```
// Banker's Algorithm/DeadlockAvoidance algo

import java.util.Scanner;

public class BankerAlgo {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);


        System.out.print("Enter the number of processes: ");
        int processCount = scanner.nextInt();
        System.out.print("Enter the number of resources: ");
        int resourceCount = scanner.nextInt();


        int max[][]  = new int[processCount][resourceCount];
        int allocation [][]  = new int[processCount][resourceCount];
        int need[][]  = new int[processCount][resourceCount];
        int available [] = new int[resourceCount];


        System.out.println("Enter the maximum resource matrix:");
        for (int i = 0; i < processCount; i++) {
            System.out.print("Process " + i + ": ");
            for (int j = 0; j < resourceCount; j++) {
                max[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Enter the allocation matrix:");
        for (int i = 0; i < processCount; i++) {
            System.out.print("Process " + i + ": ");
            for (int j = 0; j < resourceCount; j++) {
                allocation[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Enter the available resources:");
        for (int j = 0; j < resourceCount; j++) {
            available[j] = scanner.nextInt();
        }


        for (int i = 0; i < processCount; i++) {
            for (int j = 0; j < resourceCount; j++) {
                need[i][j] = max[i][j] - allocation[i][j];
            }
        }


        System.out.println("Need Matrix:");
        for (int i = 0; i < processCount; i++) {
            for (int j = 0; j < resourceCount; j++) {
                System.out.print(need[i][j] + " ");
            }
            System.out.println();
```

```
        }

        // Display Available Resources
        System.out.println("Available Resources:");
        for (int j = 0; j < resourceCount; j++) {
            System.out.print(available[j] + " ");
        }
        System.out.println();

        // Find a safe sequence and update the Available Matrix
        boolean[] finish = new boolean[processCount];
        int[] work = available.clone();
        int[] safeSequence = new int[processCount];
        int count = 0;

        while (count < processCount) {
            boolean progressMade = false;
            for (int i = 0; i < processCount; i++) {
                if (!finish[i]) {
                    int j;
                    for (j = 0; j < resourceCount; j++) {
                        if (need[i][j] > work[j]) {
                            break;
                        }
                    }
                    if (j == resourceCount) {  // If all resources can be
allocated
                        // Add allocation to available
                        for (int k = 0; k < resourceCount; k++) {
                            work[k] += allocation[i][k];
                        }
                        finish[i] = true;
                        safeSequence[count++] = i;
                        progressMade = true;
                    }
                }
            }
            if (!progressMade) {
                System.out.println("System is in an unsafe state.");
                return;
            }
        }

        // Display Safe Sequence
        System.out.println("Safe Sequence:");
        for (int i = 0; i < processCount; i++) {
            System.out.print("P" + safeSequence[i] + " ");
        }
        System.out.println();

        // Display New Available Resources
        System.out.println("New Available Resources:");
        for (int j = 0; j < resourceCount; j++) {
            System.out.print(work[j] + " ");
        }
        System.out.println();

        scanner.close();
```

```
    }
}

//output-

Enter the number of processes: 5
Enter the number of resources: 3
Enter the maximum resource matrix:
Process 0: 7 5 3
Process 1: 3 2 2
Process 2: 9 0 2
Process 3: 2 2 2
Process 4: 4 3 3
Enter the allocation matrix:
Process 0: 0 1 0
Process 1: 2 0 0
Process 2: 3 0 2
Process 3: 2 1 1
Process 4: 0 0 2
Enter the available resources:
3 3 2
Need Matrix:
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1
Available Resources:
3 3 2
Safe Sequence:
P1 P3 P4 P0 P2
New Available Resources:
10 5 7
```