Programação Orientada a Objetos
Tratamento de Exceções

Alexandre Mello

Fatec Campinas

2024

Roteiro

Introdução

- Erros irão acontecer
- Java oferece uma forma de captura de erros de tratamento de exceções
- Objetivo: capturar os erros antes que eles aconteçam
- Remoção do código de tratamento de erros da "linha principal", melhorando legibilidade
- Assim, o erro deve ser tratado por uma parte diferente do programa que o descobriu

Exceções em Java

- Em Java, as exceções são divididas em duas categorias: **Unchecked** e **Checked**

- **Unchecked** (não verificadas): o compilador NÃO verifica o código-fonte e, por isso, o seu tratamento é opcional. Exemplo: conversão de String para inteiro
- **Checked** (verificadas): o compilador verifica o código-fonte obrigando o programador a tratá-la

- Ambas as exceções podem ser tratadas de duas maneiras: estrutura `try-catch-finally` e ou cláusula `throws`

Lançando Exceções

- Para informar o erro, o método onde o mesmo ocorreu encapsula em um objeto a informação de erro e **lança** esse objeto (o objeto é encerrado)
- Em Java, um objeto de exceção é sempre uma instância de uma classe derivada de **Throwable**
- **Throwable** é a base de outras duas classes: **Error** e **Exception**
- **Error**: problema interno e não deve ser lançada explicitamente
- **Exception**: **RuntimeException** (erros de programação) e **IOException** (erro de entrada de dados)

Lançando Exceções

- As exceções que um método pode lançar são listadas junto ao cabeçalho desse método
- Para lançar uma exceção, utilizamos a palavra reservada **throw**
- Também é possível passar uma mensagem de erro

```
public class Data {
   public void setData(int dia, int mes, int ano) throws
       IOException {
      ...
      if (mes < 0 || mes > 12) {
         throw new IOException("Mês inválido!");
      }
      ...
   }
}
```

Criando uma classe de exceção

- Ao invés de utilizarmos classes existentes, podemos criar uma classe de exceção que represente o erro
- Para isso, basta que a nova classe herde de **Exception**

```
public class ValorInvalido extends IOException {
   public ValorInvalido () {
   }
   public ValorInvalido (String msg) {
      super(msg);
   }
}
```

Criando uma classe de exceção

```
public class Data {
   public void setData(int dia, int mes, int ano) throws
       ValorInvalido {
      ...
      if (mes < 0 || mes > 12) {
         throw new ValorInvalido("Mês inválido!");
      }
      ...
   }
}
```

Capturando Exceções

- Após lançar uma exceção, é necessário tratá-la em algum lugar
- Para isso, define-se um bloco **try catch**

```
try {
   Data d = new Data();
   d.setData(26, 100, 2018);
} catch (ValorInvalido e) {
   System.out.println(e.getMessage);
}
```

Capturando Exceções

- Caso seja usado um método que lance uma exceção dentro de outro, mas não se queira tratar tal exceção, esse novo método deve declarar que essa exceção pode ser lançada

```
public void teste() throws ValorInvalido {
    ...
    Data d = new Data();
    d.setData(26, 10, 2018);
    ...
}
```

A cláusula *finally*

- Uma cláusula **finally** sempre é executada em um método, sendo lançada ou não a exceção

```
Graphics g = image.getGraphics();
try {
   ...
} catch (IOException e) {
   ...
} finally {
   g.dispose();
}
```

Exercício 1

Dado o seguinte trecho de código Java, em que condições de execução será exibida na tela a mensagem "Alô Mundo!"?

```
try {
   ...
} catch (Exception e) {
} finally {
   System.out.println("Alô Mundo!");
}
```

Exercício 2

O método abaixo realiza a validação de itens. Se o item for "vazio", retorna falso, senão, retorna verdadeiro. Infelizmente, se o item for null, um erro em tempo de execução (NullPointerException) ocorrerá. Altere o método para anunciar tal exceção e para lança-la, caso o teste de nulo seja verdadeiro

```
public boolean validaItem (String item) {
   if (item.equals("")) return false;
   else return true;
}
```