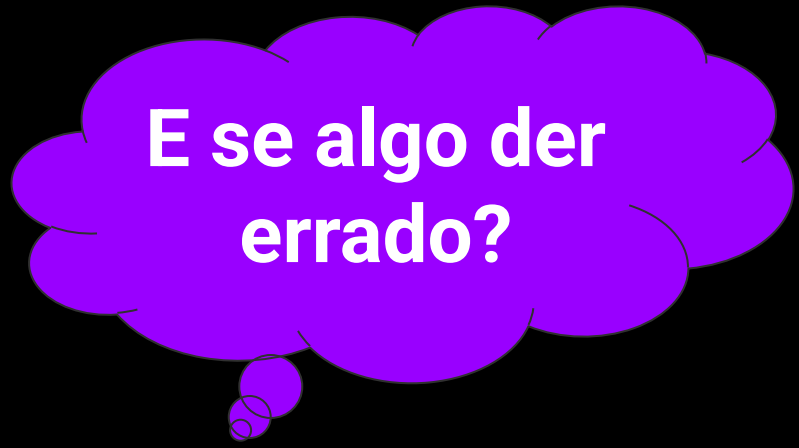
# Tratamento de exceções

Matheus Barbosa
matheus.barbosa@dcx.ufpb.br

## O que é?

Tratamento de Exceções é um mecanismo para lidar com erros de tempo de execução

**Por que usar?**




Geralmente escrevemos código em um ambiente idealizado:
o sistema de arquivos sempre contém nossos arquivos, a rede está saudável e a JVM sempre tem memória suficiente. Às vezes, chamamos isso de "caminho feliz".

```java
public static void lerArquivo() throws IOException {
    List<String> linhas = Files.readAllLines(Paths.get("dados.txt"));

    for (String linha : linhas) {
        System.out.println(linha);
    }
}
```

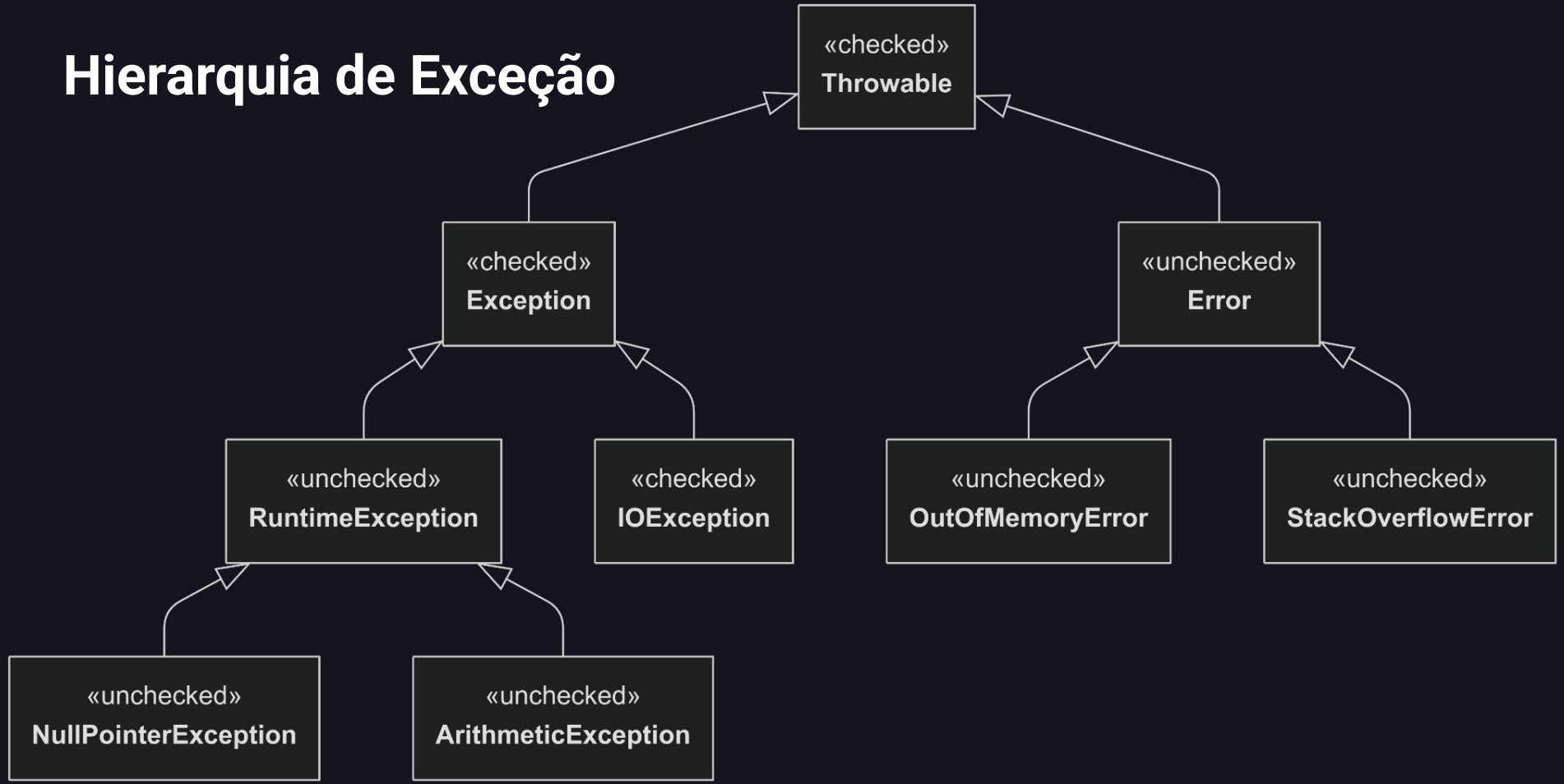
```
java.nio.file.NoSuchFileException: dados.txt
```

**Matheus Barbosa**
matheus.barbosa@dcx.ufpb.br

## Vantagens de usar Exceções

- Separar o código de tratamento de erros do código "normal"
- Propagação de erros na pilha de chamadas
- Agrupamento e diferenciação de tipos de erros

# Hierarquia de Exceção



## Principais categorias de condições excepcionais

- Exceções verificadas
- Exceções não verificadas
- Erros

# Exceções Verificadas (*checked*)

São previsíveis e o compilador obriga a tratar.

- ➔ Tentar pegar um ônibus e o motorista pode não parar. Você sabe que isso pode acontecer, então tem um plano B (Uber, outro ônibus, etc.).

- ➔ Você tenta abrir um arquivo que pode não existir. (`IOException`)

# Exceções Não Verificadas (*unchecked*)

São exceções que o compilador Java não exige que tratemos.

- Você anda de bicicleta e de repente o pneu estoura. Você não previu isso... É inesperado e acontece durante a execução.
- Divisão por zero: (`ArithmeticException`)
- Acessar posição inválida em uma lista: (`IndexOutOfBoundsException`)
- Usar uma variável nula: (`NullPointerException`)

# Error – Problemas Graves do Sistema

Não devem ser tratados pelo programa. São erros do próprio ambiente Java.

- Você tenta trabalhar, mas acaba a energia elétrica da cidade. Isso não é um problema que seu programa pode resolver. É um erro estrutural.
- Quando o sistema fica sem memória: (`OutOfMemoryError`)
- Quando métodos se chamam recursivamente sem parar: (`StackOverflowError`)

| Tipo | Verificado? | Deve tratar? | Exemplo real | Exemplo Java |
|---|---|---|---|---|
| Exception | ✅ Sim | ✅ Sim | Arquivo não encontrado | `IOException`, `SQLException` |
| Runtime Exception | ❌ Não | ❌ Opcional | Divisão por zero, valor nulo | `NullPointerException`, `ArithmeticException` |
| Error | ❌ Não | ❌ Não | Falta de memória, pane no sistema | `OutOfMemoryError`, `StackOverflowError` |

Java fornece uma maneira estruturada de lidar com exceções usando cinco palavras-chave principais: try, catch, finally, throw e throws.

## *throw*

Se não quisermos manipular a exceção nós mesmos, precisamos nos familiarizar com a palavra-chave throw

É usado para lançar uma exceção no meio do código.

```java
public int obterPontuacaoDoJogador(String arquivoDoJogador) {
    if (arquivoDoJogador == null || arquivoDoJogador.isEmpty()) {
        throw new IllegalArgumentException(
            "O nome do arquivo não pode ser nulo ou vazio."
        );
    }

    Scanner conteudo = new Scanner(new File(arquivoDoJogador));
    return Integer.parseInt(conteudo.nextLine());
}
```

## *throws*

É usado na assinatura do método para indicar que ele pode lançar uma exceção verificada (checked).

Como `FileNotFoundException` é uma exceção verificada, esta é a maneira mais simples de satisfazer o compilador, mas isso significa que qualquer um que chamar nosso método agora precisa tratá-lo também!

```
public int obterPontuacaoDoJogador(String arquivoDoJogador)
  throws FileNotFoundException {

    Scanner conteudo = new Scanner(new File(arquivoDoJogador));
    return Integer.parseInt(conteudo.nextLine());
}
```

## *try-catch*

Se quisermos tentar tratar a exceção nós mesmos, podemos usar um bloco `try-catch`

```java
public int obterPontuacaoDoJogador(String arquivoDoJogador) {
    try {
        Scanner conteudo = new Scanner(new File(arquivoDoJogador));
        return Integer.parseInt(conteudo.nextLine());
    } catch (FileNotFoundException arquivoNaoEncontrado) {
        throw new IllegalArgumentException("Arquivo não encontrado");
    }
}
```

```java
public int obterPontuacaoDoJogador(String arquivoDoJogador) {
    try {
        Scanner conteudo = new Scanner(new File(arquivoDoJogador));
        return Integer.parseInt(conteudo.nextLine());
    } catch (FileNotFoundException arquivoNaoEncontrado) {
        logger.warn("Arquivo não encontrado, reiniciando pontuação.");
        return 0;
    }
}
```

## *finally*

Há momentos em que temos código que precisa ser executado independentemente de ocorrer uma exceção, e é aí que  entra a palavra-chave finally.

```
public int obterPontuacaoDoJogador(String arquivoDoJogador)
  throws FileNotFoundException {
    Scanner conteudo = null;
    try {
        conteudo = new Scanner(new File(arquivoDoJogador));
        return Integer.parseInt(conteudo.nextLine());
    } finally {
        if (conteudo != null) {
            conteudo.close();
        }
    }
}
```

# *Blocos de captura múltipla*

Às vezes, o código pode lançar mais de uma exceção, e podemos ter mais de um bloco catch  tratando cada um individualmente

```
public int obterPontuacaoDoJogador(String arquivoDoJogador) {
    try (Scanner conteudo = new Scanner(new File(arquivoDoJogador))) {
        return Integer.parseInt(conteudo.nextLine());
    } catch (IOException e) {
        logger.warn("Não foi possível carregar o arquivo do jogador!", e);
        return 0;
    } catch (NumberFormatException e) {
        logger.warn("O arquivo do jogador está corrompido!", e);
        return 0;
    }
}
```

## *Blocos de captura da União*

Porém, quando sabemos que a maneira como lidamos com erros será a mesma, podemos capturar várias exceções no mesmo bloco:

```java
public int obterPontuacaoDoJogador(String arquivoDoJogador) {
    try (Scanner conteudo = new Scanner(new File(arquivoDoJogador))) {
        return Integer.parseInt(conteudo.nextLine());
    } catch (IOException | NumberFormatException e) {
        logger.warn("Falha ao carregar a pontuação!", e);
        return 0;
    }
}
```