CSE 341 Midterm — February 14, 2003 — Answer Key

This exam is closed book and notes. 90 points total.

For the multiple choice questions (1 through 8) circle the correct answer (just one).

The correct answer is in boldface.

1. (5 points) How can you simulate a global function in Java?

   (a) Define an interface
   (b) **Define a static member function of a class**
   (c) Define an inner class
   (d) None of the above

2. (5 points) Consider this code:

```
class Point3D {
    private int x, y, z;
    ...
};
```

   What can you tell about the location of x, y, and z in any variable of type Point?

   (a) Always on stack
   (b) Either on stack or heap
   (c) **Always on heap**
   (d) None of the above

3. (5 points) In Java, what is the relationship between Integer and int?

   (a) Integer inherits int
   (b) int inherits Integer
   (c) **Integer boxes an int into a Java object**
   (d) They are synonyms
   (e) None of the above

4. (5 points) When can the garbage collector collect the memory allocated from an object?

   (a) When it can prove there is no reference to that object from any other object
   (b) When the reference initialized with “new” goes out of scope
   (c) When the method in which the object was created returns
   (d) Only when the program terminates
   (e) **None of the above**

   We also are accepting a) — the question should have said “only when it can prove . . . ” to make this unambiguous

5. (5 points) Consider the code below:

```
Ball b1 = new Ball();
Ball b2 = b1;
b2 = (Ball)b2.clone();
```

Which statement is true?

(a) `b1 == b2 && b1.equals(b2)`

(b) **`b1 != b2 && b1.equals(b2)`**

(c) `b1 == b2 && !b1.equals(b2)`

(d) `b1 != b2 && !b1.equals(b2)`

Answer d) is also acceptable, since we didn't provide a definition of `equals` for `Ball`. (If you don't override `equals`, the default inherited from Object does an `==` comparison.)

6. (5 points) Which of the following is true about Java interfaces?

(a) A class can only implement one interface but inherit multiple classes

(b) A class can implement multiple interfaces and inherit multiple classes

(c) An interface can inherit a class

(d) **A class can only inherit one class but implement multiple interfaces**

(e) None of the above

7. (5 points) Which statement is true regarding final classes in Java?

(a) They can't implement any interface

(b) They don't inherit from any class

(c) **They don't have any subclasses**

(d) They aren't accessible outside their package

(e) None of the above

8. (5 points) What does "fail-fast" mean for an iterator?

(a) As soon as the iterator reaches the end, calling `next()` will throw an exception

(b) The iterator's state is updated appropriately, so that it doesn't fail, if the collection is changed via another iterator

(c) **The iterator throws an exception if the collection is changed except by that iterator**

(d) None of the above

However, we decided not count this question, since this was on the optional part of the homework.

9. (10 points) Consider the following program in an Algol-like language.

```
begin
j: integer;
procedure p(k: integer);
    begin
    j := j+1;
    k := j+k;
    print(k);
    end;
j := 10;
p(j);
end;
```

What is printed if j is passed by value? By reference?

By value: 21
By reference: 22

10. (10 points) Consider the following program in the same Algol-like language.

```
begin
j: integer;
procedure p(k: integer);
    begin
    print(k);
    j := j+1;
    print(k);
    end;
j := 10;
p(2*j);
end;
```

What is printed if j is passed by value? By name?

By value: 20 20
By name: 20 22

11. (10 points) Briefly explain in words the difference between overloading and overriding. Give an example of each in Java.

Overloading means that there are two or more methods with the same name, but with different numbers or types of arguments. These are separate and distinct methods, and you can have several overloaded methods in the same class. Overriding means that a method is defined in a class, and then redefined in a subclass. The method in the subclass, which overrides the inherited method, must have the same name, number and types of arguments, and return type.

A Java example of overloading is the `println` method in `PrintStream`. (`System.out` is a printstream.) There are 10 methods of this name defined in `PrintStream`, including `println(Object x)`, `println(float x)`, and `println(int x)`.

A Java example of overriding is the `equals (Object x)` method, which is defined in `Object`, and overridden in `Point` (and many other classes).

The above is all that was expected for full credit — here are some additional details.

In Java, you can't overload based just on the return type — the arguments have to be different.

Overloading also applies to operators in Java (e.g. +). Overloading occurs in most programming languages, not just object-oriented ones. For example, the + operator is overloaded in Fortran, Algol, C, and many other languages.

Overriding pertains to all object-oriented languages, and is a defining feature of being object-oriented.

12. (10 points) Consider call-by-value versus call-by-reference. When is one more advantageous than the other in regard to efficiency?

For call-by-value, the actual parameter value must be copied into the formal. The formal parameter can be accessed very efficiently, since it is stored directly in the stack frame. Thus call-by-value is more efficient for data that is stored in a small amount of memory, for example, an integer or boolean.

For call-by-reference, no copying is done; any use of the formal parameter indirects to the actual parameter. It is more efficient for data that takes a large amount of memory, such as a large array, for which copying would be expensive. (Each access takes a little longer, however.)

13. (10 points) Consider the following Java classes.

```
class Pair {
  private int i;
  private String s;
  public Pair(int i, String s) {
    this.i = i;
    this.s = s;
    }
}

class Octopus {
  public static void main(String[] args) {
    Octopus o = new Octopus();
    o.test();
    }

  public void test() {
    int j;
    String t;
    Pair p;
    j = 42;
    t = "hello Mr. squid";
    p = new Pair(j, t);
    System.out.println("leaving test");
    }
}
```

Draw a picture of Java's memory structure just before the `println` method is invoked. Include both the stack and the heap, including the stack frames for `main` and `test`.