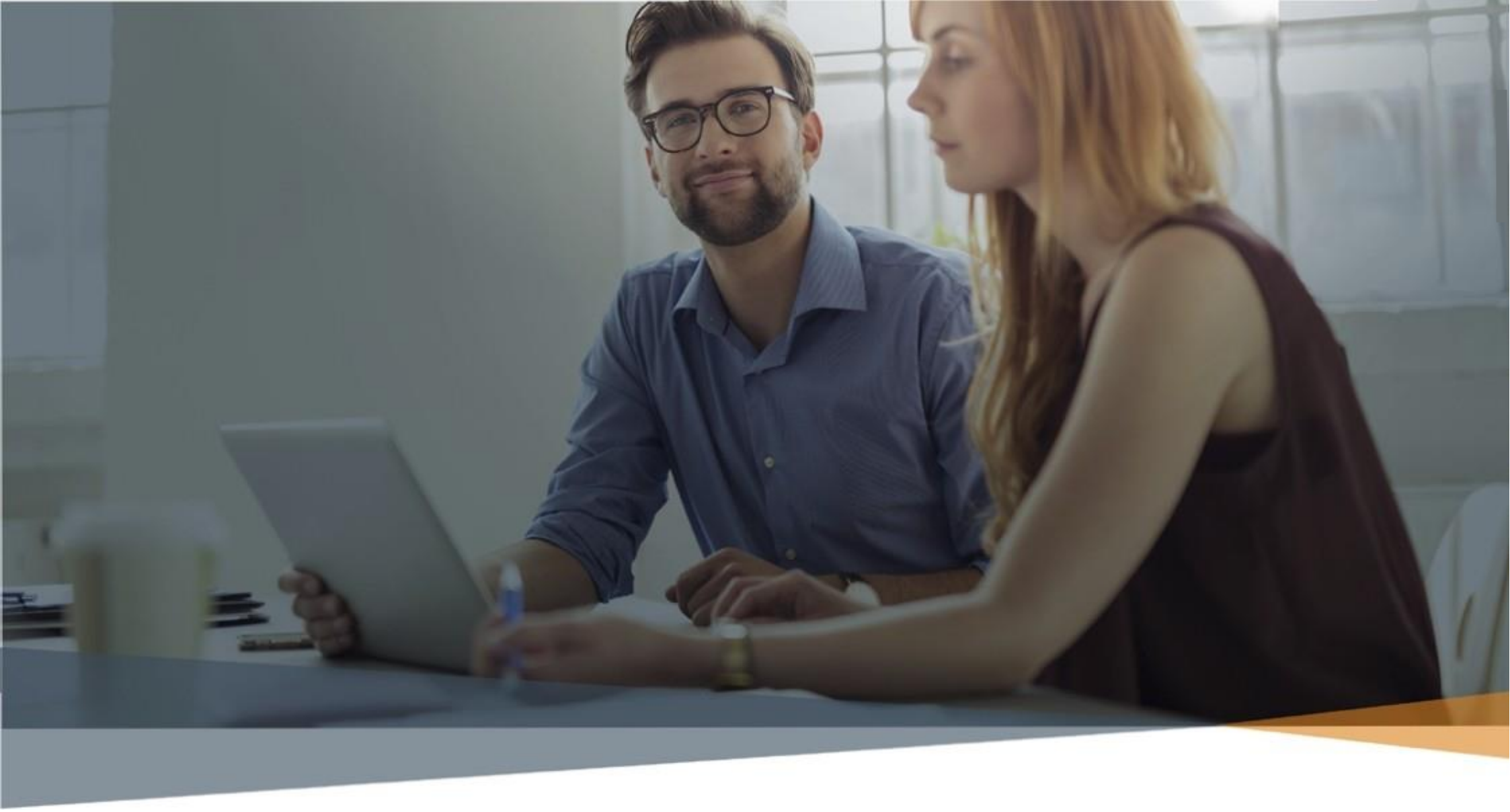
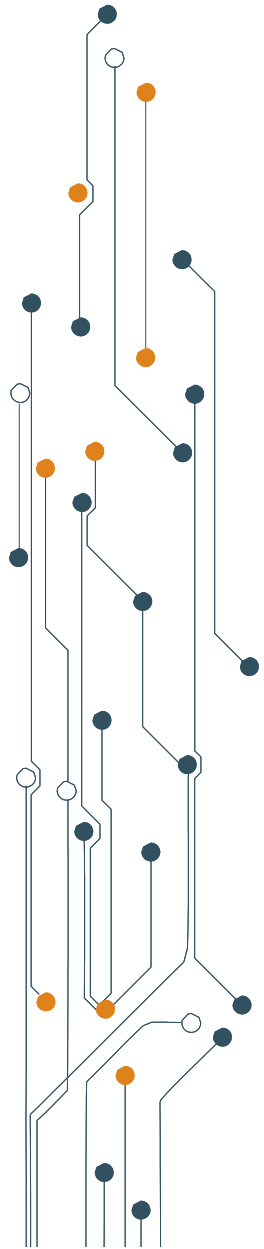


# Sintaxis del lenguaje Java

Telefónica
EDUCACIÓN DIGITAL

# Índice



## Sintaxis del lenguaje Java

# 1. Sentencias y bloques de código

Una sentencia en Java es una instrucción ejecutable que termina con el carácter ";". Este carácter es en realidad un separador de sentencias. Normalmente se ponen unas sentencias debajo de otras. No suele ser habitual poner varias sentencias en la misma línea.

En el ejemplo "HolaMundo.Java", en la función "main" se ejecuta la sentencia:

```
System.out.println("Hola mundo");
```

Como se irá viendo en este apartado y siguientes los principales tipos de sentencias son:

- De creación de posiciones de memoria (variables y objetos).
- De asignación, almacenar un valor en una posición de memoria.
- De llamada a una función, como la mostrada más arriba que llama a la función "println" que muestra un mensaje por pantalla.
- De control del flujos de programa, como son las sentencias de decisión y de iteración.

Incluso es válida la sentencia vacía formada solamente por un ";".

Un bloque es un grupo de sentencias, delimitadas por los caracteres "{" y "}". En el ejemplo siguiente se muestra un bloque formado por una única sentencia:

```
{
  System.out.println("Hola mundo");
}
```

Por claridad en la escritura y lectura del código se utilizan los espacios tabulados para ir "identando" las sentencias que están dentro de un bloque y así distinguir las de los diferentes bloques anidados unos dentro de otros, como por ejemplo en el siguiente código:

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

En este ejemplo se tienen dos bloques, uno que engloba todas las sentencias que definen la clase “HolaMundo”. Dentro de este bloque sólo existe la definición de la función “main”, que a su vez es un bloque con la sentencia que se ejecuta cuando es invocada. La sentencia o sentencias de cada bloque se “identan” o sangran con las tabulaciones.

Ejemplos de sentencias:

```
int i=0; //Sentencia de creación de variable
i = teclado.nexInt();    //Sentencia de asignación
boolean esPar = i % 2 == 0 ? true  : false;  //Sentencia de asignación
if( esPar == true){  //sentencia de decisión simple
  System.out.println(“Se  ha tecleado un número par”);
}else{
  System.out.println(“Se  ha tecleado un número impar”);
}
```

# 2. La función main

Una función es un código que se ejecuta cuando es llamada, por tanto las funciones tienes dos partes bien diferenciadas:

- **Definición**, bloque se sentencias que se ejecutan.
- **Llamada** a la función, provoca que se ejecuten las sentencias.

La función "main" es la que se tiene que definir en los programas Java para indicar que sentencias son las que dicho programa ejecuta, en definitiva y expresado de forma sencilla la función "main" es el programa Java.

La función "main" es llamada, como se comento en apartado anterior, por la JVM, cuando desde "java" se invoca la clase en la que esta definida.

Todas las funciones cuando terminan de ejecutarse devuelven siempre un valor a quien las llama, en el caso de la función main no devuelve nada, "void" en Java, al sistema, a la JVM.

En Java cuando una función no necesita ser invocada por medio de un objeto, en su definición se utiliza la palabra "static", que como veremos más adelante significa que la función o variable son de la clase, no de instancias de la misma (objetos). La función "main" es de la clase que forma la aplicación Java, no se necesita crear ningún objeto para que JVM la invoque.

Cuando se quiere utilizar una clase, una función o un dato, fuera del ámbito o bloque donde esta definido se utiliza el especificador de acceso "public". Como se estudiara existen especificadores de acceso más restrictivos para impedir el acceso al elemento que especifican. La función "main" es pública, para que la JVM pueda invocarla, al igual que la clase "HolaMundo".

La clase que contiene la función "main" de un programa Java tiene que estar en ámbito de acceso "public", además de estar almacenada en un archivo con el mismo nombre y con extensión "java", como en el ejemplo "HolaMundo".

Las funciones en su definición a la derecha de su nombre y encerrada entre paréntesis contienen la lista de parámetros que pueden recibir en su llamada. La lista puede estar vacía, puede contener un elemento o más de uno, en este caso el separador de parámetros en el carácter ",".

La función "main" recibe como parámetros cero, una o más cadenas de caracteres. **String** en Java denota cadena de caracteres, como por ejemplo "Hola mundo".

Para especificar un conjunto de datos del mismo tipo y almacenados consecutivamente en memoria se utiliza el tipo "array", en Java se indica utilizando los caracteres "[" y "]" como se verá más adelante.

En el caso de la función main:

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

args es un array que contiene una cadena por cada uno de los parámetros con los que se invoco con "java" a la clase que define la función. En el ejemplo "Hola Mundo", cuando se ejecuto se invoco sin parámetros, como se observa en la imagen siguiente:

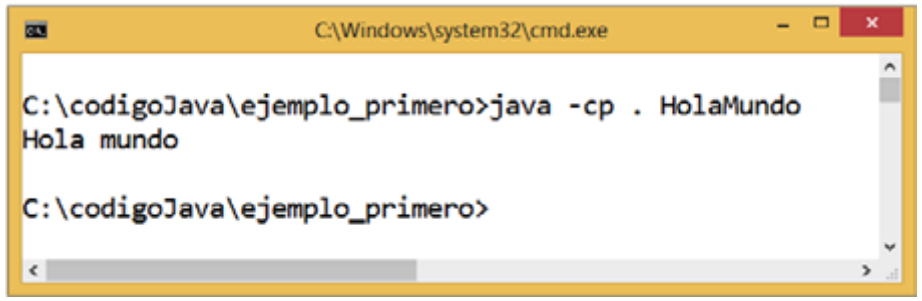



FIGURA 2.1: EJECUCIÓN DE "HOLAMUNDO" SIN LISTA DE PARÁMETROS.

En el código del ejemplo siguiente se utiliza el primer String almacenado en el array de los parámetros de la llamada a “main” para cambiar la cadena que se muestra en pantalla.

```
public class HolaMundo{
  public static void main(String[] args) {
        System.out.println(“Hola  mundo, “+ args[0]);
  }
}
```

Para acceder a un elemento de un array se introduce el índice de dicho elemento (en realidad es el “offset” o desplazamiento con respecto al primer elemento) entre los “corchetes cuadrados”, el índice del primer elemento en un array es siempre 0 (en realidad es el “offset” o desplazamiento con respecto al primer elemento). Así “args[0]” será la cadena que figure a la derecha de “HolaMundo” cuando se ejecute el código.

En la imagen siguiente se muestra como es la forma de invocar a esta nueva versión de “HolaMundo” y cuál es el resultado, después de compilar.

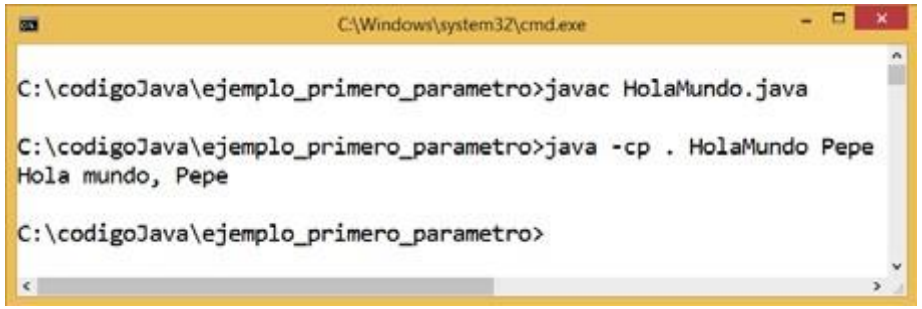



FIGURA 2.2: EJECUCIÓN DE HOLAMUNDO CON UN PARÁMETRO.

Sin embargo el código de esta nueva versión falla cuando se llama a HolaMundo sin ningún parámetro, porque se intenta acceder al elemento de índice 0 del "array" args y en este caso esta vacío, no tiene ningún elemento. El resultado de la ejecución de esta forma es el mostrado en la imagen siguiente:


FIGURA 2.2: EJECUCIÓN DE HOLAMUNDO CON UN PARÁMETRO.

El código de "main" tiene que tener en cuenta si será ejecutado su código con o sin parámetros, la forma correcta sería la siguiente:

```
public class HolaMundo{
  public static void main(String[] args) {
        if (args.length  == 0){
              System.out.println("Hola mundo");
        }
        else{
              System.out.println("Hola  mundo, "+
args[0]);
        }
  }
}
```

# 3. Palabras clave e identificadores

Como en todos los lenguajes de programación, los identificadores son conjuntos de caracteres que tienen los elementos con los que se forman los códigos, se puede hacer una clasificación en dos grandes tipos:

- **Palabras clave o reservadas**, son los identificadores que forman el léxico de Java. Las palabras public, class, void, static, son de este tipo.

**Identificadores** de usuario o simplemente identificadores, son los que crea el programador para poder utilizar los elementos como clases, funciones y variables. HolaMundo, como nombre de clase es un identificador. String también es un identificador de este tipo, porque este nombre lo creo quien desarrollo la librería de funcionalidades en las que se encuentra la definición de este tipo de datos.

En la imagen siguiente se muestran las 50 "keywords" (palabras clave) de Java que figuran en la página de Oracle: http://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html:

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

* not used
** added in 1.2
*** added in 1.4
**** added in 5.0

FIGURA 2.3: LISTA DE PALABRAS RESERVADAS DE JAVA.

Las reglas para formar identificadores validos son las siguientes:

- El primer carácter tiene que ser una letra, el carácter subrayado (_) o el carácter dólar ($).
- Puede incluir números, pero no comenzar por uno de ellos.
- No se pueden incluir espacios en blanco.
- Se distingue entre letras mayúsculas y minúsculas.
- No se pueden utilizar las palabras reservadas como identificadores.

Estas reglas son de obligado cumplimiento, pero además se tienen en cuenta otras por convección o estilo que se pueden resumir en las siguientes:

| Tipo de identificador | Convención | Ejemplo |
|---|---|---|
| **nombre de una clase** | comienza por letra mayúscula | String, System, HolaMundo |
| **nombre de función** | comienza con letra minúscula | getEdad, setNombre, calculaArea |
| **nombre de variable** | comienza por letra minúscula | edad, nombre, i |
| **nombre de constante** | en letras mayúsculas | X_MAX, PI, VERDE |

TABLA 2.1: CONVENCIÓN DE NOMBRES PARA IDENTIFICADORES.

# 4. Tipos de datos primitivos

Java es un lenguaje con comprobación estricta de tipos, por tanto todos los identificadores tienen que tener asociado un tipo. Existen 9 tipos llamados primitivos, porque son los que vienen definidos de forma implícita en el lenguaje, cada uno de ellos tiene como nombre uno de la lista de palabras reservadas. Se pueden clasificar en los siguientes grupos: carácter, enteros, punto flotante (decimales), lógico y void (nada).

En la tabla siguiente se muestran los 9 tipos, con sus características. Cada tipo tiene asociado una clase (llamada envoltorio) que permitirá trabajar con objetos que encapsulan estos tipos primitivos.

| Tipo | Representación | Bytes | Rango | Valor por omisión | Clase envoltorio |
|---|---|---|---|---|---|
| **char** | Caracter Unicode | 2 | \u0000 a \uFFFF | \u0000 | Character |
| **byte** | | 1 | -128 a 127 | 0 | Byte |
| **short** | | 2 | -32768 a 32767 | 0 | Short |
| **int** | Entero con signo | 4 | -2147483648 a 2147483647 | 0 | Integer |
| **long** | | 8 | -9223372036854775808 a 9223372036854775807 | 0 | Long |
| **float** | Punto flotante precisión simple | 4 | ±3.4*10-38 a ±3.4*1038 | 0.0 | Float |
| **double** | Punto flotante precisión doble | 8 | ±1.8*10-308 a ±1.8*10308 | 0.0 | Double |
| **boolean** | Lógico | 1 | true o false | false | Boolean |
| **void** | - | - | - | - | Void |

TABLA 2.2: TIPOS PRIMITIVOS.

# 5. Literales

Los literales son valores fijos o constantes, representados en forma legible para las personas. cada tipo primitivo tiene su forma de representar sus literales.

| Tipos | Representación | Ejemplos |
|---|---|---|
| **char** | Encerrados entre ‘ ‘ | ‘a’, ‘1’, ‘\n’, ‘\u0037 |
| **byte, short, int** | Los dígitos del número, sin parte fraccionaria. Por omisión todos son de tipo int. | 1245, -345678, 3_456_789<br>El carácter _ hace de separador de miles |
| **long** | Como los anteriores pero terminados en l o L | 34567898923L, -383838383l, 89_678_578_976L |
| **double** | Con parte decimal, por omisión todos los punto flotante son double | 0.0, -12345e-125, .0456 |
| **float** | Como los anteriores seguidos de f o F | 0.0F, -1245.35f, 345.43e12F, .0f |
| **boolean** | true o false | |
| **String** | Aunque no es un tipo primitivo, sus literales tienen representación como caracteres delimitados por “” | “Hola mundo”, “\nNombre: “ |

TABLA 2.3: LITERALES.

# 6. Variables, ámbito y duración

Una variable en Java es una posición de memoria en la que se almacenan un valor de un tipo primitivo. Cada variable tiene que estar declarada de acuerdo a uno de los tipos primitivos, no existen variables sin tipo asociado.

El contenido de las variables pueden cambiar de valor, desde que se crean hasta que dejan de existir. Esto puede cambiar según veremos utilizando modificadores como final.

Tienen un tiempo de vida y un ámbito de acceso. Se crea espacio en memoria cuando se declaran y se libera el espacio cuando llegan al final de su ámbito.

En resumen, las variables se declaran, se inicializan y se usan.

La sintaxis de declaración de una variable es la siguiente:

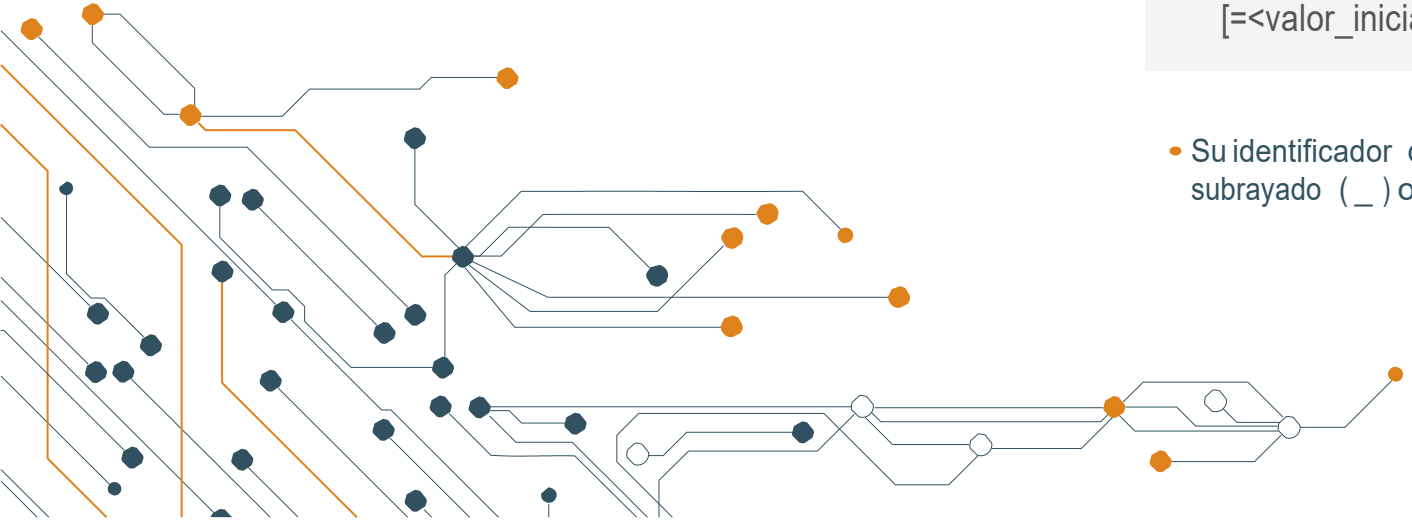
- Declaración de una variable:

```
<tipo> <identificador> [=<valor_inicial>];
```

- Declaración de un grupo de variables todas del mismo tipo:

```
<tipo> <identificador> [=<valor_inicial>]{ <identificador>
[=<valor_inicial>],};
```

- Su identificador debe comenzar con una letra, el carácter subrayado ( _ ) o el carácter dollar ( $ ), nunca por número.



Algunos ejemplos son:

```
int i=21;
char c='A';
String nombre = "Pepe";      //en realidad nombre
                               es una referencia
                              //a un objeto String
float sueldo=1245f, complemento=321.5f;
short a, c=4, d=6; f;
long numero= 4_567_890; float
base=6.0f, altura=14.3f; double
area = base * altura / 2;
```

Existen tres tipos de variables en Java:

- **Locales:** son aquellas que se declaran dentro de los bloques de una función, su ámbito es el bloque en el que se crean y duración desde que la sentencia en la que se crean hasta que finaliza el bloque, momento en el cual es liberado su espacio en memoria. Siempre tienen que ser inicializadas.
- **De instancia:** son las que se declaran en el bloque de una clase, cuando se crea un objeto de esa clase (instancia) se crea el espacio para la variable. Su ámbito es el que se especifique con los modificadores de ámbito como public, private o protected. Sea cual sea su ámbito su acceso será siempre a través del objeto al cual pertenecen. su duración es la del objeto al que pertenecen. Se crean con el objeto, se libera su espacio, cuando se libera el espacio que ocupa el objeto.
- **De clase:** son espacios de memoria que comparten todos los objetos de una clase. Están declaradas como las anteriores en el ámbito de una clase, pero precedidas del modificador static. Su duración es el tiempo en el que la clase a la que pertenecen esta en uso. La función "main" es static (de clase) como se estudio anteriormente y por esta característica no puede acceder a variables de instancia que estuvieran declaradas en la clase.

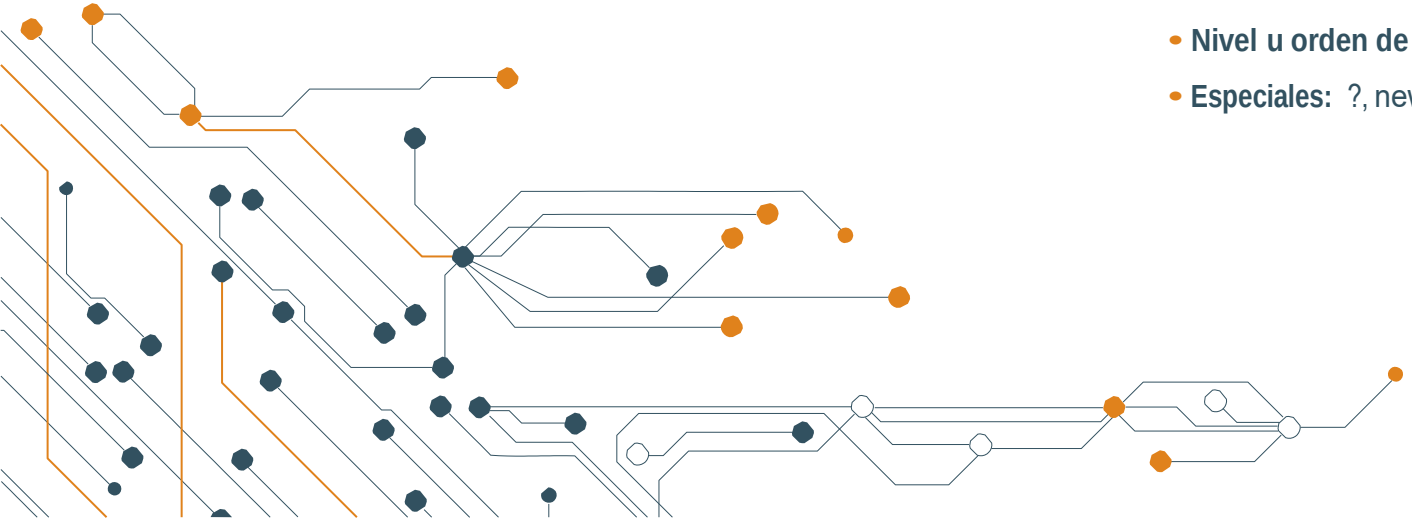
# 7. Operadores y tipos

Un operador es un símbolo que relaciona elementos mediante una funcionalidad. Los operadores indican al compilador que realice una operación. Los elementos que relaciona un operador son los operandos. Cada operador relaciona operandos de tipos determinados y concretos. Una primera clasificación de los operadores es la que distingue con cuantos operadores se relaciona un operador. En Java según este criterio existen tres tipos de operadores:

- **Unarios:** sólo tienen un operador.
- **Binarios:** se relacionan con dos operadores.
- **Ternarios:** se relacionan con tres operadores.

Según la funcionalidad del operador se pueden agrupar en las categorías siguientes:

- **Aritméticos:** +, -, *, /, %, ++, --
- **De asignación:** =, +=, -=, *=, /=, %=
- **Relacionales:** ==, !=, <, >, <=, >=
- **Lógicos:** ^, &&, ||, !, &, |
- **Concatenación:** +
- **Nivel u orden de bits:** &, !, ^, <<, >>
- **Especiales:** ?, new, [], ( )



Los operadores aritméticos +, -, * y /, se pueden aplicar a cualquier tipo numérico, incluso char.

Los operadores + y - son también unarios y cambian el signo del operando que les sucede.

El operador % es el operador resto, devuelve un entero que es el resto de la división entera de sus dos operandos. Por ejemplo 9 % 2 devuelve como resultado 1, 153 % 7 devuelve 6.

Los operandos unarios ++ y --, son de incremento y decremento de una unidad del operando con el que se relacionan, pero tienen dos funcionalidades distintas:

- **Prefijo**, preceden al operando, primero incrementan su valor y después el operando se relaciona con el resto de operandos y operadores de la expresión en la que se encuentre.
- **Postfijo**, suceden al operando, primero el operando se relaciona con los demás operandos y operadores de la expresión y después incrementan el valor.

Los operadores de asignación, cambian el valor de la variable de su izquierda por el valor de la expresión de su derecha. Los que tienen asociado un operador matemático, almacenan en el operando de su izquierda el valor de la expresión de la derecha operado con el valor de la variable de su izquierda.

Ejemplos:

```
int x = 137, y = 6, z = 19, i=1, j=5, k=6;
x%=y+z++; // x: 12 y: 6 z: 20
i++;
j=--k+i; // i: 2 j: 7 k: 5
```

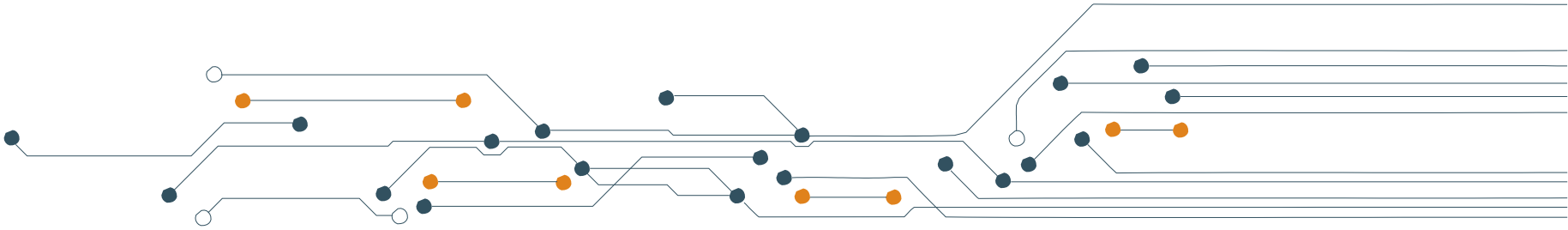
Los operadores relaciones siempre devuelven un valor booleano (true o false). Son todos binarios y comparan el valor del operando de la derecha con el de su izquierda. son aplicables a todos los tipos primitivos del lenguaje.

Los operadores lógicos siempre devuelven un valor booleano, normalmente se utilizan para concatenar expresiones con operadores relaciones. Son todos binarios salvo el de de negación. Implementan las operaciones booleanas and, or, or exclusive y not.

Su funcionalidad se recoge en la tabla siguiente:

| Operando Izquierdo | Operando derecho | & && (and) | \| \|\| (or) | ^ (or exclusivo) | ! Aplicado operando izdo. |
|---|---|---|---|---|---|
| true | true | true | true | false | false |
| true | false | false | true | true | false |
| false | true | false | true | true | true |
| false | false | false | false | false | true |

TABLA 2.4: OPERADORES LÓGICOS.



Para las operaciones lógicas and y or se puede utilizar un símbolo o dos, cuando se realiza la operación booleana con doble símbolo (&& o ||) se denomina “operador de cortocircuito”, , se aplica cuando se concatenan varios operadores booleanos en una misma expresión y su funcionalidades:

- **&&**, en cuanto un operando es false, no se sigue evaluando.
- ||, en cuanto un operando es true, no se sigue evaluando.

Ejemplos:

```
boolean a=true, b=false ,c=true;
b = a | c;              //a: true b: true c: true
c = !a;                 //a: true b: true c: false
b = c && a && b;        //a: true b: false c: false
c= a ^ !b;              //a: true b: false c: false
```

El operador de concatenación (+) ya se ha utilizado en ejemplo “HolaMundo” utilizando el parámetro de main. este operador devuelve siempre una cadena de caracteres (String). Este operador transforma a cadena de caracteres el operando que no es de este tipo.

Ejemplo:

```
String cadena=” la cadena “;
float numero= 3_456_789e-5f;
boolean booleano= false;
cadena = numero + cadena + booleano; // “34.56789
cadena false”
```

Los demás tipos de operadores se irán estudiando según se avance en el desarrollo del contenido del curso.

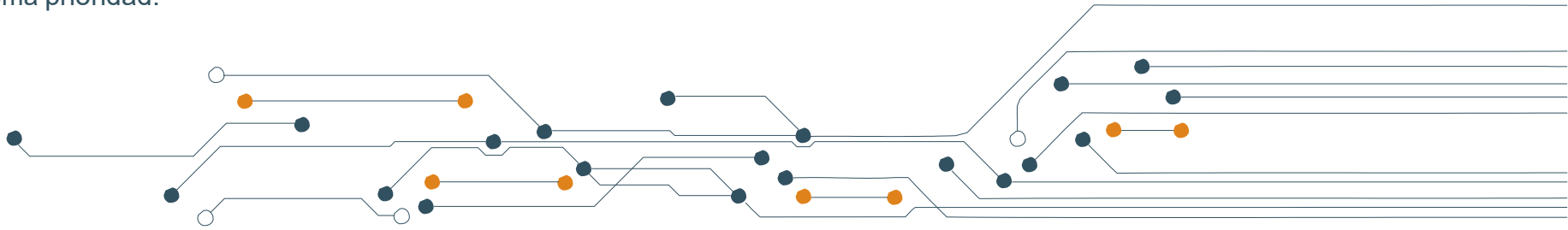
# 8. Expresiones y precedencia de operadores

Una expresión es un conjunto de operandos y operadores que produce un resultado. Las expresiones más sencillas son aquellas que utilizan o sólo un operando o sólo un operando o un operador o un operando y dos operadores, como por ejemplo:

```
System.out.println("Hola  mundo"); //"Hola mundo" expresión con un sólo operando
i++; //Un sólo operando y un operador, i se incrementa  en una unidad  (i=i+1)
i += 5; //Dos operadores y un operando (i= i + 5)
```

Cuando se utilizan dos o más operadores hay que tener en cuenta la prioridad de evaluación. Se aplican las seis siguientes:

- En una expresión los operadores se evalúan de izquierda a derecha.
- No todos los operadores tienen la misma prioridad al evaluarse una expresión.
- El orden de evaluación sólo es aplicable para los operadores que tengan la misma prioridad.
- Para cambiar el orden de evaluación según la prioridad se utilizan el operador paréntesis. Siempre se evalúa primero lo que este encerrado entre paréntesis.
- Los operadores de asignación son los que menos prioridad tienen, siempre son los últimos en evaluarse.
- A igual prioridad se evalúa de izquierda a derecha.



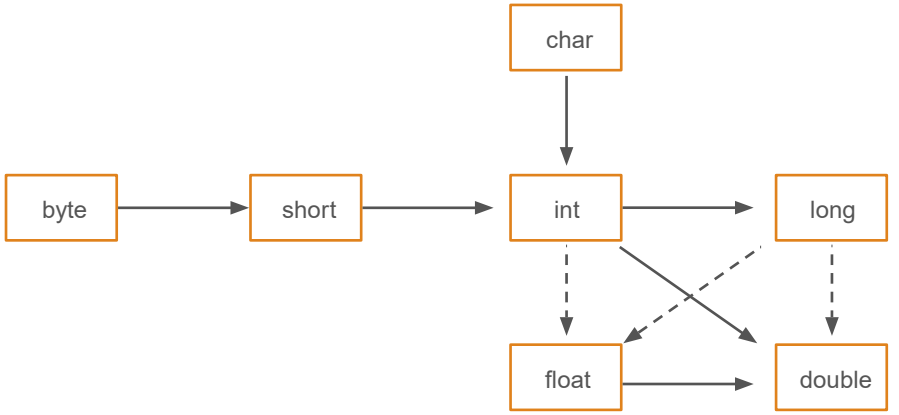
El orden de prioridad de más a menos en los operadores es el que se muestra en la tabla siguiente, mostrando también su forma de asociarse. Algunos operadores mostrados en la tabla se estudiaran más adelante:

| Operadores | Tipo | Asociatividad |
|---|---|---|
| [] . () y llamada a método | Especiales | De izquierda a derecha |
| ! ~ ++ -- +(unario) -(unario) (moldeo) new | Unarios, moldeo y creación | De derecha a izquierda |
| * / % | Aritméticos | De izquierda a derecha |
| + - | Aritméticos | De izquierda a derecha |
| << >> >>> | Nivel de bits | De izquierda a derecha |
| < <= > >= | Relacionales | De izquierda a derecha |
| == != | | De izquierda a derecha |
| & | Lógicos | De izquierda a derecha |
| ^ | | De izquierda a derecha |
| \| | | De izquierda a derecha |
| && | | De izquierda a derecha |
| \|\| | | De izquierda a derecha |
| ?: | Ternario | De izquierda a derecha |
| Todos los de asignación | | De derecha a izquierda |

TABLA 2.5: PRIORIDAD DE LOS OPERADORES.

En las asignaciones los tipos de variable y valor a asignar tienen que ser compatible. Todos los tipos numéricos son compatibles, pero hay que tener cuidado en los tamaños de los tipos del valor a asignar y el tamaño de la variable que reciba el valor, si este es más pequeño se pierde precisión.

Las conversiones válidas entre tipos numéricos son las que se indican en la imagen siguiente:




Las flechas continuas implican conversiones sin pérdida de información.
Las flechas discontinuas implican conversiones en las que puede haber perdida de información.

FIGURA 2.4: CONVERSIONES ENTRE TIPOS NUMÉRICOS.

Cuando se relacionan en expresiones, valores numéricos con operadores aritméticos, también hay que tener en cuenta las siguientes reglas:

- Si alguno de los operandos es double, el otro se convierte a double.
- Si ninguno es double, pero uno es float, el otro se convierte a float.
- Si ninguno es double o float, pero uno es long, el otro se convierte a long.
- En cualquier otro caso se convierten a int.

Estas conversiones son todas implícitas, pero también se pueden realizar conversiones de forma explícita. Se realizan utilizando la sintaxis siguiente:

variable_destino = (tipo_destino) variable_origen;

A esta operación se le conoce como casting o estrechamiento y, aunque en algunos casos puede provocar una pérdida de datos, no se producirán errores en tiempo de ejecución.

Ejemplo:

```
long l=100;
byte b;
b=(byte)l;
```

# 9. Entrada y salida básica

En Java todo el sistema de entrada y salida se basa en una jerarquía de clases que se repasara más adelante. No obstante ya se ha estado utilizando la salida estándar a la consola por medio de las función “println”, invocada para el objeto “out”, variable (objeto) miembro de la clase System.

Todo el sistema de E/S se realiza a través de flujos. Un flujo se puede definir como una abstracción que consume o produce información.

Para escribir en la consola se utiliza el flujo “out”, este flujo consume información. Para introducir información a través del teclado se

A cualquier flujo, que lleva la información de un origen a un destino, se le pueden aplicar diferentes funciones. En el caso de “out”, de momento se van a utilizar las funciones:

- println: lleva la cadena de caracteres recibida como parámetro a la consola y salta a nueva línea.
- print: igual que la anterior, pero sigue en la misma línea.

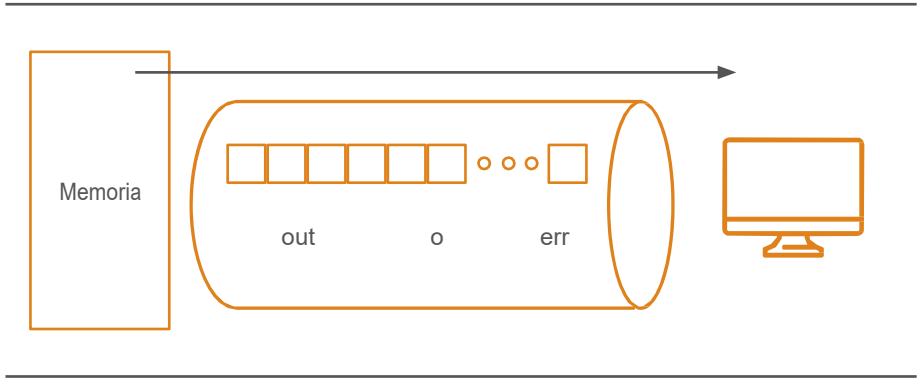



FIGURA 2.5: FLUJO DE ESCRITURA EN LA CONSOLA.

Existe otro flujo de salida que se comporta como “out”, es “err”, lleva la información a la consola pero se utiliza para indicar posibles errores en la ejecución del programa. En IDEs como Eclipse o NetBeans se utiliza con color rojo.

El flujo para introducir o producir información con el teclado es “in”. La formas más sencilla de utilizarlo es a través de un objeto de la clase Scanner. En el ejemplo siguiente se muestra como introducir una cadena en un String con los caracteres tecleados hasta que se pulsa la tecla “Enter”:

```
//Se crea objeto Scanner para el flujo “in”
Scanner teclado = new Scanner(Sytem.in);
//La función nextLine de Scanner devuelve todos los
//caracteres teclados hasta el caracter “fin de línea”
String cadena = teclado.nextLine();
```

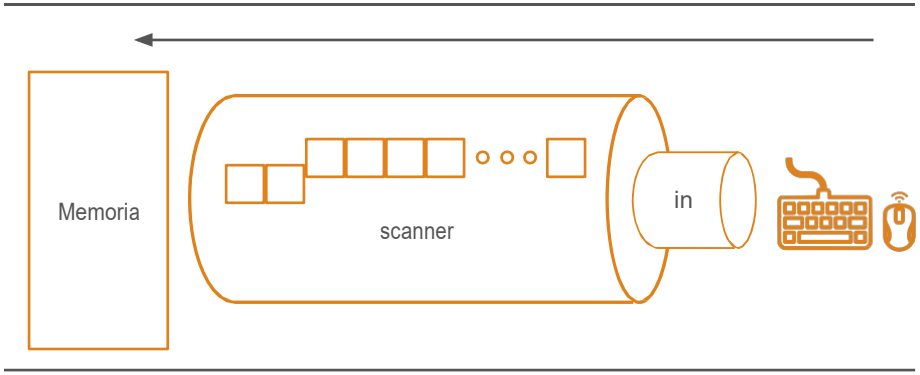



FIGURA 2.6: FLUJO DE LECTURA DEL TECLADO.

Otras funciones de Scanner para leer información son los siguientes:

- **next()**, lee como String lo tecleado hasta encontrar un espacio en blanco o fin de línea.
- **nextXXX()**, donde XXX puede ser uno de los tipos primitivos del lenguaje, excepto char. Lee el valor del tipo correspondiente. Si el valor leído no es del tipo adecuado se produce la excepción **InputMismatchException**, por tanto habrá que estar seguro que lo que se ha teclado se corresponde con el tipo adecuado de la función next.

En realidad las funciones next, leen del “buffer” asociado al objeto Scanner, todo lo tecleado se almacena en dicho buffer y este se va vaciando de acuerdo con lo que se va leyendo con las funciones next.

Si se utiliza la función next() o cualquier función nextXXX(), el caracter fin de línea no la vacían del buffer, por tanto si la siguiente sentencia utiliza nextLine(), esta saca del buffer dicho caracter y cómo es un caracter terminador para ella no lee la cadena que estuviera en la línea siguiente, a continuación del caracter fin de línea. Por ejemplo tal y como se muestra en la imagen siguiente:

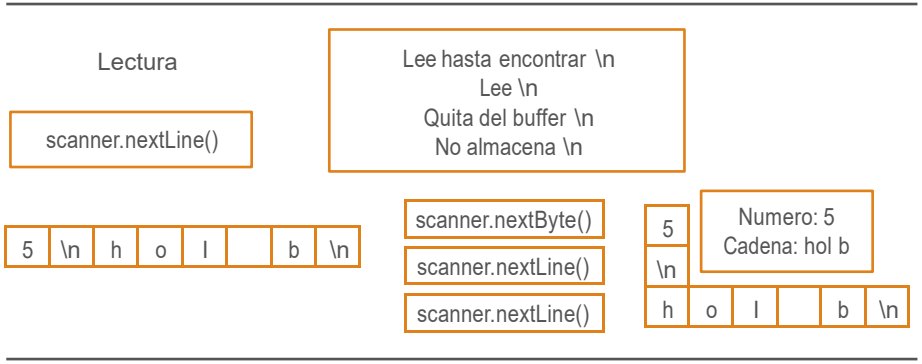



FIGURA 2.7: LECTURA DE TECLADO CON FUNCIONES DE SCANNER.

En el código siguiente se muestra como utilizar la función nextInt y la función nextLine:

```
System.out.print("Teclea un número: ");

//Teclear número y almacenarlo en variable
n=teclado.nextInt();

//Declarar una referencia para cadenas de caracteres
String nombre;
System.out.print("Teclea tu nombre: ");

//Teclear una cadena de caracteres
//Hay que recoger el CR (fin de línea) pulsado para introducir número
teclado.nextLine();
nombre=teclado.nextLine();
```

# 10. Funciones y parámetros

Como ya se ha mencionado, una función es un bloque de sentencias que se ejecutan cuando es llamada devolviendo un resultado. En la definición de una función se pueden distinguir tres partes:

- La cabecera o firma, en la que se define el tipo que devuelve, el nombre de la función y la lista de parámetros. También se incluyen en esta parte el ámbito de acceso y otros modificadores, como el ya mencionado static.

  La sintaxis puede ser esta que se indica a continuación:

  <acceso> <funcionalidad> <tipo_devuelto> <nombre_funcion>(< parametros>)

- Los paréntesis son obligatorios aunque la lista este vacía.
- El bloque de sentencias que se ejecutan, siempre encerrado entre llaves, { y }.
- El valor devuelto, es el resultado de la expresión que se codifique a la derecha de la sentencia return. Si la función no devuelve ningún valor, no hace falta incluir ninguna sentencia **return**. Dentro de una función puede haber más de un return, porque dependiendo de ciertas condiciones la función devolverá unos valores u otros.

Una función es un ámbito que tiene como duración, desde que es llamada hasta que retorna el valor en la sentencia en la que fue llamada. Todas las variables y objetos que se creen dentro de este ámbito son liberados de memoria cuando la función finaliza. Por eso todos los identificadores declarados dentro de una función son locales a dicha función. Por tanto una función define un ámbito local.

La imagen siguiente muestra la definición de una función con sus diferentes partes:

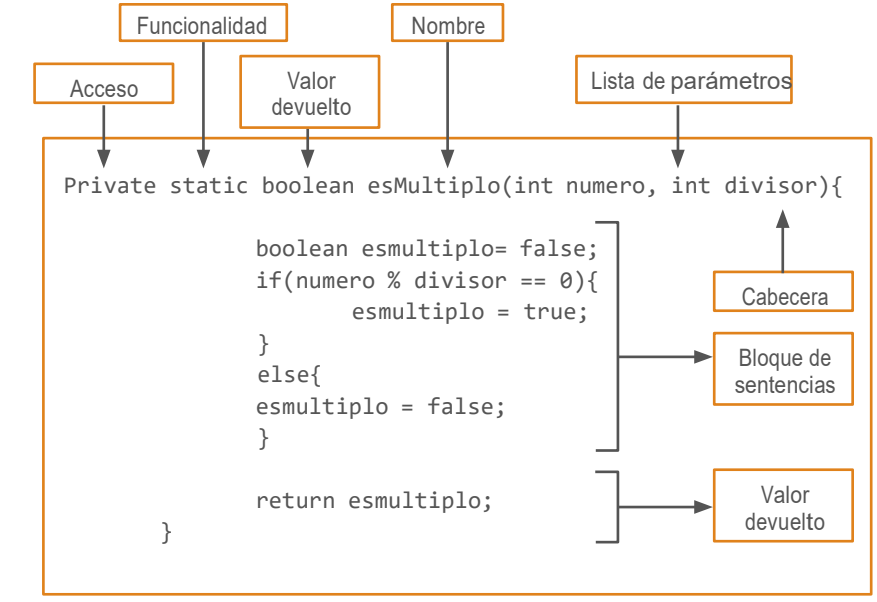



FIGURA 2.8: DEFINICIÓN DE UNA FUNCIÓN.

La llamada a una función tiene que estar conforme a su cabecera o firma, indicando el nombre y entre paréntesis la lista de valores compatibles con los tipos de los parámetros. La funciones tienen que ser llamadas donde se espere un valor del tipo que retorna la función. La siguiente imagen, muestra como se llama a la función del ejemplo anterior.

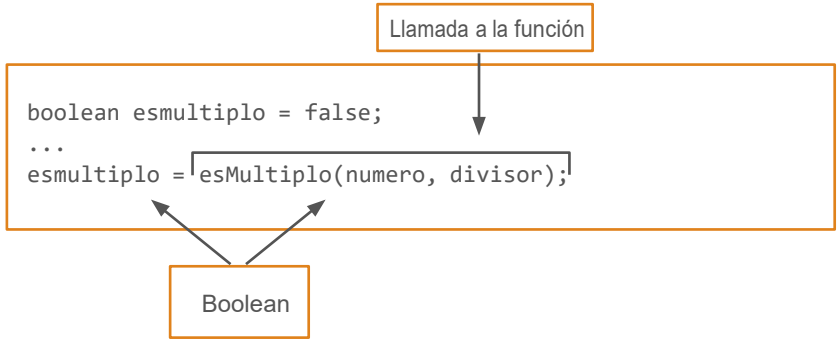
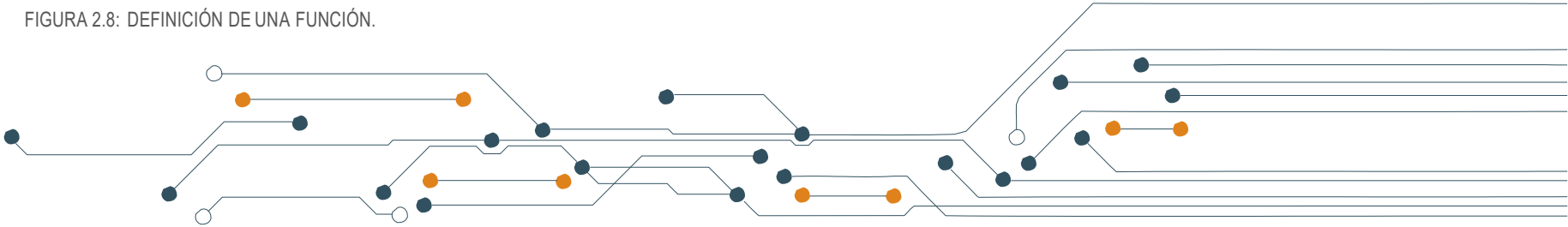



FIGURA 2.9: LLAMADA A UNA FUNCIÓN.



# 11. Comentarios

Los comentarios son anotaciones ignoradas por el compilador que ayudan a los programadores a documentar y a explicar el código, para que otros lo entiendan y para facilitar los cambios o mejoras.

Los comentarios en Java son de dos tipos:

- **De una sola línea:** vienen marcados por los caracteres **//**, que señalan el comienzo del comentario, el cual se extiende hasta el final de la línea.
- **De varias líneas:** vienen acotados por los caracteres /* al principio del comentario y el par */ al final de la última línea que forma dicho comentario.

La imagen siguiente muestra los dos tipos de comentarios.

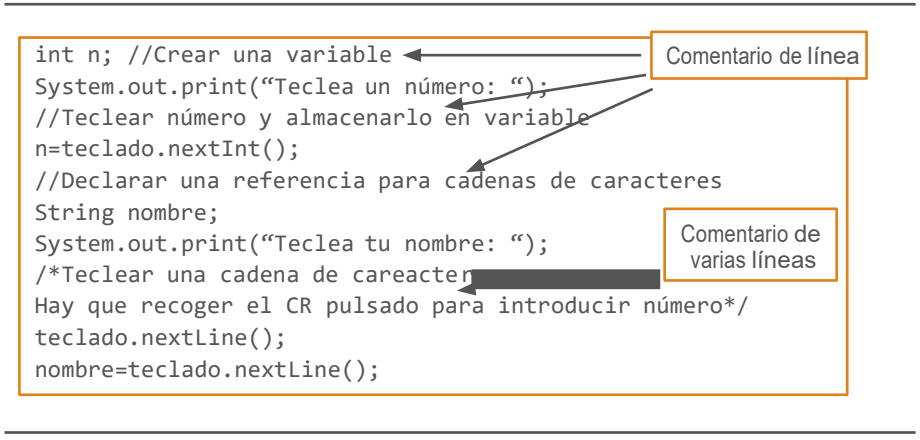
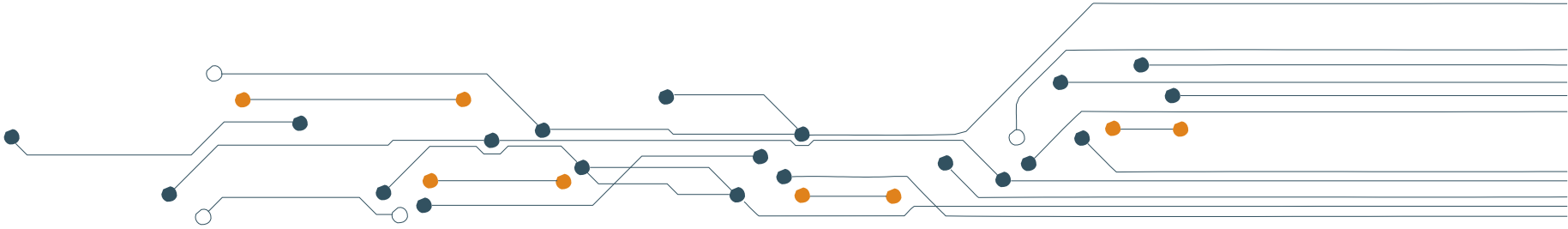



FIGURA 2.10: COMENTARIOS.



# 12. Generador de documentación javadoc

El programa **javadoc** es un generador de documentación, rastrea los ficheros .java buscando unos comentarios especiales que permiten generar páginas HTML con documentación acerca de éstos. javadoc, esta en el mismo directorio que el compilador javac y el programa java.

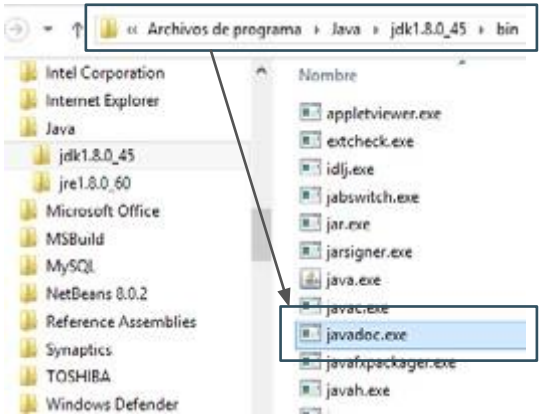



FIGURA 2.11: UBICACIÓN DE JAVADOC.

javadoc genera un fichero por cada .java encontrado, uno para los paquetes (packages.html), otro para la jerarquía de clases (tree. html) y otro con el índice de todos los elementos (AllNames.html).

Los comentarios en javadoc empiezan con /** y acaban con */. En éstos se pueden añadir etiquetas HTML (excepto las de título: <H1>).

Se pueden agrupar en las categorías que se describen a continuación.

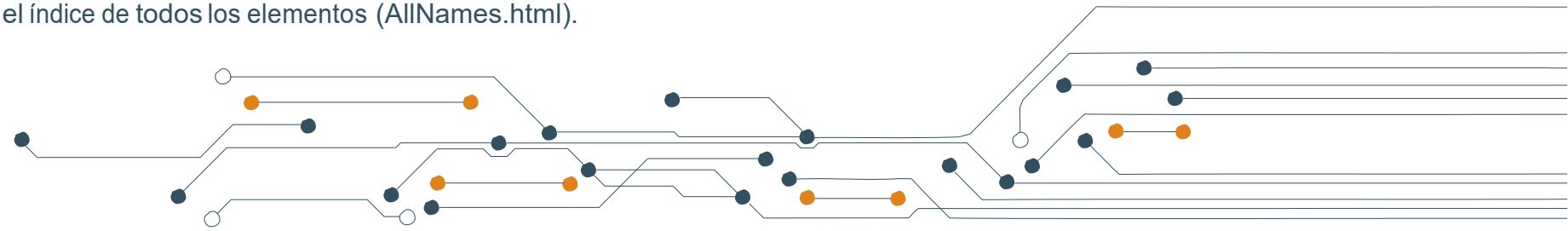
- **Etiquetas de clase o interface:** Se ponen antes de la declaración de una clase o interface:

  **@author texto: autor de la clase / interface**

  **@version texto: versión de la clase / interface**

  **@see nombreDeClase: crea un hiperenlace**

  **@since texto: indica desde cuánto existe**



- **Etiquetas de atributo (variable miembro):** Se debe poner antes de la declaración del atributo:

  **@see nombreDeClase: Crea un hiperenlace**

- **Etiquetas de función miembro:** Se colocan antes de la declaración de un constructor o un método:

  **@param nombreParametro descripcionParametro**

  **@exception nombreClase descripcion**

  **@return texto**

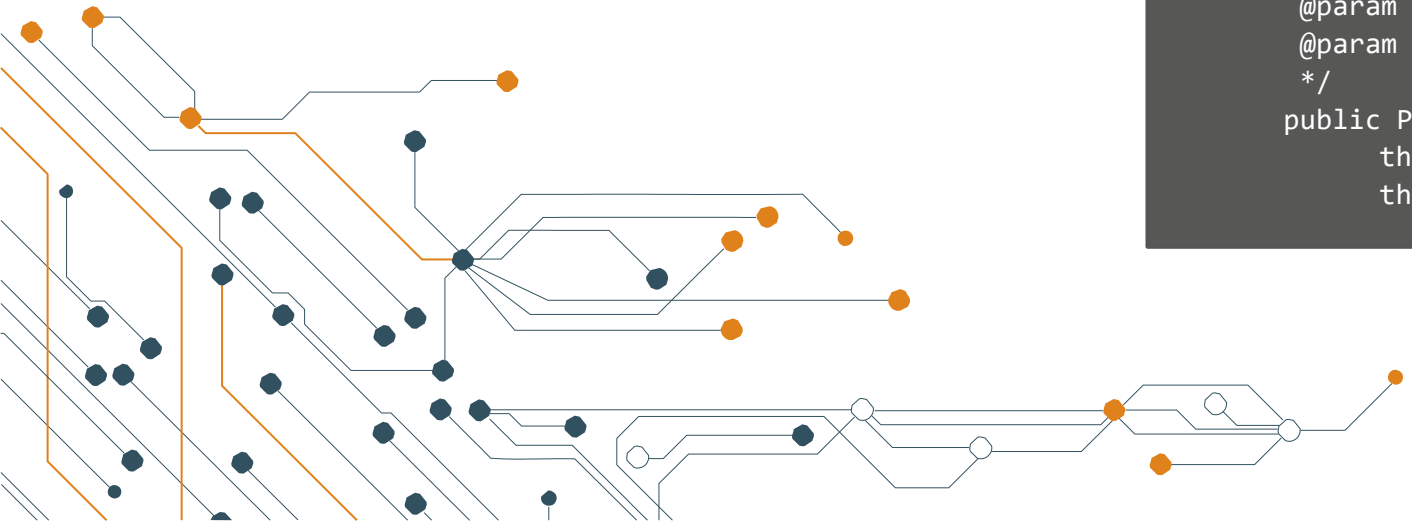
  **@see nombreDeClase**

Un ejemplo de código con comentarios para javadoc es el siguiente:

```
/**
 * Esta clase define objetos que representan  un
punto en un sistema de
 * coordenadas
 *
 * @author: Juan B. García
 * @version: 12/07/2014
 */
public class Punto }

      /**
       * Costructor que recibe dos números uno para
cada coordenada, de tipo
       * short
       *
       @param x coordenada x
       @param y coordenada y
       */
      public Punto (short x, short y) {
            this.Normalizar (x,y);
            this.nPuntos++;
```

FIGURA 2.12: COMETARIOS JAVADOC.



Telefónica

EDUCACIÓN DIGITAL