Homework 2: Object-Oriented Programming in Smalltalk

Due:

This is a group homework.

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 (print the file produced by `fileOut`) and your testing. (You can use the `script` program on Unix to make a transcript, or 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.)

1. (100 points; semantics, programming) Complete the coding of an interpreter for the language INST, whose grammar is given in Figure 1. You'll fill in the blanks of the interpreter which is in the following file: `~leavens/public/st-interpreter/INST.st`. The places you need to edit are marked by: `#youWereToEditThis print`.

   The code for this interpreter does not do parsing; instead it takes as input an abstract syntax tree. One of the things you will add is type checking, which will include checking of assignment commands. We'll explain the ideas and meaning of the various constructs in this language, and watch the newsgroup (isu.coms.541) for hints and clarifications.

P ∈ Program	TD ∈ Type-Declaration
CD ∈ Class-Declaration	Op ∈ Method
M ∈ Main-Proc	VD ∈ Variable-Declaration
F ∈ Formal	B ∈ Body
S ∈ Statement	E ∈ Expression
I ∈ Identifier	N ∈ Number
T ∈ Type	C ∈ Class

P ::= TD* CD* Op* M
TD ::= **type** T
CD ::= **class** C **implements** T **instance variables** VD **end**
VD ::= | I : T ; VD

Op ::= **method** I (F*) : T **is** B
F ::= | I @ C : T
B ::= S **return** E

M ::= **observe** VD_1 S_1 **by** VD_2 S_2

S ::= E | **if** E_1 **then** S_1 **else** S_2 **fi** | S_1 ; S_2 | I_1 := E | I_1 . I_2 := E | **skip**
E ::= N | I | I_1.I_2 | I (E*) | **new** C | **bot**

T ::= I
C ::= I

Figure 1: Abstract Syntax of INST.

You can do the following extra credit problems individually or as a group. If you do them as a group, be sure only to list members who actively participated.

2. (75 points; extra credit — semantics, programming) Extend your interpreter for INST to include subtypes. Changes in the abstract syntax are given below.

$$\mathrm{TD} ::= \texttt{type}\ \mathrm{T} \mid \texttt{type}\ \mathrm{T}_1\ \texttt{subtype of}\ \mathrm{T}_2{}^*$$

   Hand in extensions to your code with your test runs.

3. (50 points; extra credit — programming) Add a parser to the interpreter.

4. (75 points; extra credit — semantics, programming) Add a class **`any`** to the language. Use of **`any`** in a method formal makes the method work for any class, but the method is overridden by another method which is specialized on a more specific class. One way to think of this is that **`any`** is a superclass of all other classes, and more specific methods override less specific ones. For example, a method with specializers $\langle \texttt{any}, \texttt{any} \rangle$ is less specific than one with the same name and specializers $\langle \texttt{Account}, \texttt{Account} \rangle$.