CMSC 255 Introduction to Programming

Introduction to Objects and Class Implementation Lab

Goals:

- To understand the concepts of classes, objects and encapsulation
- To implement instance variables, methods and constructors
- To design a simple test class for a Java class file

In object-oriented programming, an object is a data type that has structure and state for its data fields. Additionally, methods provide operations that may access or manipulate the state of the object. The grouping of data and the operations that apply to that data forms a new data type. Objects of the data type are then used in programs to implement problem solutions. In this lab activity you will be completing the partial implementation of a Java class and then writing a program that uses objects of the class to test your implementation.

A class definition provides the blueprint that is used to create objects of that data type. The design of a class supports encapsulation by declaring the instance variables private and providing public accessor and mutator methods. Private instance variables of a class are visible by all the methods of the class, but are not visible outside of the class. Testing in the mutator methods further supports encapsulation. If an argument to a mutator method is invalid, an exception is thrown. It is up to the program that is creating and using the objects to appropriately handle the exceptional condition.

Part 1

- For this lab activity you will be completing the implementation of a class that represents a simple date. Download the file, **SimpleDate.java**, from the assignment link in Blackboard. The SimpleDate class stores the month, day, and year as integer values. As you work through the code for the SimpleDate class, support encapsulation and uphold the public interface that is specified in the comment blocks for each method.
- Save, compile, and debug any compilation errors before continuing with the next part of the assignment. Please note that you **cannot run** this file. It does not have a main method.
- Get the TA to check you off that your edits are complete and correct.

Part 2

- As you worked through the SimpleDate.java file, you noticed what is considered correct date values according to this code.
- You will now formulate test data that will test each path of this code, both correct and incorrect paths. Document your test data in your test plan along with the results you expect to get. The goal is to test the methods of SimpleDate, so when you create the test cases make sure you call the default constructor on one of them and call the parameterized constructor for the others. Be sure to pick values for the month, day, and year that will provide the opportunity to catch errors in your SimpleDate class.
- You should have at least seven test cases with expected results. Please have at least three test cases that will actually build a correct SimpleDate object.
- Get the TA to approve your test plan

Part 3

- In the same directory as your SimpleDate.java source code file, create a Java class called **SimpleDateTester**. This class will have a main method and in the main method you are to create a collection of SimpleDate objects using an ArrayList. The SimpleDate class is a programmer-created Java data type, so you'll need to include it as the generic type parameter for the ArrayList.

- Create a while loop to add the data from your test cases to create the SimpleDate as the objects of the ArrayList collection. Some of this data will cause the SimpleDate class to reject the construction of the object, and that is fine.
- Use try/catch blocks within the while loop to handle exceptions that might be thrown from SimpleDate so your program doesn't crash. Display a message for each caught exception.
- Once your loop has constructed the SimpleDate objects it can and has ended, display the contents of the collection by writing an enhanced for loop to iterate over the collection, calling the toString() method of each SimpleDate element.
- Create a new SimpleDate object to test the accessor and mutator methods. Create the code to modify this object. You will use the SimpleDate methods to increment the month, day, and year for each of the objects. For example, to increment the year for an instance of the SimpleDate object called `birthday` use the methods of SimpleDate:

```
birthday.setYear(birthday.getYear()++);
```

- Demonstrate to your TA and upload your completed files**, SimpleDate.java** and **SimpleDateTester.java**, to the Assignment link in Blackboard.