Spring, 1996 Name: ______________________________

Com S 541 — Programming Languages 1

Test on Smalltalk and OOP

Special Directions for this Test

This test has 5 questions and pages numbered 1 through 7.
This test is open book and notes.
Before you begin, please take a moment to look over the entire test so that you can budget your time.
When you write programs on this test, you may use anything documented in the *Overview and Specification of the built-in types in Little Smalltalk*, TR #91-22a. You are encouraged to define methods and classes not specifically asked for if they are useful to your programming.
Make it clear by indentation where methods start and stop.

1. (20 points) Briefly explain the design for your group's test system (from problem 7 of homework 2).

2. (a) (5 points) Name a language that we have studied that uses single dispatch as its semantics for message sends.

   (b) (5 points) Name a language that we have studied that uses multiple dispatch as its semantics for message sends.

   (c) (10 points) What is the difference between the message passing semantics for single dispatch and multiple dispatch?

   (d) (10 points) Is there any practical advantage in a language that has multiple dispatch? If not, say why not; if so, briefly describe why.

3. In this problem you will define a class `Pair`. The abstract value of a `Pair` object is a tuple, (f, s), of two objects, f and s. The object f is called the "first element" of the pair, and s is called the "second element".

   (a) (5 points) What would you type to the Little Smalltalk system prompt to define the class `Pair`?

   (b) (15 points) Using the convention that a completely blank line separates methods, write code for the methods specified below. (These methods are specified in the notation of TR #91-22.)

first:second: *self: Pair, f: Object, s: Object*
   Modifies at most: *self.*
   Effect: make **post**(*self*) be (f, s).
first *self: Pair* $\rightarrow$ *f: Object*
   Ensures: *f* is the first element of *self.*
second *self: Pair* $\rightarrow$ *s: Object*
   Ensures: *s* is the second element of *self.*
printString *self: Pair* $\rightarrow$ *s: String*
   Ensures: *s* is a representation of *self* suitable for printing.

4. In this problem you will define a class `BinaryRel` which models binary relations. The abstract value of a `BinaryRel` 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.

   (a) (5 points) What would you type to the Little Smalltalk system prompt to define the class `BinaryRel` as a subclass of `Collection`?

   (b) (5 points) Define the method `new` for this class, so that a `BinaryRel` object starts out an abstract value that is an empty set of pairs. (If you don't need to write a `new` method, but can just inherit it, write a comment that says "just inherit it".)

   (c) (20 points) Using the convention that a completely blank line separates methods, write code for the methods specified below. (These methods are specified in the notation of TR #91-22a.) Write the minimum amount of code needed, given your representation.

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

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

   do: *self: BinaryRel, 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: BinaryRel, key: Object* $\to$ *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 `BinaryRel` object. (You're supposed to know what "exposing the representation" means.)

   There is more space for your answer on the next page.

5. In this problem you will define a class that implements a type `ReflexiveRel`. The abstract value of a `ReflexiveRel` object is again a set of pairs; however, note that for every pair (*key*, *value*) in the abstract value of a `ReflexiveRel`, (*key*, *key*) and (*value*, *value*) are also in the relation.

   The type `ReflexiveRel` is intended to be a behavioral subtype of `BinaryRel`. (You are supposed to know what "behavioral subtype" means.) In addition, an object is always related 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*).

   (a) (5 points) What would you type to the Little Smalltalk system prompt to define the class `ReflexiveRel`? You should do this in a way that makes it very easy to ensure that `ReflexiveRel` is a behavioral subtype of `BinaryRel`.

   (b) (5 points) Define the method `new` for this class so that the abstract value of a new `ReflexiveRel` object starts with an empty set of pairs. Write the minimum amount of code needed, given your representation. (If you don't need to write a `new` method, but can just inherit it, write a comment that says "just inherit it".)

   (c) (15 points) Using the convention that a completely blank line separates methods, write any code needed to define the methods needed to satisfy the specification of the `relates:to:` method given above, and to make `ReflexiveRel` a behavioral subtype of `BinaryRel`. Write the minimum amount of code needed, given your representation.

   There is more space for your answer on the next page.