

PROGRAMAÇÃO I - CCT0827

Semana Aula: 9

Unidade 3. Tratamento de Exceções

Tema

Tratamento de Exceções

Palavras-chave

Exceção, try/catch, finally, throw, throws

Objetivos

O aluno deverá ser capaz de:

- Compreender o conceito de exceção.
- Compreender e aplicar o conceito de captura de exceção com try/catch.
- Compreender e aplicar o conceito de lançar exceção com throw.
- Compreender e aplicar o conceito de declarar exceção com throws.

Estrutura de Conteúdo

Introdução

Como tratar, em Java, as seguintes situações?

- Divisão por zero.
- Erro na conversão de tipos (por exemplo, converter uma string que só contém letras em número).
- Erro na abertura de um arquivo, entre outros.

Todas essas situações em Java são chamadas de exceções e existe um mecanismo específico para tratá-las.

Exemplo:

```
public class DividePorZero {  
    public static void main(String args[]) {  
        System.out.println(3/0);  
    }  
}
```

```
        System.out.println("imprime");
    }
}
```

Observamos a seguintes mensagens:

Exception in thread "main" java.lang.ArithmeticException: / by zero at DividePorZero.main(DividePorZero.java:3)

Note:

java.lang.ArithmeticException: nome da exceção
/ by zero: descrição da exceção

Ocorreu na main, no arquivo DividePorZero.java linha 3.

As exceções em Java estão organizadas em uma hierarquia de classes : No topo da hierarquia temos a classe Throwable. Suas classes filhas são Error e Exception. Veja a hierarquia completa em aula com seu professor.

O que acontece quando ocorre uma exceção?

- 1) O método cria um objeto do tipo Exception e o envia para a JVM:
 - Esse processo é chamado de "disparar uma exceção" (throw an exception)
 - O objeto Exception criado contém todas as informações sobre o erro: seu tipo, o local onde ocorreu, uma mensagem de descrição, a pilha de chamadas, etc.
- 2) A JVM procura na pilha de chamadas dos métodos quem trata aquela exceção e não encontra.

O tratamento de exceções é um mecanismo que permite que o programa defina como as situações inesperadas serão tratadas.

Com relação ao tratamento de exceções temos:

1. Blocos try...catch...finally
2. Comando throws
3. Comando throw

Bloco try...catch

```
try {
    // Código a ser tratado
} catch(Exception e) {
    // Código que será executado quando ocorrer a exceção
}
```

```
}
```

Nota:

1. Se ocorrer uma exceção no bloco do try, então a execução é automaticamente desviada para o bloco **catch**.
2. No catch devemos definir a exceção a ser tratada. Quando definimos uma exceção estamos tratando também todas as suas subclasses.
3. O **e**, mostrado na linha do catch, referência a exceção que ocorreu. Com ela é possível acessar informações sobre essa exceção.

Exemplo: Trecho de programa em Java

```
int a, b, c;  
Scanner t = new Scanner(System.in);  
try {  
    a = t.nextInt();  
    b = t.nextInt();  
    c = a / b;  
    System.out.println(a + " / " + b + " = " + c);  
} catch(Exception e) {  
    System.out.println("Erro: " + e.getMessage());  
}
```

Que exceções podem acontecer?

- Usuário digitar um número inválido para a
- Usuário digitar um número inválido para b
- Usuário digitar ZERO para b

Ao tratar várias exceções ... Os tratadores devem estar ordenados das subclasses para a superclasse.

Bloco try...catch..finally

```
try {  
    // Código a ser tratado  
} catch(Exception e) {  
    System.out.println("Erro: " + e.getMessage());  
} finally {
```

```
// Esse código será sempre executado, independente
// se houve exceção ou não
}
```

Nota:

1. A variável e referência a exceção que ocorreu. Com ela é possível acessar informações sobre essa exceção, como por exemplo, com o uso do método `getMessage`.
2. **finally** não é obrigatório. Deve ser usado para instruções de "limpeza".

As exceções do Java são classificadas como checked ou unchecked.

Para as exceções checked, o Java nos obriga a:

1. Tratar a exceções no método onde ela pode ocorrer

OU

1. Avisar que estamos cientes de que aquela exceção pode ocorrer, mas não desejamos tratá-la.

-Para a opção 1 implementamos o bloco `try...catch` visto anteriormente.

-Para a opção 2 usamos o comando `throws`

Exemplo:

Se escrevermos um programa para ler e imprimir arquivo texto ocorrerá exceção do tipo **FileNotFoundException** ou **IOException**. Note que, neste caso, o Java não compila esse código ! Para conseguirmos compilar usamos `throws`.

```
public class ImprimeArquivo {
    public static void main(String[] args) throws Exception {
        FileReader fr = new FileReader("arquivo.txt");
        BufferedReader f = new BufferedReader(fr);
        String linha;
        linha = f.readLine();
        while (linha != null) {
            System.out.println(linha);
            linha = f.readLine();
        }
        f.close();
    }
}
```

```
}  
}
```

Note que usou-se throws e agora, o programa será compilado.

Como tratar situações de erro que são específicas dos nossos programas?

Exemplo:

Para construir um retângulo, que valores seriam inválidos?

```
public Retangulo(int x, int y, int largura, int altura)
```

Note: Largura e altura devem ser maiores que zero.

Como podemos tratar a situação onde Largura ou Altura é menor ou igual a zero?

Resposta: Criando e disparando exceções!

Veja como:

- Criamos uma instância da classe Exception com o operador new;
- Disparamos a exceção com o comando throw;
- Declaramos que o método irá disparar uma exceção com o comando throws.

A classe Exception é a classe mãe de todas as exceções que nos interessa. Ela possui alguns métodos úteis e comuns a todas as exceções:

- Construtor Exception(String msg): permite criar uma exceção e armazenar a mensagem de erro.
- getMessage(): retorna a mensagem de erro.
- printStackTrace(): imprime a pilha de chamadas no mesmo formato da JVM.
- getStackTrace(): retorna a pilha de chamadas. Nesse caso você pode implementar a sua própria impressão ou salvar essa informação em outro local ou formato (por exemplo, para montar um log de erros).

Exemplo:

Se depararmos com uma situação na qual o objeto não pode ser criado, disparamos uma exceção:

```
public class Retangulo {  
  
    private int x, y, largura, altura;  
  
    public Retangulo(int x, int y, int largura, int altura) throws Exception  
  
    {
```

```
if (largura <= 0 || altura <= 0)
    throw new Exception("Retangulo deve ter largura e altura maior que zero");

    this.x = x;
    this.y = y
    this.largura = largura;
    this.altura = altura;
}
}
```

Nota: Se **largura** ou **altura** forem menores ou iguais a zero a execução do construtor será interrompida e a exceção será disparada.

Classes de exceção que devem ser trabalhadas no curso: NullPointerException, InputMismatchException, NumberFormatException, ArrayIndexOutOfBoundsException, ArithmeticException.

Estratégias de Aprendizagem

Realizar exercícios práticos que envolvam a criação e análise de programas que implementem tratamento de exceções.

Indicação de Leitura Específica

Tratamento de Exceções

DEITEL, Paul. Java: como programar (Biblioteca Virtual). 10. ed. São Paulo: Pearson, 2017. [Páginas: 201 – 202]

Tratamento de Exceções em Java - <https://www.devmedia.com.br/tratando-excecoes-em-java/25514>

Aplicação: articulação teoria e prática

Exercício:

1) Escreva um programa em Java que leia números inteiros positivos e imprima o somatório desses números. O programa deve parar de ler quando o usuário digitar um

número menor ou igual a zero. Faça o tratamento de exceção para o caso do usuário não digitar um número quando solicitado.

a) 1ª. solução : ler valores com `nextInt()`

b) 2ª. solução : ler valores com `nextLine()`

2) Teste o programa e capture a exceção, adequadamente.

```
import java.util.*;

public class Teste {

    static String nome;

    static int idade;

    public static void main(String[] args) {

        System.out.println("\nIdade: " + idade);

        System.out.println("\nTamanho do nome : " + nome.length());

    } // fim main

} // fim classe
```

Considerações Adicionais