Name: Shardul Vikram Dharmadhikari
Roll no: 8016

ASSIGNMENT NO: 09

AIM: Write a Java program to implement Banker's Algorithm

1.1 Prerequisite:
Basic concepts of Deadlock

1.2 Learning Objectives:
Understand the implementation of the Banker's Algorithm

1.3 Relevant Concepts:

Define Deadlock with an example
2. Resource allocation Graph
3. Wait-for-Graph
4. The Banker's algorithm
5. Safe and Unsafe States
6. Deadlock Characterization
7. Deadlock Prevention
8. Deadlock Avoidance
9. Banker's Algorithm
10. Limitations

THEORY:

1.4 Design Analysis Implementation Logic:

Data Structures for the Banker's Algorithm
Let n = number of processes, and m = number of resources types.

• Available: Vector of length m. If available [j] = k, there are k instances of resource type Rj available.

• Max: n x m matrix. If Max [i,j] = k, then process Pi may request at mostk instances of resource type Rj.

• Allocation: n x m matrix. If Allocation[i,j] = k then Pi is currently allocated k instances of Rj
Need: n x m matrix. If Need[i,j] = k, then Pi may need k more

• Instances of Rj to complete its task. Need [i,j]= Max[i,j] – Allocation [i,j]. **1.5**

Safety Algorithm :-

1. Let Work and Finish be vectors of length m and n, respectively.

Initialize:
Work = Available
Finish [i] = false for i - 1,3, …, n.
2. Find and i such that both:
(a) Finish [i] = false
(b) Needi<=Work If no such i exists, go to step 4.
3. Work = Work + Allocationi Finish[i] = true go to step 2.
4. If Finish [i] == true for all i, then the system is in a safe state

Resource-Request Algorithm for Process Pi

Request = request vector for process Pi.
If Requesti [j] = k then process Pi wants k instances of resource type Rj. 1. If

Requesti Needi go to step

2. Otherwise, raise error condition, <= since process has exceeded its maximum claim. 2. If Requesti Available, go to step

3. Otherwise Pi must wait, since <= resources are not available.

3. Pretend to allocate requested resources to Pi by modifying the state as follows: Available= Available - Requesti;
Allocationi = Allocationi + Requesti;
Needi = Needi – Requesti;

- **If safe state the resources are allocated to Pi.**

- **If unsafe state Pi must wait, and the old resource-allocation state is restored 1.6**

Testing:

Available system resources are:

A	B	C	D
3	1	1	2

Processes (currently allocated resources):

	A	B	C	D
P1	1	2	2	1
P2	1	0	3	3
P3	1	1	1	0

Processes (maximum resources):

	A	B	C	D
P1	3	3	2	2
P2	1	2	3	4
P3	1	1	5	0

Can the following request be served by the Resource allocation state? 1.
P3(0,0,3,0)
2. P2(0,1,0,0)

Safe sequence is not unique. A different safe sequence may also be possible. 1.7

Advantage: Avoids deadlock and it is less restrictive than deadlock prevention.

Disadvantage: Only works with fixed number of resources and processes.

- Guarantees finite time - **not** reasonable response time
- Needs advanced knowledge of maximum needs
- Not suitable for multi-access systems
- Unnecessary delays in avoiding unsafe states which may not lead to deadlock.

CONCLUSION:

Thus we learned to implement bankers algorithm.

CODE:

BankersImplementation.java:

```
/*Problem Statement: Write a JAVA program  to implement Banker's Algorithm  */

import java.util.Scanner;
public class Bankers{
   private int need[][],allocate[][],max[][],avail[][],np,nr;

   private void input(){
    Scanner sc=new Scanner(System.in);
    System.out.print("Enter no. of processes: ");
    np=sc.nextInt();  //no. of process
    System.out.print("Enter no. of Resources : ");
    nr=sc.nextInt();  //no. of resources
    need=new int[np][nr];  //initializing arrays
    max=new int[np][nr];
    allocate=new int[np][nr];
    avail=new int[1][nr];

    System.out.println("Enter allocation matrix -->");
    for(int i=0;i<np;i++)
        for(int j=0;j<nr;j++)
       allocate[i][j]=sc.nextInt();  //allocation matrix

    System.out.println("Enter max matrix -->");
    for(int i=0;i<np;i++)
        for(int j=0;j<nr;j++)
       max[i][j]=sc.nextInt();  //max matrix

      System.out.println("Enter available matrix -->");
      for(int j=0;j<nr;j++)
       avail[0][j]=sc.nextInt();  //available matrix

      sc.close();
```

```
  }

  private int[][] calc_need(){
    for(int i=0;i<np;i++)
     for(int j=0;j<nr;j++)  //calculating need matrix
      need[i][j]=max[i][j]-allocate[i][j];

    return need;
  }

  private boolean check(int i){
    //checking if all resources for ith process can be allocated
    for(int j=0;j<nr;j++)
    if(avail[0][j]<need[i][j])
      return false;

  return true;
  }

  public void isSafe(){
    input();
    calc_need();
    boolean done[]=new boolean[np];
    int j=0;

    while(j<np)
   {
//until all process allocated
    boolean allocated=false;
    for(int i=0;i<np;i++)
     if(!done[i] && check(i)){
 //trying to allocate
        for(int k=0;k<nr;k++)
        avail[0][k]=avail[0][k]-need[i][k]+max[i][k];
      System.out.println("Allocated process : "+i);
      allocated=done[i]=true;
          j++;
        }
      if(!allocated) break;  //if no allocation
    }
    if(j==np)  //if all processes are allocated
     System.out.println("\nSafely allocated");
    else
     System.out.println("All proceess cant be allocated safely");
  }

  public static void main(String[] args) {
    new Bankers().isSafe();
  }
}


/*
OUTPUT:
Enter no. of processes: 5
Enter no. of Resources : 3
```

```
Enter allocation matrix -->
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter max matrix -->
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter available matrix -->
3 3 2
Allocated process : 1
Allocated process : 3
Allocated process : 4
Allocated process : 0
Allocated process : 2

Safely allocated

*/
```