Tipos de operadores booleanos y switch

Operadores y Expresiones

Binarios

Aritméticos		Relacionales		Lógicos	
+	Suma en números y concatenación en Strings	==	Igual	&&	y lógico
-	Resta	<	Menor	||	o lógico
*	Multiplicación	<=	Menor o igual	Especiales	
/	División real	>	Mayor	=	Asignación
%	Resto o módulo	>=	Mayor o igual	instanceof	Objeto es de tipo
		!=	Distinto		

Operadores lógicos

Como operadores lógicos se han visto el operador and **&&** y el operador or **||**

Estos dos operadores son operadores **cortocircuitados**, lo que quiere decir que:

- Si en un and **&&** el primer operando es falso no se evalúa el segundo operando.
- Si en un or **||** el primer operando es true no se evalúa el segundo operando.

Operadores lógicos

Además de estos operadores existen el operador **&** y el operador **|** (**no cortocircuitados**)

En estos casos siempre se evalúan los dos operandos.

Operadores lógicos

¿Cual es la ventaja de uno con respecto a otro?

Si, por ejemplo tenemos un if en el que comprobemos si dos números son divisibles entre sí:

if(num1%num2==0)

En el caso en que num2 sea 0 tendríamos una división por 0, lo que nos da un error (llamado ***ArithmeticException***). Eso lo podemos solucionar con el && para que sólo haga la operación si el valor de num2 es distinto de 0:

if(num2!=0 && num1%num2==0)

Operadores lógicos

El uso de los operadores no cortocircuitados (**&** y **|**) debería ser en el caso en que queramos que siempre se evalúen los dos operandos independientemente del resultado de su operación.

Por ejemplo, en una condición de un while:

```
boolean encontrado=false;
int intentos=4;
while(!encontrado & (intentos-- > 0) )
```

En este caso, además de la condición booleana que nos mantiene en el bucle se quiere que se disminuya el valor de intentos.

Switch con Expresiones

Anteriormente hemos visto el uso del **switch** tradicional. A partir de la versión 12 de Java se introdujo una nueva versión de switch que sí permite el uso del mismo como expresión o como declaración.

Con este switch se varía la sintaxis para no tener que usar el break para interrumpir la ejecución de los casos.

```
switch (expresión) {
    case valor1 -> expresión1;
    case valor2 -> expresión2;
    default -> expresión3;
}
```

Switch con Expresiones

También podemos usar bloques de código en lugar de expresiones simples:

```
switch (expresión) {
    case valor1 -> {
        // ...
    }
    case valor2 -> {
        // ...
    }
    default -> {
        // ...
    }
}
```

Switch con Expresiones

En este caso podemos usar el switch para devolver un valor:

```
int dia = 3;
String nombreDia = switch (dia) {
    case 1 -> "Lunes";
    case 2 -> "Martes";
    case 3 -> "Miércoles";
    case 4 -> "Jueves";
    default -> "Día no válido";
};
System.out.println(nombreDia); // "Miércoles"
```

Switch con Expresiones

Con este switch podemos agrupar casos dentro de un mismo case:

```
String estacion = "verano";
int diasMes = switch (estacion) {
    case "invierno", "otoño" -> 30;
    case "verano", "primavera" -> 31;
    default -> 28; // febrero
};
```

Este último ejemplo, visto con el switch tradicional quedaría más extenso, aunque se puede realizar:

```
String estacion = "verano";
int diasMes;

switch (estacion) {
    case "invierno":
    case "otoño":
        diasMes = 30;
        break;
    case "verano":
    case "primavera":
        diasMes = 31;
        break;
    default:
        diasMes = 28; // febrero
        break;
}

System.out.println("Días del mes: " + diasMes);
```