# CSE 341 Midterm — November 9, 1998

Your name: **Answer Key**

---

This exam is open book, open notes. You are to work independently. 71 points total.

1. (24 points) Consider the following Java class definitions. (These compile correctly.)

```
abstract class Plant {

  // return true if this plant has chlorophyll
  abstract public boolean hasChlorophyll();

  public String description() {
    return "a plant.";
  }
}



class Tree extends Plant {
  // height of this tree in feet
  private int height;

  public Tree (int height) {
    this.height = height;
  }

  public String description() {
    return "a tree " + height + " feet tall.  also " + super.description();
  }

  public boolean hasChlorophyll() {
    return true;
  }

  static public String hardness() {
    return "unknown";
  }
}
```

```
class Oak extends Tree {

  public Oak (int height) {
    super(height);
  }

  public String description() {
    return "an oak.  also "  + super.description();
  }

  static public String hardness() {
    return "very hard";
  }
}



class Cedar extends Tree {

  public Cedar (int height) {
    super(height);
  }

  public String description() {
    return "a cedar.  also "  + super.description();
  }

  static public String hardness() {
    return "soft";
  }
}



class Mushroom extends Plant {

  public String description() {
    return "a mushroom.  possibly poisonous, so watch out.  also "
      + super.description();
  }

  public boolean hasChlorophyll() {
    return false;
  }

}
```

Now suppose we also have a class `PlantTest` with a `main` method. What is the result of compiling and executing the Java program for each of the following versions of `PlantTest` and `main`? The result might be that the program runs correctly, or it might have a compile time error, or a runtime error. If the program compiles correctly and can be run, give the output. Otherwise explain what the error is.

(a)
```
public class PlantTest {
  public static void main (String [ ] args) {
    Plant p = new Plant();
    System.out.println(p.hasChlorophyll());
    System.out.println(p.description());
  }
}
```

```
PlantTest.java:4: class Plant is an abstract class. It can't be instantiated.
    Plant p = new Plant();
              ^
1 error
Compile Error
```

(b)
```
public class PlantTest {
  public static void main (String [ ] args) {
    Plant p = new Mushroom();
    System.out.println(p.hasChlorophyll());
    System.out.println(p.description());
  }
}
```

```
Running...

false
a mushroom.  possibly poisonous, so watch out.  also a plant.
```

(c)
```
public class PlantTest {
  public static void main (String [ ] args) {
    Oak o = new Oak(150);
    Tree t = o;
    System.out.println(o.description());
    System.out.println(t.description());

    System.out.println(o.hardness());
    System.out.println(t.hardness());
  }
}
```

```
Running...

an oak.  also a tree 150 feet tall.  also a plant.
an oak.  also a tree 150 feet tall.  also a plant.
very hard
unknown
```

(d)
```
public class PlantTest {
  public static void main (String [ ] args) {
    Oak o = new Oak(150);
    Tree t = (Tree) o;
    System.out.println(t.description());
  }
}
```

```
Running...

an oak.  also a tree 150 feet tall.  also a plant.
```

(e)
```
public class PlantTest {
  public static void main (String [ ] args) {
    Tree t = new Oak(150);
    Oak o = (Oak) t;
    System.out.println(o.description());
  }
}
```

```
Running...

an oak.  also a tree 150 feet tall.  also a plant.
```

(f)
```
public class PlantTest {
  public static void main (String [ ] args) {
    Tree t = new Cedar(200);
    Oak o = (Oak) t;
    System.out.println(o.description());
  }
}
```

```
Running...

java.lang.ClassCastException: Cedar
at PlantTest.main(PlantTest.java:4)
Runtime Exception
```

2. (12 points) Suppose the following Scheme code has been evaluated:

```
(define a 100)
(define b 200)

(define (test1)
   (let ((a 5)
         (b (+ a 3)))
      (+ a b)))

(define (test2)
   (let* ((a 5)
          (b (+ a 3)))
      (+ a b)))
```

Then what is the result of evaluating the following Scheme expressions? If there is an error say so. Write your answer using the printable representation of the result (i.e., what the Scheme system outputs after evaluation).

(a) `(+ a b)`

`;Value: 300`

(b) `(test1)`

`;Value: 108`

(c) `(test2)`

`;Value: 13`

(d) `(map (lambda (b) (* 2 b)) '(1 2 3))`

`;Value 1: (2 4 6)`

3. (20 points) Suppose we have defined the following Scheme functions:

```
(define (squid x y)
  (+ x y))

(define (clam n)
  (squid (* 2 n) 20))
```

Suppose we evaluate the expression `(squid 3 4)` using the metacircular evaluator that you worked on for your Scheme project. The result is 7.

The environment that the interpreter uses when evaluating the expression `(+ x y)` is:

```
(  ((x y) 3 4)
   ((squid clam + * false true car cdr cons null?) ...)
)
```

There are two frames in the environment. The first frame in the environment has bindings for `x` and `y` (which are bound to 3 and 4 respectively). The second frame contains the global variables and their bindings (which I just indicated by `...`).

Suppose we evaluate the expression `(clam 100)`. What is returned?

```
;Value: 220
```

While evaluating `(clam 100)`, what is the environment that the interpreter uses when evaluating the expression `(* 2 n)` in the `clam` function? (You can abbreviate the global environment as above.)

```
( ((n) 100)
  ((squid clam + * false true car cdr cons null?) ...)
)
```

Still while evaluating `(clam 100)`, what is the environment that the interpreter uses when evaluating the expression `(+ x y)` in the `squid` function?

```
( ((x y) 200 20)
  ((squid clam + * false true car cdr cons null?) ...)
)
```

---

Now suppose we've defined yet another pair of functions:

```
(define (test n)
  (helper n n))

(define (helper a b)
  ((lambda (a) (+ a b)) 42))
```

Suppose we evaluate the expression `(test 100)`. What is returned? What is the environment that the interpreter uses when evaluating the expression `(+ a b)`, in other words the environment used when evaluating the body of the lambda after it has been applied to its argument?

```
( ((a) 42)
  ((a b) 100 100)
  ((helper test squid clam + * false true car cdr cons null?) ...)
)
```

4. (15 points) Why do you think the designers of Java included both abstract classes and interfaces? What can you do with an abstract class that you can't do with an interface? What can you do with an interface that you can't do with an abstract class?

## Key Points:

- Java includes both abstract classes and interfaces because there are useful uses for each of them that the other does not provide.
- Abstract classes allow subclasses to inherit partial implementations. Interfaces do not.
- Multiple interfaces can be implemented by one class thus mimicking multiple inheritance. A class can only inherit from one abstract class (or any class for that matter).

## General Discussion:

The Java implementers did not want to allow multiple inheritance for classes because superclasses could then have conflicting methods and instance variables. Multiple inheritance has however, proven to be useful for polymorphism (We want one particular class to be able to be passed to several different methods, each expecting this class have a specific programming interface). The solution, interfaces, that do not inherit functionality but guarantees that the class provide the required programming interface. For this reason Java's designers added interfaces to the Java languages.

Abstract classes can be used to give several classes shared ancestry and shared implementation of a subset of their member functions. Abstract classes are often used when it makes no sense to ever have an instance of the object, but subclasses have a common behavior (and thus implementation). The common implementation for the classes can then go into their abstract base class.

A prime example of this is the *InputStream* abstract class. The *InputStream* has methods, `int read()`, `int read(btye[]`, `int read(byte[],int,int)`, and others. Of all of these methods, only `read()` is abstract. Note that `read(btye[])` and `read(byte[],int,int)` methods call `read()` by default. A subclass could choose to override these, but they do not have to override them. The other non-abstract methods on *InputStream* default to intelligent things. For example, the `void close()` method does nothing. For more details, see the JDK documentation.