Basic elements of a Java program

Variables and operators

1. Using variables

Variables are essential elements in every program, since they let us store values in them, so that we can operate with them, or modify them along the program execution. Every variable has a *type* which lets us know what kind of information we can store in it. For instanec, there are *integer* variables, that lets us store integer numbers, or *string* variables to manage texts. We'll learn about data types in other sections.

Whenever we want to use a variable we must **declare** it. This step consists in:

- Specifying the data type of the variable (integer, text...)
- Specifying the name of the variable, which is also known as the variable *identifier*.

For instance, this way we declare a variable to store integer values, using the word `int` to specify the data type:

```
int myVariable;
```

Additionally, we can also **assign** a value to the variable. This step can be done either when we declare it, or later in the code:

```
int myVariable = 3;
int myOtherVariable;
...
myOtherVariable = 5;
```

Also, variables can change their values in later sentences.

```
int myVariable = 3;
...
myVariable = 5;
```

We can also declare multiple variables of the same type in the same line, separated by commas, and we can decide for each one if we want to assign an initial value or not:

```
int number1 = 0, number2, result = 1;
```

Regarding the **identifier**, it can contain letters (in either lower or upper case), digits, and the underscore symbol `_`, but it can't start with a digit. These are examples of valid identifiers (they usually start with a lower case letter in Java, but this is not compulsory)

```
int aNumber;
int another_number;
int number1;
int _one_more_number;
```

Whereas these are examples of non-valid identifiers:

```
int 1number;
int another number;
```

We can use the `System.out.println` instruction to show the value of a variable in the screen:

```
public class MyClass{
    public static void main(String[] args){
        int myVariable = 3;
        System.out.println(myVariable);
    }
}
```

2. Some basic Java operators

Operators let us evaluate expressions and produce a given result. For instance, if we use the addition operator `+` we can sum a couple of numbers and get the final result. This final result can either be assigned to a variable or shown in the screen.

```
int result = 3 + 4;
System.out.println(32 + 52);
```

2.1. Arithmetic operators

Arithmetic operators let us do some basic, mathematical operations with numbers. This is the list of basic arithmetic operators in Java:

Operator	Meaning
`+`	Addition
`-`	Substraction
`*`	Multiplication
`/`	Division
`%`	Division module

Regarding the *division* operator, we must take into account that it produces a result of the same type that the numbers involved. In other words, if we divide two integer numbers, such as 5 / 2, then the result will be integer (2), not real. The *module* operator gets the module of an integer division. In previous example, 5 % 2 gets the module of dividing 5 by 2, which is 1.

Operator precedence

The order in which operators are evaluated in an arithmetic expression is important. For instance, if we set an expression like this one:

```java
int result = 4 + 2 / 2;
```

Then `result` variable gets a final value of 5, because the division `2 / 2` is evaluated BEFORE the addition. This is the precedence order for arithmetic operators:

1. Multiplications, divisions and modules
2. Additions and substractions

If there are more than one operator of the same range in an operation, then they are evaluated from left to right. For instance, in this case, we first apply the multiplication, and then the division, and the final result is 2:

```java
int result = 4 * 3 / 6;
```

However, we can alter the order in which operations are evaluated in an expression, putting between **parentheses** the operations that we want to evaluate in first places. This expression has a result of 0, because we are forcing to evaluate the division 3 / 6 = 0 in first place.

```java
int result = 4 * (3 / 6);
```

This other expression has a reuslt of 3, because we are forcing to evaluate the addition before the division:

```
int result = (4 + 2) / 2;
```

> **Exercise 1:**
>
> Try to determine the final value stored in `result` variable in each one of these expressions. You can write a small Java program later to check your answers:
>
> - `int result = 4 + 8 * 2 / 4`
> - `int result = (4 + 8) * 2 / 4`
> - `int result = (4 + 8) * 3 % 5`

2.2. Assignment operators

We have already used the `=` operator to assign a value to a variable. But there are some other assignment operators that we can use if we want to include some arithmetic operation in the process. For instance, `+=` operator, which is also known as *auto-addition* operator, automatically adds the specified value to variable's current value. In this example, the final value of `result` variable is 5:

```
int result = 3;
result += 2;
```

This is the list of the assignment operators available:

Operator	Meaning
`=`	Simple assignment
`+=`	Auto-addition
`-=`	Auto-substraction
`*=`	Auto-multiplication
`/=`	Auto-division
`%=`	Auto-module

Auto increment and auto decrement operators

Java also provides two additional operators, which are a mix of arithmetic and assignment operators. These operators are the auto-increment `++` and auto-decrement `--` operators. They apply to a single variable, and automatically increase or decrease its value in 1 unit, respectively.

For instance, final value of `result` variable in the following code is 4:

```
int result = 3;
result++;
```

These operators can be placed either before or after the affected variable. There is an important difference that you must take into account regarding the operator placement:

- If we place the operator BEFORE the variable in a complex expression, then we first increase/decrease the affected variable, and then we complete the expression. For instance, in this code, the final value of `b` variable is 6, because we first increase `a` value (to 4), and then we auto-sum this value to `b`.

```
int a = 3, b = 2;
b += ++a;
```

- If we place the operator AFTER the variable in a complex expression, we first evaluate the whole expression, assign the value, and then, we increase/decrease the affected variable. The same code of previous example gets a result of 5 for `b` variable if we type it like this, although `a` variable will end with the same final result, which is 4.

```
int a = 3, b = 2;
b += a++;
```

Note that these rules don't apply if we use the auto-increment or decrement operator in a single line. These two lines have the same effect over variable `a`:

```
a++;
++a;
```

> **Exercise 2:**
>
> Determine the final value of `result` variable after running all these instructions:

```
int result = 4;
result += 3;
result *= 2;
result--;
result %= 4;
```

3. Constants declaration

Constants are values that never change. We declare constants in Java by declaring the data as `final` and `static` (we will learn later the meaning of these terms). Typically these constants are placed at the beginning of the class.

```
class MyClass
{
    static final int MAX_USERS = 10;
    ...
}
```

4. Comments

Comments help us clarify some parts of our code, by adding some "human" text. This text is ignored by the compiler, but helps the developer understand or find some parts of the code.

In Java, there are two types of comments:

- Single line comments, which are preceded by a double slash `//` :

```
// We declare an integer variable
int variable = 3;
```

- Multiple line comments, which are started by `/*` and finished by `*/`. Everything in between makes up the comment, and it's ignored by the compiler:

```
/* This is a comment of
   multiple lines before
   declaring a variable */
int variable = 3;
```