CMPSC 111
Introduction to Computer Science I
Fall 2015

Practical 6
16 October 2015
Due in Bitbucket by midnight on 23 October 2015
"Checkmark" grade

Summary

As a means of both practicing the extension of your own classes and methods and better understanding the structure of different Java classes, you will study the given Java files and modify them by adding more functionality to certain methods. Additionally, you will use the Lightweight Java Visualizer (LJV) to automatically create diagrams, like those seen in Chapter 4, that depict the state of Java objects. Then, using the "`git add`", "`git commit`", and "`git push`" commands you will upload, to your Git repository hosted by Bitbucket, the modified source code, object visualizations, and the output you obtain from running the `Practical6` program.

Review the Textbook

Be sure to read Sections 4.1 through 4.4 of your textbook to learn more about writing your own classes, constructors, and methods. Please ensure that, as you read these sections, you study the technical diagrams of an object's state, as found on pages 167 and 180 of the textbook. As you review this material, try to make a list of questions about concepts that you do not yet fully understand. In addition to discussing these questions with the teaching assistants and the course instructor, please take your own steps to answering them as you complete this assignment.

Save and Study the Provided Classes

Using the "`git pull`" command, download the files `Octopus.java`, `Utensil.java`, and `Practical6.java` from the course repository. Using the strategy that was described in a previous assignment, please copy these files into your own Bitbucket repository in a directory called `practical06/`. Study these programs first and make sure you understand them. Can you better understand the structure and behavior of the methods provided by these classes by drawing a technical diagram like the one in Figure 4.7 of your textbook? To compile them, you can type "`javac Practical6.java`" in your terminal window; to run them, please type "`java Practical6`".

Enhance the Classes

1.
   - Please find the constructor in the `Octopus.java` file. You will notice that, even though an instance of the `Octopus` class has instance variables like `weight` and `name`, these variables are not initialized by the constructor. Assuming that the value of -1, indicating that the variable has not yet been set, is an acceptable starting value for both of these variables, please add to the constructor assignment statements initializing them to -1.
   - You should also notice that the current constructor for `Octopus` only accepts the "`String n`" parameter that allows you to specify the name of the object. Since an `Octopus` also has an `age` and a `weight`, you should write a new constructor that has three formal

parameters—one for each of these instance variables—and initializes all of them correctly. You can use the "`Account`" example in Section 4.4, which we discussed in class, as a source of inspiration for the way in which you add an improved constructor for `Octopus`.

2.
   - Now, use GVim to edit the file `Practical6.java`. Declare a second `Octopus` variable (don't just change the name of the one that's there—create another one) and assign it any name and age that you want. When you create this instance of the `Octopus` class, you should use your newly defined constructor that takes multiple parameters.
   - Continuing to add code in the `Practical6.java` file, please create a second `Utensil` of any type you wish, imitating the declaration and initialization of `spat`. Assign a cost and a color to this utensil. Now, assign this utensil to the new `Octopus` you created.
   - Print out the name, age, weight, and favorite utensil of your new `Octopus` object. Finally, print out the type, cost, and color of your new utensil contained within the `Octopus`.

Visualize the Objects

1.
   - Please study the source code of `Practical6.java` to find the location where it uses a "static" method to configure the Lightweight Java Visualizer class called `LJV`. Now, find the method calls that produce a visualization of the `ocky` instance of the `Octopus` class and notice the name of the file that will contain the visualization. In your terminal, you can type "`evince ocky-before.pdf`" to see what `ocky` looks like in memory.
   - After you have studied the "before" visualization, please again run the `evince` program to view the file called "`ocky-after.pdf`". What are the similarities and differences between these two diagrams? How did the method calls change the `ocky` object?

2.
   - In the previous phase of this assignment, you were responsible for adding new code to `Practical6.java` that would construct another instance of the `Octopus` class. Please find this code and make sure that you understand how it works. To test your understanding, can you draw, on paper, what you think this object looks like in memory?
   - Following the example of the previous calls to the `LJV` class, please create a visualization of your new instance of the `Octopus` class. Remember, you will not need to call the `LJV` methods that configure the visualizer; instead, you only have to make a call to the `drawGraph` method that will save the diagram in a file called "`my-ocky.pdf`".

Completing the Practical Assignment

To finish this assignment and earn a "checkmark", you should submit, through your Bitbucket repository, the `Octopus.java` and `Practical6.java` files you edited. You should also upload the three PDF files, as created by running the "`java Practical6`" program, in the repository.

General Guidelines for Practical Sessions

- **Submit *Something*.** Your grade for this assignment is a "checkmark" indicating whether you did or did not complete the work and submit something to the Bitbucket repository. Please update your repository regularly and make sure to upload the final files on time.
- **Review the Honor Code Policy on the Syllabus.** Remember that while you may discuss your work with other students in the course, code that is nearly identical to, or merely variations on, the work of others will be taken as evidence of violating the Honor Code.