Exercising Simple Objects (2)

Purpose:
The purpose of this lab is to re-enforce the concept of an object: specifically, the notions of properties, queries, commands, and state.

Notes:
This lab asks you to experiment with some objects and answer questions relating to your observations. You should write up and turn in your observations as part of the post-lab, being sure to answer all questions.

We will use a terminal window to enter line commands for compilation and execution of Java programs. Review if necessary how to create terminal windows, create subdirectories, copy files, and navigate directories; as well as how to compile and execute Java programs.

Experimenting with objects:
The next few exercises will involve manipulating objects to get an idea of how an object appears to a client. In particular, we want you to understand an object’s properties, state, queries, and commands.

The programs below all contain one or more objects to exercise – model objects – and a user interface that allows you to interact with the model objects. In some cases the user interface is window-based (a graphical user interface); in other cases it is text based. The user interface acts as client to the model object.

The purpose of the user interface is to provide a way to access some or all of the functionality of the model object. One of the options of the user interface is to terminate the application. This is a user interface function, and not a feature of the model object. As you know, the user interacts with the user interface to select operations and enter data. The user interface in turn interacts with the model object to perform the operation, and queries the model object for state information to display to the user.

A temperature converter:
• Create a subdirectory of your Java directory, named thermometers.
• Copy all the files from ~labCourse/Labs/Lab4/thermometers to your thermometers directory. For instance, assuming the current working directory is Java:

    cp ~labCourse/Labs/Lab3/thermometers/* thermometers

The model object in this application is a temperature converter, that converts Celsius to Fahrenheit and vice versa. When the application is executed, the user interface presents a window with two fields, labeled “Celsius” and “Fahrenheit.” The user enters a number in one of the fields and presses Return. The user interface interacts with the model object which performs the conversion. The user interface displays the result in the other field. Note that the user interface does not provide an explicit menu of operations.

Before proceeding, stop and think about the kind of object that performs the temperature conversion. What are its properties? What are its queries? What are its commands?

• Compile ThermometerGUIStart.java and run thermometers.ThermometerGUIStart.

• Experiment with the program and answer the following questions:
  • What is the model object?
  • What does the state of the object consist of?
  • What are the queries the object responds to?
  • What are the commands?

• Compile ThermometerTUIStart.java and run thermometers.ThermometerTUIStart.

• Does the model have the same queries and commands as the one presented in ThermometerGUIStart?

A font manipulator:

• Create a subdirectory of your Java directory, named fonts1.

• Copy all the files from ~labCourse/Labs/Lab4/fonts1 to your fonts1 directory.

• Compile FontsGUIStart.java and run fonts1.FontsGUIStart.

  This application presents you with the text “Good things come in small packages.” To the left is a number with several choices. To the right is a font name, also with several choices.

• Experiment with the program and answer the following questions:
  • What is the model object?
  • What does the state of the object consist of?
  • What are the queries the object responds to?
  • What are the commands?

A simple editor:

• Create a subdirectory of your Java directory, named fonts2.

• Copy all the files from ~labCourse/Labs/Lab4/fonts2 to your fonts2 directory.

• Compile TinyEditorStart.java and run fonts2.TinyEditorStart.
This application presents you with a window in which you can enter text, select text with the mouse, and modify it using menu options. The model is much more complex than in the previous examples. Though we’ll treat the model as a single object, in actuality there would be several objects making up the model.

Think carefully about the state of the model object. What information must be maintained? How can it be modified?

- Experiment with the program and answer the following questions:
  - What is the model object?
  - What does the state of the object consist of?
  - What are the queries the object responds to?
  - What are the commands?

**Post-lab:**
Submit printed answers to the questions for each object, as directed by your lab instructor.

**A little more UNIX:**
In a previous lab, we saw how we could instruct the shell to attach a program’s standard input and output character streams to files. We can also instruct the shell to run several programs simultaneously and attach the standard output from one to the standard input of another. This is called a “pipeline” or simply a “pipe.”

To see how this works, we’ll do a few examples. The program `wc` (for “word count”) counts the number of lines, words, and characters in its input. (Typing `man wc` will give you all the details.) Like `cat`, it will read from standard input if no file is specified.

- In a terminal window, type
  ```bash
  ls -l
  ```
  and note the number of lines in the output.

- Now type
  ```bash
  ls -l | wc
  ```
  The vertical bar (|) is usually read “pipe.” This line instructs the shell to run the programs `ls` and `wc` simultaneously, and connect the standard output of `ls` to the standard input of `wc`. 
An extremely useful program is *grep*. Among other things, *grep* will output all of the lines of its input that contain a specified sequence of characters. *grep* is often used in pipelines.

- Type the following and observe the results.

```
  ls -l | grep rwx
  ls -l | grep java
  ls -l | grep drwx | wc
  ls -l | grep java | wc
```