Assignment 3

CSE 3341

Due Friday, November 2, by 2:45 pm

1. (10 pts) Consider the following Java code:

```
abstract class A {
  private A h;
  public A(A x) { this.h = x; if (x != null) x.set(this); } // constructor
  public A get() { return this.h; }
  public void set(A y) { this.h = y; }
  public abstract A p();
  public static void q(A x) { A y; y = x.get(); A z; z = y.p(); }
  public void r() { A x; x = this.get(); q(x); }
}
class B extends A {
  public B(A x) { super(x); } // constructor: calls A(A x)
  public A p() { return this; }
}
class C extends B {
  public C(A x) { super(x); } // constructor; calls B(A x)
  public A get() { return this; }
  public A p() { A x; x = this.get(); return x; }
}
```

Consider the following `main` method that uses the three classes from above:

```
class Main {
  public static void main(String[] args) {
    A x, y;
    // time moment 1
    x = new C(null); // "null" is the reference value "does not reference anything"
    y = new B(x);
    // time moment 2
    y.r();
    // time moment 3
  }
}
```

Part 1. (5 pts) List the sequence of *call events* and *return events* from time moment 1 to time moment 2, e.g.,

- Constructor `C(A x)` in class C is invoked (call event)
- ?
- The call to constructor `C(A x)` returns (return event)
- Constructor `B(A x)` in class B is invoked (call event)
- ?
- The call to `B(A x)` returns (return event)

Fill in the blanks with the rest of the events. Describe precisely *which method/constructor* in *which class* is called/returns at each step. Do not omit any events — *all* calls and returns that happen during this execution should be described in the sequence.

In addition, show the state of the stack and the heap *at time moment 2*. Specifically, show the activation records that are on the stack, in the order in which they appear. Show clearly

(1) each activation record and the method it corresponds to; (2) the local variables and formal parameters in each of these records; (3) the values stored in the variables and parameters on the stack (show "?" for uninitialized variables). For the state of the heap, show all run-time objects that are alive at moment 2, together with the values of their fields. If a stack/heap location has a reference value, describe clearly which entity is being referenced.

Part 2. (5 pts) List the sequence of call events and return events from time moment 2 to time moment 3, using the format described in Part 1.

2. (10 pts) The purpose of this problem is to see the use of polymorphism and dynamic dispatch in the implementation of interpreters. Consider a Core-like language with the following syntax:

   ⟨*prog*⟩ ::= `program` ⟨*decl_seq*⟩ `begin` ⟨*stmt*⟩ `end`
   ⟨*stmt*⟩ ::= ⟨*two_stmts*⟩ | ⟨*loop*⟩ | ⟨*if*⟩ | ⟨*assign*⟩
   ⟨*two_stmts*⟩ ::= ⟨*stmt*⟩ ⟨*stmt*⟩
   ⟨*loop*⟩::= `while` ⟨*bool_cond*⟩ `do` ⟨*stmt*⟩ `endwhile ;`
   ⟨*if*⟩ ::= `if` ⟨*bool_cond*⟩ `then` ⟨*stmt*⟩ `endif ;` | `if` ⟨*bool_cond*⟩ `then` ⟨*stmt*⟩ `else` ⟨*stmt*⟩ `endif ;`

   The productions for ⟨*decl_seq*⟩, ⟨*assign*⟩, and ⟨*bool_cond*⟩ are omitted; you can assume Core-like syntax for those (but you do not need to know what they look like in order to solve this problem). Note that there is no ⟨*stmt_seq*⟩; instead, ⟨*two_stmts*⟩ is one alternative for ⟨*stmt*⟩ and this gives us the option of using more than one statement in place of a single statement. Do not worry about the fact that the grammar is ambiguous.

   Suppose we wanted to implement a data structure for the parse tree of this language, using an object-oriented language such as C++ or Java. Each node in the parse trees will be represented by a separate object. An edge from a parent node n to a child node m can be encoded by a pointer/reference field in the object for n, pointing to the object for m.

   In this parse tree, each node corresponding to ⟨*stmt*⟩ has a single child. Suppose we wanted to reduce the memory usage of our interpreter, and therefore we decided to "skip" ⟨*stmt*⟩ nodes from the parse tree, and "jump" directly to the single child node. For example, for a statement `if [x<5] then x:=5 endif;`, the parse tree node for the `if` statement will have as a second child the parse tree node for the assignment statement `x:=5`, without a ⟨*stmt*⟩ node in between.

   Write the code for a set of C++ or Java classes (your preference) that will be used for this representation. That is, write classes `Prog`, `TwoStmts`, `Loop`, and `If`, as well as any additional classes you deem necessary. Do not worry about classes `DeclSeq`, `Assign`, and `BoolCond`: assume that they can be defined appropriately by someone else. In each class, *just show the fields* that are necessary to implement the data structure; do not show any methods. Your code has to be *complete* and *legal*: it should be compilable using a standard C++ or Java compiler. The code should also be reasonably commented and readable.

3. (10pts) For each class from above we want to implement a method `execute()` that can be used to execute the corresponding parse tree node, similarly to what was done in Project 1. Show the code of all methods `execute()` for the classes you defined earlier. Make sure that your implementation uses the virtual call mechanism (a.k.a. "method lookup" or "virtual dispatch") of the underlying object-oriented language to call the correct `execute()` method based on the type of the statement. Do not use explicit multi-way selections (e.g., do not use `switch` statements or sequences of `if-then-else-if` statements) to achieve this. Assume that `execute()` methods for `DeclSeq` and `Assign`, and `evaluate()` for `BoolCond`, have been defined appropriately by someone else. For the methods you show, the code should be *complete* and *legal*: it should compile using a standard C++ or Java compiler. The code should also be reasonably commented and readable.

(Please turn over.)

- Assignments should be done independently. General high-level discussion of assignments with others in the class is allowed, but **all of the actual work** should be your own. Assignments that show excessive similarities will be taken as evidence of cheating and dealt with accordingly.
- Assignments should be turned in **by the end of class** on the due day. Late assignments turned in by the end of the next class will be graded with 30% reduction. Assignments turned in later than that will not be accepted.
- Make the assignments readable and understandable. They should be handed in on regular paper, legibly written or typed. If you have more than one sheet, **staple the sheets together**. If the grader has trouble reading or understanding what you have done, points will be deducted even if it can finally be determined that you have the correct answer.
- Your solutions have to be **precise and detailed**: you have to work out **all** details that are necessary to solve the problem using the approaches discussed in class. You also have to write your solutions in a way that convinces the grader that you understand all these details. Be careful, precise, and thorough.