Homework 7: Object Oriented Programming

This homework is due Wednesday, May 1.

1. (extra credit only) Read chapter 12 of *Scheme and the Art of Programming* (which supplements Normark's paper) and do (in Scheme) one or more exercises 12.5, 12.6, 12.7, and 12.12. These readings and exercises might help you ease into object-oriented programming before you tackle the Smalltalk syntax.

2. (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,
   - `reflect`, which negates both coordinates,
   - `rotate`, which rotates the point through an angle of 90 degrees (so that that point $(4, 2)$ becomes $(2, -4)$),
   - `rho`, which gives the point's distance from the origin,
   - `theta`, which give the point's angle, counter clockwise from the positive x axis, making a `rho` and `theta` the point's polar coordinates,
   - `equal:`, which tests to see whether its argument is a point with the same coordinates.

   What does `p theta` return if `p` is the origin?

3. (extra credit only) Implement `Point` in a different way than you did in the above problem. Call the new class `Point2`. (For example, if you used rectangular coordinates as your representation before, use polar now, and vice versa). Can your program work with both `Point` and `Point2` objects? Why or why not?

4. (10 points) Design and implement a class `Triple` as a subclass of `Point`. Triple should have extra operations `z:`, `z`, and `phi`, like `x`, `x:` and `theta`, except that `phi` gives the angle from the positive z axis. Do you have to redefine `rho`?

5. (20 points) Define a class `Rectangle`. The operations should include `upperLeft:` and `lowerRight:` which take Points as arguments, `area` which computes the area of a rectangle, and `equal` which compares rectangles for equality. 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.

6. (20 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`.

7. (extra credit only) Can you implement infinite bags in Smalltalk. If so, do it!

8. (extra credit only) 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?

9. (extra credit only) 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:

Bonnie\Clyde	cooperate	defect
cooperate	(-1,-1)	(-5,0)
defect	(0,-5)	(-3,-3)

   where the entry (0,-5) means that if Bonnie defects and Clyde cooperates, then Bonnie gets a jail term of 0 years, while Clyde gets 5 years in jail (-5). 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 explaination of your program and the strategy you found to be the best, along with your experimental results.