



DIGITAL  
INNOVATION  
ONE

# Tratamento de Exceções em Java

---

Camila Cavalcante  
Tech Teacher



# Objetivo do curso

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

# Pré-requisitos

1. Java JDK 8 ou superior
2. IDE para desenvolvimento Java
3. Conhecimento BÁSICO em OOP
4. Estar disposto a aprender

# Percurso

1. Visão Geral
2. Unchecked Exception
3. Checked Exception
4. Exception Personalizada



# Mais sobre mim

- Estudante de Ciência da Computação
- Comecei no mundo da TI através do excel
- Minha motivação é a vontade de aprender coisas novas
- Nas horas vagas gosto de assistir séries e ler livros

# Redes Sociais

- <https://github.com/cami-la/exceptions-java>
- <https://www.linkedin.com/in/cami-la/>
- <https://www.instagram.com/estudent.i/>
- [https://www.instagram.com/camimi\\_la/](https://www.instagram.com/camimi_la/)

# Dúvidas durante o curso?

- > Fórum do curso
- > Comunidade [online \(discord\)](#)

# Visão Geral

## Tratamento de Exceções em Java

---

Camila Cavalcante  
Tech Teacher





# Tratamento de Exceções

- **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.

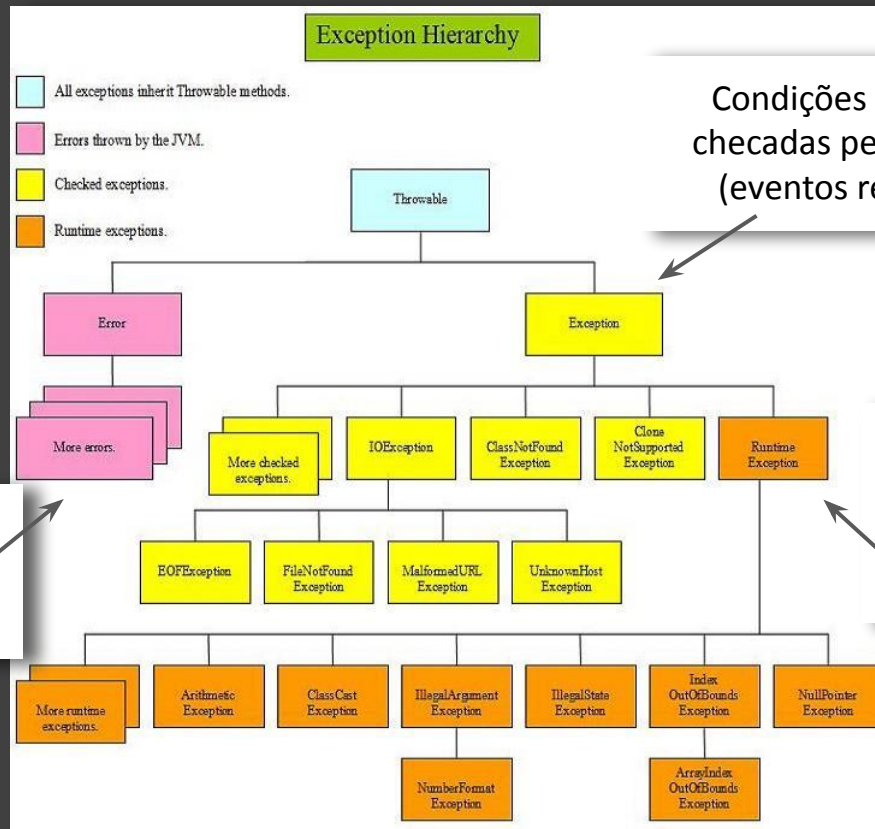


# Tratamento de Exceções

- **Error**: Usado pela JVM que serve para indicar se existe algum problema de recurso do programa, tornando a execução impossível de continuar. *programa para.*
- **Unchecked (Runtime)**: Exceptions que PODEM ser evitados se forem tratados e analisados pelo desenvolvedor. *erro de lógica.*
- **Checked Exception**: Exceptions que DEVEM ser evitados e tratados pelo desenvolvedor para o programa funcionar. *Obrigado a tratar. Se não roda o programa.*



# Tratamento de Exceções



Condições excepcionais  
cheçadas pelo compilador  
(eventos recuperáveis)

Erros não checados  
pelo compilador  
(eventos irrecuperáveis)

Exceções não  
cheçadas pelo  
compilador  
(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.

# Redes Sociais

- <https://github.com/cami-la/exceptions-java>
- <https://www.linkedin.com/in/cami-la/>
- <https://www.instagram.com/estudent.i/>
- [https://www.instagram.com/camimi\\_la/](https://www.instagram.com/camimi_la/)

# Dúvidas durante o curso?

- > Fórum do curso
- > Comunidade [online \(discord\)](#)



DIGITAL  
INNOVATION  
ONE

# try - catch - finally

## Tratamento de Exceções em Java

---

Camila Cavalcante  
Tech Teacher



# Tratamento de Exceções

```
3 ▶ public class EstruturaTryCatchFinally {  
4 ▶   public static void main(String[] args) {  
5       try {  
6           // trecho do código que pode gerar exceção  
7       } catch (Exception ex) {  
8           //tratamento da exceção  
9       } finally {  
10          //bloco que será "sempre" executado  
11      }  
12  }  
13 }
```





# 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**.



# Tratamento de Exceções

## 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.



# Tratamento de Exceções

## 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`.

# Redes Sociais

- <https://github.com/cami-la/exceptions-java>
- <https://www.linkedin.com/in/cami-la/>
- <https://www.instagram.com/estudent.i/>
- [https://www.instagram.com/camimi\\_la/](https://www.instagram.com/camimi_la/)

# Dúvidas durante o curso?

- > Fórum do curso
- > Comunidade [online \(discord\)](#)



DIGITAL  
INNOVATION  
ONE

# throw e throws

## Tratamento de Exceções em Java

---

Camila Cavalcante  
Tech Teacher



# Tratamento de Exceções

## 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 da JVM.
- Pode ser utilizada tanto para exceções *checked* ou *unchecked*.



# Tratamento de Exceções

## 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 do mesmo, indicando quais tipos de exceção o mesmo pode retornar.
- A responsabilidade de tratar o método lançado fica no código que chamou o método, podendo tratá-la ou lançá-la novamente.





DIGITAL  
INNOVATION  
ONE

# Unchecked Exception e Checked Exception

## Tratamento de Exceções em Java

---

Camila Cavalcante  
Tech Teacher



# Tratamento de Exceções

```
5 ▶ public class Exemplo_1 {  
6 ▶     public static void main(String[] args) {  
7         Scanner scan = new Scanner(System.in);  
8  
9         int resultado = dividir(scan.nextInt(), scan.nextInt());  
10        System.out.println(resultado);  
11    }  
12  
13    public static int dividir(int a, int b) {  
14        return a / b;  
15    }  
16 }
```

## Unchecked Exception

Errors e RuntimeExceptions são considerados *unchecked*, portanto o compilador não obriga que exista tratamento para eles.

Exemplo\_1 x

```
/home/cami/programs/jdk-17/bin/java -javaagent:/home/cami/programs/ides/ideaIC-2021.
```

```
4
```

```
0
```

```
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero  
    at br.com.dio.Exemplo_1.dividir(Exemplo_1.java:14)  
    at br.com.dio.Exemplo_1.main(Exemplo_1.java:9)
```

```
Process finished with exit code 1
```



# Tratamento de Exceções

## 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.

```
5 public class Exemplo_2 {
6     public static void main(String[] args) {
7         imprimeArquivoNoConsole(nomeDoArquivo: "romances-blake-crouch.txt");
8     }
9
10    private static void imprimeArquivoNoConsole(String nomeDoArquivo) {
11        File file = new File(nomeDoArquivo);
12        BufferedReader br = new BufferedReader(new FileReader(file.getName()));
13        String line = br.readLine();
14
15        BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
16
17        do{
18            bw.write(line);
19            bw.newLine();
20            line=br.readLine();
21        } while(line != null);
22        bw.flush();
23        br.close();
24    }
25 }
```

Build: Build Output x

```
exceptions-java: build failed At 9/27/21, 3:33 PM with 7 errors 996 ms
  Exemplo_2.java src/br/com/dio/exemplos 7 errors
    ✖ unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown :12
    ✖ unreported exception java.io.IOException; must be caught or declared to be thrown :13
    ✖ unreported exception java.io.IOException; must be caught or declared to be thrown :18
    ✖ unreported exception java.io.IOException; must be caught or declared to be thrown :19
    ✖ unreported exception java.io.IOException; must be caught or declared to be thrown :20
    ✖ unreported exception java.io.IOException; must be caught or declared to be thrown :22
    ✖ unreported exception java.io.IOException; must be caught or declared to be thrown :23
```



# Tratamento de Exceções

## 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.

# Exception Personalizada

## Tratamento de Exceções em Java

---

Camila Cavalcante  
Tech Teacher



# Tratamento de Exceções

## Checked Customizada

Assim como qualquer objeto, em Java também é possível criar suas próprias exceções.

```
3 public class Exemplo_3 {  
4     public static void main(String[] args) {  
5         int[] numerador = {4, 2, 5, 8, 10};  
6         int[] denominador = {2, 0, 4, 0, 2, 8};  
7  
8         for(int i = 0; i < denominador.length; i++) {  
9             int resultado = numerador[i] / denominador[i];  
10            System.out.print(resultado + " ");  
11        }  
12    }  
13 }
```

Exemplo\_3 x

```
/home/cami/programs/jdk-17/bin/java -javaagent:/home/cami/programs/ides/  
2 Exception in thread "main" java.lang.ArithmeticException: / by zero  
   at test.Exemplo_3.main(Exemplo_3.java:9)
```

Exemplo\_3 x

```
/home/cami/programs/jdk-17/bin/java -javaagent:/home/cami/programs/ides/ideaIC-2021.1.3/idea-IC-211.7628.21/lib/ide  
2 0 1 2 5 Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5  
   at test.Exemplo_3.main(Exemplo_3.java:9)
```

Process finished with exit code 1

# Tratamento de 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.



# Redes Sociais

- <https://github.com/cami-la/exceptions-java>
- <https://www.linkedin.com/in/cami-la/>
- <https://www.instagram.com/estudent.i/>
- [https://www.instagram.com/camimi\\_la/](https://www.instagram.com/camimi_la/)

# Dúvidas durante o curso?

- > Fórum do curso
- > Comunidade [online \(discord\)](#)