3.- Fundamentos de programación 3

1. Tipos de datos

El tipo boolean.

Además de los tipos primitivos numéricos (byte, short, int, long, float y double) y del tipo char, Java dispone de un tipo más, el tipo ***boolean***. Este tipo permite representar únicamente dos valores: ***true*** y ***false***.

Al estudiar la sentencia **if** hemos visto como expresar **condiciones**. Las condiciones no son más que expresiones que al evaluarse producen un resultado de tipo boolean, es decir, true o a false.

Igual que con el resto de tipos, podemos definir variables de tipo boolean y asignarles valor, ya sea de forma literal (true o false), o utilizando alguna expresión del mismo tipo que la variable.

En el siguiente ejemplo se declara una variable de tipo boolean y se le asigna un valor (utilizando directamente el literal *false*)

```
boolean esFestivo;
esFestivo = false;
```

En este otro ejemplo, se declara una variable de tipo boolean y se le asigna el resultado de una expresión. Como ocurre con la asignación de otros tipos de variables, en primer lugar se evaluará la expresión y su resultado (true o false) se almacenará en la variable:

```
int edad = tec.nextInt();
boolean esMayorEdad;
esMayorEdad = edad >=18;
```

De la misma forma que en una expresión condicional utilizamos una condición, podemos utilizar una variable de tipo boolean

```
if(edad >= 18) ….
```

será equivalente a

```
boolean esMayorEdad = edad >=18;
if(esMayorEdad) …
```

También es posible definir métodos que devuelven un resultado de tipo boolean. Por ejemplo:

```
public static boolean puedeConducir(int edad) {
   boolean puede;
   if(edad >= 18) {
        puede = true;
   } else {
        puede = false;
   }
   return puede;
}
```

… y usar la llamada al método allí donde utilizaríamos una expresión boolean. Por ejemplo:

```
int edad =  ….
if(puedeConducir(edad) {
    …
}
```

Operadores de comparación o relacionales

Al estudiar la sentencia if ***ya vimos los operadores de comparación*** *(también llamados relacionales)*. Son operadores que permiten comparar dos expresiones del mismo tipo o de tipos compatibles y que producen un resultado de tipo boolean:

Los recordamos:

Operador	Nombre de la operación	Ejemplo
< > <= >= == !=	menor mayor menor o igual mayor o igual igual distinto	temperatura < 100 minuto > 6 temperatura <= 35.6f (a+b) >= 7 inicialNombre = = 'J' valor1 ! = (valor2 – valor3)

Operadores lógicos

Además de los operadores de comparación disponemos de otros operadores que permitirán escribir expresiones más complejas.

Los operadores lógicos, al igual que los de comparación, producen un resultado de tipo boolean. Pero además, sus operandos son también de tipo boolean.

Los operadores lógicos son los siguientes:

Operador	Descripción
&& &	**Y lógica** o AND Operador binario. Su resultado es true si los dos operandos son true. False en caso contrario
|| |	**O lógica** o OR Operador binario. Su resultado es true si uno de los dos operandos es true. False cuando ambos son false.
!	**Negación** Operador unario. Si el operando es true, el resultado es false (y viceversa). Es decir, el resultado es lo contrario que el operando
^	**O exclusiva** o XOR. El Operador binario. El resultado es true cuando solo uno de los operandos es true. False en caso contrario.

El funcionamiento de los operadores lógicos puede describirse utilizando **tablas de verdad**:

x	**y**	**x && y** **x & y**	**x || y** **x | y**	**! x**	**x ^ y**
true	true	true	true		false
true	false	false	true	false	true
false	true	false	true		true
false	false	false	false	true	false

Ejemplos:

Supongamos que las variables m y p contienen las edades de Miguel y Pablo (int m = 16, p = 21). ¿Cómo expresaríamos las siguientes condiciones?

¿Son los dos mayores de edad?

```
m>=18 && p>=18 // false
```

¿Alguno de ellos es mayor de edad?

```
m>=18 || p>=18 // true
```

¿Sólo uno de ellos es mayor de edad?

```
m>=18 ^ p>=18 // true
(m>=18 && p<18) || (m<18 && p>=18) // true
```

Operadores cortocircuitados (&& y ||) vs no cortocircuitados (& y |).

Cuando en una expresión aparecen los operadores ***cortocircuitados &&*** o **||**, solo se evaluará la parte de la expresión que sea absolutamente necesaria.

En el siguiente ejemplo, solo se evaluará la expresión a>100. Como esa parte de la expresión da false como resultado, no es necesario evaluar el resto de la expresión para averiguar que la condición del if se evalua a false.

```
int a = 3, b = 3;
if(a > 100 && b == 3) ….
```

Cuando en una expresión aparecen los operadores ***no cortocircuitados &*** o **|**, se evaluará toda la expresión, aunque no sea absolutamente necesario para calcular su resultado.

En el mismo ejemplo, con operadores no cortocircuitados, se evaluaría la expresión al completo.

```
int a = 3, b = 3;
if(a > 100 & b == 3) ….
```

2. Operadores

Operadores de asignación combinados

En Java existen una serie de operadores que combinan una ***operación*** aritmética con una ***asignación***.

Operador	Equivale a
`a += b ;`	`a = a + b ;`
`a -= b ;`	`a = a – b ;`
`a *= b ;`	`a = a * b ;`
`a /= b ;`	`a = a / b ;`
`a %= b ;`	`a = a % b ;`

Operadores de incremento y decremento

Se trata de operadores unarios que incrementan / decrementan en uno la variable a la que acompañan.

variable ++;

equivale a

variable = variable + 1;

variable --;

equivale a

variable = variable – 1;

Observa el ejemplo:

```
int cont = 0;
while( cont < 5) {
   System.out.println("Hola mundo");
   cont ++; // equivale a cont = cont + 1;
}
```

El operador (++ o --) se puede poner delante o detrás de la variable. Si el incremento o decremento es lo único que aparece en la expresión, no importa que el operador vaya antes o después de la variable.

```
int cont = 0;
while( cont < 5) {
   System.out.println("Hola mundo");
   ++ cont; // equivale a cont = cont + 1;
}
```

Pre-incremento vs Post-incremento (ídem para decremento)

Como hemos dicho, cuando el incremento aparece solo en una instrucción, tiene el mismo efecto que vaya antes o después de la variable

x++; es lo mismo que ++x;

Sin embargo, cuando forma parte de una expresión más compleja, no es lo mismo x++ que ++x :

Si el incremento aparece delante de la variable, se denomina ***PREINCREMENTO***. En primer lugar se incrementa la variable, y luego se usa la variable en la expresión:

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

equivale a

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

Si el incremento aparece delante de la variable, se denomina ***POSTINCREMENTO***. En primer lugar se utiliza la variable en la expresión y, para finalizar, se incrementa el valor de la variable.:

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

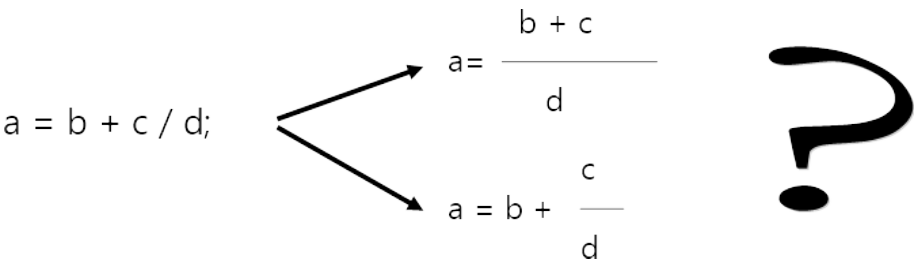
equivale a

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

Precedencia de operaciones

Cuando en una expresión aparecen varios operadores, el orden en que se ejecutan es el que se define en la siguiente tabla:

Descripción	Operadores
1º	( )
2º	++op --op -op +op !
3º	* / %
4º	+ -
5º	> >= < <=
6º	== !=
7º	&
8º	^
9º	|
10º	&&
11º	||
12º	?: (operador ternario)
13º	+= -= *= /= %= &= |= ^=



Cuando aparecen dos o más operadores que tienen la misma prioridad, se evalúan de izquierda a derecha

a = b / c * d

equivale a

a = (b / c) * d;

3. Estructuras de control

La sentencia switch (alternativa múltiple)

La sentencia switch permite ejecutar uno de entre varios bloques de código en función del valor que toma una expresión:

```
switch (expresion) {
    case valor1:
        instrucciones
        break;
    case  valor2:
        instrucciones
        break;
    case  valor3:
        instrucciones
        break;
    ...
    ...
    default:
        instrucciones
}
```

switch funciona igual que un if … else if … que haga comparaciones de igualdad:

```
if ( expresion == valor1) {
    instrucciones
} else if (expresión == valor2) {
    instrucciones
} else if (expresión == valor3) {
    Instrucciones
    ...
    ...
} else {
    instrucciones
}
```

La expresión (y los valores) debe ser de alguno de los siguientes tipos: byte, short, int, char, String. También, de alguno de los siguientes tipos que aún no hemos estudiado: Byte, Short, Integer, Character.

switch solo realiza comparaciones de igualdad. No se pueden comparar rangos de valores.

Ejemplo

```
System.out.print("Numero del día de la semana");
int dia = tec.nextInt();
switch (dia) {
    case 1: System.out.println("lunes");
            break;
    case 2: System.out.println("martes");
            break;
    case 3: System.out.println("miercoles");
            break;
    case 4: System.out.println("jueves");
            break;
    case 5: System.out.println("viernes");
            break;
    default: System.out.println("fin de semana");
}
```

La sentencia ***default*** es opcional. Si la expresión no coincide con ninguno de los valores de los case, entonces se ejecutan las instrucciones asociadas a default

```
System.out.print("Numero del día de la semana");
int dia = tec.nextInt();
switch (dia) {
    case 1: System.out.println("lunes");
            break;
    case 2: System.out.println("martes");
            break;
    case 3: System.out.println("miercoles");
            break;
    case 4: System.out.println("jueves");
            break;
    case 5: System.out.println("viernes");
            break;
    case 6: System.out.println("sabado");
            break;
    case 7: System.out.println("domingo");

}
```

El ***case*** cuyo valor coincida con la expresión marcará el inicio de la ejecución. La sentencia ***break***, que también es opcional, hace que la ejecución salte hasta el final del switch, una vez mostrado el nombre del día.

Observa el siguiente ejemplo en el que no se utiliza la sentencia ***break;***

```
System.out.print("Numero del día de la semana");
int dia = tec.nextInt();
switch (dia) {
     case 1: System.out.println("lunes");
     case 2: System.out.println("martes");
     case 3: System.out.println("miercoles");
     case 4: System.out.println("jueves");
     case 5: System.out.println("viernes");
     case 6: System.out.println("sabado");
     case 7: System.out.println("domingo");
}
```

Si el usuario introduce, por ejemplo, el valor 5, se mostrará por pantalla

```
viernes
sabado
domingo
```

Es decir, se comienza a ejecutar por el case 5 y se continua hasta que finaliza la sentencia switch.

También es posible dejar sin instrucciones algún ***case*** *para agruparlos.* Observa el siguiente ejemplo:

```
System.out.println("¿Por qué letra empieza tu nombre?")
char letra = tec.next().charAt(0);
System.out.println("Otros nombres que empiezan con la vocal " + letra
       + " son :")
switch (letra) {
      case 'a':
      case 'A': System.out.println("Ana, Andrés, Adrián");
             break;

      case 'e':
      case 'E': System.out.println("Emilio, Ernesto");
             break;

      case 'i':
      case 'I': System.out.println("Ismael, Isabel");
              break;

      case 'o':
      case 'O': System.out.println("Onofre, Osorio");
              break;

      case 'u':
      case 'U': System.out.println("ürsula, Umberto");
              break;

      default:
            System.out.println("Tu nombre no empieza con una vocal");
}
```

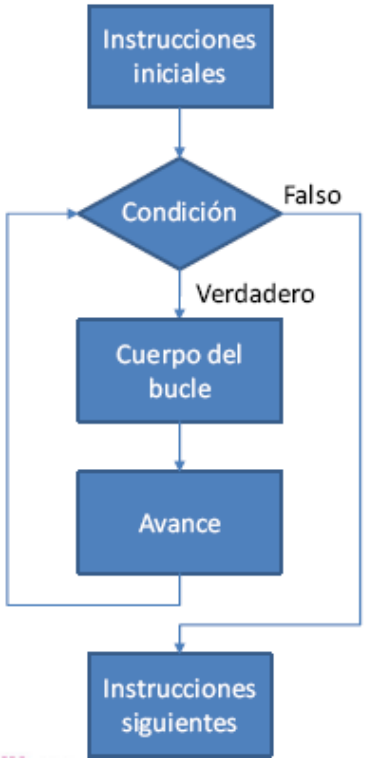
La sentencia repetitiva for

La **sintaxis** es la siguiente:

for (inicio ; condición; avance) {

instrucciones

}



Donde

- Inicio:
  - es una instrucción o varias instrucciones separadas por comas.
  - Puede incluir declaración de variables, en cuyo caso, solo serán válidas dentro de la sentencia for.
- Condición: será cualquier expresión booleana (es decir, cuyo resultado sea true o false).
- Avance
  - Es una instrucción o varias instrucciones separadas por comas.

Funcionamiento:

1.- Se ejecutan las instrucciones de inicio.

2.- Se evalúa la condición.

3.- Si la condición es verdadera

Se ejecuta la guarda del bucle.

Se ejecutan las instrucciones de avance.

Se vuelve al paso 2

4.- Si la condición es falsa se continúa con las instrucciones que haya a continuación del bucle

La sentencia for se suele utilizar para construir bucles en los que el número de repeticiones se conoce o se puede calcular de antemano. Por ejemplo: Pedir al usuario x números, contar el número de vocales que contiene una frase, …

La sentencia repetitiva do-while

La **sintaxis** es la siguiente:

do {

instrucciones

} while (condición***) ;***

Funcionamiento:

1.- Se ejecuta el cuerpo del bucle.

2.- Se evalúa la condición.

3.- Si la condición es verdadera se vuelve al paso 1

4.- Si la condición es falsa se continua con las instrucciones que haya tras el bucle.

Dado que el cuerpo se ejecuta antes de evaluar la guarda, el cuerpo se ejecutará al menos una vez. Se suele utilizar cuando el cuerpo del bucle se tiene que ejecutar antes de poder decidir si se quiere repetir de nuevo.
Ejemplo: Leer un número positivo desde teclado

```
int edad;
do {
    System.out.println("Introduce tu edad: ");
    edad = tec.nextInt();
    if(num <= 0) {
        System.out.println("Error");
    }
} while(num<=0);
System.out.println("Tienes "  + edad + " años" );
```

Observa:

- Primero se lee el número. Si es incorrecto se repite el bucle.
- La variable edad, la declaramos antes del bucle (fuera), para poder usarla dentro del bucle y después del bucle.
- La sentencia lleva punto y coma al final