

Tratamento de Exceções

Cláudio T. Kawakani
Prof. Dr. Bruno Bogaz Zarpelão

Universidade Estadual de Londrina
Departamento de Computação

Objetivos

- * O que são exceções.
- * Como utilizar try (detecta), throw (indica) e catch (trata).
- * Utilidade do bloco finally.
- * Como funciona o rastreamento de pilha para analisar a exceção.
- * Como as classes de exceção estão organizadas.
- * Como criar exceções encadeadas.

Introdução

- * Uma **exceção** é uma indicação de um problema que **pode ocorrer** durante a execução de um programa.
- * Quando não tratada, o programa pode encerrar de forma não controlada ou continuar executando de forma indesejada.

Exemplos de exceção

- * `ArrayIndexOutOfBoundsException`: acessar um elemento fora de um vetor.
- * `NullPointerException`: acessar um elemento null.
- * `ArithmeticException`: por exemplo, divisão de inteiros por zero.
- * `InputMismatchException`: método `nextInt` exige um inteiro.

Exercício 1 - Gerando Exceções

```
public class Exercicio1 {  
    public static void main (String[] args){  
        Scanner scanner = new Scanner(System.in);  
  
        System.out.print("Insira o numerador: ");  
        int numerador = scanner.nextInt();  
        System.out.print("Insira o denominador: ");  
        int denominador = scanner.nextInt();  
  
        int resultado = numerador/denominador;  
  
        System.out.printf("\nResult: %d / %d = %d\n", numerador,  
denominador, resultado);  
        System.out.println("Chegou no fim do programa.");  
    }  
}
```


Exercício 1 - Gerando Exceções

Exception in thread "main" java.lang.ArithmeticException: / by zero
at Exercicio1.main(Exercicio1.java:15)

Exercício 1 - Gerando Exceções

- * Neste mesmo código, é possível gerar uma outra exceção do tipo `InputMismatchException`.
- * Insira uma string ou um float.

Visão geral do tratamento de exceção

- * O tratamento de exceção separa a porção de código que pode causar exceções em dois blocos:
 1. Try: **detecta** uma exceção.
 2. Catch: **trata** a exceção detectada no bloco try. Deve existir um bloco catch para cada tipo de exceção.
- * Essa separação melhora a **clareza** do código.

Exercício 2 - Tratando as Exceções com Try e Catch

- * Trate as exceções utilizando as instruções try e catch.
- * O objetivo é pedir novos valores até que o usuário insira um valor válido.

Exercício 3 - Utilizando a instrução throw

- * A instrução throw indica que uma exceção pode ocorrer **dentro** de um método.

```
try{
```

```
    metodo();
```

```
} catch ...
```

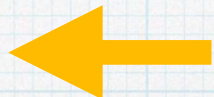
Dentro deste método
pode existir uma exceção



Ordem de execução quando ocorre uma exceção

* Modelo de terminação de tratamento e exceção

```
try{  
    System.out.println("Começo do bloco TRY");  
    int numerador = 10;  
    int denominador = 0;  
  
    System.out.println("Executa a instrução que gera exceção");  
    int resultado = numerador/denominador;  
  
    System.out.println("Final do bloco TRY");  
}  
catch (InputMismatchException inputMismatchException){  
    System.out.println("Trata o erro de Input");  
}  
catch(ArithmeticException arithmeticException){  
    System.out.println("Trata o erro de divisão por zero");  
}  
System.out.println("Chegou no fim do programa.");
```



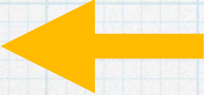
Ordem de execução quando ocorre uma exceção

* Modelo de terminação de tratamento e exceção

```
try{
    System.out.println("Começo do bloco TRY");
    int numerador = 10;
    int denominador = 0;

    System.out.println("Executa a instrução que gera exceção");
    int resultado = numerador/denominador;

    System.out.println("Final do bloco TRY");
} catch (InputMismatchException inputMismatchException){
    System.out.println("Trata o erro de Input");
} catch(ArithmeticException arithmeticException){
    System.out.println("Trata o erro de divisão por zero");
}
System.out.println("Chegou no fim do programa.");
```



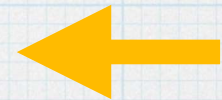
Ordem de execução quando ocorre uma exceção

* Modelo de terminação de tratamento e exceção

```
try{
    System.out.println("Começo do bloco TRY");
    int numerador = 10;
    int denominador = 0;

    System.out.println("Executa a instrução que gera exceção");
    int resultado = numerador/denominador;

    System.out.println("Final do bloco TRY");
} catch (InputMismatchException inputMismatchException){
    System.out.println("Trata o erro de Input");
} catch(ArithmeticException arithmeticException){
    System.out.println("Trata o erro de divisão por zero");
}
System.out.println("Chegou no fim do programa.");
```




Ordem de execução quando ocorre uma exceção

* O programa detecta uma exceção no bloco try.

```
try{
    System.out.println("Começo do bloco TRY");
    int numerador = 10;
    int denominador = 0;

    System.out.println("Executa a instrução que gera exceção");
    int resultado = numerador/denominador;

    System.out.println("Final do bloco TRY");
} catch (InputMismatchException inputMismatchException){
    System.out.println("Trata o erro de Input");
} catch(ArithmeticException arithmeticException){
    System.out.println("Trata o erro de divisão por zero");
}
System.out.println("Chegou no fim do programa.");
```



Ordem de execução quando ocorre uma exceção

* O bloco catch correspondente é chamado imediatamente

```
try{
    System.out.println("Começo do bloco TRY");
    int numerador = 10;
    int denominador = 0;

    System.out.println("Executa a instrução que gera exceção");
    int resultado = numerador/denominador;

    System.out.println("Final do bloco TRY");X
} catch (InputMismatchException inputMismatchException){ X
    System.out.println("Trata o erro de Input");X
} catch(ArithmeticException arithmeticException){ ←
    System.out.println("Trata o erro de divisão por zero");
}
System.out.println("Chegou no fim do programa.");
```

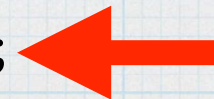

Ordem de execução quando ocorre uma exceção

* O bloco catch é chamado imediatamente

```
try{
    System.out.println("Começo do bloco TRY");
    int numerador = 10;
    int denominador = 0;

    System.out.println("Executa a instrução que gera exceção");
    int resultado = numerador/denominador;

    System.out.println("Final do bloco TRY");
} catch (InputMismatchException inputMismatchException){
    System.out.println("Trata o erro de Input");
} catch(ArithmeticException arithmeticException){
    System.out.println("Trata o erro de divisão por zero");
}
System.out.println("Chegou no fim do programa.");
```




Ordem de execução quando ocorre uma exceção

* O programa continua após o bloco catch.

```
try{
    System.out.println("Começo do bloco TRY");
    int numerador = 10;
    int denominador = 0;

    System.out.println("Executa a instrução que gera exceção");
    int resultado = numerador/denominador;

    System.out.println("Final do bloco TRY");
} catch (InputMismatchException inputMismatchException){
    System.out.println("Trata o erro de Input");
} catch(ArithmeticException arithmeticException){
    System.out.println("Trata o erro de divisão por zero");
}
System.out.println("Chegou no fim do programa.");
```




Ordem de execução quando ocorre uma exceção

* Modelo de terminação de tratamento e exceção

```
try{
    System.out.println("Começo do bloco TRY");
    int numerador = 10;
    int denominador = 0;

    System.out.println("Executa a instrução que gera exceção");
    int resultado = numerador/denominador;

    System.out.println("Final do bloco TRY");
} catch (InputMismatchException inputMismatchException){
    System.out.println("Trata o erro de Input");
} catch(ArithmeticException arithmeticException){
    System.out.println("Trata o erro de divisão por zero");
}
System.out.println("Chegou no fim do programa.");
```



Exercício 4 - Modelo de terminação

- * Quais mensagens vão aparecer no console?

Exercício 5 - Exceções Verificadas

- * Faremos um programa que lê um arquivo que não existe.

Bloco Finally

- * É um bloco que **sempre*** será executado após o bloco try.
- * É opcional e é colocado após o bloco catch.
- * Tem como objetivo **garantir** a liberação de recursos alocados no bloco try:
 - A. Fechar conexões com banco de dados.
 - B. Fechar conexões de redes.
 - C. Fechar arquivos.

Exercício 6 - Bloco Finally

- * Faremos um programa que utiliza o bloco finally.

Instrução throw

- * É possível chamar uma exceção diretamente por meio da instrução **throw**.
- * (Não confundir com a instrução **throws**).
- * É útil, por exemplo, para indicar que um objeto foi instanciado inadequadamente.

Métodos Throwable

- * São métodos para obter informações da exceção.
- * `printStackTrace`: imprime a pilha de rastreamento.
- * `getStackTrace`: obtém informações da pilha de rastreamento.
- * `getMessage`: obtém a mensagem da exceção.

Exercício Final

- * Crie o método **readFile** que deve receber como parâmetro uma String contendo o nome de um arquivo txt que possui dois números inteiros.
- * O método deve: abrir o arquivo utilizando as classes **Scanner** e **File**; Preencher um vetor de dois inteiros com o conteúdo do arquivo (utilize `nextInt()`); Retornar este vetor.
- * No main: chame o método **readFile**; Imprima os dois valores do vetor.
 1. Detecte e trate as exceções **não verificadas** usando try catch.
 2. **Lance** a exceção **verificada** para ser detectada e tratada no main.
 3. Para o "tratamento" das exceções no bloco catch, simplesmente utilize **`System.out.println("Exceção " + k);`** onde k é o nome do parâmetro utilizado no catch.
 4. Garanta que o arquivo seja fechado, mesmo que ocorra exceção.

Referências

- * Java - Como Programar - 8ª Ed. 2010,
HARVEY M. DEITEL & PAUL J. DEITEL