Experiment No:

Aim: To implement Banker's Algoritum for:
(a) Dead Lock Avoidance
(b) Dead Lock Prevention

Description:

The Deadlock avoidance algoritum examines the resource allocations so that there can never be a circular wait condition. The resource allocation state of a system can be defined by the instances of available and allocated resources. and the maximum instance of the resources demanded by the processes.

Example:

Let us consider n = 5, m = 3

Allocation	max	Available	need
P0 0 1 0	7 5 3	A B C	7 4 3
P1 2 0 0	3 2 2	3 3 2	1 2 2
P2 3 0 2	9 0 2		6 0 0
P3 2 0 1	2 2 2		0 1 1
P4 0 0 2	4 3 3		4 3 1

⇒ need = max. Allocation

First P0 is selected But, need Available. so P1 is selected

Available : 5 3 2

P3 is selected : Available : 7 4 3

P0 is selected : Available : 7 5 3

P2 is selected : Available : 10 5 5

P4 is selected : Available : 10 5 7

⇒ Execution order is : P1, P3, P0, P2, P4

Program:

```
import java.io.*;
import java.util.*;
class Bankers {
    private char res = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'};
    private int work1[], avble[];
    private int[][] max, alloc, need;
    int i, j, ch = 0, n, m;
    Scanner sc = new Scanner(System.in);
    void input() {
     Scanner sc = new Scanner(System.in);
     System.out.println("\n Enter the no of Processes:");
     n = sc.nextInt();
     System.out.println("\n Enter the no of Resources: ");
      m=sc.nextInt();
     System.out.println("\n Enter maximum instances of each resource");
     for(i=0; i<m; i++){
          System.out.println("\n Resource: "+ res[i]);
          avble[i] = sc.nextInt();
          max = new int[n][m];
          System.out.println("\n Enter the maximum matrix");
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++){
            max[i][j]=sc.nextInt();
       } }
    alloc = new int[n][m];
    System.out.println("\n Enter allocation matrix:");
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++){
            alloc[i][j] = sc.next();
      }}
    work1 = new int[m];
    for(i=0; i<m; i++){
          work[i]= new int[m];
```

```
for(j=0; j<n; j++)
    work1[j] += alloc[j][i];
    avble[j] = avble[j] - work1[j];
  }
  need = new int[n][m];
  for(i=0; i<n; i++){
    for(j=0; j<m; j++){
      need[i][j] = max[i][j] - alloc[i][j];
    }
  }
  void disp() throws Exception{
    int i,j;
    System.out.println("\nProcess\tAllocation\tMax\tNeed\tAvailable");
    System.out.println("\n\n");
    for(i=0; i<4; i++){
      for(j=0; j<m; j++)
        System.out.print("\t"+res[j]);
        System.out.print("\t");
      }
    for(i=0; i<n; i++){
        System.out.print("\n\np"+i+"\t");
    for(j=0; j<m; j++)
        System.out.print(" "+alloc[i][j]);
        System.out.print("\t\t");
    for(j=0; j<m; j++)
        System.out.print(" "+max[i][j]);
        System.out.print("\t\t");
    for(j=0; j<m; j++)
        System.out.print(" "+need[i][j]);
        System.out.print("\t");
    if(i==0){
      for(j=0; j<m; j++)
        System.out.
```

```
Void safesq() throws Exception{
    int i,j,k=0,l,flag=0,flag1=0;
    int work[]=new int[m];
    int fin[]=new int[n];
    int safesq[]=new int[n+1];
    for(i=0;i<m;i++)
        work[i]=avble[i];
    for(i=0;i<n;i++)
        fin[i]=0;
    for(i=0;i<n;i++){
        for(i=0;i<n;i++){
            flag1=0;
            if(fin[i]==0){
                for(j=0;j<m;j++){
                    if(j=0;j<m;j++){
                        if(need[i][j]>work[j]){
                            flag1=1;
                            break;
                    }}
            if(flag1==0){
                for(j=0;j<m;j++)
                    work[j]=work[j]+alloc[i][j];
                fin[i]=1;
                safesq[k]=i;
                k++;
    }}}}}
for(i=0;i<n;i++){
    if(fin[i]==0){
        System.out.println("unsafe for the given Requirement
                                        the system is");
        System.out.println("not in safe state\n");
        flag=1;
        break;
    }}
```

```
if (flag==0){
    for(i=0;i<m;i++){
        if (req[i]>avble[i]){
            flag1=1;
            break;
        }}
    if (flag1==0){
        System.out.println("request is Accepted");
    for(i=0;i<m;i++){
        avble[i]=avble[i]-req[i];
        alloc[num][i]=alloc[num][i]+req[i];
        need[num][i]=need[num][i]-req[i];
    }}
    else{
      System.out.println("\n\t the process p"+num+" has to wait
                                              as resource");
      System.out.println("Alc not Available\n");
    }
}}
Public Class DeadLock Avoidance{
    Public static void main(String[]args){
        int ch=0;
        Bankers b= new Bankers();
        Scanner sc = new Scanner(System.in);
        b.input();
        while(ch!=4){
          System.out.println("\n menu\t 1.Display Data\n
          2.Geneuatesafe sequence\n 3. resource request\n
          4. Exit\n\t Enter your choice\t");
          ch=sc.nextInt();
          Swit ch(ch){

            Case1:
              b.display();
              break;
```

```
            case 2:
                b.safesq();
                break;
            case 3:
                b.resreq();
                b.disp();
                b.safesq();
                break;
            case 4:
                System.out.println("Thank you");
                break;
            default:
                System.out.println("\n Invoid choice entered");
        }
    }
}
```

Output:

```
Enter no of process : 5
enter no. of resources: 3
enter the maximum instances of each resource
resource: A
10
resource: B
5
resource: C
7
Enter the maximum matrix
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the allocation matrix
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
```

1. Display Data
2. Generate safe sequence
3. resource request
4. exit

enter your choic:

The safe sequence is: P1 P3 P4 P0 P2

menu:

1. Display Data
2. Generate safe sequence
3. Resource Request
4. Exit

Enter your choice: C1

Thank you.