4 Abstracting over Data Definitions, Methods

Practice Problems

Practice problems help you get started, if some of the lab and lecture material is not clear. You are not required to do these problems, but make sure you understand how you would solve them. Solving them on paper is a great preparation for the exams.

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 19.4 on page 271
2. Problem 19.5 on page 271
3. Problems 19.6 - 19.11 on page 276-279

   You may complete this part with animation using the *draw* library.

Pair Programming Assignment

4.1 Problem

The file *Banking.java* contains the definitions of classes the represent bank accounts.

A. Make examples of the following accounts:

   - A checking account for Adam Smith with id 123, a minimum balance of $50 and current balance of $150.
   - A savings account for Betty Jones with id 456, a balance of $120 and interest rate of 2.5%.
   - A certificate of deposit account for Pat Malloy with id 334, a balance of $300 that has not yet matured.

B. Design the method `amtAvailable` for the classes that represent bank accounts that produces the amount that the customer can withdraw from the account.

C. Design the method `moreAvailable` that determines whether one account has more available for withdrawal than another account.

D. Design the method `withdraw` that produces a new account with the given amount withdrawn. If the amount the customer wants to withdraw exceeds the available amount, no money will be withdrawn.

   Note: In the revised version below signal that the transaction is not valid.

Revise the solution and hand in only the revised version.

A. Define an abstract class `AAccount` and lift into it all fields that are common to all accounts.

B. Revise the method `amtAvailable` for the classes that represent bank accounts: can it be lifted to the abstract class? - or does it have to be defined in each class anyway?

C. Revise the method `moreAvailable` that determines whether one account has more available for withdrawal than another account: can it be lifted to the abstract class? - or does it have to be defined in each class anyway?

D. Revise the method `withdraw` that produces a new account with the given amount withdrawn: can it be lifted to the abstract class? - or does it have to be defined in each class anyway?

   If the transaction is not valid throw an exception and describe the problem using a different message for the different kinds of accounts.

E. Define the method `sameName` that determines whether two accounts have the same name.

4.2 Problem

Design the class `CartPt` that extends the `geometry` library class `Posn`. We will not add any new fields, but by extending the library class we can add new methods to it.

Design the following methods for the class `CartPt`:

- Method `distTo` that computes the distance from this point to the given one.

- Method `fromTo` that produces a `String` that tells us in which direction do we have to travel from this point to the given one. Make sure you cover the following possibilities:

  going North, East, West, South, Northwest, Northeast, Southwest, and Southeast.

Design the class `WorldPt` that extends the `CartPt` as follows:

- In addition to the fields `x` and `y` it has two fields `int WIDTH` and `int HEIGHT` that are initialized when the class is defined. Use the width and height of your game for the two values.
- Design the method `inBounds` that consumes two `int` values (`x` and `y`) and determines whether they are within the bounds of the *canvas* of our `int WIDTH` and `int HEIGHT`.
- Modify the constructor for this class so that it throws an exception if one tries to construct a point that is not within bounds.

4.3 Problem

Make a final revision of your game. Where appropriate, add lists of game components, or other collection of objects.

The grading rubric for this problem will be as follows:

- 4 points — well designed and readable data definitions and code
- 4 points — examples of world at the start, at the end, and during the game
- 4 points — well designed methods, the design follows one task - one method rule, the methods are defined in the appropriate classes
- 4 points — tests for all methods