CPSC 1110 – LAB 05

More Inheritance (Chapter 9) and Interfaces (Chapter 10)

In this lab we will implement a simple interface type `Sequence`. This is problem **P10.2** from the book. Your primary task in this lab will be to implement a `PrimeSequence` class that must implement the `Sequence` interface. Worked example 10.1 has been uploaded to UTC Learn as a starting point for your code. Note that there are 5 .java files (`StarterCode.zip`) as well as a pdf document `Worked Example 10.1.pdf`. Additionally, you will be doing a step by step process to practice a bit more with both inheritance and interfaces in the second section of this lab. You may use BlueJ or Eclipse to complete the lab. **PLEASE COMMENT YOUR CODE.** You will have points taken off if you do not comment your code. You can see sample comments in my starter code for how you should comment your code. Keep your code neat.

You should zip all of your .java files, as well as a PDF containing screen shots of your output to submit to UTC Learn.

Some useful links:

BlueJ tutorial www.bluej.org/tutorial/tutorial-201.pdf
Java tutorial home page: http://docs.oracle.com/javase/tutorial/
Start here: http://docs.oracle.com/javase/tutorial/java/index.html
Arrays http://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html
Array Lists http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html
Inheritance http://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
Interfaces http://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html

Some helpful tips:

1) Compile often – do it.
2) You are only responsible for creating a single class, `PrimeSequence`. However, your understanding of how to do this will depend on your understanding of the provided starter code.
3) It may be helpful to use the Debugger or print statements to check your work. (always).
4) We will have a demo/tester class for this lab that tests the `SquareSequence` and `RandomSequence` classes (provided in the starter code). You will need to modify the tester to additionally test your `PrimeSequence` class. The tester class is named `SequenceDemo.java`

Tasks: Follow the directions below to complete your lab assignment

For today's lab we will be completing **Exercise P10.2** from the book.

P10.2 Write a class `PrimeSequence` that implements the `Sequence` interface of Worked Example 10.1, and produces the sequence of prime numbers. (There is a PDF file provided that has Worked example 10.1).

To start this lab you will need to take a look at the starter code, as well as the Worked Example 10.1 document that explains the project.

Next, you will need to create a new project, and run the starter code from UTC Learn (`StarterCode.zip`). (If you use eclipse you can simply make a new project, and copy/paste the starter code files into the `src` directory of your new project. Then, in eclipse, you will need to highlight the new project, and press F5 to refresh. This should populate your project explorer with the new files you just placed in the `src` directory).

At this point you should be able to run the starter code, and you should see output similar to the output generated in the Worked Example 10.1 document.

Next you will create a `PrimeSequence` class that implements the `Sequence` interface.

To implement your `next()` method, you will need to come up with a plan to generate prime numbers. Your `PrimeSequence` class method `next()` will return 2 on the first call, 3 on the second call, 5 on the third call, etc ... Also recall that a whole number greater than 1 is prime IFF (if and only if) it is divisible by only 1 and itself. So, in order to generate prime numbers, you will simply need to test each value above the previous prime number for divisibility by 2, by 3, by 4, etc …

For example, the first prime is 2. Our next prime candidate is 3. We check 3 for divisibility by 2, and see that 3 is not divisible by 2, so it is our next prime number.

Now, to generate our next prime number, we will start with 4, and check divisibility by 2. This test fails, and we move on to 5.

We test 5 for divisibility by 2, then 3, then 4. 5 is not evenly divisible by any of those values, therefore 5 is our next prime number.

You will need to implement this algorithm in a loop. You test divisibility by using the mod operator. A number `n` is divisible by a number `x`, if `n % x = 0`.

Note that this algorithm provided is very inefficient. It is a brute force algorithm. Therefore, you try to generate more than about 100,000 prime numbers it will take your algorithm a LONG time to finish. The demo class uses a value of 1000, which should run rather quickly.

You must follow the public interface outlined in this document. Follow the name conventions shown in the lab documentation. (For this lab you simply need to create a `PrimeSequence` class that implements the `Sequence` interface to adhere to my method names).

When you are finished with your project, take a screen-shots or text capture of your final output.

Part 2

Follow the directions below to complete the second part of your lab.

1) Create 3 classes, `Vehicle`, `Car`, and `Boat`. Make one of them the super class, and make the other two classes extend from this super class. Choose appropriately.
2) Create an interface, `Efficiency`, that contains a single method, `getEfficiency()`. This method should return a double.
3) Create a single instance variable, `efficiency`, in the super class from step 2.
4) Create a constructor for the super class from step two. It should accept a single parameter as a double. This parameter should be used to initialize the instance variable created in step 4.
5) Add constructors for the two subclasses that will also accept a single parameter as a double. They must call the super class constructor with this parameter.
6) Create a `printMessage()` method in each class that accepts no parameters and returns nothing. This method should print "I am a Vehicle/Car/Boat, VROOM!!!". (Choose `Vehicle`, `Car`, or `Boat` according to which class's method you are implementing).
7) Create a single `getName()` method in the super class that returns a String and accepts no parameters. This method will be inherited by the two subclasses. Return a string containing the class name.
8) Make your super class implement the `Efficiency` interface. (This will also make the subclasses implement the interface because of inheritance).
9) Create a class called `Tester` with a `main()` method.
10) Declare a constant `NUM_VEHICLES` in the `Tester` class and set its value to 10.
11) Your main method in the Tester class will have a few jobs it must do:
    a. Fill an array of type `Vehicle[]` of size `NUM_VEHICLES` with `Vehicle` objects. You need to generate a random number between 0 and 2. If the number is 0, add a Vehicle to the array. 1, add a Car, 2, add a Boat. The Vehicle/Car/Boat that you add must be initialized with a random efficiency between 0 and 100.0. Do this until you have added `NUM_VEHICLES` objects to the array.
    b. Loop through the array and call the `printMessage()` method for each object in the array. Explain in the comments before this loop why this is an example of polymorphism.
    c. Loop through this array and print the class of the object and the efficiency of the object. Your print out should look like this:

```
Boat: 64.2669
Vehicle: 69.7662
Car: 42.7745
Car: 41.8222
```

12) Create a static method in your `Tester` class called `getFirstBelowT()` that accepts an array of `Efficiency` objects and a double that represents a threshold value. This method should re-

turn an `Efficiency` object that is the first object in the array with an efficiency below the threshold parameter. Return null if no objects have an efficiency below the threshold.

13) From the main method pass the array you created in step 11a and a threshold of 20 to the `getFirstBelowT()` method. Save the return value in a `Vehicle` reference variable that you declare. You will have to cast the return value from the method to a `Vehicle` because the actual return type of the `getFirstBelowT()` method is an `Efficiency` reference variable. Use this `Vehicle` reference variable to print the type of object that had the low efficiency, as well as the efficiency. Your printout should look like this:

```
The first object with efficiency less than 20 was Boat with effi-
ciency of 12.2680
```

14) If there was no object with efficiency less than 20, you should print an appropriate message. (if the return value is null, then none of the objects in the array had an efficiency less than 20).

To Turn In via UTC Learn

You should turn in 1 .ZIP file containing your java files and a PDF document with screen-shots (or text) of your output. 1 file should be uploaded to UTC Learn. ***IMPORTANT!!!*** You should name your file in the following manner. lastname-firstname-lab05.zip. So John Smith would submit smith-john-lab05.zip.