ELEC 279 - Winter 2021
Introduction to Object-Oriented Programming
Lab 2 - Week 3

Simple Classes in JAVA

Welcome to the second 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.

1 Introduction

In this lab you will learn to apply the basics of classes and encapsulation. You can think of classes as "blueprints" for objects. They specify the instance variables (i.e., instance variables) and methods (behaviors) of an object. These 2 components are called class members, and have the following properties.

1. Instance Variables
    - are some variables or objects attached to the object of the class type
    - are instantiated when an object of the class is created with the **new** operator
    - are declared outside (typically before) methods
    - are accessible by any method of that class
    - are typically set to private
2. Methods
    - are functions attached to the object of a class
    - constructors are important methods that are used to instantiate the instance variables of an object
    - methods can be public or private. ie accessed by anyone or only by the class
    - provide the main functionality of an object

2 Getting Started

This Lab includes test code that you must run to get full marks. There are a couple of steps that must be completed first to set up the lab in Eclipse (or your IDE of choice). Like last lab, create a new folder (wherever you would like) and name it "LastName_FirstName_StudentNumber_Lab2" 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.

1. Open Eclipse
2. Start by creating a project called **"Lab2"**
   (a) Select **File→New→Java Project** to create a new project
   (b) Name the project **"Lab2"**
   (c) Click finish

3 Classes in Java

Object-Oriented Programming is based on the creation of classes of objects. Java (and Eclipse) make it vary easy to start writing classes right away. We will start by creating the main class for this lab. It contains test code to evaluate the classes you will be making.

1. Select **File→New→Class** to add a new class
2. Under **Name** enter **"Lab2"**
3. The class can be in any package of your choosing, just **make sure all of the classes from this lab are in the same package**.
4. Click finish
5. Go to OnQ. Underneath this file in Week 3 there will be a text file called "Lab 2 test code". Download that file, open it, then copy and paste the text into your new class, replacing the class declaration (which your IDE may have automatically created), and not deleting the package statement (if the class is not in the default package).

For this lab you must make two other classes. The first class will be called "Activity" and is used to store dates. We will walk you through the basics of setting up this class in Java using Eclipse.

1. Create a new class in you project and call it **"Activity"**
2. As we wish to use this class in other files ensure that it is public (i.e. use the modifier "public" in front of the class declaration)
3. Within the class brackets add three private instance variables:

   (a) private int day;
   (b) private int month;
   (c) private int year;

4. It is good practice to make your instance variables private. This prevents anyone (the users typically) from changing an object's information which could cause problems for your programs. Additionally, through the use of encapsulation techniques we can check that all changes are acceptable. In our case we will have to ensure that the date is always a valid one. so that people can't input more than the number of days in a month or more than the number of months in a year.

5. Our next step is to add a constructor

6. After your instance variables type in **Activity(){}** this will be used to override the default constructor so that the instance variables are not simply initialized with their default values.

7. Within this constructor set all of the instance variables to be equal to 1

8. Add another constructor, **Activity(int day, int month, int year){}** this will be used to initialize our instance variables to the values passed into the constructor.

9. Following **this** (hint, hint) set the year, then the month, then the day to their respective input data. Ensure that you check the date entered is valid. For this example assume that a date is only invalid if the day is not between 1 and 31, month is not between 1 and 12, or the year is negative. If the day, month, or year is invalid create an error statement using: System.out.println("Error: Date invalid.), and **set the invalid instance variable(s) to 1**. **The valid variables should still be set with their respective parameters**.

10. Now add 3 methods to get the values of each of the instance variables. These methods are called accessors (or getters) as they access our encapsulated (private) instance variables.
    (a) public int getDay(){}
    (b) public int getMonth(){}
    (c) public int getYear(){}

11. Fill in these methods to return their matching instance variable value.

12. Now we will add a mutator (set) method that sets the date to something else.

13. Write the method **public void setDate(int day, int month, int year){}**

14. Update the instance variables with these values only if it is a valid date. For this example again assume that a date is only invalid if the day is not between 1 and 31, month is not between 1 and 12, or the year is negative. If the day, month, or year is invalid create an error statement using: System.out.println("Error: Date is invalid), and **set the invalid instance variable(s) to 1**. **The valid variables should still be set with their respective parameters**.

15. Last, add a method to print the contents of our object. Name the method **public void printDate(){}** and have it print **"day.month.year"**, where the day month and year of the object are used. Hint: you will need to use string concatenation to get the output in this form

16. In the main method of the Lab2 class uncomment the following line of code: **testActivity();**

Stop Here Run your code. (note there are currently errors associated with the next class you will be making, since you haven't made it yet. Click proceed (in Eclipse) and let the program run). Take a screenshot (or snip) of your **Activity** class and the output. Save it with the name "Lab2Task1" and place it in your submission folder. (If the output and code do not fit in a single screenshot take multiple screenshots and give them informative names)

4 Expanding on Our Class

Having a class that stores the date can be very useful, but we can add more to our class! you will now create a class **"Stock"**. That will pass the test code *testStock()*.

With that done lets get to building our stock class. It is a class that describes the daily value of a stock for each of the hours the market is open. It will have all of the instance variables from our previous class (day, month, year) and two new ones. These new variables will be the stock name and the previous and current value of the stock.

1. First create a class and name it **"Stock"**

2. Copy over the contents of the **"Activity"** class to **"Stock"** class and remove the constructors (**but keep the other methods**)

3. With the other instance variables add three more.

   (a) **private String name;**

   (b) **private int previousValue;**

   (c) **private int currentValue;**

4. Add a new default constructor **Stock(){}**

5. As before have the default constructor set the instance variables for the date to 1s, but then set the name to the empty String "", and the stock values to 0.

6. Now add a constructor that will initialize all the instance variables

7. add the constructor **Stock(int day, int month, int year, String name, int previousValue, int currentValue){}**

8. Check that the date is valid (following the guidelines from the previous task), and that value (previous and current) of the stocks is not a negative number. We can't have a stock that is worth less than nothing. If any of the parameters are invalid print an error method using System.out.println("Error: invalid entry").

9. Set all the valid instance variables accordingly. Otherwise, set the invalid instance variables to the default value (1 for dates, 0 for stock values).

10. Now we must add accessors for our new instance variables

11. First create the method **public String getName(){}** and have it return the name

12. Then create **public int getPreviousValue(){}**

13. Then create **public int getCurrentValue(){}**

14. Now we will add mutators for our new instance variables

    (a) **public void setName(String name){}** changes the name atribute

    (b) **public void setCurrentValue(int currentValue){}** changes the value of the instance variable previousValue to the value of the instance variable currentValue, and changes the value of the instance variable currentValue to the parameter passed to the method. If the value entered is invalid (less than 0) the previous value should be set to the instance variable currentValue, and then the instance variable currentValue should be set to 0. An error message should also be printed using System.out.println("Error: Invalid value");

15. Add a print method **public void printStock(){}** that displays all instance variables of the class in a print statement. The statement should be in the form:
    name: day.month.year curr: currentValue prev: previousValue"

16. Lastly, add a method **public int getChange(){}** that returns the difference between the current value of the stock and the previous value of the stock.

17. In the main method of the Lab2 class add a comment to suppress **testActivity();** and uncomment **testStock();**.

Stop Here Run your code. Take a screenshot (or snip) of your **Stock** class and the output. Save it with the name "Lab2Task2". (If the output and code do not fit in a single screenshot take multiple screenshots and give them informative names)

Submitting: Now that you have completed all the tasks in Lab 2, make sure you have saved the two (or more) screen shots needed to the submission folder. Zip up your submission folder and upload it on OnQ. The Lab 2 submission page can be found in the content for Week 3, or under the assessments tab in assignments. The submission must be made by Thursday January 28 by midnight (23:59 EST).