PUC-Minas - Ciência da Computação
AED1 – Estudo Dirigido 10

Tema: Introdução à programação V
Atividade: Grupos de dados homogêneos

01.) Editar e salvar um esboço de programa em Java:

```
/**
  * Exemplo0181
  *
  * @author
  * @version 01
  */

// ---------------------------------------------  dependencias

import IO.*;

// ---------------------------------------------  definicao de classe auxiliar

/**
 * Classe para tratar arranjo de inteiros.
 */
class Arranjo
{
 /**
  * armazenador generico dos dados
  */
 public Object [ ] tabela;

 /**
  * construtor padrao
  */
 public Arranjo ( )
 {
  tabela = null;
 } // fim construtor padrao

} // fim da classe Arranjo
```

```
// ------------------------------------------------  definicao da classe principal

public class Exemplo0181
{
// ------------------------------------------------  definicao de metodo auxiliar

  /**
    * Testar definições de arranjo usando classe.
    */
  public static void metodo01 ( )
  {
   // 1. definir dados
      Arranjo a1 = null;                  // nao existe objeto
      Arranjo a2 = new Arranjo ( ); // existe objeto (sem dados, no momento)
   // 2. identificar
      IO.println ( "Definicoes de arranjo" );
   // 3. testar as definicoes de arranjo
      if ( a1 == null )
      {
        IO.println ( “Arranjo a1 nulo” );
      }
      else
      {
        IO.println ( “Arranjo a1 nao nulo” );
      } // fim se
      if ( a2 == null )
      {
        IO.println ( “Arranjo a2 nulo” );
      }
      else
      {
        IO.println ( “Arranjo a2 nao nulo” );
      } // fim se
   // 5. encerrar
      IO.println ( );
      IO.pause ( "Apertar ENTER para continuar." );
    } // fim metodo01( )

// ------------------------------------------------  definicao do metodo principal

    /**
      * main() – metodo principal
      */
    public static void main ( String [ ] args )
    {
      // identificar
         IO.println ( "EXEMPLO0181 - Programa em Java" );
         IO.println ( "Autor: _____" );
      // executar o metodo auxiliar
         metodo01 ( );
      // encerrar
         IO.pause   ( “Apertar ENTER para terminar.” );
    } // fim main( )
} // fim class Exemplo0161
```

02.) Compilar o programa.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
Em caso de dúvidas, consultar a apostila, recorrer aos monitores ou apresentá-las ao professor.

03.) Executar o programa.

Observar as saídas.
Registrar os dados e os resultados.
Em caso de erro (ou dúvida), usar comentários para registrar a ocorrência e, posteriormente, tentar resolvê-lo (ou esclarecer a dúvida).

04.) Copiar a versão atual do programa para outra nova – Exemplo0182.java.

05.) Editar mudanças no nome do programa e versão.

Acrescentar novos métodos à classe para lidar com a quantidade de elementos.

```
/**
 * construtor alternativo.
 */
public Arranjo ( int tamanho )
{
 if ( tamanho <= 0 )
 {
  IO.println ( "ERRO: quantidade invalida." );
 }
 else
 {
  tabela = new Object [ tamanho ];
 } // fim se
} // fim construtor alternativo

/**
 * informar a quantidade de posicoes reservadas.
 */
public int length ( )
{
 int tamanho = 0;

 if ( tabela != null )
 {
  tamanho = tabela.length;
 }
 return ( tamanho );
} // fim length ( )
```

Na parte principal, criar um segundo método para testes.

```
// 1. definir dados
   Arranjo a1 = null;
   Arranjo a2 = new Arranjo ( );
   Arranjo a3 = new Arranjo ( 3 );

// 3. testar as definicoes de arranjo
   if ( a3 == null )
   {
     IO.println ( "Arranjo a3 nulo" );
   }
   else
   {
     IO.println ( "Arranjo a3 nao nulo com "+a3.length( )+" posicoes." );
   } // fim se
```

06.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

07.) Executar o programa.
Observar as saídas.
Registrar os dados e os resultados.

08.) Copiar a versão atual do programa para outra nova – Exemplo0183.java.

09.) Editar mudanças no nome do programa e versão.
Acrescentar método à classe para mostrar o conteúdo de arranjo na tela.

```
/**
  * exibir dados em tabela.
  */
  public void printArray ( )
 {
  // definir dados
     int  tamanho, posicao;
  // identificar
     IO.println ( );
  // testar se a tabela foi montada
     if ( tabela == null )
     {
       IO.println ( "ERRO: Tabela vazia." );
     }
     else
     {
      // obter o tamanho da tabela
        tamanho = length( );
      // mostrar tabela
        IO.println ( "Tabela com "+tamanho+" posicoes:" );
      // repetir para cada posicao na tabela
        for ( posicao = 0;
              posicao < tamanho;
              posicao = posicao + 1 )
        {
         // mostrar o valor armazenado
           IO.println ( "posicao = "+posicao+
                        " tem valor = "+tabela [ posicao ] );
        } // fim repetir
     } // fim se
 } // fim printArray ( )
```

Na parte principal, acrescentar um terceiro método para testar a exibição de dados em arranjo.

```
/**
 * Testar entrada e saida de dados em arranjo usando classe.
 */
public static void metodo03 ( )
{
 // 1. definir dados
   Arranjo a3 = new Arranjo ( 3 );
 // 2. identificar
   IO.println ( );
   IO.println ( "Entrada e saida em arranjo" );
   IO.println ( );
 // 3. testar entrada e saida em arranjo
   if ( a3 == null )
   {
     IO.println ( "Arranjo a3 nulo" );
   }
   else
   {
     // mostrar informacoes sobre arranjo
        IO.println ( "Arranjo a3 nao nulo com "+a3.length( )+" posicoes." );
     // mostrar dados no arranjo
        IO.println ( "Mostrar dados lidos e armazenados:" );
        a3.printArray ( );
   } // fim se
 // 4. encerrar
   IO.println ( );
   IO.pause ( "Apertar ENTER para continuar." );
 } // fim metodo03( )
```

OBS.:
Observar que todos os dados serão iguais a zero, pois esse é o valor padrão para inteiros.

10.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

11.) Executar o programa.
Observar as saídas.
Registrar os dados e os resultados.

12.) Copiar a versão atual do programa para outra nova – Exemplo0184.java.

13.) Editar mudanças no nome do programa e versão.
Acrescentar à classe um método para ler valores inteiros do teclado e guardá-los em arranjo.

```
/**
 * ler valores inteiros de arquivo e guardar em uma tabela.
 * @param message - com texto a ser mostrado na tela
 */
  public void readArray ( String message )
 {
  // definir dados
     int   posicao,
           tamanho = length( );
     String linha;

  // testar se quantidade valida
     if ( tamanho <= 0 )
     {
       IO.println ( "ERRO: Tamanho invalido." );
     }
     else
     {
      // mostrar mensagem antes de ler dados
        IO.println ( message );
      // obter o tamanho da tabela
        tamanho = this.length ( );
      // repetir para cada posicao na tabela
        for ( posicao = 0;
              posicao < tamanho;
              posicao = posicao + 1 )
        {
         // ler linha do teclado
            linha = IO.readln ( );
         // armazenar em um posicao da tabela
         // como objeto em String
            tabela [ posicao ] = linha;
        } // fim repetir
     } // fim se
 } // fim readArray ( )
```

Na parte principal, acrescentar um método para testar ler e exibir de dados em arranjo.

```
/**
 * Testar entrada e saida de dados em arranjo usando classe.
 */
public static void metodo04 ( )
{
 // 1. definir dados
   Arranjo a3 = new Arranjo ( 3 );
 // 2. identificar
   IO.println ( );
   IO.println ( "Entrada e saida em arranjo" );
   IO.println ( );
 // 3. testar entrada e saida em arranjo
   if ( a3 == null )
   {
     IO.println ( "Arranjo a3 nulo" );
   }
   else
   {
     // mostrar informacoes sobre arranjo
        IO.println ( "Arranjo a3 nao nulo com "+a3.length( )+" posicoes." );
     // ler dados e guardar no arranjo
        a3.readArray ( "Entrar com dados no arranjo:" );
     // mostrar dados no arranjo
        IO.println ( "Mostrar dados lidos e armazenados:" );
        a3.printArray ( );
   } // fim se
 // 4. encerrar
   IO.println ( );
   IO.pause ( "Apertar ENTER para continuar." );
 } // fim metodo04( )
```

OBS.:
Todas as posições serão lidas e mostradas.

14.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

15.) Executar o programa.
Observar as saídas.
Registrar os dados e os resultados.

16.) Copiar a versão atual do programa para outra nova – Exemplo0185.java.

17.) Editar mudanças no nome do programa e versão.
Acrescentar outro método para ler certa quantidade de dados e guardar em arranjo.

```
/**
 * ler valores inteiros de arquivo e guardar em uma tabela.
 * @param message - com texto a ser mostrado na tela
 * @param n - quantidade de dados a serem lidos
 */
  public void readIntArray ( String message, int n )
 {
  // definir dados
    int   posicao,
          tamanho = length( );
    String linha;

  // testar se quantidade valida
     if ( tamanho <= 0 ||
          n <= 0 || n > tamanho )
    {
      IO.println ( "ERRO: Quantidade invalida." );
    }
    else
    {
     // mostrar mensagem antes de ler dados
       IO.println ( message );
     // repetir para cada posicao na tabela
       for ( posicao = 0;
              posicao < n;
             posicao = posicao + 1 )
       {
        // ler linha do teclado
           linha = IO.readln ( );
        // armazenar em um posicao da tabela
        // valor convertido para inteiro
           tabela [ posicao ] = IO.getint ( linha );
       } // fim repetir
     } // fim se
} // fim readIntArray ( )
```

Na parte principal, acrescentar um método para testar ler e exibir de dados em arranjo.

```
/**
 * Testar entrada e saida de dados em arranjo usando classe.
 */
public static void metodo05 ( )
{
 // 1. definir dados
   Arranjo a3 = new Arranjo ( 3 );
 // 2. identificar
   IO.println ( );
   IO.println ( "Entrada e saida em arranjo" );
   IO.println ( );
 // 3. testar entrada e saida em arranjo
   if ( a3 == null )
   {
     IO.println ( "Arranjo a3 nulo" );
   }
   else
   {
     // mostrar informacoes sobre arranjo
        IO.println ( "Arranjo a3 nao nulo com "+a3.length( )+" posicoes." );
     // ler dados e guardar no arranjo
         a3.readIntArray ( "Entrar com dados no arranjo:", 2 );
     // mostrar dados no arranjo
        IO.println ( "Mostrar dados lidos e armazenados:" );
        a3.printArray ( );
   } // fim se
 // 4. encerrar
   IO.println ( );
   IO.pause ( "Apertar ENTER para continuar." );
 } // fim metodo05( )
```

OBS.:
Notar que a última posição será zero, porque seu valor original não foi alterado.

18.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

19.) Executar o programa.
Observar as saídas.
Registrar os dados e os resultados.

20.) Copiar a versão atual do programa para outra nova – Exemplo0186.java.

21.) Editar mudanças no nome do programa e versão.
Acrescentar outro método para mostrar apenas certa quantidade de dados guardada em arranjo.

```
/**
  * exibir certa quantidade de dados em tabela.
  * @param n - quantidade de dados a serem mostrados
  */
  public void printIntArray ( int n )
 {
  // definir dados
    int posicao,
        tamanho = length( );
  // identificar
    IO.println ( );
  // testar se a tabela foi montada
     if ( tabela == null ||
          n <= 0 || n > tamanho )
    {
      IO.println ( "ERRO: Tabela vazia ou quantidade invalida." );
    }
    else
    {
     // mostrar tabela
       IO.println ( "Tabela com "+tamanho+" posicoes:" );
     // repetir para cada posicao na tabela
       for ( posicao = 0;
             posicao < n;
            posicao = posicao + 1 )
       {
        // mostrar o valor armazenado
          IO.println ( "posicao = "+posicao+
                       " tem valor = "+(int) tabela [ posicao ] );
       } // fim repetir
    } // fim se
 } // fim printIntArray ( )
```

Na parte principal, acrescentar um método para testar ler e exibir de dados em arranjo.

```
/**
 * Testar entrada e saida de dados em arranjo usando classe.
 */
public static void metodo06 ( )
{
 // 1. definir dados
   Arranjo a3 = new Arranjo ( 3 );
 // 2. identificar
   IO.println ( );
   IO.println ( "Entrada e saida em arranjo" );
   IO.println ( );
 // 3. testar entrada e saida em arranjo
   if ( a3 == null )
   {
     IO.println ( "Arranjo a3 nulo" );
   }
   else
   {
     // mostrar informacoes sobre arranjo
        IO.println ( "Arranjo a3 nao nulo com "+a3.length( )+" posicoes." );
     // ler dados e guardar no arranjo
        a3.readIntArray ( "Entrar com dados no arranjo:", 2 );
     // mostrar dados no arranjo
        IO.println ( "Mostrar dados lidos e armazenados:" );
        a3.printIntArray ( 2 );
   } // fim se
 // 4. encerrar
   IO.println ( );
   IO.pause ( "Apertar ENTER para continuar." );
 } // fim metodo06( )
```

22.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

23.) Executar o programa.
Observar as saídas.
Registrar os dados e os resultados.

24.) Copiar a versão atual do programa para outra nova – Exemplo0187.java.

25.) Editar mudanças no nome do programa e versão.
Acrescentar um método para testar que a atribuição tornará arranjos idênticos.

```
/**
 * Testar entrada e saida de dados em arranjo usando classe.
 */
public static void metodo07 ( )
{
 // 1. definir dados
   Arranjo a2 = null;
   Arranjo a3 = new Arranjo ( 3 );
 // 2. identificar
   IO.println ( );
   IO.println ( "Entrada e saida em arranjo" );
   IO.println ( );
 // 3. testar entrada e saida em arranjo
   if ( a3 == null )
   {
     IO.println ( "Arranjo a3 nulo" );
   }
   else
   {
     // mostrar informacoes sobre arranjo
       IO.println ( "Arranjo a3 nao nulo com "+a3.length( )+" posicoes." );
     // ler dados e guardar no arranjo
       a3.readIntArray ( "Entrar com dados no arranjo:", 2 );
     // tornar arranjos identicos
        a2 = a3;
     // mostrar dados no arranjo
       IO.println ( "Mostrar dados lidos e armazenados:" );
       a2.printIntArray ( 2 );
     // ler dados e guardar no arranjo
        a2.readIntArray ( "Entrar com dados no arranjo:", 3 );
     // mostrar dados no arranjo original
       IO.println ( "Mostrar dados lidos e armazenados:" );
        a3.printIntArray ( 3 );
     // mostrar dados no arranjo copiado
       IO.println ( "Mostrar dados lidos e armazenados:" );
       a2.printIntArray ( 3 );
   } // fim se
 // 4. encerrar
   IO.println ( );
   IO.pause ( "Apertar ENTER para continuar." );
 } // fim metodo07( )
```

OBS.:
Notar que por serem idênticos, a alteração feita em um afetará o outro. Não são cópias !

26.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

27.) Executar o programa.
Observar as saídas.
Registrar os dados e os resultados.

28.) Copiar a versão atual do programa para outra nova – Exemplo0188.java.

29.) Editar mudanças no nome do programa e versão.
Acrescentar um método para copiar dados de um arranjo para outro.

```
/**
 * clonar tabela.
 * @return nova tabela com dados copiados
 */
 public Arranjo clone ( )
{
 // definir dados
   int tamanho, posicao;
   Arranjo nova = null;

 // testar existencia de dados
   if ( tabela == null )
   {
     IO.println ( "ERRO: Tabela vazia." );
   }
   else
   {
    // obter o tamanho da tabela original
      tamanho = length( );
    // reservar espaco para a nova tabela
      nova = new Arranjo ( tamanho );
    // testar a existencia de dados
      if ( nova == null )
      {
        IO.println ( "ERRO: Nao ha' espaco." );
      }
      else
      {
       // repetir para cada posicao na tabela original
          for ( posicao = 0;
                posicao < nova.length( );
                posicao = posicao + 1 )
          {
           // copiar dado de uma posicao
             nova.tabela [ posicao ] = tabela [ posicao ];
          } // fim repetir
      } // fim se
   } // fim se

 // retornar nova tabela
   return ( nova );
} // fim clone ( )
```

Na parte principal, acrescentar um método para testar ler e exibir de dados em arranjos.

```
/**
 * Testar entrada e saida de dados em arranjo usando classe.
 */
public static void metodo08 ( )
{
 // 1. definir dados
    Arranjo a2 = null;
    Arranjo a3 = new Arranjo ( 3 );
 // 2. identificar
    IO.println ( );
    IO.println ( "Entrada e saida em arranjo" );
    IO.println ( );
 // 3. testar entrada e saida em arranjo
    if ( a3 == null )
    {
      IO.println ( "Arranjo a3 nulo" );
    }
    else
    {
      // mostrar informacoes sobre arranjo
         IO.println ( "Arranjo a3 nao nulo com "+a3.length( )+" posicoes." );
      // ler dados e guardar no arranjo
         a3.readArray ( "Entrar com dados no arranjo:", 3 );
      // tornar arranjos identicos
          a2 = a3.clone ( );
      // mostrar dados no arranjo
         IO.println ( "Mostrar dados lidos e armazenados:" );
         a2.printIntArray ( 3 );
      // ler dados e guardar no arranjo
         a2.readIntArray ( "Entrar com dados no arranjo:", 3 );
      // mostrar dados no arranjo original
         IO.println ( "Mostrar dados lidos e armazenados:" );
         a3.printArray ( );
      // mostrar dados no arranjo copiado
         IO.println ( "Mostrar dados lidos e armazenados:" );
         a2.printIntArray ( 3 );
    } // fim se
 // 4. encerrar
    IO.println ( );
    IO.pause ( "Apertar ENTER para continuar." );
  } // fim metodo08( )
```

OBS.:
Notar que a cópia de dados preservará a individualidade de cada arranjo.

30.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

31.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

32.) Copiar a versão atual do programa para outra nova – Exemplo0189.java.

33.) Editar mudanças no nome do programa e versão.
Acrescentar um método para copiar certa quantidade de dados em arranjo.

```
/**
 * copiar certa quantidade de dados em tabela.
 * @return nova tabela com dados copiados
 * @param n - quantidade de dados
 */
 public Arranjo copyArray ( int n )
{
// definir dados
   int posicao,
       tamanho = length( );
   Arranjo nova = null;

// testar existencia de dados
   if ( tabela == null ||
        n <= 0 || n > tamanho )
   {
     IO.println ( "ERRO: Tabela vazia." );
   }
   else
   {
    // reservar espaco para a nova tabela
      nova = new Arranjo ( n );
    // testar a existencia de dados
      if ( nova == null )
      {
        IO.println ( "ERRO: Nao ha' espaco." );
      }
      else
      {
       // repetir para cada posicao na tabela original
          for ( posicao = 0;
                posicao < nova.length( );
                posicao = posicao + 1 )
          {
           // copiar dado de uma posicao
             nova.tabela [ posicao ] = tabela [ posicao ];
          } // fim repetir
      } // fim se
   } // fim se

 // retornar nova tabela
    return ( nova );
} // fim copyArray ( )
```

Na parte principal, acrescentar um método para testar ler e exibir de dados em arranjos.

```
/**
 * Testar entrada e saida de dados em arranjo usando classe.
 */
public static void metodo09 ( )
{
 // 1. definir dados
   Arranjo a2 = null;
   Arranjo a3 = new Arranjo ( 3 );
 // 2. identificar
   IO.println ( );
   IO.println ( "Entrada e saida em arranjo" );
   IO.println ( );
 // 3. testar entrada e saida em arranjo
   if ( a3 == null )
   {
     IO.println ( "Arranjo a3 nulo" );
   }
   else
   {
     // mostrar informacoes sobre arranjo
        IO.println ( "Arranjo a3 nao nulo com "+a3.length( )+" posicoes." );
     // ler dados e guardar no arranjo
        a3.readIntArray ( "Entrar com dados no arranjo:", 3 );
     // tornar arranjos identicos
         a2 = a3.copyArray( 3 );
     // mostrar dados no arranjo copiado
        IO.println ( "Mostrar dados lidos e armazenados:" );
        a2.printIntArray ( 3 );
     // mostrar dados no arranjo original
        IO.println ( "Mostrar dados lidos e armazenados:" );
        a3.printIntArray ( 3 );
   } // fim se
 // 4. encerrar
   IO.println ( );
   IO.pause ( "Apertar ENTER para continuar." );
 } // fim metodo09( )
```

34.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

35.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

36.) Copiar a versão atual do programa para outra nova – Exemplo0190.java.

37.) Editar mudanças no nome do programa e versão.
Acrescentar um método à classe para exibição automática de todos os dados em arranjo.

```
/**
  * exibir dados em tabela automaticamente.
  */
  public String toString ( )
 {
  // definir dados
    String msg = null;
    int   posicao,
          tamanho;
  // testar se a tabela foi montada
    if ( tabela != null )
    {
     // obter o tamanho da tabela
       tamanho = length( );
     // repetir para cada posicao na tabela
        msg = "";
        for ( posicao = 0;
              posicao < tamanho;
              posicao = posicao + 1 )
       {
        // guardar valor armazenado
            msg = msg + " " + tabela [ posicao ];
       } // fim repetir
    } // fim se
  // retornar valores armazenados
    return ( msg );
 } // fim toString ( )
```

Na parte principal, acrescentar um método para testar ler e exibir de dados em arranjos.

```
/**
 * Testar entrada e saida de dados em arranjo usando classe.
 */
public static void metodo10 ( )
{
 // 1. definir dados
    Arranjo a2 = null;
    Arranjo a3 = new Arranjo ( 3 );
 // 2. identificar
    IO.println ( );
    IO.println ( "Entrada e saida em arranjo" );
    IO.println ( );
 // 3. testar entrada e saida em arranjo
    // ler dados e guardar no arranjo
       a3.readIntArray ( "Entrar com dados no arranjo:" );
    // tornar arranjos identicos
       a2 = a3.copyArray( 2 );
    // mostrar dados no arranjo copiado
       IO.println ( "Mostrar dados lidos e armazenados:" );
        IO.println ( ""+a2 );
    // mostrar dados no arranjo original
       IO.println ( "Mostrar dados lidos e armazenados:" );
        IO.println ( ""+a3 );
 // 4. encerrar
    IO.println ( );
    IO.pause ( "Apertar ENTER para continuar." );
 } // fim metodo10( )
```

38.) Compilar o programa novamente.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.

39.) Executar o programa.

Observar as saídas.
Registrar os dados e os resultados.

Exercícios:
DICAS GERAIS: Consultar o Anexo Java 02 na apostila para outros exemplos.
Prever, realizar e registrar todos os testes efetuados.

01.) Fazer um programa (Exemplo0191) para acrescentar um método à classe Arranjo
para ler certa quantidade de dados do teclado e armazenar no arranjo
a partir de em determinada posição.
Incluir um método para testar essa nova característica.
DICA: Testar se a posição inicial e a quantidade são válidas.

Exemplo: arranjo.readIntArray (“Arranjo 1:”, inicio, quantidade);

02.) Fazer um programa (Exemplo0192) para acrescentar um método à classe Arranjo
para mostrar certa quantidade de dados armazenados no arranjo
a partir de em determinada posição.
Incluir um método para testar essa nova característica.
DICA: Testar se a posição inicial e a quantidade são válidas.

Exemplo: arranjo.printIntArray (inicio, quantidade);

03.) Fazer um programa (Exemplo0193) para acrescentar um método à classe Arranjo
para ler dados de arquivo, dado o nome do mesmo, e armazenar em arranjo.
Incluir um método para testar essa nova característica.
DICA: Ler o tamanho também do arquivo e reservar o tamanho de acordo.

Exemplo: arranjo.fromFile (“Arquivo1.txt”);

04.) Fazer um programa (Exemplo0194) para acrescentar um método à classe Arranjo
para gravar dados de arranjo em arquivo, dado o nome do mesmo.
Incluir um método para testar essa nova característica.
DICA: Gravar o tamanho também do arquivo, primeiro, antes dos outros dados.

Exemplo: arranjo.toFile (“Arquivo2.txt”);

05.) Fazer um programa (Exemplo0195) para acrescentar um método à classe Arranjo
para ler certa quantidade de dados de arquivo, dado o nome do mesmo, e armazenar em arranjo.
Incluir um método para testar essa nova característica.
DICA: Ler o tamanho também do arquivo e reservar o tamanho de acordo,
apenas se a quantidade for válida.

Exemplo: arranjo.fromFile (“Arquivo1.txt”, quantidade);

06.) Fazer um programa (Exemplo0196) para acrescentar um método à classe Arranjo
para gravar certa quantidade de dados de arranjo em arquivo, dado o nome do mesmo.
Incluir um método para testar essa nova característica.
DICA: Gravar o tamanho também do arquivo, primeiro, antes dos outros dados,
apenas se a quantidade for válida.

Exemplo: arranjo.toFile (“Arquivo3.txt”, quantidade);

07.) Fazer um programa (Exemplo0197) para acrescentar um método à classe Arranjo
para ler certa quantidade de dados de arquivo, dado o nome do mesmo,
e armazenar em arranjo, a partir de determinada posição deste.
Incluir um método para testar essa nova característica.
DICA: Ler o tamanho também do arquivo e reservar o tamanho de acordo,
apenas se a posição inicial e a quantidade forem válidas.

Exemplo: arranjo.fromFile (“Arquivo1.txt”, inicio, quantidade);

08.) Fazer um programa (Exemplo0198) para acrescentar um método à classe Arranjo
para gravar certa quantidade de dados de arranjo em arquivo, dado o nome do mesmo,
a partir de determinada posição deste.
Incluir um método para testar essa nova característica.
DICA: Gravar o tamanho também do arquivo, primeiro, antes dos outros dados,
apenas se a quantidade for válida.

Exemplo: arranjo.toFile (“Arquivo4.txt”, inicio, quantidade);

09.) Fazer um programa (Exemplo0199) para acrescentar uma função à classe Arranjo
para inverter a ordem dos dados armazenados em arranjo
e retornar um novo arranjo com a ordem invertida.
Incluir um método para testar essa nova característica.
DICA: Testar se a posição inicial e a quantidade são válidas.

Exemplo: arranjo2 = arranjo1.invertArray ( );

10.) Fazer um programa (Exemplo0200) para acrescentar uma função à classe Arranjo
para copiar certa quantidade de dados armazenados no arranjo
a partir de em determinada posição.
Incluir um método para testar essa nova característica.
DICA: Testar se a posição inicial e a quantidade são válidas.

Exemplo: arranjo2 = arranjo1.copyArray (inicio, quantidade);

Tarefas extras

E1.) Fazer um programa para acrescentar uma função à classe Arranjo
para comparar se dois arranjos são iguais.
DICA: Testar, primeiro, se os tamanhos são iguais, antes de testar o resto.
Supor que seus elementos possam ser convertidos para **String**.

Exemplo: boolean resposta = arranjo1.equals (arranjo2);

E2.) Fazer um programa para acrescentar uma função à classe Arranjo
para clonar um arranjo.

Exemplo: arranjo2 = arranjo1.clone ( );