CSE341—Final Exam

Instructor: Greg J. Badros — Summer 1999

20 August 1999

This final exam is open book, open notes. You may use any of your class notes and your book. You may *not* use any of your neighbors' class notes or books. You may *not* use a laptop, a calculator, your neighbor, a cell phone, a pager, or any other computing or communication device. **Write your name on each page. You must answer all five questions—each question specifies how many points it is worth, and the five questions sum to 100 total points**. Partial credit will be given, so please write down your thoughts even when you think you might not have a completely correct answer. **Be sure to clearly indicate final answers.** Please answer questions by writing directly on the exam. We cannot award points to answers that we cannot read, so please **write clearly**. Enough space is provided on the test pages for correct answers; however if you need more space, please use the back sides of the exam's pages. You have 60 minutes. **Good luck!**

Name:

Student ID:

1

(**20 total pts**) Suppose you are trying to understand another programmer's Smalltalk code. Unfortunately, that programmer did not document his code very well, and used poor variable names so much of what he wrote is hard to understand. Consider the below portion of code:

```
Object subclass: #MyClass
  instanceVariableNames: 'anObject'
  classVariableNames: ''
  poolDictionaries: ''

"MyClass Instance method"
doIt: a
a whileTrue: [ anObject takeStep ]
```

If you want to use `MyClass`'s `doIt:` method, what class of object should you pass as an argument? (**5 pts**)

Any object that can understand the `whileTrue:` message. The most straightforward possibility is an object of class `Context` (i.e., a block). Additionally, the block should evaluate to an instance of class `Boolean` if we assume that `Context`'s `whileTrue:` method will be executed. □

Write a sample invocation (i.e., message-send) of `doIt:`. Include the creation of the receiver object (you may assume `MyClass` has a "`new`" class method) and an explicit argument for the message. (**5 pts**)

```
| i | i := 0.  MyClass new doIt: [ i := i + 1. i < 3 ]
```

□

...question continues onto next page...

...question continued from prior page...

How could we use the Smalltalk programming environment to help us figure out what kind of object the instance field `anObject` must refer to? (**5 pts**)

One could ask the environment to search for "Implementors Of" the `takeStep` method. □

Suppose instead that the program is written in Java. Would this make answering the previous questions easier or more difficult? Why? (**5 pts**)

Easier. Java is statically typed, and the types provide excellent documentation for what classes of objects are acceptable as arguments and can be assigned to instance fields. □

2

(**30 total pts**) Mark the compilation errors in the Java code below by putting "//" to the left of each line with an error. To the right of that line, describe what the error is. If you comment-out an error, be sure to consider the effect on the rest of the source code. (There are no syntax errors.) (**15 pts**)

```
class Car {
  public Car() {}

  public static void recall()
  { System.out.println("Recall all cars"); }

  public void drive()
  { drive("Car driving"); }

  public void drive(String s)
  { System.out.println(s); } }

// Honda is a brand of Car
class Honda extends Car {
  public Honda() {}

  public static void recall()
  { System.out.println("Recall all Hondas"); }

  public void drive(String s)
  { System.out.println("Honda " + s); }
}

public class Example {
  public static final void main(String[] args) {
    Car a = new Car();
    Car b = new Honda();
/////////////1    Honda c = new Car();
    Honda d = new Honda();

    Car.recall();
    Honda.recall();
    a.recall();
    b.recall();
/////////////2    c.recall();
    d.recall();
    System.out.println("-----");
/////////////3    Honda.drive("vroom! vroom!");
    a.drive();
    a.drive("Go");
    b.drive();
```

```
    b.drive("VROOM!");
/////////////4    c.drive();
/////////////5    c.drive("cruising");
    d.drive();
    d.drive("moving");
  }
}
```

Show the output of running the above program (i.e., the output when executing the `main` method of class `Example`) with your changes—that is, ignore the lines that you commented out above and on the previous page. (**15 pts**)

There were two primary problems that the Java compiler would catch:

- A `Car` instance cannot be assigned to a reference of type `Honda`. This one mistake requires comments numbered 1, 2, 4, and 5, above.
- The instance message `recall()` cannot be sent to the class `Honda`. This mistake requires comment number 3, above.

See comments added to code above.

A couple of reminders for explaining the below output:

- Class (i.e., static) methods may be invoked on instance objects; the binding to invoked method is done **statically** using the **type** of the instance object (not its **class**)
- Static methods may also be invoked on class objects (e.g., `Car.recall()`).
- Notice that class `Honda` inherits the method `drive()` from class `Car`, but it overrides the method `drive(String)` from class `Car`. The invocation `b.drive()` is thus dispatched to `Car.drive()` which then invokes `Honda.drive(String)`.

Output:

```
Recall all cars
Recall all Hondas
Recall all cars
Recall all cars
Recall all Hondas
-----
Car driving
Go
Honda Car driving
Honda VROOM!
Honda Car driving
Honda moving
```

□

3

(**15 total pts**) Consider the following Java functions:

```
public class CircleMath {
  /** Given a radius, compute and return the circumference
      and the area of a circle with that radius */
  static void ComputeCircumferenceAndArea(double radius,
                                          double circumference,
                                          double area)
  {
    // uses PI constant from java.lang.Math.PI
    circumference = 2 * Math.PI * radius;
    area = Math.PI * radius * radius;
  }

  public static void main(String[] args) {
    double c = 0;
    double a = 0;
    CircleMath.ComputeCircumferenceAndArea(5,c,a);
    System.out.println("Circumference = " + c);
    System.out.println("Area = " + a);
  }
}
```

What does the above code output? (**5 pts**)

Circumference = 0.0
Area = 0.0
□

Do you think this is what the programmer wanted as output? If yes, describe what feature of the Java programming language makes this work; if not, explain what is going wrong and suggest a way to fix it. (**10 pts**)

No, it's likely not what the programmer wanted or expected. The problem the programmer overlooked was that Java passes primitive types by value and so is unable to return newly-computed values through those same parameters.

Several possibilities exist for fixing the code. The easiest is to just write two functions: one for returning the circumference, another for returning the area, and have each of the functions return their single computed value. Another possibility is to return an array of length two with the two computed values inside the array (wrapped in instances of `Double`). □

4

(**20 total pts**) Consider the Smalltalk code:

```
Object subclass: #Counter
  instanceVariableNames: 'incrementer'
  classVariableNames: ''
  poolDictionaries: ''

"Counter class method"
new
  ^ super new initialize.

"Counter instance methods"
initialize
  | i |
  i := 0.
  incrementer := [ :j | i := i + j. i ].

increment: amount
  |  i   j  |
  i := 100.
  j := 5.
  ^ incrementer value: amount.
```

Given the above, what does the expression

```
Counter new increment: 2
```

answer (i.e., evaluate to) and why? (**15 pts**)

2. Because Smalltalk closures are lexically scoped, the block in `initialize` that's bound to `incrementer` refers to the `i` variable that is local to `initialize`. The `i` and `j` in `increment:` are completely unused. □

Given the same Smalltalk code, what does the expression

```
Counter new increment: 1; increment: 2
```

answer (i.e., evaluate to) and why? (**5 pts**)

3. For the same reason as above. Also recognize that `i`'s value persists after the first message send of `increment:`. The local variable `i` defined in the initialize method is stored by the block (i.e., the `Context` instance) that is assigned to the `incrementer` instance field of the `Counter` object. □

5

(**15 total pts**) The Perl programming language gives the programmer a great deal of freedom when choosing how to express a computation. Why might having "More Than One Way To Do It" be undesirable? (**15 pts**)

It can make code harder to read and understand by programmers other than the one who wrote the code. If there is only one way to write something, then programmer reading the code will be able to understand any computation that she could have written. □

End of test—be sure to check your answers if time remains.