ELEC 279 - Winter 2023
Introduction to Object-Oriented Programming
Lab 1 - Week 2

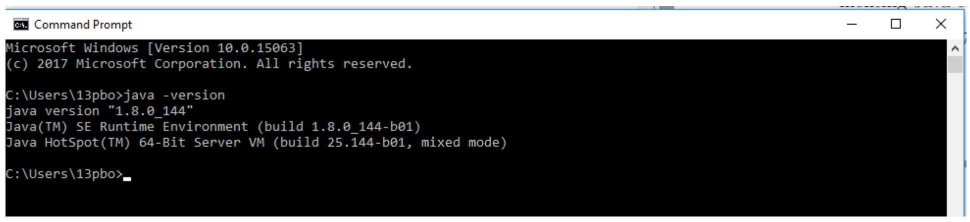
GENERAL INFORMATION

Welcome to ELEC 279 Labs where you will learn Object-Oriented Programming (OOP) by practicing. You are expected to form your own team with 2-3 students to work on the labs of this course. A team should not have more than 3 students. Working on your own is fine, but we encourage you to form a team to solve the lab questions together.

LAB 1 - TASKS

Task 1. Tools Installation and Setup: In this task, you will install and set up the tools that will enable you to write and execute Java programs on your computer.

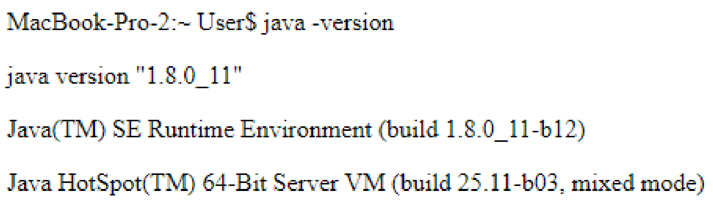
1. Checking Java Version:
   - Open the Command Window or Terminal on your computer:
     - In Microsoft Windows, you can click on the Windows or Start button to the bottom left corner, then go to Windows System folder and click on "Command Prompt." Or you can use the search box to the bottom left corner to search "Command Prompt."
     - On macOS, click on Launchpad and search for "Terminal."
   - Type "java -version" If Java Platform (JDK) is already installed, you will get something looking like the following.



```
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 below if you get the above output showing that Java is already installed and up-to-date.

2. Installing Java SDK:

- Click: Oracle
- Download the latest version (current version: Java SE Development Kit 17.0.1).
- Choose the right file for your platform (e.g., macOS or Windows).
- Install Java JDK by following the instructions.
- Check if the installation is successful by performing step 1 above. Depending on your operating system, sometimes you may need to configure your system paths, so that when you run "java -version", the computer knows where to find the "java" command.
  - For example, if you get any error indicating "java" or "javac" is not recognized, you need to add Java to your system path. As an example, if your operating system is Windows 10, you may follow the steps:
    - Search "environment variables" in the search box to the bottom left corner of your Windows screen; find "Edit Environment Variables for Your Account". Or you can go to Control Panel → System and Security → System → Advanced System Settings → Environment Variables.
    - In the bottom panel, locate and select Path, click on Edit, and then add a new entry, depending on where the *javac* and *java* executable are located. Below is an example. (After making the change, close and reopen your Command Window to make sure it knows the change made above.)

```
C:\Program Files\Common Files\Oracle\Java\javapath
```

3. Installing Eclipse as your Java IDE:

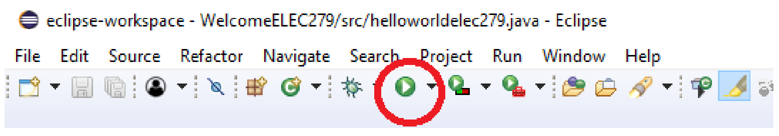
- Create a new folder on your computer. Call it "**elec279workspace**". We will use that folder to save all your Java code.
- Click: Eclipse
- Download the latest eclipse version (current: Eclipse IDE 2022-12)
- When installing Eclipse on your machine, **make sure you select "Eclipse IDE for Java Developers".**
- Follow the instructions. In most situations, you just need to follow the default setting. When it prompts for workspace directory, choose the "**elec279workspace**" folder you have created. (No worries if you did not; this can be changed later when you launch Eclipse.)
- Run Eclipse. If you see a popup window asking for workspace, use "**elec279workspace**" as your workspace. Then, you may see a window about "Welcome to the Eclipse IDE for Java Developers". There is some useful information you may want to explore, but you can also close that window if you do not need it.

4. Testing with HelloWorld:

- Once you launch Eclipse, go to **File** > **New** > **Java Project**, create a new project called "**Lab1Task1**". Follow the default configuration.
- Follow the instruction and use the default setting. It will save the project to "**elec279workspace**" workspace folder you created earlier.
- Depending on your Eclipse version, you may see a popup window for creating module-info.java. You can click either "Create" or "Don't Create". Otherwise, just use the default configuration. *finish*.
- While the project folder is opened in Eclipse, create a new Java class in the project as follows: Go to **File** > **New** > **Class**; type *package1* as the Package name; type *HelloWorld* as the Name of the class. Click *finish*.
- Open the class file and type the following code snippet:

```
package package1;
public class HelloWorld {
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:


eclipse-workspace - WelcomeELEC279/src/helloworldelec279.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Congratulations! If you have not used Java before, this is your first Java program and you have run it! You do not have to show this to TAs. Instead, after you finish all your task below, show them all together to TAs.

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.

1. Creating a Java Class:
   - Create a project in Eclipse called **"Lab1Task2"**
   - Inside the project, create a new class named **"Navigator"**:

```
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. Make sure it works.

2. 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-indepedent 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 **"todaysdate"**:

```
Date todaysdate = new Date();
```

   This line of code declares, instantiates and initializes an object called **"todays-date"**. The constructor **Date()** used to initialize **"todaysdate"** 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);
```

[**1 mark**] Execute the "**Lab1Task2**" and the "**DateApp**" projects in the IDE. Make sure they work. You need to show your results to TAs all together after you finish the tasks below.

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 class:

```
int min = 10, max = 20, average = 5;  // Initialize the variables
String myrole = ``Driver'';
byte myfirstByte = 22;          // Initializes a byte type
double pi = 3.14159; // Delcares 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
```

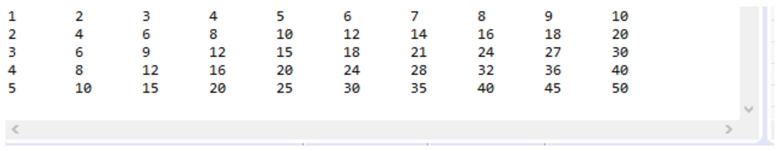
3. The **while** Loop: executes statement while the condition is true. Add the following example to your **"DateApp"** program to print current date and time 10 (ten) times. First, we declare a variable **count** that counts from 1 to 10, and prints the current date/time while count is not 10. Once we 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 count to 1, the condition (**count** <= 25) and the increment (**count++**)

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

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:



```
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
```

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.

[**1 mark**] Execute the "**TaskInputOutput.java**" and "**DateApp**" projects in the IDE. Make sure they work. You need to show your results to TAs all together after you finish the task below.

Task 4. In this task, you will apply the knowledge acquired so far in this lab to create a new Java project and solve the following problems.

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. 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.

[**1 mark**] If you finish the above last task, you receive 1 mark.

Demonstrate all your results to TAs to get your marks.

End of Lab 1! Congratulations!!