

Tema: Introdução à programação VI  
Atividade: Classes

#### INSTRUÇÕES:

- Desenvolver classes/métodos em Java para fazer funcionar o programa abaixo.
- Providenciar a documentação essencial:  
nome e matrícula,  
identificação, objetivo, parâmetros e condições especiais,  
se houver, e relatório de testes (exemplos de valores usados e condições testadas).

SUGESTÃO: Montar um menu para a escolha do método a ser testado  
(ver modelo em Lista00.java).

Testes deverão ser realizados e os valores usados deverão  
ser guardados no final do programa como comentários (`/* e */`).

O uso de recursão é opcional; se desejar utilizá-lo,  
fazer também a implementação da forma não-recursiva.

O tratamento de erro será feito pelas classes descritas abaixo,  
armazenadas em arquivos distintos.

```

// Lista de dependencias

/*
  Modelo da classe abstrata <AMyError> (UML)

// Atributo(s):

// Metodos

// Getters
+ int getError ( )

// Setters
+ void setError ( int codigo )

// Testers
+ boolean hasError ( )

*/
/**
  Classe abstrata para tratar erro.
*/
abstract class AMyError
{
  /**
    Funcao para testar
    se ha' erro.
    @return true, se houver;
    false, caso contrario
  */
  abstract public boolean hasError ( );

  /**
    Funcao para obter
    codigo de erro.
    @return codigo de erro
  */
  abstract public int getError ( );

  /**
    Metodo para guardar
    codigo de erro.
    @param codigo a ser guardado
  */
  abstract protected void setError ( int codigo );
} // end abstract class

```

```
/*  
    Modelo da interface <<IMyError>> (UML)
```

```
// Metodos
```

```
// Getters
```

```
+ String getErrorMsg ( )
```

```
*/
```

```
/**
```

```
    Interface para tratar mensagem de erro.
```

```
*/
```

```
interface IMyError
```

```
{
```

```
/*
```

```
    Codigos de erro:
```

```
0 - "[Erro]: Nao ha' erro."
```

```
1 -
```

```
2 -
```

```
3 -
```

```
*/
```

```
    public String getErrorMsg ( );
```

```
} // end interface
```

```
/*
    Modelo da classe MyError (UML)

// derivação herança interface
    MyError : <AMyError> <<IMyError>>

// Atributo(s):

- int erro

// Metodos

// Construtor
+ MyError ( )           // padrao

// Acesso
// Entrada e saida
+ String toString ( )   // compatibilidade

// Getters
+ int getError ( )

// Setters
+ void setError ( int codigo )

// Testers
+ boolean hasError ( )
```

```

*/
/**
    Classe para tratar erro.
*/
public class MyError
    extends AMyError    // heranca
    implements IMyError // interface
{
    /**
        Constante da classe.
    */
    public final static String NO_ERROR = "No_error";

    /**
        Atributo para guardar
        codigo de erro.
    */
    protected int error;

    /**
        Atributo para guardar
        codigo de erro.
    */
    public MyError ( )
    {
        setError ( 0 );    // valor inicial
    } // end construtor

    /**
        Funcao para obter
        codigo de erro.
        @return codigo de erro
    */
    public String toString ( )
    {
        return ( ""+error );
    } // end toString ( )

    /**
        Funcao para obter
        codigo de erro.
        @return codigo de erro
    */
    public int getError ( )
    {
        return ( error );
    } // end getErro ( )

```

```

/**
    Metodo para guardar
    codigo de erro.
    @param codigo a ser guardado
 */
protected void setError ( int codigo )
{
    error = codigo;
} // end setError ( )

/**
    Funcao para testar
    se ha' erro.
    @return true, se houver;
            false, caso contrario
 */
public boolean hasError ( )
{
    return ( error != 0 );
} // end hasErro ( )

/**
    Funcao para implementar
    o metodo da interface IMyError.
    @return mensagem sobre o erro
 */
public String getErrorMsg ( )
{
    return ( NO_ERROR );
} // end getErrorMsg ( )

} // end class

```

```

// Lista de dependencias
import java.io.*;

/**
 * Classe para tratar cadeias de caracteres.
 *
 * @author PUCMG
 * @version 99/99/9999 12:00:00 (v0.5)
 *
 * Modelo de classe
 * class MyString : MyError << IMyError>>
 * - int error // tratamento de erro
 * + boolean hasError ( )
 * + int getError ( )
 * + void setError ( int codigo )
 * -----
 * # protected String cadeia // atributo(s)
 * -----
 * + MyString EOL = "\n"; // constantes da classe
 * + MyString EMPTY = "";
 * -----
 * + String read ( String msg ) // metodos da classe
 * -----
 * + MyString ( ) // construtor(es)
 * + MyString ( int tamanho )
 * -----
 * + String toString ( ) // compatibilidade
 * + MyString clone ( )
 * -----
 * + int length ( ) // acesso
 * + boolean isEmpty ( )
 * + String get ( )
 * + void set ( String x )
 * + char getchar ( int position )
 * + void setchar ( int position, char valor )
 *
 * + Object getHead ( ) // servicos
 * + MyArray getTail ( )
 *
 * // operacoes
 * + void append ( String x )
 * + void prepend ( String x )
 * + boolean equals ( MyString x )
 * + int compareTo ( MyString x )
 * -----
 * + void tests ( ) // testes
 * -----
 */

```

```

public class MyString
    extends MyError
    implements IMyError
{
    // ----- tratamento de erro

    /*
    Codigos de erros na classe MyArray:

    0. Nao ha' erro.
    1. Tabela vazia.
    2. Tamanho invalido.
    3. Posicao invalida ao consultar.
    4. Posicao invalida ao atribuir.
    */
    // implementacao obrigatoria de IMyError
    // @Override // anotacao para sobrepor metodo
    public String getErrorMsg ( )
    {
        String txt = "[MyString]: ";
        switch ( getError( ) )
        {
            case 0:
                txt = txt+"No errors.";
                break;
            default:
                txt = txt+"Undefined error.";
        } // end switch
        return ( txt );
    } // end getErrorMsg ( )

    // constante(s) da classe
    /**
    * Constante da classe.
    */
    public static final MyString EMPTY = new MyString ( "" );

    // atributo(s)
    /**
    * Armazenador para cadeia de caracteres.
    */
    private String cadeia = new String ( "" );

```



```

// construtor(es)
/**
 * Construtor padrao.
 */
public MyString ( )
{
    // armazenador inicialmente vazio
} // end constructor

/**
 * Construtor alternativo.
 * @param x - valor inicial
 */
public MyString ( String x )
{
} // end constructor

/**
 * Mostrar cadeia de caracteres automaticamente.
 * @return conteudo da cadeia
 */
public String toString ( )
{
    String msg = "";

    return ( msg );
} // fim toString( )

/**
 * Clonar cadeia de caracteres.
 * @return copia do conteudo da cadeia
 */
public MyString clone ( )
{
    MyString copia = new MyString ( );

    return ( copia );
} // end clone( )

```

```

// acesso
/**
 * Funcao para testar
 * se armazenador vazio.
 * @return true, se vazio;
 *         false, caso contrario
 */
public boolean isEmpty ( )
{
    return ( true );
} // end isEmpty ( )

/**
 * Funcao para obter
 * o tamanho da cadeia de caracteres armazenada.
 * @return tamanho do conteudo da cadeia, se houver dados;
 *         zero, caso contrario
 */
public int length ( )
{
    // definir dados
    int x = 0;

    // retornar
    return ( x );
} // end length ( )

/**
 * Funcao para obter
 * o valor armazenado.
 * @return conteudo da cadeia, se houver dados;
 *         vazio, caso contrario
 */
public String get ( )
{
    return ( cadeia );
} // end get ( )

/**
 * Metodo para
 * armazenar valor.
 * @param novo conteudo para a cadeia
 */
public void set ( String x )
{
    cadeia = x;
} // end set ( )

```

```

/**
 * Funcao para obter
 * caractere de posicao valida.
 * @return caractere da posicao indicada ou
 *         espaco, caso contrario
 * @param posicao do caractere desejado
 */
public char getCharAt ( int position )
{
    // definir dado
    char x = ' ';

    // retornar
    return ( x );
} // end getCharAt ( )

/**
 * Metodo para alterar
 * caractere em posicao valida.
 * @param position - local do caractere a ser alterado
 * @param value    - novo valor
 */
public void setCharAt ( int position, char value )
{
} // end setCharAt ( )

/**
 * Funcao para obter
 * o primeiro caractere armazenado.
 * @return primeiro caractere da cadeia, se houver;
 *         vazio, caso contrario
 */
public String getHead ( )
{
    // definir dados
    String x = "";

    // retornar
    return ( x );
} // end getHead ( )

```

```

/**
 * Funcao para obter
 * o restante do conteudo armazenado,
 * @return restante da cadeia, se houver;
 *      vazio, caso contrario
 */
public MyString getTail ( )
{
    // definir dados
    MyString x = new MyString ( );

    // retornar
    return ( x );
} // end getTail ( )

/**
 * Funcao para remover espacos em branco.
 * @return cadeia de caracteres sem espacos em branco, se houver;
 *      vazio, caso contrario
 */
public MyString trim ( )
{
    // definir dados
    MyString x = new MyString ( );
    int y;

    // retornar
    return ( x );
} // end trim ( )

// servicos
/**
 * Funcao para comparar
 * conteudo com o de outro objeto desta classe.
 * @return true, se iguais;
 *      false, caso contrario
 * @param valor com o qual testar
 */
public boolean equals ( MyString valor )
{
    // definir dados
    boolean result = false;

    // retornar
    return ( result );
} // end equals ( )

```

```

/**
 * Funcao para comparar
 * conteudo com o de outro objeto desta classe.
 * @return zero, se iguais;
 *         valor positivo, se primeiro maior que o segundo;
 *         valor negativo, se primeiro menor que o segundo
 */
public int compareTo ( MyString valor )
{
    // definir dados
    int x = 0;

    // retornar
    return ( x );
} // end compareTo ( )

```

```

/**
 * Funcao para concatenar
 * conteudo com o de outro objeto desta classe.
 * @return cadeia atual concatenada com outro valor,
 *         vazio caso contrario
 * @param x conteudo para concatenar
 */
public MyString append ( MyString valor )
{
    // definir dado
    MyString x = new MyString ( );

    // retornar
    return ( x );
} // end append ( )

```

```

/**
 * Funcao para concatenar
 * conteudo de outro objeto com o desta classe.
 * @return outro valor concatenado com cadeia atual,
 *         vazio caso contrario
 * @param x conteudo para concatenar
 */
public MyString prepend ( MyString valor )
{
    // definir dado
    MyString x = new MyString ( );

    // retornar
    return ( x );
} // end prepend ( )

```

```

// testes
/**
 * Testes.
 */
public static void tests ( )
{
    // 01. definir dados
    MyString ms1 = new MyString ( );
    MyString ms2 = new MyString ( "de" );
    MyString ms3 = new MyString ( );
    MyString ms4 = null;
    MyString ms5 = null;
    MyString ms6 = new MyString ( "0" );
    MyString ms7 = new MyString ( "" );

    int c;

    // identificar
    System.out.println ( "Modulo de testes da classe MyString: " );
    System.out.println ( );

    // 02. testar atribuicao
    ms1.set ( " a bc " );

    // 03. testar concatenacao posterior
    ms3 = ms1.append ( ms2 );

    // 04. testar concatenacao anterior
    ms4 = ms1.prepend ( ms2 );

    // 05. testar clone
    ms5 = ms3.clone( );

    // 06. testar clone
    ms6.setCharAt( 0, '1' );

    // 07. testar exibicao
    System.out.println ( "01. ms1 = "+ms1.get( ) );
    System.out.println ( "02. ms2 = "+ms2.toString( ) );
    System.out.println ( "03. ms3 = ms1+ms2 = "+ms3 );
    System.out.println ( "04. ms4 = ms2+ms1 = "+ms4 );
    System.out.println ( "05. ms5 = ms3 = "      +ms5 );
    System.out.println ( "06. ms6 = "+ms6 );
    System.out.println ( "07. ms7 = "+ms7 );
}

```

```

// 08. testar existencia de dado
System.out.println ( "08. isEmpty (ms6) = "+ms6.isEmpty( ) );
System.out.println ( "09. isEmpty (ms7) = "+ms7.isEmpty( ) );
System.out.println ( "10. ms6 == ms6? "+(ms6.equals ( ms6 ) ) );
System.out.println ( "11. ms6 == ms6? "+(ms6.compareTo ( ms6 )==0) );
System.out.println ( "12. ms6 > ms7? "+(ms6.compareTo ( ms7 )> 0) );
System.out.println ( "13. ms5 > ms6? "+(ms5.compareTo ( ms6 )> 0) );

// 09. testar consumo unitario
System.out.println ( " " );
while ( ! ms5.isEmpty( ) )
{
    c = ms5.charAt( 0 );
    System.out.println ( "ms5 = "+ms5.getHead( )
        + " " + "+ms5.getTail( ).toString( )
        + " (" +(ms5.length( )-1)+")" );
    ms5 = ms5.getTail( );
} // end while

// 10. testar erros
System.out.println ( );
System.out.println ( "ms5 == \"\" -> "+ms5.isEmpty( ) );
System.out.println ( "ms5[ 0 ] = " +ms5.charAt( 0 ) + "" );
System.out.println ( "ms5[ 0 ] = \"\" +ms5.getHead( ) + "\"\" );
System.out.println ( "ms5[1:n] = " +ms5.getTail( ) );

} // end tests ( )

// acao principal
/**
 * Acao principal.
 */
public static void main ( String [ ] args )
{
    // testar
    MyString.tests ( );
} // end main ( )

} // end class

```

## SAÍDAS ESPERADAS

Modulo de testes da classe MyString:

```
01. ms1 =  a bc
02. ms2 = de
03. ms3 = ms1+ms2 =  a bc  de
04. ms4 = ms2+ms1 = de  a bc
05. ms5 = ms3 =  a bc  de
06. ms6 = 1
07. ms7 = ""
08. isEmpty (ms6) = false
09. isEmpty (ms7) = true
10. ms6 == ms6? true
11. ms6 == ms6? true
12. ms6 > ms7? true
13. ms5 > ms6? false
```

```
ms5 = +  a bc  de (11)
ms5 = + a bc  de (10)
ms5 = + a bc  de (9)
ms5 = a + bc  de (8)
ms5 = + bc  de (7)
ms5 = b + c  de (6)
ms5 = c +  de (5)
ms5 = +  de (4)
ms5 = + de (3)
ms5 = + de (2)
ms5 = d + e (1)
ms5 = e + "" (0)
```

```
ms5 == "" -> true
ms5[ 0 ] = ' '
ms5[ 0 ] = ""
ms5[1:n] = ""
```

```
ms5 =      a bc  de  a bc  de
first = a
first = bc
first = de
first = a
first = bc
first = de
```



## EXTRAS

- 01.) Acrescentar um método (e testes) para remover espaços em branco iniciais e finais ( ***trim***( ) ) .
- 02.) Acrescentar um método (e testes) para copiar uma subcadeia a partir de determinada posição ( ***substring*** ( int posição, int quantidade ) ) .