CSE 341 - Programming Languages
Final Exam - Autumn 2006

Your Name:

136 points total

Open book and notes. No laptop computers, PDAs, or similar devices. (Calculators are OK, although you won't need one.) Please answer the problems on the exam paper — if you need extra space use the back of a page. The first two questions use the following hierarchy of Java classes for simple geometric shapes. (This is the same hierarchy you used for Homework 6.)

```
public interface GeometricShape {
    public void describe();
}


public interface TwoDShape extends GeometricShape {
    public double area();
}


public interface ThreeDShape extends GeometricShape {
    public double volume();
}


public class Circle implements TwoDShape  {
    private double radius;

    public Circle (double radius) {
        this.radius = radius;
    }

    public double area() {
        return Math.PI*radius*radius;
    }

    public double circumference() {
        return Math.PI*2.0*radius;
    }

    public void describe() {
        System.out.print("Circle[radius=");
        System.out.print(radius);
        System.out.println("]");
    }

}
```

```java
public class Cone implements ThreeDShape  {
    private double radius;
    private double height;

    public Cone (double radius, double height) {
        this.radius = radius;
        this.height = height;
    }

    public double volume() {
        return (1.0/3.0)*Math.PI*radius*radius*height;
    }

    public void describe() {
        System.out.print("Cone[radius=");
        System.out.print(radius);
        System.out.print(",height=");
        System.out.print(height);
        System.out.println("]");
    }
}
```

```java
public class Rectangle implements TwoDShape  {
    private double width, height;

    public Rectangle (double width, double height) {
        this.width = width;
        this.height = height;
    }

    public double area() {
        return width*height;
    }

    public double perimeter() {
        return 2.0*(width+height);
    }

    public void describe() {
        System.out.print("Rectangle[width=");
        System.out.print(width);
        System.out.print(",height=");
        System.out.print(height);
        System.out.println("]");
    }

}
```

```java
public class Sphere implements ThreeDShape  {
    private double radius;

    public Sphere (double radius) {
        this.radius = radius;
    }

    public double volume() {
        return (4.0/3.0)*Math.PI*radius*radius*radius;
    }

    public void describe() {
        System.out.print("Sphere[radius=");
        System.out.print(radius);
        System.out.println("]");
    }

}
```

1. (8 points) Consider the following Java code fragments. For each fragment, say whether it results in a compile time error, a run time error, or executes without error.

   (a)
   ```java
   TwoDShape[] shapes;
   shapes = new Rectangle[5];
   shapes[0] = new Circle(10.0);
   ```

   (b)
   ```java
   TwoDShape[] shapes;
   shapes = new TwoDShape[5];
   shapes[0] = new Circle(10.0);
   ```

   (c)
   ```java
   ArrayList<TwoDShape> shapes;
   shapes = new ArrayList<Rectangle>();
   shapes.add(new Circle(10.0));
   ```

   (d)
   ```java
   ArrayList<TwoDShape> shapes;
   shapes = new ArrayList<TwoDShape>();
   shapes.add(new Circle(10.0));
   ```

2. (9 points) Consider three alternate implementations of a `total_volume` method to find the total volume of some shapes.

```
public static double total_volume1(ArrayList<ThreeDShape> shapes)
{
    double sum = 0;
    for(ThreeDShape shape : shapes) {
        sum = sum + shape.volume();
    }
    return sum;
}

public static double total_volume2(ArrayList<? extends GeometricShape> shapes)
{
    double sum = 0;
    for(ThreeDShape shape : shapes) {
        sum = sum + shape.volume();
    }
    return sum;
}

public static double total_volume3(ArrayList<? extends ThreeDShape> shapes)
{
    double sum = 0;
    for(ThreeDShape shape : shapes) {
        sum = sum + shape.volume();
    }
    return sum;
}
```

Which of these will compile correctly, and which will give a compile error?

Of the ones that compile correctly, do they provide different functionality? In other words, are there calls to the `total_volume` method that work for some of the correct versions but not for others? If so, give an example of a call that works for some versions but not others, and say which versions it works for. Also give an example of a call that works for all of the versions that compile correctly. (Please continue your answer on the back of this page if you need more space.)

3. (10 points) Write a CLP($\mathcal{R}$) rule `last(S,L)` that succeeds if `L` is the last element of the list `S`. Naturally, you should also be able to use the rule to find the last element of a list, or find all the lists of which `L` is the last element. Fail if `S` is empty.

4. (10 points) Show the output from CLP($\mathcal{R}$) for the following goals. If backtracking will produce additional answers, give all the additional answers (or the first three if there are infinitely many).

   (a) `last([1,2,3,4],L).`

   (b) `last(S,squid).`

5. (16 points) Consider the following CLP($\mathcal{R}$) rules to find the sum of a list of numbers:

```
sum([],0).
sum([X|Xs],X+S) :- sum(Xs,S).
```

(a) Draw the simplified derivation tree for the goal `sum([3,4],A).`

(b) Draw the simplified derivation tree for the goal `sum([5,I,10],50).`

6. (10 points) Write a `last` function in Miranda that takes a list and returns the last element in the list (analogous to the `last` rule in Question 3). What is its type? Signal an error if the list is empty. (Hint: use the Miranda function `error`.)

7. (10 points)

   (a) What is the result of evaluating the following expressions in Scheme? (Just give the value of the final expression.)

   ```
   (define b '(squid))
   (define (f a) (cons a b))
   (let ((a 'clam)
         (b '(octopus)))
     (f 'mollusc))
   ```

   (b) Suppose Scheme used dynamic scoping rather than lexical scoping. In that case, what would be the value of the final expression?

8. (12 points) There are tradeoffs between having a language that is statically typed and one that is dynamically typed. Give two advantages of static typechecking, and two advantages of dynamic typechecking. Give application contexts in which one or the other might be particularly advantageous.

9. (9 points) Consider the following example in an Algol-like language.

```
begin
integer n;
procedure p(k: integer);
    begin
    k := k+5;
    print(n);
    n := n+(2*k);
    end;
n := 0;
p(n);
print(n);
end;
```

(a) What is the output when k is passed by value?

(b) What is the output when k is passed by value result?

(c) What is the output when k is passed by reference?

10. (12 points) Aloysius Q. Hacker, CSE grad student, has decided that a quick route to a Ph.D. is to invent a new programming language that combines features of Miranda and CLP($\mathcal{R}$). His first attempt ("CLPanda", also known as the Giant Panda Language) is like CLP($\mathcal{R}$), except that the constraint solver is lazy. Goals are processed in the same way as in CLP($\mathcal{R}$), but CLPanda postpones solving any constraints until after the last step of a derivation (i.e. until just after the goal is empty). You are doing a CSE 498 project with Aloysius's advisor. The advisor has great confidence in you, and despite your protests that you are but an undergraduate, Aloysius's advisor asks you to straighten Aloysius out on this new language. Your task: to compare CLP($\mathcal{R}$) and CLPanda. Specifically:

   - Are there derivations that are infinite in one language and finite in the other? (In other words, is there a derivation that has exactly the same derivation steps in each language up to a particular goal, but in one language the derivation will no longer continue, but in the other it is infinite?)
   - If there are derivations that are infinite in one language and finite in the other, is the finite version always a successful derivation, always a failed derivation, or are there cases of both successful and failed derivations?
   - Are there any finite derivations that consist of exactly the same derivation steps in each language but that have different answers?

   For each of these cases, if there are such derivations, give an example.

11. (10 points) Scheme has built-in constructs `delay` and `force` to manage delayed evaluation. For example:

```
(define d (delay (begin (print "evaluating ...") (+ 3 4))))
(force d)
(force d)
```

Write an analogous class `Delay` in Smalltalk. You can create and initialize a new instance of `Delay` in any manner that is convenient.

Show the Smalltalk code for delaying the computation of 3+4 (and printing a note to the Transcript), in analogy with the Scheme code shown.

12. (8 points) Extend the class `Delay` with additional instance and class methods so that you can create and initialize an instance of `Delay` with one message, namely

```
Delay expr: ...
```

where ... is the expression being delayed (however you are representing it). Note carefully which of your methods are instance methods and which are class methods.

13. (12 points) Consider the following Smalltalk class definitions.

```
Object subclass: #A
  instanceVariableNames: ''

describe
  Transcript show: 'instance of A'.
  self additionalDescription.

additionalDescription
  Transcript show: ' - no further information available'.
```

```
A subclass: #B
  instanceVariableNames: ''

describe
  Transcript show: 'instance of B'.
  super describe.
```

```
B subclass: #C
  instanceVariableNames: ''

describe
  Transcript show: 'instance of C'.
  super describe.

additionalDescription
  Transcript show: ' - very C-like indeed'.
```

```
C subclass: #D
  instanceVariableNames: ''

additionalDescription
  Transcript show: ' - more like a D'.
```

What is printed when each of the following expressions is evaluated?

- `A new describe`

- `B new describe`

- `C new describe`

- `D new describe`

And now for the world's only known logic programming joke:

Q: How many CLP($\mathcal{R}$) programmers does it take to change a light bulb?

A: Maybe.