Exceptions
Exceções em Java

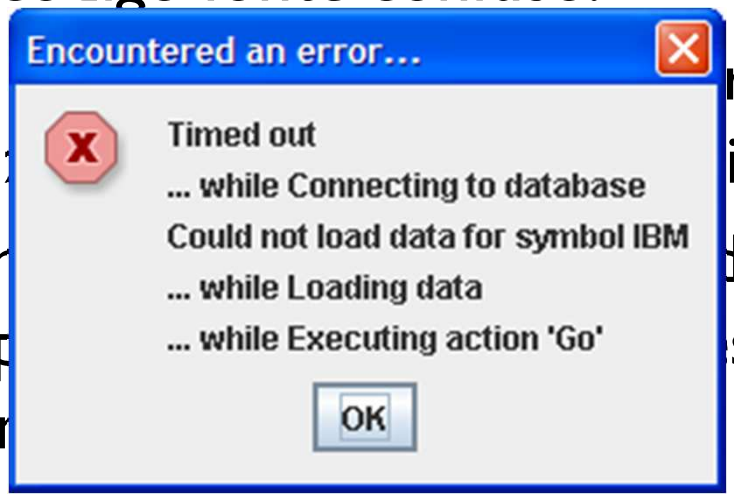
LPOO – Ling de Programação Orientada a Objetos

Prof. Leandro Carlos Fernandes

(Material adaptado a partir dos slides de Marco Reis)

Definição

- Em muitas linguagens, o controle de erros é um trabalho penoso que pode transformar o software em um emaranhado de código fonte confuso.
- Java fornece um ra manipulação de erros que produ izado.
- Com esse mecar detectam erros facilmente, sem p special para testar os valores retorn



Terminologia

- Exceção é toda ocorrência que altera o fluxo normal do sistema. Coisas como falhas de hardware, exaustão do sistema e os famosos erros.
- Quando essa condição excepcional ocorre dizemos que uma exceção será lançada (throw).
- O código que a captura é chamado de manipulador de exceção e é responsável por fazer algo, como mostrar uma mensagem para o usuário.

Algumas situações

- O arquivo texto que você tentou abrir não existe
- A conexão com a rede não está ativada
- Os operandos sendo manipulados estão fora da faixa prescrita (tente colocar um long dentro de um inteiro)
- O arquivo de classe que você quer ler está faltando

Sintaxe

- Você deve colocar o bloco de código a ser protegido dentro de uma instrução try.
- Se houver algum erro em tempo de execução, o código será desviado para a instrução catch.
- Caso haja mais de uma exceção que deva ser verificada, podemos utilizar vários catch's diferentes, obedecendo a regra de que as exceções mais específicas devem vir primeiro.
- O bloco do finally é sempre executado, com ou sem exceção.

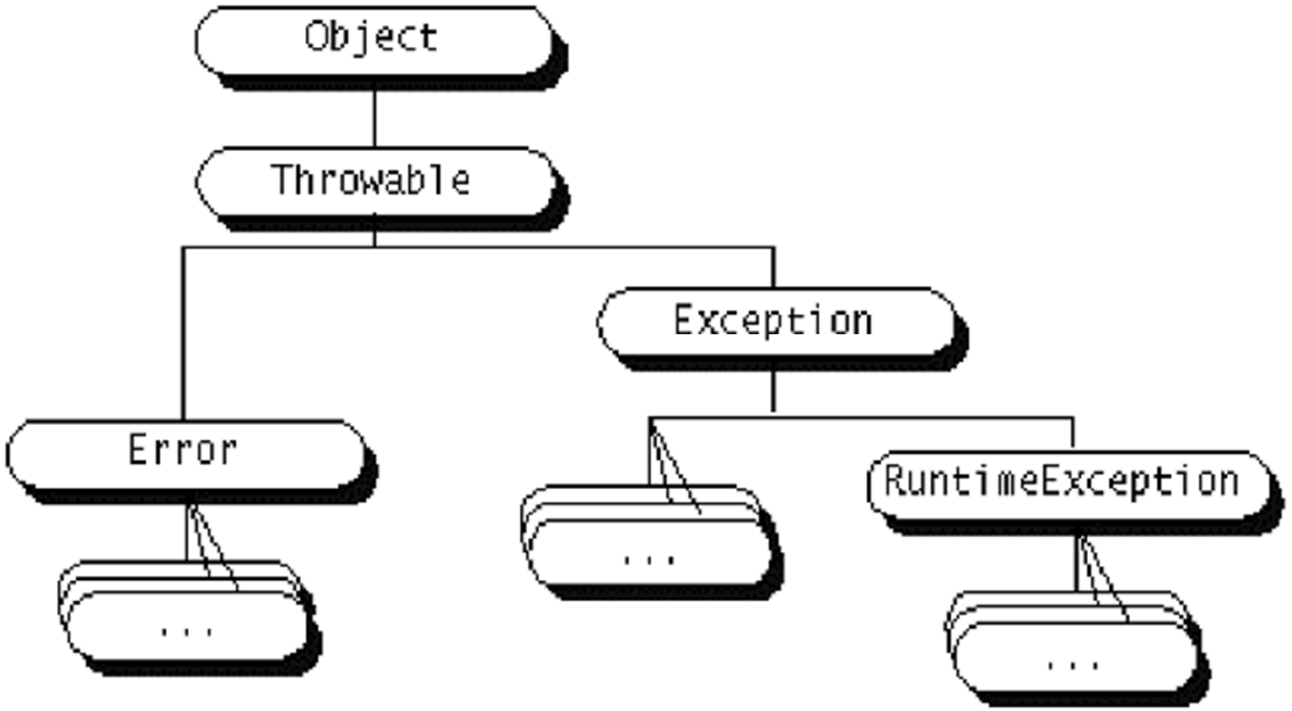
Sintaxe

```
try {
   //código protegido
} catch (UmaExcecao e) {
   //faça alguma coisa
} catch (SegundaExcecao e) {
   //faça outra coisa
} finally { //opcional
   //por fim, libere os recursos
}
```

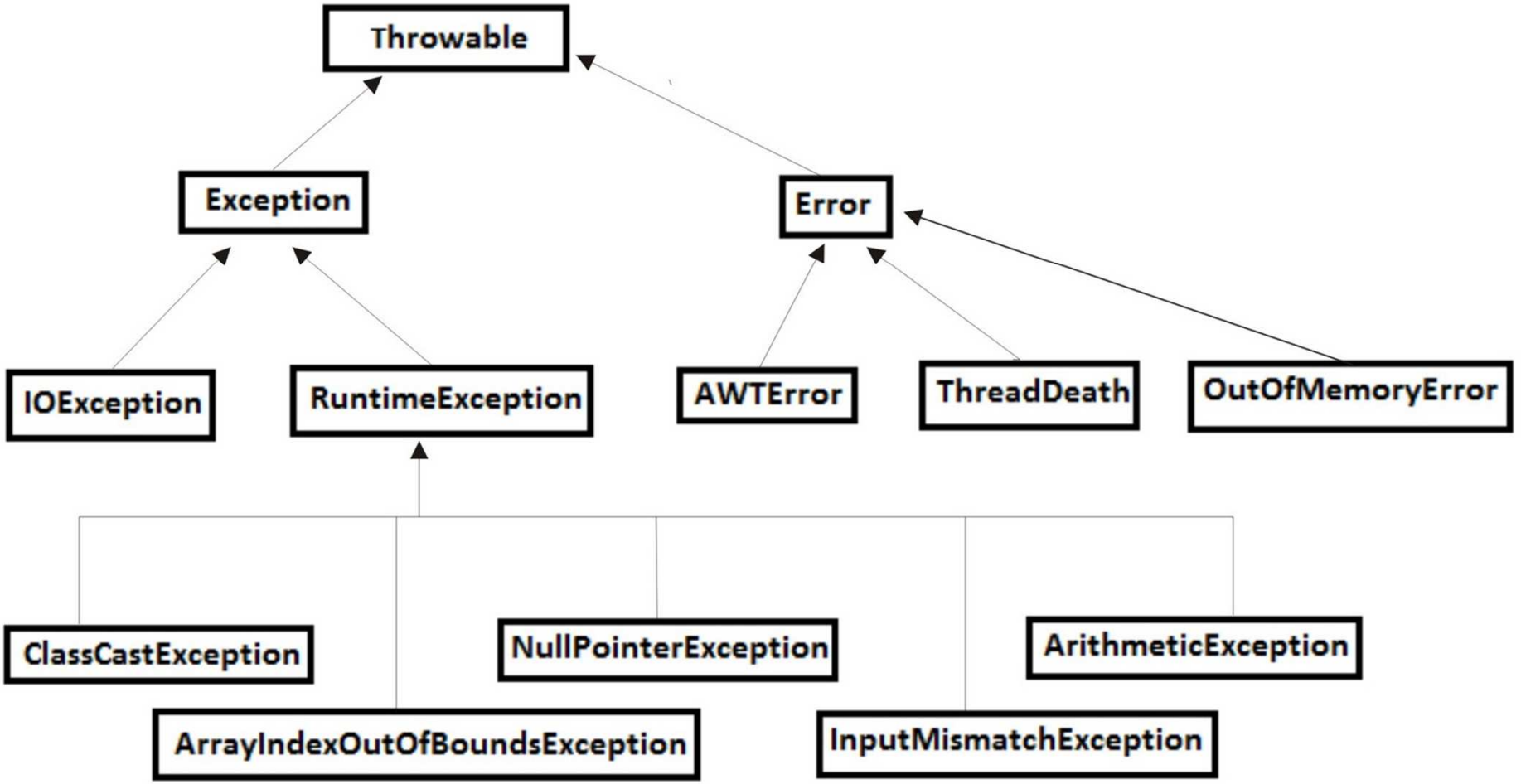
Mecanismo de pilha

- Se uma exceção não é tratada no bloco try-catch atual, ela é propagada para o método que invocou a rotina.
- Se uma exceção volta até o método main e não é tratada lá, o programa é terminado de forma anormal.

Hierarquia



Throwable


Throwable
Exception
Error
IOException
RuntimeException
AWTError
ThreadDeath
OutOfMemoryError
ClassCastException
NullPointerException
ArithmeticException
ArrayIndexOutOfBoundsException
InputMismatchException

Trowable, *Error* e suas subclasses

- O mecanismo de exceção começa com a interface *Throwable*.
- A classe *Error* e suas subclasses representam erros críticos do sistema e não podem ser tratados pelo programador.

Trabalhando sem tratamento de exceções

- Crie a classe TesteArray com um loop (for) lendo três elementos.

```
public class TesteArray {
  public static void main(String[] args) {
    String[] lista = {"Marco", "Luis", "Diego"};
    for (int i = 0; i < lista.length; i++) {
      System.out.println("Índice: " + i);
      System.out.println("Valor: " + lista[i]);
    }
  }
}
```

Forçando o erro

- Vamos forçar um erro utilizando um índice maior que o permitido.

```
public class TesteArray {
  public static void main(String[] args) {
    String[] lista = {"Marco", "Luis", "Diego"};
    for (int i = 0; i <= lista.length; i++) {
      System.out.println("Índice: " + i);
      System.out.println("Valor: " + lista[i]);
    }
  }
}
```

StackTrace do erro

- No console da IDE deve ser mostrada essa mensagem.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
At TesteArray.main(TesteArray.java:8)
```

- O próximo passo é adicionar o tratamento de exceções (try/catch).

- Devemos proteger somente a linha que apresenta que tem erro em potencial, no caso:

```
System.out.println("Valor: " + lista[i])
```

Exemplo: TesteArray

```
import javax.swing.*;

public class TesteArray {
  public static void main(String[] args) {
    String[] lista = {"Marco", "Luis", "Diego"};
    for (int i = 0; i <= lista.length; i++) {
      System.out.println("Índice: " + i);
      try {
        System.out.println("Valor: " + lista[i]);
      } catch (ArrayIndexOutOfBoundsException e) {
        JOptionPane.showMessageDialog(null,
                                      "Índice fora do array.");
      }
    }
  }
}
```

Exceções verificadas

- No exemplo seguinte temos um tipo diferente de exceção, são as exceções verificadas.
- Nesse caso o tratamento de exceções é obrigatório, no exemplo anterior era opcional (exceções não verificadas).
- Agora, para conseguir compilar o programa, você deve tratar a exceção.
- As exceções verificadas são subclasses de Exception.

Exceções não-verificadas

- São subclasses de RuntimeException.
- Não exigem tratamento, ou seja, o try/catch é opcional.

Exemplo: TestaCriacaoDeArquivos

```
import java.io.*;

public class TestaCriacaoDeArquivos {
  public static void main(String[] args) {
    File arquivo = new File("c:/NovoArquivo.txt");
    arquivo.createNewFile();
  }
}
```

Erro de compilação

- Posicione o mouse em cima do erro na linha selecionada.
- O erro indicado deve ser Unhandled Exception, ou seja, você não está manipulando a exceção.
- Devemos usar o tratamento de exceção. Nesse caso, a exceção utilizada é IOException.

Exemplo: TestaCriacaoDeArquivos

```
import java.io.*;

public class TestaCriacaoDeArquivos {

  public static void main(String[] args) {
    File arquivo = new File("c:/NovoArquivo.txt");
    try {
      arquivo.createNewFile();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

}
```

Forçando o erro

- Troque o driver do arquivo para uma letra inválida, no nosso caso “f:”.
- A JVM não consegue gravar nesse diretório e lança uma exceção.

Exemplo: TestaCriacaoDeArquivos

```
import java.io.*;
import javax.swing.*;

public class TestaCriacaoDeArquivos {

  public static void main(String[] args) {
    File arquivo = new File("f:/NovoArquivo.txt");
    try {
      arquivo.createNewFile();
    } catch (IOException e) {
      JOptionPane.showMessageDialog(null, e.getMessage());
    }
  }

}
```

Tratamento opcional

- Nas exceções não-verificadas o tratamento é opcional, mas seu programa fica desprotegido contra eventuais erros.
- A próxima classe é um exemplo simples de operação aritmética sem grandes novidades.

Exemplo: TesteAritmetico

```
public class TesteAritmetico {

  public static void main(String[] args) {
    long dividendo = 50;
    long divisor = 2;
    long quociente = dividendo / divisor;
    System.out.println("Quociente: " + quociente);
  }

}
```

Tratando a exceção

- Não existe nenhuma restrição no nosso programa que impeça a entrada de qualquer valor.
- Altere o divisor do nosso exemplo para 0.
- Rode o programa. O que aconteceu?
- Como proceder para evitar o erro?
- Implemente as correções.

Exemplo: TesteAritmetico

```
public class TesteAritmetico {

  public static void main(String[] args) {
    long dividendo = 50;
    long divisor = 0;
    long quociente = dividendo / divisor;
    System.out.println("Quociente: " + quociente);
  }

}
```

Exceções personalizadas

- O tratamento de exceção é composto basicamente de três passos: criação, declaração e gerenciamento da exceção.
- Atenção, pois todos os passos são essenciais para o tratamento das exceção:
  - Modelando a exceção através de uma classe
  - Detectar o comportamento anômalo e lançar uma exceção.
  - Gerenciar (tratar) a ocorrência de uma exceção.

Criando uma exceção

- As nossas exceções personalizadas serão subclasses de Exception.
- Uma exceção pode conter preciosas informações sobre o comportamento do sistema.
- Todas as vezes que alguma situação diferente acontecer, o usuário deve ser informado.

Declarando exceções

- Quando você escreve um método potencialmente perigoso, deve declarar que esse método pode levantar alguma exceção com o throws.
- A maneira mais simples de lançar uma exceção é usando a palavra-chave throw.
- Se estiver lançando uma exceção dentro de um método, declare-a logo após a assinatura do mesmo.

Gerenciando exceções

- Quando qualquer classe chama um método que declara exceções, podemos gerenciar essa exceção com a utilização do try/catch e, opcionalmente, finally.

Locadora

- É um exemplo simples, mas bastante interessante pois ilustra bem o processo de modelagem, geração e tratamento de exceções.
- Neste nosso exemplo criaremos exceções não-verificadas, ou seja, que não obrigarão que as chamadas ao método sejam obrigatoriamente inserida em um bloco try/catch.

Filme

```
public class Filme {
  /* Armazena o título do filme */
  private String titulo;

  /* Método de acesso para o atributo título */
  public String getTitulo() {
    return titulo;
  }

  /* Método modificador para o atributo título */
  public void setTitulo(String titulo) {
    this.titulo = titulo;
  }
}
```

Cliente

```
public class Cliente {
  /* Armazena o nome do cliente*/
  private String nome;

  /* Método de acesso */
  public String getNome() {
    return nome;
  }

  /* Método modificador */
  public void setNome(String nome) {
    this.nome = nome;
  }
}
```

Locadora

```
public class Locadora {

  /* Este método representa o operação de locação de um filme
  por um cliente */
  public void locar(Filme filme, Cliente cliente) {
    String mensagem = "O filme " + filme.getTitulo() +
         " está com o cliente " + cliente.getNome();
    System.out.println(mensagem);
  }

}
```

TesteLocadora

```
public class TesteLocadora {
  public static void main(String[] args) {

    Cliente c = new Cliente();
    c.setNome("Marco");

    Filme f = new Filme();
    f.setTitulo("Casino Royale");

    Locadora l = new Locadora();
    l.locar(f, c);
  }
}
```

Criando exceções personalizadas

- Para criar uma exceção não-verificada para sua aplicação devemos criar uma subclasse de RuntimeException, seguindo o modelo a seguir.
- A sintaxe `super(mensagem)` passa a mensagem de erro para a superclasse (neste caso, RuntimeException).
- Essa mensagem será informada mais tarde na aplicação.

FilmeJaLocado

```
public class FilmeJaLocadoException extends RuntimeException {
  public FilmeJaLocadoException(String mensagem) {
    super(mensagem);
  }
}
```

ClienteComRestricaoException

```
public class ClienteComRestricaoException extends
  RuntimeException {
  public ClienteComRestricaoException(String mensagem) {
    super(mensagem);
  }
}
```

Lançando exceções – throw

- Quando queremos lançar exceções personalizadas utilizamos a palavra reservada throw, passando a respectiva mensagem.
- Quando o nome do cliente for Marco a locadora não vai poder emprestar o filme.

Locadora

```
public class Locadora {
  public void locar(Filme filme, Cliente cliente) {
    /* Detecta a situação imprópria e lança uma exceção */
    if (cliente.getNome().equals("Marco")) {
      String mensagem = "O cliente " + cliente.getNome() +
                        " tem algum tipo de restrição.";
      throw new ClienteComRestricaoException(mensagem);
    }
    /* (Fluxo normal) Procede a tarefa normalmente */
    String mensagem = "O filme " + filme.getTitulo() +
                  " está com o cliente " + cliente.getNome();
    System.out.println(mensagem);
  }
}
```

Veja o resultado

- Rode novamente a classe TesteLocadora.
- Em seguida, vamos adicionar a próxima exceção.

Locadora

```
public class Locadora {
  public void locar(Filme filme, Cliente cliente) {
    /* Detecta uma situação imprópria qto ao cliente */
    if (cliente.getNome().equals("Marco")) {
      String mensagem = "O cliente " + cliente.getNome() +
                        " tem algum tipo de restrição.";
      throw new ClienteComRestricaoException(mensagem);
    }
    /* Detecta uma situação imprópria qto ao filme */
    if (filme.getTitulo().equals(“Casino Royale")) {
      throw new FilmeJaLocadoException("O filme " +
                      filme.getTitulo() + " já está locado.");
    }
    /* (Fluxo normal) Procede a locação */
    String mensagem = "O filme " + filme.getTitulo() +
                  " está com o cliente " + cliente.getNome();
    System.out.println(mensagem);
  }
}
```

Try/catch

- Para evitar que a aplicação apresente esse erro, devemos proteger o código problemático com um try/catch.

TesteLocadora

```
import javax.swing.*;

public class TesteLocadora {
  public static void main(String[] args) {
    Cliente c = new Cliente();
    c.setNome("Marco1");
    Filme f = new Filme();
    f.setTitulo("Casino Royale");
    Locadora l = new Locadora();
    try {
      l.locar(f, c);
    } catch (ClienteComRestricaoException e) {
      JOptionPane.showMessageDialog(null, e.getMessage());
    }
  }
}
```

TesteLocadora

```
import javax.swing.*;

public class TesteLocadora {
  public static void main(String[] args) {
    Cliente c = new Cliente();
    c.setNome("Marco1");
    Filme f = new Filme();
    f.setTitulo("Casino Royale");
    Locadora l = new Locadora();
    try {
      l.locar(f, c);
    } catch (ClienteComRestricaoException e) {
      JOptionPane.showMessageDialog(null, e.getMessage());
    } catch (FilmeJaLocadoException e) {
      JOptionPane.showMessageDialog(null, e.getMessage());
    }
  }
}
```

Exceções verificadas

- No próximo exemplo veremos as exceções verificadas.
- Todas serão subclasse de Exception.

SaldoInsuficienteException

```
public class SaldoInsuficienteException extends Exception {
  public SaldoInsuficienteException(String mensagem) {
    super(mensagem);
  }
}
```

ContaCorrente

```
public class ContaCorrente {
  private double saldoAtual;
  public void depositar(double valorDepositado) {
    saldoAtual += valorDepositado;
  }
  public void sacar(double valorSacado)
                                  throws SaldoInsuficienteException {
    if (valorSacado > saldoAtual) {
      throw new SaldoInsuficienteException(
                          "Saldo insuficiente para esta operação");
    }
    saldoAtual -= valorSacado;
  }
}
```

TerminalBancario

```
import javax.swing.JOptionPane;

public class TerminalBancario {
  public static void main(String[] args) {
    ContaCorrente c = new ContaCorrente();
    c.depositar(120);
    try {
      c.sacar(1200);
      JOptionPane.showMessageDialog(null,
                            "Operação realizada com sucesso.");
    } catch (SaldoInsuficienteException e) {
      JOptionPane.showMessageDialog(null, e.getMessage(),
                             "Atenção", JOptionPane.ERROR_MESSAGE);
    }
  }
}
```

Faça alguns testes

- Tire o try/catch e recompile o TerminalBancario.
- Analise o resultado.
- Esse resultado era esperado?
- Qual dos três passos está faltando?

Exercícios

Exercícios

- Crie as seguintes classes de exceção:
  - ValorAbaixoException,
  - ValorAcimaException e
  - ValorMuitoAcimaException.

Exceções

```
//Lembre-se que cada classe deve ser colocada em um arquivo diferente
public class ValorAbaixoException extends Exception {
  ValorAbaixoException(String mensagem) {
    super(mensagem);
  }
}
public class ValorAcimaException extends Exception {
  public ValorAcimaException(String mensagem) {
    super(mensagem);
  }
}
public class ValorMuitoAcimaException extends Exception {
  public ValorMuitoAcimaException(String mensagem) {
    super(mensagem);
  }
}
```

Exercícios

- Crie a classe ValidaNumero com um novo método chamado analisaValor(double valorAnalisado)

```java
package com.javabasico.tratamentodeexcecao;

public class ValidaNumero {
  public void analisaValor(double valorAnalisado) throws
  ValorAbaixoException,
      ValorAcimaException, ValorMuitoAcimaException {
    if (valorAnalisado <= 0) {
      throw new ValorAbaixoException("Valor abaixo");
    } else if (valorAnalisado > 100 && valorAnalisado < 1000) {
      throw new ValorAcimaException("Valor acima");
    } else if (valorAnalisado >= 1000) {
      throw new ValorMuitoAcimaException("Valor muito acima");
    }
  }
}
```

- Crie a classe TestaValidaNumero
- Para cada faixa de valores descritas abaixo, deverá lançar as respectivas exceções mostrado mensagens para o usuário.
- Ao final, ocorrendo exceções ou não, deverá mostrar uma mensagem para o usuário informando que a análise foi terminada.
  - Menor ou igual a 0:ValorAbaixoException
  - Maior que 100 e menor que 1000:ValorAcimaException
  - Maior ou igual a 1000:ValorMuitoAcimaException

```
package com.javabasico.tratamentodeexcecao;
import javax.swing.*;
public class TesteValidaNumero {
  public static void main(String[] args) {
    ValidaNumero validador = new ValidaNumero();
    try {
      validador.analisaValor(1590);
    } catch (ValorAbaixoException e) {
      e.printStackTrace();//ou
      JOptionPane.showMessageDialog(null, e.getMessage());
    } catch (ValorAcimaException e) {
      e.printStackTrace();
      JOptionPane.showMessageDialog(null, e.getMessage());
```

```
    } catch (ValorMuitoAcimaException e) {
      e.printStackTrace();
      JOptionPane.showMessageDialog(null, e.getMessage());
    }
    String mensagem = "A análise já foi concluída";
    System.out.println(mensagem); // ou
    JOptionPane.showMessageDialog(null, mensagem);
  }
}
```