# Basic elements of a Java program

## Basic input and output

In this document we are going to learn how to interact with final user. First, we will see how to print values in the screen using different instructions, and then we will see how to gather information from the keyboard and convert it to the appropriate data type.

### 1. Program output

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);
System.out.print("Have a nice day!");
```

#### 1.1. 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 similarly to the original `printf` function from C language. It has a variable number of parameters, and the first one 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
- `%c` for characters

- `%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 symbols `%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 fraction 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 characters (including the point):

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

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

```
System.out.printf("The number is %7.3f", number);
```

Then, if number is `3.14159`, the output would be `The number is  3.142`. 7 Characters, including the point, and 3 fraction digits, and two padding whitespace. Please note the rounding.

Or we can specify only the fraction digits:

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

Then, if number is `3.14159`, the output would be `The number is 3.142`. With 3 fraction digits.

Some examples of floating point formatting using printf:

```
System.out.printf("| %-8s | %f |%n",      "%f",      12345.12345);
//Decimals, rounding
System.out.printf("| %-8s | %.4f |%n",    "%.4f",    12345.12345);
//Total chars, padding right
System.out.printf("| %-8s | %15f |%n",    "%15f",    12345.12345);
//Total chars, padding left
System.out.printf("| %-8s | %-15f |%n",   "%-15f",   12345.12345);
// Total chars and Max decimals
System.out.printf("| %-8s | %15.3f |%n", "%15.3f", 12345.12345);
//Total chars, padding with 0s
System.out.printf("| %-8s | %015.3f |%n", "%015.3f", 12345.12345);
```

The output obtained:

```
| %f       | 12345,123450 |
| %.4f     | 12345,1235 |
| %15f     |    12345,123450 |
| %-15f    | 12345,123450    |
| %15.3f   |       12345,123 |
| %015.3f  | 00000012345,123 |
```

## 2. Getting user input

In order to get the user input, the easiest way may be through the `Scanner` class (Scanner doc.). We need to import `java.util.Scanner` in order to use it, and then we create a `Scanner` object and call some of its methods to read data from the user.

A Scanner breaks its input into tokens using a delimiter pattern, which by default matches **whitespace**. The resulting tokens may then be converted into values of different types using the various next methods. When reading from the console the input is not processed until enter is pressed.

Some of the methods used to read data are `next` (to read the next token as a string) and `nextInt` (to read the next integer):

```
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.next();
        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;
number1 = sc.nextInt();
number2 = sc.nextInt();
```

**Be careful when using nextLine**

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();
String text = sc.nextLine(); // Won't read Hello world
                             // Reads the end of line left by nextInt()
```

The right way of reading the data:

```
int number1 = sc.nextInt();
int number2 = sc.nextInt();
String text = sc.next();
```

`next` reads a whole token (between two delimiters, spaces or enters by default), while `nextLine`` returns the remaining text until the end of the current line.

### 2.1. Using System.console().readLine()

There's an additional way of reading data from user input. It consists in using `System.console().readLine()` method, which is similar to Scanner's `newLine` method: it reads the whole line until user presses *Intro*, so we ALWAYS read a string with this instruction, and we need to convert it to its corresponding data type later:

```
System.out.println("Write a number:");
String text = System.console().readLine();
int number = Integer.parseInt(text);
```

The main drawback of this instruction is that it doesn't work well in the terminal of some IDEs, since the terminal of this IDE is not a *system* terminal, so you can't rely on it in certain situations.

> **Exercise 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*.

> **Exercise 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.

> **Exercise 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.

> **Exercise 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.