Introducción a Java

Encarnación Sánchez Gallego

Categoría de tipos de datos

- De acuerdo **al tipo** de información que representan
  - Tipo primitivo, Variables referencia
- Según cambie **su valor** o no durante la ejecución del programa
  - Variables Constante (final en Java)
- Según **su papel** en el programa
  - Variables miembros de una clase Variables locales

Tipos Primitivos

Tabla 3.1. Tipos de dato primitivos de Java

Tipo	Representación / Valor	Tamaño (en bits)	Valor mínimo	Valor máximo	Valor por defecto
`boolean`	`true` o `false`	1	N.A.	N.A.	`false`
`char`	Carácter Unicode	16	`\u0000`	`\uFFFF`	`\u0000`
`byte`	Entero con signo	8	-128	128	0
`short`	Entero con signo	16	-32768	32767	0
`int`	Entero con signo	32	-2147483648	2147483647	0
`long`	Entero con signo	64	-9223372036854775808	9223372036854775807	0
`float`	Coma flotante de precisión simple Norma IEEE 754	32	±3.40282347E+38	±1.40239846E-45	`0.0`
`double`	Coma flotante de precisión doble Norma IEEE 754	64	±1.7976931348623157E+308	±4.9406564584124654E-324	`0.0`

3_0TiposPrimitivosYreferenciados.jpg, vemos los tipos de objetos

Ejercicio Inicial (Datos Primitivos) y cadenas de texto.

Operadores

Ver pdf inicial

Ejemplos.

Operadores

Tabla 4.1 Operador asignación

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
=	Operador asignación	`n = 4`	`n vale 4`

El operador asignación, =, es un operador binario que asigna el valor del término de la derecha al operando de la izquierda. El operando de la izquierda suele ser el identificador de una variable. El término de la derecha es, en general, una expresión de un tipo de dato compatible; en particular puede ser una constante u otra variable.

Operadores

```
public class opAsignacion {
   public static void main(String[] args) {
      int i,j;
      double x;
      char c;
      boolean b;
      String s;
      i = 15;
      j = i;
      x = 12.345;
      c = 'A';
      b = false;
      s = "Hola";


 System.out.println("i = " + i);
 System.out.println("j = " + j);
 System.out.println("x = " + x);
 System.out.println("c = " + c);
 System.out.println("b = " + b);
 System.out.println("s = " + s);
```

RESULTADO

```
$>java opAsignacion↵
i = 15
j = 15
x = 12.345
c = A
b = false
s = Hola
```

Operadores Aritméticos

Tabla 4.2 Operadores aritméticos básicos

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
-	operador unario de cambio de signo	`-4`	`-4`
+	Suma	`2.5 + 7.1`	`9.6`
-	Resta	`235.6 - 103.5`	`132.1`
*	Producto	`1.2 * 1.1`	`1.32`
/	División (tanto entera como real)	`0.050 / 0.2` `7 / 2`	`0.25` `3`
%	Resto de la división entera	`20 % 7`	`6`

Operadores aritméticos

El resultado exacto depende de los tipos de operando involucrados. Es conveniente tener en cuenta las siguientes peculiaridades:

- El **resultado** es de tipo long si, al menos, uno de los operandos es de tipo long y ninguno es real (float o double).
- El **resultado** es de tipo int si ninguno de los operandos es de tipo long y tampoco es real (float o double).
- El **resultado** es de tipo double si, al menos, uno de los operandos es de tipo double.
- El **resultado** es de tipo float si, al menos, uno de los operandos es de tipo float y ninguno es double.
- El **formato** empleado para la representación de datos enteros es el complemento a dos. En la aritmética entera no se producen nunca desbordamientos (overflow) aunque el resultado sobrepase el intervalo de representación (int o long).

Operadores aritméticos

- La **división** entera se trunca hacia 0. La división o el resto de dividir por cero es una operación válida que genera una excepción ArithmeticException que puede dar lugar a un error de ejecución y la consiguiente interrupción de la ejecución del programa.
- La **aritmética** real (en coma flotante) puede desbordar al infinito (demasiado grande, overflow) o hacia cero (demasiado pequeño, underflow).
- El **resultado de una expresión inválida**, por ejemplo, dividir infinito por infinito, no genera una excepción ni un error de ejecución: es un valor **NaN** (Not a Number).

```
public class OpAritmeticos {
   public static void main(String[] args) {
      int i,j;
      double a,b;
      i = 7;
      j = 3;
      System.out.println("* Operandos enteros: i = " + i + " ;  j = " + j);
      System.out.println("  Operador suma:     i + j = " + (i+j));
      System.out.println("  Operador resta:    i - j = " + (i-j));
      System.out.println("  Operador producto: i * j = " + (i*j));
      System.out.println("  Operador division: i / j = " + (i/j));
      System.out.println("  Operador resto:    i % j = " + (i%j));
      a = 12.5;
      b = 4.3;
      System.out.println("* Operandos reales:  a = " + a + " ; b = " + b);
      System.out.println("  Operador suma:     a + b = " + (a+b));
      System.out.println("  Operador resta:    a - b = " + (a-b));
      System.out.println("  Operador producto: a * b = " + (a*b));
      System.out.println("  Operador division: a / b = " + (a/b));
      System.out.println("  Operador resto:    a % b = " + (a%b));
```

Ejemplo aritméticos

Salida por pantalla del programa anterior:

```
* Operandos enteros:     i = 7 ;  j = 3
  Operador suma:         i + j = 10
  Operador resta:        i - j = 4
  Operador producto:     i * j = 21
  Operador division:     i / j = 2
  Operador resto:        i % j = 1
* Operandos reales:      a = 12.5 ;  b = 4.3
  Operador suma:         a + b = 16.8
  Operador resta:        a - b = 8.2
  Operador producto:     a * b = 53.75
  Operador division:     a / b = 2.906976744186047
  Operador resto:        a % b = 3.9000000000000004
```

Operadores incremental

Tabla 4.3 Operadores aritméticos incrementales

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo

++	Incremento
	`i++` primero se utiliza la variable y luego se incrementa su valor
	`++i` primero se incrementa el valor de la variable y luego se utiliza
		`4++`
		`a=5;`
		`b=a++;`
		`a=5;`
		`b=++a;`
			`5`
			
			`a vale 6 y b vale 5`
			
			`a vale 6 y b vale 6`

--	decremento	`4--`	`3`

Estos operadores suelen sustituir a veces al operador asignación y también suelen aparecer en bucles for.

```
/**
 *  Demostracion de los operadores incrementales
 *  A. Garcia-Beltran - Abril, 2008
 */
class opIncrementales {
  public static void main(String[] args) {
    int i,j;           //   Variables enteras. Podrian ser reales o char
    i = 7;
    System.out.println("* Operando entero:  i = " + i + ";");
    System.out.println(" Operador ++:       j = i++; ");
    j = i++;
    System.out.println("                    // i vale " + i + "; j vale " + j);
    i = 7;
    System.out.println("                    i = " + i + ";");
    System.out.println("                    j = ++i; ");
    j = ++i;
    System.out.println("                    // i vale " + i + "; j vale " + j);
    i = 7;
    System.out.println("* Operando entero:  i = " + i + ";");
    System.out.println(" Operador --:       j = i--; ");
    j = i--;
    System.out.println("                    // i vale " + i + "; j vale " + j);
    i = 7;
    System.out.println("                    i = " + i + ";");
    System.out.println("                    j = --i; ");
    j = --i;
    System.out.println("                    // i vale " + i + "; j vale " + j);
  }
```

Operadores Combinados

Tabla 4.4 Operadores aritméticos combinados

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+=	Suma combinada	a+=b	a=a+b
-=	Resta combinada	a-=b	a=a-b
*=	Producto combinado	a*=b	a=a*b
/=	División combinada	a/=b	a=a/b
%=	Resto combinado	a%=b	a=a%b

```
/**
 *  Demostracion de los operadores aritmeticos combinados
 *  A. Garcia-Beltran - marzo, 2008
 */
public class OpCombinados {
   public static void main(String[] args) {
      int i,j;   // Variables enteras. Podrian ser reales
      i = 7;
      j = 3;
      System.out.println("* Operandos enteros:   i = "+ i +" ;  j = "+ j);
      i += j;
      System.out.println("  Suma combinada:      i += j " + "   // i vale " + i);
      i = 7;
      i -= j;
      System.out.println("  Resta combinada:     i -= j " + "   // i vale " + i);
      i = 7;
      i *= j;
      System.out.println("  Producto combinado:  i *= j " + "   // i vale " + i);
      i = 7;
      i /= j;
      System.out.println("  Division combinada:  i /= j " + "   // i vale " + i);
      i = 7;
      i %= j;
      System.out.println("  Resto combinada:     i %= j " + "   // i vale " + i);
   }
}
```

Operadores de relación

Tabla 4.5 Operadores de relación

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
==	igual que	`7 == 38`	`false`
!=	distinto que	`'a' != 'k'`	`true`
<	menor que	`'G' < 'B'`	`false`
>	mayor que	`'b' > 'a'`	`true`
<=	menor o igual que	`7.5 <= 7.38`	`false`
>=	mayor o igual que	`38 >= 7`	`true`

Todos los valores numéricos que se comparan con NaN dan como resultado `false` excepto el operador `!=` que devuelve `true`. Esto ocurre incluso si ambos valores son NaN.

Ejemplo

```
 */
public class OpRelacionales {
   public static void main(String[] args) {
     int i,j;
     i = 7;
     j = 3;
     System.out.println("* Operandos enteros:          i = "+ i +" ;  j = "+ j);
     System.out.println("  Operador igualdad:          i == j es " + (i==j));
     System.out.println("  Operador desigualdad:       i != j es " + (i!=j));
     System.out.println("  Operador mayor que:         i > j  es " + (i>j));
     System.out.println("  Operador menor que:         i < j  es " + (i<j));
     System.out.println("  Operador mayor o igual que: i >= j es " + (i>=j));
     System.out.println("  Operador menor o igual que: i <= j es " + (i<=j));
   }
}
```

Operadores Lógicos

Tabla 4.6 Operadores booleanos

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
!	Negación - NOT (unario)	`!false` `!(5==5)`	`true` `false`
|	Suma lógica – OR (binario)	`true | false` `(5==5)|(5<4)`	`true` `true`
^	Suma lógica exclusiva – XOR (binario)	`true ^ false` `(5==5)|(5<4)`	`true` `true`
&	Producto lógico – AND (binario)	`true & false` `(5==5)&(5<4)`	`false` `false`
||	Suma lógica con cortocircuito: si el primer operando es `true` entonces el segundo se salta y el resultado es `true`	`true || false` `(5==5)||(5<4)`	`true` `true`
&&	Producto lógico con cortocircuito: si el primer operando es `false` entonces el segundo se salta y el resultado es `false`	`false && true` `(5==5)&&(5<4)`	`false` `false`


```
public class OpBooleanos {
   public static void main(String [] args) {
      System.out.println("Demostracion de operadores logicos");
      System.out.println("Negacion: ! false es       : " + (! false));
      System.out.println("          ! true es        : " + (! true));
      System.out.println("Suma:     false | false es : " + (false | false));
      System.out.println("          false | true es  : " + (false | true));
      System.out.println("          true | false es  : " + (true | false));
      System.out.println("          true | true es   : " + (true | true));
      System.out.println("Producto: false & false es : " + (false & false));
      System.out.println("          false & true es  : " + (false & true));
      System.out.println("          true & false es  : " + (true & false));
      System.out.println("          true & true es   : " + (true & true));
   }
}
```

Operadores Lógicos

Tabla 4.7 Operador condicional

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo

?:	operador condicional	`a = 4;`
		`b = a == 4 ? a+5 : 6-a;`
		`b = a > 4 ? a*7 : a+8;`
			`b vale 9`
			`b vale 12`

La sentencia de asignación:

valor = (expresionLogica ? expresion_1 : expresion_2);

como se verá más adelante es equivalente a:

```
 if (expresionLogica)
                valor = expresion_1;
else
                valor = expresion_2
```

```
public class opCondicional {
   public static void main(String[] args) {
      int i,j,k;
      i = 1;
      j = 2;
      k = i > j ? 2*i : 3*j+1;
      System.out.println("i = " + i);
      System.out.println("j = " + j);


      System.out.println("k = " + k);
      i = 2;
      j = 1;
      k = i > j ? 2*i : 3*j+1;
      System.out.println("i = " + i);
      System.out.println("j = " + j);
      System.out.println("k = " + k);
   }
}
```

Salida por pantalla del programa anterior:

```
i = 1
j = 2
k = 7
i = 2
j = 1
k = 4
```

Operadores de bit

Tabla 4.8 Operadores de bit

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
~	Negación ó complemento binario (unario)	~12	-13
|	Suma lógica binaria – OR (binario)	12|10	8
^	Suma lógica exclusiva – XOR (binario)	12^10	6
&	Producto lógico binario – AND (binario)	12&10	14
<<	Desplaza a la izquierda los bits del 1º operando tantas veces como indica el 2º operando (por la derecha siempre entra un cero)	7<<2 -7<<2	28 -28
>>	Desplaza a la derecha los bits del 1º operando tantas veces como indica el 2º operando (por la izquierda entra siempre el mismo bit más significativo anterior)	7>>2 -7>>2	1 -2
>>>	Desplaza a la derecha los bits del 1º operando tantas veces como indica el 2º operando – sin signo (por la izquierda entra siempre un cero).	7>>>2 -7>>>2	1 1073741822

```
public class OpBitEnteros2 {
  public static void main(String[] args) {
    int i,j;
    i = 12;
    j = 10;
    System.out.println("* Operandos enteros:        i = " + i + " ;  j = " + j);

    System.out.println("  Negacion o complemento:      ~i es " + (~i));
    System.out.println("  Suma logica (binaria):       i & j es " + (i&j));
    System.out.println("  Suma exclusiva (binaria):    i ^ j es " + (i^j));
    System.out.println("  Producto logico (binaria):   i | j es " + (i|j));
    i = 12;
    j = -10;
    System.out.println("* Operandos enteros:        i = " + i + " ;  j = " + j);
    System.out.println("  Negacion o complemento:      ~i es " + (~i));
    System.out.println("  Suma logica (binaria):       i & j es " + (i&j));
    System.out.println("  Suma exclusiva (binaria):    i ^ j es " + (i^j));
    System.out.println("  Producto logico (binaria):   i | j es " + (i|j));
    i = 7;
    j = 2;
    System.out.println("* Operandos enteros:        i = " + i + " ;  j = " + j);
    System.out.println("  Despl. a izquierdas:         i << j es " + (i<<j));
    System.out.println("  Despl. a derechas:           i >> j es " + (i>>j));
    System.out.println("  Despl. a derechas sin signo: i >>> j  es " + (i>>>j));

    i = -7;
    j = 2;
    System.out.println("* Operandos enteros:        i = " + i + " ;  j = " + j);
    System.out.println("  Desplazamiento a izquierdas: i << j es " + (i<<j));
    System.out.println("  Despl. a derechas:           i >> j es "+ (i>>j));
    System.out.println("  Despl. a derechas sin signo: i >>> j  es " + (i>>>j));
   }
}
```

Salida por pantalla del programa anterior:

```
* Operandos enteros:            i = 12 ;  j = 10
  Negacion o complemento:       ~i es -13
  Suma logica (binaria):        i & j es 8
  Suma exclusiva (binaria):     i ^ j es 6
  Producto logico (binaria):    i | j es 14
* Operandos enteros:            i = 12 ;  j = -10
  Negacion o complemento:       ~i es -13
  Suma logica (binaria):        i & j es 4
  Suma exclusiva (binaria):     i ^ j es -6
  Producto logico (binaria):    i | j es -2
* Operandos enteros:            i = 7 ;  j = 2
  Despl. a izquierdas:          i << j es 28
  Despl. a derechas:            i >> j es 1
  Despl. a derechas sin signo:  i >>> j  es 1
* Operandos enteros:            i = -7 ;  j = 2
  Desplazamiento a izquierdas:  i << j es -28
  Despl. a derechas:            i >> j es -2
  Despl. a derechas sin signo:  i >>> j  es 1073741822
```

Operadores concatenación de cadena

Tabla 4.9 Operador concatenación

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+	Operador concatenación	"Hola" + "Juan"	"HolaJuan"

Separadores

Tabla 4.10 Separadores en Java

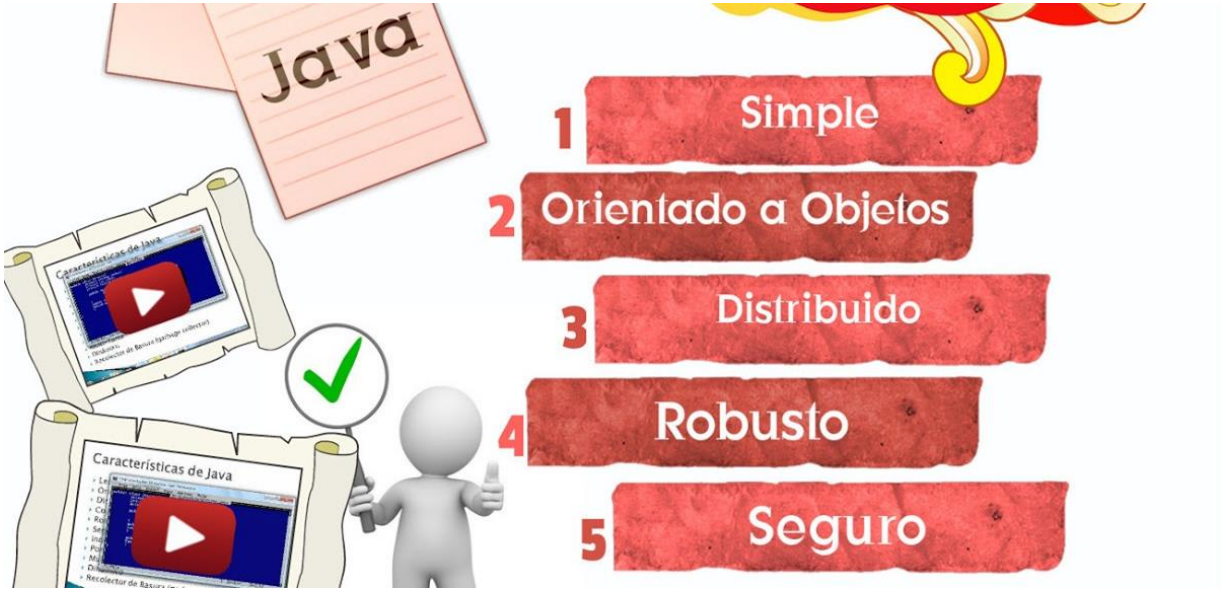
Separador	Descripción
`()`	Permiten modificar la **prioridad de una expresión**, **contener expresiones** para el control de flujo y realizar **conversiones de tipo**. Por otro lado pueden contener la **lista de parámetros** o argumentos, tanto en la definición de un método como en la llamada al mismo.
`{}`	Permiten definir **bloques de código** y ámbitos y contener los valores **iniciales** de las variables `array`
`[]`	Permiten declarar **variables de tipo `array`** (vectores o matrices) y referenciar sus elementos
`;`	Permite **separar sentencias**
`,`	Permite **separar identificadores** consecutivos en la declaración de variables y en las listas de parámetros. También se emplea para encadenar sentencias dentro de un **bucle `for`**
`.`	Permite **separar el nombre de un atributo o método** de su instancia de referencia. También separa el identificador de un **paquete** de los de los subpaquetes y clases

Prioridad entre operadores

Tabla 4.11 Prioridad de los operadores

Prior.	Operador	Tipo de operador	Operación
1	++ -- +, - ~ ¡	Aritmético Aritmético Aritmético Integral Booleano	Incremento previo o posterior (unario) Incremento previo o posterior (unario) Suma unaria, Resta unaria Cambio de bits (unario) Negación (unario)
2	(tipo)	Cualquiera	
3	*, /, %	Aritmético	Multiplicación, división, resto
4	+, - +	Aritmético Cadena	Suma, resta Concatenación de cadenas
5	<< >> >>>	Integral Integral Integral	Desplazamiento de bits a izquierda Desplazamiento de bits a derecha con inclusión de signo Desplazamiento de bits a derecha con inclusión de cero
6	<, <= >, >= instanceof	Aritmético Aritmético Objeto, tipo	Menor que, Menor o igual que Mayor que, Mayor o igual que Comparación de tipos
7	== ¡= == ¡=	Primitivo Primitivo Objeto Objeto	Igual (valores idénticos) Desigual (valores diferentes) Igual (referencia al mismo objeto) Desigual (referencia a distintos objetos)
8	& &	Integral Booleano	Cambio de bits AND Producto booleano
9	^ ^	Integral Booleano	Cambio de bits XOR Suma exclusiva booleana
10	| |	Integral Booleano	Cambio de bits OR Suma booleana
11	&&	Booleano	AND condicional
12	||	Booleano	OR condicional
13	? :	Booleano, cualquiera, cualquiera	Operador condicional (ternario)
14	= *=, /=, %= +=, -= <<=, >>= >>>= &=, ^=, |=	Variable, cualquiera	Asignación Asignación con operación

Características de Java



Mecanismos de creación de un programa

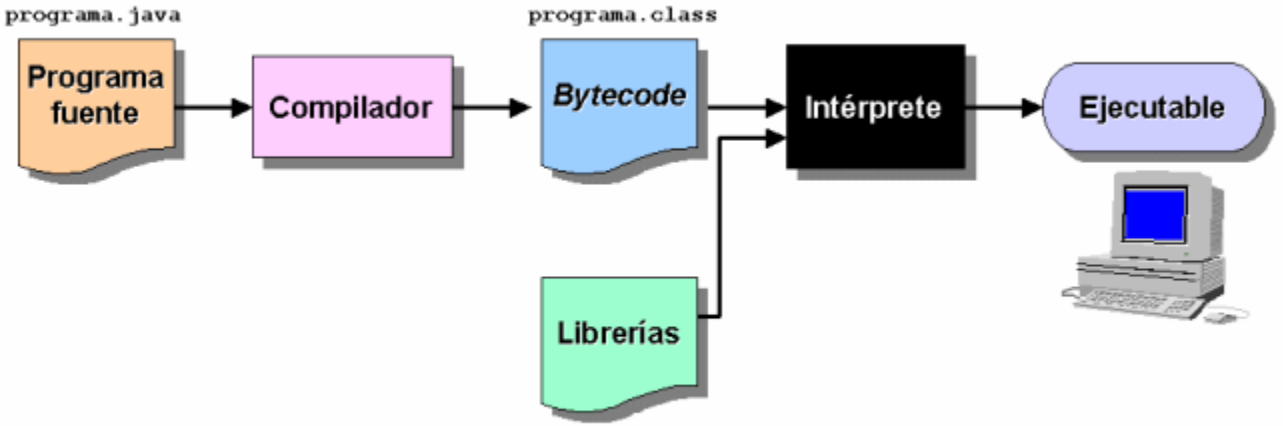
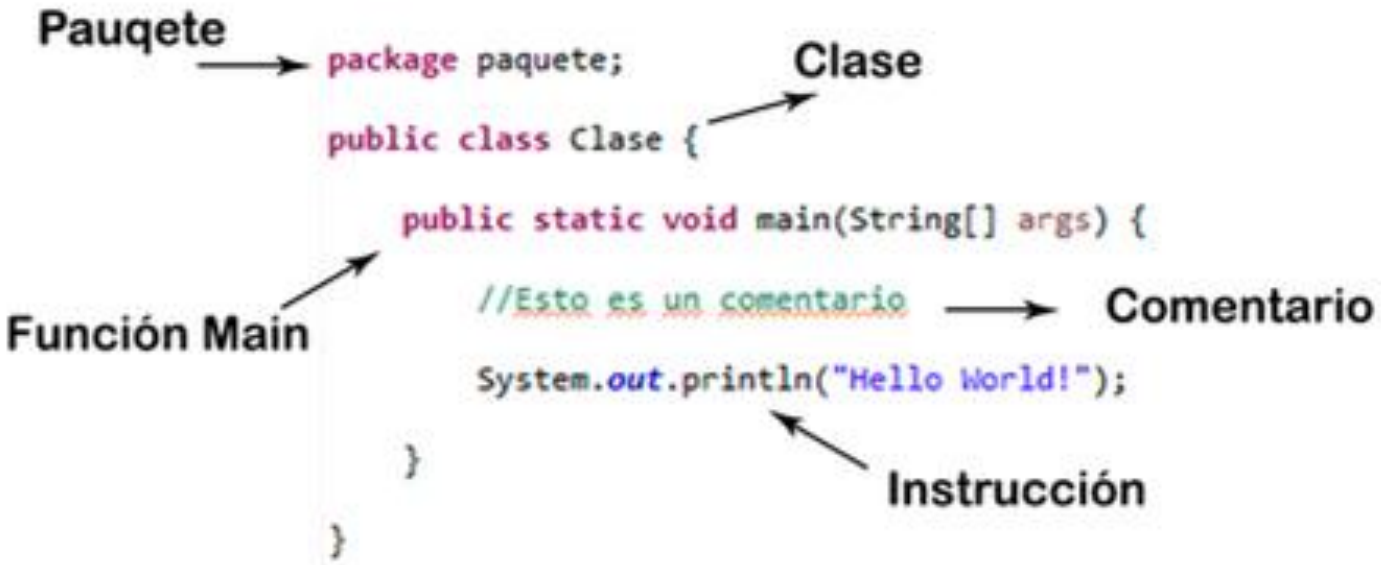


Figura 1.1. Esquema del proceso de creación de un programa con Java

Mecanismos de creación de un programa

- En este aspecto la principal originalidad de Java estriba en que es a la vez **compilado e interpretado**. Con el compilador de Java, el programa fuente con **extensión .java** es traducido a un lenguaje intermedio o pseudo-código (no es código máquina) llamado Java bytecodes generándose un programa compilado almacenado en un archivo con extensión **.class** . Este archivo puede ser posteriormente interpretado y ejecutado por el intérprete de Java (lo que se conoce como la Máquina Virtual Java o Java Virtual Machine). Por eso **Java es multi-plataforma**, ya que existe un intérprete para cada máquina diferente.

Estructura de Java

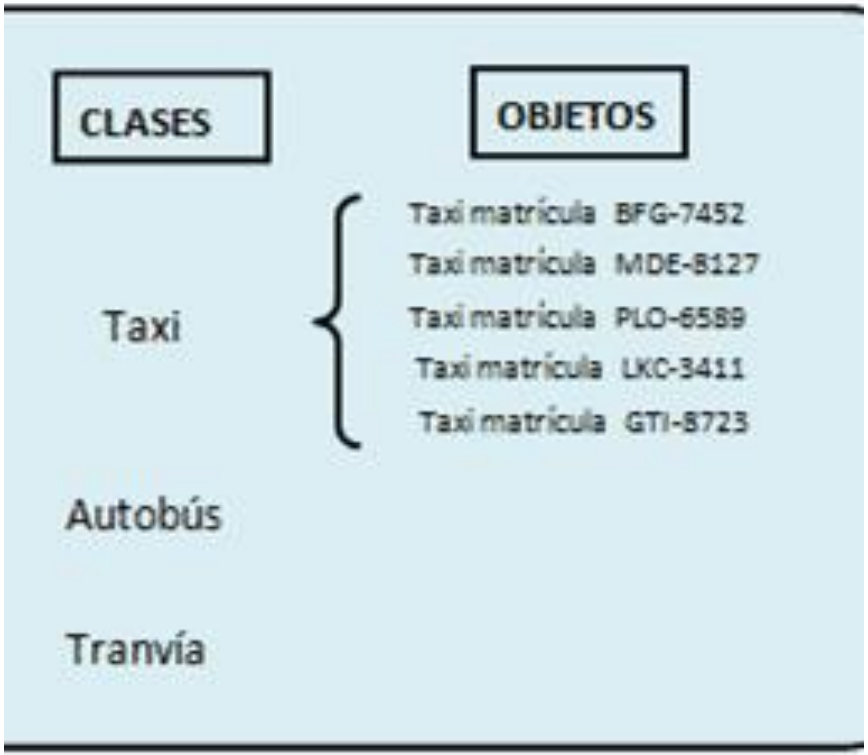


Estructura de Java

- Los paquetes en Java (**packages**) son la forma en la que Java nos permite **agrupar** de alguna manera lógica los componentes de nuestra aplicación que estén relacionados entre sí.
- Los paquetes permiten poner en su interior casi cualquier cosa como: **clases, interfaces, archivos de texto**, entre otros.
- De este modo, los paquetes en Java ayudan a darle una **buena organización** a la aplicación ya que permiten modularizar o categorizar las diferentes estructuras que componen nuestro software.

Clase en Java

- Una ***clase*** representa al conjunto de objetos que comparten una estructura y un comportamiento comunes.
- Una **clase** es una combinación específica de **atributos y métodos** y puede considerarse un tipo de dato de cualquier tipo **no** primitivo.
- Así, una **clase** es una especie de ***plantilla o prototipo*** **de objetos**: define los atributos que componen ese tipo de objetos y los métodos que pueden emplearse para trabajar con esos objetos.
- Aunque, por otro lado, una clase también puede estar compuesta por **métodos estáticos** que no necesitan de objetos (como los ejercicios realizados en clase que contienen un método estático **main**).

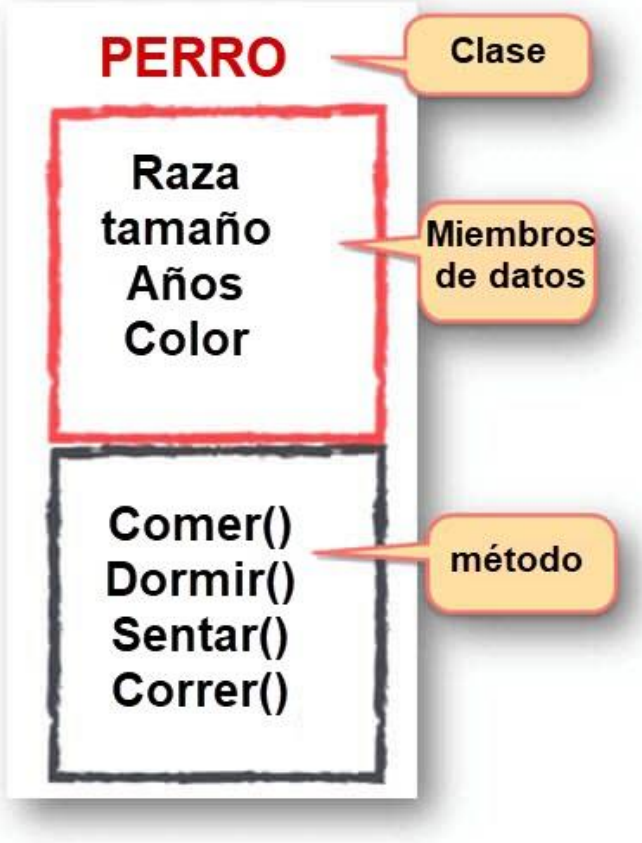

CLASES
OBJETOS
Taxi
Taxi matrícula BFG-7452
Taxi matrícula MDE-8127
Taxi matrícula PLO-6589
Taxi matrícula LKC-3411
Taxi matrícula GTI-8723
Autobús
Tranvía

Java

Clase: abstracción que define un tipo de objeto especificando qué propiedades (**atributos**) y operaciones disponibles va a tener.

Objeto: entidad existente en la memoria del ordenador que tiene unas propiedades (**atributos** o datos sobre sí mismo almacenados por el objeto) y unas operaciones disponibles específicas (**métodos**).

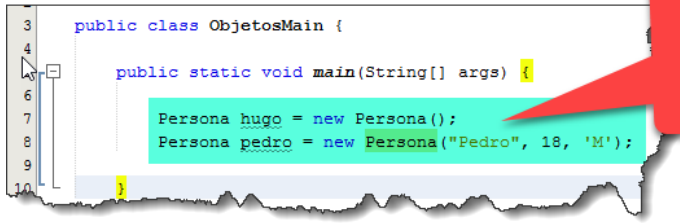
Java



Comentario

- Pueden ser colocados en cualquier parte de nuestro código en Java y comienzan por un doble **slash "//**
- Los comentarios multilínea en Java tal como el nombre lo indica nos permiten comentar varias líneas de nuestro código Java de manera mucho más sencilla en vez de esta añadiendo doble slash **"//" a cada línea.**
- Estos comentarios van cerrados entre **"/*" y "*/",** es decir comienzan donde se ponga "/*" y terminan donde esté el "*/". Estos comentarios funcionan de manera similar a los comentarios de una sola línea, pero deben tener **un comienzo y un final**. A diferencia de los comentarios de una sola línea, al poner el símbolo **"/*"** todo el código que haya tanto en la misma línea, como en las línea posteriores de este se convertirán en comentarios hasta que pongamos el "*/", de manera que si iniciamos un comentario de múltiples líneas, debemos cerrarlo, tal como sucede con las llaves o los corchetes en Java.

Función Main



Ha de ser público y estático

- **public**: Un método *público* es accesible desde fuera de la clase.
- **static**: Un método *estático* es aquel que se puede ejecutar sin una instancia de la clase.

Al ser el punto de entrada, ha de ser accesible desde fuera de la clase en la que se encuentra. Además, al ser lo primero que se ejecuta, ha de ser posible su ejecución antes de instanciar un objeto.

Además, ha de tener un tipo de devolución *void*

Como consecuencia directa de ser la primera línea de código que se ejecuta, no tiene sentido que tenga un tipo de devolución distinto de **void**, ya que no hay un código anterior que pueda hacer algo con ese valor. El método *main()* en **Java** siempre tiene un tipo de devolución **void**.

El método *main()* acepta un parámetro (y solo uno): una matriz de tipo **String**. Esta matriz recoge los valores que introduzcas a la hora de ejecutar tu aplicación desde la línea de comandos. Da igual el valor que introduzcas; el **JRE** lo transformará a **String**.

Cómo compilar archivos en Java

continuación y para aclarar todo esto, podréis ver un ejemplo de lo explicado:

```
public class Saludar {

    public static void main(String[] args) {

        //El código que iniciará nuestra aplicación

    }

}
```

En el ejemplo anterior podemos ver como se define nuestro método **main**, como vemos la clase se llama **Saludar**, así que el archivo que contendrá la clase **se deberá llamar Saludar.java**. La palabra **void** que va justo antes del nombre del método indica que este no devuelve ningún valor, así que **main** se limitará a ejecutar lo que tenga en su interior y punto.

Palabras Reservadas e Identificadores

En Java, como en cualquier otro lenguaje, existen una serie de palabras clave (keywords) que el usuario no puede utilizar como identificadores (nombres de variables, constantes y/o de funciones) y que tiene un significado especial para el compilador Java.

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

Identificador

Un identificador es un nombre que identifica a una variable, a un método o función miembro, a una clase.

Todos los lenguajes tienen ciertas reglas para componer los identificadores. En general diremos que un identificador está formado por una serie de caracteres que pueden ser letras, dígitos, guiones bajos(_) y signos de moneda ($), que no comiencen por un dígito ni tengan espacios.

- Java es sensible a mayúsculas y minúsculas (**case sensitive**) por lo que Valor y valor son diferentes identificadores.
- Además de estas restricciones, hay ciertas convenciones que hacen que el programa sea más legible, pero que no afectan a la ejecución del programa.
- La primera y fundamental es la de encontrar un nombre que sea significativo, de modo que el programa sea lo más legible posible. El tiempo que se pretende ahorrar eligiendo nombres cortos y poco significativos se pierde con creces cuando se revisa el programa después de cierto tiempo.

Tipo de identificador	Convención	Ejemplo
Nombre de una clase	Comienza por letra mayúscula	String Rectangulo CinematicaApplet
Nombre de función	Comienza con letra minúscula	calcularArea getValue setColor
Nombre de variable	Comienza por letra minúscula. Si tiene varias palabras, la primera letra en mayúscula de cada palabra excepto la primera palabra. La idea es destacar las palabras que lo forman.	area color appletSize
Nombre de constante	En letras mayúsculas	PI, MAX_ANCHO

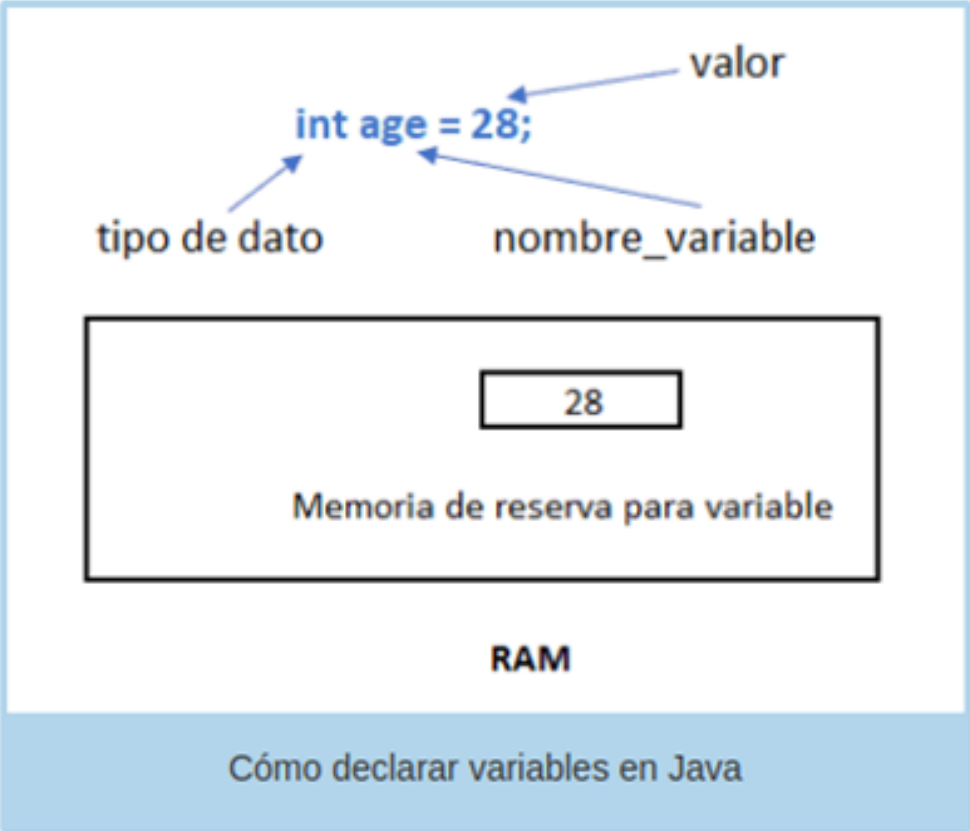
Variables en Java

Las variables son identificadores utilizados para **almacenar** un cierto **tipo de datos**. Cada variable se asocia con una determinada zona dentro de la memoria del computador. El **tamaño** de esta zona, en **bytes**, dependerá del tipo de datos del valor que se almacene en la variable. Las variables pueden contener diferentes valores durante la ejecución del programa.

Las **variables** en Java, se deben:

- **Declarar**
- **Inicializar**
- **Utilizar**

Todas las variables en Java, deben declararse antes de su uso.



Cómo declarar variables en Java

Ejemplo:

```
float simpleInterest; //Declarando variable float

int time = 10, speed = 20; //Declarando e Inicializando
la variable integer

 char var = 'h'; // Declarando e Inicializando la
 variable character
```

```
class PrimeraVariable
{
public static void main ( String[] args ) {
   String mensaje = "Valor inicial";
   System.out.println(mensaje);
   mensaje = "Valor modificado" ;
   System.out.println(mensaje);
 }
}
```

Tipos de datos enteros numéricos

Todos ellos emplean una representación que permite el almacenamiento de **números negativos y positivos**. El nombre y características de estos tipos son los siguientes:

- **byte**: como su propio nombre denota, emplea un solo byte (8 bits) de almacenamiento. Esto permite almacenar valores en el rango [-128, 127].
- **short**: usa el doble de almacenamiento que el anterior, lo cual hace posible representar cualquier valor en el rango [-32.768, 32.767].
- **int**: emplea 4 bytes de almacenamiento y es el tipo de dato entero más empleado. El rango de valores que puede representar va de -231 a 231-1.
- **long**: es el tipo entero de mayor tamaño, 8 bytes (64 bits), con un rango de valores desde -263 a 263-1.

Tipos de datos numéricos en punto flotante

Los tipos numéricos en punto flotante permiten representar números tanto muy grandes como muy pequeños además de números decimales. Java dispone de 2 tipos concretos en esta categoría:

- `float`: conocido como tipo de **precisión simple**, emplea un total de 32 bits. Con este tipo de datos es posible representar números en el rango de 1.4x10-45 a 3.4028235x1038.
- `double`: sigue un esquema de almacenamiento similar al anterior, pero usando 64 bits en lugar de 32. Esto le permite representar valores en el rango de 4.9x10-324 a 1.7976931348623157x10308.

Booleanos y caracteres

Aparte de los 6 tipos de datos que acabamos de ver, destinados a trabajar con números en distintos rangos, Java define otros dos tipos primitivos más:

- **boolean**: tiene la finalidad de facilitar el trabajo con valores "verdadero/falso" (booleanos), resultantes por regla general de evaluar expresiones. Los dos valores posibles de este tipo son true y false.
- **char**: se utiliza para almacenar caracteres individuales (letras, para entendernos). En realidad está considerado también un tipo numérico, si bien su representación habitual es la del carácter cuyo código almacena. Utiliza 16 bits y se usa la codificación UTF-16 de Unicode.

Los tipos de datos primitivos que acabamos de ver se caracterizan por poder almacenar un único valor. Salvo este reducido conjunto de tipos de datos primitivos, que facilitan el trabajo con números, caracteres y valores booleanos, todos los demás tipos de Java son **objetos**, también llamados **tipos estructurados** o "**Clases**".

Cadenas de caracteres

Aunque las cadenas de caracteres no son un tipo simple en Java, sino una instancia de la clase String, el lenguaje otorga un tratamiento bastante especial a este tipo de dato, lo cual provoca que, en ocasiones, nos parezca estar trabajando con un tipo primitivo.

Aunque cuando declaramos una cadena estamos creando un objeto, su declaración no se diferencia de la de una variable de tipo primitivo de las que acabamos de ver:

```
String nombreCurso = "Iniciación a Java";
```

Hay tres tipos de variables en Java:

- Variables locales
- Variables de instancia
- Variables estáticas

Variable local

Una variable definida dentro de un bloque, método o constructor se llama **variable local**.

- Estas variables se crean cuando el bloque ingresado o método se llama y destruye después de salir del bloque o cuando la llamada regresa del método.
- El alcance de estas variables solo existe dentro del bloque en el que se declara la variable, es decir, podemos acceder a estas variables solo dentro de ese bloque.

```
public class StudentDetails

{

    public void StudentAge()

    {   //variable local age

        int age = 0;

        age = age + 5;

        System.out.println("La edad del estudiante es : " + age);

    }
```

```
 public static void main(String args[])
     {
          StudentDetails obj = new StudentDetails();
          obj.StudentAge();
     }
 }
```

```
public class StudentDetails

{

    public void StudentAge()

    {   //variable local age

        int age = 0;

        age = age + 5;

    }

}
```


```
    public static void main(String args[])
     {
          //utilizando la variable local age fuera de su alcance
          System.out.println("La edad del estudiante es : " + age);
      }
```

3.2.Variables de instancia

Las variables de instancia son variables no estáticas y se declaran en una clase fuera de cualquier método, constructor o bloque.

- Como las variables de instancia se declaran en una clase, estas variables se crean cuando un objeto de la clase se crea y se destruye cuando se destruye el objeto.
- A diferencia de las variables locales, podemos usar especificadores de acceso para variables de instancia. Si no especificamos ningún especificador de acceso, se utilizará el especificador de acceso predeterminado.

```
class Employee{
    //define los campos (variables de instancia)
para el Empleado
    private String name;
    private String title;
    private String manager;
    //Otro código iría aqui
     }
```

Variables estáticas

Las variables estáticas también se conocen como **variables de clase.**

- Estas variables se declaran de forma similar a las variables de instancia, la diferencia es que las variables estáticas se declaran utilizando la palabra clave [java]static[/java] dentro de una clase fuera de cualquier constructor o bloque de métodos.
- A diferencia de las variables de instancia, **solo podemos tener una copia de una variable estática por clase**, independientemente de cuántos objetos creemos.
- Las variables estáticas se crean al inicio de la ejecución del programa y se destruyen automáticamente cuando finaliza la ejecución.

Para acceder a variables estáticas, no necesitamos crear ningún objeto de esa clase, simplemente podemos acceder a la variable como:

nombre_clase.nombre_variable;


```
import java.io.*;

class Emp {


  // salario como variable estatica

  public static double salary;

  public static String name = "Alex";

}
```

```
public class EmpDemo

{

     public static void main(String args[]) {



      //acceder a la variable estatica sin objeto

      Emp.salary = 1000;

      System.out.println(Emp.name + " tiene un salario promedio de: " +
Emp.salary);

   }

 }
```

```
public class PrimeraVariable {

    public static void main(String[] arg) {

        int numero1 = 3;

        int numero2 = 5;

        int suma;

        suma = numero1 + numero2;

        System.out.println("El resultado es: " + suma);

    }

}
```

Operadores Relacionales

- == Equal to
- != Not equal to
- > Greater than
- >= Greater than or equal to
- < Less than
- <= Less than or equal to

A continuación se incluyen algunos ejemplos de estos operadores aplicados a constantes:

- (2==1) // resultado=false porque la condición no se cumple
- (3<=3) // resultado=true porque la condición se cumple
- (3<3) // resultado=false porque la condición no se cumple
- (1!=1) // resultado=false porque la condición no se cumple

Operadores Condicionales

- `&&` Conditional-AND
- `||` Conditional-OR
- `?:` Ternary (shorthand for if-then-else statement)

```
if (x>y)
   mayor = x;
else
   mayor = y;
```

```
resultado = (condicion)?valor1:valor2;
```

```
mayor=(x>y)?x:y;
```

Operadores de asignación

Los operadores de asignación permiten asignar un valor a una variable. El operador de asignación más utilizado es el operador de igualdad (=), que no debe ser confundido con la igualdad matemática.

Su forma general es:

```
nombre_de_variable = expresión;
```

cuyo funcionamiento es como sigue: se evalúa expresión y el resultado se deposita en nombre_de_variable, sustituyendo cualquier otro valor que hubiera en esa posición de memoria anteriormente.

Una posible utilización de este operador es como sigue:

```
variable = variable + 1;
```

```
variable = variable + 1;
```

- Se toma el valor de variable contenido en la memoria, se le suma una unidad y el valor resultante vuelve a depositarse en memoria en la zona correspondiente al identificador variable, sustituyendo al valor que había anteriormente. El resultado ha sido incrementar el valor de variable en una unidad.
- Existen otros cuatro operadores de asignación (+=, -=, *= y /=) formados por los 4 operadores aritméticos seguidos por el carácter de igualdad. Estos operadores simplifican algunas operaciones recurrentes sobre una misma variable.

Su forma general es:

```
variable op= expresión;
```

donde op representa cualquiera de los operadores (+ - * /). La expresión anterior es equivalente a:

```
variable = variable op expresión;
```

- `distancia += 1; equivale a: distancia = distancia + 1;`
- `rango /= 2.0 equivale a: rango = rango /2.0`

Operadores lógicos

Los operadores lógicos son operadores binarios que permiten combinar los resultados de los operadores relacionales, comprobando que se cumplen simultáneamente varias condiciones, que se cumple una u otra, etc.

El lenguaje Java representa el AND(&&) el OR(||) y la negación (!), además de los operadores de lógica de bit de los que hablaremos más adelante. Su forma general es la siguiente:

```
expresión1 || expresión2

expresión1 && expresión2
```

El operador && devuelve un true si tanto expresión1 como expresión2 son verdaderas, y false en caso contrario, es decir si una de las dos expresiones o las dos son falsas;

por otra parte, el operador || devuelve true si al menos una de las expresiones es cierta

Los operadores && y || se pueden combinar entre sí, agrupados entre paréntesis, dando a veces un código de más difícil interpretación. Por ejemplo:

- **(2==1) || (-1==-1) // el resultado es true**
- **(2==2) && (3==-1) // el resultado es false**
- **((2==2) && (3==3)) || (4==0) // el resultado es true**
- **((6==6) || (8==0)) && ((5==5) && (3==2)) // el resultado es false**

En cuanto al operador negación lógica (!) devuelve false si se aplica a un valor true y devuelve true si se aplica a un valor false. Su forma general es:

!expresión

Reglas de precedencia de los operadores en Java

Operators	Precedence
postfix	`expr++ expr--`
unary	`++expr --expr +expr -expr ~ !`
multiplicative	`* / %`
additive	`+ -`
shift	`<< >> >>>`
relational	`< > <= >= instanceof`
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Los operadores con la prioridad más alta aparecen en la parte superior de la tabla. Los operadores con la prioridad más alta son evaluados antes que los de prioridad más baja. Los operadores en la misma línea tienen la misma prioridad.

Cuando tenemos operadores con la misma prioridad en la misma expresión, todos los operadores binarios de igual precedencia se evalúan de izquierda a derecha excepto los operadores de asignación que se evalúan de derecha a izquierda.

Estructuras de Selección. Sentencia IF

Esta sentencia de control permite ejecutar o no una sentencia simple o compuesta según se cumpla o no una determinada condición. Esta sentencia tiene la siguiente forma general:

```
if (condición) {
     sentencia;
}
```

Explicación:

Se evalúa la condición.

Si el resultado es true , se ejecuta sentencia; si el resultado es false, se salta sentencia

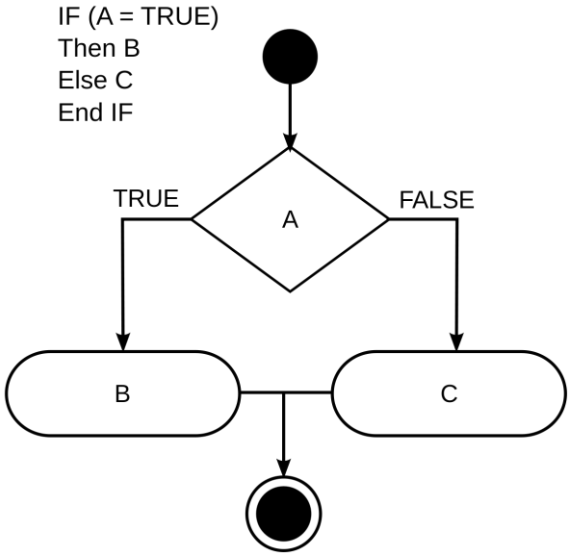
y se prosigue en la línea siguiente.

Hay que recordar que sentencia puede ser una sentencia simple o compuesta (bloque

{ ...}).

Estructuras de Selección.Sentencia if ... else

Esta sentencia permite realizar una bifurcación, ejecutando una parte u otra del programa según se cumpla o no una cierta condición.



```
if(condición) {
        sentencia_1;
}
else {
        sentencia_2;
}
```

1. Se evalúa expresión.
2. Si el resultado es true, se ejecuta sentencia_1 y se prosigue en la línea siguiente a sentencia_2; si el resultado es false, se salta sentencia_1, se ejecuta sentencia_2 y se prosigue en la línea siguiente.
3. Hay que indicar aquí también que sentencia_1 y sentencia_2 pueden ser sentencias simples o compuestas (bloques { ... }).

Estructuras de Selección. Sentencia if … else múltiple

Esta sentencia permite realizar una ramificación múltiple, ejecutando una entre varias partes del programa según se cumpla una entre n condiciones. La forma general es la siguiente:

```
if (expresión_1) {
}    sentencia_1;
else if (expresión_2){
     sentencia_2;
}
else if (expresión_3)
     sentencia_3;
}
else if (…) {
 ...
}
else {
     sentencia_n
}
```

- Se evalúa expresión_1.
- Si el resultado es true, se ejecuta sentencia_1.
- Si el resultado es false, se salta sentencia_1 y se evalúa expresión_2.
- Si el resultado es true se ejecuta sentencia_2, mientras que si es false se evalúa

expresión_3 y así sucesivamente.

- Si ninguna de las expresiónes o condiciones es true se ejecuta sentencia_n que es la opción por defecto.
- Todas las sentencias pueden ser simples o compuestas.

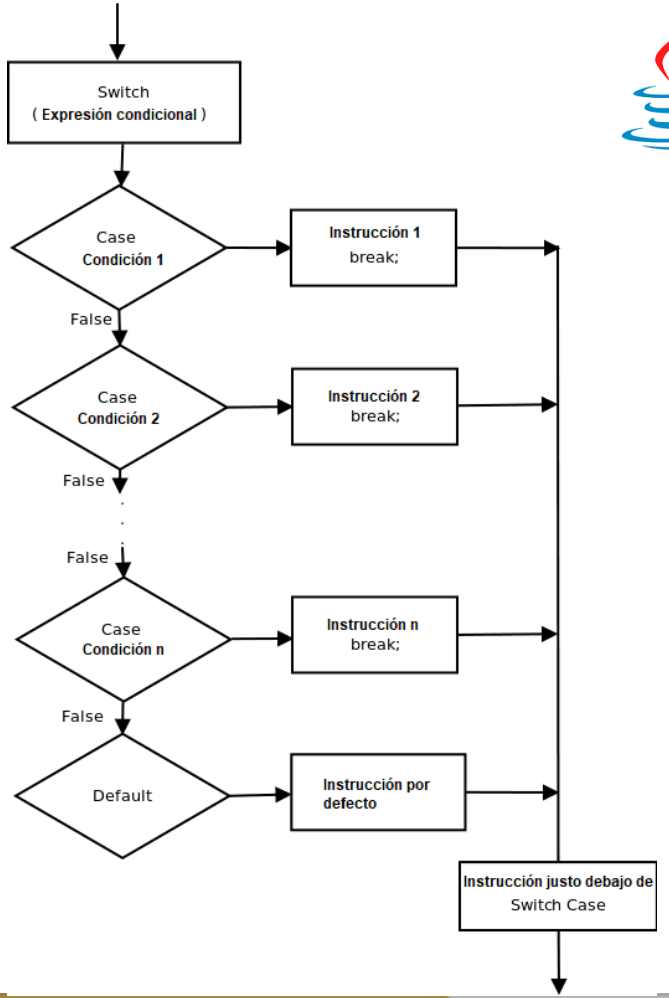
```
if (expresión_1) {
}    sentencia_1;
else if (expresión_2){
     sentencia_2;
}
else if (expresión_3)
     sentencia_3;
}
else if (…) {
  ...
}
else {
     sentencia_n
}
```

IF Anidados

```
if (a >= b) {
        if (b != 0.0)
        c = a/b;
}
else
        c = 0.0;
```

Sentencia switch

- La sentencia que se va a describir a continuación desarrolla una función similar a la de la sentencia if ... else con múltiples ramificaciones, aunque como se puede ver presenta también importantes diferencias.



La forma general de la sentencia switch es la siguiente:

```
switch (expresión o variable) {
  case const1:
       sentencia_1;
  case const2:
        sentencia_2;
  ...
  case constn:
         sentencia_n;
  [default: sentencia;]
}
```

```
        switch (day)
        {
            case 1:  dayString = "Lunes";
                     break;
            case 2:  dayString = "Martes";
                     break;
            case 3:  dayString = "Miercoles";
                     break;
            case 4:  dayString = "Jueves";
                     break;
            case 5:  dayString = "Viernes";
                     break;
            case 6:  dayString = "Sabado";
                     break;
            case 7:  dayString = "Domingo";
                     break;
            default: dayString = "Dia inválido";
                     break;
        }
        System.out.println(dayString);
    }
```

Estructuras repetitivas. Sentencia While

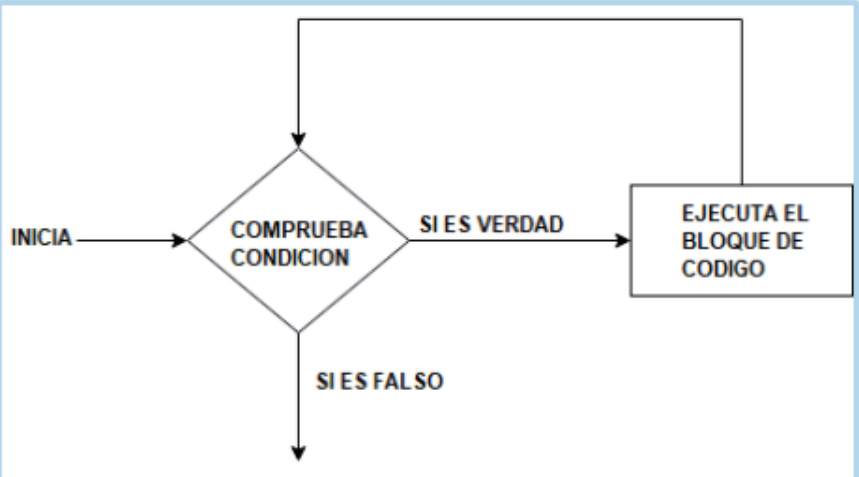


Diagrama de flujo: Bucle while en Java

Esta sentencia permite ejecutar repetidamente, mientras se cumpla una determinada condición, una sentencia o bloque de sentencias. La forma general es la siguiente:

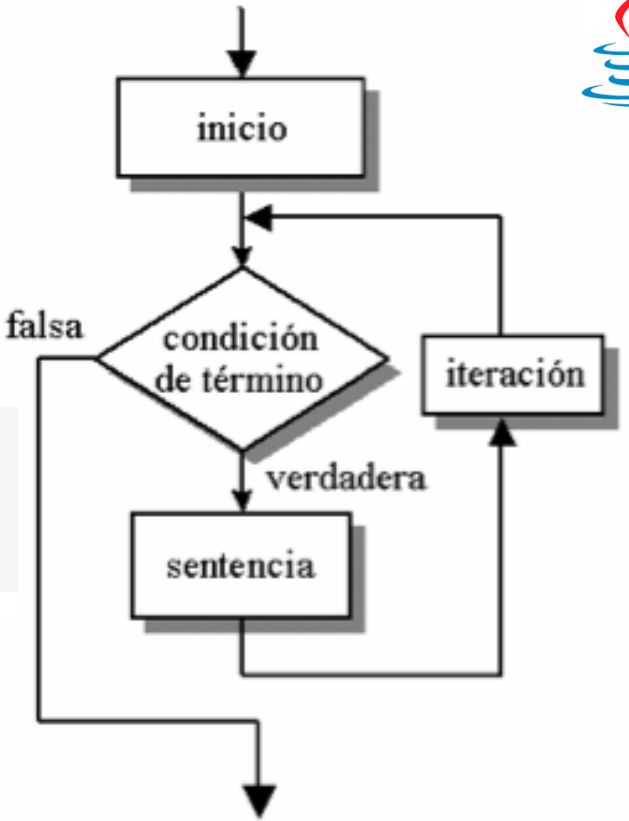
```
while (condición booleana)
{
   declaraciones del bucle ...
}
```

```java
public int leerNumero() {
    Scanner sc = new Scanner(System.in);
    int numero = -1;

    while (numero <= 0) {
        System.out.println("Introduce un numero positivo: ");
        numero = sc.nextInt();
    }

    sc.close();

    return numero;
}
```

Sentencia for

For es quizás el tipo de bucle más versátil y utilizado del lenguaje Java. Su forma general es la siguiente:

```
for (inicio; termino; iteracion)
    sentencia;
```



Sentencia do.. while

Esta sentencia funciona de modo análogo a while, con la diferencia de que la evaluación de expresión_de_control se realiza al final del bucle, después de haber ejecutado al menos una vez las sentencias entre llaves; éstas se vuelven a ejecutar mientras expresión_de_control sea true.

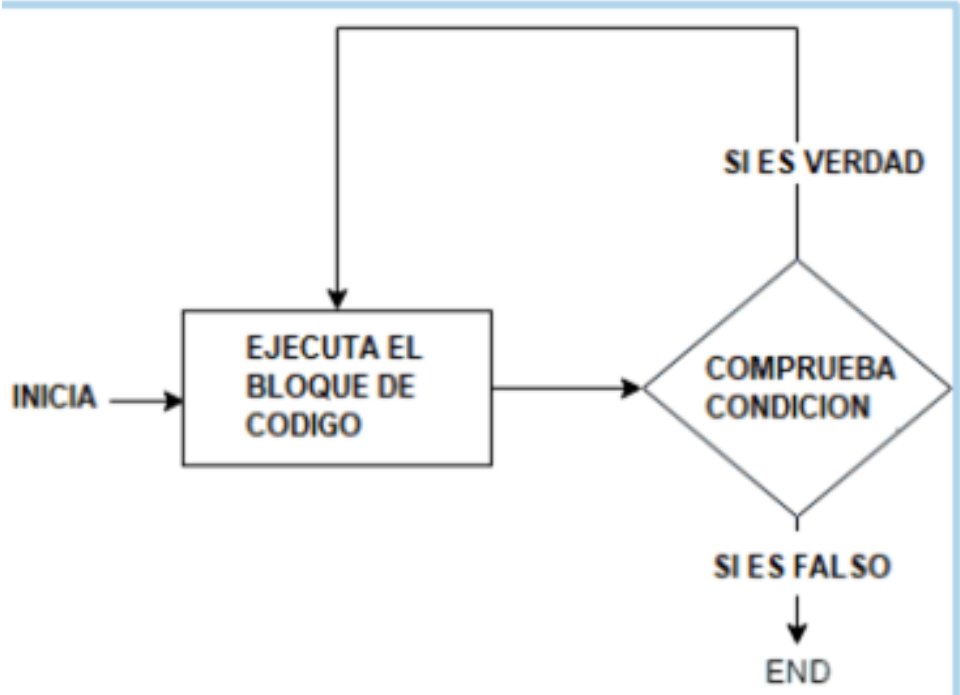


Diagrama de Flujo: Bucle do while en Java

La forma general de esta sentencia es:

```
do {

      sentencia;

} while(expresión_de_control);
```