INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

AULA:

Tratamento de Exceção em Java

Programação Orientada a Objetos

Alba Lopes, Profa.

http://docentes.ifrn.edu.br/albalopes
alba.lopes@ifrn.edu.br

O que são exceções?

- Uma **exceção** é uma condição anormal que altera ou interrompe o fluxo de execução.
- Podem ser causadas por diversas condições:
  - Erros sérios de hardware;
  - Erros simples de programação;
  - Erros de divisão por zero;
  - Valores fora de faixa
  - Valores de variáveis
  - Erro na procura/abertura de um arquivo (entrada/saída)
  - Falha na memória

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

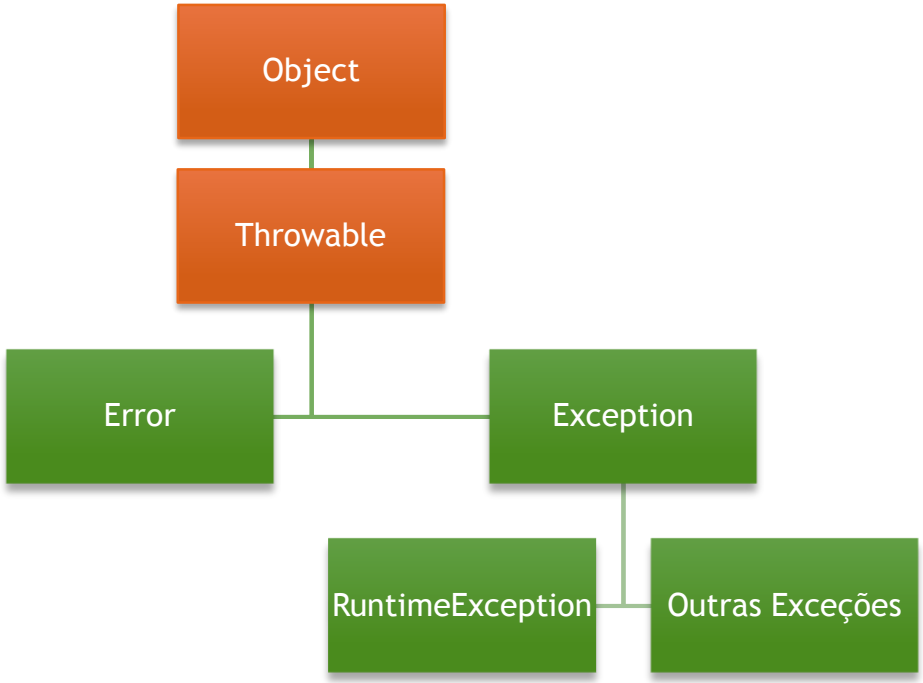
O que são exceções?

- Se não for tratada, o programa pode parar
- O uso correto de exceções torna o programa mais robusto e confiável
  - Uso exagerado polui o código e torna o programa mais lento

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Tipos de exceção

- Exceções, como (quase) tudo em Java, são objetos;



Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Tipos de Exceções

- Exceções e erros se enquadram em **três categorias**:
  - Exceções verificadas;
  - Exceções não verificadas e;
  - Erros.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exceções Verificadas

- São **verificadas pelo compilador em tempo de compilação**;
- Os métodos que lançam uma exceção devem indicar isso na declaração do método através da cláusula **`throws`**.
- Todas as exceções verificadas devem ser capturadas explicitamente com um bloco **`try – catch`**;
- Exceções do tipo **`Exception`** e todos os seus subtipos, exceto **`RunTimeException`** e os seus subtipos.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exceções Não Verificadas

- **Não verificadas em tempo de compilação**;
- Ocorrem **durante a execução por causa de erro do programador** (divisão por zero, índice fora de faixa e uso de referências null);
- Não exigem tratamento;
- As exceções não verificadas incluem as exceções do tipo **`RunTimeException`** e todos os seus subtipos.
- A regra “Se for um RunTimeException, a culpa é sua” funciona muito bem.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Erros

- Os erros **são tipicamente irrecuperáveis e apresentam condições sérias.**
- Os erros não são verificados em tempo de compilação;

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Palavras-chave

- Em Java, o código para tratamento de erros é separado de forma limpa do código que gera erros.
  - Diz-se que **o código que gera a exceção "lança" ("`throw`") uma exceção**,
  - enquanto que **o código que trata a exceção "captura" ("`catch`") a exceção.**

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Testando e capturando exceções

- Um bloco que tenta (try) chamar um método (ou conjunto de classes) que pode disparar uma exceção deve trata-la
  - Chamada normal de um método, mas que deve estar em um bloco try {...} catch {...}
- Uma exceção é um objeto que deve ser capturado (catch)
  - É nesse bloco que a exceção deve ser tratada
- Um trecho de código pode ser executado sempre
  - bloco finally

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Testando e capturando exceções

```
try
{
    //Executa código que pode disparar exceção
}
catch(Exception ex)
{
    //Trata exceção. ex é uma referência para o
    objeto da classe Excpetion a ser tratado
}
finally
{
    //Esse bloco é sempre executado,
    independente de ocorrer exceção.
}
```

Tenta Executar

Captura exceção

Sempre executa

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exceções Frequentes

- ArithmeticException
- ClassNotFoundException
- DataFormatException
- FileNotFoundException
- IndexOutOfBoundsException
- NullPointerException
- NumberFormatException
- SQLException

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Testando e capturando exceções

- Se a exceção for disparada, o bloco **try** não será mais executado a partir do ponto onde a exceção ocorreu
- O fluxo de execução muda para a captura **catch**
- Pode haver mais de um bloco **catch**. A exceção é capturada pelo bloco que tratar a exceção correspondente.
- O bloco **finally** é opcional.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Testando e capturando exceções

```
int x = 0;
try{
      int y = 100 / x;
      System.out.println ("Resultado: " + y);
}
catch (ArithmeticException e){
      System.out.println ("Operação inválida!");
      System.out.println("\n Detalhes do erro: "+ e.getMessage());
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Testando e capturando múltiplas exceções

```
Scanner sc = new Scanner(System.in);
try{
      int x = sc.nextInt();
      int y = 100 / x;
      System.out.println ("Resultado: " + y);
}
catch (InputMismatchException e){
      System.out.println ("Formato inválido!");
      System.out.println("\n Detalhes do erro:"+ e.getMessage());
}
catch (ArithmeticException e){
      System.out.println ("Operação inválida!");
      System.out.println("\n Detalhes do erro:"+ e.getMessage());
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

java.lang.Throwable

- Ancestral de todas as classes que recebem tratamento do mecanismo de exceções;
- Principais métodos:
  - **void printStackTrace():** lista a seqüência de métodos chamados até o ponto onde a exceção foi lançada;
  - **String getMessage():** contém uma mensagem indicadora da exceção;
  - **O método toString():** retorna uma descrição breve da exceção.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

java.lang.Error

- Representa um **problema grave**, de difícil (ou impossível) recuperação;
- Exemplos:
  - **OutOfMemoryError**
  - **StackOverflowError**
  - **VirtualMachineError**
  - etc.
- Geralmente causam o encerramento do programa;
- Não devem ser usadas pelos programadores.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

java.lang.Exception

- Exceções que **podem ser lançadas pelos métodos da API Java ou pelo seu programa**;
- Devem ser tratadas;
- Em geral, representam situações inesperadas, porém contornáveis;
- O programador tem contato com esta classe e suas subclasses.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

java.lang.RuntimeException

- Tipo especial de exceção;
- Não necessitam ser lançadas explicitamente pelo programa;
- Seu tratamento não é obrigatório; Ex.:
  - **NullPointerException,**
  - **IndexOutOfBoundsException,** etc.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exemplo (NumberFormatException)

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

    String var = JOptionPane.showInputDialog("Digite um número inteiro:");

    try {

        Integer i = new Integer(var);
        JOptionPane.showMessageDialog(null, "O número digitado foi: " + i);

    } catch (NumberFormatException nfe) {

        JOptionPane.showMessageDialog(null,
                "Não é possível atribuir esse valor ao número inteiro");

        System.out.println("Erro: A seguinte mensagem foi retornada:\n"+
                nfe.getMessage());
    }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exemplo (IndexOutOfBoundException)

```
public static void main(String[] args) {
    int [] values = {0, 10, 20, 30, 40, 50};

    String var = JOptionPane.showInputDialog("Digite um número: ");
    try {
       int i = Integer.parseInt(var);
       JOptionPane.showMessageDialog(null,
               "O valor na posição "+ i + " é: " + values[i]);


    } catch (IndexOutOfBoundsException iob) {

        JOptionPane.showMessageDialog(null,
                "Não é possível acessar esse índice no vetor");
        iob.printStackTrace();
    }
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

A palavra-chave throw

- Para lançar uma exceção, use a palavra **`throw`**. Qualquer exceção verificada/não verificada, ou erro, pode ser lançado.

```
if (arquivo == null)
  throw new FileNotFoundException();
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

A palavra-chave throws

- O lugar onde você anuncia qual método pode lançar uma exceção é o cabeçalho do método;
- O cabeçalho muda para refletir as exceções verificadas que o método pode lançar.

```
public FileInputStream(String name) throws FileNotFoundException {
   //...
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

A cláusula finally

- O código na cláusula finally executa se uma exceção foi ou não capturada.

```
int x = 0;
try{
      int y = 100 / x;
      System.out.println ("Resultado: " + y);
}
catch (ArithmeticException e){
      System.out.println ("Operação inválida!");
      System.out.println("\n Detalhes do erro: "+ e.getMessage());
}
finally{
      System.out.println ("Execução finalizada!");
}
```

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Exercícios

1. Receba uma String via JOptionPane e mostre o valor todo em letras maiúsculas. Trate a exceção para o caso de nenhum valor ter sido digitado para a String. Trate a exceção NullPointerException.

2. Crie um formulário JFrame em Java que receba 2 valores e efetua a divisão do primeiro pelo segundo. Trate, via exceção, o caso de formato de número inválido e divisão por zero. Mostre as mensagens em *JOptionPane*s.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br

Referências

- SIERRA, Katy; BATES, Bert. **Use a cabeça JAVA**. Ed 2, Editora Altabooks;
- Material do Curso de Orientação a Objetos dos Professores **Fábio Procópio e João Silva:** Java – Tratamento de Exceções
- Material do Curso de Ambiente de Programação do Professore **Edmilson Campos:** Tratamento de Exceções em C#.

Alba Lopes, Profa.
alba.lopes@ifrn.edu.br