CSE 341 Final Exam

(90 points total)

Winter 2001

Name: Solutions

1. (9 points) For the following Algol-like code:

```
begin
   integer i;

   procedure f(k: integer)
   begin
      i := i+1;
      print(k);
      k := 5;
      print(i);
   end;

   i := 0;
   f(i);
   print(i);
end;
```

Give the result when the code is evaluated with

(a) pass by value.

Answer: `0 1 1`

(b) pass by value-result.

Answer: `0 1 5`

(c) pass by reference.

Answer: `1 5 5`

2. (16 points) Consider the following Java class definitions:

```
public abstract class FarmAnimal
{
   public boolean hasMilk() { return false; }
}

public class Chicken extends FarmAnimal
{
   public int countEggs() { return 2; }
}

public class Goat extends FarmAnimal
{
   public boolean hasMilk() { return true; }
}
```

Give the results for compiling and executing each of the following blocks of Java code. If there are errors, indicate whether errors are runtime errors or compile time errors and what the error is.

(a)
```
{
   FarmAnimal a = new FarmAnimal();

   System.out.println(a.hasMilk());
}
```
Compile Error: `class FarmAnimal is an abstract class. It can't be instantiated.`

(b)
```
{
   FarmAnimal a = new Goat();

   System.out.println(a.hasMilk());
}
```
Answer: `true`

(c)
```
{
   FarmAnimal a = new Chicken();

   System.out.println(a.hasMilk());
}
```
Answer: `false`

(d)
```
{
    FarmAnimal a = new Chicken();

    System.out.println(a.countEggs());
}
```
Compile Error: `Method countEggs() not found in class FarmAnimal.`

(e)
```
{
    Chicken c = new Chicken();

    System.out.println(c.countEggs());
}
```
Answer: `2`

(f)
```
{
    Goat g = new Chicken();

    System.out.println(g.countEggs());
}
```
Compile Error: `Incompatible type for declaration. Can't convert Chicken to Goat`

(g)
```
{
    FarmAnimal a = new Goat();

    System.out.println(((Chicken)a).countEggs());
}
```
Runtime Error: `java.lang.ClassCastException: Goat`

(h)
```
{
    FarmAnimal a = new Chicken();

    System.out.println(((Chicken)a).countEggs());
}
```
Answer: `2`

3. (10 points) True or False

   (a) False Even on a machine with multiple processors, only one Java thread can be executing the same line of Java code at once.

   (b) False In Java when methods are overloaded, the correct method to call is determined at runtime.

   (c) True An array in Java is a subclass of Object.

   (d) False An `int` in Java is a subclass of Object.

   (e) False In Scheme there is no difference between the number 5 and a zero argument function that returns the number 5.

   (f) False A language that is statically scoped also has to be statically typed.

   (g) False I/O is not considered a side-effect.

   (h) False A Java class can subclass multiple abstract classes but only one non-abstract class.

   (i) False In Java, all `RuntimeExceptions` must be caught or declared as thrown by functions that could raise them.

   (j) False In Java, `Strings` are immutable so if you want to change a string you need to cast it to a `StringBuffer`.

4. (5 points) In the starter code for the Scheme interpreter project, the `SchemeObject` class had methods `getCar()` and `getCdr()`. Both of these threw `ClassCastExceptions`:

```
public SchemeObject getCdr()
{
   throw new ClassCastException("Took cdr of " + this);
}

public SchemeObject getCar()
{
   throw new ClassCastException("Took car of " + this); }
}
```

   These methods were then overridden in `SchemePair`, a subclass of `SchemeObject`, to actually return the `car` and `cdr` of the pair. Discuss this design choice. Why is it good? Why is it bad?

   Good: This allows us to avoid having to make type casts to extract elements from a list. `getCdr()` returns a `SchemeObject` so to get the `cadr` of a list stored in `SchemeObject` variable `o` (if only `SchemePairs` had `getC[ad]r` methods) we would have to use: `((SchemePair)((SchemePair)o).getCdr()).getCar()`. This is confusing to read and a pain to type.

   Bad: `SchemeObjects` do not have `car` and `cdr` fields. Therefore, having `getC[ad]r` methods on `SchemeObject` is not natural. We have also bypassed the compiler's static type system and will not get useful compile error messages from the compiler. Instead we will get runtime exceptions which are harder to debug.

5. (3 points) Compare pattern matching in Miranda to overloading in Java.

   Pattern matching in Miranda uses the (dynamic) value of the argument to choose which body of the function use. Only one function is defined. In Java, overloading uses the static type of the argument to determine which of several different functions with the same name is called.

6. (3 points) Can static functions in Java be overridden? Why or why not?

   No. Static functions cannot be dynamically dispatched because there is no `this` object to dispatch on. It is true that a static function in a subclass can have the same name as a static function in a parent class; however, the subclass function does not override the parent class function. They are just two different functions.

7. (3 points) Why is the `java.lang.String` class declared `final`?

   It is declared as `final` so that it may not be subclassed. If you could create a subclass of `String`, you could override methods to make a subclass of `String` that is mutable. This is disallowed.

8. (3 points) What is a disadvantage of garbage collection in comparison to managing your memory yourself.

   Typically a programmer can implement their own memory management more efficiently that a general purpose garbage collector can. Garbage collection would be slower.

9. (3 points) A colleague of yours is designing a new language, *Junk*, which is like Java except that it has no garbage collection. Instead, you use `new` and `delete` much like you do in C++. Discuss the utility of immutable Strings in *Junk*.

   Strings in *Junk* cannot be immutable. The `delete` operation, which must exist for all objects, would be a side-effect. There is also the issue of who's responsibility it is to free a string that is shared by several data structures. Because of this, the programmer will likely want a distinct copy of the string for each data structure. Immutable strings are worthwhile when many data structures can share the same string.

10. (5 points) The following non-R^{4}RS compliant code executes in Guile.

```
guile> (define ((add x) y) (+ x y))
guile> add
#<procedure add (x)>
guile> (add 4)
#<procedure (y)>
guile> ((add 4) 5)
9
```

(a) Describe `add`.

Answer: It is a curried two argument addition function.

(b) This is a curious *feature* of Guile. You can assume that the Guile designers did not deliberately make `define` work like this. What is the most logical explanation for what is going on under the hood in Guile that causes the above results?

This is an artifact of the *source-to-source* transformation that is used in Guile to convert the `define` special-form for functions into the simple `define` with a `lambda`. Guile's source-to-source transformation works as follows:

```
(define (<s-expr> <arg-list>) <body-list>)
```

⇒

```
(define <s-expr> (lambda (<arg-list>) <body-list>))
```

This means

```
(define ((add x) y) (+ x y))
```

gets transformed to

```
(define (add x) (lambda (y) (+ x y)))
```

which in turn gets transformed to

```
(define add (lambda (x) (lambda (y) (+ x y))))
```

and this is the definition of a two argument curried addition function.

11. (5 points)

(a) Write a Scheme function `all-but-last` that takes in a list and returns the list without the last element. It is an error to pass `all-but-last` the empty list. For example:

```
(all-but-last '(1 2 3))  => (1 2)
(all-but-last '())       => error!
```

```
(define (all-but-last lst)
   (if (null? (cdr lst))
       '()
       (cons (car lst) (all-but-last (cdr lst)))
   )
)
```

(b) Is your implementation *tail-recursive*?

Answer: Our solution is not. Your solution may be.

12. (5 points)

(a) Write `all_but_last` in Miranda.

```
all_but_last [x] = []
all_but_last (x:xs) = x : all_but_last xs
```

(b) What is the type of `all_but_last`?

Answer: `[*]->[*]`

13. (10 points) Given the following Scheme code:

```
(define x 1)
(define y 10)
(define z 100)

(define frog
   (let ((x 3))
        (lambda (y) (+ x y z))
   )
)

(define (toad z x) (frog (/ z x)))
```

Suppose we now call `(toad 1 2)`.

(a) What is the result?

Answer: `103.5`

(b) Show the defining environment of `frog`. A diagram would suffice.

```
(((x 3))                                     ;; local frame
 ((x 1) (y 10) (z 100) (+ <+ proc>) ...))   ;; global frame
```

(c) Show the environment where the expression `(/ z x)` is evaluated.

```
(((x 2) (z 1)                                ;; local frame
 ((x 1) (y 10) (z 100) (+ <+ proc>) ...))   ;; global frame
```

(d) Show the environment where the expression `(+ x y z)` is evaluated.

```
(((y .5))                                    ;; local frame
 ((x 3))                                     ;;
 ((x 1) (y 10) (z 100) (+ <+ proc>) ...))   ;; global frame
```

14. (10 points) Write a Java program, `Factorial`, that takes as input exactly one command line argument. This argument should be an integer and the program should compute its factorial and print it out. For example:

```
fiji% java Factorial 5
120
```

Gracefully handle errors (Your program should not print a stack trace). You may wish to use `java.lang.Integer`'s `parseInt()` function which is documented in the *Java 2 API Specification* as follows:

> `public static int parseInt(String s) throws NumberFormatException`
> Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign `'-'` (`'\u002d'`) to indicate a negative value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseInt(java.lang.String, int)` method.
>
> **Parameters:** `s` - a string.
>
> **Returns:** the integer represented by the argument in decimal.
>
> **Throws:** `NumberFormatException` - if the string does not contain a parsable integer.

```
public class Factorial
{
   public static final String USAGE = "USAGE: java Factorial <number>";
   public static int factorial(int n)
   {
      int prod = 1;
      for (int i=1; i <= n; i++)
         prod *= i;
      return prod;
   }
   public static void main(String[] args)
   {
      if (args.length != 1)
         { System.err.println(USAGE); return; }

      try
      {
         System.out.println(factorial(Integer.parseInt(args[0])));
      }
      catch (NumberFormatException e)
      {
         System.err.println(args[0] + " must be a number.");
         System.err.println(USAGE);
}  }  }
```