
FIAP

Domain Driven Design
using Java

AGENDA

1. Tratamento de exceções em Java
2. Exercícios

Tratamento de exceções em Java

- O tratamento de exceções em Java permite capturar e gerenciar erros durante a execução do programa.
- Ajuda a evitar que o programa termine inesperadamente e fornece uma maneira controlada de lidar com situações excepcionais.
- A hierarquia de exceções em Java é organizada em uma árvore de classes, todas derivadas da classe base *Throwable*.

Hierarquia de Exceções

- ***Throwable***: A classe raiz de todas as exceções e erros.
  - Subclasses principais: ***Error e Exception***.

Hierarquia de Exceções

- **Error**: Representa problemas graves que normalmente não devem ser tratados pelo programa.
  - Exemplos: ***OutOfMemoryError***, ***StackOverflowError***, ***VirtualMachineError***.

Hierarquia de Exceções

- **Exception**: Representa condições que podem ser capturadas e tratadas pelo programa.
  - Subdivisões principais: *Checked Exceptions* e Unchecked Exceptions.

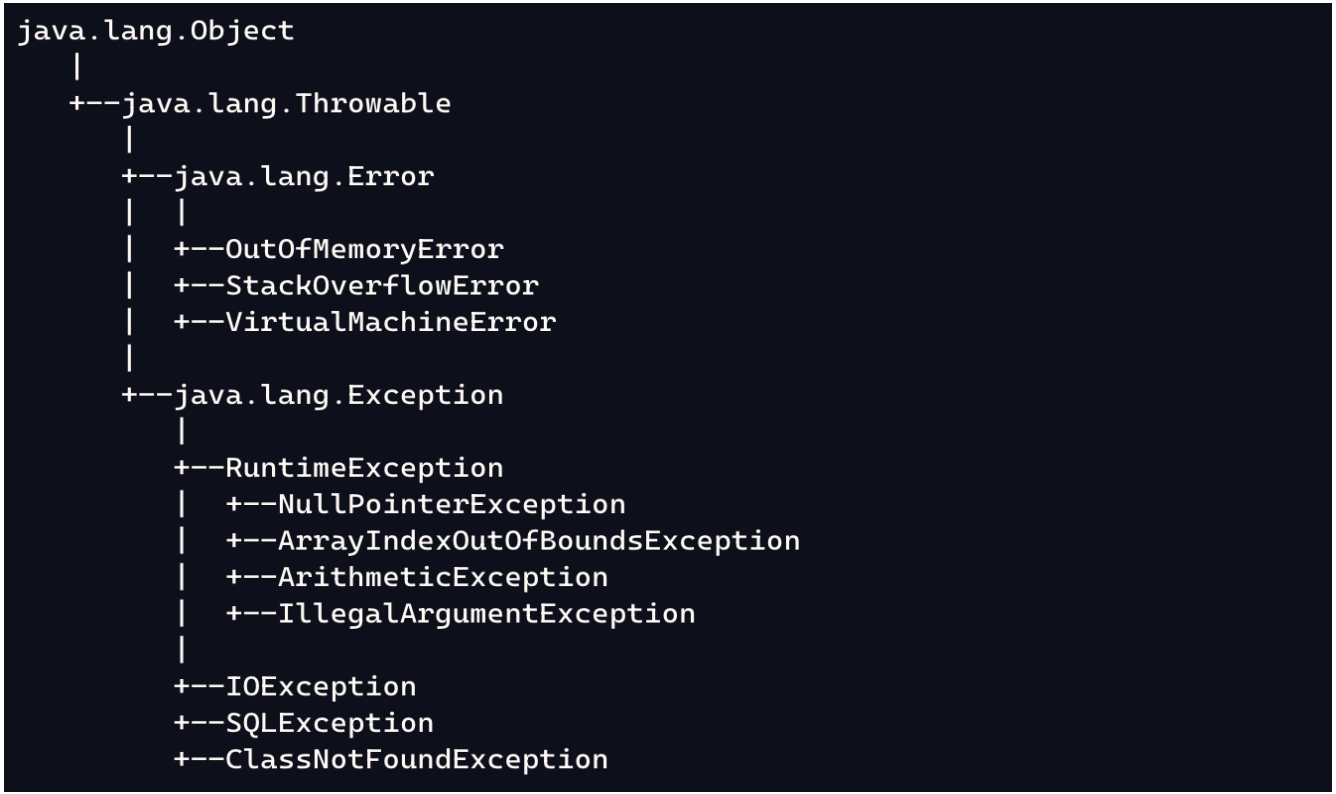
Checked Exceptions

- **Checked Exceptions**: Devem ser tratadas explicitamente usando blocos *try-catch* ou declaradas com a palavra-chave *throws*.
  - Exemplos: *IOException*, *SQLException*, *ClassNotFoundExceptionxemplos*.

Unchecked Exceptions

- **Unchecked Exceptions**: Incluem todas as subclasses de *RuntimeException*.
  - Exemplos: *NullPointerException*, *ArrayIndexOutOfBoundsException*, *ArithmeticException*, *IllegalArgumentExceptionxemplos*.

Diagrama Simplificado da Hierarquia



Estrutura Básica do Tratamento de Exceções

- **Bloco *try***: Contém o código que pode lançar uma exceção.

```
try {
    // Código que pode lançar uma exceção
    int result = 10 / 0;
}
```

Estrutura Básica do Tratamento de Exceções

- **Bloco *catch***: Captura a exceção lançada no bloco ***try***.

```
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Divisão por zero não é permitida.");
}
```

Estrutura Básica do Tratamento de Exceções

- **Bloco finally**: Contém código que será executado independentemente de uma exceção ter sido lançada ou não.

```
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Divisão por zero não é permitida.");
} finally {
    System.out.println("Este bloco sempre será executado.");
}
```

Estrutura Básica do Tratamento de Exceções

- **Palavra-chave throw**: Usada para lançar explicitamente uma exceção.

```
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Divisão por zero não é permitida.");
} finally {
    System.out.println("Este bloco sempre será executado.");
}
```

Estrutura Básica do Tratamento de Exceções

- **Palavra-chave throw**: Usada para lançar explicitamente uma exceção.

```
public void checkAge(int age) {
    if (age < 18) {
        throw new IllegalArgumentException("Idade deve ser maior
    }
}
```

Estrutura Básica do Tratamento de Exceções

- **Palavra-chave throws**: Usada no cabeçalho de um método para indicar que o método pode lançar uma ou mais exceções.

```
public void readFile(String fileName) throws IOException {
    FileReader file = new FileReader(fileName);
}
```

Criando Sua Própria Exceção

- Em Java, é possível criar suas próprias exceções personalizadas estendendo a classe *Exception* ou *RuntimeException*.
- Exceções personalizadas são úteis quando você precisa definir condições de erro específicas para o seu aplicativo.

Criando Sua Própria Exceção

- Exemplo de Exceção Personalizada:

```
// Classe de exceção personalizada estendendo Exception
public class MinhaExcecao extends Exception {
    // Construtor com uma mensagem de erro
    public MinhaExcecao(String mensagem) {
        super(mensagem);
    }
}
```

Usando a Exceção Personalizada

```java
public class TesteExcecao {
    public static void main(String[] args) {
        try {
            verificaIdade(15);
        } catch (MinhaExcecao e) {
            System.out.println("Exceção Capturada: " + e.getMessage());
        }
    }

    // Método que lança a exceção personalizada
    public static void verificaIdade(int idade) throws MinhaExcecao {
        if (idade < 18) {
            throw new MinhaExcecao("Idade deve ser maior ou igual a 18.");
        } else {
            System.out.println("Idade válida: " + idade);
        }
    }
}
```

Boas práticas

- **Especifique exceções**: Capture exceções específicas em vez de capturar Exception diretamente.
- **Nunca deixe o bloco catch vazio**: Sempre trate a exceção de alguma forma.
- **Libere recursos no finally**: Sempre feche recursos como arquivos ou conexões no bloco finally.
- **Evite lançar exceções desnecessárias:** Lance exceções apenas quando necessário e significativo para o fluxo do programa.

Exercício

1. Crie um projeto para cadastramento de pessoas e implemente algumas validações utilizando tratamento de exceção. Alguns exemplos: Idade não permitida; Endereço inválido; etc.


FIAP