```
/*****************************************************************************
Java program to implement Banker's Algorithm
for deadlock handling.
*****************************************************************************/
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
     Scanner sc=new Scanner(System.in);
      int N,M;


    System.out.println("\n***Banker's Algorithm***");
    System.out.println("\nSteps:");
    System.out.println("\nAccept details");
    System.out.println("\nDisplay details");
    System.out.println("\nDisplay safe sequence");


    System.out.println("\nEnter the number of processes:");
    N=sc.nextInt();
    System.out.println("\nEnter the number of resource types:");
    M=sc.nextInt();

    int MaxResources[] =new int[M];
    int Allocation[][]=new int[N][M];
    int SafeSequence[]=new int[N];;

    int Available[]=new int[M];
    System.out.println("\nEnter the maximum instances for each resource type:");

    for(int t=0;t<M;t++)
    {
        System.out.println("For resource:"+t);
        MaxResources[t]=sc.nextInt();
    }


    int maximumForEachP[][]=new int[N][M];

    //Enter the elements of maximum matrix
    for(int i=0;i<N;i++)
    {
        System.out.println("\nFor process:"+i);
        System.out.println("Enter the maximum available instances for each resource type:");
        for(int j=0;j<M;j++)
        {
           maximumForEachP[i][j]=sc.nextInt();
        }
    }

     System.out.println("\nEnter the resources allocated for each process\n");
    //Enter the elements of allocation matrix
```

```
for(int i=0;i<N;i++)
{
   System.out.println("For process:"+i);
   System.out.println("Enter instances for resource type:");

    for(int j=0;j<M;j++)
    {
     Allocation[i][j]=sc.nextInt();
    }
}


int NeedforEachP[][]=new int[N][M];
boolean finishedProcess[]=new boolean[N];

int TAlloted[]=new int[M];
int Work[]=new int[M];
for(int j=0;j<M;j++)
{
  for(int i=0;i<N;i++)
   {
    TAlloted[j]+=Allocation[i][j];
  }
}
System.out.print("\nTAlloted[j]");
for(int j=0;j<M;j++)
{
   System.out.print(TAlloted[j]+" ");
}
System.out.print("\nMaxResources[j]");
for(int j=0;j<M;j++)
{
   System.out.print(MaxResources[j]+" ");
}
for(int i=0;i<M;i++)
{
    Work[i] = MaxResources[i]-TAlloted[i];
}
 for(int k=0;k<N;k++)
 {
     finishedProcess[k]=false;
 }


System.out.print("\n\nAllocation:");
for(int i=0;i<N;i++)
{
   System.out.print("\nFor Procress"+i+":     ");
   for(int j=0;j<M;j++)
    {
        System.out.print(Allocation[i][j]+" ");

    }
```

```
}
System.out.print("\n\nMaximum:");
for(int i=0;i<N;i++)
{
   System.out.print("\nFor Procress"+i+":    ");
   for(int j=0;j<M;j++)
   {
      System.out.print(maximumForEachP[i][j]+" ");
   }
}
System.out.print("\n\nNeed:");
for(int i=0;i<N;i++)
{
   System.out.print("\nFor Procress"+i+":    ");
    for(int j=0;j<M;j++)
    {
    NeedforEachP[i][j]=maximumForEachP[i][j]-Allocation[i][j];
    System.out.print(NeedforEachP[i][j]+" ");
    }
}


System.out.print("\n\nAvailable Resources or Work:   ");
for(int i=0;i<M;i++)
System.out.print(Work[i]+"  ");


int count=0;
int flag=0;
int c=0;
do
{
   for (int i = 0;i<N; i++)
   {
      count++;
      if(finishedProcess[i]==false)
      {
        int j;
        flag=0;
        for (j = 0;j<M; j++)
        {
          if (NeedforEachP[i][j]<=Work[j])
          {
            flag++;
          }
        }
        if (flag==M)
        {
          SafeSequence[c]=i;
          c++;
          finishedProcess[i]=true;
          for (j=0;j<M; j++)
          {
               Work[j] = Work[j]+Allocation[i][j];
             }
             System.out.print("\nAfter execution of P"+i+":Available:");
             for(int t=0;t<M;t++)
             {
               System.out.print(Work[t]+" ");
             }
           }
         }//close if
      }//close for

   }while(count<=2*N);//close do while loop

   if(c==N)
   {
      int i;
      System.out.println("\nThe SAFE Sequence for the given system is");
      for (i=0;i<(N-1); i++)
      {
         System.out.print("P"+SafeSequence[i]+",");
      }
      System.out.print("P"+SafeSequence[i]);
   }
   else
   {
      System.out.println("The System is UnSafe!");
   }


  }
}
/*OUTPUT

***Banker's Algorithm***

Steps:

Accept details

Display details

Display safe sequence

Enter the number of processes:
5

Enter the number of resource types:
3

Enter the maximum instances for each resource type:
For resource:0
10
For resource:1
5
```

```
For resource:2
7

For process:0
Enter the maximum available instances for each resource type:
7
5
3

For process:1
Enter the maximum available instances for each resource type:
3
2
2

For process:2
Enter the maximum available instances for each resource type:
9
0
2

For process:3
Enter the maximum available instances for each resource type:
2
2
2

For process:4
Enter the maximum available instances for each resource type:
4
3
3

Enter the resources allocated for each process

For process:0
Enter instances for resource type:
0
1
0
For process:1
Enter instances for resource type:
2
0
0
For process:2
Enter instances for resource type:
3
0
2
For process:3
Enter instances for resource type:
2
1
```

```
1
For process:4
Enter instances for resource type:
0
0
2

TAlloted[j]7 2 5
MaxResources[j]10 5 7

Allocation:
For Procress0:    0 1 0
For Procress1:    2 0 0
For Procress2:    3 0 2
For Procress3:    2 1 1
For Procress4:    0 0 2

Maximum:
For Procress0:    7 5 3
For Procress1:    3 2 2
For Procress2:    9 0 2
For Procress3:    2 2 2
For Procress4:    4 3 3

Need:
For Procress0:    7 4 3
For Procress1:    1 2 2
For Procress2:    6 0 0
For Procress3:    0 1 1
For Procress4:    4 3 1

Available Resources or Work:   3  3  2
After execution of P1:Available:5 3 2
After execution of P3:Available:7 4 3
After execution of P4:Available:7 4 5
After execution of P0:Available:7 5 5
After execution of P2:Available:10 5 7
The SAFE Sequence for the given system is
P1,P3,P4,P0,P2
*/
```