Homework 5: Object Oriented (OO) Programming

Due: 1,3,6 on April 9, and the rest of the regular problems on April 23.

In this homework, you will learn about OO programming For all Smalltalk programs, you must run your code with Little Smalltalk. You must also provide evidence that your program is correct (for example, test cases). Hand in a printout of your code and your testing. Some questions just require an answer in English. Be sure to clearly label what problem each function solves with a comment.

Don't use designs or write code that look like C! We'll take points off for poor designs or style.

This is an individual homework. All problems are to be done on your own.

We will pass out some aids and readings about Smalltalk. There are also some books on reserve at the library.

1. This is an introductory reading problem. Find a published article or book, about object-oriented design or programming, from the *Introduction to the Literature* handout. Read it and summarize it in a paragraph for each of the following parts. (Don't make this very long, please. You have a maximum of one page.)

   (a) (10 points) What is the problem(s) addressed by the paper or book? (If you read a survey paper instead of a paper solving a problem, then say that and describe instead the areas covered by the survey.) Do you think it is a significant problem (or, for a survey, that these are significant areas)? Why?

   (b) (20 points) Describe the main points of the argument and the solution(s). (For a survey paper, describe the main techniques surveyed.)

   (c) (10 points) How is this claimed to be an advance on related work? (For a survey paper, this point is moot unless certain ideas are claimed as new with that paper. If there are no such claims, then just say so.) Do you believe these claims? Why?

2. (extra credit, 20 points) Do the same thing as in the above problem for another paper, but this time the paper cannot be a survey paper, and must be about OO programming languages, type systems, or semantics.

3. (10 points) Define (in Little Smalltalk) a class `Point` of 2-dimensional points having the following instance operations:

   - `x:`, which sets the point's x coordinate,
   - `y:`, which sets the point's y coordinate,
   - `x`, which returns the point's x coordinate,
   - `y`, which returns the point's y coordinate,
   - `yxReflect`, which negates both coordinates, For example, a `Point` with value $(4, 2)$ has its value changed to $(-4, -2)$.
   - `yxRotate`, which rotates the point through an angle of 90 degrees in the sense of rotating the y axis to the x axis, hence the name. For example, a Point with value $(4, 2)$ has its value changed to $(2, -4)$),
   - `equalAsPoint:`, which tests to see whether its argument is a point with the same x and y coordinates.

4. (extra credit, 10 points) Implement `Point` in a different way than you did in the above problem. Call the new class `Point2`. (For example, if you used two instance variables before, use an array now, or vice versa). Can a program work with both `Point` and `Point2` objects? Why or why not?

5. (extra credit, 10 points) Specify the class `Point` using the notation of TR91-22.

6. (15 points) Design and implement a class `ThreeDPoint` as a subclass of `Point`. `ThreeDPoint` should have extra operations `z:` and `z`, like `x` and `x:` for `Point`. Also add an operation `equalAsThreeDPoint:` that tests equality of all coordinates, not just the first two.

7. (extra credit, 5 points) Explain what happens if you define instead of `equalAsPoint:` and `equalAsThreeDPoint:` just the operation `=`. (Can anything go wrong?)

8. (extra credit, 10 points) Can you think of a better way to name the operations `equalAsPoint:` and `equalAsThreeDPoint:` so that the naming of such operations is uniform, and so that the Smalltalk `=` can be defined as a special case? Explain your idea.

9. (15 points) Define a class `Rectangle`. The operations should include `upperLeft:` and `lowerRight:` which take `Point`s as arguments, `area` which computes the area of a rectangle, and `=` which compares rectangles for equality. (Two rectangles are equal if they have the sides and location in the plane.) You should add any other operations you think necessary: part of your grade will be based on your choice of operations for this type. Don't design too many operations.

10. (15 points) Design and implement a class `Bag` that follows the pattern of the other collections in the "Overview and Specification" document. Again, part of your grade will be based on your choice of the operations for this type. You should model `Bag` on `Set`. (Hint: you won't have to write much code if you do this right.)

11. (extra credit, 10 points) Can you implement bags that can hold an infinite number of elements (for example, all the odd integers) in Smalltalk? If so, do it!

12. (extra credit, 10–30 points depending) Pick some thing or concept that is specific to some application area (e.g., databases or train robbery) and design a few Smalltalk classes that model these things or concepts.

13. (extra credit, 15 points) Dynamic scoping tends to make type checking and verification difficult. What is dynamically scoped in Smalltalk and how does it make type checking and verification difficult?

14. (40 points) Object-oriented programming techniques are touted as being good for simulation. In this problem you will investigate an iterated prisoner's dilemma, using simulation techniques and Smalltalk.

    The prisoner's dilemma is a general model of social interactions. The idea is that people are often faced with situations where they may either "cooperate" or "defect". As the standard example, consider two criminals, Bonnie and Clyde, who have been arrested on a minor charge but are suspected of committing a major crime. The police make the following offer to each: "if you confess and implicate your partner, then you will go free and your partner will get 5 years in jail, but if you do not confess, you will have to serve one year in jail on the minor charge. Oh by the way, if you both confess, you will each serve 3 years in jail." In this case cooperation (among the prisoners) means keeping silent, defection means implicating the partner. The police offer can be represented by the following payoff matrix:

You\Other	cooperate	defect
cooperate	-1	-5
defect	0	-3

    where the entry 0 means that if you defects and your accomplice cooperates (with you), then you get a jail term of 0 years. The entry of -5 means that, if you cooperate (with your partner) but your partner defects, then you get 5 years in jail. Each prisoner wants to maximize his or her payoff (i.e., each wants to get the fewest years in jail).

    The iterated prisoner's dilemma is like the prisoner's dilemma, only repeated over many encounters. Imagine Bonnie and Clyde both cooperate, and after a year in jail, they get together again, rob more banks, and then wind up in the same situation. More to the point, imagine a world of agents who are continually interacting. For example, consider graduate students and their use of disk space. One

may cooperate by deleting unused files, but this takes some time and effort. Hence there is a cost to cooperation. If others use disk space without deleting unneeded files (the defectors), you are just subsidizing their lethargy with your time! However, if everyone cooperates, then there will usually be plenty of disk space for all. Similar examples can be found in many social interactions, from driving a car to trade relations among nations.

Your task is write a simulation of the iterated prisoner's dilemma in Smalltalk, and test at least three (3) different strategies for the behavior of prisoners to see which is best. A strategy is an algorithm for deciding whether to cooperate or defect, based on internal state such as a history of how the other agent has cooperated or defected in previous rounds or a random number generator. You should submit an English explanation of your program and the strategy you found to be the best, along with your experimental results.

Your program should be based on the little Smalltalk code found in files in the directory

```
~leavens/WWW/ComS541/homework/prisoners
```

Look in the `README` file in that directory for further directions. In brief, there are some methods you have to add, and some classes you have to design and add.