9. Bankers Algorithm

Aim :- Bankers Algorithm for deadlock detection and avoidance

Problem Statement :- write a Java program to implement Banker's Algorithm.

Theory :-

Following Data structures are used to implement the Banker's Algorithm :-

1. Available :-
2. Max
3. Allocation
4. Need

Need [i,j] = Max [i,j] - Allocation [i,j]

* Safety Algorithm :-

1) Let Work and Finish be vectors of length 'm' & 'n' respectively.
Initialize : Work = Available
Finish [i] = false; for i = 1, 2, 3, 4 .... n

2) Find an i such that both
   a) finish [i] = false
   b) Need$_i$ <= Work
   if no such i exists goto step (4)

3) Work = Work + Allocation [i]
   Finish [i] = true
   goto step (2)

4) if finish [i] = true for all i
   then the system is in a safe state.

question 1: What will be content of Need matrix

- Need [i, j] = Max [i, j] - Allocation [i, j]

Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Assignment No. 09

Problem Satement: Write a Java program to implement Banker's Algorithm

1. Banker's Algorithm Program:

```
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 and resources : ");
np=sc.nextInt(); //no. of process  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 and resources : 4
3
Enter allocation matrix -->
0
1
0
2
0
0
3
0
2
2
1
1
Enter max matrix -->
7
5
3
3
2
2
9
0
2
2
2
2
Enter available matrix -->
3
3
2
Allocated process : 1
Allocated process : 3
Allocated process : 0
Allocated process : 2

Safely allocated
```