

3 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
3. 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.

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

Pair Programming Assignment

3.1 Problem

Start with the file **soccer-team.java**.

You are given the class definition and some sample data for classes that represent information about a youth soccer league. The league keeps a list of teams as follows. Each team is represented by its captain, the team name, and a list of additional players on the team. The captain is also considered to be a player. We also record the age of every player.

- A. Write down on a paper the names of all teams, for each team list its players, and the name of its captain. *Submit this as a comment at the end of your Examples class.*

- B. Design the method `count` that counts the total number of players in this league. Design the templates as you go along. (We will grade the templates!)
- C. Design the method `listAll` that produces a list of all players in this league.

3.2 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
  contains 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 kiloBytes.

- 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 the names of all files in any of its subdirectories, sub-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.

3.3 Problem: Extra Credit

Start with the file `excel-cells.java`.

For this problem you will use the classes that represent the values in the cells of a spreadsheet. For each cell we record the row and column where the cell is located, and the data stored in that cell. The data can either be a numerical (integer) value or a formula. Each formula can be one of three possible functions: `+` (representing addition), `mn` (producing the minimum of the two cells), or `*` (computing the product) and involves two other cells in the computation.

A. Make an example of the following spreadsheet segment:

	A	B	C	D	E
1	7	4	3	2	5
2	- A1 E1	+ B1 C1			* A2 D1
3		+ B2 B1			mn B3 D1
4		+ B3 B2			mn B4 D1
5		+ B4 B3			* A2 E4

- B. Design the method `value` that computes the value of this cell.
- C. Design the method `countFun` that computes the number of function applications needed to compute the value of this cell.
- D. Design the method `countPlus` that computes the number of `Plus` applications needed to compute the value of this cell.

Make sure you design templates, use helper methods, and follow the containment and inheritance arrows in the diagram.

3.4 Problem: Extra Credit

We are given two sorted lists and would like to combine them into one list. For simplicity we will deal with just files of `Strings`. The `String` class defines the following method for comparing two `Strings`:

```
// 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 is after the given
int compare(String that){ ... }
```

- A. Design the method `isSorted` that determines whether this list of `Strings` is sorted lexicographically.
- B. Design the method `merge` that is invoked by a sorted list of `Strings`, consumes another sorted list of `Strings`, and produces a new list of `Strings` 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.

3.5 Problem

Creative Project

Complete the design of your game by adding the methods that implement the game actions — in response to the key events, or as the clock ticks away.

Design one method at a time, make sure you follow the *Design Recipe*, and once all the parts are there, run the game (see below).

Note: I will be more impressed with a well designed simple game than with a game that has all kinds of fancy options, but the code is not readable, methods are jumbled together, there are no tests, and there are no purpose statements.

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

- Change the line that defines your `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 clock 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.