


Section 5: Binary Files of Objects, and Inheritance

Starting with this section, you may pair up with anyone who is agreeable to pairing up with you. That's right; no need to find someone new! Decide who will be the first driver, and let's get started.

PART I: Creating a Serializable Class

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 revisit the measuring cup object idea from last week's section, 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 1** Raise your hand. Your SL will come over and look at your answer to the question and your completed classes.

(Continued ...)

PART II: Writing an Object to a Binary File

Last week we used a program that created a pair of binary files, so that we could compare their sizes. Now, you will write (most of) a program that writes a `MeasuringCupSet` object to a binary file.


1. In the Section Activity area is a file named `Section5.java`. Load it into DrJava, alongside the files from Part I (we still need them).
2. As you can see, it doesn't have a lot of content. There are brief comments stating what needs to be added, but here are more details:
 - (a) Add to the set the six cup capacities given in the comment ($1/4$, $1/3$, $1/2$, $2/3$, $3/4$ and $1/1$). Recall that `MeasuringCupSet` has an instance method for adding cups.
 - (b) To verify that the cups were properly added, write a loop that gets the references to the cups and uses them to display the capacities of all of the cups in the set. The output format is up to you.
 - (c) Using the information presented in class last Wednesday, use just one call to the `writeObject()` method of `ObjectOutputStream` to write the object referenced by `set` to a file named `section5.out`. Don't forget to appropriately handle (e.g., print the stack trace) checked exceptions. You may put all of the `ObjectOutputStream`-related calls in one try block.
3. Compile `Section5` and, of course, fix any compilation errors.
4. Try to run `Section5`. After fixing other run-time errors, you will eventually see one like this:

```
java.io.NotSerializableException: MeasuringCup
```

Here's the problem: We made `MeasuringCupSet` serializable, but it depends on `MeasuringCup`, which in turn depends on `Fraction`. To write a `MeasuringCupSet` object to a file, all objects it contains must have been created by classes that are also be serializable.

Modify those classes so that they, too, are serializable.

5. Now your code should compile and run. Use the `ls -l` command in a terminal window to check that the `section5.out` file was created. How many bytes does the file contain?

 **CHECKPOINT 2** Raise your hand. Your SL will come over and check your work.

PART III: A Little Bit of Inheritance

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?

(Continued ...)

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 we covered in class Monday) 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 3** Raise your hand. Your SL will come over and verify that you've completed the program.

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 see if your workspace is literally squeaky clean.

You're free to go! But, if you have time, we recommend that you use it to work some more on the programming assignment (individually, of course), if you're not already done.