Estágio Curricular Obrigatório - 2018/2

Empresa: Terris Tecnologia

Marcelo Gervazoni Carbonera
marcelocarbonera@live.com

Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco

30 de Novembro de 2018

Objetivos da Aula

- Relembrar sobre o tratamento de exceções

Relembrando: Aula Passada

- Interface de software
- Programação Orientada a Objetos

Introdução

Toda execução de código acontece sempre da forma esperada?

Mundo Ideal

- Arquivos sempre existem
- Dados estão no formato correto
- Há produtos suficientes no estoque
- Divisão por zero

Mundo Real

- Arquivos nem sempre existem
- Dados estão em formato diferente
- Sem estoque
- Divisão por zero

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Introdução

Exceção

Situação anormal é denominada **exceção**.

Quando uma exceção é detectada?

- Em tempo de compilação
- Em tempo de execução
  - Essa que temos que tratar
- **Como tratar essa exceção?**
  - Um print na tela alertando o usuário?
    - E se a exceção for gerada remotamente (no servidor)?
  - Depois do print, o que fazer?
    - Continuar a execução do programa ou fechar?

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Exceções e Orientação a Objetos

Linguens OO dão suporte a exceções

- Classes de erros
- Possíveis tipos de erros e seus tratamentos são agrupados.
- Não é necessário interromper o programa.
- O mesmo erro pode ser tratado quantas vezes for necessário.

Premissa básica

- Separação da execução "normal" do "tratamento de erros"
- Simplifica a criação de longos trecho de código
- Mais fácil garantir que não há erros sem tratamento

Exceções e Orientação a Objetos

O que é preciso para tratar exceções?

- uma representação para a exceção
- uma forma de lançar a exceção
- uma forma de tratar a exceção

Em Java

- Para representar - **classes**
- Para lançar/disparar - comando **throw**
- Para tratar/capturar - estrutura **try-catch-finally**

Representação de Exceções em Java - Classe

Exceção deve herdar de **Exception**.

```
1   class MyException extends Exception {
2   private int i;
3   public MyException() {}
4
5   public MyException(String msg) {
6      super(msg);
7   }
8   public MyException(String msg, int x) {
9      super(msg);
10     i = x
11  }
12  public int val() {
13     return i;
14  }
15
16  // se quiser mudar a mensagem padrao
17  public String toString() {
18     return "Minha String Personalizada: " + this.getMessage();
19  }
20 }
```

Lançando/Disparando Exceções em Java

Utiliza-se **throws** para lançar uma exceção.

```
1   public class MainExcecoes {
2
3   public static void a() throws MyException {
4       System.out.println("Throwing MyException from a()");
5       throw new MyException();
6   }
7
8   public static void b() throws MyException {
9       System.out.println("Throwing MyException from b()");
10      throw new MyException("Originated in b()");
11  }
12
13  public static void c() throws MyException {
14      System.out.println("Throwing MyException from c()");
15      throw new MyException("Originated in c()", 21);
16  }
```

Tratando/Capturando Exceções em Java

- **Região protegida:** trecho de código que pode gerar exceções.
- **Manipuladores de exceções:** tratam as exceções que ocorreram dentro da região protegida, aparecem logo após a região protegida.
- Em Java: estrutura **try-catch-finally**.
  - **try:** indica a região protegida
  - **catch:** manipula uma exceção
  - **finally:** trecho de código que será sempre executado, não importa o que aconteça no **try-catch**

```
1      try {
2     // Codigo da regiao protegida
3      }
4      catch(ClasseDeExcecao e) {
5     // Manipula aquele tipo de erro
6      }
7      finally {
8     // Codigo a ser executado apos tratamento excecao
9      }
```

Tratando/Capturando Exceções em Java

```
11      public static void main(String[] args) {
12          try {
13              a();
14          } catch (MyException e) {
15              e.printStackTrace();
16          }
17
18          try {
19              b();
20          } catch (MyException e) {
21              e.printStackTrace();
22          }
23
24          try {
25              c();
26          } catch (MyException e) {
27              e.printStackTrace();
28              System.out.println("e.val() = " + e.val());
29          }
30      }
```

Exemplo do uso da clausula **finally**

Utiliza-se a clausula **finally** quando é necessário garantir que um código seja sempre executado, independete se o bloco **try-catch** seja executado corretamente ou não.

- **Ex:** ao abrir um arquivo, se na sua leitura der um erro o mesmo não deveria ficar aberto.

```
1  void readFile(String name) throws IOException {
2     FileReader file = null;
3     try {
4       file = new FileReader(name);
5       ... // le o arquivo
6     } catch (Exception e) {
7       System.out.println(e);
8     } finally {
9       if (file != null) file.close();
10    }
11 }
```

Exercícios - Prática 1

- Implementar os exercícios 3 e 4 da Prática 1, tratando as devidas exceções, conforme descrito na prática.