

Introduction to Java
CS9053 Section I
Thursday 6:00 PM – 8:30 PM
Release: January 25th, 2024
Due Date: Feb. 1st, 11:59 PM
Prof. Dean Christakos

Assignment 1

This assignment will give you some practice on the basics of what you'll be doing in your more complex Java Assignments – setting up a program, printing output, receiving input/arguments, and doing basic calculations.

Part 0 (Ungraded)

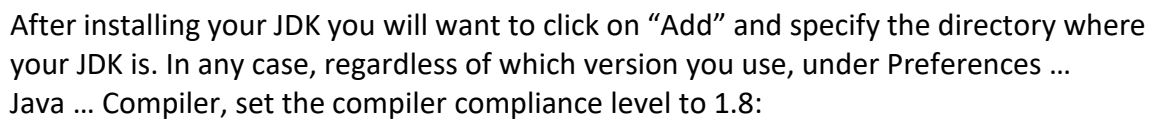
1. This is to get you started. I will be using the Eclipse IDE available at <http://eclipse.org>. I am using the latest release downloaded from here: <https://www.eclipse.org/downloads/packages/release>. Remember that you are downloading the Java Standard Edition IDE, not the Enterprise Edition.

The IDE (Integrated Development Environment) is the application where you will be writing and executing your code. However, the compiler is separate. Eclipse does have its own built-in compiler for PC and Linux users, but MacOS users will have to install the Java Development Kit (JDK) on their own.

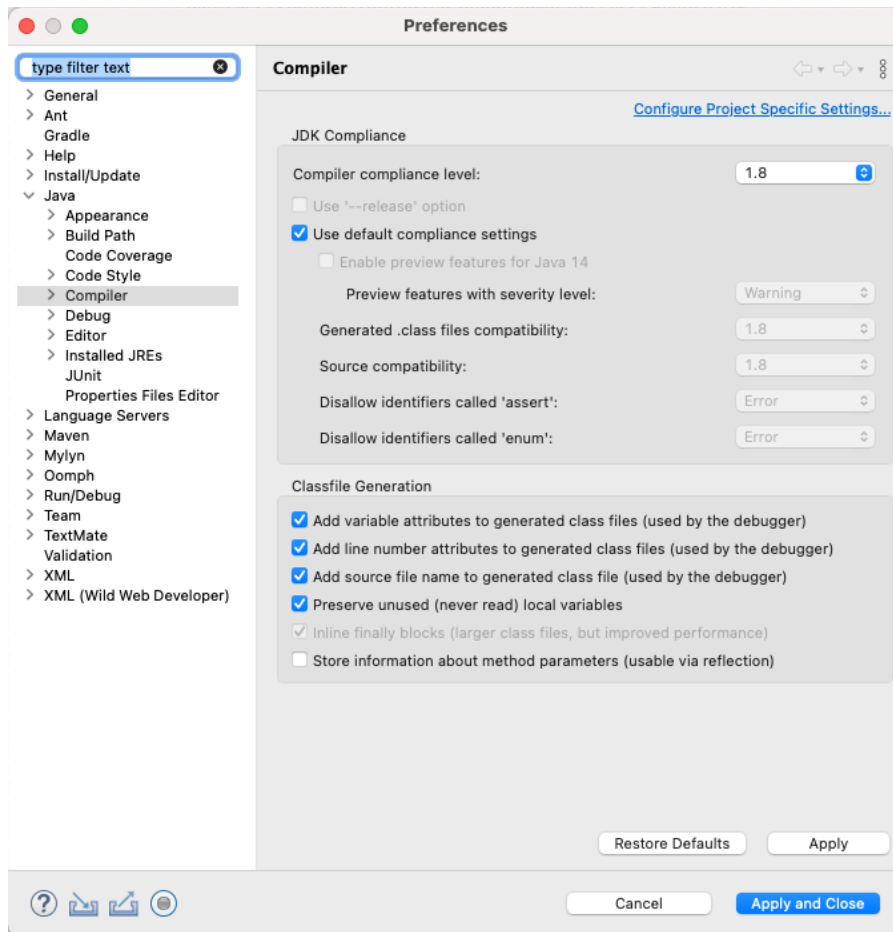
I highly suggest all of you install the JDK on your own. We will be using Java 1.8, but all JDK versions about 1.8 will be able to compile for that version. Use whatever method for downloading and installing your JDK is and put the bin directory for the JDK in your PATH. Various Java distributions are available at <https://www.oracle.com/java/technologies/downloads/>. Type “java -version” in your terminal and you should get a result like this, depending on your exact version:

```
java version "1.8.0_45"  
Java(TM) SE Runtime Environment (build 1.8.0_45-b14)  
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
```

Then in Eclipse under the Eclipse menu go to Preferences ... Java ... Installed JREs



After installing your JDK you will want to click on “Add” and specify the directory where your JDK is. In any case, regardless of which version you use, under Preferences ... Java ... Compiler, set the compiler compliance level to 1.8:



2. Now that you have Eclipse working and have your workspace set up, you'll want to get started with the Assignment 1 Code.

Download Assignment1.zip from Assignment 1 on Brightspace. There are a couple ways of importing the project into your workspace, but the easiest is to unzip Assignment1.zip into your workspace, and then Click on File ... Open Projects from File System and click on the directory Assignment1

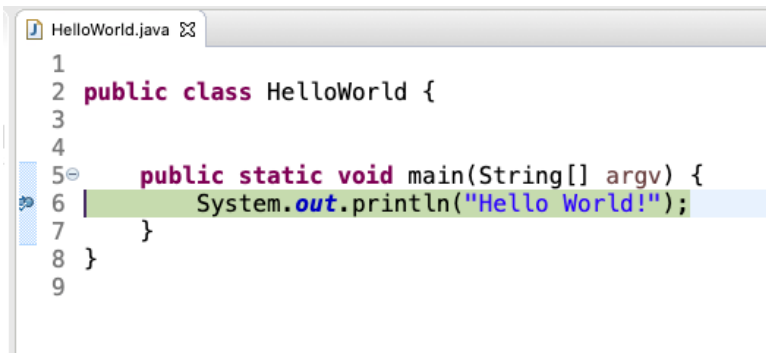
3. In the Part0 folder, open the default package, and then the HelloWorld.java file. In the file, right click on Run As ... Java Application. In the console, beneath the code, it should print "Hello, World!"

4. In the HelloWorld.java code window, move your mouse just to the left of the number six, which indicates the line number. Double click on it, and a blue dot should appear.



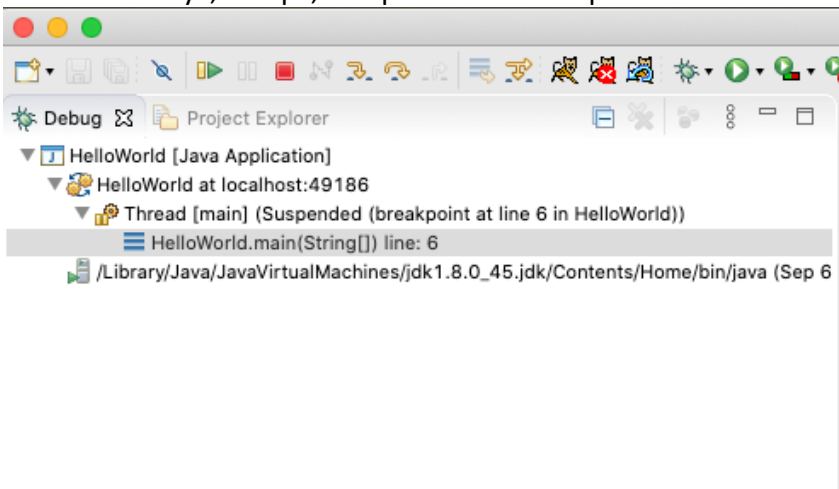
```
1
2 public class HelloWorld {
3
4
5     public static void main(String[] argv) {
6         System.out.println("Hello World!");
7     }
8 }
9
```

This is a breakpoint. It will stop the code when you're running in debug mode. Right click in the code window and click on Debug As ... Java Application. You will see something like this:



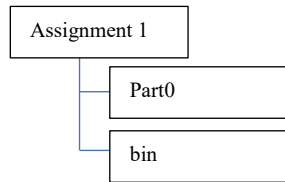
```
1
2 public class HelloWorld {
3
4
5     public static void main(String[] argv) {
6         System.out.println("Hello World!");
7     }
8 }
9
```

The code has stopped at line 6, and nothing has printed out on the console (yet). From here, there are two things you can do. If you look at the right side of the toolbar, you will see a "Play", "Stop", "Step into" and "Step over" controls:



If you press the "play" button, the program will complete. If you press the "stop" button, the program will end. If you press the "Step over" button (second arrow button after the "Stop" button), the program will print out "Hello World!" and proceed to line 7. Press it again and the program will complete.

5. Let's take a look at what's going on "under the hood." In your terminal window go to your `Assignment1` directory.



Look in the directory `Part0` and your `HelloWorld.java` code should be there. Then look in the `bin` directory. There should be a program called `HelloWorld.class`. From the `bin` directory, type `java HelloWorld`. This should output "Hello World!".

Now, delete `HelloWorld.class` and go back into the `Assignment1` directory. Execute the Java compiler with `javac -d bin/ Part0/HelloWorld.java`. In the `bin` directory you should see the `HelloWorld.class` file while you just created. This is the compiled Java bytecode program you've just created, and you can run it with `java HelloWorld` and get the same result. Obviously, Eclipse does all of this for you, but you should see how it works yourself.

(Most problems will not be 3 ½ pages long. This is just something to get you started)

Part I – Printing Output

1. Print out your initials using “big letters”. Everyone calls me “Dean”, and my middle name, the spelling of which is beyond the scope of this assignment, starts with a K, so I would print out DKC like so:

```

DDDD  K  K  CCC
D  D  K  K  C  C
D  D  K  K  C
D  D  K  K  C
D  D  K  K  C
D  D  K  K  C  C
DDDD  K  K  CCC

```

Using the `MyInitials.java` class, do the same for your initials using `System.out.println` statements. Here's a guide to big letters:

[illegible]

2. In the file `PrintResults.java`, there are a set of statements that execute mathematical operations. Using `System.out.println()` statements, print out the results of these operations that is assigned to the variables. They should have the format:

```
the value of a is [Value]
```

You can step through each of these statements using breakpoints and the debugger if you want, to see out it works, and you can see the variable values in the “expressions” tab to the right of your code window.

Part II – Input

1. Look at the file `GetNameConsole.java`. What this does is prompt you to enter your name, and it assigns what the user types to the variable “name.”

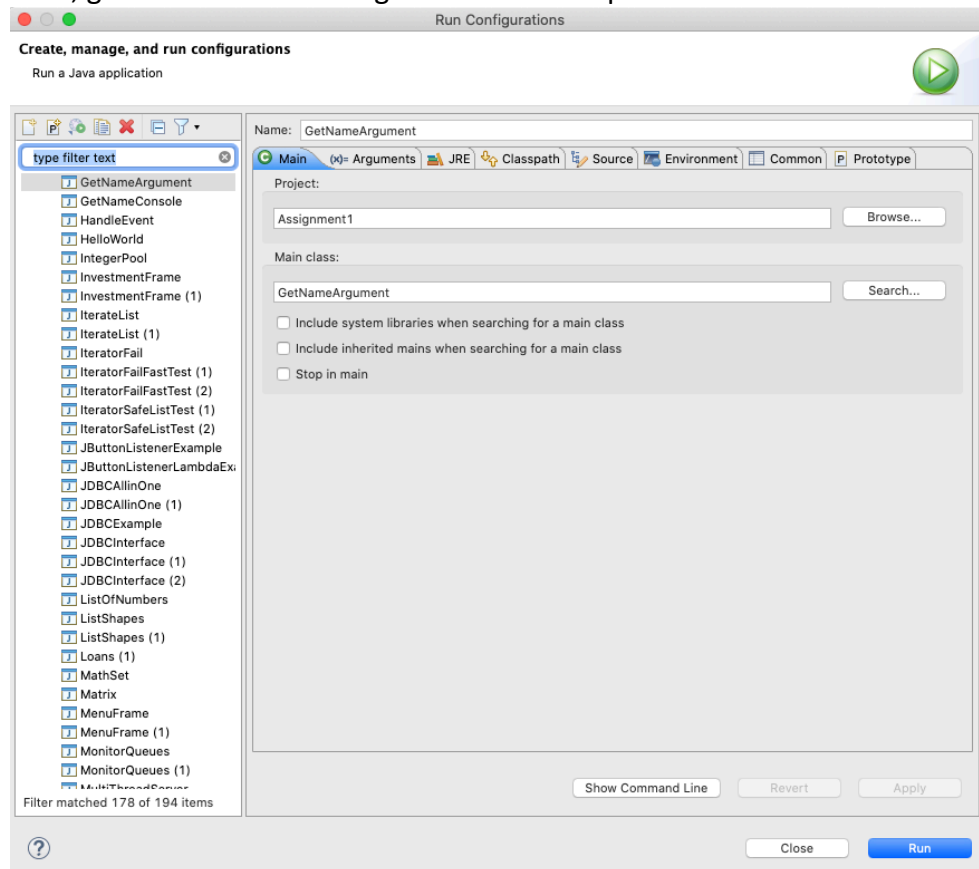
Have the program take the variable “name” and use it to print out:

```
Hello, [name]!
```

Where `[name]` is the value of the variable “name”.

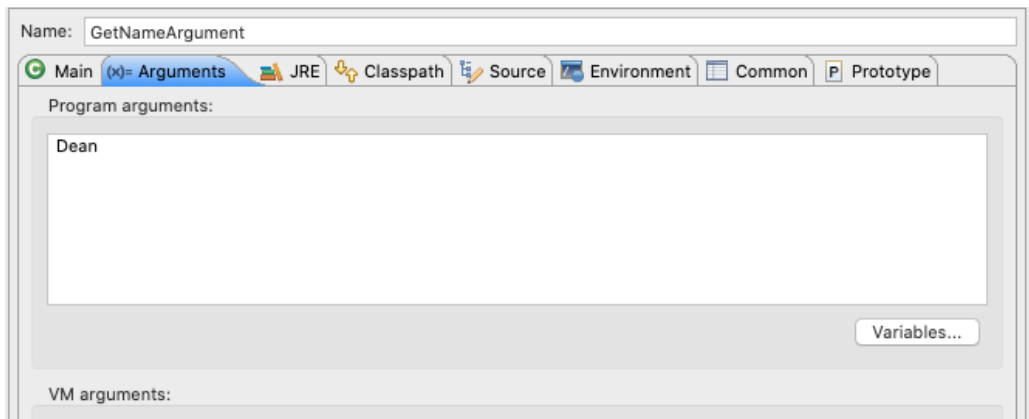
2. This will show you how to pass command-line arguments into your main method and use those in your program. First, open the file `GetNameArgument.java`

Next, go to Run ... Run Configurations. It will open this window:



You are going to create a “GetNameArgument” run configuration and associate it with the class `GetNameArgument`.

Then click on the “Arguments” tab and set the command line argument and set your name as the command line argument



We are only using one command line argument, but command line arguments are separated by spaces. The first command line argument can be accessed with `args[0]`.

Write a program that prints out “Hello [name]!” using the command line argument in `args[0]`.

IN ADDITION TO YOUR CODE, ATTACH A SCREENSHOT OF THE OUTPUT

PART III

Quick review of functions/methods:

We can quickly define a method inside a class that can be called from the main method. For example, take a Fahrenheit to Celsius conversion::

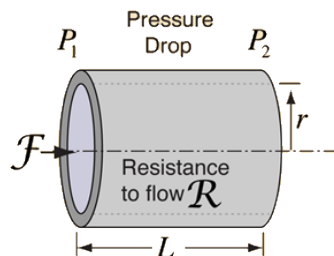
```
public class FahrenheitToCelsius {  
  
    public static double getCelsius(double fahrenheit) {  
        double celsius = (5.0 / 9) * (fahrenheit - 32);  
        return celsius;  
    }  
  
    public static void main(String[] args) {  
  
        double Fahrenheit = 75;  
        // Convert Fahrenheit to Celsius  
        double celsius = FahrenheitToCelsius.getCelsius(fahrenheit);  
        System.out.println("Fahrenheit " + fahrenheit + " is " +  
            celsius + " in Celsius");  
    }  
}
```

For now, let's just look at the method prototype and explain:

```
public static double getCelsius(double fahrenheit)
```

“**public**” is the visibility. This means it can be accessed from any other class. “**static**” (which is optional) means that it is not part of an object and can be called directly from the main method, which is also static. “**double**” refers to the return type. If it returns nothing, we can use “**void**.” “**getCelsius**” is the method name. Finally, within the parentheses, we have a comma separated list of parameters, with their types followed by variable names.

1. Poiseuille’s Law gives the flow rate of a fluid through a pipe:



This is given by the equation $Q = \frac{(P_2 - P_1)\pi r^4}{8\eta L}$, where P is pressure, r is the radius of the tube, η (eta) is the viscosity of the liquid, and L is the length of the tube.

Variable	Meaning	Value
r	radius of a typical home water supply pipe – 1 in. (in meters)	.0127m
L	Length of pipe	5m
η	Dynamic viscosity of water	$8.9 \cdot 10^{-4}$ Pa·s

Q will be the flow rate in m^3/s of water

Assume the pressure difference here will be 22,000 Pa.

Convert Q to liters/sec. (1 liter = 1000 cm^3)

`PrintResults.java` should show you how to do basic math, but nothing too complicated in terms of Java syntax is required. You should do the calculation for the `flowRate` variable by calling the `calculateFlowRate` method which will return the flow rate, and then the `System.out.println` statement that is already there should print out the correct result after you convert it to liters/sec from m^3/s . `calculateFlowRate` should take as parameters the radius, pipe length, dynamic viscosity, and pressure difference.

I've given you the value of π by using the Java's `Math.PI` constant and assigning it to a variable called `pi`.

Also, Java accepts scientific notation. You can see that `dynamicViscosity` is written as `8.9E-4`