

Capítulo 10 – Orientação ao Uso de Objetos

Introdução

Paralelo ao desenvolvimento da tecnologia associada aos computadores observa-se o contínuo esforço em se lidar com o aumento da complexidade dos programas. A evolução das linguagens de programação faz parte desse esforço.

As primeiras linguagens de montagem (*assembly*) já tinham por objetivo facilitar a tarefa de se lidar com componentes diversos e estabelecer relações funcionais entre eles. A primeira linguagem de programação de alto nível (FORTRAN), além disso, buscava favorecer a clareza, facilitar o entendimento e aumentar a eficiência dos programas.

Estilos de programação desenvolvem-se, igualmente, à medida que as linguagens evoluem.

A partir dos anos 60, estilos, como o da *programação estruturada*, passaram a influenciar o modo como se programar e o desenvolvimento de linguagens que os apoiassem. Algol, Pascal e C são exemplos de linguagens estruturadas. Esse estilo permite tratar o aumento da complexidade dos programas estabelecendo seqüências de etapas a serem cumpridas. A cada etapa elementos são selecionados e desenvolvidos, melhorando o conjunto de forma global e contínua (refinamento sucessivo).

Entretanto, a complexidade dos programas evolui com rapidez e com escala ainda maiores. Em meados dos anos 80, um outro estilo, o da *programação orientada a objetos*, começou a ganhar mais espaço. Procura aproveitar as melhores idéias da programação estruturada com outros conceitos associados às formas de definição e emprego dos elementos, facilitando sua decomposição em conjuntos mais estreitamente relacionados. Linguagens como Smalltalk e Eiffel são exemplos típicos desse tipo de estilo; mas há linguagens como C++ e Java que ainda guardam características da programação estruturada, mas procuram incorporar esses avanços traduzindo a definição da estrutura e comportamento comuns aos elementos de certo conjunto na forma de instâncias (objetos) de determinada classe.

Classes são gabaritos com os quais se podem definir objetos, descrever suas propriedades (atributos) e especificar o seu comportamento através de métodos que especificam que funcionalidades podem lhes ser aplicadas.

Dentre as vantagens desse estilo de programação pode-se citar:

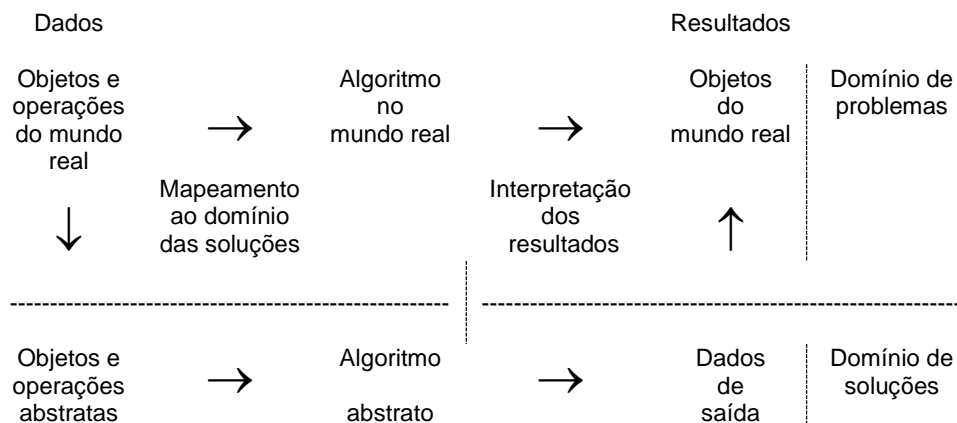
- o aumento na reutilização do código;
- a redução do custo de manutenção do código;
- o aumento da produtividade.

Três importantes conceitos estão ligados ao desenvolvimento de programas com orientação a objetos:

- *encapsulamento* que representa o princípio de projeto pelo qual cada componente deve agregar toda a informação necessária para a sua manipulação e guardar uma decisão de projeto tornando-a pública, ou não, se for preciso ou conveniente;
- *herança* que é um mecanismo pelo qual as características comuns podem ser agrupadas em uma descrição (*superclasse*) e essa ser usada para se definir outras (*subclasses*). O relacionamento entre subclasses e superclasse pode ser definido pela:
 - especificação na qual a superclasse descreve o que uma subclasse deve conter, mas não implementa qualquer funcionalidade (*interface*);
 - extensão em que cada subclasse pode acrescentar atributos ou métodos às definições originais da superclasse (*herança estrita*);
 - combinação de ambas herdando a interface, redefinindo ou introduzindo novos elementos (*herança polimórfica*);
- *polimorfismo* que é o princípio pelo qual as classes derivadas podem invocar métodos com comportamentos distintos que possuem a mesma identificação (assinatura) na classe original, especializados em cada uma (*overriding*), abstraídos de seus detalhes. Esse mecanismo é diferente da *sobrecarga* de métodos, no qual há diferentes listas de argumentos.

Orientação a objetos

O princípio básico é identificar os principais tipos de objetos requeridos com base em seu conhecimento sobre o domínio da aplicação, ou dos problemas (abstrações da aplicação) e o domínio da implementação, ou das soluções (abstrações de implementação). O esquema abaixo pretende ilustrar melhor.



Novos conceitos básicos sobre a abstração de dados associados à orientação a objeto foram introduzidos sobre o modo de relacionar dados.



Classes

Classes são descrições de um conjunto uniforme de propriedades (atributos) e comportamentos (métodos) comuns.

Classe
atributos
métodos

Subclasse::Superclasse (pacote)
atributos
métodos

Definição de classe

Forma geral:

```

classe <nome da classe>:<nome da superclasse> (<projeto>)
| <visibilidade>:
|   <tipo 1> <atributo 1>
|   ...
|   <tipo N> <lista de atributos>
| <visibilidade>:
|   <tipo 1> <método 1>
|   ...
|   <tipo N> <método N>
fim classe ! <nome da classe> !
  
```

Exemplo:

Definição de uma classe para tratar valor inteiro positivo:

Positivo
+ inteiro valor

- nome da classe
- atributo público

ou

```

classe Positivo
| público:
|   inteiro valor;
fim classe | Positivo !
  
```

Observações:

- Os atributos poderão ser:
 - (+) público
 - (-) privado
 - (#) protegido
- Uma instância dessa classe poderá fazer referência ao atributo usando um operador de pertinência (.):

objeto.valor

Instanciação

Forma geral:

<nome da classe> <nome do objeto>

ou para se instanciar um objeto segundo uma referência padrão (construtor),
que sempre deverá ter o mesmo nome da classe:

<nome da classe> <nome do objeto> \leftarrow <construtor padrão da classe> ()

ou para se instanciar uma referência para um objeto nulo de uma classe:

<nome da classe> <nome do objeto> \leftarrow ε (objeto nulo)

É conveniente que cada classe possua além de seus atributos e métodos:

- - seu próprio tratamento de erro;
- - sua descrição de referência padrão (construtor) para ser instanciada.

Exemplo:

Positivo	
+ inteiro valor	- nome da classe
+ inteiro erro	- atributo público
+ Positivo ()	- atributo público
	- construtor (padrão)

A forma aperfeiçoada da classe será:

```

classe Positivo
| público:
| ! atributos !
|   inteiro valor;
|   inteiro erro ;
|
| ! métodos !
|   construtor Positivo ( )
|   | ! sem nenhuma especificação, ainda
|   fim construtor ! Positivo !
fim classe | Positivo !

```

Uma instância dessa classe poderá ser obtida da seguinte forma:

Exemplo:

Positivo x \leftarrow Positivo ();

x.valor \leftarrow 0;

! atribuir zero ao valor !

x.erro \leftarrow 0;

! indicar nenhum erro !

Sobrecarga de métodos

Permite-se a sobrecarga de métodos, desde que as listas de parâmetros sejam distintas (em quantidade e/ou tipos).

Exemplo:

Para garantir valores pré-definidos pode-se sobrecarregar construtores:

Positivo	
+ inteiro valor	- nome da classe
+ inteiro erro	- atributo público
	- atributo público
+ Positivo ()	- construtor (padrão)
+ Positivo (inteiro inicial)	- construtor alternativo com valor inicial
+ Positivo (Positivo inicial)	- construtor alternativo com outro Positivo

A nova forma da classe será:

```

classe Positivo
| público:
| ! atributos !
|   inteiro valor;
|   inteiro erro ;
|
| ! métodos !
|   construtor Positivo ( )
|   | ! sem nenhuma especificação, ainda !
|   fim construtor ! Positivo !
|
|   construtor Positivo ( inteiro inicial )
|   | valor ← inicial
|   | erro ← 0          ! nenhum erro !
|   fim construtor ! Positivo !
|
|   construtor Positivo ( Positivo inicial )
|   | valor ← inicial.valor      ! valor do outro Positivo !
|   | erro ← inicial.erro       ! erro do outro Positivo !
|   fim construtor ! Positivo !
|
fim classe | Positivo !

```



```

| procedimento Atribuir ( inteiro novo )
| | se ( novo > 0 )
| | | valor ← novo
| | | erro ← 0
| | | senão
| | | | valor ← 0      ! para não ficar vazio |
| | | | erro ← 1      ! para indicar existência de erro !
| | fim se
| fim procedimento ! Atribuir !

| procedimento Atribuir ( Positivo novo )
| | Atribuir ( novo.valor )
| fim procedimento ! Atribuir !

| inteiro função Extrair_Valor
| | retornar ( valor )
| fim função ! Extrair_Valor !

| inteiro função Extrair_Erro
| | retornar ( erro )
| fim função ! Extrair_Erro !

| fim classe | Positivo !

```

A qual permitirá definir objetos como os abaixo:

```

Positivo x1 = new Positivo ( );
Positivo x2 = new Positivo ( 3 );
Positivo x3 = new Positivo ( x2.Extrair_Valor ( ) );
Positivo x4 = new Positivo ( x3 );

```

Auto-referência

Se necessário, um objeto poderá fazer referências a si próprio.

```

| ! atributos !
| | construtor Positivo ( inteiro inicial )
| | | próprio.valor ← inicial      ! auto-referência ao seu atributo !
| | | próprio.erro ← 0            ! para dar ênfase ou tirar dúvida !
| | fim construtor ! Positivo !

| | construtor Positivo ( Positivo inicial )
| | | próprio.valor ← inicial.valor ! auto-referência para evitar dúvida !
| | | próprio.erro ← inicial.erro  ! ou dar ênfase aos seus atributos !
| | fim construtor ! Positivo !

```

ou

```

| ! métodos !
| | construtor Positivo ( )
| | | próprio.Positivo ( 0 )      ! auto-referência ao construtor alternativo !
| | fim construtor ! Positivo !

```



```

| ! métodos !
|
| inteiro função Extrair_Erro
| | retornar ( erro )
| fim função ! Extrair_Erro !
|
| lógico função Vazia
| | retornar ( livre = 0 )
| fim função ! Vazia !
|
| procedimento Inserir ( inteiro dado )
|
| | ! adicionar um valor à tabela !
| | se ( livre >= tamanho ) ! se espaço esgotado !
| | | erro ← 1 ! indicar que não há mais espaço !
| | senão ! se houver espaço !
| | | livre ← livre + 1 ! passar para a próxima posição !
| | | valor [ livre ] ← dado ! guardar valor !
| | | erro ← 0 ! indicar nenhum erro !
| | fim se
|
| fim procedimento ! Inserir !
|
| inteiro função Retirar
|
| | ! remover o primeiro valor da tabela !
| | ! definição de dado local !
| | inteiro dado ← 0 ! valor inicial, caso haja erro !
| | inteiro x
|
| | se ( Vazia ) ! se tabela vazia !
| | | erro ← 2 ! indicar que não há dado !
| | senão ! se houver dado !
| | | dado ← valor [ 1 ] ! separar o primeiro dado !
| | | repetir para ( x ← 1 : (livre-1) : 1 )
| | | | valor [ x ] ← valor [ x+1 ] ! redistribuir dados !
| | | fim repetir
| | | livre ← livre - 1 ! indicar um dado a menos !
| | | erro ← 0 ! indicar nenhum erro !
| | fim se
|
| | retornar ( dado )
|
| fim função ! Retirar !

```

```

| inteiro função Consultar ( inteiro posição )
|
| ! consultar valor em uma posição da tabela !
| ! definição de dado local !
|   inteiro dado ← 0           ! valor inicial, caso haja erro !
|
|   se ( Vazia )               ! se tabela vazia !
|   | erro ← 2                 ! indicar que não há dado !
|   senão                     ! se houver dado !
|   | se ( posição < 1 | posição > livre )
|   | | erro ← 3               ! indicar posição inválida !
|   | senão
|   | | dado ← valor [ posição ] ! copiar dado !
|   | | erro ← 0               ! indicar nenhum erro !
|   | fim se ! posição inválida !
|   fim se ! tabela vazia !
|
|   retornar ( dado )
|
fim função ! Consultar !

inteiro função Limite
|
| ! indicar a quantidade máxima de dados na tabela !
|   retornar ( 10 )
|
fim função ! Limite !

inteiro função Quantidade
|
| ! indicar a quantidade atual de dados na tabela !
|   retornar ( livre )
|
fim função ! Quantidade !

fim classe ! Tabela !

```

Exemplos de usos:

Teste 1:

```
! definir dados !
Tabela a ← Tabela ( 5 )
inteiro x
! adicionar dados !
tela ← ( "Tabela com capacidade para ", a.Limite, " dados." )
a.Inserir ( 1 )
se ( a.Extrair_Erro = 0 )
| a.Inserir ( 2 )
| se ( a.Extrair_Erro = 0 )
| | a.Inserir ( 3 )
| | se ( a.Extrair_Erro = 0 )
| | | tela ← ( "Tabela com ", a.Quantidade, " dados: " )
| | | repetir enquanto ( ~ a.Vazia )
| | | | x ← a.Retirar
| | | | tela ← x
| | | fim repetir
| | fim se
| fim se
fim se
```

Teste 2:

```
! definir dados !
Tabela a ← Tabela ( 5 )
inteiro x, y
! adicionar dados !
tela ← ( "Tabela com capacidade para ", a.Limite, " dados." )
y = 1
repetir enquanto ( y <= 3 & a.Extrair_Erro = 0 )
| a.Inserir ( y )
| y ← y + 1
fim repetir
! mostrar dados !
tela ← ( "Tabela com ", a.Quantidade, " dados: " )
repetir para ( x ← 1 : a.Quantidade : 1 )
| y ← a.Consultar ( x )
| se ( a.Extrair_Erro = 0 )
| | tela ← y
| fim se
fim repetir
```

Protótipo em C++:

```
// bibliotecas necessarias
#include <stdlib.h>
#include <string>
#include <iostream>

using namespace std;

/**
 * Tabela - modelo de classe para lidar com tabelas
 *
 */

typedef double Object;

/**
 * Tabela - modelo de classe para lidar com Tabelas
 *
 */

class Tabela
{
// tratamento de erro
private:
    int erro; // indicador de erro
    string m; // mensagem de erro

public:
    int getError ( );

// tratamento da Tabela
private:
// definir armazenadores
    int livre; // posicao livre
    int tamanho; // quantidade de dados
    Object valor[10]; // armazenador de dados

public:
// definir metodos publicos
    string saida;

    Tabela ( int quantidade );
    string toString ( );
    int empty ( );
    void push ( Object y );
    Object pop ( );
    Object peek ( int index );
    int length ( );
    int size ( );
}; // fim class Tabela
```

```

/**
 * getError ( ) - obter codigo do erro
 * <br>
 * @return - codigo do erro
 *         0 - nao ha' erro
 *         1 - posicao invalida
 */
int Tabela :: getError ( )
{
    return ( erro );
} // fim getError ( )

/**
 * Tabela ( ) - construtor do tipo Tabela
 */
Tabela :: Tabela ( int quantidade )
{
    // definir dados locais
    int x, y;

    // atribuir valores iniciais
    for ( x = 0; x < 10; x = x + 1 )
    {
        valor [ x ] = 0.0 ;
    }

    if ( quantidade <= 0 || quantidade > 10 )
    {
        tamanho = 0;
        erro = 1;           // valor invalido
    }
    else
    {
        tamanho = quantidade;
        erro = 0;           // nao ha' erro
    }
    livre = 0;
} // fim construtor padrao

/**
 * toString - converter conteudo para String
 * <br>
 * @return - indicar o codigo de erro atual
 */
string Tabela :: toString ( )
{
    // definir dado local
    int x;
    char buffer [20];
    saida = "";
    // coletar dados
    for ( x = 0; x < length( ); x = x + 1 )
    {
        saida = saida + "\n";
        sprintf ( buffer, "%t%f", valor [ x ] );
        saida = saida + buffer;
    } // fim for
    return ( saida );
} // fim toString ( )

```

```

/**
 * empty ( ) - informar se a tabela esta' vazia
 * <br>
 * @return resposta - se estiver vazia (true), ou nao (false)
 */
int Tabela :: empty ( )
{
    return ( livre == 0 );
} // fim empty ( )

/**
 * push( ) - adicionar um valor 'a Tabela
 * <br>
 * @param x      - posicao a ser alterada
 * @param dado - objeto a ser adicionado
 */
void Tabela :: push ( Object y )
{
    // adicionar um valor 'a Tabela
    if ( livre >= tamanho )
    {
        erro = 1;                // posicao invalida
    }
    else
    {
        valor [ livre ] = y;      // copiar dado para a posicao
        livre = livre + 1;
        erro = 0;                // nao ha' erro
    }
} // fim push ( )

/**
 * pop ( ) - obter valor da primeira posicao da tabela
 * <br>
 * @return - objeto obtido da primeira posicao, ou null
 */
Object Tabela :: pop ( )
{
    // definir dados locais
    int x;
    Object z = NULL;

    // obter um valor de uma posicao
    if ( livre == 0 )
    {
        erro = 1;                // posicao invalida
    }
    else
    {
        z = valor [ 0 ];
        for ( x = 0; x < (livre-1); x = x + 1 )
            valor [ x ] = valor [ x+1 ];
        livre = livre - 1;
        erro = 0;                // nao ha' erro
    } // fim se
    return ( z );
} // fim pop ( )

```

```

/**
 * peek ( ) - consultar valor da primeira posicao da tabela
 * <br>
 * @return - objeto copiado da primeira posicao, ou null
 */
Object Tabela :: peek ( int index )
{
    // definir dados locais
    Object z = NULL;

    // obter um valor da tabela
    if ( empty ( ) )           // tabela vazia
    {
        erro = 2;              // nao ha' elementos
    }
    else                       // ha' elementos
    {
        if ( index < 0 || index >= livre )
        {
            erro = 3;          // posicao invalida
        }
        else
        {
            z = valor [index]; // copiar o valor da posicao
        } // fim se ( posicao invalida )
    } // fim se ( tabela vazia )
    return ( z );              // retornar o elemento obtido
} // fim peek ( )

/**
 * length ( ) - informar a quantidade atual de dados na tabela
 * <br>
 * @return - numero de posicoes ocupadas na tabela
 */
int Tabela :: length ( )
{
    return ( livre );
} // fim length ( )

/**
 * size ( ) - informar a quantidade maxima de dados na tabela
 * <br>
 * @return - numero maximo de posicoes no vetor
 */
int Tabela :: size ( )
{
    return ( tamanho );
} // fim size ( )

```

```

/**
 * teste1 - testar colocacao de dados
 */
void teste1 ( )
{
    // definir dados
    Tabela a ( 5 );
    Object z;
    int x, y;

    // identificar
    cout << "Teste 1 - Colocar valores em posicao" ;
    cout << "\n" ;

    // adicionar dados
    cout << "Tabela com capacidade para " << a.size ( ) << " dados." ;
    a.push ( 1.0 );
    if ( a.getError ( ) == 0 )
    {
        a.push ( 2.0 );
        if ( a.getError ( ) == 0 )
        {
            a.push ( 3.0 );
            if ( a.getError ( ) == 0 )
            {
                // mostrar dados
                cout << "\n" ;
                cout << "\nDados no Tabela: " ;
                cout << "\n" ;

                // mostrar dados
                cout << "\n" ;
                cout << "\nTabela com " << a.length ( ) << " elementos:" ;
                cout << "\n" ;
                cout << a.toString ( ) ;

                // esvaziar a tabela
                cout << "\n" ;
                cout << "\nEsvaziar a tabela com " << a.length ( ) << " elementos:" ;
                cout << "\n" ;
                while ( ! a.empty ( ) )
                {
                    z = a.pop ( );
                    cout << "\n" << z ;
                } // fim repetir

                cout << "\n";
                cout << "\nTabela com " << a.length ( ) << " elementos." ;
                cout << "\n";

                } // fim se
            } // fim se
        } // fim se
    } // fim teste1
}

```



```

/**
 * teste2 - testar alteracao e consulta de dados
 */
void teste2 ( )
{
    // definir dados
    Tabela a ( 5 );
    Object z;
    int x, y;

    // identificar
    cout << "\nTeste 2 - Alterar e consultar valores" ;
    cout << "\n";

    // adicionar dados
    for ( x = 0; x < 3; x = x + 1 )
    {
        a.push ( x*0.1 );
    } // fim repetir

    // mostrar dados
    cout << "\n";
    cout << "\nTabela com " << a.length ( ) << " elementos:" ;
    cout << "\n";

    for ( y = 0; y < a.length ( ); y = y + 1 )
    {
        z = a.peek ( y );
        cout << "\n" << z ;
    } // fim repetir

    // esvaziar a tabela
    cout << "\n";
    cout << "\nEsvaziar a tabela com " << a.length ( ) << " elementos:" ;
    cout << "\n";

    x = a.length ( );
    y = 0;
    do
    {
        z = a.pop ( );
        cout << "\nElemento na posicao (" << y << ") igual a " << z ;
        y = y + 1;
    }
    while ( y < x );

    cout << "\n";
    cout << "\nTabela com " << a.length ( ) << " elementos." ;
    cout << "\n";

} // fim teste2

```

```

/**
 * main - testar Tabelas com objetos
 */
int main ( )
{
    // definir dados
    int opcao = 1;

    // repetir ate' parar
    do
    {
        // oferecer opcoes
        cout << "\n\nTeste de Tabelas" ;
        cout << "\n" ;
        cout << "Opcoes:" ;
        cout << "\n" ;
        cout << "\n0. Terminar" ;
        cout << "\n1. Testes de posicao" ;
        cout << "\n2. Testes com alteracao e consulta" ;
        cout << "\n" ;
        cout << "\nEscolher sua opcao : " ;
        cin  >>  opcao;

        // escolher opcao
        switch ( opcao )
        {
            case 0:
                cout << "\nEncerrar testes." ;
                break;
            case 1:
                teste1 ( );
                break;
            case 2:
                teste2 ( );
                break;
            default:
                cout << "\nERRO: Opcao invalida." ;
        } // fim escolher

        // encerrar
        cout << "\n" ;
        cout << "\nApertar ENTER." ;
        getchar ( ) ;
    }
    while ( opcao != 0 );
    return EXIT_SUCCESS;
} // fim main

```

Protótipo em C#:

```

/*
 * Tabela - modelo de classe para lidar com tabelas
 *
 */

class Tabela
{
    // definir armazenadores
    public int erro;          // indicador de erro

    public int tamanho;      // quantidade de elementos
    public int livre;        // proxima posicao livre
    public object [ ] valor; // armazenador de dados

/*
 * Tabela ( ) - construtor padrao da classe Tabela
 */
    public Tabela ( int quantidade )
    {
        // atribuir valores iniciais
        erro      = 0;          // nao ha' erro

        tamanho = quantidade;
        livre    = 0;
        valor    = new object [ tamanho ];
    } // fim construtor padrao

/*
 * ToString - converter conteudo para string
 * @return - converter o conteudo da tabela para string
 */
    public override string ToString ( )
    {
        // definir dado local
        string saida = "";
        int x;
        // coletar dados
        for ( x = 0; x < livre; x = x + 1 )
        {
            saida = saida + "\n" + valor [ x ];
        } // fim for
        return ( saida );
    } // fim ToString ( )

/*
 * empty ( ) - informar se a tabela esta' vazia
 * @return resposta - se estiver vazia (true), ou nao (false)
 */
    public bool empty ( )
    {
        return ( livre == 0 );
    } // fim empty ( )

```

```

/*
 * push ( ) - adicionar um valor 'a tabela
 * <br>
 * @param resultado - se houver inclusao (true), ou nao (false)
 * @param x - objeto a ser adicionado
 */
public void push ( object x )
{
    // adicionar um valor 'a tabela
    if ( livre >= tamanho )           // se espaco esgotado
    {
        erro = 1;                     // nao ha' mais espaco
    }
    else                             // se houver espaco
    {
        valor [ livre ] = x;           // guardar valor
        livre = livre + 1;             // passar para a proxima posicao
        erro = 0;                     // nao ha' erro
    } // fim se
} // fim push ( )

/*
 * pop ( ) - retirar um valor da tabela
 * @param resultado - objeto obtido, ou null
 */
public object pop ( )
{
    // definir dado local
    object x = null;
    int y;
    // remover um valor da tabela
    if ( empty ( ) )                 // tabela vazia
    {
        erro = 2;                     // nao ha' elementos
    }
    else                             // ha' elementos
    {
        x = valor [ 0 ];              // separar o primeiro
                                     // deslocar os outros
        for ( y = 0; y < livre-1; y = y + 1 )
            valor [ y ] = valor [ y + 1 ];
        livre = livre - 1;            // indicar um elemento a menos
        valor [ livre ] = null;       // desconectar a referencia

        erro = 0;                     // nao ha' erro
    }
    return ( x );                     // retornar o elemento separado
} // fim pop ( )

```

```

/*
 * peek ( ) - retornar o valor de uma posicao da tabela
 * @param resultado - objeto obtido, ou null
 */
public object peek ( int index )
{
    // definir dado local
    object x = null;
    // obter um valor da tabela
    if ( empty ( ) )           // tabela vazia
    {
        erro = 2;              // nao ha' elementos
    }
    else                       // ha' elementos
    {
        if ( index < 0 || index >= livre )
        {
            erro = 3;          // posicao invalida
        }
        else
        {
            x = valor [ index ]; // copiar o valor da posicao
        } // fim se ( posicao invalida )
    } // fim se ( tabela vazia )
    return ( x );              // retornar o elemento obtido
} // fim peek ( )

/*
 * maxLength ( ) - informar a quantidade maxima de valores na tabela
 * @return - numero maximo de valores na tabela
 */
public int maxLength ( )
{
    return ( tamanho );
} // fim maxLength ( )

/**
 * length ( ) - informar o numero atual de valores na tabela
 * @return - numero atual de valores na tabela
 */
public int length ( )
{
    return ( livre );
} // fim length ( )

} // fim class Tabela

```

```

/*
 * Testar_Tabela - modelo de classe para testar tabelas
 *
 */

public class Testar_Tabela
{
/*
 * teste1 - testar tabela com caracteres
 */
public static void teste1 ( )
{
// definir dados
Tabela a = new Tabela ( 10 );
object x;

// identificar
System.Console.WriteLine ( "Teste 1 - Tabela de caracteres" );
System.Console.WriteLine ( );

// adicionar dados
a.push ( "123" );
if ( a.erro == 0 )
{
a.push ( "abc" );
if ( a.erro == 0 )
{
// mostrar dados
System.Console.WriteLine ( "Tabela com " + a.length( ) + " elementos:" );
while ( ! a.empty ( ) )
{
System.Console.WriteLine ( a );
System.Console.WriteLine ( );
x = (string) a.pop ( );
System.Console.WriteLine ( "Retirado da tabela o elemento " + x );
System.Console.WriteLine ( );
}
} // fim se
} // fim se

// mostrar dados
System.Console.WriteLine ( "Tabela com capacidade para " +
a.maxLength( ) + " elementos:" );
System.Console.WriteLine ( a );
} // fim teste1

```

```

/*
 * teste2 - testar tabela com inteiros
 */
public static void teste2 ( )
{
    // definir dados
    Tabela a = new Tabela ( 10 );
    object x;
    int y, z;

    // identificar
    System.Console.WriteLine ( "Teste 2 - Tabela de inteiros" );
    System.Console.WriteLine ( );

    // adicionar dados
    y = 0;
    while ( y < 3 && a.erro == 0 )
    {
        a.push ( y );
        y = y + 1;
    } // fim while

    // mostrar dados
    System.Console.WriteLine ( "Tabela com " + a.length( ) + " elementos:" );
    System.Console.WriteLine ( a );
    System.Console.WriteLine ( );

    z = a.length ( );
    for ( y = 0; y < z; y = y + 1 )
    {
        if ( a.peek ( y ) is int )
            x = (int) a.pop ( );
        else
            x = null;
        System.Console.WriteLine ( "Retirado da tabela o elemento " + x );
        System.Console.WriteLine ( );
    } // fim repetir
} // fim teste2

```

```

/*
 * main - testar tabelas com objetos
 */
public static void Main ( )
{
    // definir dados
    char opcao = '0';

    // repetir ate' parar
    do
    {
        // oferecer opcoes
        System.Console.WriteLine ( );
        System.Console.WriteLine ( "Teste de tabelas" );
        System.Console.WriteLine ( );
        System.Console.WriteLine ( "Opcoes:" );
        System.Console.WriteLine ( );
        System.Console.WriteLine ( "0. Terminar" );
        System.Console.WriteLine ( "1. Caracteres" );
        System.Console.WriteLine ( "2. Inteiros" );
        System.Console.WriteLine ( );
        System.Console.Write ( "Escolher sua opcao : " );
        try
        {
            opcao = (char) System.Console.Read ( );
            System.Console.ReadLine ( );
        }
        catch ( System.Exception ioex )
        {
            System.Console.WriteLine ( "Erro na leitura da opcao." );
            System.Console.WriteLine ( "Programa encerrado com " + ioex );
            System.Environment.Exit ( 1 );
        } // fim da regio critica
        System.Console.WriteLine ( );
        // escolher opcao
        switch ( opcao )
        {
            case '0':
                System.Console.WriteLine ( "Encerrar testes." );
                break;
            case '1':
                teste1 ( );
                break;
            case '2':
                teste2 ( );
                break;
            default:
                System.Console.WriteLine ( "ERRO: Opcao invalida." );
                break;
        } // fim escolher
    }
}

```



```
// encerrar
System.Console.WriteLine ( );
System.Console.WriteLine ( "Apertar ENTER." );
try
{
    if ( opcao != '0' )
    {
        System.Console.ReadLine ( );
    } // fim se
}
catch ( System.Exception ioex )
{
    System.Console.WriteLine ( "Programa encerrado com " + ioex );
    System.Environment.Exit ( 1 );
} // fim da regioao critica
}
while ( opcao != '0' );
} // fim Main
} // fim class Testar_Tabela
```

Protótipo em Java:

```

/**
 * Tabela - modelo de classe para lidar com tabelas
 *
 */

class Tabela
{
// definir armazenadores
public int erro;          // indicador de erro

public int tamanho;      // quantidade de elementos
public int livre;        // proxima posicao livre
public Object [ ] valor; // armazenador de dados

/**
 * Tabela ( ) - construtor padrao da classe Tabela
 */
public Tabela ( int quantidade )
{
// atribuir valores iniciais
erro = 0;          // nao ha' erro

tamanho = quantidade;
livre = 0;
valor = new Object [ tamanho ];
} // fim construtor padrao

/**
 * toString - converter conteudo para String
 * <br>
 * @return - converter o conteudo da tabela para String
 */
public String toString ( )
{
// definir dado local
String saida = "";
int x;
// coletar dados
for ( x = 0; x < livre ; x = x + 1 )
{
saida = saida + "\n" + valor [ x ];
} // fim for
return ( saida );
} // fim toString ( )

/**
 * empty ( ) - informar se a tabela esta' vazia
 * <br>
 * @return resposta - se estiver vazia (true), ou nao (false)
 */
public boolean empty ( )
{
return ( livre == 0 );
} // fim empty ( )

```

```

/**
 * push ( ) - adicionar um valor 'a tabela
 * <br>
 * @param resultado - se houver inclusao (true), ou nao (false)
 * @param x - objeto a ser adicionado
 */
public void push ( Object x )
{
    // adicionar um valor 'a tabela
    if ( livre >= tamanho )      // se espaco esgotado
    {
        erro = 1;                // nao ha' mais espaco
    }
    else                        // se houver espaco
    {
        valor [ livre ] = x;      // guardar valor
        livre = livre + 1;        // passar para a proxima posicao
        erro = 0;                // nao ha' erro
    } // fim se
} // fim push ( )

/**
 * pop ( ) - retirar um valor da tabela
 * <br>
 * @param resultado - objeto obtido, ou null
 */
public Object pop ( )
{
    // definir dado local
    Object x = null;
    int y;
    // remover um valor da tabela
    if ( empty ( ) )            // tabela vazia
    {
        erro = 2;                // nao ha' elementos
    }
    else                        // ha' elementos
    {
        x = valor [ 0 ];         // separar o primeiro
                                // deslocar os outros
        for ( y = 0; y < livre-1; y = y + 1 )
            valor [ y ] = valor [ y + 1 ];
        livre = livre - 1;        // indicar um elemento a menos
        valor [ livre ] = null;    // desconectar a referencia

        erro = 0;                // nao ha' erro
    }
    return ( x );                // retornar o elemento separado
} // fim pop ( )

```

```

/**
 * peek ( ) - retornar o valor de uma posicao da tabela
 * <br>
 * @param resultado - objeto obtido, ou null
 */
public Object peek ( int index )
{
    // definir dado local
    Object x = null;
    int y;
    // obter um valor da tabela
    if ( empty ( ) )           // tabela vazia
    {
        erro = 2;              // nao ha' elementos
    }
    else                       // ha' elementos
    {
        if ( index < 0 || index >= livre )
        {
            erro = 3;          // posicao invalida
        }
        else
        {
            x = valor [ index ]; // copiar o valor da posicao
        } // fim se ( posicao invalida )
    } // fim se ( tabela vazia )
    return ( x );              // retornar o elemento obtido
} // fim peek ( )

/**
 * maxLength ( ) - informar a quantidade maxima de valores na tabela
 * <br>
 * @return - numero maximo de valores na tabela
 */
public int maxLength ( )
{
    return ( tamanho );
} // fim maxLength ( )

/**
 * length ( ) - informar o numero atual de valores na tabela
 * <br>
 * @return - numero atual de valores na tabela
 */
public int length ( )
{
    return ( livre );
} // fim length ( )

```

```

/**
 * className ( ) - informar o nome da classe de um objeto
 * <br>
 * @return - nome da classe de um objeto na tabela
 */
public String className ( )
{
    String resposta = "";
    Object x = null;;

    if ( empty ( ) )
    {
        erro = 2;                // nao ha' elementos
    }
    else
    {
        x = valor [ 0 ];          // extrair a informacao do objeto
        resposta = x.getClass( ).getName ( );
    }
    return ( resposta );
} // fim className ( )
} // fim class Tabela

```

```

/**
 * Testar_Tabela - modelo de classe para testar tabelas
 *
 */

public class Testar_Tabela
{
/**
 * teste1 - testar tabela com caracteres
 */
public static void teste1 ( )
{
// definir dados
Tabela a = new Tabela ( 10 );
Object x;

// identificar
System.out.println ( "Teste 1 - Tabela de caracteres" );
System.out.println ( );

// adicionar dados
a.push ( "123" );
if ( a.erro == 0 )
{
a.push ( "abc" );
if ( a.erro == 0 )
{
// mostrar dados
System.out.println ( "Tabela com " + a.length() + " elementos:" );
while ( ! a.empty ( ) )
{
System.out.println ( a );
System.out.println ( );
x = (String) a.pop ( );
System.out.println ( "Retirado da tabela o elemento " + x );
System.out.println ( );
}
} // fim se
} // fim se

// mostrar dados
System.out.println ( "Tabela com capacidade para " +
a.maxLength() + " elementos:" );
System.out.println ( a );
} // fim teste1

```

```

/**
 * teste2 - testar tabela com inteiros
 */
public static void teste2 ( )
{
    // definir dados
    Tabela a = new Tabela ( 10 );
    Object x;
    int y, z;

    // identificar
    System.out.println ( "Teste 2 - Tabela de inteiros" );
    System.out.println ( );

    // adicionar dados
    y = 0;
    while ( y < 3 && a.erro == 0 )
    {
        a.push ( new Integer ( y ) );
        y = y + 1;
    } // fim while

    // mostrar dados
    System.out.println ( "Tabela com " + a.length( ) + " elementos:" );
    System.out.println ( a );
    System.out.println ( );

    z = a.length ( );
    for ( y = 0; y < z; y = y + 1 )
    {
        if ( a.className( ).equals( "java.lang.Integer" ) )
            x = (Integer) a.pop ( );
        else
            x = null;
        System.out.println ( "Retirado da tabela o elemento " + x );
        System.out.println ( );
    } // fim repetir
} // fim teste2

```

```

/**
 * main - testar tabelas com objetos
 */
public static void main ( String [ ] args )
{
    // definir dados
    final char enterKey = 13;
    char opcao = '0';
    char enter;

    // repetir ate' parar
    do
    {
        // oferecer opcoes
        System.out.println ( );
        System.out.println ( "Teste de tabelas" );
        System.out.println ( );
        System.out.println ( "Opcoes:" );
        System.out.println ( );
        System.out.println ( "0. Terminar" );
        System.out.println ( "1. Caracteres" );
        System.out.println ( "2. Inteiros" );
        System.out.println ( );
        System.out.print ( "Escolher sua opcao : " );
        try
        {
            opcao = (char) System.in.read ( );
            enter = (char) System.in.read ( );
        }
        catch ( Exception ioex )
        {
            System.out.println ( "Erro na leitura da opcao." );
            System.out.println ( "Programa encerrado." );
            System.exit ( 0 );
        }
        // fim da regioao critica
        System.out.println ( );
        // escolher opcao
        switch ( opcao )
        {
            case '0':
                System.out.println ( "Encerrar testes." );
                break;
            case '1':
                teste1 ( );
                break;
            case '2':
                teste2 ( );
                break;
            default:
                System.out.println ( "ERRO: Opcao invalida." );
        }
        // fim escolher
    }
}

```



```
// encerrar
System.out.println ( );
System.out.println ( "Apertar ENTER." );
try
{
    if ( opcao != '0' )
    {
        do
            opcao = (char) System.in.read ( );
        while ( opcao != enterKey );
        enter = (char) System.in.read ( );
    } // fim se
}
catch ( Exception ioex )
{
    /* nao fazer nada */
} // fim da regioao critica
}
while ( opcao != '0' );
} // fim main
} // fim class Testar_Tabela
```

Programa em Python:

```
class Tabela:
    def __init__( self, maximo=10 ):
        self.maximo = maximo
        self.index = 0
        self.error = 0          # nenhum erro
        self.items = [ [ ] for _ in range ( self.maximo+1 ) ]

    def setError ( self, codigo ):
        self.error = codigo

    def getError ( self ):
        return ( self.error )

    def isEmpty ( self ):
        return self.length ( ) == 0

    def peek ( self, index=0 ):
        if ( self.size ( ) == 0 ):
            self.error = 1          # tabela vazia
            print ( "\nERROR: Empty.\n" )
            return None
        else:
            if ( index < 0 or index >= self.size ( ) ):
                self.error = 1          # posicao invalida
                return None
            else:
                return self.items [ index ]

    def pop ( self ):
        if ( self.length ( ) == 0 ):
            self.error = 1          # tabela vazia
            print ( "\nERROR: Empty.\n" )
            return None
        else:
            value = self.items [ 0 ]
            for _ in range ( 0, self.index ):
                self.items [ _ ] = self.items [ _ + 1 ]
            self.index = self.index - 1
            return value

    def push ( self, index, value ):
        if ( self.length ( ) >= self.maximo ):
            self.error = 2          # tabela cheia
            print ( "\nERROR: Full.\n" )
        else:
            if ( index < 0 or index > self.size ( ) ):
                self.error = 1          # posicao invalida
                return None
            else:
                self.items [ self.index ] = value
                self.index = index + 1;
```

```

def toString ( self ):
    if ( self.size ( ) == 0 ):
        return "{}"
    else:
        txt = "{ "
        for _ in range ( self.index ):
            txt = txt + str ( self.items [ _ ] ) + " "
        txt = txt + "}"
        return txt

def purge ( self ):
    del self.items [:]

def length ( self ):
    return ( self.index )

def size ( self ):
    return ( len ( self.items ) - 1 )

# testes
def teste1 ( ):
    # definir dados
    a = Tabela ( 5 );
    x = 0;
    y = 0;

    # identificar
    print ( "Teste 1 - Colocar valores em posicao" )

    # adicionar dados
    print ( "\nTabela com capacidade para ", a.size ( ), " dados." )
    a.push ( 0, 1.0 );
    if ( a.getError ( ) == 0 ):
        a.push ( 1, 2.0 )
    if ( a.getError ( ) == 0 ):
        a.push ( 2, 3.0 );
    if ( a.getError ( ) == 0 ):
        # mostrar dados
        print ( "\nDados na Tabela: " )

    # mostrar dados
    print ( "\nTabela com ", a.length ( ), " elementos:\n" )
    print ( a.toString ( ) )

    # esvaziar a tabela
    print ( "\nEsvaziar a tabela com ", a.length ( ), " elementos:\n" )
    while ( not a.isEmpty ( ) ):
        z = a.pop ( );
        print ( z )
    # fim repetir

    print ( "\nTabela com ", a.length ( ), " elementos." )

    # fim se
    # fim se
    # fim se
# fim teste1 ( )

```

```

def teste2( ):
    # definir dados
    a = Tabela ( 5 );
    x = 0;
    y = 0;

    # identificar
    print ( "Teste 2 - Colocar valores em posicao" )

    # adicionar dados
    for x in range ( 3 ):
        a.push ( x, (x+1)*0.1 )
    # fim repetir

    # mostrar dados
    print ( "\nTabela com ", a.length ( ), " elementos.\n" )

    for y in range ( a.length ( ) ):
        z = a.peek ( y )
        print ( z )
    # fim repetir

# fim teste2 ( )

# acao principal
def main( ):
    # definir dados
    opcao = 1;

    # repetir ate' parar
    while ( opcao != 0 ):
        # oferecer opcoes
        print ( "\nTeste de Tabelas" )
        print ( "\nOpcoes:" )
        print ( "0. Terminar" )
        print ( "1. Testes de posicao" )
        print ( "2. Testes com alteracao e consulta" )
        print ( )
        print ( "Escolher sua opcao : ", end=" " )
        opcao = int ( input ( ) );

        # escolher opcao
        if ( opcao == 1 ):
            teste1 ( )
        elif ( opcao == 2 ):
            teste2 ( )
        else:
            if ( opcao != 0 ):
                print ( "\nERRO: Opcao invalida." )
            # fim se
    # fim repetir

# fim main ( )

main ( )

```

2. Modelo para tratar vetores

Vetor	- nome da classe
- inteiro erro	- indicador de erro
- inteiro tamanho	- quantidade real de elementos
- real valor [10]	- armazenador para no máximo 10 dados
+ Vetor (inteiro quantidade)	- construtor (padrão)
+ procedimento Alterar (inteiro posição, real dado)	- alterar dado em uma posição do vetor
+ real função Consultar (inteiro posição)	- copiar dado de uma posição do vetor
+ inteiro função Limite	- indicar a quantidade máxima de dados
+ inteiro função Quantidade	- indicar a quantidade atual de dados

Descrição detalhada:

```

classe Vetor
|
| privado:
|
| ! atributos !
| inteiro erro                ! indicador de erro !
|
| inteiro ultimo              ! quantidade atual de elementos !
| inteiro tamanho            ! quantidade máxima de elementos !
| real valor [ 10 ]          ! armazenador para no máximo 10 dados !
|
| ! métodos !
| construtor Vetor ( inteiro quantidade )
|
| ! definição de dado local !
| inteiro x
| ! atribuir valores iniciais !
| repetir para ( x ← 1 : 1 : 10 )
| | valor [ x ] ← 0.0          ! para prevenir usos futuros !
| fim repetir
| se ( quantidade < 1 | quantidade > 10 )
| | tamanho ← 0                ! permanecerá vazio !
| | erro ← 1                    ! indicar falta de espaço !
| senão
| | tamanho ← quantidade       ! por enquanto, vazio !
| | erro ← 0                    ! indicar nenhum erro !
| fim se
| ultimo = 0
fim construtor ! Vetor !

```

```

! métodos !

inteiro função Extrair_Erro
| retornar ( erro )
fim função ! Extrair_Erro !

procedimento Alterar ( inteiro posição, inteiro dado )
|
| ! alterar dado em uma posição do vetor !
| se ( posição < 1 | posição > 10 )
| | erro ← 3 ! indicar posição inválida !
| | senão
| | | valor [ posição ] ← dado ! copiar dado !
| | | se ( posição > ultimo )
| | | | ultimo = posição ! marcar ultimo !
| | | fim se
| | erro ← 0 ! indicar nenhum erro !
| fim se ! posição inválida !
|
fim procedimento ! Alterar !

real função Consultar ( inteiro posição )
|
| ! consultar valor em uma posição do vetor !
| ! definição de dado local !
| real dado ← 0.0 ! valor inicial, caso haja erro !
|
| se ( posição < 1 | posição > 10 )
| | erro ← 3 ! indicar posição inválida !
| | senão
| | | dado ← valor [ posição ] ! copiar dado !
| | | erro ← 0 ! indicar nenhum erro !
| fim se ! posição inválida !
|
| retornar ( dado )
|
fim função ! Consultar !

inteiro função Limite
|
| ! indicar a quantidade máxima de dados no vetor !
| retornar ( tamanho )
|
fim função ! Limite !

inteiro função Quantidade
|
| ! indicar a quantidade atual de dados no vetor !
| retornar ( ultimo )
|
fim função ! Quantidade !

fim classe ! Vetor !

```

Exemplos de usos:

Teste 1:

```
! definir dados !
Vetor a ← Vetor ( 5 )
inteiro y
real x
! adicionar dados !
tela ← ( "Vetor com capacidade para ", a.Limite, " dados." )
a.Alterar ( 1, 0.1 )
se ( a.Extrair_Erro = 0 )
| a. Alterar ( 2, 0.2 )
| se ( a.Extrair_Erro = 0 )
| | a. Alterar ( 3, 0.3 )
| | se ( a.Extrair_Erro = 0 )
| | | tela ← ( "Dados no vetor: " )
| | | repetir para ( y ← 1: 1: 3 )
| | | | x ← a.Consultar ( y )
| | | | tela ← x
| | | fim repetir
| | fim se
| fim se
fim se
```

Teste 2:

```
! definir dados !
Vetor a ← Vetor ( 5 )
inteiro y
real x
! adicionar dados !
tela ← ( "Vetor com capacidade para ", a.Quantidade, " dados." )
y = 1
repetir enquanto ( y <= 3 & a.Extrair_Erro = 0 )
| a.Alterar ( y, y / 10.0 )
| y ← y + 1
fim repetir
! mostrar dados !
tela ← ( "Dados no vetor: " )
repetir para ( y ← 1 : a.Quantidade : 1 )
| x ← a.Consultar ( y )
| se ( a.Extrair_Erro = 0 )
| | tela ← x
| fim se
fim repetir
```

Protótipo em C++:

```
// bibliotecas necessarias
#include <stdlib.h>
#include <string>
#include <iostream>

using namespace std;

/**
 * Vetor - modelo de classe para lidar com vetores
 *
 */

typedef double Object;

/**
 * Vetor - modelo de classe para lidar com Vetores
 *
 */

class Vetor
{
// tratamento de erro
private:
    int erro;                // indicador de erro
    string m;                // mensagem de erro

public:
    string msgError ( );
    int  getError   ( );
    void setError   ( int code );
    void resetError ( );

// tratamento do vetor
private:
// definir armazenadores
    int ultimo ;            // posicao do ultimo
    int tamanho;            // quantidade de dados
    Object valor[10];        // armazenador de dados

public:
// definir metodos publicos
    string saida;

    Vetor ( int quantidade );
    string toString ( );
    void set      ( int x, Object y );
    Object get     ( int x );
    int  length   ( );
    int  size     ( );
}; // fim class Vetor
```



```

/**
 * getError ( ) - obter codigo do erro
 * <br>
 * @return - codigo do erro
 *          0 - nao ha' erro
 *          1 - posicao invalida
 */
int Vetor :: getError ( )
{
    return ( erro );
} // fim getError ( )

/**
 * msgError ( ) - obter mensagem de erro
 * <br>
 * @return - mensagem de erro
 *          0 - nao ha' erro
 *          1 - posicao invalida
 */
string Vetor :: msgError ( )
{
    m = "ERRO: ";
    // selecionar mensagem
    if ( erro == 0 )
        m = m + "Nao ha' erro.";
    else
        if ( erro == 1 )
            m = m + "Posicao invalida.";
        else
            m = m + "Indefinido.";
    // retornar mensagem
    return ( m );
} // fim msgError ( )

/**
 * setError ( ) - definir codigo do erro
 * <br>
 */
void Vetor :: setError ( int code )
{
    erro = code;
} // fim setError ( )

/**
 * resetError ( ) - anular codigo do erro
 * <br>
 */
void Vetor :: resetError ( )
{
    setError ( 0 );
} // fim resetError ( )

```

```

/**
 * Vetor ( ) - construtor do tipo Vetor
 */
Vetor :: Vetor ( int quantidade )
{
    // definir dados locais
    int x, y;

    // atribuir valores iniciais
    for ( x = 0; x < 10; x = x + 1 )
    {
        valor [ x ] = 0.0 ;
    }

    if ( quantidade <= 0 | quantidade > 10 )
    {
        tamanho = 0;
        setError ( 2 );           // valor invalido
    }
    else
    {
        tamanho = quantidade;
        resetError ( );
    }
    ultimo = 0;
} // fim construtor padrao

/**
 * toString - converter conteudo para String
 * <br>
 * @return - indicar o codigo de erro atual
 */
string Vetor :: toString ( )
{
    // definir dado local
    int x;
    char buffer [20];

    saida = "";

    // coletar dados
    for ( x = 0; x < length( ); x = x + 1 )
    {
        saida = saida + "\n";
        sprintf ( buffer, "\t%f", valor [ x ] );
        saida = saida + buffer;
    } // fim for
    return ( saida );
} // fim toString ( )

```

```

/**
 * set ( ) - adicionar um valor ao Vetor
 * <br>
 * @param x - posicao a ser alterada
 * @param dado - objeto a ser adicionado
 */
void Vetor :: set ( int x, Object y )
{
    // adicionar um valor ao Vetor
    if ( x < 0 || x >= tamanho )
    {
        setError ( 1 );           // posicao invalida
    }
    else
    {
        valor [ x ] = y;           // copiar dado para a posicao
        if ( x > ultimo )
            ultimo = x;
        setError ( 0 );           // nao ha' erro
    }
} // fim set ( )

/**
 * get ( ) - obter valor de uma posicao do Vetor
 * <br>
 * @return - objeto obtido da posicao, ou null
 * @param x - posicao onde obter objeto
 */
Object Vetor :: get ( int x )
{
    // definir dado local
    Object z = NULL;
    // obter um valor de uma posicao
    if ( x < 0 || x >= tamanho )
    {
        setError ( 1 );           // posicao invalida
    }
    else
    {
        z = valor [ x ];
        setError ( 0 );           // nao ha' erro
    } // fim se
    return ( z );
} // fim get ( )

/**
 * length ( ) - informar a quantidade atual de dados no vetor
 * <br>
 * @return - numero de posicoes ocupadas no vetor
 */
int Vetor :: length ( )
{
    return ( ultimo+1 );
} // fim length ( )

```

```
/**  
 * size ( ) - informar a quantidade maxima de dados no vetor  
 * <br>  
 * @return - numero maximo de posicoes no vetor  
 */  
int Vetor :: size ( )  
{  
    return ( tamanho );  
} // fim size ( )
```

```

/**
 * teste1 - testar colocacao de dados
 */
void teste1 ( )
{
// definir dados
Vetor a ( 5 );
Object z;
int x, y;

// identificar
cout << "Teste 1 - Colocar valores em posicao" ;
cout << "\n" ;

// adicionar dados
cout << "Vetor com capacidade para " << a.size ( ) << " dados." ;
a.set ( 0, 1.0 );
if ( a.getError() == 0 )
{
a.set ( 1, 2.0 );
if ( a.getError() == 0 )
{
a.set ( 2, 3.0 );
if ( a.getError() == 0 )
{
// mostrar dados
cout << "\n" ;
cout << "\nDados no vetor: " ;
cout << "\n" ;
for ( y = 0; y < a.length ( ); y = y + 1 )
{
x = a.get ( y );
if ( x == NULL )
cout << "\n" << y << " : vazio" ;
else
cout << "\n" << y << " : " << x ;
} // fim repetir
} // fim se
} // fim se
} // fim se

// mostrar dados
cout << "\n" ;
cout << "\nVetor com " << a.length ( ) << " elementos:" ;
cout << "\n" ;
cout << a.toString ( ) ;
} // fim teste1

```

```

/**
 * teste2 - testar alteracao e consulta de dados
 */
void teste2 ( )
{
    // definir dados
    Vetor  a ( 5 );
    Object z;
    int    x;

    // identificar
    cout << "\nTeste 2 - Alterar e consultar valores" ;
    cout << "\n";

    // adicionar dados
    for ( x = 0; x < 3; x = x + 1 )
    {
        a.set ( x, ( x*0.1 ) );
    } // fim repetir

    // mostrar dados
    cout << "\nVetor com " << a.length ( ) << " elementos:" ;
    cout << "\n";

    for ( x = 0; x < a.length( ); x = x + 1 )
    {
        z = a.get ( x );
        cout << "\nElemento na posicao (" << x << ") igual a " << z ;
    } // fim repetir
} // fim teste2

```

```

/**
 * main - testar vetores com objetos
 */
int main ( )
{
    // definir dados
    int opcao = 1;

    // repetir ate' parar
    do
    {
        // oferecer opcoes
        cout << "\n\nTeste de vetores" ;
        cout << "\n" ;
        cout << "Opcoes:" ;
        cout << "\n" ;
        cout << "\n0. Terminar" ;
        cout << "\n1. Testes de posicao" ;
        cout << "\n2. Testes com alteracao e consulta" ;
        cout << "\n" ;
        cout << "\nEscolher sua opcao : " ;
        cin >> opcao;
        cout << "\n" ;

        // escolher opcao
        switch ( opcao )
        {
            case 0:
                cout << "\nEncerrar testes." ;
                break;
            case 1:
                teste1 ( );
                break;
            case 2:
                teste2 ( );
                break;
            default:
                cout << "\nERRO: Opcao invalida." ;
        } // fim escolher

        // encerrar
        cout << "\n" ;
        cout << "\nApertar ENTER." ;
        getchar ( ) ;
    }
    while ( opcao != 0 );
    return EXIT_SUCCESS;
} // fim main

```

Protótipo em C#:

```

/*
 * Vetor - modelo de classe para lidar com vetores
 *
 */

class Vetor
{
    // definir tratamento de erro
    private int erro = 0;          // indicador de erro

    /*
     * getError ( ) - obter código do erro
     * @return - código do erro
     *      0 - não há erro
     *      1 - posição inválida
     */
    public int getError ( )
    {
        return ( erro );
    } // fim getError ( )

    /*
     * errorMsg ( ) - obter mensagem de erro
     * <br>
     * @return - mensagem de erro
     *      0 - não há erro
     *      1 - posição inválida
     */
    public string errorMsg ( )
    {
        // definir dado local
        string msg = "ERRO: ";
        // selecionar mensagem
        switch ( erro )
        {
            case 0:
                msg = msg + "Não há erro.";
                break;
            case 1:
                msg = msg + "Posição inválida.";
                break;
            default:
                msg = msg + "Indefinido.";
                break;
        } // fim selecionar
        // retornar mensagem
        return ( msg );
    } // fim errorMsg ( )

```



```

/**
 * setError ( ) - definir codigo do erro
 */
public void setError ( int code )
{
    erro = code;
} // fim setError ( )

/**
 * resetError ( ) - anular codigo do erro
 */
public void resetError ( )
{
    erro = 0;
} // fim resetError ( )

// definir armazenadores
private int ultimo ;           // posicao do ultimo
private int tamanho;          // quantidade de elementos
private object [ ] valor;      // armazenador de dados

/**
 * Vetor ( ) - construtor padrao da classe Vetor
 */
public Vetor ( int quantidade )
{
    // atribuir valores iniciais
    erro      = 0;              // nao ha' erro

    ultimo    = 0;
    tamanho = quantidade;
    valor      = new object [ tamanho ];
} // fim construtor padrao

/**
 * ToString - converter conteudo para string
 * @return - converter o conteudo do Vetor para string
 */
public override string ToString ( )
{
    // definir dado local
    string saida = "";
    int x;
    // coletar dados
    for ( x = 0; x < length ( ) ; x = x + 1 )
    {
        saida = saida + "\n" + valor [ x ];
    } // fim for
    return ( saida );
} // fim ToString ( )

```

```

/**
 * set ( ) - modificar valor em uma posicao do vetor
 * @param resultado - se houver modificacao (true), ou nao (false)
 * @param index - posicao onde alterar objeto
 * @param x - objeto alterado
 */
public void set ( int index, object x )
{
    // adicionar um valor ao vetor
    if ( index < 0 || index >= tamanho )
    {
        setError ( 1 );          // posicao invalida
    }
    else
    {
        valor [ index ] = x;
        if ( index > ultimo )
            ultimo = index;
        resetError ( );          // nao ha' erro
    } // fim se
} // fim set ( )

/**
 * get ( ) - obter valor de uma posicao do vetor
 * @param resultado - objeto obtido, ou null
 * @param index - posicao onde obter objeto
 */
public object get ( int index )
{
    // definir dado local
    object x = null;
    // obter um valor de uma posicao
    if ( index < 0 || index >= tamanho )
    {
        setError ( 1 );          // posicao invalida
    }
    else
    {
        x = valor [ index ];
        resetError ( );          // nao ha' erro
    } // fim se
    return ( x );
} // fim get ( )

/**
 * length ( ) - informar o tamanho atual do vetor
 * @return - quantidade atual de dados no vetor
 */
public int length ( )
{
    return ( ultimo+1 );
} // fim length ( )

```

```
/**
 * size ( ) - informar o tamanho maximo do vetor
 * @return - quantidade maxima de dados no vetor
 */
public int size ( )
{
    return ( tamanho );
} // fim size ( )

} // fim class Vetor
```

```

/*
 * Testar_Vetor - modelo de classe para testar vetores
 *
 */

public class Testar_Vetor
{
    /*
     * teste1 - testar vetor com caracteres
     */
    public static void teste1 ( )
    {
        // definir dados
        Vetor  a = new Vetor ( 5 );
        object x;
        int    y;
        // identificar
        System.Console.WriteLine ( "Teste 1 - Vetor com caracteres" );
        System.Console.WriteLine ( );
        // adicionar dados
        System.Console.WriteLine ( "Vetor com capacidade para " + a.size ( ) + " dados." );
        a.set ( 0, "123" );
        if ( a.getError( ) == 0 )
        {
            a.set ( 1, "abc" );
            if ( a.getError( ) == 0 )
            {
                // mostrar dados
                for ( y = 0; y < a.length ( ); y = y + 1 )
                {
                    x = a.get ( y );
                    if ( x == null )
                        System.Console.WriteLine ( y + " : vazio" );
                    else
                        System.Console.WriteLine ( y + " : " + x );
                } // fim repetir
            } // fim se
        } // fim se
        // mostrar dados
        System.Console.WriteLine ( );
        System.Console.WriteLine ( "Vetor com " + a.length( ) + " elementos:" );
        System.Console.WriteLine ( a );
    } // fim teste1
}

```

```

/**
 * teste2 - testar vetor com inteiros
 */
public static void teste2 ( )
{
    // definir dados
    Vetor a = new Vetor ( 5 );
    object x;
    int y, z;
    // identificar
    System.Console.WriteLine ( "Teste 2 - Vetor com inteiros" );
    System.Console.WriteLine ( );
    // adicionar dados
    y = 0;
    while ( y < 5 && a.getError ( ) == 0 )
    {
        a.set ( y, y );
        y = y + 1;
    } // fim while
    // mostrar dados
    System.Console.WriteLine ( "Vetor com " + a.length( ) + " elementos:" );
    System.Console.WriteLine ( a );
    System.Console.WriteLine ( );

    z = a.length ( );
    for ( y = 0; y < z; y = y + 1 )
    {
        x = a.get ( y );
        System.Console.WriteLine ( "Elemento na posicao " + y + " igual a " + x );
        System.Console.WriteLine ( );
    } // fim repetir
} // fim teste2

```

```

/**
 * Main - testar tabelas com objetos
 */
public static void Main ( )
{
    // definir dados
    char opcao = '0';

    // repetir ate' parar
    do
    {
        // oferecer opcoes
        System.Console.WriteLine ( );
        System.Console.WriteLine ( "Teste de vetores" );
        System.Console.WriteLine ( );
        System.Console.WriteLine ( "Opcoes:" );
        System.Console.WriteLine ( );
        System.Console.WriteLine ( "0. Terminar" );
        System.Console.WriteLine ( "1. Caracteres" );
        System.Console.WriteLine ( "2. Inteiros" );
        System.Console.WriteLine ( );
        System.Console.Write ( "Escolher sua opcao : " );
        try
        {
            opcao = (char) System.Console.Read ( );
            System.Console.ReadLine ( );
        }
        catch ( System.Exception ioex )
        {
            System.Console.WriteLine ( "Erro na leitura da opcao." );
            System.Console.WriteLine ( "Programa encerrado com " + ioex );
            System.Environment.Exit ( 1 );
        }
        // fim da regio critica
        System.Console.WriteLine ( );
        System.Console.WriteLine ( );
        // escolher opcao
        switch ( (int) opcao )
        {
            case '0':
                System.Console.WriteLine ( "Encerrar testes." );
                break;
            case '1':
                teste1 ( );
                break;
            case '2':
                teste2 ( );
                break;
            default:
                System.Console.WriteLine ( "ERRO: Opcao invalida." );
                break;
        }
        // fim escolher
    }
}

```

```
// encerrar
System.Console.WriteLine ( );
System.Console.WriteLine ( "Apertar ENTER." );
try
{
    if ( opcao != '0' )
    {
        System.Console.ReadLine ( );
    } // fim se
}
catch ( System.Exception ioex )
{
    System.Console.WriteLine ( "Programa encerrado com " + ioex );
    System.Environment.Exit ( 1 );
} // fim da regioao critica
}
while ( opcao != '0' );
} // fim Main

} // fim class Testar_Vetor
```

Protótipo em Java:

```

/**
 * Vetor - modelo de classe para lidar com vetores
 *
 */

class Vetor
{
    // definir tratamento de erro
    private int erro = 0;    // indicador de erro

    /**
     * getError ( ) - obter codigo do erro
     * <br>
     * @return - codigo do erro
     *          0 - nao ha' erro
     *          1 - posicao invalida
     */
    public int getError ( )
    {
        return ( erro );
    } // fim getError ( )

    /**
     * errorMsg ( ) - obter mensagem de erro
     * <br>
     * @return - mensagem de erro
     *          0 - nao ha' erro
     *          1 - posicao invalida
     */
    public String errorMsg ( )
    {
        // definir dado local
        String msg = new String ( "ERRO: " );
        // selecionar mensagem
        switch ( erro )
        {
            case 0:
                msg = msg + "Nao ha' erro.";
                break;
            case 1:
                msg = msg + "Posicao invalida.";
                break;
            default:
                msg = msg + "Indefinido.";
        } // fim selecionar
        // retornar mensagem
        return ( msg );
    } // fim errorMsg ( )

```



```

/**
 * setError ( ) - definir codigo do erro
 * <br>
 */
public void setError ( int code )
{
    erro = code;
} // fim setError ( )

/**
 * resetError ( ) - anular codigo do erro
 * <br>
 */
public void resetError ( )
{
    erro = 0;
} // fim resetError ( )

// definir armazenadores
private int ultimo ;           // posicao do ultimo
private int tamanho;          // quantidade de elementos
private Object [ ] valor;      // armazenador de dados

/**
 * Vetor ( ) - construtor padrao da classe Vetor
 */
public Vetor ( int quantidade )
{
    // atribuir valores iniciais
    erro      = 0;              // nao ha' erro

    ultimo    = 0;
    tamanho = quantidade;
    valor     = new Object [ tamanho ];
} // fim construtor padrao

/**
 * toString - converter conteudo para String
 * <br>
 * @return - converter o conteudo do Vetor para String
 */
public String toString ( )
{
    // definir dado local
    String saida = "";
    int x;
    // coletar dados
    for ( x = 0; x < length ( ) ; x = x + 1 )
    {
        saida = saida + "\n" + valor [ x ];
    } // fim for
    return ( saida );
} // fim toString ( )

```

```

/**
 * set ( ) - modificar valor em uma posicao do vetor
 * <br>
 * @param resultado - se houver modificacao (true), ou nao (false)
 * @param index - posicao onde alterar objeto
 * @param x - objeto alterado
 */
public void set ( int index, Object x )
{
    // definir dado local
    boolean resposta = false;
    // adicionar um valor ao vetor
    if ( index < 0 || index >= tamanho )
    {
        setError ( 1 );          // posicao invalida
    }
    else
    {
        valor [ index ] = x;
        if ( index > ultimo )
            ultimo = index;
        resetError ( );          // nao ha' erro
    } // fim se
} // fim set ( )

/**
 * get ( ) - obter valor de uma posicao do vetor
 * <br>
 * @param resultado - objeto obtido, ou null
 * @param index - posicao onde obter objeto
 */
public Object get ( int index )
{
    // definir dado local
    Object x = null;
    // obter um valor de uma posicao
    if ( index < 0 || index >= tamanho )
    {
        setError ( 1 );          // posicao invalida
    }
    else
    {
        x = valor [ index ];
        resetError ( );          // nao ha' erro
    } // fim se
    return ( x );
} // fim get ( )

/**
 * length ( ) - informar o tamanho atual do vetor
 * <br>
 * @return - quantidade atual de dados no vetor
 */
public int length ( )
{
    return ( ultimo+1 );
} // fim length ( )

```

```
/**
 * size ( ) - informar o tamanho maximo do vetor
 * <br>
 * @return - quantidade maxima de dados no vetor
 */
public int size ( )
{
    return ( tamanho );
} // fim size ( )

} // fim class Vetor
```

```

/**
 * Testar_Vetor - modelo de classe para testar vetores
 *
 */

public class Testar_Vetor
{
/**
 * teste1 - testar vetor com caracteres
 */
public static void teste1 ( )
{
// definir dados
Vetor a = new Vetor ( 5 );
Object x;
int y;
// identificar
System.out.println ( "Teste 1 - Vetor com caracteres" );
System.out.println ( );
// adicionar dados
System.out.println ( "Vetor com capacidade para " + a.size ( ) + " dados." );
a.set ( 0, "123" );
if ( a.getError ( ) == 0 )
{
a.set ( 1, "abc" );
if ( a.getError ( ) == 0 )
{
// mostrar dados
for ( y = 0; y < a.length ( ); y = y + 1 )
{
x = a.get ( y );
if ( x == null )
System.out.println ( y + " : vazio" );
else
System.out.println ( y + " : " + x );
} // fim repetir
} // fim se
} // fim se
// mostrar dados
System.out.println ( );
System.out.println ( "Vetor com " + a.length ( ) + " elementos:" );
System.out.println ( a );
} // fim teste1

```

```

/**
 * teste2 - testar vetor com inteiros
 */
public static void teste2 ( )
{
    // definir dados
    Vetor a = new Vetor ( 5 );
    Object x;
    int y, z;
    // identificar
    System.out.println ( "Teste 2 - Vetor com inteiros" );
    System.out.println ( );
    // adicionar dados
    y = 0;
    while ( y < 5 && a.getError ( ) == 0 )
    {
        a.set ( y, new Integer ( y ) );
        y = y + 1;
    } // fim while
    // mostrar dados
    System.out.println ( "Vetor com " + a.length ( ) + " elementos:" );
    System.out.println ( a );
    System.out.println ( );

    z = a.length ( );
    for ( y = 0; y < z; y = y + 1 )
    {
        x = a.get ( y );
        System.out.println ( "Elemento na posicao " + y + " igual a " + x );
        System.out.println ( );
    } // fim repetir
} // fim teste2

```

```

/**
 * main - testar tabelas com objetos
 */
public static void main ( String [ ] args )
{
    // definir dados
    final char enterKey = 13;
    char opcao = '0';
    char enter;

    // repetir ate' parar
    do
    {
        // oferecer opcoes
        System.out.println ( );
        System.out.println ( "Teste de vetores" );
        System.out.println ( );
        System.out.println ( "Opcoes:" );
        System.out.println ( );
        System.out.println ( "0. Terminar" );
        System.out.println ( "1. Caracteres" );
        System.out.println ( "2. Inteiros" );
        System.out.println ( );
        System.out.print ( "Escolher sua opcao : " );
        try
        {
            opcao = (char) System.in.read ( );
            enter = (char) System.in.read ( );
        }
        catch ( Exception ioex )
        {
            System.out.println ( "Erro na leitura da opcao." );
            System.out.println ( "Programa encerrado." );
            System.exit ( 0 );
        }
        // fim da regioao critica
        System.out.println ( );
        System.out.println ( );
        // escolher opcao
        switch ( opcao )
        {
            case '0':
                System.out.println ( "Encerrar testes." );
                break;
            case '1':
                teste1 ( );
                break;
            case '2':
                teste2 ( );
                break;
            default:
                System.out.println ( "ERRO: Opcao invalida." );
        }
        // fim escolher
    }
}

```

```
// encerrar
System.out.println ( );
System.out.println ( "Apertar ENTER." );
try
{
    if ( opcao != '0' )
    {
        do
            opcao = (char) System.in.read ( );
        while ( opcao != enterKey );
        enter = (char) System.in.read ( );
    } // fim se
}
catch ( Exception ioex )
{
    /* nao fazer nada */
} // fim da regioao critica
}
while ( opcao != '0' );
} // fim main

} // fim class Testar_Vetor
```

Programa em Python:

```
class Vetor:
    def __init__( self, maximo=10 ):
        self.maximo = maximo
        self.index = 0
        self.error = 0          # nenhum erro
        self.items = [ ] for _ in range ( self.maximo+1 )

    def setError ( self, codigo ):
        self.error = codigo

    def getError ( self ):
        return ( self.error )

    def isEmpty ( self ):
        return self.length ( ) == 0

    def get ( self, index ):
        if ( self.size ( ) == 0 ):
            self.error = 1      # tabela vazia
            print ( "\nERROR: Empty.\n" )
            return None
        else:
            if ( index < 0 or index >= self.size ( ) ):
                self.error = 1   # posicao invalida
                return None
            else:
                return self.items [ index ]

    def set ( self, index, value ):
        if ( self.length ( ) >= self.maximo ):
            self.error = 2      # tabela cheia
            print ( "\nERROR: Full.\n" )
        else:
            if ( index < 0 or index > self.size ( ) ):
                self.error = 1   # posicao invalida
                return None
            else:
                self.items [ self.index ] = value
                self.index = index + 1;

    def toString ( self ):
        if ( self.size ( ) == 0 ):
            return "{}"
        else:
            txt = "{ "
            for _ in range ( self.index ):
                txt = txt + str ( self.items [ _ ] ) + " "
            txt = txt + "}"
            return txt

    def length ( self ):
        return ( self.index )

    def size ( self ):
        return ( len ( self.items ) - 1 )
```



```

# testes
def teste1():
    # definir dados
    a = Vetor ( 5 );
    x = 0;
    y = 0;

    # identificar
    print ( "Teste 1 - Colocar valores em posicao" )

    # adicionar dados
    print ( "\nVetor com capacidade para ", a.size ( ), " dados." )
    a.set ( 0, 1.0 );
    if ( a.getError() == 0 ):
        a.set ( 1, 2.0 )
    if ( a.getError() == 0 ):
        a.set ( 2, 3.0 );
    if ( a.getError() == 0 ):
        # mostrar dados
        print ( "\nDados no Vetor: " )
        # fim se
    # fim se
    # fim se

    # mostrar dados
    print ( "\nVetor com ", a.length ( ), " elementos:\n" )
    print ( a.toString ( ) )

# fim teste1 ( )

def teste2():
    # definir dados
    a = Vetor ( 5 );
    x = 0;
    y = 0;

    # identificar
    print ( "Teste 2 - Colocar valores em posicao" )

    # adicionar dados
    for x in range ( 3 ):
        a.set ( x, (x+1)*10 )
    # fim repetir

    # mostrar dados
    print ( "\nVetor com ", a.length ( ), " elementos.\n" )

    for y in range ( a.length ( ) ):
        z = a.get ( y )
        print ( z )
    # fim repetir

# fim teste2 ( )

```

```
# acao principal
def main():
    # definir dados
    opcao = 1;

    # repetir ate' parar
    while ( opcao != 0 ):
        # oferecer opcoes
        print ( "\nTeste de Vetores" )
        print ( "\nOpcoes:" )
        print ( "0. Terminar" )
        print ( "1. Testes de posicao" )
        print ( "2. Testes com alteracao e consulta" )
        print ( )
        print ( "Escolher sua opcao : ", end=" " )
        opcao = int ( input ( ) );

        # escolher opcao
        if ( opcao == 1 ):
            teste1 ( )
        elif ( opcao == 2 ):
            teste2 ( )
        else:
            if ( opcao != 0 ):
                print ( "\nERRO: Opcao invalida." )
            # fim se
    # fim repetir

# fim main ( )

main ( )
```

3. Modelo para tratar matrizes

Matriz	- nome da classe
- inteiro erro	- indicador de erro
- inteiro max_linhas	- quantidade de linhas
- inteiro max_colunas	- quantidade de colunas
- real valor [10] [10]	- armazenador para no máximo 10 dados
+ Matriz (inteiro linhas, inteiro colunas)	- construtor (padrão)
+ procedimento Alterar (inteiro x, inteiro y, real dado)	- alterar dado em uma posição da matriz
+ real função Consultar (inteiro x, inteiro y, real dado)	- copiar dado de uma posição da matriz
+ inteiro função Linhas	- indicar a quantidade atual de linhas
+ inteiro função Colunas	- indicar a quantidade atual de colunas

Descrição detalhada:

```

classe Matriz
|
| privado:
|
| ! atributos !
| inteiro erro                ! indicador de erro !
|
| inteiro max_linhas          ! quantidade de linhas      !
| inteiro max_colunas         ! quantidade colunas       !
| real  valor [ 10 ] [ 10 ]   ! armazenador para no máximo 100 dados !
|
| ! métodos !
| construtor Matriz ( inteiro linhas, inteiro colunas )
|
| | ! definição de dado local !
| | inteiro x, y
| | ! atribuir valores iniciais !
| | repetir para ( x ← 1 : 1 : 10 )
| | | repetir para ( y ← 1 : 1 : 10 )
| | | | valor [ x ] [ y ] ← 0.0 ! para prevenir usos futuros !
| | | fim repetir
| | fim repetir
| | se (( linhas < 1 | linhas > 10 ) | ( colunas < 1 | colunas > 10 ))
| | | max_linhas ← 0          ! permanecerá vazio      !
| | | max_colunas ← 0         ! permanecerá vazio      !
| | | erro ← 1                ! indicar falta de espaço  !
| | senão
| | | max_linhas ← linhas    ! quantidade atual de linhas !
| | | max_colunas ← colunas  ! quantidade atual de colunas !
| | | erro ← 0               ! indicar nenhum erro      !
| | fim se
|
| fim construtor ! Matriz !

```

```

! métodos !

inteiro função Extrair_Erro
| retornar ( erro )
fim função ! Extrair_Erro !

procedimento Alterar ( inteiro x, inteiro y, inteiro dado )
|
| ! alterar dado em uma posição da matriz !
| se (( linhas < 1 | linhas > max_linhas ) |
|   ( colunas < 1 | colunas > max_colunas ))
|   | erro ← 3           ! indicar posição inválida   !
|   | senão
|   |   | valor [ x ] [ y ] ← dado ! copiar dado           !
|   |   | erro ← 0           ! indicar nenhum erro       !
|   |   | fim se ! posição inválida !
|   |
| fim procedimento ! Alterar !

real função Consultar ( inteiro x, inteiro y )
|
| ! consultar valor em uma posição da matriz !
| ! definição de dado local !
| real dado ← 0.0           ! valor inicial, caso haja erro !
|
| se (( linhas < 1 | linhas > max_linhas ) |
|   ( colunas < 1 | colunas > max_colunas ))
|   | erro ← 3           ! indicar posição inválida   !
|   | senão
|   |   | dado ← valor [ x ] [ y ] ! copiar dado           !
|   |   | erro ← 0           ! indicar nenhum erro       !
|   |   | fim se ! posição inválida !
|   |
|   | retornar ( dado )
|   |
| fim função ! Consultar !

inteiro função Linhas
|
| ! indicar a quantidade atual de linhas da matriz !
| retornar ( max_linhas )
|
fim função ! Linhas !

inteiro função Colunas
|
| ! indicar a quantidade atual de colunas da matriz !
| retornar ( max_colunas )
|
fim função ! Colunas !

fim classe ! Matriz !

```

Exemplos de usos:

Teste 1:

```

! definir dados !
Matrizr a ← Matriz ( 3, 3 )
inteiro x, y
real z
! adicionar dados !
tela ← ( "Teste da colocacao de valor em posicao invalida" )
a.Alterar ( -1, 10, 0.1 )
se ( a.Extrair_Erro ≠ 0 )
| tela ← "ERRO: Posicao invalida" )
fim se

tela ← ( "Teste da obtencao de valor em posicao invalida" )
z ← a.Consultar ( -1, 10 )
se ( a.Extrair_Erro ≠ 0 )
| tela ← "ERRO: Posicao invalida" )
fim se

tela ← ( "Teste da colocacao de valores em posicoes validas" )
repetir para ( x ← 1 : 1: 3 )
| repetir para ( y ← 1 : 1: 3 )
| | a.Alterar ( x, y, ( x * 10.0 + y ) )
| | se ( a.Extrair_Erro ≠ 0 )
| | | tela ← "ERRO: Posicao invalida" )
| | fim se
| fim repetir
fim repetir

! mostrar dados !
tela ← ( "Matriz com " + a.Linhas + "x" + a.Colunas ( ) + " elementos:" );
repetir para ( x ← 1 : 1: 3 )
| repetir para ( y ← 1 : 1: 3 )
| | z = a.Consultar ( x, y )
| | se ( a.Extrair_Erro ≠ 0 )
| | | tela ← "ERRO: Posicao invalida" )
| | senão
| | | tela ← ( "Elemento na posicao ( " + x + ", " + y + " ) igual a " + z );
| | fim se
| fim repetir
fim repetir

```

Protótipo em C++:

```
// bibliotecas necessarias

#include <stdlib.h>
#include <string>
#include <iostream>

using namespace std;

/**
 * Error - modelo de classe para lidar com erros
 *
 */

class Error
{
private:
// definir tratamento de erro
int erro; // indicador de erro
string m; // mensagem de erro

public:
Error ( );
string toString ( );
string msg ( );
int get ( );
void set ( int code );
void reset ( );
}; // fim class Error

/**
 * Error ( ) - construtor da classe Error
 * <br>
 */
Error :: Error ( )
{
erro = 0;
} // fim construtor padrao

/**
 * toString - converter conteudo para String
 * <br>
 * @return - indicar o codigo de erro atual
 */
string Error :: toString ( )
{
return ( "ERRO = " + erro );
} // fim toString ( )
```

```

/**
 * get ( ) - obter codigo do erro
 * <br>
 * @return - codigo do erro
 *          0 - nao ha' erro
 *          1 - posicao invalida
 */
int Error :: get ( )
{
    return ( erro );
} // fim get ( )

/**
 * msg ( ) - obter mensagem de erro
 * <br>
 * @return - mensagem de erro
 *          0 - nao ha' erro
 *          1 - posicao invalida
 */
string Error :: msg ( )
{
    m = "ERRO: ";
    // selecionar mensagem
    if ( erro == 0 )
        m = m + "Nao ha' erro.";
    else
        if ( erro == 1 )
            m = m + "Posicao invalida.";
        else
            m = m + "Indefinido.";
    // retornar mensagem
    return ( m );
} // fim msg ( )

/**
 * set ( ) - definir codigo do erro
 * <br>
 */
void Error ::set ( int code )
{
    erro = code;
} // fim set ( )

/**
 * reset ( ) - anular codigo do erro
 * <br>
 */
void Error :: reset ( )
{
    set ( 0 );
} // fim reset ( )

```

```

/**
 * Matriz - modelo de classe para lidar com matrizes
 *
 */

typedef double Object;

class Matriz
{
private:
// definir armazenadores
    int      maxLinhas ,           // quantidade de linhas
           maxColunas;           // quantidade de colunas
    int      linhas ,             // linhas usadas
           colunas;              // colunas usadas
    Object valor[10][10];         // armazenador de dados

public:
// definir tratamento de erro
    Error erro;
    string saida;

    Matriz ( int nLinhas, int nColunas );
    string toString ( );
    void set      ( int x, int y, Object z );
    Object get     ( int x, int y );
    int maxLines   ( );
    int maxColumns ( );
    int lines      ( );
    int columns    ( );
}; // fim class Matriz

```



```

/**
 * Matriz ( ) - construtor do tipo Matriz
 */
Matriz :: Matriz ( int nLinhas, int nColunas )
{
    // definir dados locais
    int x, y;

    // atribuir valores iniciais
    for ( x = 0; x < 10; x = x + 1 )
        for ( y = 0; y < 10; y = y + 1 )
        {
            set ( x, y, 0.0 );
        }

    if ( nLinhas <= 0 || nColunas <= 0 )
    {
        maxLinhas = 0;
        maxColunas = 0;
        erro.set ( 2 );           // valor invalido
    }
    else
    {
        maxLinhas = nLinhas ;
        maxColunas = nColunas;
        erro.reset ( );
    }
    linhas = 0;
    colunas = 0;
} // fim construtor padrao

/**
 * toString - converter conteudo para String
 * <br>
 * @return - indicar o codigo de erro atual
 */
string Matriz :: toString ( )
{
    // definir dado local
    int x, y;
    char buffer [20];

    saida = "";

    // coletar dados
    for ( x = 0; x < linhas; x = x + 1 )
    {
        saida = saida + "\n";
        for ( y = 0; y < colunas; y = y + 1 )
        {
            sprintf ( buffer, "%t%f", valor [ x ] [ y ] );
            saida = saida + buffer;
        } // fim for
    } // fim for
    return ( saida );
} // fim toString ( )

```

```

/**
 * set ( ) - adicionar um valor 'a matriz
 * <br>
 * @param x - linha a ser alterada
 * @param y - coluna a ser alterada
 * @param z - objeto a ser adicionado
 */
void Matriz :: set ( int x, int y, Object z )
{
    // adicionar um valor 'a matriz
    if ( ( x < 0 || x >= maxLinhas ) ||
        ( y < 0 || y >= maxColunas ) )
    {
        erro.set ( 1 );           // posicao invalida
    }
    else
    {
        valor [ x ] [ y ] = z;

        if ( x > linhas )
            linhas = x;           // guardar ultima linha usada
        if ( y > colunas )
            colunas = y;          // guardar ultima coluna usada

        erro.set ( 0 );           // nao ha' erro
    }
} // fim set ( )

/**
 * get ( ) - obter valor de uma posicao da matriz
 * <br>
 * @return   - objeto obtido da posicao, ou null
 * @param x - linha de onde obter objeto
 * @param y - coluna de onde obter objeto
 */
Object Matriz :: get ( int x, int y )
{
    // definir dado local
    Object z = NULL;
    // obter um valor de uma posicao
    if ( ( x < 0 || x >= maxLinhas ) ||
        ( y < 0 || y >= maxColunas ) )
    {
        erro.set ( 1 );           // posicao invalida
    }
    else
    {
        z = valor [ x ] [ y ];
        erro.set ( 0 );           // nao ha' erro
    } // fim se
    return ( z );
} // fim get ( )

```

```

/**
 * maxLines ( ) - informar a quantidade maxima de linhas na matriz
 * <br>
 * @return - numero maximo de linhas na matriz
 */
int Matriz :: maxLines ( )
{
    return ( maxLinhas );
} // fim maxLines ( )

/**
 * maxColumns ( ) - informar a quantidade maxima de colunas na matriz
 * <br>
 * @return - numero maximo de colunas na matriz
 */
int Matriz :: maxColumns ( )
{
    return ( maxColunas );
} // fim maxColumns ( )

/**
 * lines ( ) - informar a quantidade de linhas usadas na matriz
 * <br>
 * @return - numero de linhas usadas na matriz
 */
int Matriz :: lines ( )
{
    return ( linhas+1 );
} // fim lines ( )

/**
 * columns ( ) - informar a quantidade de colunas usadas na matriz
 * <br>
 * @return - numero de colunas usadas na matriz
 */
int Matriz :: columns ( )
{
    return ( colunas+1 );
} // fim columns ( )

```

```

/**
 * teste1 - testar colocacao de dados
 */
void teste1 ( )
{
    // definir dados
    Matriz a ( 5, 5 );
    Object z;
    int x, y;

    // identificar
    cout << "Teste 1 - Colocar valores em posicao" ;
    cout << "\n" ;

    // testar posicao invalida
    cout << "\nTeste de colocar valor em posicao invalida" ;
    cout << "\n" ;
    a.set ( -1, 10, 0.1 );
    if ( a.erro.get ( ) != 0 )
        cout << a.erro.msg ( ) ;

    // testar obter valor de posicao vazia
    cout << "\nTeste de obter valor de posicao invalida" ;
    cout << "\n";
    a.get ( -1, 10 );
    if ( a.erro.get ( ) != 0 )
        cout << a.erro.msg ( ) ;

    // adicionar dados
    for ( x = 0; x < 3; x = x + 1 )
        for ( y = 0; y < 3; y = y + 1 )
        {
            a.set ( x, y, ( x * 10.0 + y ) );
            if ( a.erro.get ( ) != 0 )
                cout << a.erro.msg ( ) ;
        } // fim repetir

    // mostrar dados
    cout << "\n" ;
    cout << "\nMatriz com " << a.lines ( ) << "x" << a.columns ( ) << " elementos:" ;
    cout << "\n" ;
    cout << a.toString ( ) ;
} // fim teste1

```

```

/**
 * teste2 - testar alteracao e obtencao de dados
 */
void teste2 ( )
{
    // definir dados
    Matriz  a ( 5, 5 );
    Object  z;
    int     x, y;

    // identificar
    cout << "\nTeste 2 - Alterar e consultar valores" ;
    cout << "\n";

    // adicionar dados
    for ( x = 0; x < 3; x = x + 1 )
        for ( y = 0; y < 3; y = y + 1 )
        {
            a.set ( x, y, ( x*10.0 + y ) );
        } // fim repetir

    // mostrar dados
    cout << "\nMatriz com " << a.lines ( ) << "x" << a.columns ( ) << " elementos:" ;
    cout << "\n";

    for ( x = 0; x < a.lines( ); x = x + 1 )
    {
        for ( y = 0; y < a.columns( ); y = y + 1 )
        {
            z = a.get ( x, y );
            cout << "\nElemento na posicao (" << x << ", " << y << ") igual a " << z ;
        } // fim repetir
    } // fim repetir
} // fim teste2

```

```

/**
 * main - testar matrizes com objetos
 */
int main ( )
{
    // definir dados
    int opcao = 1;

    // repetir ate' parar
    do
    {
        // oferecer opcoes
        cout << "\n\nTeste de matrizes" ;
        cout << "\n" ;
        cout << "Opcoes:" ;
        cout << "\n" ;
        cout << "\n0. Terminar" ;
        cout << "\n1. Testes de posicao" ;
        cout << "\n2. Testes com alteracao e consulta" ;
        cout << "\n" ;
        cout << "\nEscolher sua opcao : " ;
        cin  >>  opcao;
        cout << "\n" ;

        // escolher opcao
        switch ( opcao )
        {
            case 0:
                cout << "\nEncerrar testes." ;
                break;
            case 1:
                teste1 ( );
                break;
            case 2:
                teste2 ( );
                break;
            default:
                cout << "\nERRO: Opcao invalida." ;
        } // fim escolher

        // encerrar
        cout << "\n" ;
        cout << "\nApertar ENTER." ;
        getchar ( ) ;
    }
    while ( opcao != 0 );
    return EXIT_SUCCESS;
} // fim main

```

Protótipo em C#:

```

/*
 * Erro - modelo de classe para lidar com erros.
 *
 */

class Erro
{
    // definir tratamento de erro
    private int erro;           // indicador de erro

/*
 * Erro ( ) - construtor da classe Error.
 */
    public Erro ( )
    {
        erro = 0;
    } // fim construtor padrao

/*
 * ToString - converter conteudo para string.
 * @return - indicar o codigo de erro atual
 */
    public override string ToString ( )
    {
        return ( "ERRO = " + erro );
    } // fim ToString ( )

/*
 * get ( ) - obter codigo do erro.
 * @return - codigo do erro
 *         0 - nao ha' erro
 *         1 - posicao invalida
 */
    public int get ( )
    {
        return ( erro );
    } // fim get ( )

```

```

/*
 * msg ( ) - obter mensagem de erro.
 * @return - mensagem de erro
 *      0 - nao ha' erro
 *      1 - posicao invalida
 */
public string msg ( )
{
    // definir dado local
    string m = "ERRO: ";
    // selecionar mensagem
    switch ( erro )
    {
        case 0:
            m = m + "Nao ha' erro.";
            break;
        case 1:
            m = m + "Posicao invalida.";
            break;
        default:
            m = m + "Indefinido.";
            break;
    } // fim selecionar
    // retornar mensagem
    return ( m );
} // fim msg ( )

/*
 * set ( ) - definir codigo do erro.
 */
public void set ( int code )
{
    erro = code;
} // fim set ( )

/*
 * reset ( ) - anular codigo do erro.
 */
public void reset ( )
{
    set ( 0 );
} // fim reset ( )

} // fim class Error

```



```

/*
 * Matriz - modelo de classe para lidar com matrizes.
 *
 */

class Matriz
{
    // definir tratamento de erro
    public Erro erro = new Erro ( );

    // definir armazenadores
    private int maxLinhas,      // quantidade de linhas
               maxColunas;     // quantidade de colunas
    private int linhas ,       // linhas usadas
               colunas;        // colunas usadas
    private object [ , ] valor; // armazenador de dados

    /*
     * Matriz ( ) - construtor do tipo Matriz.
     * @param nLinhas - quantidade de linhas
     * @param nColunas - quantidade de colunas
     */
    public Matriz ( int nLinhas, int nColunas )
    {
        // atribuir valores iniciais
        if ( nLinhas <= 0 | nColunas <= 0 )
        {
            maxLinhas = 0;
            maxColunas = 0;
            valor = null;
            erro.set ( 2 ); // valor invalido
        }
        else
        {
            maxLinhas = nLinhas ;
            maxColunas = nColunas;
            valor = new object [ maxLinhas, maxColunas ];
            erro.reset ( );
        }
        linhas = 0;
        colunas = 0;
    } // fim construtor padrao

    /*
     * maxLines ( ) - informar a quantidade maxima de linhas na matriz
     * @return - numero maximo de linhas na matriz
     */
    public int maxLines ( )
    {
        return ( maxLinhas );
    } // fim maxLines ( )

```

```

/*
 * maxColumns ( ) - informar a quantidade maxima de colunas na matriz.
 * @return      - numero maximo de colunas na matriz
 */
public int maxColumns ( )
{
    return ( maxColunas );
} // fim maxColumns ( )

/*
 * lines ( ) - informar a quantidade de linhas usadas na matriz.
 * @return - numero de linhas usadas na matriz
 */
public int lines ( )
{
    return ( linhas+1 );
} // fim lines ( )

/*
 * columns ( ) - informar a quantidade de colunas usadas na matriz.
 * @return      - numero de colunas usadas na matriz
 */
public int columns ( )
{
    return ( colunas+1 );
} // fim columns ( )

/*
 * ToString - converter conteudo para string.
 * @return - indicar o codigo de erro atual
 */
public override string ToString ( )
{
    // definir dado local
    string saida = "";
    int x, y;
    // coletar dados
    for ( x = 0; x < lines( ) ; x = x + 1 )
    {
        for ( y = 0; y < columns( ) ; y = y + 1 )
        {
            saida = saida + "\t" + valor [ x, y ];
        } // fim for
        saida = saida + "\n";
    } // fim for
    return ( saida );
} // fim ToString ( )

```

```

/*
 * set ( )      - adicionar um valor 'a matriz.
 * @param x - linha a ser alterada
 * @param y - coluna a ser alterada
 * @param z - objeto a ser adicionado
 */
public void set ( int x, int y, object z )
{
    // adicionar um valor 'a matriz
    if ( ( x < 0 || x >= maxLinhas ) ||
        ( y < 0 || y >= maxColunas ) )
    {
        erro.set ( 1 );           // posicao invalida
    }
    else
    {
        valor [ x, y ] = z;
        if ( x > linhas )
            linhas = x;           // guardar ultima linha usada
        if ( y > colunas )
            colunas = y;          // guardar ultima coluna usada

        erro.reset ( );           // nao ha' erro
    }
} // fim set ( )

/*
 * get ( )      - obter valor de uma posicao da matriz.
 * @return      - objeto obtido da posicao, ou null
 * @param x - linha de onde obter objeto
 * @param y - coluna de onde obter objeto
 */
public object get ( int x, int y )
{
    // definir dado local
    object z = null;
    // obter um valor de uma posicao
    if ( ( x < 0 || x >= maxLinhas ) ||
        ( y < 0 || y >= maxColunas ) )
    {
        erro.set ( 1 );           // posicao invalida
    }
    else
    {
        z = valor [ x, y ];
        erro.reset ( );           // nao ha' erro
    } // fim se
    return ( z );
} // fim get ( )

} // fim class Matriz

```

```

/*
 * Testar_Matriz - modelo de classe para testar matrizes.
 *
 */

public class Testar_Matriz
{
/*
 * teste1 - testar matriz com caracteres.
 */
    public static void teste1 ( )
    {
        // definir dados
        Matriz a = new Matriz ( 5, 5 );
        int x, y;

        // identificar
        System.Console.WriteLine ( "Teste 1 - Matriz com caracteres" );
        System.Console.WriteLine ( );

        // testar posicao invalida
        System.Console.WriteLine ( "Teste de colocar valor em posicao invalida" );
        System.Console.WriteLine ( );
        a.set ( -1, 10, "erro" );
        if ( a.erro.get ( ) != 0 )
            System.Console.WriteLine ( a.erro.msg ( ) );

        // testar obter valor de posicao vazia
        System.Console.WriteLine ( );
        System.Console.WriteLine ( "Teste de obter valor de posicao invalida" );
        System.Console.WriteLine ( );
        a.get ( -1, 10 );
        if ( a.erro.get ( ) != 0 )
            System.Console.WriteLine ( a.erro.msg ( ) );

        // adicionar dados
        for ( x = 0; x < 3; x = x + 1 )
            for ( y = 0; y < 3; y = y + 1 )
            {
                a.set ( x, y, ( "" + x + y ) );
                if ( a.erro.get ( ) != 0 )
                    System.Console.WriteLine ( a.erro.msg ( ) );
            } // fim repetir

        // mostrar dados
        System.Console.WriteLine ( );
        System.Console.WriteLine ( "Matriz com " +
                                   a.lines ( ) + "x" + a.columns ( ) + " elementos:" );
        System.Console.WriteLine ( a );
    } // fim teste1
}

```

```

/*
 * teste2 - testar matriz com inteiros.
 */
public static void teste2 ( )
{
    // definir dados
    Matriz a = new Matriz ( 5, 5 );
    object z;
    int x, y;

    // identificar
    System.Console.WriteLine ( "Teste 2 - Matriz com inteiros" );
    System.Console.WriteLine ( );

    // adicionar dados
    for ( x = 0; x < 3; x = x + 1 )
        for ( y = 0; y < 3; y = y + 1 )
        {
            a.set ( x, y, ( x*10 + y ) );
        } // fim repetir

    // mostrar dados
    System.Console.WriteLine ( "Matriz com " +
                               a.lines ( ) + "x" + a.columns ( ) + " elementos:" );
    System.Console.WriteLine ( );

    for ( x = 0; x < a.lines( ); x = x + 1 )
    {
        for ( y = 0; y < a.columns( ); y = y + 1 )
        {
            z = a.get ( x, y );
            System.Console.WriteLine ( "Elemento na posicao (" + x + "," + y + ") igual a " + z );
        } // fim repetir
    } // fim repetir
} // fim teste2

```

```

/*
 * Main - testar matrizes com objetos.
 */
public static void Main ( )
{
    // definir dados
    char opcao = '0';

    // repetir ate' parar
    do
    {
        // oferecer opcoes
        System.Console.WriteLine ( );
        System.Console.WriteLine ( "Teste de matrizes" );
        System.Console.WriteLine ( );
        System.Console.WriteLine ( "Opcoes:" );
        System.Console.WriteLine ( );
        System.Console.WriteLine ( "0. Terminar" );
        System.Console.WriteLine ( "1. Testes de posicao" );
        System.Console.WriteLine ( "2. Testes com alteracao e consulta" );
        System.Console.WriteLine ( );
        System.Console.Write ( "Escolher sua opcao : " );
        try
        {
            opcao = (char) System.Console.Read ( );
            System.Console.ReadLine ( );
        }
        catch ( System.Exception ioex )
        {
            System.Console.WriteLine ( "Erro na leitura da opcao." );
            System.Console.WriteLine ( "Programa encerrado com " + ioex );
            System.Environment.Exit ( 1 );
        }
        // fim da regioao critica
        System.Console.WriteLine ( );

        // escolher opcao
        switch ( (int) opcao )
        {
            case '0':
                System.Console.WriteLine ( "Encerrar testes." );
                break;
            case '1':
                teste1 ( );
                break;
            case '2':
                teste2 ( );
                break;
            default:
                System.Console.WriteLine ( "ERRO: Opcao invalida." );
                break;
        }
        // fim escolher
    }
}

```

```
// encerrar
System.Console.WriteLine ( );
System.Console.WriteLine ( "Apertar ENTER." );
try
{
    if ( opcao != '0' )
    {
        System.Console.ReadLine ( );
    } // fim se
}
catch ( System.Exception ioex )
{
    System.Console.WriteLine ( "Programa encerrado com " + ioex );
} // fim da regioao critica
}
while ( opcao != '0' );
} // fim main
} // fim class Testar_Matriz
```

Protótipo em Java:

```

/**
 * Erro - modelo de classe para lidar com erros.
 *
 */

class Erro
{
// definir tratamento de erro
    private int erro;           // indicador de erro

/**
 * Erro ( ) - construtor da classe Error.
 * <br>
 */
    public Erro ( )
    {
        erro = 0;
    } // fim construtor padrao

/**
 * toString - converter conteudo para String.
 * <br>
 * @return - indicar o codigo de erro atual
 */
    public String toString ( )
    {
        return ( "ERRO = " + erro );
    } // fim toString ( )

/**
 * get ( ) - obter codigo do erro.
 * <br>
 * @return - codigo do erro
 *          0 - nao ha' erro
 *          1 - posicao invalida
 */
    public int get ( )
    {
        return ( erro );
    } // fim get ( )

```



```

/**
 * msg ( ) - obter mensagem de erro.
 * <br>
 * @return - mensagem de erro
 *          0 - nao ha' erro
 *          1 - posicao invalida
 */
public String msg ( )
{
    // definir dado local
    String m = new String ( "ERRO: " );
    // selecionar mensagem
    switch ( erro )
    {
        case 0:
            m = m + "Nao ha' erro.";
            break;
        case 1:
            m = m + "Posicao invalida.";
            break;
        default:
            m = m + "Indefinido.";
    } // fim selecionar
    // retornar mensagem
    return ( m );
} // fim msg ( )

/**
 * set ( ) - definir codigo do erro.
 * <br>
 */
public void set ( int code )
{
    erro = code;
} // fim set ( )

/**
 * reset ( ) - anular codigo do erro.
 * <br>
 */
public void reset ( )
{
    set ( 0 );
} // fim reset ( )

} // fim class Error

```

```

/**
 * Matriz - modelo de classe para lidar com matrizes.
 *
 */

class Matriz
{
// definir tratamento de erro
public Erro erro = new Erro ( );

// definir armazenadores
private int maxLinhas, // quantidade de linhas
           maxColunas; // quantidade de colunas
private int linhas ,   // linhas usadas
           colunas;     // colunas usadas
private Object [ ][ ] valor; // armazenador de dados

/**
 * Matriz ( ) - construtor do tipo Matriz.
 * @param nLinhas - quantidade de linhas
 * @param nColunas - quantidade de colunas
 */
public Matriz ( int nLinhas, int nColunas )
{
// definir dados locais
int x, y;

// atribuir valores iniciais
if ( nLinhas <= 0 | nColunas <= 0 )
{
    maxLinhas = 0;
    maxColunas = 0;
    valor = null;
    erro.set ( 2 ); // valor invalido
}
else
{
    maxLinhas = nLinhas ;
    maxColunas = nColunas;
    valor = new Object [ maxLinhas ][ maxColunas ];
    erro.reset ( );
}
linhas = 0;
colunas = 0;
} // fim construtor padrao

/**
 * maxLines ( ) - informar a quantidade maxima de linhas na matriz
 * <br>
 * @return - numero maximo de linhas na matriz
 */
public int maxLines ( )
{
    return ( maxLinhas );
} // fim maxLines ( )

```

```

/**
 * maxColumns ( ) - informar a quantidade maxima de colunas na matriz.
 * <br>
 * @return      - numero maximo de colunas na matriz
 */
public int maxColumns ( )
{
    return ( maxColunas );
} // fim maxColumns ( )

/**
 * lines ( ) - informar a quantidade de linhas usadas na matriz.
 * <br>
 * @return - numero de linhas usadas na matriz
 */
public int lines ( )
{
    return ( linhas+1 );
} // fim lines ( )

/**
 * columns ( ) - informar a quantidade de colunas usadas na matriz.
 * <br>
 * @return  - numero de colunas usadas na matriz
 */
public int columns ( )
{
    return ( colunas+1 );
} // fim columns ( )

/**
 * toString - converter conteudo para String.
 * <br>
 * @return - indicar o codigo de erro atual
 */
public String toString ( )
{
    // definir dado local
    String saida = "";
    int x, y;
    // coletar dados
    for ( x = 0; x < lines( ) ; x = x + 1 )
    {
        for ( y = 0; y < columns( ) ; y = y + 1 )
        {
            saida = saida + "\t" + valor [ x ] [ y ];
        } // fim for
        saida = saida + "\n";
    } // fim for
    return ( saida );
} // fim toString ( )

```

```

/**
 * set ( )      - adicionar um valor 'a matriz.
 * <br>
 * @param x - linha a ser alterada
 * @param y - coluna a ser alterada
 * @param z - objeto a ser adicionado
 */
public void set ( int x, int y, Object z )
{
    // adicionar um valor 'a matriz
    if ( ( x < 0 || x >= maxLinhas ) ||
        ( y < 0 || y >= maxColunas ) )
    {
        erro.set ( 1 );           // posicao invalida
    }
    else
    {
        valor [ x ] [ y ] = z;
        if ( x > linhas )
            linhas = x;           // guardar ultima linha usada
        if ( y > colunas )
            colunas = y;          // guardar ultima coluna usada

        erro.reset ( );          // nao ha' erro
    }
} // fim set ( )

/**
 * get ( )      - obter valor de uma posicao da matriz.
 * <br>
 * @return      - objeto obtido da posicao, ou null
 * @param x - linha de onde obter objeto
 * @param y - coluna de onde obter objeto
 */
public Object get ( int x, int y )
{
    // definir dado local
    Object z = null;
    // obter um valor de uma posicao
    if ( ( x < 0 || x >= maxLinhas ) ||
        ( y < 0 || y >= maxColunas ) )
    {
        erro.set ( 1 );           // posicao invalida
    }
    else
    {
        z = valor [ x ] [ y ];
        erro.reset ( );          // nao ha' erro
    } // fim se
    return ( z );
} // fim get ( )

} // fim class Matriz

```

```

/**
 * Testar_Matriz - modelo de classe para testar matrizes.
 *
 */

public class Testar_Matriz
{
    /**
     * teste1 - testar matriz com caracteres.
     */
    public static void teste1 ( )
    {
        // definir dados
        Matriz a = new Matriz ( 5, 5 );
        Object z;
        int x, y;

        // identificar
        System.out.println ( "Teste 1 - Matriz com caracteres" );
        System.out.println ( );

        // testar posicao invalida
        System.out.println ( "Teste de colocar valor em posicao invalida" );
        System.out.println ( );
        a.set ( -1, 10, new String ( "erro" ) );
        if ( a.erro.get ( ) != 0 )
            System.out.println ( a.erro.msg ( ) );

        // testar obter valor de posicao vazia
        System.out.println ( );
        System.out.println ( "Teste de obter valor de posicao invalida" );
        System.out.println ( );
        a.get ( -1, 10 );
        if ( a.erro.get ( ) != 0 )
            System.out.println ( a.erro.msg ( ) );

        // adicionar dados
        for ( x = 0; x < 3; x = x + 1 )
            for ( y = 0; y < 3; y = y + 1 )
            {
                a.set ( x, y, new String ( "" + x + y ) );
                if ( a.erro.get ( ) != 0 )
                    System.out.println ( a.erro.msg ( ) );
            } // fim repetir

        // mostrar dados
        System.out.println ( );
        System.out.println ( "Matriz com " + a.lines ( ) + "x" + a.columns ( ) + " elementos:" );
        System.out.println ( a );
    } // fim teste1
}

```

```

/**
 * teste2 - testar matriz com inteiros.
 */
public static void teste2 ( )
{
    // definir dados
    Matriz a = new Matriz ( 5, 5 );
    Object z;
    int x, y;

    // identificar
    System.out.println ( "Teste 2 - Matriz com inteiros" );
    System.out.println ( );

    // adicionar dados
    for ( x = 0; x < 3; x = x + 1 )
        for ( y = 0; y < 3; y = y + 1 )
        {
            a.set ( x, y, new Integer ( x*10 + y ) );
        } // fim repetir

    // mostrar dados
    System.out.println ( "Matriz com " + a.lines ( ) + "x" + a.columns ( ) + " elementos:" );
    System.out.println ( );

    for ( x = 0; x < a.lines( ); x = x + 1 )
    {
        for ( y = 0; y < a.columns( ); y = y + 1 )
        {
            z = a.get ( x, y );
            System.out.println ( "Elemento na posicao (" + x + "," + y + ") igual a " + z );
        } // fim repetir
    } // fim repetir
} // fim teste2

```

```

/**
 * main - testar matrizes com objetos.
 */
public static void main ( String [ ] args )
{
    // definir dados
    final char enterKey = 13;
    char opcao = '0';
    char enter;

    // repetir ate' parar
    do
    {
        // oferecer opcoes
        System.out.println ( );
        System.out.println ( "Teste de matrizes" );
        System.out.println ( );
        System.out.println ( "Opcoes:" );
        System.out.println ( );
        System.out.println ( "0. Terminar" );
        System.out.println ( "1. Testes de posicao" );
        System.out.println ( "2. Testes com alteracao e consulta" );
        System.out.println ( );
        System.out.print ( "Escolher sua opcao : " );
        try
        {
            opcao = (char) System.in.read ( );
            enter = (char) System.in.read ( );
        }
        catch ( Exception ioex )
        {
            System.out.println ( "Erro na leitura da opcao." );
            System.out.println ( "Programa encerrado." );
            System.exit ( 0 );
        }
        // fim da regioao critica
        System.out.println ( );

        // escolher opcao
        switch ( opcao )
        {
            case '0':
                System.out.println ( "Encerrar testes." );
                break;
            case '1':
                teste1 ( );
                break;
            case '2':
                teste2 ( );
                break;
            default:
                System.out.println ( "ERRO: Opcao invalida." );
        }
        // fim escolher
    }
}

```

```
// encerrar
System.out.println ( );
System.out.println ( "Apertar ENTER." );
try
{
    if ( opcao != '0' )
    {
        do
            opcao = (char) System.in.read ( );
        while ( opcao != enterKey );
        enter = (char) System.in.read ( );
    } // fim se
}
catch ( Exception ioex )
{
    /* nao fazer nada */
} // fim da regioao critica
}
while ( opcao != '0' );
} // fim main
} // fim class Testar_Matriz
```


Programa em Python:

Error - modelo de classe para lidar com erros
class Error:

 # construtor da classe Error

 def __init__(self):

 self.msg = ""

 self.error = 0 # nenhum erro

 # get () - obter codigo do erro

 def getError (self):

 return (self.error)

 # definir codigo do erro

 def setError (self, code):

 self.error = code

 # anular codigo do erro

 def resetError (self):

 self.error = 0

 # obter mensagem de erro

 def msgError (self):

 txt = "ERRO: ";

 # selecionar mensagem

 if (self.error == 0):

 txt = txt + "Nao ha' erro."

 elif (self.error == 1):

 txt = txt + "Valor invalido."

 else:

 txt = txt + "Indefinido.";

 return txt;

 # fim msg ()

fim class Error

```

# Matriz - modelo de classe para lidar com matrizes
# com heranca da classe Error
class Matriz ( Error ):
    def __init__( self, rows=10, columns=10 ):
        Error.__init__( self )
        self.maxRows = 0
        if ( rows <= 0 ):
            self.setError ( 1 ) # valor invalido
        else:
            self.maxRows = rows
            self.maxColumns = columns
            if ( columns <= 0 ): # valor invalido
                self.setError ( 1 )
            else:
                self.maxColumns = columns
        if ( self.getError ( ) == 0 ):
            self.items = [ [0 for x in range ( self.maxColumns ) ] for y in range( self.maxRows ) ]

    def isEmpty ( self ):
        return self.length ( ) == 0

    def get ( self, row, column ):
        if ( row < 0 or row >= self.rows ( ) or \
            column < 0 or column >= self.columns ( ) ):
            self.setError ( 1 ) # posicao invalida
            print ( self.msgError ( ) )
            return None
        else:
            self.resetError ( )
            return self.items [ row ][ column ]

    def set ( self, row, column, value ):
        if ( row < 0 or row >= self.rows ( ) or \
            column < 0 or column >= self.columns ( ) ):
            self.setError ( 1 ) # posicao invalida
            print ( self.msgError ( ) )
        else:
            self.setError ( 0 ) # nenhum erro agora
            self.items [ row ][ column ] = value

    def toString ( self ):
        if ( self.rows ( ) == 0 or self.columns ( ) == 0 ):
            return "{}"
        else:
            txt = ""
            for x in range ( self.maxRows ):
                for y in range ( self.maxColumns ):
                    txt = txt + str ( self.items [x][y] ) + "\t"
                txt = txt + "\n"
            return txt
        # fim se

    def rows ( self ):
        return ( self.maxRows )

    def columns ( self ):
        return ( self.maxColumns )

# fim class Matriz

```

```

# teste1 - testar colocacao de dados
def teste1 ( ):
# definir dados
    a = Matriz ( 5, 5 )
    x = 0
    y = 0

# identificar
    print ( "Teste 1 - Colocar valores em posicao\n" )

# testar posicao invalida
    print ( "\nTeste para colocar valor em posicao invalida\n" )
    a.set ( -1, 10, 0.1 );
    if ( a.getError ( ) != 0 ):
        print ( "Erro = ", a.getError ( ) )

# testar obter valor de posicao vazia
    print ( "\nTeste de obter valor de posicao invalida" )
    print ( )
    a.get ( -1, 10 );
    if ( a.getError ( ) != 0 ):
        print ( "Erro = ", a.getError ( ) );

# adicionar dados
    for x in range ( 3 ):
        for y in range ( 3 ):
            a.set ( x, y, ( x * 10.0 + y ) );
            if ( a.getError ( ) != 0 ):
                print ( "Erro = ", a.getError ( ) )
        # fim repetir
    # fim repetir

# mostrar dados
    print ( "\nMatriz com ", a.rows ( ), "x", a.columns ( ), " elementos:\n" )
    print ( a.toString ( ) );

# fim teste1

```

```
# teste2 - testar alteracao e obtencao de dados
def teste2 ( ):
# definir dados
    a = Matriz ( 5, 5 )
    x = 0
    y = 0

# identificar
    print ( "Teste 2 - Colocar valores em posicao\n" )

# adicionar dados
    for x in range ( 3 ):
        for y in range ( 3 ):
            a.set ( x, y, ( x * 10.0 + y ) )
        # fim repetir
    # fim repetir

# mostrar dados
    print ( "\nMatriz com ", a.rows ( ), "x", a.columns ( ), " elementos:\n" )

    for x in range ( 3 ):
        for y in range ( 3 ):
            z = a.get ( x, y )
            print ( z, "\t", end=" " )
        # fim repetir
    print ( )
    # fim repetir

# fim teste2
```

```

# acao principal
def main( ):
    # definir dados
    opcao = 1;

    # repetir ate' parar
    while ( opcao != 0 ):
        # oferecer opcoes
        print ( "\nTeste de Matrizes" )
        print ( "\nOpcoes:" )
        print ( "0. Terminar" )
        print ( "1. Testes de posicao" )
        print ( "2. Testes com alteracao e consulta" )
        print ( )
        print ( "Escolher sua opcao : ", end=" " )
        opcao = int ( input ( ) );

        # escolher opcao
        if ( opcao == 0 ):
            pass
        elif ( opcao == 1 ):
            teste1 ( )
        elif ( opcao == 2 ):
            teste2 ( )
        else:
            print ( "\nERRO: Opcao invalida." )
        # fim se
    # fim repetir

    print ( "\nPressionar ENTER para terminar" )
    input ( )
# fim main ( )

main ( )

```

Exercícios propostos.

1. Fazer um algoritmo para:
 - definir uma classe para tratar hora, minutos e segundos;
 - ler duas indicações de tempo diferentes;
 - calcular e imprimir a diferença entre os dois.
2. Fazer um algoritmo para:
 - definir uma classe para representar um número complexo na forma cartesiana;
 - ler dois complexos diferentes;
 - calcular e imprimir:
 - a soma
 - a diferença
 - o produto
 - o quociente
 - a norma
 - o ângulo
3. Fazer um algoritmo para:
 - definir uma classe para representar um número complexo na forma polar;
 - ler dois números complexos diferentes;
 - calcular e imprimir:
 - a soma
 - a diferença
 - o produto
4. Fazer um algoritmo para:
 - definir uma classe para tratar informações de um aluno:
 - número
 - nome
 - número de disciplinas cursadas no semestre
 - lista com até 10 disciplinas
 - gravar um arquivo com informações de uma turma;
 - o último aluno, que não entrará no arquivo, tem número igual a zero.
5. Fazer um algoritmo para:
 - ler o arquivo do problema anterior;
 - calcular e imprimir:
 - quantos alunos matricularam em mais de 03 disciplinas;
 - o maior número de disciplinas cursadas;
 - o nome dos alunos com o maior número de disciplinas cursadas.
6. Fazer um algoritmo para:
 - inserir mais 05 alunos na turma pela ordem do seu número.
7. Fazer um algoritmo para:
 - copiar o arquivo anterior, removendo 02 alunos cujos números serão lidos pelo teclado.
8. Fazer um algoritmo para:
 - ler o arquivo anterior e gravar um outro arquivo, indexando-o por ordem alfabética.
9. Fazer um algoritmo para:
 - ler o arquivo indexado gerado no problema anterior e acrescentar mais 03 alunos na ordem alfabética.
10. Fazer um algoritmo para:
 - ler o arquivo indexado gerado no problema anterior e remover 07 alunos cujos números serão fornecidos do teclado.