Objective

This lab extends the work you did last week in Lab 1 by having you write two additional classes that implement the Time interface and design JUnit tests for classes.

General Idea

The goal of the first two labs is to learn how to write multiple implementations of a data type. The data type we consider is Time which stores hours, minutes, and seconds. The key concept is that the client program (in this case the class containing the main method) does not need to be aware of the particular implementation of the data type being used.

Instructions:

1. Open the project you worked with in Lab 1 last week. You should have the interface Time.java, the class TimeA.java that implements the interface, the class Relay.java that contains the client code, and the two JUnit test classes TimeATestAdd.java and TimeATestToString.java in your project. Make sure your code is correct and your tests run without error.

2. Add a third constructor to TimeA that initializes the TimeA object with a string in the format “hh:mm:ss”. Properly document the constructor with Javadoc comments.

   Create a JUnit test class named TimeAConstructorTests that tests each of the three constructors. Include at least three tests for each of the three constructors for a total of at least nine tests. Take a snapshot of the JUnit summary window (green bar and test case names) and include it with your lab writeup. Document the class with proper Javadoc documentation and then print the test case class and include it with your writeup.

3. The purpose of using an interface is to be able to create a new class that implements Time in a different way. The class TimeA implements Time by storing three int values: hours, minutes, and seconds. An alternative implementation stores just the total number of seconds.

   Implement the Time interface using a new class TimeB. The easiest way to begin is to select the TimeA.java file and choose to Save As... and name the new file TimeB.java. When you look at TimeB, you will have errors because the file name and class name do not agree. If you left-click on the error flag, and elect to change the type name to TimeB, these errors will disappear. At this point,
you should be able to change all of the TimeA's in Relay.java to TimeB's and the program should work as before.

[This is because there is no difference in the code - at the moment.]

4. Now you will need to change the private instance variables in the class TimeB to the new implementation. Instead of storing the number of hours, number of minutes, and number of seconds, use a single int variable to store the total number of seconds. (Caution with your naming here, it would be best to name this variable so it is distinct from seconds that was used in the first implementation.) There are other changes required to the class. In particular, you will need to change the constructors and the getters. If things are done well, the other changes should be minimal (or none at all).

Once you have completed the TimeB class go to the Relay class and change all of the types to TimeB. After you have made the necessary changes, run the program. **Describe the results.**

5. Create a class TimeBTests that has JUnit tests to test the TimeB class. Include a minimum of four tests of the toString method and a minimum of four tests of the add method. In your tests for the add method include tests where the sum of the seconds carries into the minutes but the sum of the minutes does not carry into the hours, where the sum of the minutes carries into the hours but the sum of the seconds does not carry into the minutes, and where the sum of the seconds carries into the minutes which in turn triggers the sum of the minutes to carry into the hours.

Take a snipshot of the JUnit summary window (green bar and test case names) and include it with your lab writeup. Document the class with proper Javadoc documentation and then print the test case class and include it with your writeup.

6. Now you will implement the Time interface using a third representation of time. Name this class TimeC. TimeC will represent a time using a single string in the form “hh:mm:ss”. Complete the implementation of TimeC by changing the instance variables, the code for the constructors, getters, and the other two methods.

As you did in Step 3 above, once you have completed the TimeC class go to the Relay class and change all of the types to TimeC. After you have made the necessary changes, run the program. **Describe the results.**

7. As you did in Step 4 above, create a class TimeCTests that has JUnit test to test the TimeC class. Use the same set of tests that you used in Step 4.

Take a snipshot of the JUnit summary window (green bar and test case names) and include it with your lab writeup. Document the class with proper Javadoc documentation and then print the test case class and include it with your writeup.

8. The purpose of this lab is to show how "replaceable parts" can work. Thus, set the array in Relay.java to use Time objects and store a TimeA into one of the three array cells, a TimeB into
another array cell, and a $TimeC$ into the third cell. Run the program. What happens? To the best of your ability, explain why.

9. For each of the files document the modifications you made to the code and include your name(s) and date. When you are done, print out the current versions of all of your files.

Hand in:

Hand in a cover page, a document answering the questions from Steps 4, 6, and 8, the snapshots requested in Steps 2, 5, and 7, and the code listings from Steps 2, 5, 7, and 9.

Due Date:

The report is due at class time on Monday, February 1.

Assignment Type (see Academic Practices and Policies Document):

Help Policy in Effect for This Assignment: Group Project with Limited Collaboration

In particular, you may discuss the assignment and concepts related to the assignment with the following persons, in addition to an instructor in this course: any member of your group; any St. Bonaventure Computer Science instructor; and any student enrolled in CS 132.

You may use the following materials produced by other students: materials produced by members of your group.