Control structures

Boolean type and operators

In this section we are going to talk about the basics of structured programming using logical information. So, we are going to introduce the boolean data types that lets us check some conditions and decide which piece of code we should run next. This way, we can choose among different results depending on a given input condition, or even repeat the execution of a given piece of code many times without having to re-type it again.

1. Boolean data type

Boolean data type is another basic data type (just like integers, characters or floating point numbers) that lets us represent two opposite values: *true* and *false*. In Java, this data type is represented by the word `boolean`. We can declare variables of this type, and also assign an initial value to them (from the range `true` and `false`):

```
boolean b = true;
```

We can even read boolean values from user input, using `nextBoolean` method from the *Scanner* variable. In this case, user must type *true* or *false* in the keyboard, which is not very intuitive.

```
Scanner sc = new Scanner(System.in);
boolean b = sc.nextBoolean();
```

2. Some additional operators

Related with this boolean data type, there are some Java operators that we can use. In this section we are going to talk about relational and logical operators.

2.1. Relational operators

Relational operators let us compare two different values, and check if one of them is greater, or lower, or equal than the other. This is the complete list of relational operators:

Operator	Meaning
`>`	Greater than
`>=`	Greater or equal than
`<`	Lower than
`<=`	Lower or equal than
`==`	Equal to
`!=`	Not equal to

Note that, if we want to check if two values are equal, we use `==` comparator instead of just `=`, which is used for assignment purposes. We can join these operators with boolean values to determine if some comparisons are true or false:

```
int n = 10;
boolean check1 = n > 5;     // true
boolean check2 = n != 10;   // false
```

Regarding strings, we can't use these comparators, because they don't work as expected (our program compiles, but results may be unpredictable). If we want to check if two strings are the same, we use `equals` instruction instead of `==` comparator:

```
String s1 = "Hello";

boolean check1 = s1 == "Hello";      // Does not work as expected
boolean check2 = s1.equals("Hello"); // OK
```

We'll learn more about how to deal with string values in later sections.

2.2. Logical operators

Logical operators join two or more simple comparisons to build a complex one. This way we can check if every comparison in the list is true, or at least one of them. The final result of this complex expression is also a boolean value. This is the list of logical operators:

Operator	Meaning
`&&`	AND operator
`||`	OR operator
`!`	NOT operator

Regarding **AND operator** `&&`, it joins two comparisons, so that the final result will be true if both comparisons are true. Otherwise it will be false:

```
int n = 10, m = 5;

boolean c1 = n > 5 && m < 10;   // true && true = true
boolean c2 = n > 5 && m > 10;   // true && false = false
```

OR operator `||` also joins two comparisons, but in this case the final result will be true if any of the comparisons joined (or both) are true:

```
int n = 10, m = 5;

boolean c1 = n > 5 || m > 10;   // true || false = true
boolean c2 = n < 5 || m > 10;   // false || false = false
```

Finally, the **NOT operator** `!` is a unary operator, this is, it affects only one expression (not two), and changes the value of this expression (this is, if the expression was true, the final result is false, and vice versa).

```
int n = 10;

bool c1 = n > 5;    // true
bool c2 = !c1;      // false
```

The **precedence** of these operators is important:

1. First of all, we evaluate every expression between **parentheses**
2. Then, we evaluate **NOT** operators
3. Next, we check **AND** operators
4. Finally, we look for **OR** operators

Also, you must take into account that both AND and OR operators work in *short circuit* mode. This means that:

- Regarding AND operator, if the first expression is *false*, second expression is not checked (final result will be *false* anyway)
- Regarding OR operator, if the first expression is *true*, second expression is not checked (final result will be *true* anyway).

> **Exercise 1:**
>
> Try to guess the final result of these expressions `e1`, `e2` and `e3`. You can then write a short program to check your answers:

```
int a = 3, b = 5, c = 8;

boolean e1 = a < 2 && b >= 5 || c == 8;
boolean e2 = a < 2 && (b >= 5 || c == 8);
boolean e3 = !(a < 2) && (b >= 5 || c == 8);
```