Computer Science 3718 (Fall 2003)
Class Exam #2 Answers

1. (26 marks)

   a) (12 marks) Consider the following Java program:

   ```java
   class XObj {
   public int  a = 1, b = 2;

   public static void main(String[] arg){
     XObj  x = new XObj();
     YObj y = new YObj(), z;
     int b = 4, c = 5;

     z.printVal();
     x.printVal();
     c = 6; y = z; b = 7;
     y.printVal();
   }

   public void printVal(){
     int c = 8;
     System.out.println("Xver: " + a + " + b + " + c);  
     a = 13;
   }
   }

   class YObj extends XObj {
     public int b = 9;
     private int c = 10;

     public void printVal(){
       System.out.println("Yver: " + a + " + b + " + c);  
       int c = 11;
     }
   }
   }
   ```

   Please give the output produced by executing this program.

   Yver: 1 9 10
   Xver: 1 2 8
   Yver: 1 9 10
b) **(14 marks)** Consider the following UML class-relationship diagram:

There are several design restrictions on these classes:
- Classes C2 and C3 cannot have associated object-instances.
- Field f3 in class C4 and method m1 in class C2 are inherited by and accessible in all of their subclasses.
- Field f2 in class C2 and method m2 in class C4 are associated with their respective classes and not with object-instances of those classes.
- Field f1 in class C2 is a constant with value 5.
- Method m1() in class C1 cannot be overridden in any subclass.
- Classes C1 and C5 cannot have subclasses.
- All remaining fields and methods are associated with object-instances of their respective classes.
- All remaining methods are accessible only within the classes in which they are defined and all remaining fields are accessible to all classes.

Please give the **Java** code “skeletons” (class definitions with all fields and the first line of each method (you can leave out the constructors)) for all classes in this diagram under these design restrictions.

```java
final class C1 extends C3 {
    public double f1;
    private final int m1(){...}
}

abstract class C2 {
    public final int f1 = 5;
    public static int f2;
    protected int m1(){...}

    abstract class C3 extends C2 {
        public int f4;
        private int m3(){...}
    }

    class C4 extends C3 {
        protected int f3;
        private static int m2(){...}
    }

    final class C5 extends C2 {
        public int f3;
        private int m2(){...}
    }
```
2. **(24 marks)** Give Java code for class Chart as specified in the MIS handout. This class should include all specified fields and methods associated with this class. All fields should be accessible only within this class, and all requested methods should be publicly available. You may assume the existence of all classes and methods described or mentioned in the MIS handout as well as the `Date` method

    public boolean isBetween(Date start, Date end)

which returns `true` if the calling `Date`-object specified a date between `start` and `end` and `false` otherwise.

class Chart {
    ChartEntry[] CA;
    Patient P;
    Doctor Dp;

    public Chart(Patient P, Doctor Dp){
        this.P = P;
        this.Dp = Dp;
        CA = null;
    } // End of Chart constructor

    public Doctor getDDoctor(){
        return(Dp);
    } // End of method getDDoctor

    public Doctor changeDDoctor(Doctor newDp){
        Dp = newDp;
    } // End of method changeDDoctor

    public void addEntry(ChartEntry newEntry){
        if (CA == null) {
            CA = new ChartEntry[1];
            CA[0] = newEntry;
        } else {
            ChartEntry[] CAT = new ChartEntry[CA.length + 1];
            System.arraycopy(CA, 0, CAT, 1, CA.length);
            CAT[0] = newEntry;
            CA = CAT;
        }
    } // End of method addEntry
public DoctorSet getDoctorsDate(Date d1, Date d2){
    DoctorSet CDS, MDS = new DoctorSet(null);
    Doctor DT;
    int i;

    if (CA != null) {
        for (i = 0; i < CA.length; i++){
            if (CA[i].getMatch().isBetween(d1,d2)) {
                CDS = CA[i].getDoctors();
                i = 1;
                while ((DT = CDS.getDoctorInd(i++) ) != null) {
                    MDS.addDoctor(DT);
                }
            }
        }
    }
    return(MDS);
}
} // End of method getDoctorsDate

public ChartEntry[] getEntriesDate(Date d1, Date d2){
    ChartEntry[] CAT = new ChartEntry[0];
    int i, numEntries = 0;

    if (CA != null) {
        for (i = 0; i < CA.length; i++)
            if ((CA[i].getMatch()).isBetween(d1,d2))
                numEntries++;
        CAT = new ChartEntry[numEntries];
        numEntries = 0;
        for (i = 0; i < CA.length; i++)
            if ((CA[i].getMatch()).isBetween(d1,d2))
                CAT[numEntries++] = CA[i];
    }
    return(CAT);
}
} // End of method getEntriesDate

} // End of class Chart