Supervision Questions
Chapters 1-4

Matthew Johnson
Trinity Hall
mjj29@cam.ac.uk

October 26, 2002

2-1

```
class Integer
{
   Integer(int i)
   {
      value = i;
   }
   int value;
}

....

Integer int1 = new Integer(8);
Integer int2 = new Integer(53);
System.out.println(int1.value);
```

A *class* is a definition of a type of object, and says how any instance of that class would behave, what interfaces it has, what data it can store, and the definition of the methods that operate on its data, encapsulated into one item. In this case `Integer` is a simple class that just holds one value. To actually use a class (except in the case of `static` items) you need to create an instance of the *object* it represents. The call `new Integer(8)` creates an object which is an instance of the class `Integer` and initializes it using the constructor with the value 8. The identifiers `int1` and `int2` are *object references*. They are pointers in memory to the location of the objects that they refer to. Objects in Java cannot be accessed or instantiated directly, but must be accessed via object references (in contrast to C++ or Pascal, which differentiate between static and dynamic memory allocation). The *type* of a variable refers to a definition of the storage of the variable, and the operations you can do on it. You declare the type when you instantiate the variable. For example, the type of `value` in the example is `int` and the type of `int1` is `Integer`.

2-2

If s is a sub-type of t, that is in Java, if s `extends` t, then it can be used wherever something of type t can be used. This is possible, because Java performs dynamic method invocation, so when you access a method of something of type t, it checks what object the reference is referring to, and looks up the actual type of that object. It then invokes the method from the class that the object was actually instantiated from.

2-3

The syntax `super.super` isn't valid in Java. This isn't too much of an issue, because the typical use of the super keyword, is when extending methods from a class you are inheriting from. When you override a method by default it replaces the original. If you want to merely extend it, then you call `super`, and then add in the extra code. You rarely need to use this to call methods from anything other than your immediate parent class. For accessing fields, you can simulate multiple `super` calls by casting `this` to the appropriate type, and then accessing the field. The advantages of disallowing `super.super` are mainly that you don't have to check that it is a valid class. Since everything extends at least `Object`, then `super` is always valid, but `super.super` isn't necessarily.

3-2

Abstract classes are very similar to interfaces in Java, and can often be used in similar places. However, there are several difference. Abstract classes can have any number of their methods implemented in the abstract class, even all of them. A class extending them only has to provide implementations for the methods that the abstract class doesn't implement itself. Interfaces, on the other hand, have no implementations. This may seem like a case for only using abstract classes, but since Java doesn't allow multiple inheritance, you can only inherit from one class. You can, however, implement as many interfaces as you like. This trade off has helped the Java language from not allowing clashes of implementations of methods, but does slightly restrict what you can do.

3-5

The following Java program implements the given example using an anonymous inner class.

```
interface Closure
{
   public void apply ();
}

class test
{
   static Closure myCounter (final int start)
   {
      return new Closure () {
         int counter = start;
         public void apply ()
         {
            System.out.println (counter++);
         }
      };
   }

   public static void main (String [] args)
   {
      Closure count = myCounter (5);
      for (int i = 1; i <= 6; i++) count.apply ();
   }
}
```

When compiled and run you get the following output:

```
mjj29@binford:~/Docs/personal/cambridge/CompSci/csaa$ java test
5
6
7
8
9
10
```