The Java Main Method

Introduction

In this article from my free Java 8 Course, I will be discussing the `public static void main(String[] args)` method. Up until this point in the series, we have run our code only through the JUnit framework. This is a sound, methodological practice, however, this is different from how our program would be run in production. Now let's explore how our code would run outside of the development environment.

The Main Method

Initially, the code you write in a computer program is just static text lying around passively in a file. To execute the code, the Java Runtime Environment (JRE) is responsible for loading the compiled program and starting its run. To execute the code the JRE needs an entry point. When running a Java class from the command line, the entry point the JRE looks for is the following method:

```
public static void main(String[] args) {
     /*
     * JRE starts by executing any code in here
     */
}
```

Example 1

Let's look at each part of the method in detail:

- ***public*** - allows the method to be called from outside the class.
- ***static*** - allows the method to be called without having an instance of the class created.
- ***void*** - it returns no value.
- ***main()*** - To execute your program, Java will specifically look for a method of the name "main".
- ***String[] args*** - You can call your program with a number of arguments. Your program can access those arguments from this array.

The `String` array is called `args` by default. However, you should avoid abbreviations in variable names, as they will make your code difficult to read - therefore, I recommend that you use `arguments` as the name of the array, as you can see in Example 2:

```
public static void main(String[] arguments){
     /*
     * JRE starts by executing any code in here
     */
}
```
Example 2

Both the codes in Example 1 and Example 2 are recognized by the JRE. Also, one thing about the main method that you might find interesting is that you don't even need to use an array - you can replace the array by a parameter of variable length - "vargs" in short:

```
public static void main(String... arguments){
     /*
     * JRE starts by executing any code in here
     */
}
```
Example 3

The "vargs parameter" shown in Example 3 is like a more flexible version of an array - if you directly call this method, for example from a test, it has the advantage of accepting a variable number of `String` arguments, for example `main("BMW", "Porsche", "Mercedes")`, without having to create an array upfront. To be honest, I never really use a vargs parameter for the main method, but it I think it is a nice detail to know and show off ;-).

The static main method that we use as an access point is very specific. If you modify it beyond the ways I've discussed, it won't work as you intend it to. If you want to drive your colleagues nuts ;-), you are free to deviate from that pattern, for example by making the method `int` instead of `void`, as shown in Example 4:

```
public int main(String[] arguments) {
     return 42;
}
```
Example 4

This will create a method of the name `main`, but it won't be recognized as "THE" main method, and therefore the program won't be able to run using this method as a starting point.

Coding Examples

In Example 5, we will create a class called `CarSelector` and add a main method to it. It prints out each of the command line arguments back to the console:

```
package com.marcusbiel.java8course;

public class CarSelector {

public static void main(String[] arguments){
    for (String argument: arguments) {
        System.out.println("processing car: " + argument);
    }
}
```

Example 5

With the help of the main method we can execute this code without using its test to call it, as we have done up until this point in this course.

Compiling using the Command Line

To run our program from the command line, we must first navigate to the root folder of our source code, as I show in Example 6. In our case, this is src/main/java. As a side note, this is the default folder structure for "Maven", a build management tool I highlighted earlier when I talked about Java Tools. This is how I would do this on a unix terminal:

```
marcus-macbook-pro:~ marcus$ cd src
marcus-macbook-pro:src marcus$ cd main
marcus-macbook-pro:main marcus$ cd java
```

Example 6

As the full name of our class is `com.marcusbiel.java8course.car.CarSelector`, the Java source file is stored in a subfolder "`com/marcusbiel/java8course/car/`". To compile the code we type:

```
marcus-macbook-pro:java marcus$ javac
com/marcusbiel/java8course/car/CarSelector.java
```

Example 7

This will create a file called `CarSelector.class` in the same folder as `CarSelector.java`,

and we can finally execute our program:

```
marcus-macbook-pro:java marcus$ java
com/marcusbiel/java8course/car/CarSelector
```

Example 8

Alternatively, we could also refer to the class by using it's fully classified Java name:

```
marcus-macbook-pro:java marcus$ java
com.marcusbiel.java8course.car.CarSelector
```

Example 9

As you can see, calling our class without any arguments actually does nothing. So let's add some arguments:

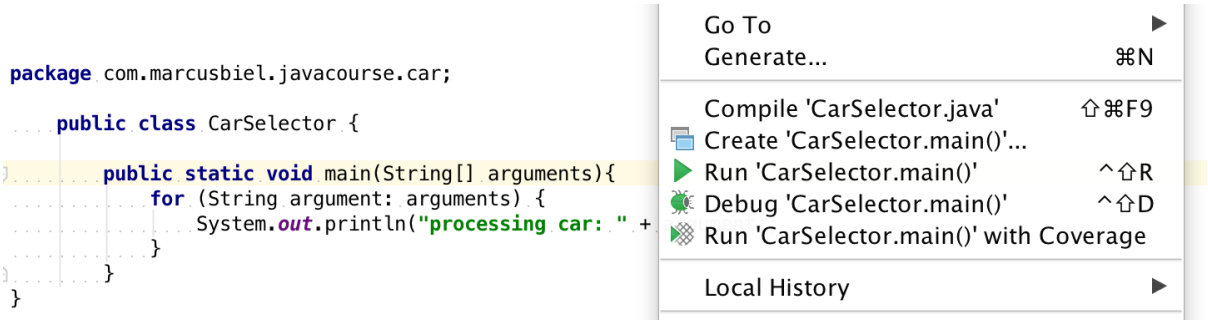
```
marcus-macbook-pro:java marcus$ java com.marcusbiel.java8course.car.CarSelector BMW
Porsche Mercedes
processing car: BMW
processing car: Porsche
processing car: Mercedes
```

Example 10

Hooray! We have successfully executed our own program from the console!

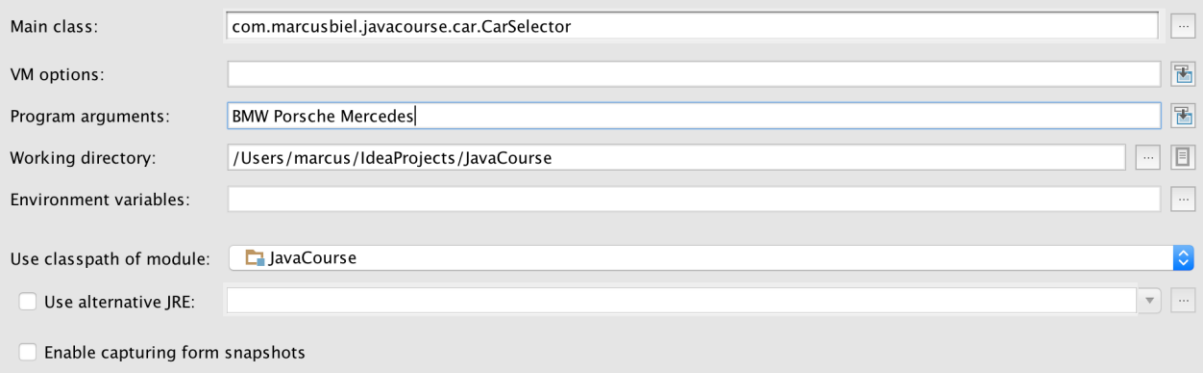
Running your Program using IntelliJ IDEA

To run our program from IntelliJ IDEA, we simply right click the method, and choose "Run 'CarSelector.main'" from the context menu, as shown in Example 11.



Example 11

If we change the signature of the `main()` method, the “Run ‘CarSelector.main’” command will disappear from the context menu, as we no longer will have a valid entry point. However, when we run it, nothing is printed. This is because no one is passing the `main()` method any arguments. To do that in the IDE: from the “Run” menu, choose “edit configurations…”, and in the “configuration” tab add space separated strings to “Program Parameters”.



Example 12

Now when we run the `main()` method, we see our cars printed out:

```
processing car: BMW
processing car: Porsche
processing car: Mercedes
```

Example 13

Commentary

If you’ve completed another Java course before this one, or even if this is your first course, you might be wondering why I have deferred the introduction of the `main()` method until this relatively advanced stage in the course. I did this for a few reasons. Firstly, I believe that it’s important to give you the tools to completely understand something before I introduce it. If you didn’t know what `public static void` meant, or you didn’t know what an array was, it wouldn’t have been fair to teach it to you. Now that you have some knowledge about all these things, you can begin to fully understand how this method works.

Another reason I chose to delay this discussion for so long is because in object oriented development, static variables and methods should be used sparsely. There are some instances where you would use static modifier, but I don't want to promote its use in this course.

Finally, you will rarely have to write a main method yourself. For every program (of any size) there is only one main method, and by the time you've joined a project, it probably was already written by someone else.

Thanks for reading!