Exercício:

Com base nos conceitos estudados, implemente uma classe que possibilite a instanciação de matrizes de números naturais, com qualquer quantidade de linhas e colunas. Especifique os membros de dados que julgar necessário, para que a interface da classe contemple a possibilidade de inicializar os elementos das matrizes, e imprimi-las com layout adequado.

Linguagem de Programação C++

O que acontecerá se um programa *driver* para a classe Matriz efetuasse a seguinte sequência de instruções:

```
#include "Matriz.h"
...
int main()
{
  char opcao;
  ...
  switch (opcao)
  {
    case 1:
    {
      int l, c;
      cout << endl << "Entre com o numero de linhas da matriz: ";
      cin >> l;
      cout << endl << "Entre com o numero de colunas da matriz: ";
      cin >> c;
      Matriz m(l, c);
      ...
    }
    ...
  }
  ...
}
```

Destrutor

Para resolvermos este problema, existe, em cada classe, uma função membro especial, denominada Destrutor, que permitem providenciar a desalocação controlada dos objetos instanciados.

Estas funções membros são nomeadas na forma ~<nome_da_classe> (nome da classe prefixado com til).

Exemplo:

```
class Matriz
{
   ...
   public:
      ~Matriz();
}
```

Sendo assim, uma função-membro destrutor, poderá tomar as providências necessárias para a desalocação adequada do objeto receptor.

Por exemplo: desalocar nodo a nodo uma lista encadeada ou um vetor alocado dinamicamente.

É muito relevante frisar que um destrutor **não recebe parâmetro** e **nem retorna um valor**. Não sendo possível a especificação de nenhum tipo de retorno, nem mesmo o void. Cada classe possui **apenas um destrutor**.

Uma classe sempre possui um destrutor, se o programador não fornecer um destrutor explicitamente, o compilador cria um destrutor "vazio", que desempenha um papel importante em objetos criados por herança e composição.

Observação: Construtoras e destrutoras não são herdadas!

Exercício:

Agora, com base no conceito anterior adapte sua solução para o exercício do slide 189 sobre a classe Matriz.

Linguagem de Programação C++

Exercício:

Com base no que vimos a respeito da representação do construtor de uma classe em um diagrama de classes em UML siga sua lógica e construa um diagrama de classe em UML que represente a classe Matriz.

Matriz
-elementos: int * -numeroDeLinhas: int -numeroDeColunas: int
+<<construtor>> Matriz(in int,in int) +<<destrutor>> ~Matriz() +setElemento(in int,in int,in int): void +getElemento(in int,in int): int +inicialize(): void +imprima(): void -setNumeroDeLinhas(in int): void -setNumeroDeColunas(in int): void -getNumeroDeLinhas(): int -getNumeroDeColunas(): int

Lembre-se que neste caso o diagrama de classes apresentado está direcionado para a linguagem C++ o que não é aconselhável.

Empacotador de pré-processador

Creio que alguns de vocês já devem ter se perguntado sobre a possibilidade de inserir, por acidente, a definição de uma classe mais de uma vez em um programa, pois em um programa grande ocorrem muitas inclusões de arquivos cabeçalhos que por sua vez podem incluir outros arquivos cabeçalhos.

Para evitar a ocorrência deste erro devemos utilizar o empacotador de pré-processador **#ifndef**. O qual representa 'se não definido'.

Sua sintaxe é:

```
#ifndef <ROTULO>
#define <ROTULO>
   ...
#endif
```

Para uma compreensão adequada analisaremos sua utilização na classe Matriz.

```
//conteúdo do arquivo Matriz.h
#ifndef MATRIZ_H
#define MATRIZ_H
  class Matriz
  {
    private:
      int *elementos;
      int numeroDeLinhas;
      int numeroDeColunas;
      void setNumeroDeLinhas(int);
      void setNumeroDeColunas(int);
      int getNumeroDeLinhas();
      int getNumeroDeColunas();
    public:
      Matriz(int, int);
      ~ Matriz();
      void setElemento(int, int, int);
      int getElemento(int, int);
      void inicialize();
      void imprima();
  };
#endif
```

Exercício:

Visando melhor fixar a utilização da alocação dinâmica de memória, explorando as funções-membros construtor e destrutor, explorar o princípio do ocultamento de implementação e demonstrar as vantagens na manutenibilidade de sistemas gerados com a utilização da OO, adapte a solução do exercício do slide 189 sobre a classe Matriz. Faça com que este armazene os elementos da matriz através de um membro de dados int ** ao invés de int * ou vise e versa para quem gerou inicialmente sua solução com int **. Utilize o empacotador de pré-processador.