

**ELEC 279 - Winter 2018**  
Introduction to Object-Oriented Programming  
**Lab 1 - Week 2**

**GENERAL INFORMATION**

Welcome to ELEC 279 Lab Session where you will learn object-oriented programming (OOP) by practicing and work as a team. Throughout the lab sessions, you will be working in the *pair programming model*<sup>1</sup> - usually two (2) programmers per workbench and in rare cases, a group of three (3) programmers will be permitted at the approval of a *Graduate Teaching Assistant* coordinating the lab. Please, arrive early to the lab and get settled to start working on the lab - arriving 15 minutes after the start of a lab is considered late and could affect your participation point for the lab.

**Pair Programming Model:** In order to promote an equal learning environment, we will be evaluating your participation and results in the labs, which means every member of your group must participate. In the pair programming model, each of the two team members can either be the *driver* or the *navigator* and their roles are:

- *the driver*: the person typing at the keyboard
- *the navigator*: the person with the "eagle eyes," watching for mistakes and/or typos.

One of you can only assume one role at a time, and you can take turns to interchange the role during each lab session. An example is, for Lab 1 Task 1, one person assumes the *driver* role while the other person serves as the *navigator*. And for Task 2, the *navigator* for Task 1 becomes the *driver*, and vice versa. This is the model we will adopt for this course throughout the lab sessions in this semester.

**Lab Grading and Assessment:** Note that your TAs are here in this Lab to help you learn. Feel free to ask questions as the TAs will be going around to make sure all members of the group are participating. For each Lab assessment, YOU MUST demonstrate your results for each **Task** to get credit for that **Task** - do not wait until the end to show all the tasks, demonstrate each task to the TAs as you progress. Remember, the total Labs is worth 15% of your final grade.

**Lab 1 Objectives:** At the end of this Introductory Lab session (Lab 1), you would have accomplished the following:

1. Get up and running with all the tools and the environment needed to learn OOP with Java in this course.
2. Get started with working in a team or pair programming environment - as an aspiring programmer or computer scientist/engineer, you will be working in teams, it is good to get started now.
3. Get started with writing and executing basic Java program - designing and implementing Java classes.

---

<sup>1</sup><https://tinyurl.com/ydf9mwos>

**Icebreaker Time:** In the next 2 minutes, have a non-formal chat or discussion with your new team member and get to know each other.

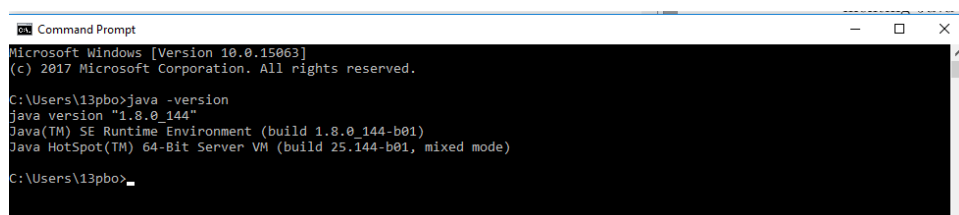
**Enable onQ Message:** Sign in to onQ, click on your name on the top right corner → click on Notifications → check the checkbox next to “News - item updated” and “News - item available.” It will be a good idea to set notifications for other course related items, so that you don’t miss out on updates and due dates.

## LAB 1 - TASKS

**Task 1. Tools Installation and Setup:** In this task, you will install and setup all the tools that will enable you to write and execute Java programs on your laptop and the workbench in the lab.

### 1. Checking Java Version:

- Open the command line or Terminal on your machine:
  - on Terminals in this Lab studio, click on the Windows or Start button to the bottom left corner then go to Windows System folder and click on “Command Prompt.”
  - on your other Windows PC, click on the Windows or Start button to the bottom left corner and search for “cmd” and click on “Command Prompt.”
  - on macOS, click on Launchpad and search for “Terminal”
- Type “java -version.” If Java Platform (JDK) is already installed, you will get the following output.



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\13pbo>java -version
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)
Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)

C:\Users\13pbo>
```

```
MacBook-Pro-2:~ User$ java -version

java version "1.8.0_11"

Java(TM) SE Runtime Environment (build 1.8.0_11-b12)

Java HotSpot(TM) 64-Bit Server VM (build 25.11-b03, mixed mode)
```

- Skip step 2 if you get the above output showing that Java SDK is already installed and up-to-date.

### 2. Installing Java SDK:

- Go to Oracle

- Download the latest version (current version: Java Platform (JDK) 8u111 / 8u112).
- Choose the right file for your platform (macOS or Windows).
- Install Java JDK and follow the instructions
- Check if the installation is successful by performing step 1 above.

### 3. Installing Eclipse as your Java IDE:

- Create a new folder (*preferably in C: or Z: drive for future use in this lab and safety of your files*) to include the files that you will create and develop. Call it “**elec279workspace**”
- Go to Eclipse
- Download the latest eclipse version (current: Eclipse Oxygen - *as the time of writing this Lab Manual*)
- Install Eclipse for Java development on your machine - **make sure you select Eclipse for Java Developers.**
- Follow the instructions. When it prompts for workspace directory, choose the “**elec279workspace**” folder you have created.
- Run Eclipse, and create a project in step 4 to test the installation and get credit for Task 1.

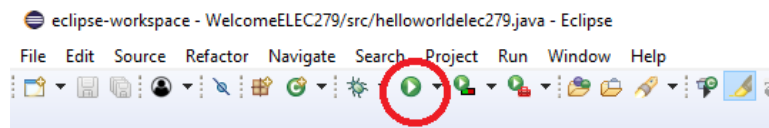
### 4. Testing with Hello World:

- Open Eclipse, go to **File > New > Java Project**, create a new project called “**Lab1Task1**”:
- When prompted, save the project to “**elec279workspace**” workspace folder you created earlier.
- While the project folder is opened in Eclipse, create a new Java class file named “**helloworldelec279**”: **File > New > Class**
- Open the class file and type the following code snippet:

```
public class helloworldelec279 {
    public static void main(String[] args) {
        //This is a comment line
        //This line output to the screen
        System.out.println(“Hello My friends in ELEC 279”);
    }
}
```

- Build and run your project: **Run > Run** or Click on the button shown below:

**CHECKPOINT:** Show that your team has completed this task by executing the “**Lab1Task1**” project and show the output to the Grad. TA for credit.



**N.B:** If the above setup is completed only on the workbench in the lab, it is recommended that, at your convenient time, you setup the above environment on your laptop as well because you will need your laptop for individual *quizzes* later on in this course.

**Task 2. Designing and Implementing a Class:** In this task, you will be creating a new project and implement a Java class to execute some basic Java code. You will begin by creating a Java class that will have main method and print some text to screen.

**STOP.** It is time to switch *driver* and *navigator* roles if you haven't done so.

1. Creating a Java Class:

- Create a project in Eclipse called “**Lab1task2**”
- Inside the project, create a new class file named “**Navigator.java**” and define the class - **The class name should be exactly the same as the file name:**

```
public class Navigator {  
    ...  
}
```

- Inside the class created above, create the main method:

```
public static void main(String[] args) {  
    ...  
}
```

**Note:** The logical collection of different parts of a large program is called package. The key word “**public**” means that the class is available across packages. The argument (**String [] args**) to the main method shows that when the class Navigator is executed, an array of strings will be sent to the program for the initialization. As this course progresses, this will become obvious and understandable.

- To print and display text to screen, add the following statement inside the main method:

```
System.out.println(“The Navigator is now the Driver”);
```

- Execute and run the project using the IDE.

2. Executing Java program in Command prompt or Terminal:

- Open command prompt or Terminal (mac OS)
- Change directory to where your project in Step 1 is stored:

- On Windows: type “dir” command to see your current directory and type “cd yourDirectory” to change directory
- On MAC: type “ls” command to see your current directory and type “cd yourDirectory” to change directory
- Compile your program using the command: `javac Navigator.java`. This will create another file “**Navigator.class**” with the .class extension.
- Run the .class file using: `java Navigator`.
  - If you get any error that “javac” is not recognized, add Java’s /bin to your system path using the following steps:
    - \* Go to Control Panel → System → Advanced System Settings → Environment Variables.
    - \* In the bottom panel, locate and select Path, click on Edit, and then Add a new entry as:

```
C:\Program Files\Java\jdk1.8.0_101\bin
```

**STOP.** It is time to switch *driver* and *navigator* roles if you haven’t done so.

3. Importing and Using other Classes: the program written above is the simplest Java program that one can write. As you progress through this course, you will be using other classes within your own project or class. For instance, the **System** class in the **System.out.println** provides access to the system’s standard input and output streams. Now, let us implement a new Java program called “**DateApp**”, which will use another system class called “**Date**”. The Date class provides access to system-independent date functionality.

- Create a new project called “**DateApp**”
- Inside the new project folder, create a new class file called “**DateApptask**” and implement its appropriate class
- Create the main method inside the class
- Now import the Date class (system class) at the top of your implemented class - your declaration should look something like this:

```
import java.util.Date;
public class DateApptask {
    ...
}
```

- Inside the main method, declare an **object** called “**today'sdate**”:

```
Date today'sdate = new Date();
```

This line of code declares, instantiates and initializes an object called “**today'sdate**”. The constructor **Date()** used to initialize “**today'sdate**” is the default constructor which initializes a new **Date** object that contains the current day and time.

- Print the current date and time to screen:

```
System.out.println(todaysdate);
```

**CHECKPOINT:** Execute the “**Lab1Task2**” and the “**DateApp**” projects in both the IDE and the command prompt. Show the outputs to the Grad. TA for credit.

**Task 3. Variables, Loops and Input/Output (I/O)** Often times, we need to store some values for our program to use or manipulate during run-time; the named storage for these values are called variables. Also, there are times we need to run a block of code multiple times - this is where Loops become handy. In this task, we will see how variables and loops work.

1. Variable Types: without closing your “**DateApp**” project, declare the following variables inside your main function:

```
int min = 10, max = 20, average = 5; // Initialize the variables
String myrole = "Driver";
byte myfirstByte = 22; // Initializes a byte type
double pi = 3.14159; // Declares a pi to be variable of type double
char mychar = 'N'; // Variable type Char
```

2. Add new statements to print each variable to screen:

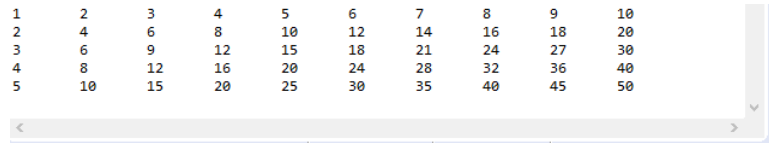
```
System.out.println("I am now the " + myrole); //Print Integer
System.out.println("Our minimum score is " + min); //Print String
System.out.println("We have a byte " + myfirstByte); //Print byte
System.out.println("And double type is " + pi); //Print double
System.out.println("A char looks like " + mychar); // print character
```

3. The **while** Loop: executes statement while the condition is true. Add the following example to your “**DateApp**” program to print current date and time 10 (ten) times. First, we declare a variable **count** that counts from 1 to 10, and prints the current date/time while count is not 10. Once we count to 10, the loop terminates:

```
int count = 1; //Our counter variable
while(count <= 10){
    System.out.println(todaysdate);
    count = count + 1; //Increment count
}
```

4. **for** Loop: the **for** loop semantics consist of the initializer, which sets counter to 1, the condition (**counter** <= 25) and the increment (**count++**)

```
for(int counter = 1; counter <= 25; counter++){
    System.out.println(todaysdate)
}
```



1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50

5. The **nested loop** - this is used to nest one loop inside of another. Supposed we want to print a 5-by-10 table shown below:

add the following code snippet to your “**DateApp**” and run the program to see the output.

```
for(int row = 1; row <= 5; row++){
    for(int column = 1; column <= 10; column++){
        System.out.print(row * column + “\t”) ; //Print each row
    }
    System.out.println(); //New row
}
```

6. Java Input and Output (I/O): So far, we have used the **System.out** to output some information to the user. Often times, we want to write Java programs that take inputs from users and also output data to the screen. To read inputs from a command line, we use the System class **Scanner**, which is defined in the Java package **java.util.Scanner**. Download the Java program **LabExampleIO.java** from **onQ** under **Lab 1 - Week 2** and save it to your working directory. Examine this Java program and read the comments to understand each line. Execute the program to test it.

**Task:** Create a new Java project called **Task6InOutExample** with a class file called **TaskInputOutput.java** that reads inputs from the command line. This should program should take two integers as input and returns the multiplication and addition of these two input integers.

**CHECKPOINT:** Execute the “**TaskInputOutput.java**” and “**DateApp**” projects again in either the IDE or the command prompt. Show the output to the Grad. TA for credit.

**COMPLETE LAB EXERCISE ON THE NEXT PAGE**

## PRACTICE EXERCISE

Now, it is time to switch roles again. In this task, you will apply the knowledge acquired so far in this lab to create new Java project each and solve the following problems. Please, take turn and solve one problem each.

1. **Time Tracker App:** Your friend works in a grocery store part time and needs an app to keep track of his/her work hours. Create a Java program that output how many hours your friend worked assuming your friend works 3 hours per day.

- Name your Project as “**WorkHourAppTask4**”
- Name the class file as “**WorkHourApp.java**”.
- Declare and initialize an integer variable called **hoursperday** and ask user to enter the value to this variable.
- Declare and initialize an integer variable called **numdays** and ask user to enter the value to this variable
- Use **for** loop to print total hours worked as you count the number of days. For instance, Day 1, Total Hours worked = 1 Day \* 3 hours = 3 hours. Day 2, Total Hours worked = 2 Days \* 3 hours = 6 hours. Print this out in a Loop until you reach the total number of days worked.

### 2. Multiplication Table App:

- Name your Project as “**Lab1ChallengeTask4**”
- Name the class file as “**Lab1Task4.java**”.
- Create necessary class and main method.
- Declare an integer variable and ask user to enter an integer value.
- Using **for** loop, write a code to print the multiplication table of the integer value up to 20.

**CHECKPOINT:** Each group member should demonstrate their result to Grad TA for credit.

**End of Lab 1**