

INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
PARAÍBA

# **Exceções e Controle de Erros**

## ***Programação Orientado a Objetos***

---

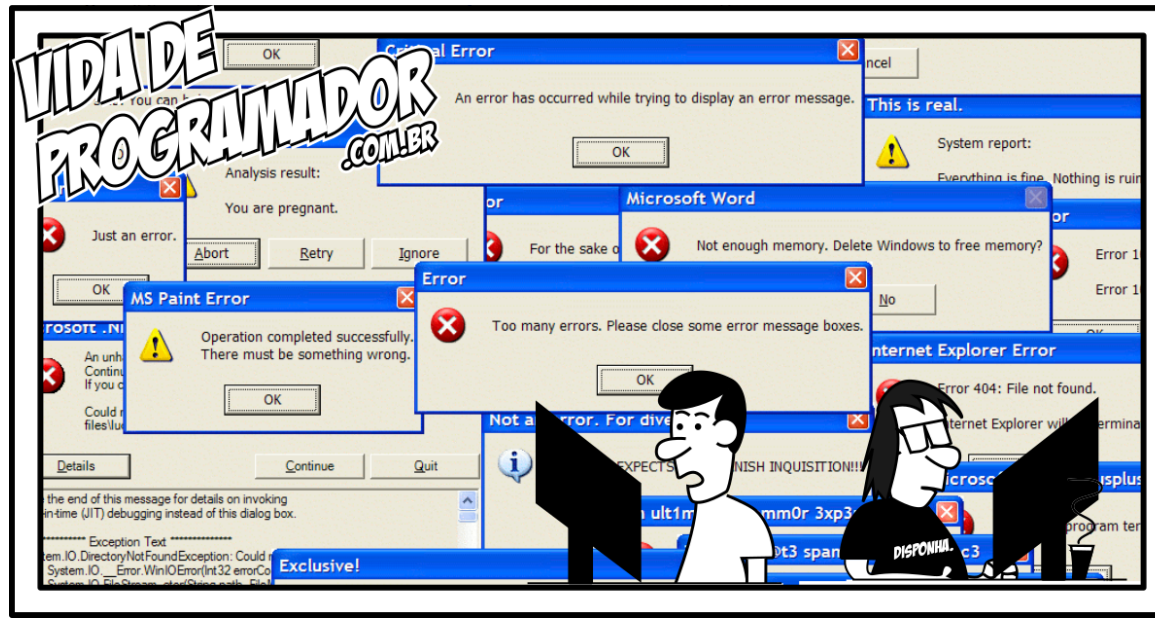
**Profs. Danielle Chaves e Francisco Dantas**

*{cmedanielle, franciscodnn}@gmail.com*

*Campina Grande/PB – 2017*

# O que estudaremos nesta aula?

O que é uma exceção? Como garantir que um método irá cumprir com o que ele se propôs a fazer? Como tratar erros em Java?



# O que é uma exceção?

---

- Uma exceção representa uma **situação que normalmente não ocorre;**
- Ocorrência de **algo de estranho ou inesperado no sistema,** oriundo de algum erro de lógica ou de acesso a uma operação ou um recurso que não esteja disponível.

# O que causa uma exceção?

---

- Uma exceção pode ser lançada quando:
  - Há um **acesso a um arquivo ou a uma URL inexistente**;
  - É realizada uma **divisão por zero**;
  - Se quer escrever em um **arquivo sem permissão de escrita**;
  - Ocorre um acesso a um **índice que está fora dos limites de um vetor**;
  - Há um **acesso a um banco de dados que encontra-se indisponível**;
  - É realizada uma operação utilizando uma **referência nula**;
  - ...

# O que causa uma exceção?

- Exemplo:

## NullPointerException

```
String s = null;  
s.isEmpty();
```

```
Exception in thread "main" java.lang.NullPointerException  
    at Teste.main(Teste.java:5)
```

**Acesso a uma referência nula.**

# Exceções em Java

---

- Em Java, quando ocorre algum problema na execução de um programa, é “lançada uma exceção” (*throw*) e, caso a JVM não encontre uma forma de tratar a exceção, o programa é encerrado;
- Toda vez que uma exceção é lançada, é criado um objeto da classe **Exception**, que descreve o erro ocorrido.

# Exceções em Java

- Vamos testar:

```
class TesteErro {  
    public static void main(String[] args) {  
        System.out.println("inicio do main");  
        metodo1();  
        System.out.println("fim do main");  
    }  
  
    static void metodo1() {  
        System.out.println("inicio do metodo1");  
        metodo2();  
        System.out.println("fim do metodo1");  
    }  
}
```

```
static void metodo2() {  
    System.out.println("inicio do metodo2");  
    int[] array = new int[10];  
    for (int i = 0; i <= 15; i++) {  
        array[i] = i;  
        System.out.println(i);  
    }  
    System.out.println("fim do metodo2");  
}
```

# Exceções em Java

- Vamos testar:

## Problema?

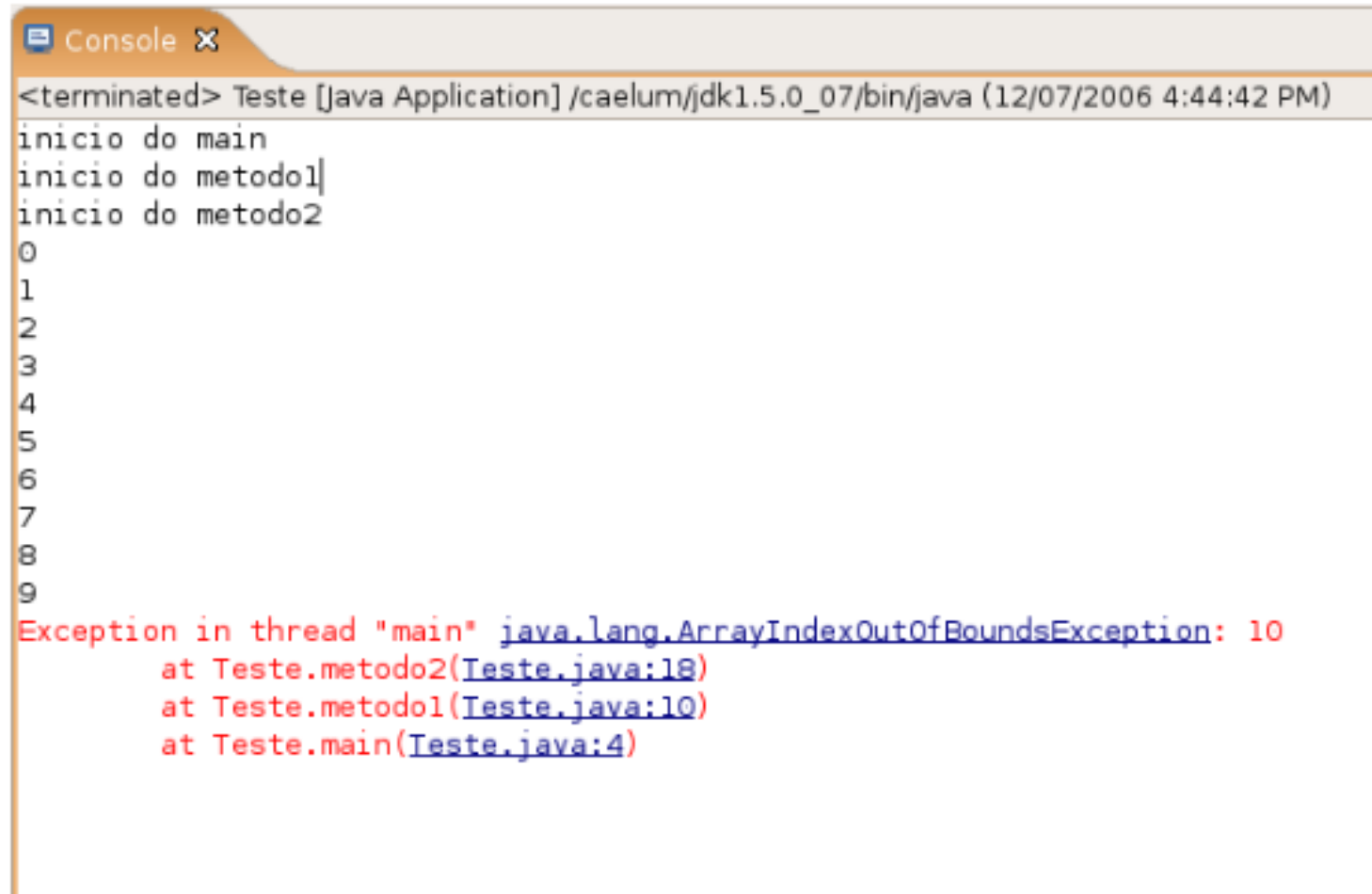
**metodo2** está acessando um índice de array indevido: o **índice** estará fora dos limites da array quando chegar em **10!**

```
static void metodo2() {  
    System.out.println("inicio do metodo2");  
    int[] array = new int[10];  
    for (int i = 0; i <= 15; i++) {  
        array[i] = i;  
        System.out.println(i);  
    }  
    System.out.println("fim do metodo2");  
}
```



# Exceções em Java

- Se executarmos o código...

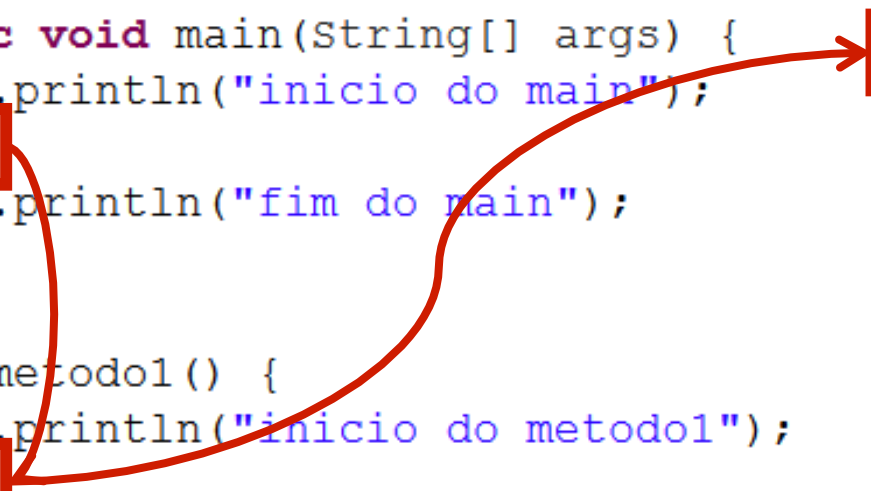


```
Console X
<terminated> Teste [Java Application] /caelum/jdk1.5.0_07/bin/java (12/07/2006 4:44:42 PM)
inicio do main
inicio do metodo1
inicio do metodo2
0
1
2
3
4
5
6
7
8
9
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
    at Teste.metodo2(Teste.java:18)
    at Teste.metodo1(Teste.java:10)
    at Teste.main(Teste.java:4)
```

# Exceções em Java

- Uma observação Importante:

```
class TesteErro {  
    public static void main(String[] args) {  
        System.out.println("inicio do main");  
        metodo1();  
        System.out.println("fim do main");  
    }  
  
    static void metodo1() {  
        System.out.println("inicio do metodo1");  
        metodo2();  
        System.out.println("fim do metodo1");  
    }  
  
    static void metodo2() {  
        System.out.println("inicio do metodo2");  
        int[] array = new int[10];  
        for (int i = 0; i <= 15; i++) {  
            array[i] = i;  
            System.out.println(i);  
        }  
        System.out.println("fim do metodo2");  
    }  
}
```



The diagram illustrates a recursive call between two static methods. A red box highlights the `metodo1()` call inside the `main` method. Another red box highlights the `metodo2()` call inside the `metodo1` method. A red arrow points from the `metodo2()` call in `metodo1` back to the `metodo1()` call in `main`, indicating that `metodo2` calls `metodo1`, which in turn calls `metodo2`, creating a recursive loop.

# Exceções em Java

- Uma observação importante...

Em Java, toda invocação de método é empilhada em uma estrutura de dados em áreas de memória isoladas.

Quando um método termina sua execução, ele volta para o método que o invocou. Isso ocorre por meio da pilha de execução (*stack*)



# Exceções em Java

- Uma observação importante...



```
Console X
<terminated> Teste [Java Application] /caelum/jdk1.5.0_07/bin
inicio do main
inicio do metodo1
inicio do metodo2
0
1
2
3
4
5
6
7
8
9
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
    at Teste.metodo2(Teste.java:18)
    at Teste.metodo1(Teste.java:10)
    at Teste.main(Teste.java:4)
```

Essa saída é conhecida  
como rastro da pilha  
(*stacktrace*)

# E como tratar exceções em Java?

- É possível tentar (*try*) executar um trecho de código perigoso e, caso o problema gere determinado tipo de erro, ele será pego (*catch*).

**É importante lembrar que cada exceção no Java tem um tipo, que pode ter atributos e métodos.**

# E como tratar exceções em Java?

- Exemplos:

```
try {  
    for (int i = 0; i <= 15; i++) {  
        array[i] = i;  
        System.out.println(i);  
    }  
} catch (ArrayIndexOutOfBoundsException e) {  
    System.out.println("erro: " + e);  
}
```

```
for (int i = 0; i <= 15; i++) {  
    try {  
        array[i] = i;  
        System.out.println(i);  
    } catch (ArrayIndexOutOfBoundsException e) {  
        System.out.println("erro: " + e);  
    }  
}
```

**Qual a diferença nestas duas formas de tratamento do erro?**

# E como tratar exceções em Java?

- Exemplos:

```
Console X
<terminated> Teste (1) [Java Application] /caelum/jdk1.5.0_07/bin/j
inicio do main
inicio do metodo1
inicio do metodo2
0
1
2
3
4
5
6
7
8
9
erro: java.lang.ArrayIndexOutOfBoundsException: 10
fim do metodo2
fim do metodo1
fim do main
```

```
Console X
<terminated> Teste (1) [Java Application] /caelum/jdk1.5.0_07/bin
inicio do main
inicio do metodo1
inicio do metodo2
0
1
2
3
4
5
6
7
8
9
erro: java.lang.ArrayIndexOutOfBoundsException: 10
erro: java.lang.ArrayIndexOutOfBoundsException: 11
erro: java.lang.ArrayIndexOutOfBoundsException: 12
erro: java.lang.ArrayIndexOutOfBoundsException: 13
erro: java.lang.ArrayIndexOutOfBoundsException: 14
erro: java.lang.ArrayIndexOutOfBoundsException: 15
fim do metodo2
fim do metodo1
fim do main
```

# E como tratar exceções em Java?

- Exemplos:

Agora tente colocar o *try/catch* em volta da chamada do método2 e depois do método1.

O que muda?



# Exceções em Java

- *Checked Exceptions*

- Exceções que **devem ser tratadas obrigatoriamente**;
- O compilador sempre irá checar se a exceção está sendo devidamente tratada;
- Utilizadas para **erros recuperáveis**, quando é possível prever a ocorrência de determinado erro.

Quando o tratamento de uma exceção é opcional, a chamamos de ***unchecked exception (runtime)***.

# Exceções em Java

---

- Exemplo:
  - Abrir um arquivo para leitura

```
class Teste {  
    public static void metodo() {  
        new java.io.FileInputStream("arquivo.txt");  
    }  
}
```

# Exceções em Java

- Tratando *checked exceptions*:
  - Utilizando try/catch

```
public static void metodo() {  
  
    try {  
        new java.io.FileInputStream("arquivo.txt");  
    } catch (java.io.FileNotFoundException e) {  
        System.out.println("Nao foi possível abrir o arquivo para  
leitura");  
    }  
  
}
```

# Exceções em Java

- Tratando *checked exceptions*:
  - Delegando a responsabilidade para quem chamar o método

```
public static void metodo() throws java.io.FileNotFoundException {  
    new java.io.FileInputStream("arquivo.txt");  
}
```

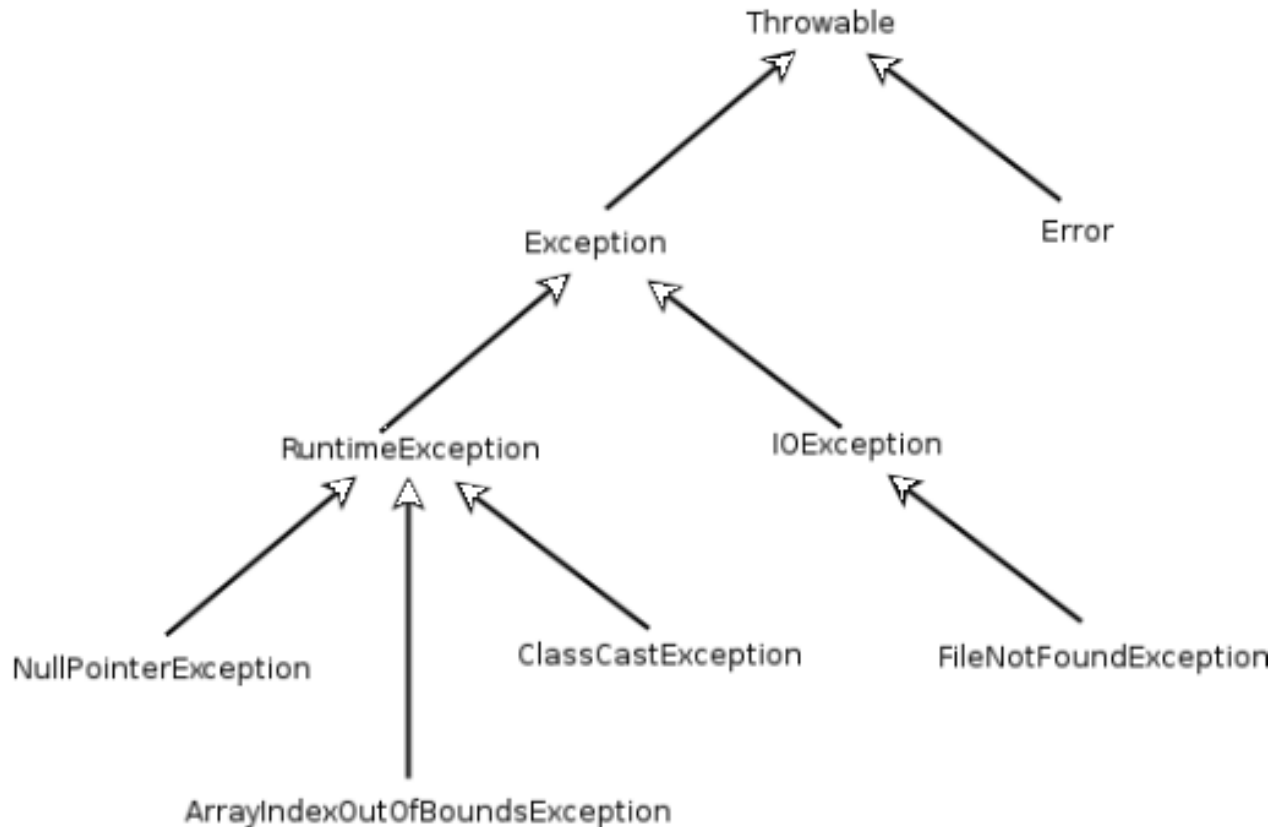
# Exceções em Java

- Tratando *checked exceptions*:
  - Delegando a responsabilidade para quem chamar o método

Ao delegar a responsabilidade, é importante lembrar que quem chama um método que está lançando uma exceção pode não saber tratar o erro!

# A Classe *Throwable*

- Classe mãe de todos os erros e exceções em Java.



# Algumas Observações Importantes...

- É possível tratar mais de um erro quase que ao mesmo tempo:

```
try {
    objeto.metodoQuePodeLancarIOeSQLException();
} catch (IOException e) {
    // ..
} catch (SQLException e) {
    // ..
}

public void abre(String arquivo) throws IOException, SQLException {
    // ..
}
```

# Algumas Observações Importantes...

- É possível, também, escolher tratar algumas exceções e declarar as outras no *throws*:

```
public void abre(String arquivo) throws IOException {  
    try {  
        objeto.metodoQuePodeLancarIOeSQLException();  
    } catch (SQLException e) {  
        // ..  
    }  
}
```



# Algumas Observações Importantes...

- É desnecessário declarar no *throws* as *exceptions* que são *unchecked*, porém é permitido e às vezes, facilita a leitura e a documentação do seu código;
- Há uma diferença entre as palavras *throws* e *throw*:

```
public void abre(String arquivo) throws IOException {  
    try {  
        objeto.metodoQuePodeLancarIOeSQLException();  
    } catch (SQLException e) {  
        // ..  
    }  
}
```

```
void saca(double valor) {  
    if (this.saldo < valor) {  
        throw new RuntimeException();  
    } else {  
        this.saldo-=valor;  
    }  
}
```

# Algumas Observações Importantes...

- Informando o motivo do lançamento de uma exceção:

```
Conta cc = new ContaCorrente();  
cc.deposita(100);
```

```
try {  
    cc.saca(100);  
} catch (IllegalArgumentException e) {  
    System.out.println(e.getMessage());  
}
```

```
void saca(double valor) {  
    if (this.saldo < valor) {  
        throw new IllegalArgumentException("Saldo insuficiente");  
    } else {  
        this.saldo -= valor;  
    }  
}
```

# Algumas Observações Importantes...

- É possível criar seu próprio tipo de exceção:

```
public class SaldoInsuficienteException extends RuntimeException {  
  
    SaldoInsuficienteException(String message) {  
        super(message);  
    }  
}
```

Exceção do tipo *unchecked*,  
**SaldoInsuficienteException**, para operações na  
classe Conta

# Algumas Observações Importantes...

- É possível criar seu próprio tipo de exceção:

```
public class SaldoInsuficienteException extends Exception {  
  
    SaldoInsuficienteException(String message) {  
        super(message);  
    }  
}
```

Exceção do tipo *checked*,  
**SaldoInsuficienteException**, para operações na  
classe Conta

# Algumas Observações Importantes...

- É possível criar seu próprio tipo de exceção:

```
void saca(double valor) {  
    if (this.saldo < valor) {  
        throw new SaldoInsuficienteException("Saldo Insuficiente," +  
            "tente um valor menor");  
    } else {  
        this.saldo -= valor;  
    }  
}
```

```
public static void main(String[] args) {  
    Conta cc = new ContaCorrente();  
    cc.deposita(10);  
  
    try {  
        cc.saca(100);  
    } catch (SaldoInsuficienteException e) {  
        System.out.println(e.getMessage());  
    }  
}
```

# Algumas Observações Importantes...

- E *finally*...

- Indica o que deve ser feito após o término do bloco *try* ou de um *catch* qualquer.

```
try {  
    // bloco try  
} catch (IOException ex) {  
    // bloco catch 1  
} catch (SQLException sqlex) {  
    // bloco catch 2  
} finally {  
    // bloco que será sempre executado, independente  
    // se houve ou não exception e se ela foi tratada ou não  
}
```

# Algumas Observações Importantes...

- Quanto ao *finally*:
  - *Seu trecho de código será sempre executado após tratada a exceção:*
    - *Útil para fechar arquivos, fechar conexões (por exemplo, com banco de dados), registrar ocorrências (logs), etc.*
  - *É possível haver apenas os blocos `try {...} finally{...}`, sem o `catch{...}`;*
  - *Quando usar `try {...} finally{...}` ?*

# Algumas Observações Importantes...

- Quanto ao *finally*:
  - Quando usar *try {...} finally{...}* ?

```
String mensagem = "Texto a ser salvo";
String arquivo = ".\arquivo.txt";
try {
    OutraClasse.salvarTextoNoArquivo(mensagem, arquivo);
    // desvio para outra classe, que deverá tratar exceção para a existência do arquivo
} finally {
    // Porém, sempre devem ser armazenados no log o que foi salvo e em que local
    ArquivoLog.salvar("Foi salva a mensagem "+mensagem+" no arquivo "+arquivo);
}
```



# Dicas de Leitura & Referências Utilizadas

---

- [Apostila Java e Orientação a Objetos – Capítulo 11](#)  
[Exceções e Controle de Erros](#)
- [Javadoc da Classe Throwable](#)
- [Tratando exceções em Java](#) (por Robson Fernando)