Homework 1: Introduction and Introductory Smalltalk Programming

Due: problem 1 on Sept 16, 1997, problem 2 on Sept. 18, 1997, and the rest on Sept. 30, 1997.

The purpose of this homework is to ensure basic skills in Smalltalk programming, and to help you explore the literature a bit. 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 Smalltalk Express (i.e., Smalltalk/V). Hand in a printout of the file out of your code and your testing. For classes you write, you can file out the entire class by selecting the Classes menu in the class browser, then choosing "File Out ...". For code you add to an existing class, just file out the new or changed methods (from the Methods menu in the class browser, choose "File Out ..."). For testing, test your programs in a workspace; if your program has textual output, use "Show It" from the Smalltalk menu and then save the workspace in a file (using the File menu, and choosing "Save As ...").

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

Readings: Chapters 2-3, 5, 6-7, and 9 of *Discovering Smalltalk* by Wilf LaLonde (Benjamin-Cummings, 1994). For background on OO ideas and OO design, see Chapters 1 and 2 of the *Design Patterns* book by Gamma, *et al.* (See also the *Introduction to the Literature* handout for more readings.)

1. (15 points) In "Discovering Smalltalk", section 3.9, the classes Point and Pen are described.

   Add a method `ngon:withSide:` to the class `Pen`. This method should take two integer arguments, `n`, which is the number of sides and thus strictly greater than 2, and `len`, which is the number of pixels on each side and thus positive. It draws an `n`-gon each of whose sides is `len` pixels, that is to the right and below the pen's current point. For example

   ```
   | myPen |
   myPen := (GraphPane openWindow: 'Testing') pen.
   myPen ngon: 3 withSide: 50.
   ```

   should draw an equilateral triangle at the current point.

2. (30 points) Make a subclass `TestRandom` of the class `Random` in the following file.

   `http://www.cs.iastate.edu/~leavens/ComS541/homework/lottery/random.cls`

   The subclass should support an instance method `visualize` that graphs (using `GraphPanes` and `Pens`) the function $f(i) =$ "the result of the ith call to the `next` method" (for some fixed instance of `TestRandom`). That is, draw the x and y axes, label them, and draw a dot at the graph coordinate $(i, f(i))$. Graph points with $i = 1$ to $i = 400$.

3. (10 points extra credit) From the picture you draw, can you tell how good the random number generator is? If it's bad can you fix it?

4. (30 points extra credit) Make a class to draw graphs of functions. Demonstrate its use on various problems.

5. (40 points) In this problem you will define a class `BinaryRelation` which models binary relations. The abstract value of a `BinaryRelation` object is a set of pairs of objects. For example, a binary relation with abstract value $\{(1,2),(1,3),(5,4)\}$ relates 1 to 2, 1 to 3, and 5 to 4.

You are to define the class `BinaryRelation` as a subclass of `Collection`. (It can be a subclass of some subclass of `Collection`, but must inherit from `Collection` to receive credit.)

The class methods should include **`new`**, which returns an empty binary relation. However, you don't need to override **`new`** if what you inherit works correctly.

Instance methods should include the following. These are specified using structured English; I'll explain the notation in class, but it's also explained in ISU CS TR #91-22a.

relate:to: *self: BinaryRelation, key: Object, value: Object*
Modifies at most: *self.*
Effect: make **post**(*self*) be the union of **pre**(*self*) with $\{(key, value)\}$.

relates:to: *self: BinaryRelation, key: Object, value: Object* $\rightarrow$ *b: Boolean*
Ensures: *b* is true just when *self* contains the pair $(key, value)$.

do: *self: BinaryRelation, aBlock: Block*
Requires: *aBlock* takes one argument, which has type `Pair`, and does not modify *self.*
Modifies at most: the objects modified by the execution of *aBlock.*
Effect: evaluate *aBlock* for each `Pair` object that represents a pair, $(key, value)$, such that *self* relates *key* to *value*.

at: *self: BinaryRelation, key: Object* $\rightarrow$ *s: Set*
Ensures: *s* is a new `Set` object whose abstract value is the set of objects to which *key* is related in *self*; that is, $\mathbf{post}(s) = \{v \mid (key, v) \in self\}$.

Be sure that your methods for **`do:`** and **`at:`** do not expose the representation of a `BinaryRelation` object. Also be sure that printing works as desired for binary relations, so that you can do adequate testing; you may have to redefine **`printOn:`** to make this work as needed.

6. (30 points) Add a subclass `ReflexiveRelation` to `BinaryRelation`. Instances of `ReflexiveRelation` always relate every object to itself by the **`relates:to:`** method. That is, the **`relates:to:`** method satisfies the following specification.

relates:to: *self: ReflexiveRel, key: Object, value: Object* $\rightarrow$ *b: Boolean*
Ensures: *b* is true just when either $key = value$ or *self* contains the pair $(key, value)$.

According to the specification above, if $key = value$, then the *key* does not have to be in the domain of the relation.

Note, you may need to (or find it convenient to) override a few other methods to make `ReflexiveRelation` work as desired.

7. This problem discusses binary methods.

   (a) (5 points) What is the difference between the `==` and the `=` methods in Smalltalk?

   (b) (5 points) Implement an `=` method for the class `BinaryRelation`. This method should follow the general pattern for `=` methods. If you don't need to add any code to your class to have `=` work correctly say so instead of writing the code.

   (c) (5 points) Does your `=` method for binary relations work equally well when comparing arguments that may be reflexive relations? Why or why not?

   (d) (5 points) Suppose the type **`Graph`** were added as a subclass of **`BinaryRelation`**. Let's imagine that a **`Graph`**, besides storing the points of a relation, also stores a **`GraphPane`** window. Does the `=` method inherited from **`BinaryRealtion`** do the right thing for **`Graph`** arguments? Does it do the right thing for mixed **`Graph`** and **`BinaryRelation`** arguments?

8. (100 points) Write a Smalltalk program to do something interesting worth 100 points. For example, you could do something related to something in your creative component, or write a simple game or animation system. It's best if what you do reuses some of the built in classes in Smalltalk/V to do something interesting, by adding new classes to the system.

Don't change any existing code in the system, unless you need to fix a bug. Instead add subclasses that override methods you want to change. Besides being good OO coding practice, this prevents you from crashing the system by accidently modifying some crucial part.

9. Read a published research article that is related to OO programming languages (such as Smalltalk) from the "Introduction to the Literature" handout. By a published research article, I mean an article that is not in a trade journal (e.g., it has references at the end) and that is not on the web, but in the library. (It's okay to get a copy of a published article from the web, however.) An article from the OOPSLA or ECOOP conferences is fine. An article primarily about OO programming is not acceptable; the article should be about programming *languages*. I am happy to suggest articles for you, or evaluate your choices for suitability if you are in doubt.

   Write a short (1 or 2 page) review of the article, stating:

   - (10 points) what the problem was that the article was claiming to solve, (or if the article is a survey article, then the scope of the survey),
   - (20 points) the main points made in the article and what you learned from it,
   -
   - (20 points) what contribution it make vs. any related work mentioned in the article (or if the article was a survey article, your evaluation of its completeness and clarity of thought and expression, and a statement of what you learned from it).