```
// Java program to illustrate Banker's Algorithm
import java.util.*;

class bankers
{

// Number of processes
static int P = 5;

// Number of resources
static int R = 3;

// Function to find the need of each process
static void calculateNeed(int need[][], int maxm[][],
                                    int allot[][])
{
        // Calculating Need of each P
        for (int i = 0 ; i < P ; i++)
                for (int j = 0 ; j < R ; j++)

                        // Need of instance = maxm instance -
                        //                                 allocated instance
                        need[i][j] = maxm[i][j] - allot[i][j];
}

// Function to find the system is in safe state or not
static boolean isSafe(int processes[], int avail[], int maxm[][],
                        int allot[][])
{
        int [][]need = new int[P][R];

        // Function to calculate need matrix
        calculateNeed(need, maxm, allot);

        // Mark all processes as infinish
        boolean []finish = new boolean[P];

        // To store safe sequence
        int []safeSeq = new int[P];

        // Make a copy of available resources
        int []work = new int[R];
        for (int i = 0; i < R ; i++)
                work[i] = avail[i];

        // While all processes are not finished
        // or system is not in safe state.
        int count = 0;
        while (count < P)
```

```
{
        // Find a process which is not finish and
        // whose needs can be satisfied with current
        // work[] resources.
        boolean found = false;
        for (int p = 0; p < P; p++)
        {
                // First check if a process is finished,
                // if no, go for next condition
                if (finish[p] == false)
                {
                        // Check if for all resources of
                        // current P need is less
                        // than work
                        int j;
                        for (j = 0; j < R; j++)
                                if (need[p][j] > work[j])
                                        break;

                        // If all needs of p were satisfied.
                        if (j == R)
                        {
                                // Add the allocated resources of
                                // current P to the available/work
                                // resources i.e.free the resources
                                for (int k = 0 ; k < R ; k++)
                                        work[k] += allot[p][k];

                                // Add this process to safe sequence.
                                safeSeq[count++] = p;

                                // Mark this p as finished
                                finish[p] = true;

                                found = true;
                        }
                }
        }

        // If we could not find a next process in safe
        // sequence.
        if (found == false)
        {
                System.out.print("System is not in safe state");
                return false;
        }
}

// If system is in safe state then
```

```
        // safe sequence will be as below
        System.out.print("System is in safe state.\nSafe"
                +" sequence is: ");
        for (int i = 0; i < P ; i++)
                System.out.print(safeSeq[i] + " ");

        return true;
}

// Driver code
public static void main(String[] args)
{
        int processes[] = {0, 1, 2, 3, 4};

        // Available instances of resources
        int avail[] = {3, 3, 2};

        // Maximum R that can be allocated
        // to processes
        int maxm[][] = {{7, 5, 3},
                                        {3, 2, 2},
                                        {9, 0, 2},
                                        {2, 2, 2},
                                        {4, 3, 3}};

        // Resources allocated to processes
        int allot[][] = {{0, 1, 0},
                                        {2, 0, 0},
                                        {3, 0, 2},
                                        {2, 1, 1},
                                        {0, 0, 2}};

        // Check system is in safe state or not
        isSafe(processes, avail, maxm, allot);
}
}
```

Output :

```
ubuntu@RE-LAB:~/Documents$ java bankers.java
System is in safe state.
Safe sequence is: 1 3 4 0 2 ubuntu@RE-LAB:~/Documents$
```