```
// ALGORITMO BANQUEIRO EXERCICIO
import java.util.Scanner;

public class BankerAlgorithm {
    private int numProcesses;  // número de processos
    private int numResources;  // número de recursos
    private int[][] allocation;  // matriz de alocação
    private int[][] max;  // matriz de máxima necessidade
    private int[] available;  // vetor de recursos disponíveis
    private boolean[] isSafe;  // vetor que indica se um processo é seguro

    public void initializeValues() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite o número de processos: ");
        numProcesses = scanner.nextInt();
        System.out.print("Digite o número de recursos: ");
        numResources = scanner.nextInt();
        allocation = new int[numProcesses][numResources];
        max = new int[numProcesses][numResources];
        available = new int[numResources];
        isSafe = new boolean[numProcesses];
        System.out.println("Digite a matriz de alocação:");
        readMatrix(scanner, allocation);
        System.out.println("Digite a matriz de máxima necessidade:");
        readMatrix(scanner, max);
        System.out.println("Digite o vetor de recursos disponíveis:");
        readVector(scanner, available);
    }

    public void calculateNeed(int[][] need) {
        for (int i = 0; i < numProcesses; i++) {
            for (int j = 0; j < numResources; j++) {
                need[i][j] = max[i][j] - allocation[i][j];
            }
        }
    }

    public void execute() {
        int[][] need = new int[numProcesses][numResources];
        calculateNeed(need);
        int[] work = available.clone();
        int count = 0;
        while (count < numProcesses) {
            boolean found = false;
            for (int i = 0; i < numProcesses; i++) {
                if (!isSafe[i]) {
                    boolean canExecute = true;
                    for (int j = 0; j < numResources; j++) {
                        if (need[i][j] > work[j]) {
                            canExecute = false;
                            break;
                        }
```

```
                }
                if (canExecute) {
                    isSafe[i] = true;
                    found = true;
                    count++;
                    for (int j = 0; j < numResources; j++) {
                        work[j] += allocation[i][j];
                    }
                }
            }
        }
        if (!found) {
            break;
        }
    }
    if (count < numProcesses) {
        System.out.println("O sistema está inseguro!");
    } else {
        System.out.println("O sistema está seguro!");
    }
}

public void readMatrix(Scanner scanner, int[][] matrix) {
    for (int i = 0; i < numProcesses; i++) {
        for (int j = 0; j < numResources; j++) {
            matrix[i][j] = scanner.nextInt();
        }
    }
}

public void readVector(Scanner scanner, int[] vector) {
    for (int i = 0; i < numResources; i++) {
        vector[i] = scanner.nextInt();
    }
}

public static void main(String[] args) {
    BankerAlgorithm banker = new BankerAlgorithm();
    banker.initializeValues();
    banker.execute();
}
}
```

QUESTÕES
Sobre o exercício acima, responda as seguintes questões:
Observações:
(i) em algum momento pode ser necessário “popular” uma estrutura de dados
(ii) Eventualmente, para responder as questões a seguir, podem ser necessárias outras fontes de consulta.
1)O que é a matriz de alocação utilizada pelo algoritmo do Banqueiro?
R. A matriz de alocação no algoritmo do banqueiro é uma matriz que rastreia a quantidade de recursos de cada tipo que estão atualmente alocados a cada processo no sistema. Ela é fundamental para determinar se uma solicitação de recursos pode ser concedida de forma segura, evitando impasses.

2)Como é calculada a matriz de necessidade no algoritmo do Banqueiro?
R. A matriz de necessidade no algoritmo do banqueiro é calculada subtraindo-se a matriz de alocação da matriz de máxima demanda. É usada para representar a quantidade de recursos de cada tipo que um processo ainda precisa para concluir sua execução.

3) Como é determinado se o sistema está seguro ou inseguro pelo algoritmo do Banqueiro? (Talvez seja necessária uma reflexão sobre o que é “sistema seguro” e “sistema inseguro”).
R. O sistema está seguro se houver uma sequência de processos na qual todos possam ser executados sem causar impasses, com base nas condições descritas acima. Caso contrário, o sistema é considerado inseguro, o que significa que as alocações atuais de recursos podem levar a impasses.

4)O que acontece se o algoritmo do Banqueiro identificar que o sistema está inseguro?
R. Se o algoritmo do Banqueiro identificar que o sistema está inseguro, isso significa que não é possível conceder as solicitações de recursos dos processos sem colocar o sistema em um estado de impasse.

5)O que é a dimensão de necessidade (need) e como é calculada no código?
R. Dimensão de necessidade geralmente se refere ao número de tipos diferentes de recursos em um sistema que utiliza o algoritmo do banqueiro para evitar impasses

6) O algoritmo apresentado em sala foi pensado para a alocação de recursos estáticos, ou seja, o desenvolvedor, a priori teria como saber a quantidade de recursos tais como memória RAM, espaço em disco, núcleos de processamento estão disponíveis. Por outro lado, em ambientes de cloud computing, tal alocação pode ser feita de forma dinâmica, ou seja, de acordo com as necessidades. Realize uma reflexão sobre essa situação e responda se o Algoritmo do Banqueiro continuaria válido, não teria necessidade alguma ou seria ainda mais necessário neste contexto.
R. Acredito que o algoritmo do banqueiro seja muito útil em diversos cenários. No entanto, há outras formas atualmente de se lidar com questões relacionadas a empasses causados por deadlocks. Ou seja, varia de acordo com o contexto.