5 Abstracting classes: lifting fields, lifting methods, creating a super class, a union, deriving subclasses; Using libraries: The World teachpack.

5.1 Problem (16.2)

Abstract over the common fields of *Lion*, *Snake*, and *Monkey* in the class diagram of figure 1. First revise the class diagram, then define the classes including the constructors. ∎

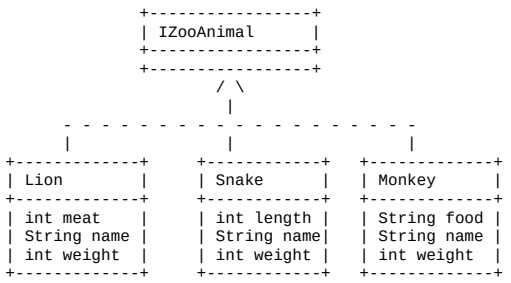

Figure 1: A class diagram for zoo animals

5.2 Problem (16.3)

Abstract over the common fields in the class diagram of figure 2. Revise the class diagram and the classes. ∎

5.3 Problem

Develop the method to compute the cost of the ride per passenger, given the number of miles traveled, and assuming the vehicle is filled to its capacity. First, develop the method for the original class hierarchy as shown in figure 2. Then, lift the method within the class hierarchy designed in the previous problem and rerun the original tests. ∎

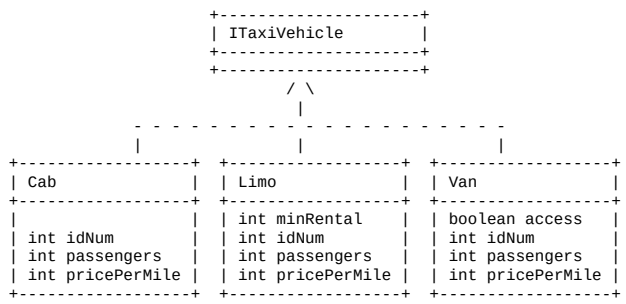

Figure 2: A class diagram for taxis

5.4 Problem (Problem 17.1)

Compare the two classes in figure 3. A *Set* is a collection of integers that contains each element at most once. A *Bag* is also a collecion of integers, but an integer may show up many times in a bag.

Your tasks are:

1. The two class definitions use lists of integers to keep track of elements. Design a representation of lists of integers with this interface:

   ```
   // a list of integers
   interface ILin {
     int howMany(int i);
   }
   ```

   The constructors are *MTin* for the empty list and *Cin* for constructing a list from an integer and an existing list.

2. Develop examples of *Set*s and *Bag*s.

3. Develop functional examples for all methods in *Set* and *Bag*. Turn them into tests.

4. The two classes clearly share a number of similarities. Create a union and lift the commonalities into an abstract superclass. Name the union interface *ICollection*. Don't forget to re-run your test suite at each step.

```
// a set of integers:
// contains an integer at most once
class Set {
  ILin elements;

  Set(ILin elements) {
     this.elements = elements;
  }

  // add i to this set
  // unless it is already in there
  Set add(int i) {
    if (this.in(i))
      return this;
    else
      return
        new Set(new Cin(i,this.elements));
  }

  // is i a member of this set?
  boolean in(int i) {
    return
      this.elements.howMany(i) > 0;
  }
}
```

```
// a bag of integers
class Bag {
  ILin elements;

  Bag(ILin elements) {
    this.elements = elements;
  }

  // add i to this bag
  Bag add(int i) {
    return
      new Bag(new Cin(i,this.elements));
  }

  // is i a member of this bag?
  boolean in(int i) {
    return
      this.elements.howMany(i) > 0;
  }

  // how often is i a
  // member of this bag?
  int howMany(int i) {
    return
      this.elements.howMany(i);
  }
}
```

Figure 3: Collections of integers

5. Develop the method *size*, which determines how many elements a *Bag* or a *Set* contain. If a *Bag* contains an integer n times, it contributes n to *size*.

6. Develop the method *rem*, which removes a given integer. If a *Bag* contains an integer more than once, only one of them is removed.

Most object-oriented languages provide libraries like *Set*, and *Bag*. Find the documentation for your local Java implementation and read up on *Set*s. ∎

5.5 Problem

Design the game Pong. Pong consists of a ball that is falling down diagonally from left to right and a paddle at the bottom, controlled by the left and right arrow keys. The goal is for the paddle to hit the ball as it gets near the ground.

We will not yet worry about the details of how the game ends, how we keep the score, or what the ball does once it either hits the bottom or is hit by the paddle.

Here is a list of concerns to start with.

1. Design the class *Ball* to represent the ball in the *World*. Include the method *sameBall* that can be used to test whether one ball is the same as another ball. Include the method that draws the ball on the given canvas.

2. Add the method *moveDown* to the class *Ball* that moves the ball three pixels down and right.

3. Add the method that determines whether the ball is out of bounds, for a given size of the canvas.

4. Design the class Paddle to represent the paddle in the game. Include the method *samePaddle* that can be used to test whether one paddle is the same as another paddle. Include the method that draws the paddle on the given canvas.

5. Add the method movePaddle that consumes a key event (a String) and produces a new paddle moved either left or right by 5 pixels, if left or right key is hit, and in the same place otherwise.

6. Add the method hitBall that consumes a ball and determines whether the paddle hit the given ball.

7. Design the class PaddleWorld that extends World and includes a paddle and a ball. To do so, you must design the methods draw, erase, onKeyEvent and onTick.

8. Design the method makeBall that produces a new ball at a random height on the left side of the canvas.

9. The method draw should draw the paddle and the ball on a black background, the method erase should just draw the background.

10. The onKeyEvent should move the paddle, as specified in the movePaddle method, leaving the ball at the same place.

11. The onTick method should move the ball down until it either is hit by the paddle or goes out of bounds. When that happens a new ball comes into play.

12. Run the world.

■

5.6 Writing Problem

Writing assignments are separate from the rest of the assignment for the week. You should work on this assignment alone, and submit your work individually on the due date for the rest of the homework. The answer should be about two paragraphs long — not to exceed half a page or 300 words.

There is a huge number of programming languages available to programmers today. Investigate why are new languages invented, how are they different from each other, why are some better than other, or more popular. Support your findings with references to at least two sources you have read. ■