

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

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

```
/**
 * Exemplo0141
 *
 * @author
 * @version 01
 */

// ----- dependencias

import IO.*;

// ----- definicao da classe principal

public class Exemplo0141
{
// ----- definicao de metodo auxiliar

/**
 * ler valores inteiros de arquivo e guardar em arranjo.
 * @return tabela com os valores lidos de arquivo
 * @param nome do arquivo com os dados
 */
    public static int [ ] lerDoArquivo ( String nome )
    {
        // definir dados
        FILE arquivo = new FILE ( FILE.INPUT, nome );
        int tamanho;
        int x;
        int [ ] tabela = null;
        String linha;

        // identificar
        IO.println ( "Montar arranjo com valores lidos de arquivo" );
        // tentar ler uma linha do arquivo
        linha = arquivo.readLine ( );
    }
}
```

```

// testar a disponibilidade de dados
if ( linha == null )
{
    IO.println ( "ERRO: Nao ha' dados no arquivo." );
}
else
{
    // tentar obter a quantidade de dados
    tamanho = IO.getint ( linha );
    // testar se quantidade valida
    if ( tamanho <= 0 )
    {
        IO.println ( "ERRO: Tamanho invalido." );
    }
    else
    {
        // reservar espaco para os dados
        tabela = new int [ tamanho ];
        // repetir para cada dado no arquivo
        for ( x = 0; x < tamanho; x = x + 1 )
        {
            // ler linha do arquivo
            linha = arquivo.readLine ( );
            // armazenar em uma posicao do arranjo
            // valor convertido para inteiro
            tabela [ x ] = IO.getint ( linha );
        } // fim for
    } // fim se
} // fim se
// fechar arquivo
arquivo.close ( );
// retornar dados lidos
return ( tabela );
} // fim lerDoArquivo ( )

/**
 * recuperar dados de arquivo.
 */
public static void teste01 ( )
{
    // 1. definir dados
    int [ ] tabela = null;

    // 2. identificar
    IO.println ( );
    IO.println ( "Recuperar dados de arquivo" );
    IO.println ( );

    // 3. montar dados
    tabela = lerDoArquivo ( "DADOS.TXT" );

```

```

// 4. testar a existencia de dados
if ( tabela == null )
{
    IO.println ( "ERRO: Arranjo nulo." );
}
else
{
    if ( tabela.length == 0 )
    {
        IO.println ( "ERRO: Arranjo vazio." );
    }
    else
    {
        IO.println ( "Arranjo montado com " +
                    tabela.length + " dados." );
    } // fim se
} // fim se

// 5. encerrar
IO.println ( );
IO.pause ( "Apertar ENTER para continuar." );
} // fim teste01 ( )

// ----- definicao do metodo principal

/**
 * main() – metodo principal
 */
public static void main ( String [ ] args )
{
    // identificar
    IO.println ( "EXEMPLO0141 - Programa em Java" );
    IO.println ( "Autor: _____" );
    // executar o metodo auxiliar
    teste01 ( );
    // encerrar
    IO.pause ( "Apertar ENTER para terminar." );
} // fim main( )
} // fim class Exemplo0141

```

OBS.:

O tamanho em linhas é dado como quantidade de grupos,
 enquanto em colunas é dado como quantidade de elementos em um grupo.
 O arquivo de dados deverá ser preparado de acordo com o seguinte formato:

```

4
1
2
3
4

```

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 – Exemplo0142.java.
- 05.) Editar mudanças no nome do programa e versão.
Acrescentar outro método para mostrar dados em arranjo.
Na parte principal, editar a chamada do método para o novo.

```
// ----- definicao de metodo auxiliar
```

```
/**
 * exibir dados em arranjo.
 * @param tabela - arranjo com dados
 */
public static void mostrar ( int [ ] tabela )
{
    // definir dados
    int tamanho;
    int x;
    // identificar
    IO.println ( );
    // testar se a arranjo foi montado
    if ( tabela == null )
    {
        IO.println ( "ERRO: Tabela vazia." );
    }
    else
    {
        // verificar se tamanho valido
        tamanho = tabela.length;
        if ( tamanho <= 0 )
        {
            IO.println ( "ERRO: Arranjo vazio." );
        }
        else
        {
            // mostrar arranjo
            IO.println ( "Arranjo montado com " +
                tamanho + " dados." );

            // repetir para cada dado no arranjo
            for ( x = 0; x < tamanho; x = x + 1 )
            {
                // mostrar dado em um posicao da arranjo
                // convertido para inteiro
                IO.print ( " " + tabela [ x ] );
            } // fim for
            // mudar de linha
            IO.println ( );
        } // fim se
    } // fim se
} // fim mostrar ( )
```

```

/**
 * recuperar e mostrar dados de arquivo.
 */
public static void teste02 ( )
{
    // 1. definir dados
    int [ ] tabela = null;

    // 2. identificar
    IO.println ( );
    IO.println ( "Recuperar e mostrar dados de arquivo" );
    IO.println ( );

    // 3. montar dados
    tabela = lerDoArquivo ( "DADOS.TXT" );

    // 4. testar a existencia de dados
    if ( tabela == null )
    {
        IO.println ( "ERRO: Arranjo nulo." );
    }
    else
    {
        if ( tabela.length == 0 )
        {
            IO.println ( "ERRO: Arranjo vazio." );
        }
        else
        {
            mostrar ( tabela );
        } // fim se
    } // fim se

    // 5. encerrar
    IO.println ( );
    IO.pause ( "Apertar ENTER para continuar." );
} // fim teste02 ( )

```

OBS.:

Só poderá ser mostrado o arranjo em que existir algum conteúdo (diferente de **null** = inexistência de dados).

- 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 – Exemplo0143.java.
- 09.) Editar mudanças no nome do programa e versão.
Acrescentar função para copiar e retornar dados de um arranjo.
Na parte principal, editar a chamada do método para o novo.

```
// ----- definicao de metodo auxiliar
```

```
/**
 * copiar dados em arranjo.
 * @return novo arranjo com dados copiados
 * @param tabela - arranjo com dados
 */
public static int [ ] copiar ( int [ ] tabela )
{
    // definir dados
    int tamanho;
    int x;
    int [ ] novo = null;

    // testar existencia de dados
    if ( tabela == null )
    {
        IO.println ( "ERRO: Arranjo vazio." );
    }
    else
    {
        // reservar espaco para novo arranjo de dados
        tamanho = tabela.length;
        novo = new int [ tamanho ];
        // testar se espaco reservado
        if ( novo == null )
        {
            IO.println ( "ERRO: Nao ha' espaco." );
        }
        else
        {
            // repetir para cada dado no arquivo
            for ( x = 0; x < tamanho; x = x + 1 )
            {
                // copiar cada posicao do arranjo
                novo[ x ] = tabela [ x ];
            } // fim for
        } // fim se
    } // fim se

    // retornar novo arranjo
    return ( novo );
} // fim copiar ( )
```

```

/**
 * recuperar e mostrar dados de arquivo.
 */
public static void teste03 ( )
{
    // 1. definir dados
    int [ ] tabela1 = null;
    int [ ] tabela2 = null;

    // 2. identificar
    IO.println ( );
    IO.println ( "Recuperar e mostrar dados de arquivo" );
    IO.println ( );

    // 3. montar dados
    tabela1 = lerDoArquivo ( "DADOS.TXT" );

    // 4. testar a existencia de dados
    if ( tabela1 == null )
    {
        IO.println ( "ERRO: Arranjo nulo." );
    }
    else
    {
        if ( tabela1.length == 0 )
        {
            IO.println ( "ERRO: Arranjo vazio." );
        }
        else
        {
            mostrar ( tabela1 );
            tabela2 = copiar ( tabela1 );
            IO.println ( );
            IO.println ( "Apos copiar:" );
            mostrar ( tabela2 );
        } // fim se
    } // fim se

    // 5. encerrar
    IO.println ( );
    IO.pause ( "Apertar ENTER para continuar." );
} // fim teste03 ( )

```

OBS.:

Se existir dados no arranjo original, espaço para mesma quantidade deverá ser reservado.

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 – Exemplo0144.java.

- 13.) Editar mudanças no nome do programa e versão.
Acrescentar outra função para somar e retornar dados em arranjos.
Na parte principal, editar a chamada do método para o novo.

```
// ----- definicao de metodo auxiliar
```

```
/**
 * somar dados em arranjos.
 * @return novo arranjo com dados somados
 * @param tabela1 - arranjo com dados
 * @param tabela2 - arranjo com dados
 */
public static int [ ] somar
    ( int [ ] tabela1,
      int [ ] tabela2 )
{
    // definir dados
    int tamanho;
    int x;
    int [ ] novo = null;

    // testar existencia de dados
    if ( tabela1 == null || tabela2 == null )
    {
        IO.println ( "ERRO: Arranjo vazio." );
    }
    else
    {
        // testar se tamanhos validos
        if ( tabela1.length == 0 ||
            tabela2.length == 0 )
        {
            IO.println ( "ERRO: Tamanho(s) invalido(s)." );
        }
        else
        {
            // reservar espaco na novo arranjo com dados
            tamanho = tabela1.length;
            // reservar espaco para os dados
            novo = new int [ tamanho ];
            // testar o espaco disponivel
            if ( novo == null )
            {
                IO.println ( "ERRO: Nao ha' espaco." );
            }
            else
            {
                // repetir para cada dado no arquivo
                for ( x = 0; x < tamanho; x = x + 1 )
                {
                    // somar dados em cada posicao do arranjo
                    novo [ x ] = tabela1 [ x ] + tabela2 [ x ];
                } // fim for
            } // fim se
        } // fim se
    } // fim se
}
```



```

// retornar novo arranjo
return ( novo );
} // fim somar ( )

/**
 * recuperar, somar e mostrar dados de arquivo.
 */
public static void teste04 ( )
{
// 1. definir dados
int [ ] tabela1 = null;
int [ ] tabela2 = null;
int [ ] tabela3 = null;

// 2. identificar
IO.println ( );
IO.println ( "Recuperar, somar e mostrar dados de arquivo" );
IO.println ( );

// 3. montar dados
tabela1 = lerDoArquivo ( "DADOS1.TXT" );
tabela2 = lerDoArquivo ( "DADOS2.TXT" );

// 4. testar a existencia de dados
if ( tabela1 == null || tabela2 == null )
{
IO.println ( "ERRO: Arranjo nulo." );
}
else
{
if ( tabela1.length == 0 ||
tabela2.length == 0 )
{
IO.println ( "ERRO: Tamanho invalido." );
}
else
{
IO.println ( "Somar arranjos: " );
mostrar ( tabela1 );
IO.println ( "\n+" );
mostrar ( tabela2 );
IO.println ( );
IO.println ( "Resultado:" );
tabela3 = somar ( tabela1, tabela2 );
mostrar ( tabela3 );
} // fim se
} // fim se

// 5. encerrar
IO.println ( );
IO.pause ( "Apertar ENTER para continuar." );
} // fim teste04 ( )

```

OBS.:

Só poderão ser somados arranjos com mesma quantidade de dados.

- 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 – Exemplo0145.java.
- 17.) Editar mudanças no nome do programa e versão.
Acrescentar outra função para somar e retornar dados em arranjos, escalados por constante.
Na parte principal, editar a chamada do método para o novo.

```
// ----- definicao de metodo auxiliar
```

```
/**
 * somar dados em arranjos.
 * @return novo arranjo com dados somados
 * @param tabela1 - arranjo com dados
 * @param constante - constante para escalar dados
 * @param tabela2 - arranjo com dados
 */
public static int [ ] somar
    ( int [ ] tabela1,
      int constante,
      int [ ] tabela2 )
{
    // definir dados
    int tamanho;
    int x;
    int [ ] novo = null;

    // testar existencia de dados
    if ( tabela1 == null || tabela2 == null )
    {
        IO.println ( "ERRO: Arranjo vazio." );
    }
    else
    {
        // testar se tamanhos validos
        if ( tabela1.length == 0 ||
            tabela2.length == 0 )
        {
            IO.println ( "ERRO: Tamanho(s) invalido(s)." );
        }
        else
        {
            // reservar espaco para o novo arranjo com dados
            tamanho = tabela1.length;
            // reservar espaco para os dados
            novo = new int [ tamanho ];
        }
    }
}
```

```

// testar o espaco disponivel
if ( novo == null )
{
    IO.println ( "ERRO: Nao ha' espaco." );
}
else
{
    // repetir para cada dado no arquivo
    for ( x = 0; x < tamanho; x = x + 1 )
    {
        // somar dados em cada posicao do arranjo
        novo [ x ] = tabela1 [ x ] + constante * tabela2 [ x ];
    } // fim for
} // fim se
} // fim se
} // fim se

// retornar novo arranjo
return ( novo );
} // fim somar ( )

/**
 * recuperar, somar e mostrar dados de arquivo.
 */
public static void teste05 ( )
{
    // 1. definir dados
    int [ ] tabela1 = null;
    int [ ] tabela2 = null;
    int [ ] tabela3 = null;

    // 2. identificar
    IO.println ( );
    IO.println ( "Recuperar, somar e mostrar dados de arquivo" );
    IO.println ( );

    // 3. montar dados
    tabela1 = lerDoArquivo ( "DADOS1.TXT" );
    tabela2 = lerDoArquivo ( "DADOS2.TXT" );

    // 4. testar a existencia de dados
    if ( tabela1 == null || tabela2 == null )
    {
        IO.println ( "ERRO: Arranjo nulo." );
    }
    else
    {
        {
            if ( tabela1.length == 0 ||
                tabela2.length == 0 )
            {
                IO.println ( "ERRO: Tamanho invalido." );
            }
        }
    }
}

```

```

else
{
    IO.println ( "Somar arranjos: " );
    mostrar ( tabela1 );
    IO.println ( "\n+" );
    mostrar ( tabela2 );
    IO.println ( );
    IO.println ( "Resultado:" );
    tabela3 = somar ( tabela1, -1, tabela2 );
    mostrar ( tabela3 );
} // fim se
} // fim se

// 5. encerrar
IO.println ( );
IO.pause ( "Apertar ENTER para continuar." );
} // fim teste05 ( )

```

OBS.:

Só poderão ser comparados os arranjos com mesma quantidade de dados.

- 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 – Exemplo0146.java.

- 21.) Editar mudanças no nome do programa e versão.
Acrescentar função para comparar dois arranjos.
Na parte principal, editar a chamada do método para o novo.

```
// ----- definicao de metodo auxiliar

/**
 * comparar dados em arranjos.
 * @return se arranjos iguais, ou nao
 * @param tabela1 - arranjo com dados
 * @param tabela2 - arranjo com dados
 */
public static boolean comparar
    ( int [ ] tabela1,
      int [ ] tabela2 )
{
    // definir dados
    boolean resposta = true;
    int tamanho;
    int x;

    // testar existencia de dados
    if ( tabela1 == null || tabela2 == null )
    {
        IO.println ( "ERRO: Arranjo vazio." );
    }
    else
    {
        // testar se tamanhos validos
        if ( tabela1.length == 0 ||
            tabela2.length == 0 )
        {
            IO.println ( "ERRO: Tamanho(s) invalido(s)." );
        }
        else
        {
            // repetir para cada posicao dos arranjos
            tamanho = tabela1.length;
            for ( x = 0; x < tamanho; x = x + 1 )
            {
                // comparar cada posicao dos arranjos
                resposta = resposta &&
                    ( tabela1 [ x ] == tabela2 [ x ] );
            } // fim for
        } // fim se
    } // fim se

    // retornar novo arranjo
    return ( resposta );
} // fim comparar ( )
```

```

/**
 * recuperar e comparar dados de arquivos.
 */
public static void teste06 ( )
{
    // 1. definir dados
    int [ ] tabela1 = null;
    int [ ] tabela2 = null;
    String nome1, nome2;
    boolean resposta;
    // 2. identificar
    IO.println ( );
    IO.println ( "Recuperar e comparar dados de arquivos" );
    IO.println ( );
    // 3. montar dados
    nome1 = IO.readString ( "Qual o nome do primeiro arquivo? " );
    nome2 = IO.readString ( "Qual o nome do segundo arquivo? " );
    tabela1 = lerDoArquivo ( nome1 );
    tabela2 = lerDoArquivo ( nome2 );
    // 4. testar a existencia de dados
    if ( tabela1 == null || tabela2 == null )
    {
        IO.println ( "ERRO: Arranjo nulo." );
    }
    else
    {
        if ( tabela1.length == 0 ||
            tabela2.length == 0 )
        {
            IO.println ( "ERRO: Tamanho invalido." );
        }
        else
        {
            IO.println ( "Somar arranjos: " );
            mostrar ( tabela1 );
            IO.println ( "\n+" );
            mostrar ( tabela2 );
            IO.println ( );
            IO.println ( "Resultado:" );
            resposta = comparar ( tabela1, tabela2 );
            if ( resposta )
            {
                IO.println ( "arranjos iguais." );
            }
            else
            {
                IO.println ( "arranjos diferentes." );
            }
        } // fim se
    } // fim se
} // fim se
// 5. encerrar
IO.println ( );
IO.pause ( "Apertar ENTER para continuar." );
} // fim teste06 ( )

```

OBS.:

Só poderão ser comparados os arranjos com mesma quantidade de dados.

- 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 – Exemplo0147.java.
- 25.) Editar mudanças no nome do programa e versão.
Acrescentar uma função para dizer se um arranjo é nulo
(possui todos os valores iguais a zero).
Na parte principal, incluir uma chamada para esse método.

```
// ----- definicao de metodo auxiliar

/**
 * comparar se dados em arranjos sao nulos.
 * @return se dados na arranjo sao iguais a zero, ou nao
 * @param tabela - arranjo com dados
 */
public static boolean eNulo
    ( int [ ] tabela )
{
    // definir dados
    boolean resposta = true;
    int tamanho;
    int x;

    // testar existencia de dados
    if ( tabela == null )
    {
        IO.println ( "ERRO: Arranjo vazio." );
    }
    else
    {
        // testar se tamanhos validos
        if ( tabela.length == 0 )
        {
            IO.println ( "ERRO: Tamanho invalido." );
        }
        else
        {
            // repetir para cada posicao dos arranjos
            tamanho = tabela.length;
            for ( x = 0; x < tamanho; x = x + 1 )
            {
                // comparar cada posicao do arranjo
                resposta = resposta && ( tabela [ x ] == 0 );
            } // fim for
        } // fim se
    } // fim se
    // retornar resultado
    return ( resposta );
} // fim eNulo ( )
```

```

/**
 * recuperar e comparar dados de arquivo.
 */
public static void teste07 ( )
{
    // 1. definir dados
    int [ ] tabela = null;
    String nome;
    boolean resposta;

    // 2. identificar
    IO.println ( );
    IO.println ( "Recuperar e comparar dados de arquivos" );
    IO.println ( );

    // 3. montar dados
    nome = IO.readString ( "Qual o nome do arquivo? " );
    tabela = lerDoArquivo ( nome );

    // 4. testar a existencia de dados
    if ( tabela == null )
    {
        IO.println ( "ERRO: Arranjo vazio." );
    }
    else
    {
        if ( tabela.length == 0 )
        {
            IO.println ( "ERRO: Tamanho invalido." );
        }
        else
        {
            IO.println ( "Testar se arranjo nulo: " );
            mostrar ( tabela );
            IO.println ( );
            IO.println ( "Resultado:" );
            resposta = eNulo ( tabela );
            if ( resposta )
            {
                IO.println ( "arranjo nulo." );
            }
            else
            {
                IO.println ( "arranjo nao e' nulo." );
            }
        } // fim se
    } // fim se
} // fim se

// 5. encerrar
IO.println ( );
IO.pause ( "Apertar ENTER para continuar." );
} // fim teste07 ( )

```

OBS.:

Só deverá ser verificado o arranjo que possuir dados (não ser vazio).

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 – Exemplo0148.java.
- 29.) Editar mudanças no nome do programa e versão.
Acrescentar uma função para dizer se um arranjo é positivo
(tem todos os dados maiores ou iguais a zero).
Na parte principal, incluir chamadas para esse método.

```
// ----- definicao de metodo auxiliar

/**
 * comparar se dados em arranjos sao positivos.
 * @return se arranjo igual 'a identidade
 * @param tabela - arranjo com dados
 */
public static boolean ePositivo
    ( int [ ] tabela )

{
    // definir dados
    boolean resposta = true;
    int tamanho;
    int x;

    // testar existencia de dados
    if ( tabela == null )
    {
        IO.println ( "ERRO: Arranjo vazio." );
    }
    else
    {
        // testar se tamanhos validos
        if ( tabela.length == 0 )
        {
            IO.println ( "ERRO: Tamanho invalido." );
        }
        else
        {
            // repetir para cada posicao do arranjo
            tamanho = tabela.length;
            for ( x = 0; x < tamanho; x = x + 1 )
            {
                // verificar cada posicao do arranjo
                if ( tabela [ x ] >= 0 )
                {
                    // se ha' valor positivo
                    resposta = resposta && ( tabela [ x ] >= 0 );
                } // fim se
            } // fim for
        } // fim se
    } // fim se
}
```

```

// retornar resultado
    return ( resposta );
} // fim ePositivo ( )

/**
 * recuperar e comparar dados de arquivo.
 */
public static void teste08 ( )
{
    // 1. definir dados
    int [ ] tabela = null;
    String nome;
    boolean resposta;

    // 2. identificar
    IO.println ( );
    IO.println ( "Recuperar e comparar dados de arquivos" );
    IO.println ( );

    // 3. montar dados
    nome = IO.readString ( "Qual o nome do arquivo? " );
    tabela = lerDoArquivo ( nome );

    // 4. testar a existencia de dados
    if ( tabela == null )
    {
        IO.println ( "ERRO: Arranjo vazio." );
    }
    else
    {
        if ( tabela.length == 0 )
        {
            IO.println ( "ERRO: Tamanho invalido." );
        }
        else
        {
            IO.println ( "Testar se arranjo positivo: " );
            mostrar ( tabela );
            IO.println ( );
            IO.println ( "Resultado:" );
            resposta = ePositivo ( tabela );
            if ( resposta )
            {
                IO.println ( "arranjo positivo." );
            }
            else
            {
                IO.println ( "arranjo nao e' positivo." );
            }
        } // fim se
    } // fim se
} // fim se

// 5. encerrar
IO.println ( );
IO.pause ( "Apertar ENTER para continuar." );
} // fim teste08 ( )

```

- 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 – Exemplo0149.java.
- 33.) Editar mudanças no nome do programa e versão.
Acrescentar uma função para dizer se um arranjo é o simétrico de outro.
Na parte principal, incluir chamadas para essa função.

```
// ----- definicao de metodo auxiliar

/**
 * comparar dados em arranjos.
 * @return se arranjos diferentes, ou nao
 * @param tabela1 - arranjo com dados
 * @param tabela2 - arranjo com dados
 */
public static boolean eSimetrico
    ( int [ ] tabela1, int [ ] tabela2 )
{
    // definir dados
    boolean resposta = true;
    int tamanho;
    int x;

    // testar existencia de dados
    if ( tabela1 == null || tabela2 == null )
    {
        IO.println ( "ERRO: Arranjo vazio." );
    }
    else
    {
        // testar se tamanhos validos
        if ( tabela1.length == 0 || tabela2.length == 0 )
        {
            IO.println ( "ERRO: Tamanho(s) invalido(s)." );
        }
        else
        {
            // repetir para cada posicao dos arranjos
            tamanho = tabela1.length;
            for ( x = 0; x < tamanho; x = x + 1 )
            {
                // comparar cada posicao dos arranjos
                resposta = resposta && ( tabela1 [ x ] == (-1) * tabela2 [ x ] );
            } // fim for
        } // fim se
    } // fim se
    // retornar resultado
    return ( resposta );
} // fim eSimetrico ( )
```

```

/**
 * recuperar e comparar dados de arquivo.
 */
public static void teste09 ( )
{
    // 1. definir dados
    int [ ] tabela1 = null;
    int [ ] tabela2 = null;
    String nome1, nome2;
    boolean resposta;
    // 2. identificar
    IO.println ( );
    IO.println ( "Recuperar e comparar dados de arquivos" );
    IO.println ( );
    // 3. montar dados
    nome1 = IO.readString ( "Qual o nome do primeiro arquivo? " );
    nome2 = IO.readString ( "Qual o nome do segundo arquivo? " );
    tabela1 = lerDoArquivo ( nome1 );
    tabela2 = lerDoArquivo ( nome2 );
    // 4. testar a existencia de dados
    if ( tabela1 == null || tabela2 == null )
    {
        IO.println ( "ERRO: Arranjo nulo." );
    }
    else
    {
        if ( tabela1.length == 0 ||
            tabela2.length == 0 )
        {
            IO.println ( "ERRO: Tamanho invalido." );
        }
        else
        {
            IO.println ( "Testar se arranjos diferentes: " );
            mostrar ( tabela1 );
            IO.println ( );
            mostrar ( tabela2 );
            IO.println ( );
            IO.println ( "Resultado:" );
            resposta = eSimetrico ( tabela1, tabela2 );
            if ( resposta )
            {
                IO.println ( "arranjos simetricos." );
            }
            else
            {
                IO.println ( "arranjos nao sao simetricos." );
            }
        } // fim se
    } // fim se
} // fim se
// 5. encerrar
IO.println ( );
IO.pause ( "Apertar ENTER para continuar." );
} // fim teste09 ( )

```

OBS.:

Só poderão ser comparados arranjos com mesma quantidade de dados.

- 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 – Exemplo0150.java.
- 37.) Editar mudanças no nome do programa e versão.
Acrescentar uma função para calcular o produto escalar de dois arranjos.
Na parte principal, incluir chamadas para esse método.

```
// ----- definicao de metodo auxiliar
```

```
/**
 * produto escalar de dois arranjos.
 * @return novo arranjo com o produto
 * @param tabela1 - arranjo com dados
 * @param tabela2 - arranjo com dados
 */
public static int produtoEscalar
    ( int [ ] tabela1,
      int [ ] tabela2 )
{
    // definir dados
    int soma = 0;
    int tamanho;
    int x;

    // testar existencia de dados
    if ( tabela1 == null || tabela2 == null )
    {
        IO.println ( "ERRO: arranjo vazia." );
    }
    else
    {
        // testar se tamanhos validos
        if ( tabela1.length == 0 ||
            tabela2.length == 0 )
        {
            IO.println ( "ERRO: Tamanho(s) invalido(s)." );
        }
        else
        {
            // repetir para cada dado no arquivo
            tamanho = tabela1.length;
            for ( x = 0; x < tamanho; x = x + 1 )
            {
                // acumular o produto cada posicao da arranjo
                soma = soma + tabela1 [ x ] * tabela2 [ x ];
            } // fim for
        } // fim se
    } // fim se
    // retornar resultado
    return ( soma );
} // fim produtoEscalar ( )
```

```

/**
 * recuperar, multiplicar e mostrar dados de arquivos.
 */
public static void teste10 ( )
{
    // 1. definir dados
    int [ ] tabela1 = null;
    int [ ] tabela2 = null;
    int produto;

    // 2. identificar
    IO.println ( );
    IO.println ( "Recuperar, multiplicar e mostrar dados de arquivos" );
    IO.println ( );

    // 3. montar dados
    tabela1 = lerDoArquivo ( "DADOS1.TXT" );
    tabela2 = lerDoArquivo ( "DADOS2.TXT" );

    // 4. testar a existencia de dados
    if ( tabela1 == null || tabela2 == null )
    {
        IO.println ( "ERRO: arranjo nula." );
    }
    else
    {
        if ( tabela1.length == 0 ||
            tabela2.length == 0 )
        {
            IO.println ( "ERRO: Tamanho invalido." );
        }
        else
        {
            IO.println ( "Produto escalar de arranjos: " );
            mostrar ( tabela1 );
            IO.println ( "\n+" );
            mostrar ( tabela2 );
            IO.println ( );
            IO.println ( "Resultado:" );
            produto = produtoEscalar ( tabela1, tabela2 );
            IO.println ( "produto escalar = " + produto );
        } // fim se
    } // fim se

    // 5. encerrar
    IO.println ( );
    IO.pause ( "Apertar ENTER para continuar." );
} // fim teste10 ( )

```

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 (Exemplo0151) com função para ler o tamanho de um arranjo para inteiros do teclado, bem como todos os seus elementos.
Verificar se as dimensões não são nulas ou negativas.
Para testar, ler dados e mostrá-los na tela por outro método.
- 02.) Fazer um programa (Exemplo0152) com método para gravar um arranjo com inteiros em arquivo.
O arranjo e o nome do arquivo serão dados como parâmetros.
Para testar, usar o arranjo lido no exercício anterior.
Guardar primeiro o tamanho, depois os elementos, um dado por linha.
Usar funções para ler e recuperar o arranjo do arquivo, antes de mostrá-lo na tela.
- 03.) Fazer um programa (Exemplo0153) com função para gerar um valor inteiro aleatoriamente dentro de um intervalo, cujos limites de início e de fim serão passados como parâmetros;
Para para testar, ler os limites do intervalo do teclado;
ler a quantidade de elementos (N) a serem gerados;
gerar essa quantidade (N) de valores aleatórios dentro do intervalo e armazená-los em arranjo;
gravá-los, um por linha, em um arquivo ("DADOS.TXT").
A primeira linha do arquivo deverá informar a quantidade de números aleatórios (N) gravados em seguida.
DICA: Usar a conversão (int) Math.random().

Exemplo: valor = Arranjo.gerarInt (inferior, superior);

- 04.) Fazer um programa (Exemplo0154) com função para receber um nome de arquivo como parâmetro e retornar um arranjo com os valores lidos;
Para testar, definir outra função inteira para procurar o maior valor em um arranjo.
DICA: Usar o primeiro valor da tabela como referência inicial para o menor valor (inicial).

Exemplo: arranjo = Arranjo.lerArquivo ("DADOS.TXT");
maior = acharMenor (arranjo);

05.) Fazer um programa (Exemplo0155) com função para achar o maior e o menor dos valores um arranjo; Para testar, ler o arquivo ("DADOS.TXT") e armazenar os dados em um arranjo; gravar em outro arquivo apenas o maior e o menor valor encontrados, se forem diferentes.
DICA: Usar o primeiro dado do arquivo, após a quantidade de dados, como referência inicial para o maior e o menor valor.

```
Exemplo: arranjo = Arranjo.lerArquivo ( "DADOS.TXT" );  
tabelinha = Arranjo.maiorEmenor ( arranjo );
```

06.) Fazer um programa (Exemplo0156) com função para determinar a soma dos valores em um arranjo; definir outra função para determinar a média dos valores em um arranjo, usando a função anterior. Para testar, ler o arquivo ("DADOS.TXT") armazenar os dados em um arranjo; separar em dois outros arquivos, os valores maiores ou iguais à média, e os menores que ela.

```
Exemplo: arranjo = Arranjo.lerArquivo ( "DADOS.TXT" );  
media = Arranjo.acharMedia ( arranjo );
```

07.) Fazer um programa (Exemplo0157) com função para receber como parâmetro um arranjo e dizer se está ordenado, ou não, em ordem crescente.
DICA: Testar se não está desordenado, ou seja, se existe algum valor que seja maior que o seguinte.
Não usar break !

08.) Fazer um programa (Exemplo0158) com função para: procurar por determinado valor em arranjo e dizer se esse valor pode ser nele encontrado, dada a posição inicial para se iniciar a procura. Para testar, ler o arquivo ("DADOS.TXT"), e armazenar os dados em arranjo; ler do teclado um valor inteiro para ser procurado; determinar se o valor procurado existe no arranjo.

```
Exemplo: arranjo = Arranjo.lerArquivo ( "DADOS.TXT" );  
existe = Arranjo.acharValor ( arranjo, procurado, 0 );
```


09.) Fazer um programa (Exemplo0159) com função procurar por determinado valor em arranjo e dizer onde se encontra esse valor, dada a posição inicial para iniciar a procura. Para testar, ler o arquivo ("DADOS.TXT"), e armazenar os dados em arranjo; ler do teclado um valor inteiro para ser procurado; determinar onde o valor procurado está no arranjo.

Exemplo: arranjo = Arranjo.lerArquivo ("DADOS.TXT");
onde = Arranjo.acharPosicao (arranjo, procurado, 0);

10.) Fazer um programa (Exemplo0160) com função procurar por determinado valor em arranjo e dizer quantas vezes esse valor pode ser encontrado, dada a posição inicial para iniciar a procura. Para testar, ler o arquivo ("DADOS.TXT"), e armazenar os dados em arranjo; ler do teclado um valor inteiro para ser procurado; determinar onde o valor procurado está no arranjo.

Exemplo: arranjo = Arranjo.lerArquivo ("DADOS.TXT");
vezes = Arranjo.acharQuantos (arranjo, procurado, 0);

Tarefas extras

E1.) Fazer um programa para ler um valor inteiro do teclado, e mediante uma função calcular e retornar seus divisores em arranjo, o qual deverá ser mostrado na tela após o retorno e gravado em arquivo ("DIVISORES.TXT").

E2.) Fazer um programa para ler um arquivo ("PALAVRAS.TXT"), e mediante uma função retornar as dez primeiras palavras que comecem pela letra 'a' (ou 'A'), se houver. As palavras encontradas deverão ser exibidas na tela, após retorno.