**Code**

Bankers.java

import java.util.\*;

import java.io.\*;

import java.util.Scanner;

// create Bankers class to implement Banker's algorithm in Java

class Bankers

{

// create findNeedValue() method to calculate the need of each process

static void findNeedValue(int needArray[][], int maxArray[][], int allocationArray[][], int totalProcess, int totalResources)

{

// use nested for loop to calculate Need for each process

for (int i = 0 ; i < totalProcess ; i++){ // for each process

for (int j = 0 ; j < totalResources ; j++){ //for each resource

needArray[i][j] = maxArray[i][j] - allocationArray[i][j];

}

}

}

// create checkSafeSystem() method to determine whether the system is in safe state or not

static boolean checkSafeSystem(int processes[], int availableArray[], int maxArray[][], int allocationArray[][], int totalProcess, int totalResources)

{

int [][]needArray = new int[totalProcess][totalResources];

// call findNeedValue() method to calculate needArray

findNeedValue(needArray, maxArray, allocationArray, totalProcess, totalResources);

// all the process should be infinished in starting

boolean []finishProcesses = new boolean[totalProcess];

// initialize safeSequenceArray that store safe sequenced

int []safeSequenceArray = new int[totalProcess];

// initialize workArray as a copy of the available resources

int []workArray = new int[totalResources];

for (int i = 0; i < totalResources ; i++) //use for loop to copy each available resource in the workArray

workArray[i] = availableArray[i];

// initialize counter variable whose value will be 0 when the system is not in the safe state or when all the processes are not finished.

int counter = 0;

// use loop to iterate the statements until all the processes are not finished

while (counter < totalProcess)

{

// find infinished process which needs can be satisfied with the current work resource.

boolean foundSafeSystem = false;

for (int m = 0; m < totalProcess; m++)

{

if (finishProcesses[m] == false) // when process is not finished

{

int j;

//use for loop to check whether the need of each process for all the resources is less than the work

for (j = 0; j < totalResources; j++)

if (needArray[m][j] > workArray[j]) //check need of current resource for current process with work

break;

// the value of J and totalResources will be equal when all the needs of current process are satisfied

if (j == totalResources)

{

for (int k = 0 ; k < totalResources ; k++)

workArray[k] += allocationArray[m][k];

// add current process in the safeSequenceArray

safeSequenceArray[counter++] = m;

// make this process finished

finishProcesses[m] = true;

foundSafeSystem = true;

}

}

}

// the system will not be in the safe state when the value of the foundSafeSystem is false

if (foundSafeSystem == false)

{

System.out.print("The system is not in the safe state because lack of resources");

return false;

}

}

// print the safe sequence

System.out.print("The system is in safe sequence and the sequence is as follows: ");

for (int i = 0; i < totalProcess ; i++)

System.out.print("P"+safeSequenceArray[i] + " ");

return true;

}

// main() method start

public static void main(String[] args)

{

int numberOfProcesses, numberOfResources;

//create scanner class object to get input from user

Scanner sc = new Scanner(System.in);

// get total number of resources from the user

System.out.println("Enter total number of processes");

numberOfProcesses = sc.nextInt();

// get total number of resources from the user

System.out.println("Enter total number of resources");

numberOfResources = sc.nextInt();

int processes[] = new int[numberOfProcesses];

for(int i = 0; i < numberOfProcesses; i++){

processes[i] = i;

}

int availableArray[] = new int[numberOfResources];

for( int i = 0; i < numberOfResources; i++){

System.out.println("Enter the availability of resource"+ i +": ");

availableArray[i] = sc.nextInt();

}

int maxArray[][] = new int[numberOfProcesses][numberOfResources];

for( int i = 0; i < numberOfProcesses; i++){

for( int j = 0; j < numberOfResources; j++){

System.out.println("Enter the maximum resource"+ j +" that can be allocated to process"+ i +": ");

maxArray[i][j] = sc.nextInt();

}

}

int allocationArray[][] = new int[numberOfProcesses][numberOfResources];

for( int i = 0; i < numberOfProcesses; i++){

for( int j = 0; j < numberOfResources; j++){

System.out.println("How many instances of resource"+ j +" are allocated to process"+ i +"? ");

allocationArray[i][j] = sc.nextInt();

}

}

//call checkSafeSystem() method to check whether the system is in safe state or not

checkSafeSystem(processes, availableArray, maxArray, allocationArray, numberOfProcesses, numberOfResources);

}

}

**Output**



