2.1 Sentencias condicionales

1. Introducción

Los programas se construyen para procesar datos, manipulándolos de formas diferentes dependiendo de los valores que tengan. Los lenguajes de programación deben proveer estructuras que les permitan a los programadores controlar el flujo de ejecución de un programa dependiendo de los datos que procesan. Para ello, se incluyen las sentencias de control de flujo, que alteran el fujo de ejecución para tomar decisiones o repetir sentencias.

La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras básicas: secuencia, sentencias condicionales y bucles.

Una subrutina o subprograma, como idea general, se presenta como un subalgoritmo que forma parte del algoritmo principal, el cual permite resolver una tarea específica. Por ejemplo, los métodos en Java.

2. Secuencia

La ejecución secuencial es el más básico de los mecanismos de control de flujo y consiste en la ejecución de instrucciones en el orden en que se encuentran en el código fuente del programa.

Hasta ahora las instrucciones que hemos visto, son instrucciones que se ejecutan secuencialmente; es decir, podemos saber lo que hace el programa leyendo las líneas de izquierda a derecha y de arriba abajo.

3. Sentencias condicionales

La sentencia condicional consiste en ejecutar instrucciones diferentes dependiendo del resultado de evaluar una expresión lógica. Una **expresión lógica** es cualquier tipo de expresión que devuelva un resultado booleano (true o false). Las expresiones lógicas se construyen por medio de variables booleanas o bien a través de los operadores relacionales (==, >, <,...) y/o lógicos (&&,||, !).

3.1 Condicional simple

Se ejecutan una serie de instrucciones en el caso de que la expresión lógica sea verdadera:

```
if(expresiónLógica){
    instrucción1
    instrucción2
    ....
}
```

Ejemplo:

```
package tema2_1_SentenciasCondicionales;

import java.util.Scanner;

public class If {

    @SuppressWarnings("resource")
    public void show() {

        Scanner keyboard = new Scanner(System.in);
        int age;
        String name;

        System.out.print("Introduce tu nombre: ");
        name = keyboard.nextLine();
        System.out.print("Introduce tu edad: ");
        age = keyboard.nextInt();

        if (age >= 18) {
            System.out.printf("Bienvenido/a %s\n", name);
            System.out.println("Eres mayor de edad");
        }

    }

    public static void main(String[] args) {

        new If().show();

    }

}
```

3.2 Condicional doble

Es igual que la anterior, solo que se añade un apartado **else** que contiene instrucciones que se ejecutarán si la expresión evaluada por el if es falsa.

```
if(expresiónLógica){
    instrucciones   //se ejecutan si la expresión lógica es verdadera
    ....
}
else{
    instrucciones   //se ejecutan si la expresión lógica es falsa
    ...
}
```

Ejemplo:

```
package tema2_1_SentenciasCondicionales;

import java.util.Scanner;

public class IfElse {

    @SuppressWarnings("resource")
    public void show() {

        Scanner keyboard = new Scanner(System.in);
        int age;
        String name;

        System.out.print("Introduce tu nombre: ");
        name = keyboard.nextLine();
        System.out.print("Introduce tu edad: ");
        age = keyboard.nextInt();

        if (age >= 18) {
            System.out.printf("Bienvenido/a %s\n", name);
            System.out.println("Eres mayor de edad");
        } else {
            System.out.printf("Bienvenido/a %s\n", name);
            System.out.println("Eres menor de edad");
        }

    }

    public static void main(String[] args) {

        new IfElse().show();

    }

}
```

3.3 Condicional múltiple

3.3.1 Anidación

Dentro de una sentencia *if* se puede colocar otra sentencia *if*. A esto se le llama anidación y permite crear programas donde se valoren expresiones complejas. La nueva sentencia puede ir tanto en la parte *if* como en la parte *else*.

Las anidaciones se utilizan muchísimo al programar. Solo hay que tener en cuenta que siempre se debe cerrar primero el último *if* que se abrió. Es muy importante también tabular el código correctamente para que las anidaciones sean legibles.

Ejemplo:

```
if (x==1) {
    instrucciones
    ...
}
else {
    if(x==2) {
        instrucciones
    ...
    }
    else {
        if(x==3) {
            instrucciones
            ...
        }
    }
}
```

Una forma más legible de escribir ese mismo código dando lugar a la llamada estructura **if-else-if** sería:

```
if (x==1) {
    instrucciones
    ...
}
else if (x==2) {
    instrucciones
    ...
}
else if (x==3) {
    instrucciones
    ...
}
```

Se van comprobando las condiciones en orden y cuando una condición se cumpla, se ejecutan sus instrucciones y después se sale de la estructura *if-else-if* ya que esta estructura se utiliza cuando solamente se cumple una de las condiciones. Por eso, no estaría bien hacer lo siguiente:

```
if (x==1){   //Forma incorrecta de programar
    instrucciones
    ...
}
if (x==2){
    instrucciones
    ...
}
if (x==3){
    instrucciones
    ...
}
```

El motivo de que no sea adecuado es porque se pierde tiempo en comprobar todas las condiciones. Por ejemplo, si x vale 1, se ejecutan sus instrucciones correspondientes y luego se comprobaría si x vale 2, si x vale 3, etc. cuando no se va a cumplir ninguna más ya que si x vale 1 no puede valer ni 2 ni 3.

Ejemplo:

```
package tema2_1_SentenciasCondicionales;

import java.util.Scanner;

public class IfElseIf {

    @SuppressWarnings("resource")
    public void show() {

        Scanner keyboard = new Scanner(System.in);
        int age;

        System.out.print("Introduce tu edad: ");
        age = keyboard.nextInt();

        if (age >= 18) {
            System.out.println("Eres mayor de edad");
        } else if (age >= 16 && age < 18) {
            System.out.println("Eres menor de edad pero tienes ciertos
privilegios");
        } else if (age >= 14 && age < 16) {
            System.out.println("Eres menor de edad y además no tienes
privilegios");
        } else {
            System.out.println("Eres menor de edad");
        }

    }

    public static void main(String[] args) {

        new IfElseIf().show();

    }
```

```
}
```

3.3.2 Switch

```
switch(expresión) {
   case valor1 :
      instrucciones
      break; // opcional

   case valor2 :
      instrucciones
      break; // opcional
   ....
   ....
   default : // opcional
      instrucciones
      break; // opcional
}
```

El cuerpo de una sentencia switch se conoce como **bloque switch**.

Esta sentencia evalúa una expresión y cada **case** contiene un posible valor del resultado de dicha expresión; si efectivamente el resultado equivale a ese valor, se ejecutan las instrucciones de ese *case* y de los siguientes.

La instrucción **break** se utiliza para salir del *switch*. De tal modo que si queremos que para un determinado valor se ejecuten las instrucciones de un apartado *case* y solo las de ese apartado, entonces habrá que finalizar ese *case* con un *break*. Cuando se alcanza una sentencia *break*, el *switch* termina y el flujo de control salta a la siguiente línea que sigue a la sentencia *switch*.

Fall through condition (condición de caída): esta condición se produce en la sentencia *switch* cuando no se utiliza *break* en un *case* y causa la ejecución de los siguientes *case* hasta que no se produce un *break* o se sale de la sentencia *switch*. Aunque a veces es necesario, se percibe como propenso a errores.

El bloque **default** sirve para ejecutar instrucciones para los casos en los que la expresión no se ajuste a ningún *case*. Técnicamente, el *break* del *default* no es necesario porque el flujo se sale de la sentencia *switch*, pero se recomienda utilizarlo para que la modificación del código sea más fácil y menos propensa a errores.

Funciona con los tipos de datos primitivos, con los tipos enumerados, con las cadenas (a partir de Java 7) y con los wrappers.

Ejemplo:

```
package tema2_1_SentenciasCondicionales;

import java.util.Scanner;

public class Switch1 {

    @SuppressWarnings("resource")
    public void show() {

        Scanner keyboard = new Scanner(System.in);
```

```
        int weekday;

        System.out.print("Introduce un número del 1 al 7 correspondiente al día
de la semana: ");
        weekday = keyboard.nextInt();

        switch (weekday) {
        case 1:
            System.out.println("Lunes");
            break;
        case 2:
            System.out.println("Martes");
            break;
        case 3:
            System.out.println("Miércoles");
            break;
        case 4:
            System.out.println("Jueves");
            break;
        case 5:
            System.out.println("Viernes");
            break;
        case 6:
            System.out.println("Sábado");
            break;
        case 7:
            System.out.println("Domingo");
            break;
        default:
            System.out.println("Día incorrecto");
            break;
        }

    }

    public static void main(String[] args) {

        new Switch1().show();

    }

}
```

Ejemplo:

```
package tema2_1_SentenciasCondicionales;

import java.util.Scanner;

public class Switch2 {

    @SuppressWarnings("resource")
    public void show() {

        Scanner keyboard = new Scanner(System.in);
```

```
        int month, year, numDays = 0;

        System.out.print("Introduce un número del 1 al 12 correspondiente a un
mes: ");
        month = keyboard.nextInt();
        System.out.print("Introduce un año: ");
        year = keyboard.nextInt();

        switch (month) {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            numDays = 31;
            break;
        case 4:
        case 6:
        case 9:
        case 11:
            numDays = 30;
            break;
        case 2://Se calcula si es un año bisiesto
            if (((year % 4 == 0) && !(year % 100 == 0)) || (year % 400 == 0)) {
                numDays = 29;
            } else {
                numDays = 28;
            }
            break;
        default:
            System.out.println("Mes inválido");
            break;
        }
        if (month >= 1 && month <= 12) {
            System.out.printf("Número de días del mes %d del año %d: %d", month,
year, numDays);
        }

    }

    public static void main(String[] args) {

        new Switch2().show();

    }

}
```

La decisión de utilizar sentencias *if-else-if* o una sentencia *switch* se basa en la legibilidad y en la expresión que la sentencia está probando. Una sentencia *if-else-if* puede comprobar expresiones basadas en rangos de valores o condiciones, mientras que una sentencia *switch* comprueba expresiones basadas solo en un único valor.

El switch clásico es una sentencia, no una expresión. En la versión 14 de java aparece una nueva sintaxis para el switch que puede ser utilizada como una expresión. Para ello se utiliza el operador `->` que indica que cada rama produce un valor. El operador `->` denota la ausencia de fall through.

```
package tema2_1_SentenciasCondicionales;

import java.util.Random;

public class Switch3 {

    public void show() {

        Random random = new Random();
        int seasonCode;
        String seasonName;

        seasonCode = random.nextInt(4);
        System.out.println(seasonCode);
        /* Al ser el switch una expresión, se puede
         * asignar el valor devuelto a una variable:
         */
        seasonName = switch (seasonCode) {
            case 0 -> "Spring";
            case 1 -> "Summer";
            case 2 -> "Fall";
            case 3 -> "Winter";
            default -> "???";
        };

        System.out.println(seasonName);

    }

    public static void main(String[] args) {

        new Switch3().show();

    }

}
```

Se pueden proporcionar varias etiquetas para cada caso separadas por comas:

```
package tema2_1_SentenciasCondicionales;

import java.util.Random;

public class Switch4 {

    public void show() {

        Random random = new Random();
        int seasonCode, numLetters;
```

```
        String seasonName;

        seasonCode = random.nextInt(4);
        seasonName = switch (seasonCode) {
            case 0 -> "Spring";
            case 1 -> "Summer";
            case 2 -> "Fall";
            case 3 -> "Winter";
            default -> "???";
        };

        System.out.println(seasonName);

        numLetters = switch (seasonName) {
            //Varias etiquetas separadas por comas:
            case "Spring", "Summer", "Winter" -> 6;
            case "Fall" -> 4;
            default -> -1;
        };

        System.out.println(numLetters);

    }

    public static void main(String[] args) {

        new Switch4().show();

    }

}
```

También se pueden poner bloques (conjuntos de sentencias delimitadas por llaves) detrás del operador `->`. En este caso, el bloque debe salir a través de una sentencia `yield` que devuelva el valor del bloque. Al igual que return sale de un método con un valor, `yield` sale del bloque con un valor.

```
package tema2_1_SentenciasCondicionales;

import java.util.Random;

public class Switch5 {

    public void show() {

        Random random = new Random();
        int seasonCode;
        String seasonName;

        seasonCode = random.nextInt(4);
        seasonName = switch (seasonCode) {
            /* El bloque debe salir a través de una
             * sentencia yield que devuelva el valor
             * del bloque:
             */
```

```
            case 0 -> {
                System.out.println("spring time!");
                yield "Spring";
            }
            case 1 -> "Summer";
            case 2 -> "Fall";
            case 3 -> "Winter";
            default -> "???";
        };

        System.out.println(seasonName);

    }

    public static void main(String[] args) {

        new Switch5().show();

    }

}
```

El operador `->` denota la ausencia de fall through. Por simetría, se proporciona una versión de las expresiones switch con fall through usando `:` en lugar de `->`:

```
package tema2_1_SentenciasCondicionales;

import java.util.Random;

public class Switch6 {

    public void show() {

        Random random = new Random();
        int seasonCode;
        String seasonName;

        seasonCode = random.nextInt(4);
        System.out.println(seasonCode);
        seasonName = switch (seasonCode) {
            case 0:
                System.out.println("spring time!");
            case 1:
                yield "Summer";
            case 2:
                yield "Fall";
            case 3:
                yield "Winter";
            default:
                yield "???";
        };

        System.out.println(seasonName);

    }
```

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

        new Switch6().show();

    }

}
```

En el caso de que el código de la estación salga 0, se produce fall through dando el siguiente resultado por consola:

```
0
spring time!
Summer
```