# Objetivo do curso

Ao final deste curso, o Dev será capaz de identificar possíveis exceções de uma aplicação Java e interpretar eventuais pilhas de exceção. Assim, você estará preparado para capturar e tratar essas exceções, deixando suas soluções ainda mais robustas.

**Pe**

1

2.

3.

4.

[

Nome do palestrante

]

[

Posição

]

[

Nome do curso

]

**[**

**Nome da aula**

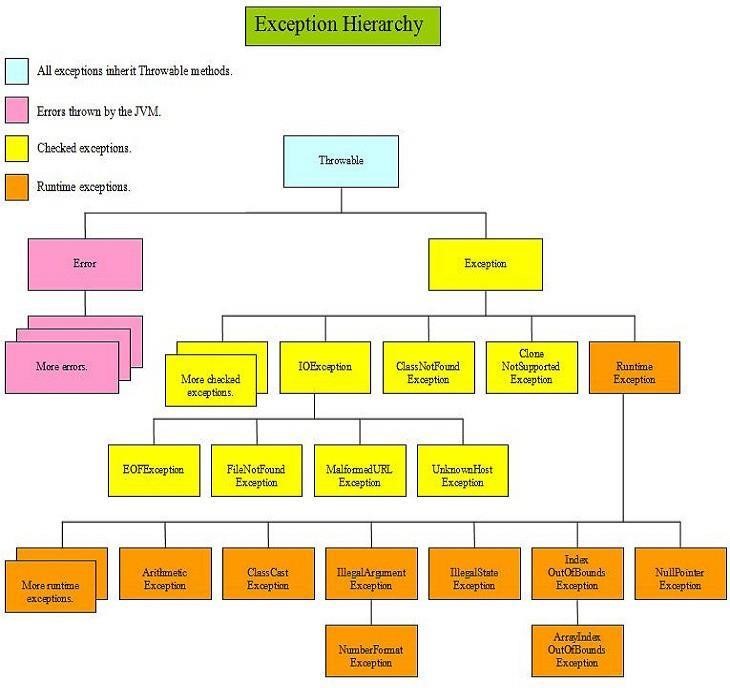
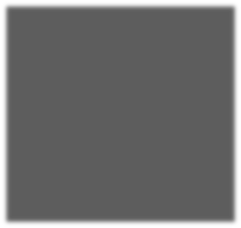
**]**

**Visão Geral**

Tratamento de Exceções em Java

* Exceção é um evento que interrompe o fluxo normal do processamento de uma classe.
* O uso correto de exceções torna o programa mais robusto e confiável.
* Com o tratamento de exceções, um programa pode continuar executando depois de lidar com um problema.
* Importante: Incorpore sua estratégia de tratamento de exceções no sistema desde o princípio do processo do projeto.
* Pode ser difícil incluir um tratamento de exceções eficiente depois que um sistema foi implementado.
* **Error**: Usado pela JVM que serve para indicar se existe algum problema de recurso do programa, tornando a execução impossível de continuar.
* **Unchecked (Runtime)**: Exceptions que PODEM ser evitados se forem tratados e analisados pelo desenvolvedor.
* **Checked Exception**: Exceptions que DEVEM ser evitados e tratados pelo desenvolvedor para o programa funcionar.

**Tratamento de Exceções**



Condições excepcionais

checadas pelo compilador

(

eventos recuperáveis

)



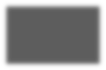
Erros não checados

pelo compilador

(

eventos irrecuperáveis

)



Exceções não

checadas pelos

compiladores

(

erros de lógica

)

**Tratamento de Exceções**

●

**try,**

**catch,**

**finally**

:

Cada

uma

dessas

palavras,

juntas,

definem

blocos

para

o

tratamento

de

exceções

.

●

**throws**

:

Declara

que

um

método

pode

lançar

uma

ou

várias

exceções

.

●

**throw**

:

Lança

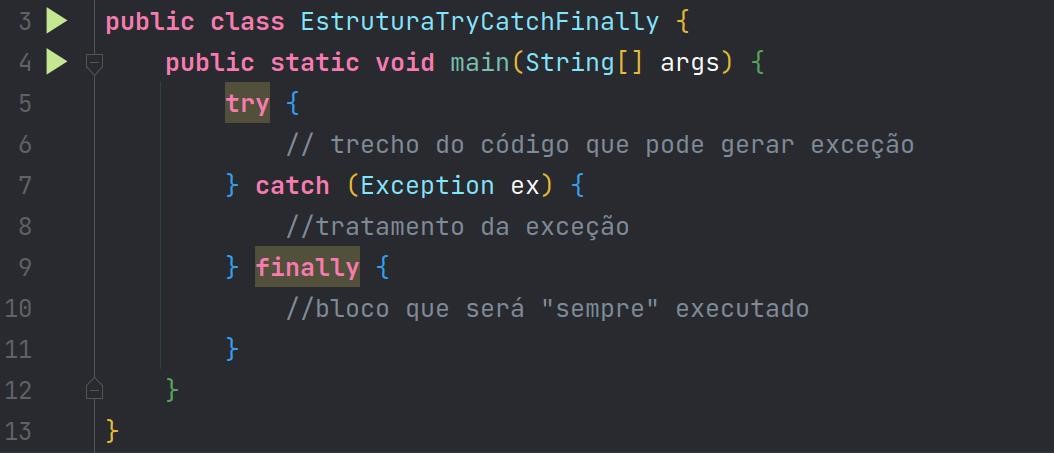
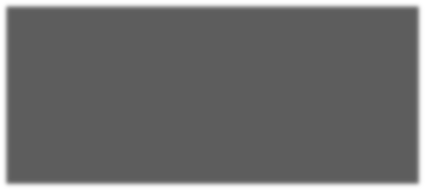
explicitamente

uma

exception

.

**Tratamento de Exceções**



Bloco ***try***:

* Região onde se encontra o código que queremos verificar se irá ou não lançar uma exceção.
* Caso ocorra uma exceção em algum ponto, o restante do código contido no bloco *try* não será executado.
* O bloco *try* não pode ser declarado sozinho, por tanto, precisa estar seguido de um ou vários blocos catch e/ou de um bloco finally.

Bloco ***catch***:

* Região onde se encontra o possível tratamento da exceção. Isso significa que só será executado caso o bloco try apresentar alguma exceção.
* Recebe como argumento a classe ou subclasse da possível exceção.
* No seu escopo ficam as instruções de como tratar essa exceção.
* Pode haver mais de um bloco catch, porém, será executado apenas o primeiro bloco que identificar a exceção.

Bloco

***finally***

:

●

Este

bloco

é

opcional,

mas

caso

seja

construído,

quase

sempre

será

executado

.

(

A

menos

que

seja

forçado,

por

exemplo,

com

um

System

.

exit(

0

)

,

no

catch)

.

●

Dentro

do

bloco

finally,

poderá

conter

outros

blocos

try,

catch,

bem

como

outro

finally

.

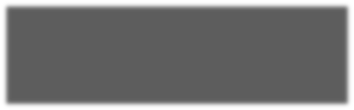
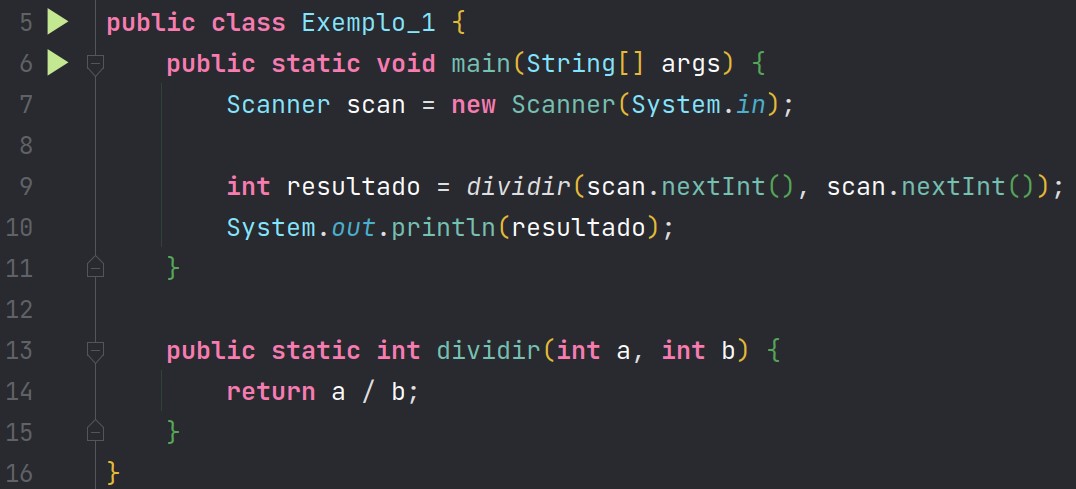
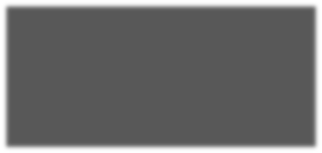
Cláusula ***throw***:

* Para lançar exceções explicitamente, use a cláusula *throw*.
* Usada principalmente para lançar **exceções personalizadas**.
* Caso um tratador adequado não seja encontrado no bloco onde a exceção foi lançada, ela é propagada para o nível mais externo.
* A propagação contínua até que algum tratador seja encontrado ou até chegar ao nível daJVM.
* Pode ser utilizada tanto para exceções *checked* ou *unchecked.*

Cláusula ***throws***:

* O *throws* quando declarado no método, servirá apenas para informar ao compilador que estamos cientes da possibilidade de apresentar alguma Exception neste método.
* Para que um método possa disparar uma exceção é necessário colocar a cláusula throws na definição dele, indicando quais tipos de exceção ele pode retornar.
* A responsabilidade de tratar o método lançado fica no código que chamou o método, podendo tratá-la ou lançá-la novamente.

**Tratamento de Exceções**



Errors

e

RuntimeExceptions

são

considerados

*uncheked*

,

portanto

o

compilador

não

obriga

que

exista

tratamento

para

eles

.

**Unchecked**

**Exception**

### Unchecked Exception

* Herdam da classe **RuntimeException** ou da classe **Error**.
* O compilador não verifica o código para ver se a exceção foi capturada ou declarada.
* Se uma exceção não-verificada ocorrer e não tiver sido capturada, o programa terminará ou executará com resultados inesperados.
* Em geral, podem ser evitadas com uma codificação adequada.

**Tratamento de Exceções**

**Checked Exception**

Costumam

indicar

que

uma

condição

necessária

para

a

execução

de

um

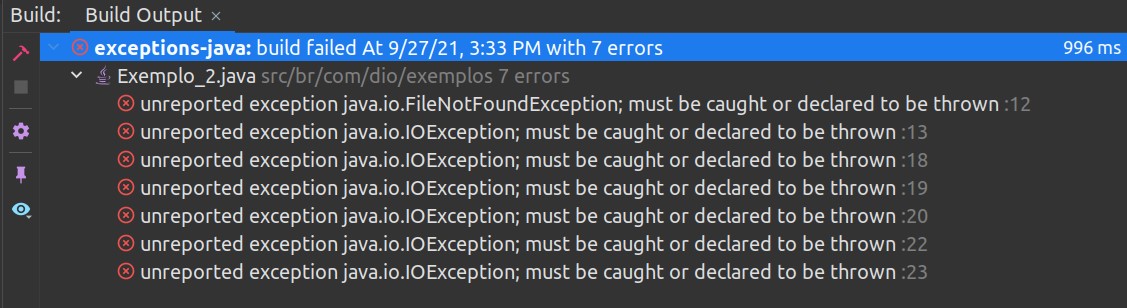
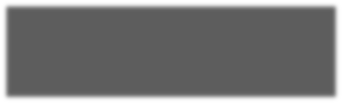
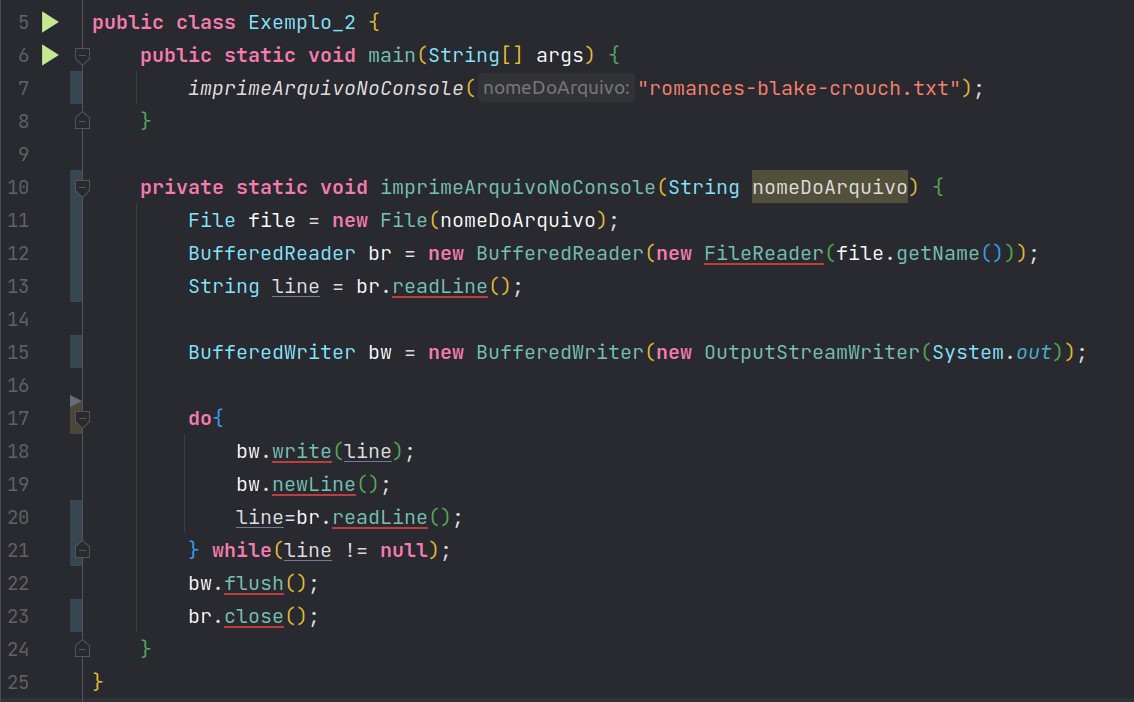
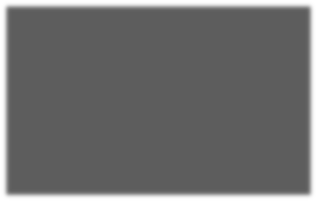
programa

não

está

presente

.



### Checked Exception

* As exceções que são herdadas da classe Exception, mas não de RuntimeException.
* O compilador impõe um requisito do tipo ‘capturar ou declarar’.
* O compilador verifica cada chamada de método e declaração de método para determinar se o método lança (*throws*) exceções verificadas. Se lançar, o compilador assegura que a exceção verificada é capturada ou declarada em uma cláusula *throws*. Caso não capturada nem declarada, ocorre um erro de compilação.

**Tratamento de Exceções**

**Checked**

**Customizada**

Assim

como

qualquer

objeto,

em

Java

também

é

possível

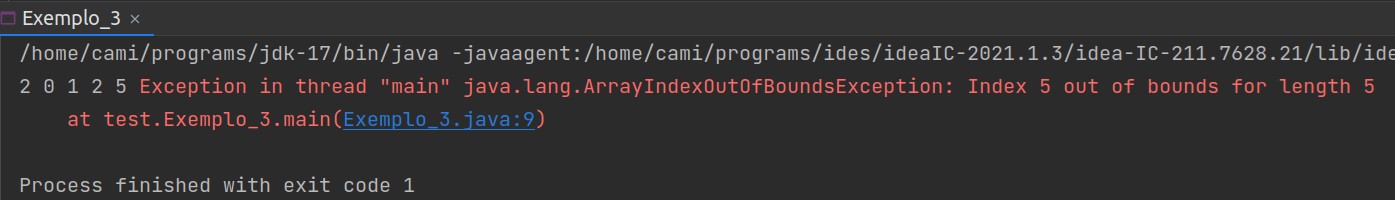
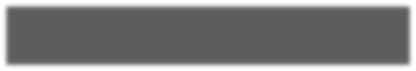
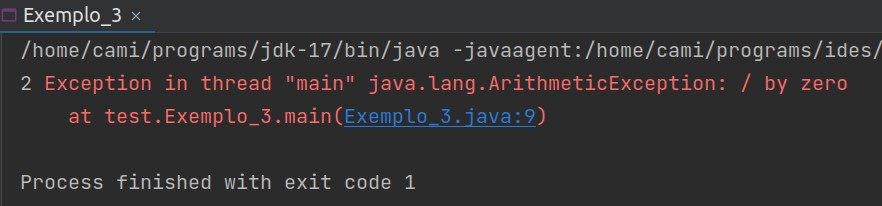
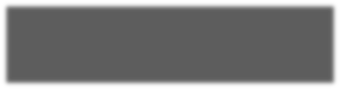
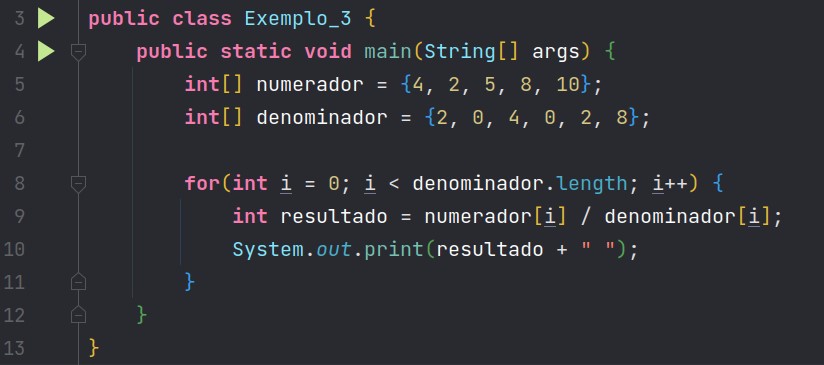
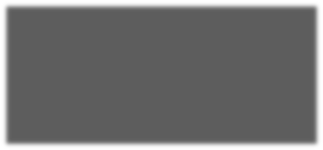
criar

suas

próprias

exceções

.



**Exception Personalizada:**

* Programadores podem achar útil declarar suas próprias classes de exceção.
* Essas Exceptions são específicas aos problemas que podem ocorrer quando outro programador empregar suas classes reutilizáveis.
* Uma nova classe de exceção deve estender uma classe de exceção existente que assegura que a classe pode ser utilizada com o mecanismo de tratamento de exceções, logo essas Exceções customizadas são derivadas da classe Exception.