Lab 2 - Time and Again

David B. Levine
Computer Science Department
St. Bonaventure University

Objective

This lab reviews some concepts from CS 131 and introduces multiple implementations of an *interface*. This lab also introduces JUnit testing.

General Idea

The goal of this lab 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 being used.

Instructions:

1. Start Eclipse on your machine. Make sure that you are in your own workspace. Create a new project named Relays. Copy the files Relay.java, TimeA.java, TimeATestAdd.java, TimeATestToString.java, and Time.java into the project.

2. First open Time.java. This file contains a Java interface. How many methods does this file define? What interface does this file extend?

   Now open TimeA.java. Compare the overall structure of the two files. Describe the differences you see in the contents and structure of the files. Do you see any reference to *TimeA* in Time.java? If so, describe where it occurs. Do you see any reference to *Time* in TimeA.java? If so, describe where it occurs.

   Note: *TimeA* is a ***class*** that ***implements*** the ***interface*** *Time*.

3. The file TimeA.java should be showing a number of errors. What is the cause of these errors? Fix them.

4. Open the file Relay.java and look at it. Notice that it contains only a main method. In its current form, it is designed to store three *TimeA*'s in an array and then print them out. Run this main. What output is produced? Look at the code in TimeA.java and explain why this output is produced.

5. In general, all classes in Java (including *TimeA*) should have a toString method. While TimeA does have such a method, it is perhaps less than helpful. Rewrite the method so that it does the correct thing.

   We are going to be using **JUnit** testing framework to test the code to see if it does the correct thing. TimeATestToString.java contains two tests of the toString method. Open

TimeATestToString.java. You will have a syntax error since you have not yet added the JUnit library. Click on the yellow light bulb on the left margin of the first appearance of `@Test` in the file. Select the suggestion **Add JUnit 4 library to the build path**. Clicking **OK** should clear up all of the syntax errors in this file.

Run the test program TimeATestToString.java as a JUnit test. This will open a new window in the left panel with either a green bar or a red bar. The green bar indicates that both tests are correct meaning that you fixed toString correctly. If you get a red bar then one or both tests fail and you should be able to identify which test(s) failed. Verify that you have a correct toString so far.

Actually, you should also be testing times that include zeros. Add at least two new tests to the test case to cover this eventuality and once again verify that you get a green bar. Take a snipshot of the JUnit summary window (green bar and test case names) and include it with your lab writeup. Print the modified test case class and include it with your writeup as well.

6. Next, we will examine the add method. Run the JUnit test case TestTimeAAdd.java. What kind of results do you get? Given the code in the add method, can you explain this? Fix the add method in TimeA. Also, add at least two more tests to the test case so that it tests the method more thoroughly. Verify that everything is good and once again, take a snipshot of the JUnit summary window (green bar and test case names) and include it with your lab writeup. Print the modified test case class and include it with your writeup as well.

7. Now modify the main method in the *Relay* class so that in addition to printing the various times, it also prints their sum (the sum of all 3 times) with an appropriate label.

8. One great benefit of using an interface is having the ability to create a new class that implements *Time* in a different way. Implement the Time class 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.]

9. Now you will need to change the private 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 variable to store the total number of seconds. 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.

10. Now go to your two test cases and change all of the TimeA's to TimeB's. Verify that both your add and your toString methods work properly.

11. Next, 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.

12. Now change the array so that it holds *Time* objects (rather than *TimeB* or *TimeA* as earlier). Does the program behave correctly? Why or why not? Now, instead change the array to hold *TimeA*

objects even though you are creating *TimeB*. What goes wrong? Why? Finally, try to create a *Time* object as one of the types. What happens?

13. The purpose of this lab is to show how "replaceable parts" can work. Thus, set the array to use *Time* objects and store a *TimeA* into ONE (any one) of the three array cells while leaving the other two with a *TimeB*. Run the program. What happens? To the best of your ability, explain why.

14. Print each of the files that you modified (TimeA.java, TimeB.java, and Relay.java) and include them with your report.

Hand in:

Hand in a cover page, a document answering the questions from Steps 2, 3, 4, 5, 6, 11, 12, and 13 and the files requested in Steps 5, 6, and 14. (Note: you should have printed five files of code.)

Due Date:

The report is due at 4 p.m. on Friday, January 24.

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.