# Lecture 02

# Programming in Java and Java Development Kit (JDK)

***Song of the day***: ***If You Know What's Right*** *by Her's (2018).*

## Sections



## Part 1: *Our First Java Program*

Before we get into the Java environment, let's just actually *write* a quick program in Java for better context. Every single Java program must exist within the confines of a **class**. By convention, we write class names using capitalised camel-case (i.e. `LikeThis` and not `Like_This`):

```
package helloWorld;  // For now, consider this the name of our current folder

class HelloWorld {
    // Our actual functionality will go here...
}
```

**Code Block 1**: The "skeleton" of a Java class. If you attempt to write code outside a Java class, the Java compiler will not let you run it (in fact, will not even bother trying to compile it).

Alright, so how to we tell Java to run things? If you want a Java class to execute something when it is compiled and ran, you need what is called a driver, or a **main**, function. Java's version of the `main()` is infamously verbose and difficult to remember if you don't know what each individual part of it does, but we'll learn what each of these things mean in due time:

```
package helloWorld;

class HelloWorld {
    public static void main(String[] args) {
```

```
        // Our driver code will go here...
    }
}
```

**Code Block 2**: The Java `main()` function defined for the `HelloWorld` class.

The `public`, `static`, and `void` keywords will become massively important later on, but for now you don't have to worry about what they, nor what the `String[] args` mean. Let's just write a quick *"Hello, World!"*—the canonical first-line-of-code—inside the `main()` and tell Java to run it for now:

```
package helloWorld;

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

**Code Block 3**: A **Java *"Hello, World!"*** defined for the `HelloWorld` class.

If you've worked with other languages before, such as Python, `System.out.println()` is basically the equivalent of a `print()` statement—it tells Java to display whatever is passed into the set of parentheses onto our screen. In this case, that whatever is the **string** of characters `"Hello, World!"` (strings of characters, or "strings" for short, are denoted in Java by quotation `"` marks).

## Part 2: *Compiling a Java file*

Cool, so how to we run this? Since Java is a *compiled* language, we first need to break down our source code into something called "bytecode". This, essentially, is our code deconstructed in such a way that the *Java Virtual Machine* (JVM) can interpret and tell your computer to run. The way we do this in our command line / Terminal is as follows:

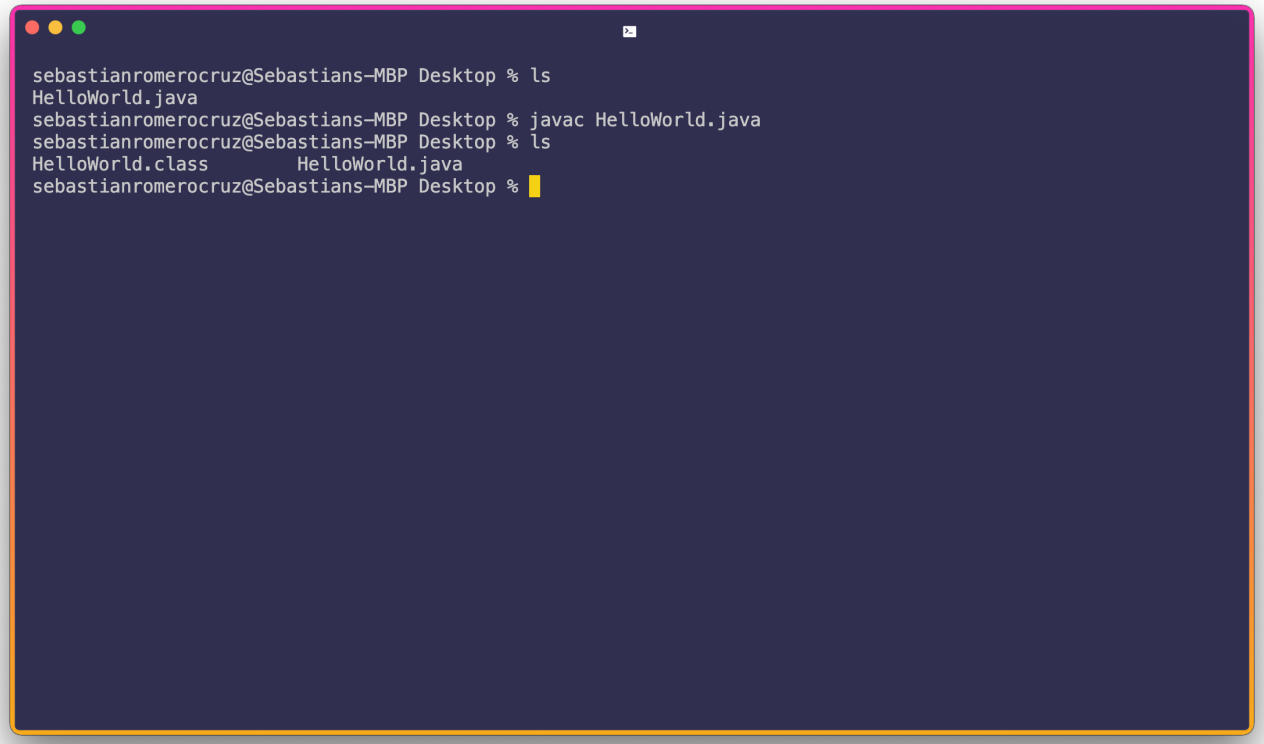



**Figure 1**: A Terminal (the Unix equivalent of the Windows Command line) compiling our `HelloWorld.java` file.

In order, my terminal is doing the following:

1. Listing ( `ls` ) the files inside my Desktop folder.
2. **Compiling** our `HelloWorld.java` file by using the `javac` command.
3. Listing the files inside my Desktop folder once more. Notice that, post-compiling, there is a new file called `HelloWorld.class` . This is the file containing the aforementioned bytecode.

If we use the Terminal command `cat` , we can get a quick glimpse of the contents of this file:

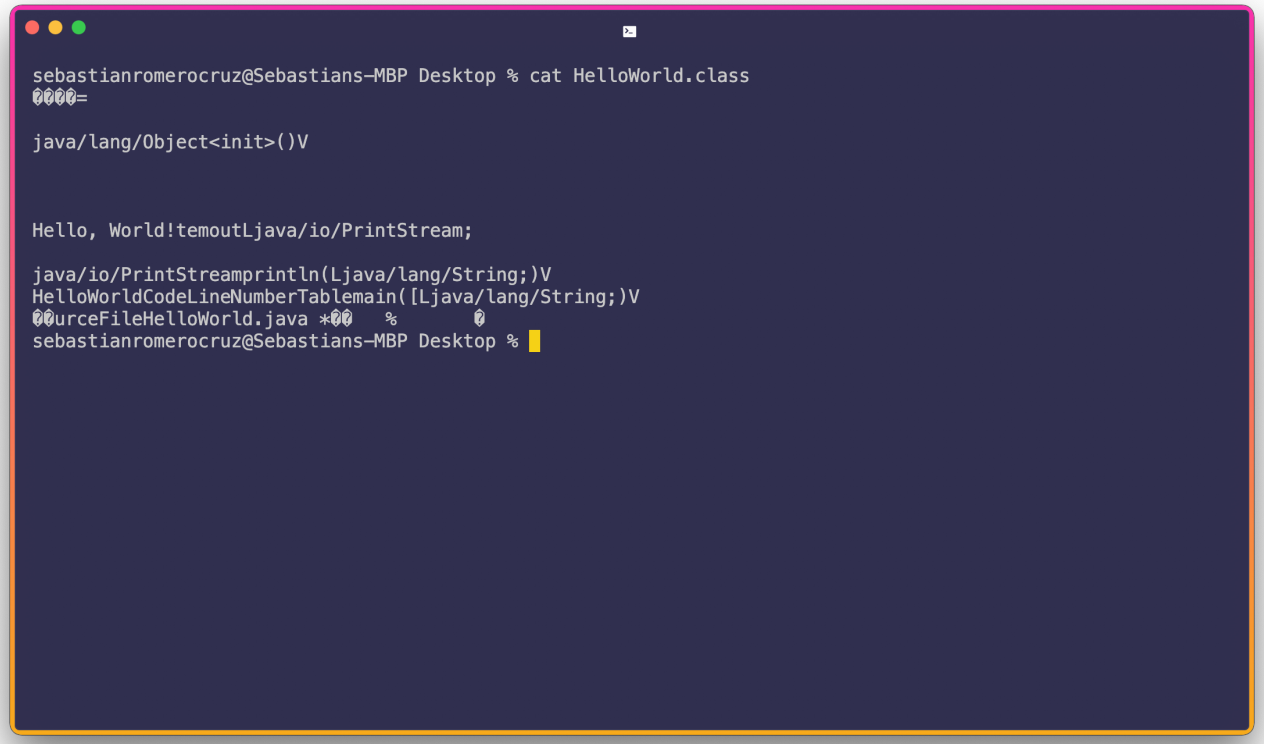



**Figure 2**: The Terminal showing us the contents of `HelloWorld.class`. It's contents, intended for your computer to interpret, are thus very difficult for *us* to understand.

## Part 3: *Running a Java file*

Just compiling a Java file is not enough to actually run it—that's actually what our `.class` file is for. In order to run it, we use the `java` command:

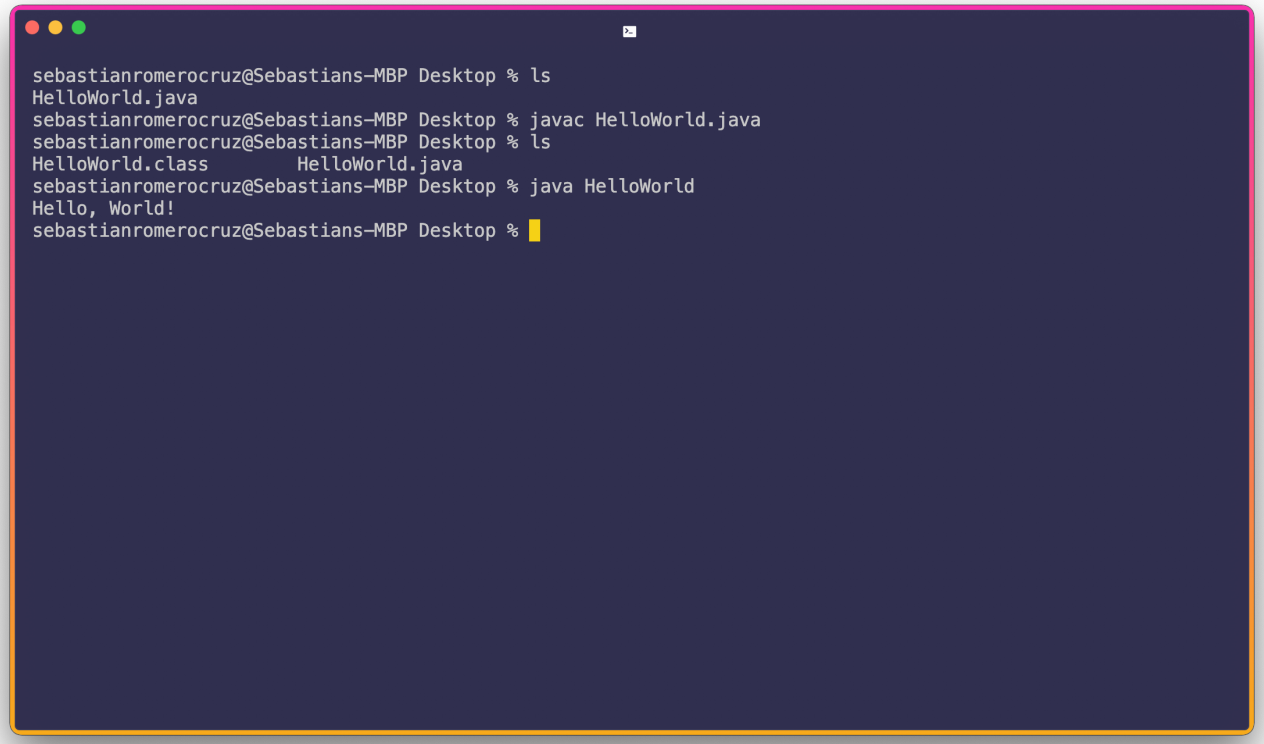



**Figure 3**: Running our `HelloWorld` program. Note that, when using the `java` command, we don't include the `.class` extension.

There it is! Our Java file's output: a nice "Hello, World!". Congrats; you're a programmer now.

## Part 4: *Demystifying Java*

Java is sometimes (and infamously) referred to as a verbose language. This is a diplomatic way of saying that, until a certain point, nobody has any idea of what half of what they are writing means (especially if they have never programmed before). Let's take our `HelloWorld` program from earlier and use the following table to make sense of it:

| Character | Name | Description |
|---|---|---|
| `{}` | Opening and closing braces | Denote a block of enclosed statements (a.k.a. code) |
| `()` | Opening and closing parentheses | Denotes the execution of a method (e.g. the `main()` method) |
| `[]` | Opening and closing brackets | Denotes an array. More on these in week 5 |
| `//` | Double slashes | Precede a comment line |
| `"""` | Opening and closing quotation marks | Enclose a string (i.e. a sequence of characters) |

| Character | Name | Description |
|---|---|---|
| `;` | Semicolon | Mark the end of a statement (i.e. a line of code) |

**Figure 4**: Special characters in Java.

Line 1, for instance:

```
package helloWorld;
```

Could be read in English as:

> This line of code consists of the `package` **keyword** and the package **name** `helloWorld`. We know that this line ends before of the semicolon `;` (also known as the **statement terminator**).

The proceeding lines:

```
class HelloWorld {
    // Our actual functionality will go here...
}
```

Could be read as:

> The `class` keyword is being used to **define** a class with the **name** `HelloWorld`, whose functionality will be contained within the proceeding opening and closing brackets `{}`.

Our "print" statement:

```
System.out.println("Hello, World!");
```

Could be read in English as:

> The `out.println()` **method**, which belongs to the `System` class, is being executed.

And so on and so forth. Not knowing how to "read" lines of code is not super important in the beginning (I much prefer that you actually *write* some code), but knowing the correct terminology certainly helps when listening to me talk, so please don't hesitate in asking if you don't recognise any words!

## Part 5: *Our Second (and third, and fourth, and...) Java Program*

Let's practice what we have learned by writing a couple more Java classes. Let's write one that introduced ourselves. I will call mine `Name`, but you can call it whatever you like:

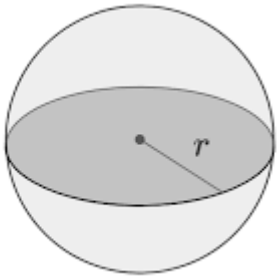
```
package helloWorld;

class Name {
    public static void main(String[] args) {
        System.out.println("Hello, my name is Sebastián.");
    }
}
```

**Code Block 4**: A Java **program** introducing me. Welcome to the future.

Compiling the `Name.java` file and running the `Name.class` bytecode file will result in the following output:

```
Hello, my name is Sebastián.
```

Nice. Let's try something involving things other than text. Let's write a program that prints the volume of the Earth, whose radius is approximately 6,371,000 metres. Recall, too, the formula for a sphere's volume:




$$V = \frac{2}{3}\tau r^3$$

**Figure 5**: The formula for the volume of a sphere. Here, the character τ (tau) is equivalent to 2π. I'm well aware that the Earth is actually an oblate spheroid, by the way. But I'm not doing that crap.

I will call this class `DisplayEarthVolume`:

```
package helloWorld;

public class DisplayEarthVolume {
    public static void main(String[] args) {
        System.out.println((2.0 / 3.0) * 2 * 3.14156 * 6371000);
    }
}
```

**Code Block 5**: A Java **program** to display the approximate volume of the Earth.

Compiling and running will result in the following output:

```
2.6686505013333336E7
```

Note here the following:

- I preceded the `class` keyword with the `public` keyword. We'll learn about what this means later, but for the most part, Java classes tend to be defined this way.
- Numbers in Java don't use `,` to, and can be defined with and without decimals (i.e. `2` vs `2.0`). We'll talk about this more next week.
- The output, being quite large, was displayed in **scientific notation**. `E` here is equivalent to saying "times ten to the power of...".