Fall, 1997 Name: ________________________________

Com S 541 — Programming Languages 1

Test on Smalltalk and OO Languages

Special Directions for this Test

This test has 7 questions and pages numbered 1 through 6.
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 Smalltalk Express (i.e., Smalltalk V). You are encouraged to define methods and classes not specifically asked for if they are useful to your programming.
Use the "file out" syntax to declare classes, and to distinguish class methods from instance methods. You can use the exclamation point (`!`) to show where methods end, but also make these clear by using blank lines between methods and proper indentation.

1. (10 points) Write an implementation of the `=` method for the class `Set`. Use the `includes:` method of `Set` and any other methods that `Set`s support that you need. (Also, remember that you may write additional helping methods if needed.) The following are examples, where `==>` separates Smalltalk input from what would be printed by the "Show it" command.

```
Set new = Set new
  ==> true
Set new = (Set new add: 1; yourself)
  ==> false
(Set new add: 1; yourself) = (Set new add: 1; yourself)
  ==> true
(Set new add: 2; add: 1; yourself) = (Set new add: 1; add: 2; yourself)
  ==> true
```

2. (10 points) Recall that in Smalltalk Express, `Dictionary` is a subclass of `Set`. The class `Dictionary` contains instances of the class `Association`. Each association contains a pair of objects, a key and a value. Assume that `Dictionary` inherits the `=` and `includes:` methods of `Set` (unchanged). Briefly state (a) what the behavior of `Set`'s `includes:` method must be for `Dictionary`'s inherited `=` method to work properly, and (b) why this makes it work properly. (If you like, for part (a) you can give the code for `Set`'s `includes:` method that is needed.)

The following two problems are related. You may want to read them both before solving either one.

3. (30 points) In this problem you will implement, in Smalltalk, an abstract class **Automata**, which models finite state machines. Instances of concrete subclasses of Automata have a finite number of states, some of which are designated as *accepting* states. When an automata "reads" a character from its input, it makes a transition from its current state to another (possibly different) state. When an automata is in an accepting state, it is considered to have accepted the input it has "read" so far. When an automata is not in an accepting state, it is considered to have rejected the input it has "read" so far.

   The class `Automata` itself relies on the abstract methods `accepted`, which tells whether the automata has accepted the input it has read so far, and `read:` which handles the reading of a character. The specification of the other methods in the class are as follows.

   rejected *self: Automata* $\rightarrow$ *result: Boolean*
   Ensures: *result* is true just when *self* has not accepted the input it has "read" so far.

   readAll: *self: Automata, aString: String*
   Modifies at most: *self*
   Effect: **post**(*self*) is in the state obtained by "reading" the characters of *aString* in order from left to right.

   accepts: *self: Automata, aString: String* $\rightarrow$ *result: Boolean*
   Modifies at most: *self*
   Ensures: *result* is true just when **post**(*self*) is in an accepting state after having "read" then characters of *aString* in order from left to right.

   Write your implementation by continuing the following.

```
Object subclass: #Automata instanceVariableNames: ''
   classVariableNames: '' poolDictionaries: '' !
!Automata class methods ! !

!Automata methods !
accepted
   "answer true if the input has been accepted."
   ^self implementedBySubclass!

read: aChar
   "read a character and move to the next state"
   self implementedBySubclass!
```

4. (10 points) Write a subclass of `Automata`, called `Ones`, that accepts its input if it consists of 0 or more occurrences of the character `1`. For example, your class should have the following behavior. (Again, in these examples, `==>` separates Smalltalk input from what would be printed by the "Show it" command.)

```
Ones new accepted                  ==> true
(Ones new read: $1) accepted       ==> true
(Ones new read: $2) accepted       ==> false
(Ones new readAll: '111') accepted ==> true
Ones new accepts: '111'            ==> true
Ones new accepts: '111e11'         ==> false
Ones new accepts: '111e11'         ==> false
```

5. (15 points) Unlike C++, both Java and Smalltalk do not allow pointer arithmetic. Also unlike C++, both Java and Smalltalk have automatic garbage collection. Briefly explain, what, if anything, is the relation between the lack of pointer arithmetic and the ability of a language to support automatic garbage collection.

6. (15 points) Briefly explain the difference between overloading and overriding of methods in Java. Give an example.

7. (35 points) List the important differences between C++ and Smalltalk. Focus on significant semantic differences, and ignore syntactic issues. (Do not explain why these differences arose, just list them.)