

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

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

```
/**
 * Exemplo0221
 *
 * @author
 * @version 01
 */

// ----- dependencias

import IO.*;

// ----- definicao de classe auxiliar

/**
 * Classe para tratar contatos.
 */
class Contato
{
    /**
     * atributos.
     */
    public String nome;
    public String fone;

    /**
     * construtor padrao.
     */
    public Contato ( )
    {
        // atribuir valores iniciais nulos
        nome = null;
        fone = null;
    } // fim construtor padrao
} // fim da classe Contato
```

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

```
public class Exemplo0221
```

```
{
```

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

```
/**
```

```
 * Testar definições da classe Contato.
```

```
 */
```

```
public void metodo01 ( )
```

```
{
```

```
 // 1. definir dados
```

```
     Contato a1 = null;
```

```
     Contato a2 = new Contato ( );
```

```
 // 2. identificar
```

```
     IO.println ( "Definicoes da Contato" );
```

```
 // 3. testar as definicoes da Contato
```

```
     if ( a1 == null )
```

```
     {
```

```
         IO.println ( "Contato a1 nulo" );
```

```
     }
```

```
     else
```

```
     {
```

```
         IO.println ( "Contato a1 nao nulo" );
```

```
     } // fim se
```

```
     if ( a2 == null )
```

```
     {
```

```
         IO.println ( "Contato a2 nulo" );
```

```
     }
```

```
     else
```

```
     {
```

```
         IO.println ( "Contato a2 nao nulo" );
```

```
     } // fim se
```

```
 // 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 ( "EXEMPLO0201 - Programa em Java" );
```

```
     IO.println ( "Autor: _____" );
```

```
 // criar e executar o metodo auxiliar
```

```
     Exemplo0221 m1 = new Exemplo0221 ( );
```

```
     m1.metodo01 ( );
```

```
 // encerrar
```

```
     IO.pause ( "Apertar ENTER para terminar." );
```

```
 } // fim main( )
```

```
 } // fim class Exemplo0221
```

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 – Exemplo0222.java.

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

Acrescentar novo construtor à classe para criar objeto com valores iniciais.

```
/**
 * construtor alternativo.
 */
public Contato (String nomeInicial, String foneInicial )
{
    // testar e inserir valores iniciais

    // <incluir código para testar e atribuir valores>

} // fim construtor alternativo
```

Na parte principal, criar um segundo método para testes.

```
/**
 * Testar definições da classe Contato.
 */
public void metodo02 ( )
{
    // 1. definir dados
    Contato a1 = null;
    Contato a2 = new Contato ( );
    Contato a3 = new Contato ( "nome1", "1111-1111" );
    // 2. identificar
    IO.println ( "Definicoes da Contato" );
    // 3. testar as definicoes de arranjo
    // ...
    if ( a3 == null )
    {
        IO.println ( "Contato a3 nulo" );
    }
    else
    {
        IO.println ( "Contato a3 nao nulo com "+a3.nome+" e " + a3.fone );
    } // fim se
    // encerrar
    IO.println ( );
    IO.pause ( "Apertar ENTER para continuar." );
} // fim metodo02 ( )
```

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 – Exemplo0223.java.

- 09.) Editar mudanças no nome do programa e versão.  
Acrescentar tratamento de erro à classe Contato.

```
/**
 * Classe para tratar Contato de contatos.
 */
class Contato
{
    /**
     * tratamento de erro.
     * Codigos de erro:
     * 1. Nome invalido.
     * 2. Fone invalido.
     */
    private int erro;

    /**
     * obter o codigo de erro.
     */
    public int getErro ( )
    {
        return ( erro );
    } // end getErro ( )

    /**
     * estabelecer novo codigo de erro.
     */
    private void setErro ( int codigo )
    {
        erro = codigo;
    } // end setErro ( )

    /**
     * atributos.
     */
    public String nome;
    public String fone;

    // ...

}
```

Acrescentar ao construtor alternativo a atribuição de códigos de erros:  
(1) para nome inválido e (2) para fone inválido.

```
/**
 * construtor alternativo.
 */
public Contato (String nomeInicial, String foneInicial )
{
    // testar e inserir valores iniciais
    setErro ( 0 ); // ainda não há erro
    // <incluir outros testes, atribuições e códigos para erros>

} // fim construtor alternativo
```

Na parte principal, acrescentar um terceiro método para testar o tratamento de erro.

```
/**
 * Testar definições da classe Contato.
 */
public void metodo03 ( )
{
    // 1. definir dados
    Contato a1 = new Contato ( "", "1111-1111" );
    Contato a2 = new Contato ( "nome1", null );
    Contato a3 = new Contato ( "nome1", "1111-1111" );
    // 2. identificar
    IO.println ( "Definicoes da Contato" );
    // 3. testar as definicoes de arranjo
    // <conforme o modelo abaixo testar cada contato>
    if ( a3 == null )
    {
        IO.println ( "Contato a3 nulo" );
    }
    else
    {
        if ( a3.getErro ( ) != 0 )
        {
            IO.println ( "Contato a3 com erro " + a3.getErro ( ) );
        }
        else
        {
            IO.println ( "Contato a3 nao nulo com "+a3.nome+" e " + a3.fone );
        } // fim se
    } // fim se
    // encerrar
    IO.println ( );
    IO.pause ( "Apertar ENTER para continuar." );
} // fim metodo03 ( )
```

OBS.:

É conveniente modificar os construtores para usarem a nova forma para atribuir valor inicial ao erro. Observar que obter o código de erro é possível, mas alterá-lo fora da classe, não.

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 – Exemplo0224.java.

- 13.) Editar mudanças no nome do programa e versão.  
Acrescentar um método para indicar a existência de erro.

```
/**
 * indicar a existencia de erro.
 */
public boolean hasErro ( )
{
    return ( getErro() != 0 );
} // end hasErro ( )

/**
 * obter o codigo de erro.
 */
public int getErro ( )
{
    return ( erro );
} // end getErro ( )

/**
 * estabelecer novo codigo de erro.
 */
private void setErro ( int codigo )
{
    erro = codigo;
} // end setErro ( )
```

Na parte principal, acrescentar um terceiro método para testar o tratamento de erro.

```
/**
 * Testar definições da classe Contato.
 */
public void metodo04 ( )
{
    // 1. definir dados
    Contato a1 = new Contato ( "", "1111-1111" );
    Contato a2 = new Contato ( "nome1", null );
    Contato a3 = new Contato ( "nome1", "1111-1111" );
    // 2. identificar
    IO.println ( "Definicoes da Contato" );
    // 3. testar as definicoes de arranjo
    // <conforme o modelo abaixo testar cada contato>
    if ( a3 == null )
    {
        IO.println ( "Contato a3 nulo" );
    }
    else
    {
        if ( a3.hasErro ( ) )
        {
            IO.println ( "Contato a3 com erro " + a3.getErro ( ) );
        }
        else
        {
            IO.println ( "Contato a3 nao nulo com "+a3.nome+" e " + a3.fone );
        } // fim se
    } // fim se
    // encerrar
    IO.println ( );
    IO.pause ( "Apertar ENTER para continuar." );
} // fim metodo04 ( )
```

OBS.:

Que o novo método facilitará os testes para verificação de possíveis erros.

- 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 – Exemplo0225.java.



- 17.) Editar mudanças no nome do programa e versão.  
Acrescentar métodos públicos para obter o nome e o telefone.

```
/**
 * obter o nome.
 */
public String getNome ( )
{
    return ( nome );
} // end getNome ( )

/**
 * obter o telefone.
 */
public String getFone ( )
{
    return ( fone );
} // end getFone ( )
```

Na parte principal, acrescentar um método para testes.

```
/**
 * Testar definições da classe Contato.
 */
public void metodo05 ( )
{
    // 1. definir dados
    Contato a1 = new Contato ( "", "1111-1111" );
    Contato a2 = new Contato ( "nome1", null );
    Contato a3 = new Contato ( "nome1", "1111-1111" );
    // 2. identificar
    IO.println ( "Definicoes da Contato" );
    // 3. testar as definicoes de arranjo
    // <conforme o modelo abaixo testar cada contato>
    if ( a3 == null )
    {
        IO.println ( "Contato a3 nulo" );
    }
    else
    {
        if ( a3.hasErro ( ) )
        {
            IO.println ( "Contato a3 com erro " + a3.getErro ( ) );
        }
        else
        {
            IO.println ( "Contato a3 nao nulo com "
                + a3.getNome( )+" e " + a3.getFone( ) );
        }
    } // fim se
} // fim se
// encerrar
IO.println ( );
IO.pause ( "Apertar ENTER para continuar." );
} // fim metodo05 ( )
```

OBS.:

Notar que todos os usos públicos anteriores terão que ser revistos.

- 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 – Exemplo0226.java.
- 21.) Editar mudanças no nome do programa e versão.  
Acrescentar métodos públicos para obter e atribuir nome e telefone combinados.

```
/**
 * estabelecer novo nome.
 */
public void setNome ( String novoNome )
{
    nome = novoNome;
} // end setNome ( )

/**
 * estabelecer novo telefone.
 */
public void setFone ( String novoFone )
{
    fone = novoFone;
} // end setFone ( )
```

Na parte principal, acrescentar um método para testar a atribuição de valores ao objeto.

```
/**
 * Testar definições da classe Contato.
 */
public void metodo06 ( )
{
    // 1. definir dados
    Contato a1 = new Contato ( );
    Contato a2 = new Contato ( );
    Contato a3 = new Contato ( );
    // 2. identificar
    IO.println ( "Definicoes da Contato" );
    // 3. testar as definicoes de arranjo
    // <conforme o modelo abaixo testar cada contato>
    if ( a3 == null )
    {
        IO.println ( "Contato a3 nulo" );
    }
    else
    {
        a3.setNome ( "nome1" );
        a3.setFone ( "1111-1111" );
        if ( a3.hasErro ( ) )
        {
            IO.println ( "Contato a3 com erro " + a3.getErro ( ) );
        }
        else
        {
            IO.println ( "Contato a3 nao nulo com "
                + a3.getNome( )+" e " + a3.getFone( ) );
        }
    }
    // fim se
} // fim se
// 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 – Exemplo0227.java.

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

Modificar o construtor alternativo para usar também os métodos de atribuição, caso não houver erros.

```
/**
 * construtor alternativo.
 */
public Contato ( String nomeInicial,
                String foneInicial )
{
    // testar e atribuir nome inicial
    setErro ( 0 );
    if ( true )      // <modificar o teste>
    {
        setErro ( 1 ); // ERRO: Nome invalido.
    }
    else
    {
        setNome ( nomeInicial );
    } // fim se
    // testar e atribuir telefone inicial
    // <incluir teste semelhante para telefone>
} // fim construtor alternativo
```

Na parte principal, acrescentar um método para testar a atribuição de valores ao objeto.

```
/**
 * Testar definições da classe Contato.
 */
public void metodo07 ( )
{
    // 1. definir dados
    Contato a1 = new Contato ( "", "1111-1111" );
    Contato a2 = new Contato ( "nome1", null );
    Contato a3 = new Contato ( );
    // 2. identificar
    IO.println ( "Definicoes da Contato" );
    // 3. testar as definicoes de arranjo
    // <conforme o modelo abaixo testar cada contato>
    if ( a3 == null )
    {
        IO.println ( "Contato a3 nulo" );
    }
    else
    {
        a3.setNome ( "nome1" );
        a3.setFone ( "1111-1111" );
        if ( a3.hasErro ( ) )
        {
            IO.println ( "Contato a3 com erro " + a3.getErro ( ) );
        }
        else
        {
            IO.println ( "Contato a3 nao nulo com "
                + a3.getNome( )+" e " + a3.getFone( ) );
        }
    } // fim se
} // fim se
// encerrar
IO.println ( );
IO.pause ( "Apertar ENTER para continuar." );
} // fim metodo07 ( )
```

OBS.:

Notar que **todas** as atribuições a partir de agora deverão usar o método para acesso, a fim de garantir a consistência dos dados.

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 – Exemplo0228.java.

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

Acrescentar um método à classe para facilitar a exibição dos dados contidos em Contato.

```
/**
 * obter os conteudos do objeto.
 *
 * @return dados contidos no objeto.
 */
public String toString ( )
{
    return ( ""+nome+ " - "+fone );
} // end toString ( )
```

Na parte principal, acrescentar um método para testar a exibição de dados contidos no objeto.

```
/**
 * Testar definições da classe Contato.
 */
public void metodo08 ( )
{
    // 1. definir dados
    Contato a1 = new Contato ( "", "1111-1111" );
    Contato a2 = new Contato ( "nome1", null );
    Contato a3 = new Contato ( );
    // 2. identificar
    IO.println ( "Definicoes da Contato" );
    // 3. testar as definicoes de arranjo
    // <conforme o modelo abaixo testar cada contato>
    if ( a3 == null )
    {
        IO.println ( "Contato a3 nulo" );
    }
    else
    {
        a3.setNome ( "nome1" );
        a3.setFone ( "1111-1111" );
        if ( a3.hasErro ( ) )
        {
            IO.println ( "Contato a3 com erro " + a3.getErro ( ) );
        }
        else
        {
            IO.println ( "Contato a3 nao nulo com " + a3 );
            IO.println ( "Contato a3 nao nulo com " + a3.toString ( ) );
        }
    } // fim se
} // fim se
// encerrar
IO.println ( );
IO.pause ( "Apertar ENTER para continuar." );
} // fim metodo08 ( )
```

OBS.:

Notar que as chamadas são equivalentes, mas a primeira é mais simples.

- 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 – Exemplo0229.java.
- 33.) Editar mudanças no nome do programa e versão.  
Acrescentar um método à classe para clonar os dados em um objeto.

```
/**
 * clonar os conteudos do objeto.
 *
 * @return copia dos dados contidos no objeto.
 */
public Contato clone ( )
{
    Contato copia = new Contato ( getNome( ), getFone( ) );
    return ( copia );
} // end clone ( )
```

Na parte principal, acrescentar um método para testar a cópia de dados contidos no objeto.

```
/**
 * Testar definições da classe Contato.
 */
public void metodo09 ( )
{
    // 1. definir dados
    Contato a1 = new Contato ( "", "1111-1111" );
    Contato a2 = new Contato ( "nome1", null );
    Contato a3 = new Contato ( );
    // 2. identificar
    IO.println ( "Definicoes da Contato" );
    // 3. testar as definicoes de arranjo
    if ( a3 == null )
    {
        IO.println ( "Contato a3 nulo" );
    }
    else
    {
        a3.setNome ( "nome3" );
        a3.setFone ( "3333-3333" );
        a1 = a3.clone ( );
        if ( a1.hasErro ( ) )
        {
            IO.println ( "Contato a1 com erro " + a1.getErro ( ) );
        }
        else
        {
            IO.println ( "Contato a1 nao nulo com " + a1 );
        } // fim se
    } // fim se
    // encerrar
    IO.println ( );
    IO.pause ( "Apertar ENTER para continuar." );
} // fim metodo09( )
```

OBS.:

Notar que a cópia de dados preservará a individualidade de cada objeto.

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 – Exemplo0230.java.



- 37.) Editar mudanças no nome do programa e versão.  
Modificar o método para clonar dados, para lidar com erros.  
Acrescentar mais um código aos tipos de erros possíveis.

```
/**
 * clonar os conteudos do objeto.
 *
 * @return copia dos dados contidos no objeto.
 */
public Contato clone ( )
{
    Contato copia = new Contato ( );
    if ( copia == null || hasErro( ) )
    {
        setErro ( 3 ); // ERRO: Problema ao copiar.
    }
    else
    {
        if ( copia != null )
        {
            copia.setNome ( getNome( ) );
            copia.setFone ( getFone( ) );
        } // fim se
    } // fim se
    return ( copia );
} // end clone ( )
```

Na parte principal, acrescentar um método para testar o novo método para cópia.

```
/**
 * Testar definições da classe Contato.
 */
public void metodo10 ()
{
    // 1. definir dados
    Contato a1 = new Contato ( "", "1111-1111" );
    Contato a2 = new Contato ( "nome1", null );
    Contato a3 = new Contato ( );
    // 2. identificar
    IO.println ( "Definicoes da Contato" );
    // 3. testar as definicoes de arranjo
    if ( a3 == null )
    {
        IO.println ( "Contato a3 nulo" );
    }
    else
    {
        a3 = a1.clone ( );
        if ( a3.hasErro ( ) )
        {
            IO.println ( "Contato a3 com erro " + a3.getErro ( ) );
        }
        else
        {
            IO.println ( "Contato a3 nao nulo com " + a3 );
        }
    }
    // fim se
} // fim se
// encerrar
IO.println ( );
IO.pause ( "Apertar ENTER para continuar." );
} // fim metodo10()
```

OBS.:

Notar que a cópia de dados poderá existir, válida, mas nula, ainda que o original contenha erro.

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 (Exemplo0231) para acrescentar um método público à classe Contato para ler e atribuir um valor ao nome (atributo do objeto).  
Incluir um método para testar essa nova característica.  
DICA: Testar se o nome não está vazio.

Exemplo: contato1.readNome ( "Nome: " );

- 02.) Fazer um programa (Exemplo0232) para acrescentar um método público à classe Contato para ler e atribuir um valor ao telefone (atributo do objeto).  
Incluir um método para testar essa nova característica.  
DICA: Testar se o telefone não está vazio.

Exemplo: contato1.readFone ( "Fone: " );

- 03.) Fazer um programa (Exemplo0233) para acrescentar um método privado à classe Contato para testar se o valor de um telefone é válido, ou não.  
Incluir um método para testar essa nova característica.  
DICA: Testar se as posições contêm apenas algarismos e o símbolo '-'.

- 04.) Fazer um programa (Exemplo0234) para acrescentar um método público à classe Contato para ler dados de arquivo, dado o nome do mesmo, e armazenar em um objeto dessa classe.  
Incluir um método para testar essa nova característica.

Exemplo: contato1.fromFile ( "Pessoa1.txt" );

- 05.) Fazer um programa (Exemplo0235) para acrescentar um método à classe Contato para gravar dados de uma pessoa 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: contato1.toFile ( "Pessoa1.txt" );

- 06.) Fazer um programa (Exemplo0236) para acrescentar à classe Contato um novo atributo para um segundo telefone e modificar os construtores para lidar com isso. Incluir um método para testar essa nova característica.

Exemplo: contato1 = new Contato ( "nome1", "1111-1111", "2222-2222" );

- 07.) Fazer um programa (Exemplo237) para acrescentar um método público à classe Contato um novo atributo para indicar quantos telefones estão associados a cada objeto. Incluir um método para obter essa informação. Incluir um método para testar essa nova característica.

Exemplo: int n = contato1.telefones ( );

- 08.) Fazer um programa (Exemplo0239) para acrescentar um método público à classe Contato para atribuir o valor do segundo telefone. Incluir um método para testar essa nova característica. DICA: Se o contato só tiver um telefone, perguntar se quer acrescentar mais um número, e mudar automaticamente a quantidade deles, se assim for desejado.

Exemplo: contato.setFone2a ( "3333-3333" );

- 09.) Fazer um programa (Exemplo0238) para acrescentar um método público à classe Contato para alterar o valor apenas do segundo telefone. Incluir um método para testar essa nova característica. DICA: Se o contato não tiver dois telefones, uma situação de erro deverá ser indicada.

Exemplo: contato.setFone2b ( "3333-3333" );

- 10.) Fazer um programa (Exemplo0240) para acrescentar um método público à classe Contato para remover apenas o valor do segundo telefone. Incluir um método para testar essa nova característica. DICA: Se o contato só tiver um telefone, uma situação de erro deverá ser indicada.

Exemplo: contato.setFone2c ( null );

## Tarefas extras

- E1.) Fazer modificações na classe Contato para lidar com qualquer quantidade de telefones, menor que 10.  
Incluir testes para essa nova característica.  
DICA: Guardar a quantidade de telefones e, separadamente, os telefones em arranjo.
- E2.) Fazer modificações na classe Contato para lidar também com endereços (residencial e profissional).  
Incluir testes para essa nova característica.  
DICA: Guardar separadamente o endereço residencial e o profissional.