ELEC 279 - Winter 2020
Introduction to Object-Oriented Programming
Lab 1 - Week 2

GENERAL INFORMATION

Welcome to the first ELEC 279 Lab where you will learn to apply object-oriented programming (OOP) concepts. During the scheduled lab sessions you will have time to work on the lab assessment with the help of the Teaching Assistants and Instructor. You will be placed into breakout Zoom groups with a Teaching Assistant. The instructor will come around and check on the groups and help answer questions.

Lab Grading and Assessment: Attending the scheduled lab session with the instructor and TAs is not mandatory, but **it is mandatory to submit each lab assessment**. Lab assessments are **due Thursdays by Midnight** in the week they are assigned. Feel free to complete the lab assessment on your own, or you can attempt it before the scheduled lab sessions then come with questions. For each Lab assessment, YOU MUST submit your results for each **Task** to get credit for that **Task**. Remember, the total Labs are 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 writing and executing basic Java program - designing and implementing Java classes.

Enable onQ Notifications: I will routinely post course news in the ANNOUNCEMENTS section on the course homepage. Turn your notifications on so that you receive emails each time I post a news item.

Sign in to onQ, click on your name in the top right corner → click on Notifications → check the checkbox next to "News - item updated" and "News - item available." It is a good idea to set notifications for other course related items, so that you don't miss out on updates and due dates.

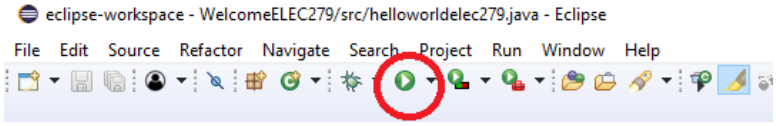
LAB 1 - TASKS

Getting Started: Create a new folder (wherever you would like) and name it "LastName_FirstName_StudentNumber_Lab1" where you fill in the corresponding placeholders with your name and student number. This folder, which I will be referring to as your submission folder, will be where you put elements from each task for final submission on OnQ.

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 computer. If you followed along with Week 1 Video 2 and installed Java SDK and Eclipse you can skip to step 3. It is not required that you use Eclipse, but I will be referring to it specifically in the lecture videos and labs. You may use any IDE so long as the output file has the .java extension.

1. Installing Java SDK:
   - Go to Oracle
   - Download the latest version (current version: Java SE 15.0.1). If you already have the latest version skip to step 2.
   - Choose the right file for your platform (macOS or Windows).
   - Install Java JDK and follow the instructions
2. Installing Eclipse as your Java IDE:
   - Go to Eclipse
   - Download the latest eclipse version (current: Eclipse IDE 2020-12 - *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, create a directory called "**elec279workspace**".
   - Run Eclipse, and create a project in step 4 to test the installation and get credit for Task 1.
3. 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 ELEC 279");
}
}
```



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

CHECKPOINT: Take a screen capture or snip (using the snipping tool in windows, or shift-command-5 on mac) of your IDE window showing both your code and the output from running the code. Save this file in your submission folder with the name "Lab1Task1"

Task 2. Designing and Implementing a Class: Now that you have confirmed that your IDE is working it is time to make your own class. In this task, you will be creating a new project and implementing a Java class to execute some basic Java code. You will begin by creating a Java class that will have a 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 file named **"MyFirstClass.java"** and define the class - **The class name should be exactly the same as the file name:**

```
public class MyFirstClass {
...
}
```

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

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

     Note: The key words used in this class and method definition will be explained in lecture videos. It is okay if at this time you do not fully understand why you are using them. As the course progresses this will become clear.
   - To print and display text to screen, add the following statement inside the main method:

```
System.out.println(``I have created my first class");
```

   - Execute and run the project using the IDE.
2. Executing Java program in Command prompt or Terminal: In general for this course you can run all of your Java programs using the IDE. In this next step, I am going to show you how you could instead run your programs using the Command prompt or Terminal. This could be useful to you in the future for other courses/ projects.

- Open command prompt or Terminal (mac OS)
- Change directory to where the .java file for your class 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 MyFirstClass.java*. This will create another file **"MyFirstClass.class"** with the .class extension.
- Run the .class file using: *java MyFirstClass*.
  - If you get an error that "could not find or load the main class" then your class is likely not in the default package, and we need to run it differently.
    - First check that you have spelled the class correctly (capitalization matters)
    - Change directory to the src folder of your project and instead run *java InsertPackageNameHere.MyFirstClass* where you specify the package your class is in.
  - If you get an 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\jdk1.8.0_101\bin
```

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, we have seen that 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 (as you have done in the previous tasks)
   - 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 **"todaysdate"**. Note here we are using the declaration of a reference type variable as shown in lecture. The constructor **Date()** used to initialize **"todaysdate"** is the default constructor (as discussed in this weeks lecture) which initializes a new **Date** object that contains the current day and time by default.

- Print the current date and time to screen:

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

4. Java Input and Output

CHECKPOINT: Execute the "**Lab1Task2**" in the command prompt/ terminal and the "**DateApp**" projects in the IDE. Take a screen capture or snip of the command terminal showing the output for **Lab1Task2** and a capture/ snip of the IDE (showing both your code and the output) for **DateApp**. Name these files Lab1Task2 and DateAppTask, respectively, and add them to your submission folder.

Task 3. Variables and Loops 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;
int max = 20;
int average = 5;  // Initialize the variables
String myrole = ''my first string'';
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 have made'' +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 statements until the loop condition is false. 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 (count++ would be more efficient)
}
```

4. **for** loop: Let's make a for loop to accomplish the same task in order to highlight the creation of these loops. Add the following code snippet to your **"DateApp"** program . The **for** loop semantics consist of the initializer, which sets counter to 1, the condition (**counter** <= 10) and the update (**counter++**)

```
for(int counter = 1; counter <= 10; counter++){
System.out.println(todaysdate)
}
```

5. The **nested loop** - loops can be nested inside each other. 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
}
```

CHECKPOINT: Execute the "**DateApp**" project again in either the IDE or the command prompt. Take a screen capture or a snip showing your code and the output and save it in the submission folder using the name "Lab1Task3".

Task 4. Flow Control Often times, we want to only execute certain parts of the code based on some conditions. This is flow control. In this task, we will see we can use flow control, how the use of the flow control methods differs, and how to be as efficient as possible.

1. Create a new project in Eclipse called "Lab1Task4"
2. Inside the project, create a new class file named "MealsApp.java" and define the class. **The class name should be the same as the file name**
3. Inside the class created create the main method (as you have done for the previous tasks)

4. **Switch** statement: Let's say you wanted to plan what you are having for dinner this weekend, but you are also particularly forgetful so need to remind yourself. Let's make a switch statement to help us out with this problem. Add the following code snippet to your "MealsApp" in the main method then run the program to see its output.

```
String day = "Saturday";
String meal = "";
switch(day){
    case "Saturday":
        meal = "Soup";
        break;
    case "Sunday":
        meal = "Salad";
        break;
}
System.out.println(meal);
```

5. **if-else** statement: As the weekend is only 2 days that last problem was a rather inefficient way to program a solution. Let's make an if-else statement instead. Replace the previous code snippet following code snippet to your "MealsApp" then run the program to confirm the output is the same as the previous part.

```
String day = "Saturday";
String meal = "Salad";
if(day.equals("Saturday") //see the string methods are useful
    meal = "Soup";
else
    meal = "Salad";
System.out.println(meal);
```

6. **Conditional** statement: Great we have been more efficient in our programming for this weekend meal problem. But we can be even more efficient by using the conditional operator instead of an if-else statement. Replace the previous code snippet with the following code snippet to your "MealsApp" then run the program to confirm the output is the same as the previous part.

```
String day = "Saturday";
String meal = (day.equals("Saturday") ? "Soup": "Salad");
System.out.println(meal);
```

CHECKPOINT: And with that we have reduced our weekend meal problem from 10 lines of code to 3. Execute the "**MealsApp**" project again in either the IDE or the command prompt. Take a screen capture or a snip showing both your code and the output and save it in the submission folder using the name "Lab1Task4".

Problem

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

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, 5 days a week. In later lectures we will look at how to receive input from the user to change the hours per day and number of days to make this app more useful, but this is all your "friend" needs for now.

    - Name your Project as **"WorkHourAppTask5"**
    - Name the class file as **"WorkHourApp.java"**.
    - Declare and initialize an integer variable called **hoursPerDay** and set the value to 3.
    - Declare and initialize an integer variable called **numDays** and set the value to 5
    - Use a **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.

CHECKPOINT: Execute your code, then take a screen capture or a snip showing both your code and the output and save it in the submission folder using the name "Lab1Task5".

Submitting: Now that you have completed all the tasks in Lab 1, make sure you have saved all of the files needed to the submission folder. Zip up your submission folder and upload it on OnQ. The Lab 1 submission page can be found in the content for Week 2, or under the assessments tab in assignments. The submission must be made by Thursday January 21 by midnight (23:59 EST).

End of Lab 1