Índice

[Tratamento de Erros em Java 1](#_ikggjc9t25fo)

[Links e afins 1](#_v49ndfn8hwjl)

[Dúvidas 1](#_huow9eiz5of6)

[Temporários 2](#_xipq7wjk3kuo)

# 

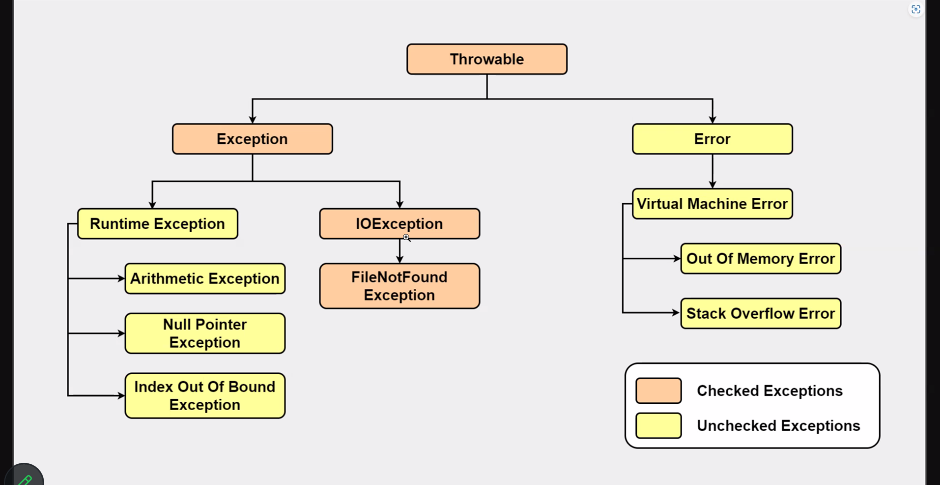
**Assuntos: Tratamento de Erros em Java: Exceções, Mulheres na Tecnologia.**

**Abertura Alunos:**

Reflexão diária: Como lidar com ansiedade de falar em público

Flávia Ribeiro e Isabelli Pereira.

# Tratamento de Erros em Java



Classe Exceptions:

* **Throwable:** É a Super Classe (Classe mãe) de todos os Erros e Exceções.
* **Error:** Não são exceções, e sim erros que jamais poderiam ter acontecido, como estouro da memória (Out Of Memory Error), por exemplo.
* **Exception:** São as classes que deveriam aqui, lançar exceções e não erros de programação. Exemplo: tentar abrir um arquivo que não existe.
* **RuntimeException:** São exceções que indicam erros de programas (não de lógica, pois senão não passaria pelo compilador).
* **IOException:** São exceções que indicam erros de entrada/saída por falha ou interrupção. Exemplo: interrupção ou falha na leitura de um arquivo em disco.

Exceções do tipo checked:

* **ClassNotFoundException:** esta exceção é gerada quando tentamos acessar uma classe cuja definição não foi encontrada.

Exemplo: na IDE, durante o desenvolvimento do código, ao tentar importar uma Classe que não existe, esta Exceção será disparada.

* **FileNotFoundException:** esta exceção é gerada quando um arquivo não está acessível ou não abre.

Exemplo: ao tentar abrir um arquivo externo que não existe, esta Exceção será disparada.

* **NoSuchFieldException:** esta exceção é lançada quando uma classe não contém o Atributo ou variável especificada.

Exemplo: ao tentar acessar um Atributo que não existe na Classe, esta Exceção será disparada.

* **NoSuchMethodException:** esta exceção é lançada ao acessar um método que não foi encontrado.

Exemplo: ao tentar acessar um Método que não existe na Classe, esta Exceção será disparada.

Exceções do tipo unchecked:

* **ArithmeticException:** esta exceção é lançada quando ocorre uma condição excepcional em uma operação aritmética.

Exemplo: se em uma divisão, o dividendo (segundo parâmetro) for igual a zero, esta Exceção será disparada.

* **ArrayIndexOutOfBoundsException:** esta exceção é lançada para indicar que um array foi acessado com um índice ilegal.

Exemplo: se o índice passado for negativo ou maior ou igual ao tamanho da matriz, esta Exceção será disparada.

* **NullPointerException:** esta exceção é gerada ao se referir aos membros de um objeto nulo. Nulo não representa nada. Para evitar esta exceção é muito comum utilizar a Classe Optional.

Exemplo: ao executar um Método que não seja do tipo void, se ele retornar um Objeto nulo, esta Exceção será disparada.

* **NumberFormatException:** esta exceção é gerada quando um método não pode converter uma String em um formato numérico (int, float, double, entre outros).

Exemplo: ao tentar converter uma String, que contém apenas letras em um número, esta Exceção será disparada.

* **IllegalArgumentException:** esta exceção geralmente é utilizada para validar valores de parâmetros que não estão nas condições adequadas ao programa.

Exemplo: se um parâmetro do seu método do tipo inteiro, nunca poderá ser maior do que 100, para evitar falhas no sistema, você pode fazer uso desta exceção.

* **IllegalStateException:** esta exceção lançará um erro ou mensagem de erro quando tentarmos invocar um Método específico em um momento inapropriado.

Exemplo: ao adicionar um elemento em uma Estrutura de dados do tipo Fila, devemos garantir que a fila não esteja cheia, caso contrário esta Exceção será disparada.

É necessário verificar se os erros são parte da máquina, IDE, e afins, ou se se tratam de erros de código e estão dentro do nosso escopo de mudança.

As exceções são coisas tratáveis pelo programador.

Runtime Exception costumam ser erros de sintaxe.

## Comandos

**try { } catch ( ) { } finally { }** Comando para monitorar o código, e tratá-lo sem quebrar o projeto, ele gera um aviso para o usuário.

**Exemplo:**

**try {*código*}**

**catch (tipo*Erro nomeErro*) { *comando a ser executado ao erro ser encontrado*}**

**finally {*comando a ser executado sempre*}**

O try especifica o código a ser monitorado.

O catch trata o erro.

O finally SEMPRE é executado, muito usado para limpar o cache, ou para simplesmente evitar a quebra de código.

Pode-se criar um catch para cada erro, ou usar termos lógicos como || ou && para tratar os erros.

O ‘nomeErro’ é um nome a ser dado para a variável desse erro, para que ele possa ser citado posteriormente.

**System.err.println( )** Comando para printar na tela um erro, geralmente com uma formatação mais adequada, como coloração vermelha e afins.

**throw new *nomeErro* (*mensagem de erro*)** Comando onde você cria uma exceção e a insere um comando a ser executado caso este erro em específico seja gerado.

O erro aparece assim: 

Perceba que depois do nome do erro, ele dá a mensagem desejada.

# 

# 

# 

# Links e afins

**Cookbook PR05, Projeto Conta Bancária, Exceptions:**

<https://github.com/conteudoGeneration/cookbook_java_fullstack/blob/main/01_java/pr05.md>

**Cookbook PR06, Projeto Conta Bancária, Classes Abstratas:**

<https://github.com/conteudoGeneration/cookbook_java_fullstack/blob/main/01_java/pr06.md>

**Cookbook PR07, Projeto Conta Bancária, Interface:**

<https://github.com/conteudoGeneration/cookbook_java_fullstack/blob/main/01_java/pr07.md>

# 

# Dúvidas

# Temporários