Projeto 2

Matrizes

Unidade Curricular de
Laboratório de Programação

2020/2021

Objetivos

- Leitura de dados de um ficheiro;
- Utilizar *arrays* bidimensionais para implementar o conceito de matriz.

Antes de Começar

Criar um projeto java e importar os ficheiros

- Fazer *download* do ficheiro `alunosProjeto2.zip` na página de LabP;
  Este ficheiro contém um ficheiro `TestsMatrixOperation.java`, com os testes para as operações sobre matrizes, e ficheiros de dados para exemplo;
- No Eclipse,
  - Criar um projeto Java escolhendo **File** > **New** > **Project** > **Java Project**
  - Importar para o projeto o ficheiro de testes para uma pasta própria (consultar o guião "Introdução ao IDE Eclipse") e os ficheiros de dados para a raiz do projeto.

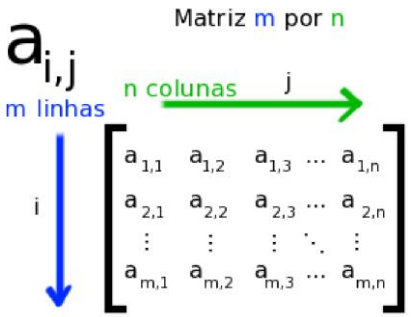
Algumas informações úteis

De modo a poder realizar este projeto deverá recordar como utilizar as classes que lhe permitem ler dados de um ficheiro de inteiros, nomeadamente a classe `Scanner`.

Poderá consultar a informação disponibilizada na disciplina de IP sobre *arrays*, (https://introcs.cs.princeton.edu/java/14array/), com foco especial em *arrays* bidimensionais.

Enunciado

O conceito matemático de *matriz* corresponde a uma tabela de **m** x **n** elementos representada sob a forma de um quadro com **m** linhas e **n** colunas. As matrizes são utilizadas, por exemplo, na resolução de sistemas de equações lineares, ou para realizar transformações geométricas em computação gráfica.

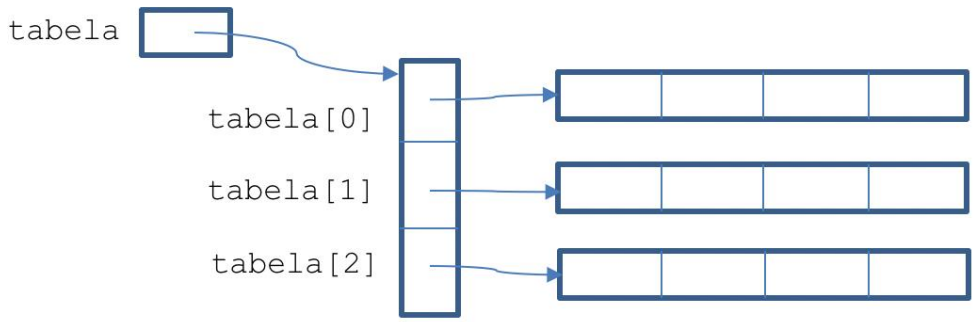


O objetivo deste trabalho é desenvolver uma classe `MatrixOperation` que contenha funções (métodos `static`) que efetuem operações elementares sobre matrizes de números inteiros.

Tendo em conta a definição anterior, uma matriz pode ser representada em Java por um *array* bidimensional. A seguinte declaração reserva espaço de memória para uma matriz de valores inteiros com 3 linhas e 4 colunas :

```
int[ ][ ] tabela = new int[3][4];
```

que podemos representar graficamente da seguinte forma



O primeiro índice do *array* bidimensional corresponde sempre ao índice da linha e o segundo índice, ao índice da coluna. O número de linhas obtém-se através de `tabela.length` e o número de elementos da linha `i` por `tabela[i].length` (para `i` no intervalo `[0, tabela.length -1]`).

Tenha em atenção que, ao fazer a seguinte declaração e atribuição

```
int[ ][ ] outra = tabela;
```

não se criou em memória um novo *array* igual ao original; está-se simplesmente a dar acesso, através da variável `outra`, ao *array* referenciado pela variável `tabela`. Por exemplo, a instrução

```
outra[1][2] = 20;
```

está a atribuir o valor 20 à posição `outra[1][2]` que é também a posição `tabela[1][2]`.

Nem todos os *arrays* bidimensionais são matrizes. Em java é possível definir *arrays* bidimensionais *irregulares* (em inglês *ragged arrays*), isto é, *arrays* cujas linhas não têm todas o mesmo comprimento.

Uma **matriz** é um *array* bidimensional *regular*, ou seja, em que todas as linhas têm o mesmo comprimento.

Neste projeto trabalhar-se-á apenas com matrizes. Além disso, usaremos **matrizes quadradas**, que são matrizes em que o número de linhas é igual ao número de colunas.

O que fazer então?

Deve desenvolver duas classes:

- a classe `MatrixOperation` cujos métodos implementam operações sobre matrizes e
- a classe `RunMatrix` que é cliente da classe `MatrixOperation` e contém o método `main` que lê valores de ficheiros e interage com o utilizador.

Descrevem-se de seguida as duas classes.

A classe `MatrixOperation` deve implementar os métodos abaixo descritos. Pode incluir outros métodos que considere necessários, desde que sejam privados.

Vai reparar que, à exceção do método `isSquare`, todos os métodos têm como pré-condição que a matriz argumento é uma matriz quadrada. Deve incluir sempre as pré-condições na documentação dos métodos.

- `public static boolean isSquare (int[][] m)` que determina se o *array* bidimensional `m` é regular (todas as linhas têm o mesmo comprimento) e, além disso, se representa uma matriz quadrada. *Nota*: a matriz é quadrada se o número de linhas é igual ao número de colunas. O número de linhas e colunas é um inteiro positivo.

- `public static boolean isIdentity (int[][] m)` que, assumindo que `m` é matriz quadrada, verifica se `m` é a matriz identidade. *Nota*: numa matriz identidade os valores da diagonal principal, m_{ii}, são iguais a `1` e os restantes são zero.

- `public static int[][] diagonal (int[][] m)` que, assumindo que `m` é matriz quadrada, devolve a matriz diagonal de `m`. *Nota*: Dada uma matriz `m`, a sua matriz diagonal, m_d, é uma matriz com zeros em todas as posições exceto na diagonal principal, onde os valores são os mesmos da diagonal principal de `m`.

- `public static int[] diagonalVector (int[][] m)` que, assumindo que `m` é matriz quadrada, devolve a diagonal principal de `m` sob a forma de um *array* de inteiros.

- `public static int[][] transpose (int[][] m)` que, assumindo que `m` é matriz quadrada, devolve a matriz transposta de `m`. *Nota*: dada uma matriz `m`, a sua transposta, m^T, tem na posição m^T_{ij} o valor m_{ji}.

- `public static int sumCalculation (int[][] m, int index, int type)` que devolve o somatório da linha, da coluna, ou da diagonal principal de `m`, consoante o valor do parâmetro de entrada `type` :
  - `type` ==1: a função devolve a soma da linha `index` da matriz
  - `type` ==2: a função devolve a soma da coluna `index` da matriz
  - `type` ==3: a função devolve a soma da diagonal da matriz (o valor `index` é ignorado).

  Assume-se que os valores dos parâmetros respeitam: `m` é matriz quadrada, `type` tem um valor no intervalo [1,3] e `index` é maior ou igual a zero e menor que o número de linhas de `m` (no caso de `type` ser 1) ou menor que o número de colunas de `m` (no caso de `type` ser 2).

- `public static int[][] multiplyLine (int[][] m, int scalar, int index)` que, assumindo que `m` é matriz quadrada e `index` é maior ou igual a zero e menor que o número de linhas de `m`, devolve uma matriz em tudo igual a `m` exceto na linha de índice `index`; esta obtém-se multiplicando os elementos da linha de índice `index` de `m` pelo número inteiro `scalar`.

- `public static int[][] subtractLine (int[][] m, int index)` que, assumindo que `m` é matriz quadrada e `index` é maior ou igual a zero e menor que o número de linhas de `m`, devolve uma matriz em que todas as linhas, exceto a de índice `index` , se obtêm subtraindo das linhas correspondentes de `m` os valores da linha de índice `index` da matriz `m`.
  Exemplo: se as linhas de `m` forem `1 3 5 , 2 7 1 , 7 9 4`, e `index` for `0`, as linhas da matriz resultado serão `1 3 5 , 1 4 -4 , 6 6 -1`.

A classe `RunMatrix` é uma classe cliente da classe `MatrixOperation` e destina-se a executar os métodos implementados nesta classe. Tenha em atenção que antes de invocar cada método deve verificar se estão garantidas as suas pré-condições.

A matriz de dados é lida de um ficheiro, cujo formato é explicado mais à frente, e **os resultados da execução são apresentados na consola**.

O método `main` desta classe deve executar as seguintes operações invocando, sempre que possível, os métodos da classe `MatrixOperation`:

1. Ler uma matriz a partir do ficheiro de nome "`inputMatrix1.txt`". Chamemos-lhe `matrix1`.

- Se o ficheiro não existir, o programa não deverá terminar abruptamente: a exceção correspondente deverá ser tratada pelo programa de modo a que o utilizador seja devidamente informado do problema.
- Se o ficheiro não contiver uma matriz quadrada, deverá ser apresentada a mensagem de erro: "O ficheiro nao contem matriz quadrada" e terminar a execução.

2. Verificar se `matrix1` é a matriz identidade.
3. Escrever a matriz diagonal de `matrix1`.
4. Escrever o *array* diagonal de `matrix1`.
5. Escrever a matriz transposta de `matrix1`.
6. Escrever o somatório de cada uma das linhas, de cada uma das colunas e da diagonal principal da matriz `matrix1` (por esta ordem).
7. Pedir ao utilizador o índice de uma linha da matriz (chamemos-lhe `index`) e um inteiro (chamemos-lhe `scalar`) para multiplicar por essa linha.
   Se o valor de `index` não respeitar as pré-condições do método `multiplyLine`, deve voltar a pedir um valor.
8. Escrever a matriz que se obtém multiplicando a linha com índice `index` da matriz `matrix1` pelo valor inteiro `scalar` fornecido pelo utilizador.
9. Verificar qual a linha de `matrix1` cujo somatório é maior. Chamemos `iMaxSum` ao índice dessa linha;
10. Determinar o maior valor da linha de índice `iMaxSum` de `matrix1` . Chamemos `max` a esse valor;
11. Obter a matriz que resulta de multiplicar a linha de índice `iMaxSum` de `matrix1` por um inteiro. Chamemos `scalar` a esse valor:

    Este inteiro `scalar` deve ser tal que transforma `max` no menor inteiro possível maior ou igual a 10. Se `max` for 0, então `scalar` deverá ser 1;

    Exemplo: se `max` for 4, `scalar` deve ser 3 pois `max x 3` é 12; se `scalar` fosse 2, `max x 2` daria um valor < 10; se `scalar` fosse 4 daria 16 que, embora maior que 10, não é o menor inteiro maior que 10.
12. A partir desta nova matriz, obter a matriz que resulta de subtrair a linha de índice `iMaxSum` das restantes linhas.
13. Escrever esta última matriz e os valores de `iMaxSum`, `max` e `scalar`.

Experimente o seu método `main` fazendo a leitura da matriz inicial a partir de outros ficheiros de texto, fornecidos por nós ou criados por si.

Ficheiro de dados

Assuma que uma matriz é representada por um bloco de linhas, que se inicia com uma linha onde está escrito o número de linhas e o número de colunas da matriz, e em que cada uma das restantes linhas corresponde a uma linha da matriz.

Exemplo de ficheiro de dados:

```
3 3
4 0 0
0 1 0
2 -3 1
```

Para este ficheiro de dados, o resultado da execução das operações solicitadas para a classe `RunMatrix` deverá ser análogo ao seguinte:

```
Matriz de entrada
4 0 0
0 1 0
2 -3 1
Nao eh matriz identidade
Matriz diagonal
4 0 0
0 1 0
0 0 1
Diagonal principal
4 1 1
Matriz transposta
4 0 2
0 1 -3
0 0 1
Soma dos elementos de cada linha: 4 1 0
Soma dos elementos de cada coluna: 6 -2 1
Soma dos elementos da diagonal: 6
A linha com indice 0 tem o valor maximo da soma das linhas
O maximo da linha com indice 0 eh 4
Matriz apos multiplicacao da linha com indice 0 por 3
e restantes linhas a que foi subtraida a linha com indice 0:
12 0 0
-12 1 0
-10 -3 1
```

O que entregar

Deve criar o ficheiro `P2fcxxxxx.zip`, onde `xxxxx` é o seu número de aluno, contendo os ficheiros:

`MatrixOperation.java` e `RunMatrix.java`

ATENÇÃO: Antes de submeter o trabalho, verifique que documentou o seu projeto (incluindo `@author` com o seu número de aluno).

Importante

O facto das vossas classes passarem nos testes fornecidos não significa que a classe esteja 100% correta. Há testes e verificações que não são feitas de propósito, de modo a incentivar os alunos a irem à procura de pontos de eventuais falhas no código. Além disso, os pesos para a avaliação dados a cada um dos testes pode ser diferente.