JUNAID GIRKAR
60004190057
SE COMPS A-3

OPERATING SYSTEM
EXPERIMENT - 7
CODE & OUTPUT

AIM: Implementation of Bankers Algorithm

CODE:

```
import java.util.*;
class Bankers
{
    static int P=5;
    static int R=3;
    static void printMatrix(int m[][])
    {
        for(int i=0;i<P;i++)
        {
            for(int j=0;j<R;j++)
            {
                System.out.print(m[i][j] +" ");
            }
            System.out.println();
        }
    }
    static void calculateNeed(int need[][], int max[][],int alloc[][])
    {
        for (int i = 0 ; i < P ; i++)
        {
            for (int j = 0 ; j < R ; j++)
            {
                need[i][j] = max[i][j] - alloc[i][j];
            }
        }
        System.out.println("\nNeed Matrix :");
```

```
        printMatrix(need);
    }
    static void isSafe(int processes[], int avail[], int max[][],int
alloc[][])
    {
        int [][]need = new int[P][R];
        calculateNeed(need, max, alloc);
        boolean []finish = new boolean[P];
        int []safeSeq = new int[P];
        int []work = new int[R];
        for (int i = 0; i < R ; i++)
        {
            work[i] = avail[i];
        }
        int count = 0;
        while (count < P)
        {
            boolean found = false;
            for (int p = 0; p < P; p++)
            {
                if (finish[p] == false)
                {
                    int j;
                    for (j = 0; j < R; j++)
                    {
                        if (need[p][j] > work[j])
                            break;
                    }
                    if (j == R)
                    {
                        for (int k = 0 ; k < R ; k++)
                        {
                            work[k] += alloc[p][k];
                        }
                        safeSeq[count++] = processes[p];
                        finish[p] = true;
                        found = true;
                    }
                }
            }
            if (found == false)
            {
                System.out.print("\nSystem is not in safe state. Deadlock
can occur.");
                return;
            }
        }
        System.out.print("\nSystem is in safe state.\nSafe sequence is: ");
```

```
        for (int i = 0; i < P-1 ; i++)
        {
            System.out.print("P" + safeSeq[i] + "->");
        }
        System.out.print(safeSeq[P-1]);
    }
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the no of processes : ");
        P = sc.nextInt();
        int processes[] = new int [P];
        for(int i=0;i<P;i++)
        {
            processes[i]=i+1;
        }
        System.out.println("Enter total no of resources : ");
        R = sc.nextInt();
        int totalResources[] = new int[R];
        System.out.println("Enter total no of resources present : ");
        for(int i=0;i<R;i++)
        {
            totalResources[i]=sc.nextInt();
        }
        int avail[] = new int[R];
        int max[][] = new int[P][R];
        int alloc[][] = new int[P][R];
        System.out.println("Enter Maximum need Matrix :");
        for(int i=0;i<P;i++)
        {
            for(int j=0;j<R;j++)
            {
                max[i][j]=sc.nextInt();
            }
        }
        System.out.println("Enter Allocation Matrix :");
        for(int i=0;i<P;i++)
        {
            for(int j=0;j<R;j++)
            {
                alloc[i][j]=sc.nextInt();
            }
        }
        for(int i=0;i<R;i++)
        {
            avail[i]=0;
            for(int j=0;j<P;j++)
            {
```

```
                avail[i]=avail[i] + alloc[j][i];
            }
            avail[i]=totalResources[i]-avail[i];
        }
        System.out.println("Available Matrix :");
        for(int i=0;i<R;i++)
        {
            System.out.print(avail[i] + " ");
        }
        System.out.println("\nMaximum Need Matrix :");
        printMatrix(max);
        System.out.println("\nAllocated Matrix :");
        printMatrix(alloc);
        isSafe(processes, avail, max, alloc);
        sc.close();
    }
}
```

OUTPUT:

```
Enter the no of processes :
5
Enter total no of resources :
3
Enter total no of resources present :
10
5
7
Enter Maximum need Matrix :
7 5 3
3 2 2
9 0 2
4 2 2
5 3 3
Enter Allocation Matrix :
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Available Matrix :
3 3 2
Maximum Need Matrix :
7 5 3
3 2 2
9 0 2
4 2 2
5 3 3

Allocated Matrix :
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Need Matrix :
7 4 3
1 2 2
6 0 0
2 1 1
5 3 1

System is in safe state.
Safe sequence is: P2->P4->P5->P1->3
```