Switch

El **"switch"** es una *estructura de control que nos permite seleccionar uno de varios bloques de código para ejecutar*, dependiendo del valor de una expresión o variable.

La expresión dentro del *switch* se evalúa y se compara con los casos definidos dentro de él. Cada caso representa una opción diferente, y se ejecutará el bloque de código correspondiente si el valor de la expresión coincide con el caso.

Para comprenderlo mejor, veamos el siguiente ejemplo:

```
public static void main(String[] args) {
        //Sentencias que se ejecutan antes del "switch".
        switch (opcion) {
            case 1:
                //Sentencias que se ejecutan si "opción" tiene el "valor 1".
                System.out.println("Seleccionaste la opción 1");
                break; //Palabra clave "break" para salir del "switch".
            case 2:
                //Sentencias que se ejecutan si "opción" tiene el "valor 2".
                System.out.println("Seleccionaste la opción 2");
                break;
            case 3:
                //Sentencias que se ejecutan si "opción" tiene el "valor 3".
                System.out.println("Seleccionaste la opción 3");
                break;
            default:
                //Sentencias que se ejecutan si "opción" no coincide con el
                valor de ningún "case".
                System.out.println("Opción inválida");
                break;
        }
        //Sentencias que se ejecutan después del "switch".
    }
```

Ten en cuenta que *"opcion"* puede ser una variable de tipo texto también. La palabra reservada *"break"* se utiliza para salir del switch después de ejecutar el

bloque correspondiente a un caso. Sin el "break", el código continuaría ejecutando los casos siguientes, incluso si no coinciden con el valor de la expresión.

Aquí te compartimos otro ejemplo donde podemos agrupar varios casos para ejecutar la misma línea de código:

```
public static void main(String[] args) {
        switch (opcion) {
            case "A":
            case "B":
            case "C":
                System.out.println("Las opciones A, B y C están
                deshabilitadas");
                break;
            case "D":
                System.out.println("Seleccionaste la opción D");
                break;
            case "E":
                System.out.println("Seleccionaste la opción E");
                break;
            default:
                System.out.println("Opción inválida");
                break;
        }
    }
```

Switch expression

“Switch expression” es una forma nueva y mejorada de utilizar el switch que se introdujo en Java 12. Esta nueva característica *proporciona una sintaxis más flexible y mejorada que permite evaluar diferentes tipos de datos, incluyendo cadenas de texto (strings), y utilizar expresiones más complejas*.

Veamos un ejemplo:

```
public static void main(String[] args) {
    //Sentencias que se ejecutan antes del "switch".
     switch (opcion) {
        case 1  -> System.out.println("Seleccionaste la opción 1");
        case 2  -> System.out.println("Seleccionaste la opción 2");
        case 3  -> System.out.println("Seleccionaste la opción 3");
        default -> System.out.println("Opción inválida");
```

```
    }
   //Sentencias que se ejecutan después del "switch".
}
```

En este nuevo enfoque, la sintaxis es más concisa y *no se requiere el uso de la palabra reservada "break"*. Cada caso se representa con una *flecha (->)* seguida del *bloque de código* que se ejecutará si el valor de la expresión coincide con el caso (“case”).

Es importante tener en cuenta que en el Switch Expression, si deseas incluir varias líneas de código para un caso determinado, debes utilizar *llaves ({})* para agruparlas, como se muestra en el siguiente ejemplo:

```
public static void main(String[] args) {
        //Sentencias que se ejecutan antes del "switch".
        switch (opcion) {
            case 1  -> {
                System.out.print("Seleccionaste la opción:");
                System.out.print(" 1");
           }
            case 2  -> System.out.println("Seleccionaste la opción 2");
            case 3  -> System.out.println("Seleccionaste la opción 3");
            default -> System.out.println("Opción inválida");
        }
        //Sentencias que se ejecutan después del "switch".
    }
```

En este caso, el *“case 1”* tiene varias líneas de código y se agrupan utilizando *llaves ({})* para delimitar el bloque que se ejecutará.

Switch como expresión

Como su nombre lo indica, **el switch como expresión es un nuevo enfoque que permite que el switch funcione como una expresión**, lo cual significa que puede calcular directamente un valor. *Antes de esta actualización, el switch solo se podía utilizar como una declaración.*

Para comprender esta diferencia, analicemos cómo se modificaría el valor de una variable utilizando el switch statement tradicional y cómo se haría con el switch como expresión.

Imaginemos que deseamos obtener la cantidad de días de un mes a partir de su nombre. Con el switch statement tradicional, tendríamos que declarar la variable antes del switch y asignarle un valor dentro de cada case:

```
public static void main(String[] args) {
        String mes = "February";
        int numeroDeDias;
        switch (mes) {
            case "February":
                numeroDeDias = 28;
                break;
            case "April":
            case "June":
            case "September":
            case "November":
                numeroDeDias = 30;
                break;
            case "January":
            case "March":
            case "May":
            case "July":
            case "August":
            case "October":
            case "December":
                numeroDeDias = 31;
                break;
            default:
                numeroDeDias = 0;
        }
        System.out.println(mes+" tiene "+numeroDeDias+" días.");
}
```

En cambio, con el switch como expresión, podemos asignar directamente el resultado del switch a la variable, sin necesidad de declararla previamente:

```
public static void main(String[] args) {
        String mes = "February";
        int numeroDeDias = switch (month) {
            case "February" -> 28;
            case "April", "June", "September", "November" -> 30;
            case "January", "March", "May", "July", "August", "October", "December" ->
31;
            default -> 0;
        };
```

```
        System.out.println(mes+" tiene "+numeroDeDias+" días.");
}
```

En este último caso, utilizamos el switch como expresión para asignar directamente el resultado del cálculo de la cantidad de días a la variable "numeroDeDias". Esto nos permite simplificar el código y hacerlo más conciso.

Yield

Al utilizar bloques de código en una *expresión switch* para manejar múltiples líneas de código, **se emplea la palabra clave "yield" para indicar el valor de retorno del case**.

A continuación, te mostramos un ejemplo de código que utiliza esta estructura:

```
public static void main(String[] args) {
        String position = "director";
        boolean alcanzoObjetivos = true;
        double bonus = switch (position) {
            case "temporal" -> 50;
            case "empleado" -> 1000;
            case "director" -> {
                double bonusBase = 2000;
                double bonusPorRendimiento = alcanzoObjetivos ? 500 : 0;
                double bonusTotal = bonusBase + bonusPorRendimiento;
                yield bonusTotal;
            }
            default -> 0;
        };
        System.out.println("El bonus del "+position+" es $"+bonus);
}
```

En el programa, se define una variable *"position"* con el valor *"director"* y una variable booleana *"alcanzoObjetivos"* con el valor *verdadero*. Luego, se realiza una *expresión switch* basada en la variable *"position"*. En el *case "director"*, se realiza un cálculo de bonificación que incluye un bono base y un bono adicional basado en si se alcanzaron los objetivos. El resultado se asigna a la variable *"bonusTotal"* y se utiliza la *palabra clave "yield"* para devolver dicho valor. En caso de no coincidir con ningún *case*, se asigna el valor 0. Finalmente, se imprime en pantalla el resultado del cálculo del bonus correspondiente a la posición *"director"*.

Exhaustividad

La **exhaustividad** en una *expresión switch* es un concepto fundamental que implica cubrir todos los casos (“cases”) posibles. Si conoces de antemano todos los valores que puede tomar tu variable, es necesario tener un case correspondiente para cada uno de ellos. En caso de no conocer todos los posibles valores, es recomendable incluir una *cláusula default* para manejar cualquier valor no previsto.

> 💡 *Es importante destacar que este principio es especialmente relevante al trabajar con tipos de datos como **int** o **string**, los cuales pueden tener un rango de valores amplio o indefinido.*

Ahondaremos más sobre este tema cuando veamos estructuras de datos con un rango de valores más acotado, como los Enums.

Coincidencia de patrones

En las versiones más recientes de Java, se ha introducido una característica muy útil conocida como **"Pattern matching for switch" (coincidencia de patrones para switch)**.

Esta funcionalidad **proporciona una forma más eficiente y legible de manejar diferentes tipos de objetos en nuestros programas**, lo que puede mejorar significativamente nuestro código. Sin embargo, es importante tener en cuenta que para comprender y utilizar eficazmente esta característica, es necesario tener un buen entendimiento de otros conceptos de programación orientada a objetos en Java.

Aunque no profundizaremos en el *“pattern matching for switch”* en este momento, es relevante destacar su existencia y su estudio detallado más adelante en el curso, una vez que hayamos abordado los conceptos necesarios.
