Control structures

Selective structures

Selective or conditional structures let us choose among different paths in our code depending on a given condition (or conditions) that must be checked. In this document we'll see the selective structures provided by Java.

1. The "if" clause

You can use the `if` basic structure to run a piece of code if a given condition is true:

```
if (age >= 18){
    System.out.println("You are old enough");
}
```

Alternatively, you can use `if..else` structure to choose among a given path (determined by a condition) or its opposite:

```
if (age >= 18){
    System.out.println("You are old enough");
}
else{
    System.out.println("You are not adult yet");
}
```

Finally, if there is more than two paths to choose among, we can join `if .. else if .. else if .. else` structures to choose among several paths depending on the condition relative to each one. Only one path will be chosen.

```
if (number > 0){
    System.out.println("It is positive");
}
else if (number < -10){
    System.out.println("It is under -10");
}
else{
    System.out.println("It is between -10 and 0");
}
```

The condition in each `if` or `else if` clause can be either a simple condition (like the ones shown in previous examples) or a complex condition, joined by logical operators `&&` and/or `||`:

```
if (age >= 18 && age <= 30){
    System.out.println("You are between 18 and 30");
}
```

2. The "switch" clause

Besides, there's a `switch` clause that lets us evaluate the value of a simple variable or expression. Each of the possible values of this expression can be represented with a `case` clause. Finally, we can use a `default` case to represent any other possible value that has not been covered by previous *case* clauses.

The data managed in the `switch` clause must be a primitive type; strings are NOT allowed in early versions of Java (Java 6 and earlier). We need to add a `break` instruction at the end of each *case* to exit the *switch* clause; otherwise, the program keeps running the instruction of next *case* clause. In other languages, such as C#, the *break* instruction is compulsory in the *case* clauses that are not empty, but this does not happen in Java, so we must take care of this situation.

```
switch(number){
    case 0:
        System.out.println("It is 0");
        break;
    case 1:
        System.out.println("It is 1");
        //fallthrough
    case 2:
        System.out.println("It is 2");
        break;
    default:
        System.out.println("Unknown number");
}
```

In previous example, if number is 1, it would output the messages "It is 1" and "It is 2", since there is no `break` clause at case 1.

Java 12 introduced switch expressions, allowing switches to "return" a value and introducing the new `case L ->` labels that eliminate the need for break statements to prevent fall through. You can use a `yield`` statement to specify the value of a switch expression.

> **Exercise 1:**
>
> Create a program called **MarkCheck** that asks the user to enter 3 marks. The program must print one of these messages, depending on the mark values:
>
> - *All marks are greater or equal than 4*
> - *Some marks are not greater or equal than 4*
> - *No mark is greater or equal than 4*

> **Exercise 2:**
>
> Create a program called **GramOunceConverter2** that will be an improved version of a previous exercise. In this case, the user will type a weight (float), and a unit (`g` for grams, `o` for ounces). Then, depending on the unit chosen, the program will convert the weight to the opposite unit. For instance, if the user types a weight of 33 and chooses `o` as unit, then the program must convert 33 ounces to grams. You must solve this program using a `switch` structure. If the unit is other than `g` or `o` , then the program must output an error message: "Unexpected unit", with no final result.