Programação Orientada a Objetos

Prof.ª Ma. Jessica Oliveira

Católica
Universidade Católica de Brasília

Aula 07 – 07/04/2025

Tratamento de Exceções e Manipulação de Erros.

Católica
Universidade Católica de Brasília

O que são erros?

- Em Java (e em outras linguagens de programação), os erros representam **situações graves e inesperadas que normalmente não podem ser resolvidas pelo próprio programa**.
- Erros costumam ocorrer por falhas que não dependem diretamente do código que estamos escrevendo, como problemas de memória, falhas do sistema operacional ou erros na máquina virtual do Java (JVM).
- Esses erros fazem parte da classe **`Error`**, que é uma subclasse da classe **`Throwable`** (a classe raiz para todas as exceções e erros em Java).

O que são erros?

- **Resumo técnico:**
  - Não devem ser capturados ou tratados com try/catch;
  - Indicam falhas severas de ambiente;
  - São raros no desenvolvimento comum de software.

O que são exceções?

- As exceções são **situações anormais que ocorrem durante a execução de um programa**, mas que podem ser previstas e tratadas pelo programador. São eventos inesperados, mas possíveis, como:
  - Tentar dividir um número por zero;
  - Acessar uma posição inválida de um *array*;
  - Tentar abrir um arquivo que não existe.
- A diferença entre erro e exceção é justamente essa: **as exceções são recuperáveis**, ou seja, o programa pode continuar funcionando desde que o problema seja tratado corretamente.
- Em Java, as exceções também derivam da classe **`Throwable`**, mas pertencem à hierarquia da classe **`Exception`**.

Por que tratar exceções?

- Se um erro ocorrer e não for tratado, o programa será encerrado abruptamente, apresentando uma mensagem de erro ao usuário.
- Isso afeta a experiência de uso, pode causar perda de dados e prejudica a credibilidade da aplicação.
- Tratar exceções permite:
  - Identificar a causa do erro;
  - Evitar que o programa pare de funcionar;
  - Informar o usuário de forma clara;
  - Propor soluções alternativas dentro do sistema.

Hierarquia de Classes de Erro e Exceção.

Católica
Universidade Católica de Brasília

Throwable.

- É a superclasse de todos os erros e exceções.
- Possui dois principais "filhos":
    - **`Error`**: para erros graves de sistema.
    - **`Exception`**: para situações que podem ser tratadas.

Error.

- Usado pelo próprio Java para indicar problemas do ambiente ou da JVM.
- **Não deve ser tratado pelo código da aplicação.**

Exception.

- **São as exceções tratáveis.**
- Essa classe é dividida em dois grupos:
  - Exceções Verificadas (*Checked Exceptions*);
  - Exceções Não Verificadas (*Unchecked Exceptions*).

Checked Exceptions.

- São verificadas em tempo de compilação.
- O programador é obrigado a tratá-las com **`try/catch`** ou declarar com **`throws`**.
- Exemplos:
  - **`IOException`** – erro ao ler ou gravar arquivos;
  - **`SQLException`** – erro ao acessar o banco de dados.

Unchecked Exceptions.

- São verificadas apenas em tempo de execução.
- O tratamento não é obrigatório, mas recomendado.
- Herdam da classe **`RuntimeException`**.
- Exemplos:
  - **`ArithmeticException`** – divisão por zero;
  - **`NullPointerException`** – uso de variável não inicializada;
  - **`IndexOutOfBoundsException`** – acesso fora dos limites de um *array*.

Blocos de Tratamento: *try*, *catch* e *finally*.

Importante saber...

- Java possui uma estrutura especial para tratamento de exceções. Ela é formada por três blocos:

```
try {
    // código que pode lançar exceção
} catch (TipoDaExcecao e) {
    // tratamento da exceção
} finally {
    // código que será executado sempre, com ou sem erro
}
```

Bloco *try*.

- É onde colocamos o código que pode gerar uma exceção. Ele deve conter apenas instruções passíveis de falha. Exemplo:

```
try {
    int resultado = 10 / 0; // Lança ArithmeticException
}
```

Bloco *catch*.

- Captura e trata a exceção lançada no bloco **`try`**. Ele funciona como uma rede de segurança. Você pode personalizar a mensagem, registrar o erro, entre outros. Exemplo:

```
catch (ArithmeticException e) {
    System.out.println("Erro: divisão por zero!");
}
```

Bloco *catch*.

- Também é possível capturar múltiplos tipos de exceção com vários **`catch`**.

```
try {
    // código
} catch (NullPointerException e) {
    // trata nulo
} catch (ArrayIndexOutOfBoundsException e) {
    // trata acesso fora do array
}
```

Bloco *finally*.

- O bloco **`finally`** sempre será executado, mesmo que não ocorra nenhuma exceção ou que uma exceção tenha sido capturada.
- É ideal para fechar arquivos, liberar conexões com banco de dados, ou outras tarefas de limpeza.

```
finally {
    System.out.println("Finalizando execução.");
}
```

Lançando exceções: *throw* e *throws*.

Palavra-chave *throw*.

- Usada para lançar uma exceção no meio do código.
- Exemplo:

```
if (idade < 0) {
    throw new IllegalArgumentException("Idade não pode ser negativa.");
}
```

- A exceção pode ser uma das já existentes em Java (como **`IllegalArgumentException`**) ou uma classe personalizada criada por você.

Palavra-chave *throws*.

- Usada na assinatura de um método ou construtor para indicar que ele pode lançar uma exceção.
- Exemplo:

```
public Livro(String titulo, String autor, int paginas)
throws IllegalArgumentException {
    if (paginas <= 0) {
        throw new IllegalArgumentException("Número de
páginas inválido.");
    }
}
```

Palavra-chave *throws*.

- Regra: se o método lançar uma exceção verificada (**`checked`**), ele deve declarar **`throws`**. Se for uma exceção não verificada, o uso de **`throws`** é opcional, mas recomendável para clareza.

Boas práticas no tratamento de exceções.

- Nunca deixe o **`catch`** vazio, pois ele ignora o erro, o que dificulta identificar falhas.
- Seja específico ao capturar exceções: evite capturar **`Exception`** ou **`Throwable`** diretamente, a menos que tenha uma boa razão. Sempre que possível, capture o tipo exato de exceção que você espera.
- Não use exceções para controlar fluxo normal do programa.
- A mensagem de uma exceção deve ser clara e específica, pois isso facilita o diagnóstico do erro.
- Evite excesso de **`try/catch`** espalhado, criando métodos auxiliares com validações, isso melhora a legibilidade do código e separa responsabilidades.

Microprojeto 07

🎯 **Objetivo:** implementar tratamento de erros para garantir robustez ao sistema.

Para o pós-aula...

- Implementar blocos **`try-catch-finally`** para tratamento de erros.
- Criar exceções personalizadas para cenários específicos do projeto.
- Testar a resiliência do código ao lidar com entradas inválidas.
- Documentar como as exceções foram tratadas.

Dúvidas?

jessica.oliveira@p.ucb.br