Section 4: 
Testing, Inheritance, and Interfaces

Your section leader should have told you to pair up with a new student, someone different than you paired with the last three weeks. This is the last week we’ll require new partners!

Decide who will be the first driver, and let’s get started.

PART I: Writing Test Cases

As part of Program #3, you needed to write your own testing class for objects created by your BinaryNumber class. The purpose of this part of this week’s section is to encourage you to think some more about choosing test cases. Remember, both documentation and test cases are best created before code is written.

1. Get out a sheet of paper, or open a word processor — your choice. This part will ask you several questions, and you’ll want to write/type your answers.

2. Imagine that we need a class named OrderedStrings. Each OrderedStrings object stores and manipulates a sorted list (small to large, alphabetically) of String objects. Here is the set of public methods (including the constructor) of this class:

   • OrderedStrings() – creates an OrderedStrings object containing no strings but with a maximum capacity of 5 strings.
   • size() – returns the quantity of strings in this OrderedStrings object’s list.
   • location(String) – returns the position of the given String object in the list (first position is 1, second is 2, etc.). If the string isn’t in the OrderedStrings() object, location() returns 0.
   • insert(String) – added the given String object to this OrderedStrings object’s list, in the correct location. If the string already exists in the list, or the list is full, insert() returns false, otherwise it returns true.

Let’s assume that size() works correctly. Under that assumption, how can you use size() to test that the constructor is working correctly? (Make note of your answer; your SL will want to know.)

3. Hopefully, that one was easy; there’s just one correct state of a new OrderedStrings() object. Now let’s think about testing location(). Is there anything about the correct operation of location() that we can test without using insert()? Again, make note of your answer.

4. As your answer to the last question indicates, in order to fully test location() we need to use (and therefore start testing) insert(), too. This co-dependence of methods is common in testing, and a pain when debugging — a test that gives an incorrect result could be caused by either method, or by an incorrect test!

For the moment, assume that insert() works as it should. How can we use insert() to more completely test location()?

(Continued …)
5. Finally, we can think about `insert()`. Because calling this method can change the order of the values (remember, the list needs to be sorted alphabetically), testing it will be more involved than the testing of the other methods.

Take several minutes to discuss (and write down!) different insertion situations `insert()` must handle correctly, and how you can use `size()` and `location()` to check that that each were actually performed correctly.

6. One last question: Did you need to know anything other than the documentation for these methods to plan their testing? (That is, did you need to know how the list is going to be stored? What the instance variables will be? Whether or not `size()` re-counts the elements in the list on each call or remembers the size between calls?)

![CHECKPOINT 1]

Raise your hand. Your SL will come over and ask you about your answers to some of those questions.

PART II: A Little Bit of Inheritance

_You are remembering to switch drivers, aren’t you?_

_We’ve just learned what inheritance is, and its value in an object-oriented language in support of code reuse. In this part, you’ll use inheritance to base a new class on an existing class._

1. On the class web page is `Points.java`, which is the program for this part of the section activity. Load it into DrJava.

2. If you try to compile it (go ahead), you’ll find that it doesn’t compile. A look at `main()` will tell you that it’s expecting to find a class named `ThreeDPoint`, which doesn’t yet exist. We have a class named `TwoDPoint`, though, which is in the same file.

3. We _could_ create the `ThreeDPoint` class from scratch, but a little thought will show that a three-dimensional point class is just a simple extension of a two-dimensional point class. What do we need to add to `TwoDPoint` to allow it to handle a three-dimensional point?

4. Below the `TwoDPoint` class, create a class named `ThreeDPoint`. In addition to needing to incorporate your answers to the last question, `ThreeDPoint` needs to:
   
   (a) Extend (inherit from) `TwoDPoint`.
   
   (b) Have its own constructor that:
      
      i. Accepts three coordinates instead of two
      
      ii. Calls `TwoDPoint`’s constructor (using the “super()” notation used in `T04n04.java`) to initialize the first two coordinates
      
      iii. Initializes the third coordinate

Don’t forget to add in the details covering your answers to the question in point #3!

5. With those details in place, your `ThreeDPoint` class should allow `Part3` to compile, run, and produce the expected output.

![CHECKPOINT 2]

Raise your hand. Your SL will come over and verify that you’ve completed the program.

(Continued …)
PART III: Creating a Serializable Class (and Working with Arrays of Objects!)

We recently learned that, before we can write objects to a binary file, we must mark the class as being serializable (that is, as being expressible as a sequence of bytes). In this part of today’s activity, you and your partner will work with a class that creates virtual measuring cups (those stackable metal or plastic sized cups used by cooks to measure dry ingredients, gradated from small (say, 1/8th of a cup) up to a full cup), and will create a serializable class for a measuring cup set.

1. Visit the class web page, find the Fraction.java file in the Topic 1 section of the collection of example programs, and load it into DrJava.

2. In the Section Activity area are files named MeasuringCup.java and MeasuringCupSet.java. You guessed it – you need to load them into DrJava, too.

3. The MeasuringCup class doesn’t have a lot of functionality; just one constructor and two getters. You can see that MeasuringCup is using a Fraction object to hold the capacity of a measuring cup. We’ve completed the constructor for you, but we do have a question: Are MeasuringCup objects composed of Fraction objects, or are MeasuringCup objects adapting Fraction objects? Make note of your answer; your SL will want to know!

4. Complete the stubs of the getNumerator() and getDenominator() methods. This will be straightforward if you take advantage of the functionality that a Fraction object provides.

5. Time to turn our attention to the MeasuringCupSet class. Its job is to be the ‘factory’ that creates objects that hold sets of measuring cups. It, too, is currently incomplete; you need to complete it by:

   (a) Completing the stub of the constructor. A new MeasuringCupSet object is to hold no measuring cups, but needs an array capable of holding MAX_CUPS cups.

   (b) Completing the getCup() method. It’s a getter that returns a reference to the cup at the given index. Don’t worry about error checking; that is, assume the argument is within the array’s boundaries.

   (c) Completing the addACup() method. It should add the given cup reference to the end of the existing collection of cup references. Again, don’t worry about error situations.

   (d) Telling Java that MeasuringCupSet is serializable. To do this, add “implements Serializable” after “class MeasuringCupSet”, and don’t forget to import Serializable.

6. Make sure that you can compile MeasuringCup and MeasuringCupSet without errors!

   ✔ CHECKPOINT 3 Raise your hand. Your SL will come over, ask about your answer to the question, and look at your completed classes.

PART IV: Clean Up!

1. Log out of your computer.

2. Pick up your papers, writing implements, cell phones, trash, etc.

3. Push in your chair(s).

   ✔ CHECKPOINT 4 Raise your hand. Your SL will come over and give your workspace the white-glove test . . . if s/he happens to have a white glove handy.

You’re free to go! But, if you have time, think about writing a program that creates a set of measuring cups, and writes the objects of the set to a binary file.