4 Designing Methods for Complex Class Hierarchies

Portfolio Problems

Work out as complete programs the following exercises from the textbook. You need not work out all the methods, but make sure you stop only when you see that you really understand the design process.

Problems:

1. Problem 15.8 on page 175
2. Problem 15.11 on page 176

Pair Programming Assignment

4.1 Problem

This problem continues the work on mobiles we have started during the lectures. The file **mobile-methods-lecture.java** contains the data definitions, examples of data, and the method `totalWeight` we have designed in class.

A. Design the method `totalLength` that computes the length (or actually the height) of the mobile when it hangs. Be careful. Just adding up all the lengths of the strings will not work.

B. Design the method `isBalanced` that determines whether the mobile is properly balanced. For each strut (the bar on which the two other mobiles hang) you must make sure that the weight hanging off the strut multiplied by its length is the same as the weight hanging on the opposite side - multiplied by the length of its strut. If you do not understand what this means, make a simple toy mobile and play with it, to see what it takes to make it balanced.

   Make sure you design several examples of mobiles that are not balanced.

C. Design the method `draw()` that consumes a `Canvas` and a `Posn` that represents the point where the top of the mobile will hang. The methods draws the mobile with black lines for the struts, and for the hanging lines. For a simple mobile, at the end of the line there should be disk of the appropriate color and with the size proportionate to its weight shown at the end of the line.

4.2 Problem

You are trying to organize the file directories on your computer. Your friend gave you a start by designing the data definitions given in the **files-directories.java** file. She even gave you some examples of data.

A. Make an additional example(s) of data that allows us to represent the following directory and its contents:

```
Directory Pages
 contains the following: file index.html
                         file pictures.html
                         directory Pictures
                         directory Quotations
Directory Pictures
 conatains the files: home.jpeg,
                      friend.jpeg.
                      brother.jpeg
Directory Quotations contains files: twain.txt,
                                     cervantes.txt
```

Choose any sizes for these files. Assume the sizes are given in kilo-Bytes.

B. Design the method `totalSize` that computes the size of the directory by adding the sizes of all files in it. Additionally, every directory needs 4 kiloBytes for keeping track of its list of files and subdirectories.

C. Design the method `allFiles` that produces a list of all files of the given kind in the directory (and all of its subdirectories).

Note: You must include the templates for all classes in this problem, except the interfaces and classes that represent empty lists.

Note: Use helper method where appropriate.

4.3 Problem

We are given two soretd files and would like to combine them into one file. For simplicity we will deal with just files of `String`s. The `String` class defines the fillowing method for comparing two `String`s:

```
// compare this String to the given in lexicographical order
// produce an int < 0 if the this String is before the given
// produce 0 if this String is equal to the given
// produce an int > 0 if the this String isafter the given
```

A. Design the method `isSorted` that determines whether this list of `String`s is sorted lexicographically.

B. Design the method `merge` that is invoked by a sorted list of `String`s, consumes another sorted list of `String`s, and produces a new list of `String`s that contains all items from both lists in a sorted order.

Note: You must include the templates for all classes in this problem, except the interfaces and classes that represent empty lists.

Note: Use helper method where appropriate.

4.4 Problem

Creative Project

This week you will complete the game project.

A. All that is left for you to do is to design the method that determine when the world ends and to run the game.

Note: Take the time to make the code look good, be readable, well tested; make the game display a bit nicer.

When you are ready to run the game do the following:

- Change the line that defines you `GameWorld` class to be:

  ```
  class GameWorld extends World{
  ```

  Of course, you will use whatever is the name of your class that defines your world. If you have named it just `World`, you need to change its name to something different.

- Change the language level to *Intermediate ProfessorJ*.
- Include on your `Examples` class the method

  ```
  boolean go(){
    return this.myInitialWorld.bigBang(200, 300, 0.1);
  }
  ```

  — assuming you have defined `myInitialWorld` in the `Examples` class, want your `Canvas` to be 200 pixels wide and 300 pixels tall, and want the clok tick at every 0.1 second. Of course, you choose your own names, sizes, and the speed.
- Run the program, then in the **Interactions** window type the following:

  ```
  > Examples e = new Examples();
  > e.go()
  ```

- Have fun.