

Understanding Complex Data

Due: 2/8/2011 10:00pm

Portfolio Problems

1. Problem 15.8 on page 175

2. Design an interface (and classes) to represent a list of numbers (**`int`**s), then design the following methods. Use a helper method with an accumulator, and make sure your purpose statements explain the meaning of your accumulators:

```
// Sum all the numbers in this list
int sumElements();

// Find the maximum number in this list
int max();

// Find the minimum number in this list
int min();
```

   Hint: for the second and third methods you should start your accumulator with the *smallest* and *largest* integers respectively (think about it...). These values are handily defined in the `Integer` class, i.e., `Integer.MIN_VALUE` and `Integer.MAX_VALUE`.

Pair Programming Assignment

Important: Be sure to create the `Assignment-04/src` directory and name the files exactly as requested for each problem. *We will not grade incorrectly named/organized submissions*.

Also, make sure you design templates, use helper methods, and follow the containment and inheritance arrows in the diagram. *In this assignment we will grade the templates*.

4.1 Problem

Work out the following problems using your class hierarchy that represents lists of `City`s from the the previous homework. Again, save these problems in the file `Cities.java`, but in your repository's `Assignment-04/src` directory:

A. Design the method `totalDistance` that computes the total distance between the cities in this list. *Hint:* Remember how we used the accumulator in class (long ago)?

B. Design the method `contains` that determines whether or not this list contains the given `City`. *Hint*: how should we compare cities? What class should be responsible?

C. Design the method `hasLoop` that determines whether traveling through the cities in this list will cause us visit the `City` twice.

4.2 Problem

Start with the file `ExcelCells.java`. Save your solution in the file `ExcelCells.java` in your repository's `Assignment-04/src` directory:

For this problem we use classes that represent data in the cells of a spreadsheet. For each cell we record its row and column, where the cell is located, and the data (`IData`) stored. An (`IData`) is either a number (**`int`**) or a `Formula`. Each formula can be one of three possible functions: **`+`** (representing addition), **`min`** (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	8	3	4	6	2
2	min(A1,E1)	+(B1,C1)			*(B2,D1)
3	*(A1,A2)	+(B2,B1)			min(A3,D1)
4		+(B3,B2)			min(B4,D1)
5		+(B4,B3)			*(B5,E4)

B. Draw this spreadsheet on paper and fill in the values that should show in each cell.

C. Design the method `value` that computes the value of this cell. *Hint*: follow the recipe... examples really help.

D. Design the method `countFuns` that computes the number of function applications (`Formula`s) involved in computing the value of this `Cell`.

E. Design the method `countPlus` that computes the number of `Plus` applications needed to compute the value of this `Cell`.

4.3 Problem

Revise your solution to the problem from last assignment that dealt with bank accounts. Save these problems in the file `Banking.java`, in your repository's `Assignment-04/src` directory:

A. Define an **`abstract`** class `AAccount` and abstract the fields common to all `IAccounts`.

B. Revise the method `amtAvailable` for the classes. Can it be *lifted* to the abstract class? Or, does it have to be defined in each class that **`extends`** `AAccount`?

C. Revise the method `moreAvailable`, which determines whether this account has more available for withdrawal than the given account. Again, can this method be lifted to the **`abstract`** class? Or does it have to be defined in each class?

D. Revise the method `withdraw` that produces a new `IAccount`, same as this one, but with the given amount withdrawn. Can this method be lifted to the **`abstract`** class? Or does it have to be defined in each class?

E. Define the method `sameName` that determines whether this account has the same name as the given account. Where can this method be implemented?

4.4 Problem

It is time to have some fun. Can the chicken cross the road? It is your job to give her a chance to try.

Save your solution in `ChickenWorld.java`, in your repository's `Assignment-04/src` directory. For this problem you'll need to add the `JavaWorld-3.jar` (*Note the "3"*) library to your project and place it in your `EclipseJars` directory. See the Lab for directions and other links to set this up.

4.4.1 The Chicken Game

Your task is to create a little "*game*", where a chicken (represented by a `Circle` or a `Star` or some other shape ... be creative) tries to cross the road and avoid getting hit by cars traveling across the screen.

More precisely, create a class `ChickenWorld` that **`extends`** `World` and contains a representation of a *chicken* and a list of cars. The player (chicken) should get a number of lives, and should use arrow keys to move *left/right/up/down*. For simplicity have a fixed number of cars that move at the same speed. When a car is completely off screen it should jump back to the start on the other side. If you are feeling ambitious you can have multiple lanes moving in opposite directions.

If the chicken is struck by a car, then the player loses a life and goes back to the beginning. If all the lives are used up then you should display `"Game Over"` (or some other message) and the game should stop (see documentation for the `World` class from the link below, in particular `stopWhen` and `lastScene`).

You need to design at least three methods for `ChickenWorld`: `toDraw`, `onTick`, and `onKey`. The signatures and purpose statements are given in a comment, and more information can be found in the JavaWorld documentation:

```
http://www.ccs.neu.edu/home/chadwick/javaworld/doc/index.html
```

Be sure to *design* the methods including examples and tests before you code the method bodies. If methods get too complicated or you think the implementation should be spread over the classes then feel free to design helpers. You'll need to design methods for motion, interaction and collision detection. Make sure you design your methods well, but feel free to implement more advanced/interesting features.

`JavaWorld` library provides classes to represent `Scene`s and `Image`s with *overloaded* methods like `placeImage`, which accepts a `Posn`, or two

ints:

```
// Place an Image on this Scene at the given Posn
Scene placeImage(Image i, Posn p);

// Place an Image on this Scene at the given X/Y
Scene placeImage(Image i, int x, int y);
```

The file `ChickenWorld.java` is provided as a starter. We've given you the skeleton of the world, and a `CartPt` class that **`extends`** `Posn`. Feel free to add any helper methods needed to this class, and since every `CartPt` is also a `Posn`, they can still be passed to `placeImage`. Part of the point of this assignment is for you (and your partner) to design the data and methods required... so embrace the freedom we've given you!

Be creative! And design well!