariablesTest2
{
    public static void main(String[] args)
    {
        boolean unBoolean = true;
        byte unByte = 10;
        short unShort = 10;
        int unInt = 10;
        long unLong = 10;
        float unFloat = 3.14F;
        double unDouble = 3.14;
        char unChar = 'A';
        String unString = new String("Hola");

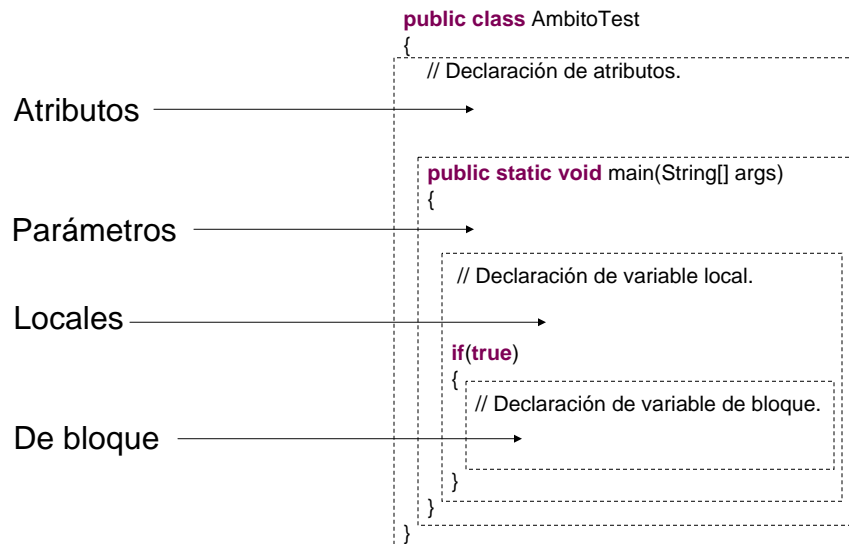
        System.out.println("El boolean vale: " + unBoolean);
        System.out.println("El byte vale: " + unByte);
        System.out.println("El short vale: " + unShort);
        System.out.println("El int vale: " + unInt);
        System.out.println("El long vale: " + unLong);
        System.out.println("El float vale: " + unFloat);
        System.out.println("El double vale: " + unDouble);
        System.out.println("El char vale: " + unChar);
        System.out.println("El String vale: " + unString);
    }
}
```



Ámbito de las variables

- El ámbito de una variable es la zona de código donde se puede referenciar dicha variable a través de su identificador.
- El lugar de definición de una variable establece su ámbito.
- Ámbitos:
 - Atributos (o variables miembro).
 - Parámetros de método.
 - Variables locales: siempre hay que inicializarlas.
 - Variables de bloque: siempre hay que inicializarlas.

Ámbito de las variables



Ejemplo

```
public class AmbitoTest2
{
    public static void main(String[] args)
    {
        if(true)
        {
            int i = 12;
        }
        System.out.println("El valor de i es: " + i);
    }
}
```

The screenshot shows a window titled "Tasks (1 item)" with a table of tasks. The first task indicates a compilation error.

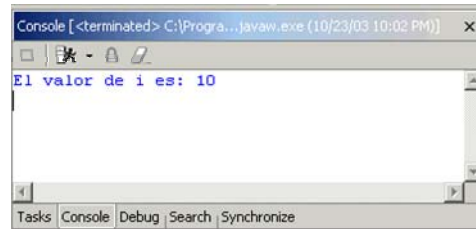
✓	!	Description	Resource	In Folder	Location
✗		i cannot be resolved	AmbitoTest2.java	Ejemplos Universidad	line 9

Tasks Console Debug Search Synchronize

Ejemplo

```
public class AmbitoTest3
{
    static int i = 5;

    public static void main(String[] args)
    {
        int i = 10;
        System.out.println("El valor de i es: " + i);
    }
}
```



Conversiones entre tipos

- Existen cuatro entornos de conversión en Java:
 - Promoción aritmética. (ej: short a int a float).
 - Asignación. (ej: **long** l = 42;).
 - Llamada a métodos con parámetros. (ej: f(**long** p) -> f(5)).
 - Casting. (ej: **int** i = (**int**)42L).
- Las conversiones implícitas se resuelven en tiempo de compilación.
- El *upcasting* se realizan implícitamente.
- El *downcasting* se realizan explícitamente y se resuelve en tiempo de ejecución.

Ejercicio



Identificar que sentencias son correctas y cuáles no:

1. `int x = 34.5;`
2. `boolean boo = x;`
3. `int g = 17;`
4. `int y = g;`
5. `y = y + 10;`
6. `short s;`
7. `s = y;`
8. `byte b = 3;`
9. `byte v = b;`
10. `short n = 12;`
11. `v = n;`
12. `byte k = 128;`
13. `int p = 3 * g + y;`

Ejercicio (solución)

1. `int x = 34.5;` -> `int x = (int)34.5;`
2. `boolean boo = x;` -> no hay solución.
3. `int g = 17;`
4. `int y = g;`
5. `y = y + 10;`
6. `short s;`
7. `s = y;` -> `s = (short)y;`
8. `byte b = 3;`
9. `byte v = b;`
10. `short n = 12;`
11. `v = n;` -> `v = (byte)n;`
12. `byte k = 128;` -> `byte k = (byte)128;`
13. `int p = 3 * g + y;`

Operadores

- Los operadores realizan funciones sobre uno, dos o tres operandos. Por tanto tenemos:
 - Operadores unarios: pueden ser prefijos o postfijos.
op operador ó *operador op* (Ejemplo: contador++).
 - Operadores binarios.
operador op operador (Ejemplo: contador + 1).
 - Operadores ternarios.
operador ? operador : operador
- Los operadores siempre devuelven un valor que depende del operador y del tipo de los operandos.

Operadores

- Los operadores se pueden dividir en las siguientes categorías:
 - Aritméticos.
 - Relacionales.
 - Condicionales.
 - De desplazamiento.
 - Lógicos.
 - De asignación.
 - Otros.

Operadores aritméticos



Tenemos los siguientes operadores aritméticos:



+ : suma dos operandos ($op1 + op2$).

Nota: en el caso de Strings concatena.



- : resta dos operandos ($op1 - op2$).



* : multiplica dos operandos ($op1 * op2$).



/ : divide dos operandos ($op1 / op2$).



% : calcula el resto de la división ($op1 \% op2$).

Operadores aritméticos



¿De qué tipo es el valor que devuelven?:



long: cuando ninguno de los operandos es float o double y hay al menos uno que es long.



int: cuando ninguno de los operandos es float, double o long.



double: cuando al menos hay uno de los operandos es double.



float: cuando ninguno de los operandos es double y hay al menos uno que es float.

Operadores aritméticos



También existen operadores aritméticos unarios:



+op: convierten a op en int en caso de que fuese byte, short o char.



-op: cambia el signo a op.



++op: incrementa op en 1 (evaluando op después de incrementarse).



op++: incrementa op en 1 (evaluando op antes de incrementarse)



--op: decrementa op en 1 (evaluando op después de decrementarse).



op--: decrementa op en 1 (evaluando op antes de decrementarse)

Ejemplo

```
public class OperadoresUnariosTest
{
    public static void main(String[] args)
    {
        int x = 0;
        int y = 0;
        y = ++x;
        System.out.println("y vale: " + y + ", x vale: " + x);
        y = x++;
        System.out.println("y vale: " + y + ", x vale: " + x);
    }
}
```



Operadores relacionales



Tenemos los siguientes operadores relacionales:



>: compara si un operando es mayor que otro ($op1 > op2$).



<: compara si un operando es menor que otro ($op1 < op2$).



==: compara si un operando es igual que otro ($op1 == op2$).



!=: compara si un operando es distinto que otro ($op1 != op2$).



>=: compara si un operando es mayor o igual que otro ($op1 >= op2$).



<=: compara si un operando es menor o igual que otro ($op1 <= op2$).

Operadores condicionales



Suelen combinarse con los relacionales para crear expresiones mas complejas.



Tenemos los siguientes operadores condicionales:



&&: AND lógico. Chequea si ambos operandos son verdaderos ($op1 \&\& op2$).



||: OR lógico. Chequea si uno de los dos operandos es verdadero ($op1 || op2$).



!: NOT lógico. Niega al operador ($!op$).

Operadores de desplazamiento



Tenemos los siguientes operadores:



>>: desplaza los bits del primer operando hacia la derecha tantas veces como indique el segundo operando (op1 >> op2).



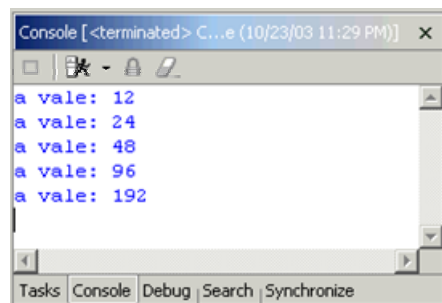
<<: desplaza los bits del primer operando hacia la izquierda tantas veces como indique el segundo operando (op1 << op2).



>>>: desplaza los bits del primer operando hacia la derecha tantas veces como indique el segundo operando pero sin signo (op1 >>> op2).

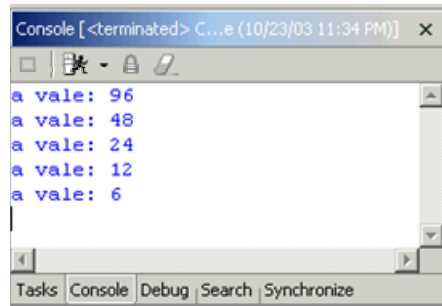
Ejemplo

```
public class Multiplicador
{
    public static void main(String[] args)
    {
        int a = 6;
        a = a << 1;
        System.out.println("a vale: " + a);
        a = a << 1;
        System.out.println("a vale: " + a);
        a = a << 1;
        System.out.println("a vale: " + a);
        a = a << 1;
        System.out.println("a vale: " + a);
        a = a << 1;
        System.out.println("a vale: " + a);
    }
}
```



Ejemplo

```
public class Dividor
{
    public static void main(String[] args)
    {
        int a = 192;
        a = a >> 1;
        System.out.println("a vale: " + a);
        a = a >> 1;
        System.out.println("a vale: " + a);
        a = a >> 1;
        System.out.println("a vale: " + a);
        a = a >> 1;
        System.out.println("a vale: " + a);
        a = a >> 1;
        System.out.println("a vale: " + a);
    }
}
```



Operadores lógicos



Tenemos los siguientes operadores lógicos:

&: AND lógico a nivel de bit (op1 & op2).

op1	op2	resultado
0	0	0
0	1	0
1	0	0
1	1	1

Operadores lógicos

○ |: OR lógico a nivel de bit ($op1 \mid op2$).

op1	op2	resultado
0	0	0
0	1	1
1	0	1
1	1	1

Operadores lógicos






○ ^: XOR lógico a nivel de bit ($op1 \wedge op2$).

op1	op2	resultado
0	0	0
0	1	1
1	0	1
1	1	0

○ ~: complemento a nivel de bit ($\sim op1$).








Operadores de asignación

 Tenemos los siguientes operadores de asignación:


-  = : guarda el valor del segundo operando en el primero (op1 = op2).
-  += : guarda la suma de los dos operandos en el primero (op1 += op2).
-  -= : guarda la resta de los dos operandos en el primero (op1 -= op2).
-  *= : guarda la multiplicación de los dos operandos en el primero (op1 *= op2).
-  /=, %=, &=, |=, ^=, <<=, >>=, >>>=

Otros operadores

 Existen otros operadores en Java como:

-  ?: : se trata de una abreviatura de la estructura if-then-else (op1?op2:op3).
-  [] : utilizado para declarar, crear y acceder a arrays.
-  . : utilizado para acceder a atributos y métodos de objetos.
-  (parámetros) : utilizado para pasar parámetros a un método.
-  (tipo) : utilizado para realizar castings (conversiones).
-  new : utilizado para crear objetos nuevos.
-  instanceof : chequea si el primer operando es una instancia del segundo operando.

Sentencias de control de flujo

- Sin las sentencias de control de flujo, el código Java se ejecutaría linealmente desde la primera línea hasta la última.
- Existen cuatro tipos de sentencias:
 - Bucles: while, do-while, for y for/in. 
 - Bifurcaciones: if-then-else y switch-case.
 - Gestión de excepciones: try-catch-finally y throw.
 - De ruptura: break, continue, label: y return.

Sentencias while y do-while

- La sentencia *while* se utiliza para ejecutar continuamente un bloque de código mientras que la condición del *while* sea *true*.

```
while(expresión)
{
    sentencias;
}
```
- La sentencia *do-while* es parecida a la sentencia *while* pero asegura que como mínimo el bloque de código se ejecuta una vez.

```
do
{
    sentencias;
}
while(expresión);
```


Sentencia for



La sentencia *for* facilita la ejecución de un bloque de código un número determinado de veces.



```
for(inicialización; terminación; incremento)
{
    sentencias;
}
```



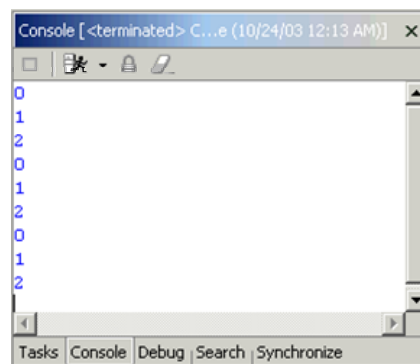
Nota: las variables definidas en la sentencia de inicialización son locales al bloque. Por tanto dejan de existir una vez se haya terminado el bucle.

```
public class Bucles
{
    public static void main(String[] args)
    {
        int cont1 = 0;
        while(cont1 < 3)
        {
            System.out.println(cont1);
            cont1++;
        }

        int cont2 = 0;
        do
        {
            System.out.println(cont2);
            cont2++;
        }
        while(cont2 < 3);

        for(int cont3 = 0; cont3 < 3; cont3++)
        {
            System.out.println(cont3);
        }
    }
}
```

Ejemplo



Sentencia for/in

- Esta nueva sentencia del J2SE 5.0 nos facilita la iteración por los elementos de cualquier tipo de colección: arrays, listas, etc...

- ```
for(inicialización: colección) Nota: Se usa ":" en vez de "=".
{
 sentencias;
}
```

- Ejemplo:

- ```
public void listar(int[] param)  
{  
    for(int i: param)  
        System.out.println(i);  
}
```

Sentencia for/in

- Básicamente, se trata de una simplificación a la hora de codificar.

- Es decir, al final, el compilador convierte el código en una sentencia *for* convencional:

- ```
public void listar(int[] param)
{
 for(int i=0; i<param.length; i++)
 System.out.println(param[i]);
}
```

- Veremos como trabajar con este nuevo tipo de sentencia en el capítulo dedicado a J2SE 5.0

## Sentencia if-then-else



La sentencia *if-then-else* permite elegir qué bloque de código ejecutar entre dos posibilidades.

```
if(expresión)
{
 sentencias;
}
```

```
if(expresión)
{
 sentencias;
}
else
{
 sentencias;
}
```

```
if(expresión)
{
 sentencias;
}
else if(expresión)
{
 sentencias;
}
else
{
 sentencias;
}
```

## Sentencia switch



La sentencia *switch* es un caso particular de la sentencia *if-then-else if-else*. Evalúa una expresión del tipo *int* o que pueda ser convertida a *int* de forma implícita.

```
switch(intExpresión)
{
 case intExpresión:
 sentencias;
 break;

 default:
 sentencias;
}
```

## Sentencias de ruptura


- Break: sirve para detener la ejecución tanto de los bucles como de la sentencia *switch*. Y por tanto saltar a la siguiente línea de código después del bucle o *switch*.
- Continue: sirve para detener la ejecución del bloque de código de un bucle y volver a evaluar la condición de este.
- Return: sirve para finalizar la ejecución de un método.

## Ejercicio

- Identificar si este código compila bien. Si no compila solucionarlo. Si compila decir cuál sería la salida.


```
public class Temp
{
 public static void main(String[] args)
 {
 int x = 1;
 while(x<10)
 {
 if(x>3)
 {
 System.out.println("Hola");
 }
 }
 }
}
```

## Ejercicio (solución)

-  El código compila bien. Pero entra en un bucle infinito. Habría que modificarlo con la línea azul y saldría la palabra “Hola” siete veces por pantalla.

```
public class Temp
{
 public static void main(String[] args)
 {
 int x = 1;
 while(x<10)
 {
 x = x + 1;
 if(x>3)
 {
 System.out.println("Hola");
 }
 }
 }
}
```

## Ejercicio

-  Identificar si este código compila bien. Si no compila solucionarlo. Si compila decir cuál sería la salida.

```
public class Temp
{
 public static void main(String[] args)
 {
 int x = 5;
 while(x>1)
 {
 x = x - 1;
 if(x<3)
 {
 System.out.println("Hola");
 }
 }
 }
}
```

## Ejercicio (solución)

- Compila y saldría la palabra “Hola” dos veces por pantalla.

## Ejercicio

- Al siguiente programa Java le falta un trozo de código.

```
public class Temp
{
 public static void main(String[] args)
 {
 int x = 0;
 int y = 0;
 while(x<5)
 {

???

 System.out.print(x + " " + y + " ");
 x = x + 1;
 }
 }
}
```

## Ejercicio



Seleccionar para cada trozo de código de la izquierda, la salida por pantalla al ejecutar el programa anterior con ese trozo de código.

```
y = x - y;
```

```
y = y + x;
```

```
y = y + 2;
```

```
if(y < 4)
```

```
 y = y - 1;
```

```
x = x + 1;
```

```
y = y + x;
```

```
if(y < 5)
```

```
{
```

```
 x = x + 1;
```

```
 if(y < 3)
```

```
 x = x - 1;
```

```
}
```

```
y = y + 2;
```

```
2 2 4 6
```

```
1 1 3 4 5 9
```

```
0 2 1 4 2 6 3 8
```

```
0 2 1 4 3 6 4 8
```

```
0 0 1 1 2 1 3 2 4 2
```

```
1 1 2 1 3 2 4 2 5 3
```

```
0 0 1 1 2 3 3 6 4 10
```

```
0 1 1 2 2 4 3 6 4 8
```

## Ejercicio (solución)

```
y = x - y;
```

```
y = y + x;
```

```
y = y + 2;
```

```
if(y < 4)
```

```
 y = y - 1;
```

```
x = x + 1;
```

```
y = y + x;
```

```
if(y < 5)
```

```
{
```

```
 x = x + 1;
```

```
 if(y < 3)
```

```
 x = x - 1;
```

```
}
```

```
y = y + 2;
```

```
2 2 4 6
```

```
1 1 3 4 5 9
```

```
0 2 1 4 2 6 3 8
```

```
0 2 1 4 3 6 4 8
```

```
0 0 1 1 2 1 3 2 4 2
```

```
1 1 2 1 3 2 4 2 5 3
```

```
0 0 1 1 2 3 3 6 4 10
```

```
0 1 1 2 2 4 3 6 4 8
```

# Bibliografía



## Head First Java

Kathy Sierra y Bert Bates.  
O'Reilly



## Learning Java (2<sup>nd</sup> edition)

Patrick Niemeyer y Jonathan Knudsen.  
O'Reilly.



## Thinking in Java (3<sup>rd</sup> edition)

Bruce Eckel.  
Prentice Hall.



## The Java tutorial

<http://java.sun.com/docs/books/tutorial/>