1. Use the following code to illustrate the Bankers algorithm and explain what is occurring at each step. Annotate your output to illustrate what is happening.

```
1 //on a high level:
2 //Banker's algorithm keeps deadlock at bay by tracking resources
3 //and allocating appropriately. It does this by tracking 3 things:
4 //The maximum a process can allocate (MAX)
5 //The amount it has already allocated (ALLOCATE)
6 //The resources available to use (AVAIL)
7 //It allows resources to be allocated if the amount requested (NEED) is less
8 //than or equal to the amount available. If not, it waits until they are.
9
10 import java.util.Scanner;
11
12 public class Bankers{
13     //Banker's algorithm works on 3 things:
14     //This implementation includes extras: NEED and 2 helper variables
15     //np,nr store user input and put it into the right place in input()
16     private int need[][],allocate[][],max[][],avail[][],np,nr;
17
18     //input() handles adding all the variables to the arrays:
19     //MAX,ALLOCATE,AVAIL
20     private void input(){
21      //Scans user input
22      Scanner sc=new Scanner(System.in);
23      System.out.print("Enter no. of processes and resources : ");
24      //Sets length/width of the 2D arrays using user input
25      np=sc.nextInt();  //no. of process
26      nr=sc.nextInt();  //no. of resources
```

```
27      need=new int[np][nr];  //initializing arrays
28      max=new int[np][nr];
29      allocate=new int[np][nr];
30      avail=new int[1][nr];
31
32      System.out.println("Enter allocation matrix -->");
33      for(int i=0;i<np;i++)
34           for(int j=0;j<nr;j++)
35          allocate[i][j]=sc.nextInt();  //allocation matrix
36
37      System.out.println("Enter max matrix -->");
38      for(int i=0;i<np;i++)
39           for(int j=0;j<nr;j++)
40          max[i][j]=sc.nextInt();  //max matrix
41
42         System.out.println("Enter available matrix -->");
43         for(int j=0;j<nr;j++)
44          avail[0][j]=sc.nextInt();  //available matrix
45
46         sc.close();
47     }
48
49     private int[][] calc_need(){
50        for(int i=0;i<np;i++)
51          for(int j=0;j<nr;j++)  //calculating need matrix
52           need[i][j]=max[i][j]-allocate[i][j];
53
54        return need;
55     }
56
```

```
57      private boolean check(int i){
58         //checking if all resources for ith process can be allocated
59         for(int j=0;j<nr;j++)
60         if(avail[0][j]<need[i][j])
61            return false;
62
63      return true;
64      }
65
66      public void isSafe(){
67         input();
68         calc_need();
69         boolean done[]=new boolean[np];
70         int j=0;
71
72         while(j<np){  //until all process allocated
73         boolean allocated=false;
74         for(int i=0;i<np;i++)
75          if(!done[i] && check(i)){  //trying to allocate
76              for(int k=0;k<nr;k++)
77              avail[0][k]=avail[0][k]-need[i][k]+max[i][k];
78            System.out.println("Allocated process : "+i);
79            allocated=done[i]=true;
80                  j++;
81               }
82            if(!allocated) break;  //if no allocation
83         }
84         if(j==np)  //if all processes are allocated
85          System.out.println("\nSafely allocated");
86         else
```

```
87          System.out.println("All proceess cant be allocated safely");
88      }
89
90      public static void main(String[] args) {
91          new Bankers().isSafe();
92      }
93  }
```

2. Give examples of inputs where a safe allocation of processes occurs and one where processes cannot be allocated safely.

3. What conditions cause the former to happen? The latter? Clearly indicate these in your writeup. (e.g., for all i, j, when max[i][j] <avail[i][j])

4. From a big picture perspective, why is this implementation of resource allocation so widely appreciated?