4 Methods for Complex Class Hierarchies; Libraries

Etudes

4.1 Etude

Problem 4.5 asked you to design data definitions for files of images, text, and sounds. Problem 14.4 asked you to add methods *timeToDownload*, *smallerThan*, and *sameName*.

Design the appropriate abstract class that eliminates as much repetition in the code as possible. ∎

4.2 Etude

Problem 18.3. ∎

4.3 Etude

In problem 14.5 you designed methods *unitPrice*, *lowerUnitPrice*, and *cheaperThan* for the classes that represent the inventory in a grocery store.

Design the appropriate abstract class that eliminates as much repetition in the code as possible. ∎

4.4 Etude

Problem 19.1. ∎

Note:

If you feel that you really understand the concepts covered in these etudes, you may do only two out of four. However, make sure you know how to solve all of them.

Main Assignment — Part 1

We continue working with the classes that help us draw the city map and its attractions that we did in Assignment 2.

The code contained extensive amount of repetition. Design the abstract class that allows you to eliminate as much repetition as possible. Make sure you run the same tests as you did in the original solution.

Submit both the old and the new code. You are allowed to make changes and corections to the original code (e.g. adding examples if you did not have them before). Both versions must have a matching set of tests.

Main Assignment — Part 2: Rat Race

The goal of this exercise is to design an interactive game. The rat lives in a cage and rambles around looking for something to eat. There are globs of stuff around - some are food and some are poison. If the rat does not eat for some time, he starves to death. If the rat finds a food glob, his life expectancy increases as he eats the food. If he finds poison, he dies instantly.

Rat's life is simulated on a canvas. The rat moves in response to the key events - as the user hits the arrow keys, the rat moves in the corresponding direction. The world starts with a collection of globs of food and poison. As the timer ticks away, the rat gets hungrier and closer to death, unless he finds some food to eat.

4.5 Problem

Design the class(es) that represent the rat, and design the methods that simulate the rat's behavior:

1. Analyze the information needed to represent the rat. Make examples (in English) of several instances of a rat. Then, design the class(es) that represent the rat in this game. And we do not have to remind you that ...

2. Design the method that consumes a *String* that represents a key event and produces a rat that has moved in the direction specified by the key event. Of course, that the rat only moves if one of the appropriate keys was hit.

3. Design the method *canEat* that determines whether the rat is close enough to the given location (where, presumably, some food or poison lies).

4. Design the method *poison* that produces a dead rat.
5. Design the method *starve* that produces a new rat, one hour hungrier that this rat, possibly a dead rat.
6. Design the method *draw* that displays the rat on the given *Canvas*.
7. Design the class *RatWorld* that contains one rat. Design the methods *draw*, *erase*, *onKeyEvent* that allow you to show the rat and control the moves using the arrow keys. You will need to implement the skeleton of the method *onTick* as well — you learned how to do it in the lab.
8. Now design the *onTick* method that just invokes the *starve* method in the class(es) that represent a rat. Additionally, if the rat is dead, the game ends by returning the result of the *endOfWorld*("The rat died.") method invocation.

■

4.6 Problem

Design the classes that represents the globs of food or poison in the rat cage. Then add methods that simulate the rat's interaction with the globs (finding a glob, eating it, etc.). Note, that the rat's life expectancy increases proportionately to the amount of food in each food glob. However, any amount of poison is fatal to the rat.

1. Analyze the information needed to represent the globs. Make examples (in English) of several instances of a both kinds of globs. Then, design the classes that represent the globs in this game.
2. Design the method(s) *foundGlob* that determine whether a given rat has found this glob. (Remember the method *canEat* in the class(es) for rats?)
3. Design the method *feed* that produces a new instance of the given rat after he ate this glob. You may need to add new method to the class *Rat*, that simulates the rat eating a food glob.
4. Design the method *draw* that will display this glob in the given *Canvas*. Make sure that the player can distinguish between the food globs and poisonous globs. It would also be helpful, if the display indicated the relative sizes of the globs.

5. Add one glob to the class *RatWorld* that contains one rat. Modify the methods *draw*, *erase* to show the glob as well.

6. Now modify the method *onTick* so that if the rat is close to the glob, he will eat it.

■

4.7 Problem

Design the class(es) that represent a list of globs and complete the game design.

1. Design the classes that represent a list of globs in this game.

2. Design the method *draw* that displays this list of globs in the given *Canvas*

3. Design the method *feedRat* that produces a new rat that consumed the first glob in this list that was close enough to the rat.

4. Design the method *removeEaten* that replaces the first glob in this list that was close enough to the rat, (so that the rat ate it), with a new randomly chosen glob randomly placed it in the field.

5. Modify the *RatWorld* so that instead of one glob, it contains a list of them. Modify the *onTick* method so that the rat now has a choice of globs among all globs in a list of globs.

■