Java – Aula 06

Banco de Dados – Exemplo com JTable

Tratamento de Exceções

19/09/2012

Celso Olivete Júnior

olivete@fct.unesp.br

Banco de dados - exemplo com JTable

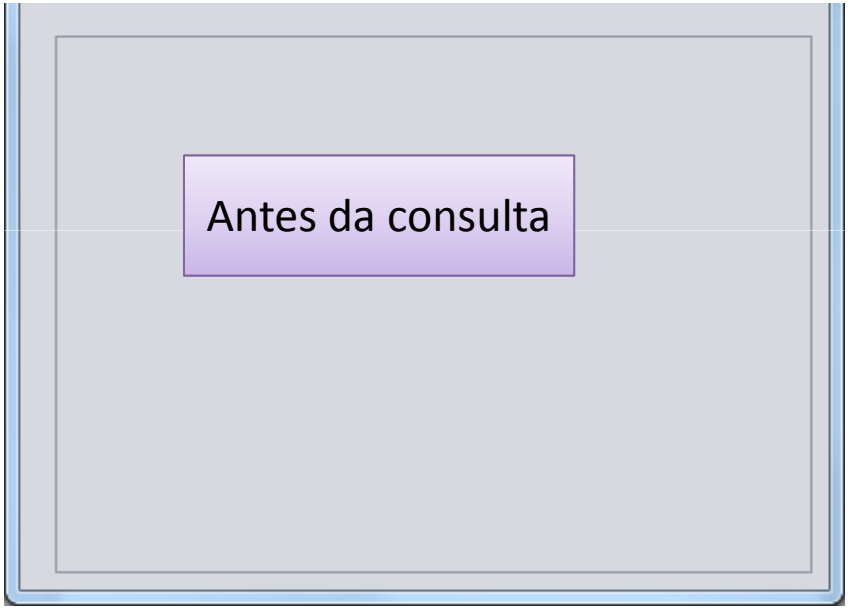
- permite exibir os dados do BD na forma de tabela

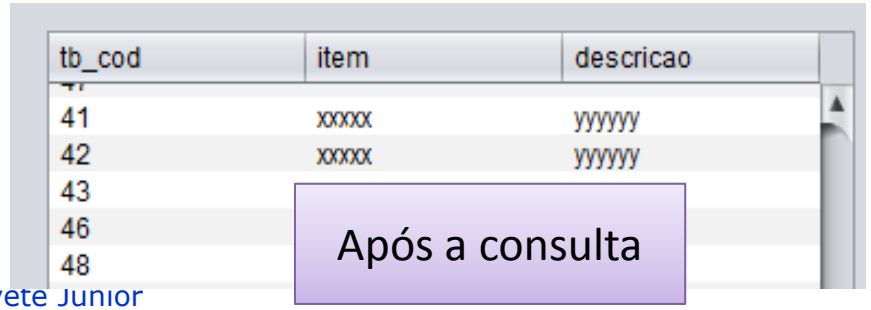
tb_cod	item	descricao
41	xxxxx	yyyyyy
42	xxxxx	yyyyyy
43	xxxxx	yyyy
46		
48		

O componente JTable: realizando uma consulta

- o componente pode ser definido através do pacote Swing

- serão definidas as colunas e as linhas em tempo de execução



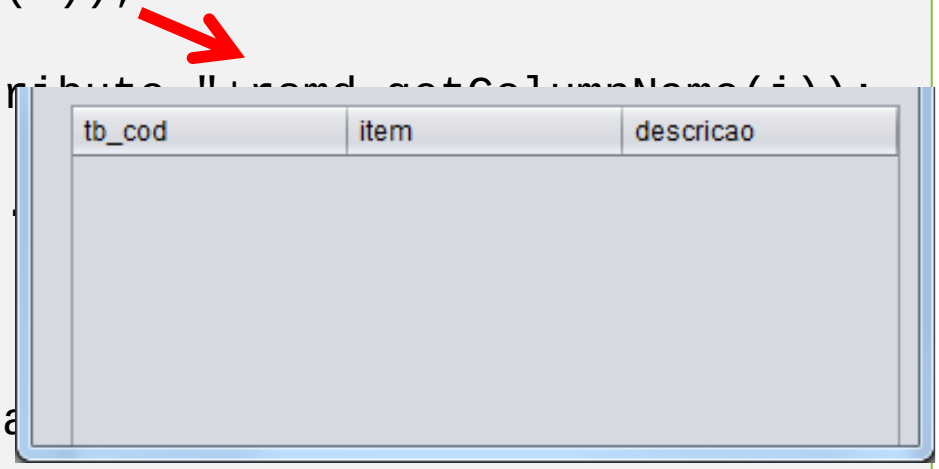


O componente JTable

```
...
1. //estabelece conexão com o MySql e seleciona BD
2.  conexao();
3. //objeto Statement para submeter o SQL para o banco de dados
4. Statement statement = connect.createStatement();
5. //cria a instrução SQL
6. String query = "SELECT*FROM tb_dados";
7. //submete a consulta a partir de statement e armazena o
   //resultado em um objeto do tipo resultset
8. ResultSet rs = statement.executeQuery( query );
9. //cria um objeto DefaultTableModel para gerenciar os dados que
   //serão inseridos na jTable
10.DefaultTableModel dtm = (DefaultTableModel) jTable1.getModel();
11.//objeto ResultSetMetaData - metadados que serão utilizados
   //para descrever o conteúdo do resultset
12.ResultSetMetaData rsmd = rs.getMetaData();
...//continua
```

```
13....
14.ResultSetMetaData rsmd = rs.getMetaData();
15.//método getColumnCount() do ResultSetMetaData retorna o total de
   //colunas da tabela
16.for(int i=1;i<=rsmd.getColumnCount();i++)
17.      {
18.//metodo getColumnName() recupera o nome de cada uma das colunas
   //metodo addColumn do DefaultTableModel adiciona ao Jtable cada
   //uma das colunas
19.dtm.addColumn(rsmd.getColumnName(i));
20.//exibe o nome do atributo
21.System.out.print(" \n Nome do atributo "+rsmd.getColumnName(i));
22.//exibe o tipo do atributo
23.System.out.print("\n Tipo do atr." + rsmd.getColumnTypeName(i));
24.//exibe a precisão do atributo
25.System.out.print(" \n precisão " +rsmd.getPrecision(i));
26.//exibe o nome da tabela
27.System.out.print(" \n Nome tabela "+rsmd.getTableName(i));
28.      }
```

```
13....
14.ResultSetMetaData rsmd = rs.getMetaData();
15.//método getColumnCount() do ResultSetMetaData retorna o total de
   //colunas da tabela
16.for(int i=1;i<=rsmd.getColumnCount();i++)
17.      {
18.//metodo getColumnName() recupera o nome de cada uma das colunas
   //metodo addColumn do DefaultTableModel adiciona ao Jtable cada
   //uma das colunas
19.dtm.addColumn(rsmd.getColumnName(i));
20.//exibe o nome do atributo
21.System.out.print(" \n Nome do atr
22.//exibe o tipo do atributo
23.System.out.print("\n Tipo do atr
24.//exibe a precisão do atributo
25.System.out.print(" \n precisão "
26.//exibe o nome da tabela
27.System.out.print(" \n Nome tabela
28.      }
```

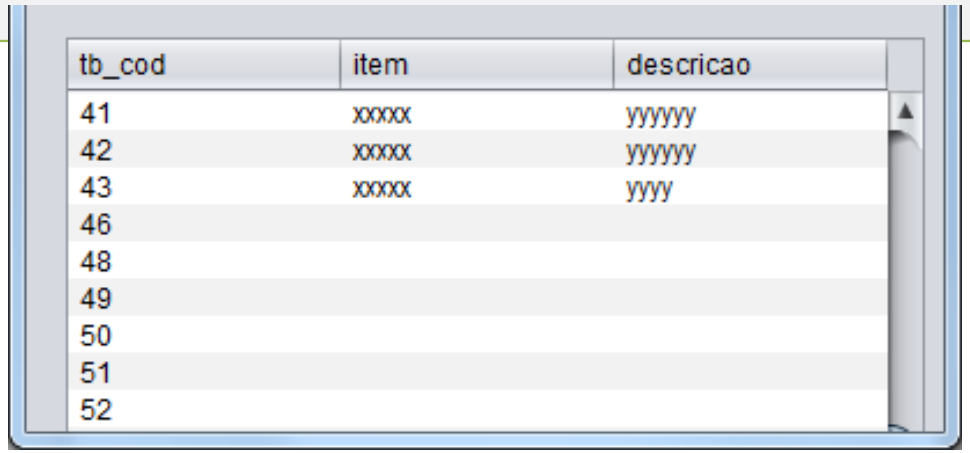


```
13....
14.ResultSetMetaData rsmd = rs.getMetaData();
15.//método getColumnCount() do ResultSetMetaData retorna o total de
   //colunas da tabela
16.for(int i=1;i<=rsmd.getColumnCount();i++)
17.      {
18.//metodo getColumnName() recupera o nome de cada uma das colunas
   //metodo addColumn do DefaultTableModel adiciona ao Jtable cada
   //uma das colunas
19.dtm.addColumn(rsmd.getColumnName(i));
20.//exibe o nome do atributo
21.System.out.print(" \n Nome do atributo "+rsmd.getColumnName(i));
22.//exibe o tipo do atributo
23.System.out.print("\n Tipo do atr." + rsmd.getColumnTypeName(i));
24.//exibe a precisão do atributo
25.System.out.print(" \n precisão " +rsmd.getPrecision(i));
26.//exibe o nome da tabela
27.System.out.print(" \n Nome tabela "+rsmd.getTableName(i));
28.      }
```

```
13....
14.ResultSetMetaData rsmd = rs.getMetaD
15.//método getColumnCount() do ResultS                    l de
   //colunas da tabela
16.for(int i=1;i<=rsmd.getColumnCount()
17.      {
18.//metodo getColumnName() recupera o                     nas
   //metodo addColumn do DefaultTableMo                     da
   //uma das colunas
19.dtm.addColumn(rsmd.getColumnName(i));
20.//exibe o nome do atributo
21.System.out.print(" \n Nome do atributo "+rsmd.getColumnName(i));
22.//exibe o tipo do atributo
23.System.out.print("\n Tipo do atr." + rsmd.getColumnTypeName(i));
24.//exibe a precisão do atributo
25.System.out.print(" \n precisão " +rsmd.getPrecision(i));
26.//exibe o nome da tabela
27.System.out.print(" \n Nome tabela "+rsmd.getTableName(i));
28.      }
```

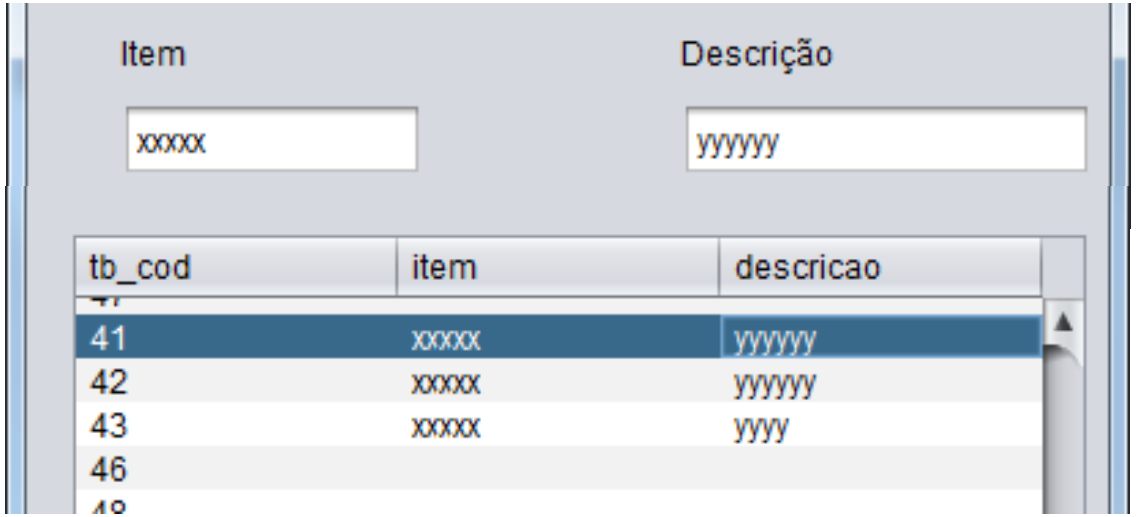
```
Nome do atributo tb_cod
Tipo do atributo INT
precisão 11
Nome tabela tb_dados

Nome do atributo item
Tipo do atributo VARCHAR
precisão 20
Nome tabela tb_dados

Nome do atributo descricao
Tipo do atributo VARCHAR
precisão 40
Nome tabela tb_dados
```

```
29...
30. //percorrendo o resultset e adicionando valor de cada atributo
   no JTable
31.          while(rs.next())
32.          {
33.           //recuperando o valor do atributo utilizando índice
34.           int codigo=rs.getInt(1);
35.          //recuperando o valor do atributo utilizando o nome
36.          String info=rs.getString("item");
37.          String descricao=rs.getString("descricao");
38.//adiciona os valores recuperados ao DefaultTableModel dtm
39.          dtm.addRow(new Object[]{codigo,info,descricao});
40.
41.          }
42....
```



O componente Jtable – recuperando os valores da linha selecionada – **Evento mouseClicked**

```
1. //recupera o valor de acordo com a linha selecionada e de acordo com
   //a coluna desejada joga o valor pro respectivo campo
2. jTFitem.setText(jTable1.getValueAt(jTable1.getSelectedRow(),
   dtm.findColumn("item")).toString());
3. jTFdescricao.setText(jTable1.getValueAt(jTable1.getSelectedRow(),
   dtm.findColumn("descricao")).toString());
```



O componente JTable

- **removendo um componente**
- a partir do objeto DefaultTableModel e com o método removeRow, passando como parâmetro a linha selecionada na jTable

```
dtm.removeRow(jTable1.getSelectedRow());
```

Exercício I: utilizando o componente JTable

- implemente a funcionalidade *localizar* um registro, utilize a jTable para mostrar o resultado

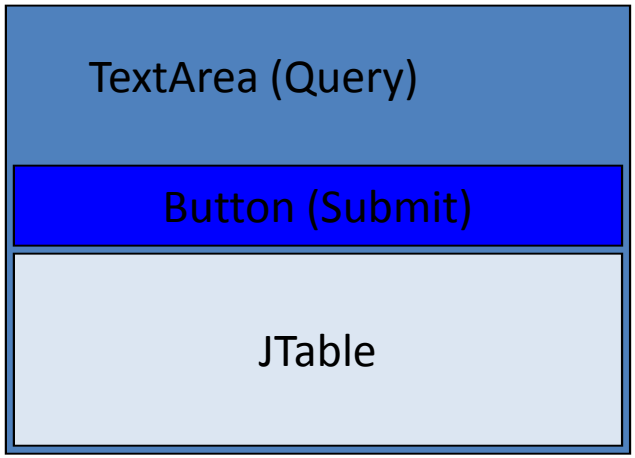
- implemente um evento responsável por *excluir* um registro do BD ao clicar sobre um item na jTable – utilize um jOptionPane para confirmar a exclusão

- implemente a funcionalidade *alterar* um registro – utilize o jTable para facilitar a alteração.

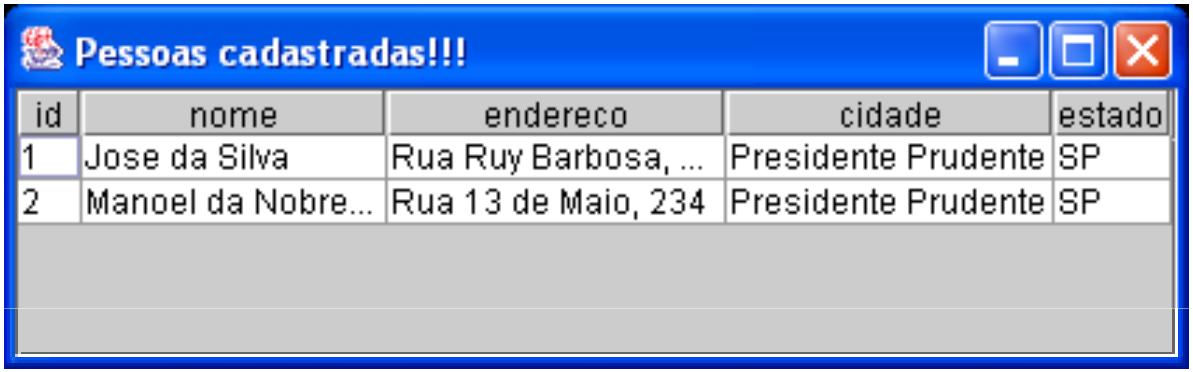
Exercício II: utilizando o componente JTable

- Implementar um ambiente para consulta SQL usando Jtable (Lembre-se: nem sempre a consulta será do tipo *** from**). O JTable será recriado de acordo com o resultado da consulta;

- Adicione tratamento de exceções



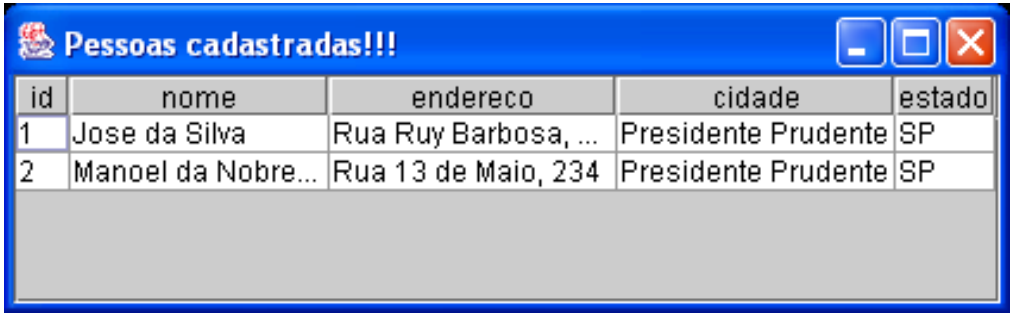
Exemplo: criando um Jtable de acordo com o resultado da consulta




```
private void getTable(){
        Statement st;
        ResultSet rs;
        try{
                String query = "SELECT * FROM pessoa";
                st = connect.createStatement();
                rs = st.executeQuery(query);
                mostraResultado(rs);
                st.close();
        }catch(SQLException sqlex){
                sqlex.printStackTrace();
    }
}
```

Exemplo: criando um Jtable de acordo com o resultado da consulta

```
private void mostraResultado(ResultSet rs) throws SQLException {
      Vector cabecalho = new Vector();
      Vector tuplas = new Vector();
      try {
            ResultSetMetaData rsmd = rs.getMetaData();
            for (int i = 1; i<=rsmd.getColumnCount(); i++)
                  cabecalho.addElement(rsmd.getColumnName(i));
      do{
            tuplas.addElement(getTupla(rs,rsmd));
      }while(rs.next());

      table = new JTable(tuplas, cabecalho);
      }catch(SQLException sqlex){
      .....   }
}
```



```
private Vector getTupla(ResultSet rs, ResultSetMetaData rsmd) throws SQLException{
    Vector tupla = new Vector();

    for (int i=1; i<=rsmd.getColumnCount(); i++)
    {
        switch(rsmd.getColumnType(i)){
                case Types.VARCHAR:
                  tupla.addElement(rs.getString(i));
                  break;
                case Types.INTEGER:
                  tupla.addElement(new Long(rs.getLong(i)));
                  break;
                case Types.SMALLINT:
                  tupla.addElement(new Long(rs.getLong(i)));
                  break;
                case Types.CHAR:
                  tupla.addElement(rs.getString(i));
                  break;
                default:
                   System.out.println("O Tipo era: " + rsmd.getColumnTypeName(i));
        }
    } return tupla;
}
```

Tratamento de exceções

Visão Geral do Tratamento de Exceções

- Exceção
  - Indicação de um problema durante a execução
  - Exemplo: Divisão por zero, parâmetro de método inválido, índice de vetor fora do limite, etc
- O Tratamento é fornecido para permitir aos programas capturar e tratar erros
- Permite capturar:
  - Todas as exceções
  - Todas as exceções de um certo tipo
  - Todas as exceções de tipos relacionados
- Isso torna os programas mais robustos, claros e tolerantes a falhas

Visão Geral do Tratamento de Exceções

- O uso do Tratamento de Exceções
  - Processa situações excepcionais em que um método é incapaz de completar sua tarefa por razões que ele não pode controlar;
  - Processa exceções de componentes do programa que não são projetadas para tratar essas exceções diretamente;
  - Remove o código de tratamento de erro da linha principal de execução;
- Um método detecta um erro e dispara uma exceção
  - O tratador de exceção processa o erro;
  - Exceções não capturadas podem levar a efeitos adversos
    - Por exemplo: podem terminar a execução do programa.

Visão Geral do Tratamento de Exceções

- O código que pode gerar algum erro é colocado dentro de um bloco **try**
  - O código para o tratamento de erros são colocados dentro de cláusulas **catch**
  - A cláusula **finally** é opcional e é executada sempre
  - A cláusula **finally** é ideal para o código que libera recursos
  - O bloco **try** deve ser seguido por uma cláusula **catch** ou **finally**
- Modelo de terminação do tratamento de exceções:
  - O bloco em que a exceção ocorreu expira (o controle do programa não retorna diretamente para o ponto de disparo)
- Cláusula **throws** especifica a exceção que o método dispara

Visão Geral do Tratamento de Exceções

- Bloco **try**

```
try {
instruções que podem disparar uma exceção
 } catch ( TipoDeExceção ref) {
instrução para processar uma exceção
 }
```

- O bloco **finally** é executado independente de uma exceção ser disparada ou não
- O tratador de exceção não pode acessar objetos definidos no bloco **try**, pois o bloco expira antes que o tratador comece a ser executado

Visão Geral do Tratamento de Exceções

- Disparando uma exceção
  - A instrução **throw** é executada para indicar que uma exceção ocorreu (ou seja, o código não pode ser completado com sucesso);
  - Essa instrução especifica um objeto a ser disparado;
  - O operando de um **throw** pode ser qualquer classe derivada da classe **Throwable** (pacote java.lang);
  - **Exception** é uma subclasse que refere-se a exceções causadas por problemas que devem ser capturados durante a execução do programa a fim de torná-lo mais robusto.
  - Independente do ponto em que ocorre a exceção (método profundamente aninhado), o controle seguirá para o tratador do bloco **catch**;
  - Ex: Os métodos chamados dentro do bloco **try** podem disparar exceções;
    - Uma instrução que tenta acessar um array com índice inválido dispara uma **ArrayIndexOfBoundsException**

Visão Geral do Tratamento de Exceções

- Capturando um exceção
  - Os tratadores de exceção ficam contidos em blocos **catch**
  - Dentre os parênteses deve ser especificado o nome da classe (especifica o tipo de exceção a ser capturada);

```
catch (Throwable th) {
   // captura todas as exceções e erros
}
catch (Exception ex) {
   // captura todas as exceções
}
```

- Geralmente não se especifica tratadores de exceção do tipo **Throwable**, porque **Error's** normalmente não devem ser capturados (**referem-se a problemas de sistema particularmente sérios!**)

Visão Geral do Tratamento de Exceções

- Capturando um exceção
  - Podem haver blocos **catch** que não coincida com o objeto disparado em particular;
  - Isso faz com que se procure uma próxima cláusula **catch**, até mesmo dentro do bloco **try** que o envolve;
  - Em algum momento o programa pode determinar que não há um tratador correspondente na pilha de execução e se o programa não for baseado em GUI, ele é finalizado.
  - Em programas com GUI, podem continuar executando incorretamente.
  - O primeiro tratador que corresponder ao tipo de exceção é executado (Todos os outros tratadores de exceção para o bloco **try** correspondente são ignorados!)

Visão Geral do Tratamento de Exceções

- Capturando

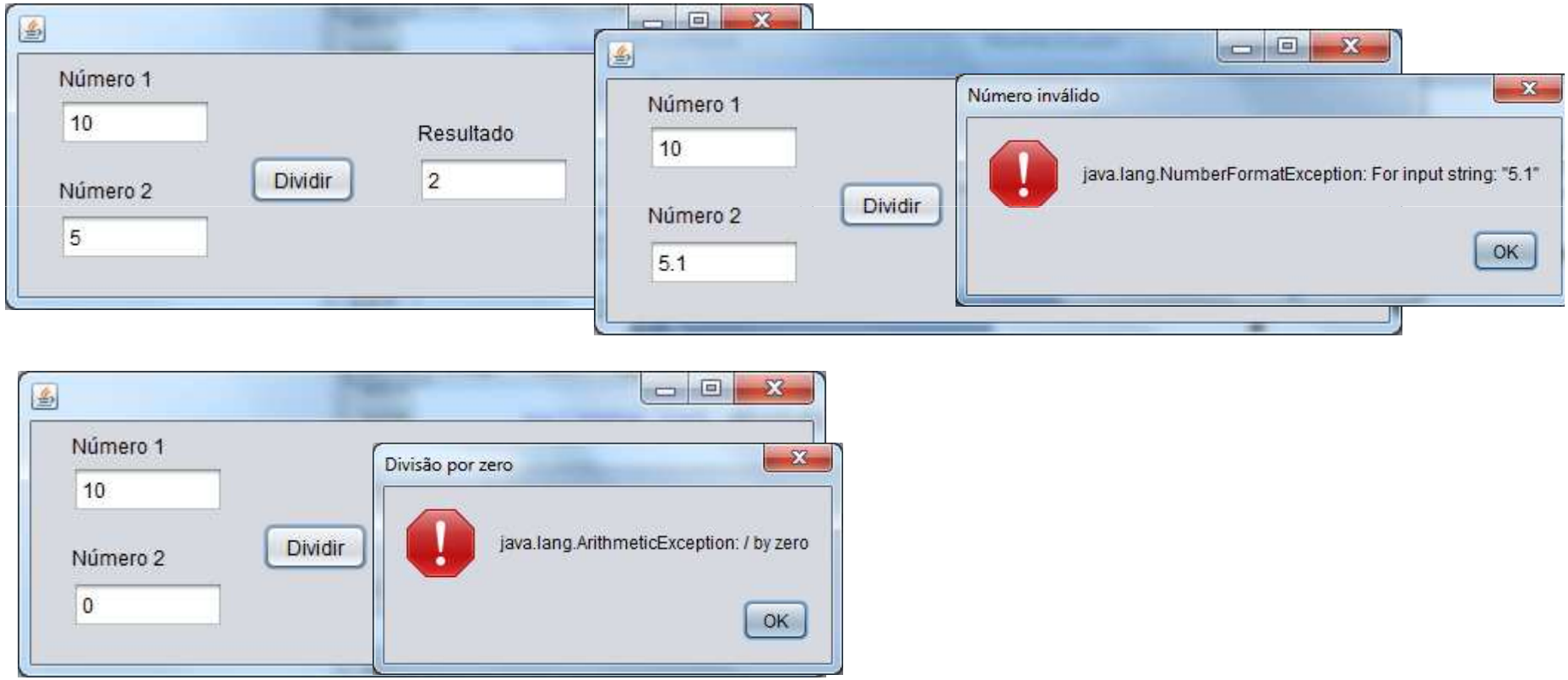
```
try
{
  // Trecho de código protegido
}
catch (TipoExcecao1 e1)
{
  // Tratamento da exceção tipo 1
}
catch (TipoExcecao2 e2)
{
  // Tratamento da exceção tipo 2
}

[finally
{
  // Código a ser executado antes
  // do bloco try terminar
}]
```

  - Podem ha… objeto disparado em particular;
  - Isso faz … a **catch**, até mesmo dentro do b…
  - Em algum … não há um tratador correspond… não for baseado em GUI, ele é …
  - Em progra… incorretamente.
  - O primeir… exceção é executado (Todos os … **try** correspondente são ignorac…

Exemplo de tratamento de Exceção: Divisão por Zero

- Esse é um erro comum de programação!
- Dispara **ArithmeticException**

Exemplo de tratamento de Exceção: Divisão por Zero



Exemplo de tratamento de Exceção: Divisão por Zero

```
1.  try {
2.              int n1 = Integer.parseInt(jTFN1.getText());
3.              int n2 = Integer.parseInt(jTFN2.getText());
4.              //método que retorna a divisão de n1 por n2
5.              int res = dividir(n1,n2);
6.              jTFNRes.setText(String.valueOf(res));
7.          } catch ( NumberFormatException  eNFE) {
8.
9.              JOptionPane.showMessageDialog(this,
                      eNFE.toString(), "Número inválido",
10.                   JOptionPane.ERROR_MESSAGE );
11.
12.         }
13.         catch (ArithmeticException eAE )
14.         {
15.             JOptionPane.showMessageDialog(this,
16.                   eAE.toString(), "Divisão por zero",
17.                   JOptionPane.ERROR_MESSAGE );
18.         }
```

Exemplo de tratamento de Exceção: Divisão por Zero

```
1.  try {
2.              int n1 = Integer.parseInt(jTFN1.getText());
3.              int n2 = Integer.parseInt(jTFN2.getText());
4.              //método que retorna a divisão de n1 por n2
5.              int res = dividir(n1,n2);
6.              jTFNRes.setText(String.valueOf(res));
7.          } catch ( NumberFormatException  eNFE) {
8.
9.              JOptionPane.showMessageDialog(this,
                     eNFE.toString(), "Número inválido",
10.                  JOptionPane.ERROR_MESSAGE );
11.
12.         }
13.         catch (ArithmeticException eAE )
14.         {
15.             JOptionPane.showMessageDialog(this,
16.                   eAE.toString(), "Divisão por zero",
17.                   JOptionPane.ERROR_MESSAGE );
18.         }
```

Trata o formato numérico

Exemplo de tratamento de Exceção: Divisão por Zero

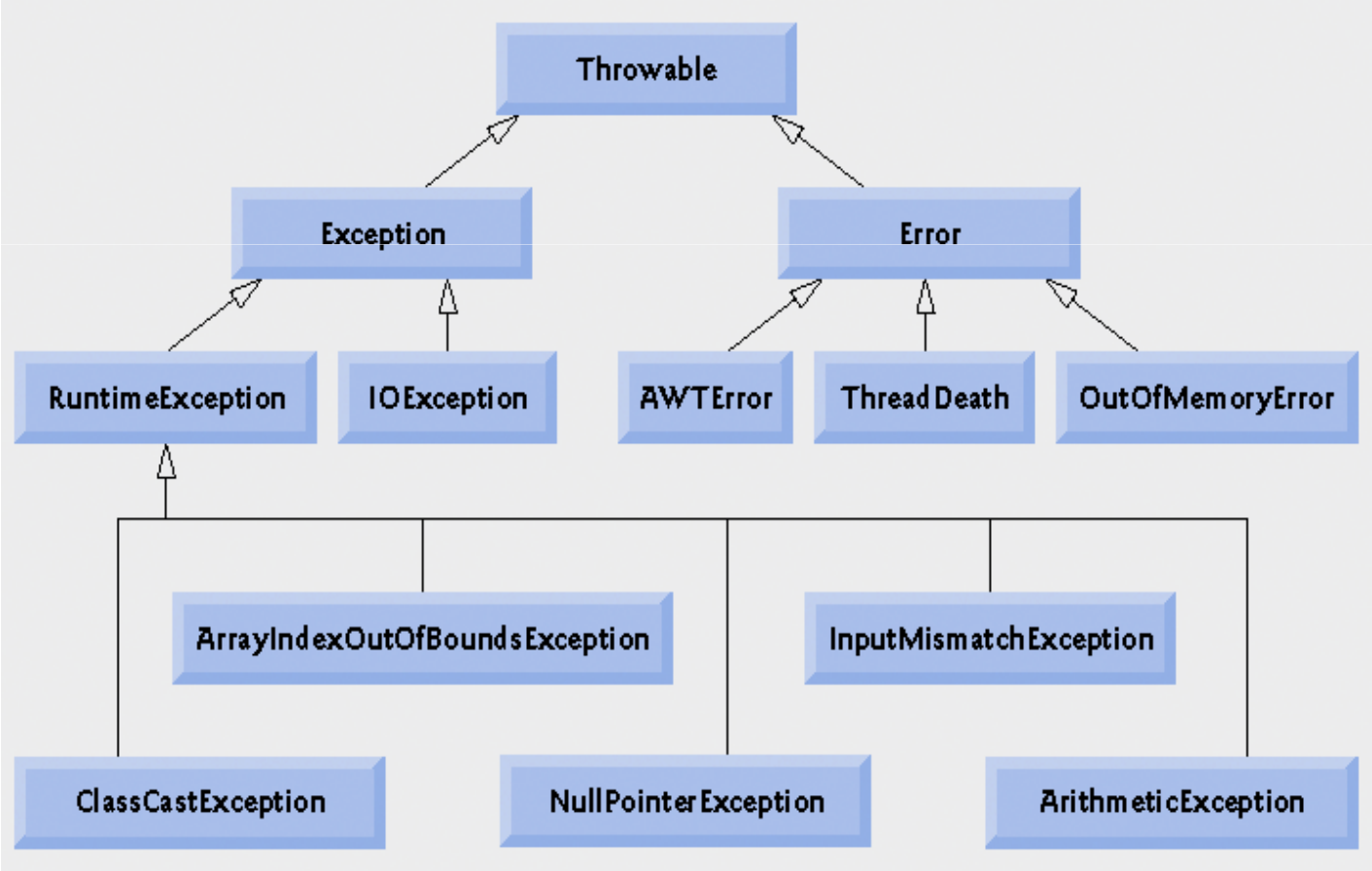
Caso um objeto Exception for definido no bloco **catch qualquer tipo de** exceção será capturada.

```
1.  try {
2.              int n1 = Integer.parseInt(jTFN1.getText());
3.              int n2 = Integer.parseInt(jTFN2.getText());
4.              //método que retorna a divisão de n1 por n2
5.              int res = dividir(n1,n2);
6.              jTFNRes.setText(String.valueOf(res));
7.          } catch ( NumberFormatException  eNFE) {
8.
9.              JOptionPane.showMessageDialog(this,
                     eNFE.toString(), "Número inválido",
10.                  JOptionPane.ERROR_MESSAGE );
11.
12.         }
13.         catch (ArithmeticException eAE )
14.         {
15.             JOptionPane.showMessageDialog(this,
16.                   eAE.toString(), "Divisão por zero",
17.                   JOptionPane.ERROR_MESSAGE );
18.         }
```

Trata o formato numérico

Trata erro aritmético

Hierarquia da classe Throwable: visão parcial

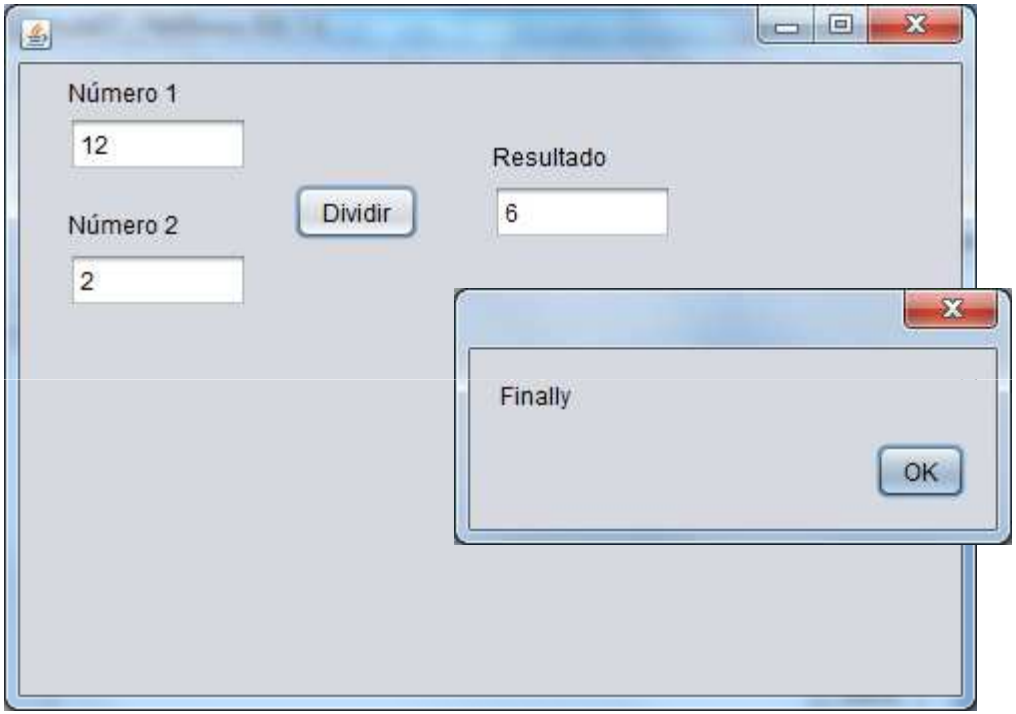


Cláusula finally

- Aparece depois da última cláusula catch;
- Ele é sempre executado;
  - Independente se uma exceção for ou não disparada;
  - Mesmo se o bloco **try** for encerrado com uma instrução **return**, **break** ou **continue**;
- Usado para a liberação de recursos.

```
1.  try {
2.              int n1 = Integer.parseInt(jTFN1.getText());
3.              int n2 = Integer.parseInt(jTFN2.getText());
4.              //método que retorna a divisão de n1 por n2
5.              int res = dividir(n1,n2);
6.              jTFNRes.setText(String.valueOf(res));
7.          } catch ( NumberFormatException  eNFE) {
8.
9.              JOptionPane.showMessageDialog(this, eNFE.toString(),
   "Número inválido", JOptionPane.ERROR_MESSAGE );
10.
11.         }
12.         catch (ArithmeticException eAE )
13.         {
14.             JOptionPane.showMessageDialog(this, eAE.toString(),
   "Divisão por zero", JOptionPane.ERROR_MESSAGE );
15.         }
16.
17.         finally
18.         {
19.             JOptionPane.showMessageDialog(this, "Finally", null,
   JOptionPane.PLAIN_MESSAGE );
20.         }
```

Cláusula finally



Mais um exemplo...

```
1. public class TrataExcecao{
2. public static void main (String [] args)
3. {
4. try
5. {
6.      int vetor[] = new int[100];
7.      vetor[100] = 1;
8.      System.out.println("Dentro do bloco try...");
9. }
10.catch(ArrayIndexOutOfBoundsException e)
11.{
12.     System.out.println("Ocorreu a exceção: " +
        e.getMessage());
13.     System.out.println("Ocorreu a exceção: " + e);
14.}
15.     System.out.println("Após o tratamento de exceções...");
16.}
17.}
```

Pelo **modelo de terminação, esta** instrução não será executada.

Altere o tipo de exceção para **Exception. O que ocorre? Por quê?**
Altere o tipo de exceção para **ArithmeticException. O que ocorre? Por quê?**

Mais um exemplo...

```
1. public class TrataExcecao2 {
2. public static void main (String [] args) {
3. try {
4.     int vetor[] = new int[100];
5.     float valor1 = 1, valor2 = 0;
6.     System.out.print(valor1 + "/" + valor2 + " = ");
7.     System.out.println(valor1/valor2);
8.     System.out.println("vetor[100] = " + vetor[100]);
9. }
10.catch(ArrayIndexOutOfBoundsException e) {
11.    System.out.println("Índice inválido. Exceção: " + e);
12.}
13.catch(ArithmeticException e) {
14.    System.out.println("Erro aritmético. Exceção: " + e);
15.}
16.catch(Exception e) {
17.    System.out.println("Ocorreu a exceção: " + e);
18.}
19.    System.out.println("Após o tratamento de exceções...");
20.}
21.}
```

O que vai ocorrer nesse caso?

Declarando novos tipos de exceção

- Além das exceções já previstas na linguagem Java, pode-se gerar novas exceções sempre que necessário. Por exemplo, para considerar uma situação para a qual não está prevista uma exceção pela linguagem Java.
- Para gerar uma exceção, deve-se usar a instrução throw.
- Exceções geradas pela instrução throw também podem ser capturadas e tratadas em um bloco catch.

Declarando novos tipos de exceção

- Utilizando o conceito de hierarquia de classes, pode-se criar uma classe tratadora de exceções derivada de uma classe de exceção existente

Declarando novos tipos de exceção - exemplo

```
public class UsaClasseTratadoraDeExcecao {
  public static void main(String[] args) {
       try
       {
       facaAlgo(7);
       facaAlgo(0);
       }
       catch(NovaExcecao e)
       {
       System.out.println("Exceção: " + e.toString());
       }
}
static void facaAlgo(int valor) throws NovaExcecao {
       if (valor == 0)
              throw new NovaExcecao(valor);
}
}
```

Chamada do método que irá disparar uma nova exceção e o tratamento deste tipo de exceção.

Método que gera uma exceção do novo tipo criado quando recebe um valor = 0.

Declarando novos tipos de exceção - exemplo

- A classe tratadora de exceção deve ser uma subclasse da classe Exception.

```
public class NovaExcecao extends Exception
{
	int valor;
	public NovaExcecao(int v)
	{
		valor = v;
	}
	public String toString()
	{
	return "Nova exceção " + valor;
	}
}
```

Exercícios: implemente os exemplos...