1. (26 marks)

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

```java
class XObj {
    public int a = 9, b = 3, c = 7;

    public static void main(String[] arg){
        XObj z = new XObj(), x;
        YObj y = new YObj();
        int[] w = {-1, 20};
        int b = 4, c = 5;

        System.out.println(z.modVal(w) + " " + w[0] + " " + w[1]);
        System.out.println(y.modVal(w) + " " + w[1]);
        b = 6; x = z; c = 7;
        System.out.println(x.modVal(w) + " " + w[1] + " " + w[0]);
    }

    public String modVal(int[] x){
        x[0] = a;
        int a = 11;
        return("Xver: " + a + " " + b + " " + c);
    }
}

class YObj extends XObj {
    private int c = 2;

    public String modVal(int[] y){
        int c = 13;
        y[1] = 42;
        return("Yver: " + a + " " + b + " " + c + " " + y[0]);
    }
}

Please give the output produced by executing this program.
b) (14 marks) Consider the following UML class-relationship diagram:

There are several design restrictions on these classes:

- Classes C3 and C5 cannot have associated object-instances.
- Field f3 in class C4 and method m1 in class C1 are inherited by and accessible in all of their subclasses.
- Field f2 in class C3 and method m2 in class C5 are associated with their respective classes and not with object-instances of those classes.
- Field f4 in class C3 is a constant with value 5.
- Method m2() in class C4 cannot be overridden in any subclass.
- Class C4 cannot have subclasses.
- All remaining fields and methods are associated with object-instances of their respective classes.
- All remaining fields are accessible only within the classes in which they are defined and all remaining methods 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.
Answer:

```java
class C1 extends C2 {
    private double f1;
    protected int f3;
    protected int m1() {...}
}

class C2 extends C3 {
    private int f1;
    private int f3;
    public int m1() {...}
}

abstract class C3 {
    private static int f2;
    private final int f4 = 5;
    public int m3() {...}
}

abstract class C4 extends C5 {
    private double f1; 
    protected int f3; 
    public final int m2() {...}
}

final class C5 extends C2 {
    private int f1; 
    protected int f3; 
    public static int m2() {...}
}
```

2. (24 marks) Give Java code for methods Schedule, makeAppt, getAppts, and delApptRoom in class Schedule as specified in the MIS handout. This code should include the class header and all specified fields 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.

Answer:

```java
class Schedule {
    private Appointment[] AP;

    public Schedule { AP = new Appointment[0]; }

    public boolean makeAppt(Appointment A) {
        Appointment[] APT;
        int i;

        for (i = 0; i < AP.length; i++)
            if ((A.getRoom().equals(AP[i].getRoom())) &&
                Date.dateRangeOverlap(A.getTime(), AP[i].getTime()))
                return(false);

        APT = new Appointment[AP.length + 1];
        for (i = 0; i < AP.length; i++)
            APT[i] = AP[i];

        return(true);
    }
}
```
APT[AP.length] = A;
AP = APT;
return(true);
}

public Appointment[] getAppts(String[] DR, String room) throws InvalidDateException, InvalidDateRangeException {
  Appointment[] APT;
  int i, count = 0;

  for (i = 0; i < DR.length; i++)
    if (!Date.isValidDate(DR[i]))
      throw new InvalidDateException(DR[i]);
  if (!Date.isValidDateRange(DR))
    throw new InvalidDateRangeException(DR[0] + " -- " + DR[1]);

  for (i = 0; i < AP.length; i++)
    if (room.equals(AP[i].getRoom()) && Date.dateRangeOverlap(DR, AP[i].getTime()))
      count++;
  APT = new Appointment[count];
  count = 0;
  for (i = 0; i < AP.length; i++)
    if (room.equals(AP[i].getRoom()) && Date.dateRangeOverlap(DR, AP[i].getTime()))
      APT[count++] = AP[i];
  return(APT);
}

public void delApptRoom(String room) {
  Appointment[] APT;
  int i, count = 0;

  for (i = 0; i < AP.length; i++)
    if (room.equals(AP[i].getRoom()))
      count++;
  APT = new Appointment[AP.length - count];
  count = 0;
  for (i = 0; i < AP.length; i++)
    if (!room.equals(AP[i].getRoom()))
      APT[count++] = AP[i];
  AP = APT;
}