by Nacho Iborra

Development Environments
Block 2
Unit 1: Basic structure of a Java program. I/O. Primitive types. Operators

2.1.1. Introduction to Java

Java is an object-oriented programming language created by Sun Microsystems in the middle of the 90's. Its main purpose was to implement programs that were able to run in multiple platforms, such as different operating systems, and even some electrical appliances (refrigerators, washing machines...). Currently, the language is being maintained by Oracle.

As we have seen in previous units. Java compiles its source files into an intermediate language, that can be run on the **Java Virtual Machine**. This way, Java programs are multi platform, we just need to install this virtual machine on our chosen operating system.

2.1.1.1. Required software

If you want to develop Java applications, you need to install Java JDK (Java Development Kit). You should have it installed already, from previous units. Besides, you will need an IDE to create your Java projects. You can choose among:

- A simple IDE such as **Geany**, which can be automatically configured to work with JDK and use the corresponding compiler.
- A specific IDE such as **IntelliJ IDEA** that lets us create more complex Java projects with multiple source files, libraries and so on.

For the first units of this block (until we learn something about classes), we will use Geany rather than IntelliJ, because it will be easier to create Java programs with a single source file. Later, we will move to IntelliJ IDEA to deal with projects with multiple classes and source files.

2.1.2. First steps with Java

2.1.2.1. Our first java program

Let's see how to start with Java, by creating a simple Java program that prints "Hello" in the screen.

```
public class MyClass
{
    public static void main(String[] args)
    {
        System.out.println("Hello");
    }
}
```

The structure of this program is very similar to the same program written in C#: we always need to define a class, even if we only need a `main` function or method. This `main` function is written in lower case in Java.

Besides, every public class must have the same name as the source file that contains it, so we need to store the source code of the example above in a file called `MyClass.java` (Java source files have `.java` extension).

2.1.2.2. Compiling and running the program

If we compile this program in Geany, it will use the `javac` compiler internally to produce a file called `MyClass.class` in the same folder than the source file. This is the compiled file that can be run on the Java Virtual Machine. Then, if we run the program, Geany will use the `java` tool to run it and output "Hello" message in the screen.

2.1.3. Variables and basic data types

To deal with basic data types (integer or floating point numbers, characters, boolean values and strings) we have the following data types in Java:

- `byte`, `short`, `int`, `long` for integer numbers
- `float` and `double` for real numbers
- `char` for characters
- `boolean` for boolean types, whose values may be `true` or `false`
- `String` for strings (this type starts with uppercase in Java)

There are no modifiers such as *signed/unsigned*, as in other languages (C, C++). The ranges of these data types are more or less the same than in other languages.

2.1.3.1. Variable declaration

We declare variables in Java like we do in languages such as C#: we specify the data type for the variable, and then the variable name. We can also declare multiple variables of the same type in the same line, separated by commas, and we can even assign an initial value to them:

```
int number = 0, result = 1;
char symbol;
String name = "Pepe";
```

Variables can be declared anywhere in the code, but it is a good practice to declare them at the beginning of the function that uses them (`main` function, in these first examples).

2.1.3.2. Some useful conversions

There are some type conversions that are very usual in Java applications:

- If we want to convert between numeric types, we just need to typecast them. For instance, we can convert from `float` to `int` this way:

```
float pi = 3.1416;
int piInteger = (int)pi;
```

- If we want to convert from String to integer, or float, or double there are some useful methods inside `Integer`, `Float` or `Double` classes:

```
int number = Integer.parseInt("42");
float floatNumber = Float.parseFloat("3.1416");
double doubleNumber = Double.parseDouble("3.141592654");
```

- We may also need to convert from a numeric value to a string. This conversion is quite easy by joining an empty string `""` with the corresponding numeric value:

```
int number = 23;
String numberString = "" + number;
```

2.1.4. Operations and operators

Java has more or less the same set of operators that we can find in many other languages:

- **Arithmetic operators**: `+`, `-`, `*`, `/`, `%` (module), `++` (increment), `--` (decrement)
- **Assignments**: `=`, `+=`, `-=`, `*=`, `/=`, `%=`
- **Comparisons**: `>`, `>=`, `<`, `<=`, `==`, `!=`
- **Logical**: `&&` (AND), `||` (OR), `!` (NOT)

> **NOTE:** the comparison `==` DOES NOT WORK with strings. You must use the `equals` method (`string1.equals(string2)`)

2.1.5. Basic input / output

Let's see now how to print information in the screen and how to ask the user to enter data.

2.1.5.1. Basic input: Scanner

In order to get the user input, the easiest way may be through the `Scanner` object. We need to import `java.util.Scanner` in order to use it, and then we create a `Scanner` element and call some of its methods to read data from the user. Some of them are `nextLine` (to read a whole line of text until the user presses Enter) and `nextInt` (to read an integer explicitly):

```
import java.util.Scanner;
...
public class ClassName
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        int number = sc.nextInt();
        String text = sc.nextLine();
        sc.close();
    }
```

There are some other methods, such as `nextFloat`, `nextBoolean`... but they are very similar to `nextInt`, and they help us read specific data types from the input, instead of reading texts and then converting them into the corresponding type (as `Console.ReadLine` does in C#). You can introduce this data separated by whitespaces or new lines (Intro).

```
int number1, number2;
// These two numbers can be entered either separated by
// whitespaces or by new lines
number1 = sc.nextInt();
number2 = sc.nextInt();
```

Be careful when combining data types

Let's suppose that you have to read this information from the input:

```
23 43
Hello world
```

You may think that you need to use `nextInt` method twice, and then `nextLine` method to read the last string, but this approach is NOT correct: when you use `nextInt` to read the integer values, you don't read the end of line that exists beyond number 43, so, when you use `nextLine` method once, you just read this new line, but not the second line. The correct sequence would be this one:

```
int number1 = sc.nextInt();
int number2 = sc.nextInt();
sc.nextLine(); // Read and ignore end of first line
String text = sc.nextLine();
```

2.1.5.2. Basic output: System.out.print(ln)

On the other side, you can use the `System.out.print` or `System.out.println` instruction (depending on whether you want a new line at the end or not) to print messages to the screen. You can join multiple values by using the link operator (+):

```
int result = 12;
System.out.println("The result is " + result);
```

2.1.5.3. Formatted output

Apart from traditional `System.out.println` instruction to print data, we can use some other options if we want this data to have a given output format. To do this, we can use `System.out.printf` instruction instead of the previous one. This instruction behaves in a similar way than the original `printf` function from C language. It has a variable number of parameters, and the first of all is the string to be printed out. Then, this string can have some special characters inside it, which determine the data types that must replace these characters. For instance, if we use this instruction:

```
System.out.printf("The number is %d", number);
```

then the symbol `%d` will be replaced by the variable `number`, and this variable must be an integer (this is what `%d` means).

There are some other symbols to represent different data types. Here are some of them:

- `%d` for integer types (`long` , `int`)
- `%f` for real types (`float` and `double`)
- `%s` for strings
- `%n` to represent a new line (similar to `\n` , but platform independent). In this case, we don't need to add a parameter at the end of `printf` .

We can place as many symbols as we want inside the output string, and then we will need to add the corresponding number of parameters at the end of the `printf` instruction. For instance:

```
System.out.printf("The average of %d and %d is %f",
    number1, number2, average);
```

Besides the primary simbols `%d` and `%f` , we can add some other information between the '%' and the letter, that specify some format information.

Specifying integer digits

For instance, if we want to output an integer with a given number of digits, we can do it this way:

```
System.out.printf("The number is %05d", number);
```

where `05` means that the integer is going to have, at least, 5 digits, and if there are not enough digits in the number, then it will be filled with zeros. The output of this instruction if number is `33` would be `The number is 00033` . If we don't put the `0` , then the number will be filled with whitespaces. So this instruction:

```
System.out.printf("The number is %10d", number);
```

if number is `33` , it would produce the following output: `The number is         33` .

Specifying decimal digits

In the same way that we format integer numbers, we can format real numbers. We can use the same pattern seen before to specify the total number of integer digits:

```
System.out.printf("The number is %3f", number);
```

But, besides, we can specify the total number of decimal digits by adding a point and the total number desired, this way:

```
System.out.printf("The number is %3.3f", number);
```

Then, if number is `3.14159`, the output would be `The number is   3.142`.

2.1.6. Constants declaration

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. We can set them public, protected or private, depending on which classes are going to access these values (we will also learn later when to use each one of these modifiers).

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

Proposed exercises:

2.1.6.1. Create a program called *FormattedDate* with a class with the same name inside. The program will ask the user to enter the day, month and year of birth (all values are integers). Then, it will print his birth date with the format *d/m/y*. For instance, if the user types day = 7, month = 11, year = 1990, the program will output *7/11/1990*.

2.1.6.2. Create a program called *GramOunceConverter* that converts from grams to ounces. The program will ask the user to enter a weight in grams (an integer number), and then it will show the corresponding weight in ounces (a real number), taking into account that 1 ounce = 28.3495 grams.

2.1.6.3. Create a program called *NumbersStrings*. This program must ask the user to enter 4 numbers, that will be stored in 4 `String` variables. Then, the program will join the first pair of numbers into a single integer value, and the second pair of numbers into another integer value, and then add these values. For instance, if the user types the numbers 23, 11, 45 and 112, then the program will create a first integer value of 2311 and a second integer value of 45112. Then, it will add these two values and get a final result of 47423.

2.1.6.4. Create a program called *CircleArea* that defines a float constant called `PI` with the value `3.14159`. Then, the program will ask the user to enter the radius of a circle, and it will output the area of the circle (`PI` * radius * radius). This area will be printed with two decimal digits.