Homework 2: Introductory Smalltalk Programminng

Due:

Since the purpose of this homework is to ensure basic skills in Smalltalk programming, this is an individual homework. That is, for this homework, you are to do the work on your own, not in groups.

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. (You can run Smalltalk from within an emacs shell to get test output captured, or you can use the Smalltalk printing primitives to print test output to a file.)

Don't write code that looks like C! We'll take points off for poor style.

1. (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.
2. (extra credit, 10 points for the programming and semantics tracks) 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?
3. (extra credit, 10 points for applications—software engineering) Specify the class `Point` using the notation of TR91-22.
4. (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.
5. (extra credit, 5 points, programming and semantics) Explain what happens if you define instead of `equalAsPoint:` and `equalAsThreeDPoint:` just the operation `=`. (Can anything go wrong?)
6. (extra credit, 10 points, programming and semantics) 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.
7. (15 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 `=` 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.

8. (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.)

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

10. (extra credit, 10–30 points depending, applications track) 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.

11. (extra credit 15 points, semantics and applications—software engineering) 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?