Orientação a Objetos
Exceções em Java

Prof. Lucas Boaventura

Tipos de Erro

- **Erros:**
  - Erros de sintaxe
  - Erros de lógica
  - Erros que dependem de lógica e/ou entidades externas ao programa não controlados pelo programador.
    - Ex: entrada de usuário, abrir arquivo, comunicação via rede, etc.

Tipos de Erro

- Erros de sintaxe podem ser avaliados em tempo de compilação
  - Tratamento => o compilador aponta erros.

- Erros de lógica e externos somente são detectados em tempo de execução (runtime error) -> Exceções

Exceções

- Uso de mecanismos de detecção de erros
  - Exceções: provém um meio de lidar com ocorrências não previstas transferindo o controle do programa para funções especiais (handlers).

- Separa o código que identifica erro do código que reporta o erro.
- Aumenta a confiabilidade da aplicação.

Exceções

```java
public class App {
    public static void main(String[] args) {
        f1();
    }
    static void f1() {
        try {
            // ...
            f2();
            // ...
        } catch (MinhaExcecao e) {
            // Código de tratamento do erro
            System.out.println("Exceção capturada em f1: " + e.getMessage());
        }
    }
    static void f2() {
        // ...
        f3();
        // ...
    }
```

```java
static void f3() {
        // ...
        f4();
        // ...
    }
    static void f4() {
        // ...
        boolean condicaoDeErro = true; // Simulação da condição de erro
        if (condicaoDeErro) {
            throw new MinhaExcecao("Algo deu errado em f4()");
        }
        // ...
    }
    // Definição da exceção personalizada
    static class MinhaExcecao extends RuntimeException {
        public MinhaExcecao(String mensagem) {
            super(mensagem);
        }
    }
}
```

Exceções - Try

- **Try**: Bloco de código sob inspeção. É possível ter blocos **try** dentro de outro bloco **try**.

```
try {
    // ...
    f2();
    // ...
} catch (MinhaExcecao e) {
    // Código de tratamento do erro
    System.out.println("Exceção capturada em f1: " + e.getMessage());
}
```

Exceções - Catch

- **Catch:** função de tratamento de exceções. Cada função catch lida com um tipo de exceção definida pelo argumento que recebe.

- É possível haver várias funções catch para cada bloco try. A função catch com argumento “..." trata qualquer tipo de exceção.

Exceções - Com Try/Catch

- Tratamento de Múltiplas Exceções:
  - Com Try, Catch

```
try {
    objeto.metodoQuePodeLancarIOeSQLException();
} catch (IOException e) {
    // ..
} catch (SQLException e) {
    // ..
}
```

Exceções - Throw

- **Throw**: originador de uma exceção.

  Usado dentro de funções onde se verifica o erro para transmitir as exceções às funções de tratamento (handlers) adequadas.

- Pode ser usado para propagar exceções.
- O argumento da função **throw** deve ser do mesmo tipo esperado pela função de tratamento da exceção (**catch**).

Exceções - Try, Catch e Throws

- Tratamento de Múltiplas Exceções:
  - Combinação de Try, Catch e Throws

```
public void abre(String arquivo) throws IOException {
  try {
    objeto.metodoQuePodeLancarIOeSQLException();
  } catch (SQLException e) {
     // ..
  }
}
```

Exceções - getMessage()

- O método getMessage() da Classe Exception

```
try {
    new java.io.FileInputStream("arquivo.txt");
}
catch (java.io.FileNotFoundException e) {
    System.out.println(e.getMessage());
}
```

Exceções - Esceções Customizadas

- Exceções Customizadas que herdam da classe Exception

```java
public class IdadeInvalidaException extends Exception {
    public IdadeInvalidaException(String mensagem) {
        super(mensagem);
    }
}
```

Exceções - Finally

- Finally

```
try {
    // bloco try
} catch (IOException ex) {
    // bloco catch 1
} catch (SQLException sqlex) {
    // bloco catch 2
} finally {
    // bloco que será sempre executado, independente
    // se houve ou não exception e se ela foi tratada ou não
}
```

UML - Agregação e Composição

Dúvidas?