




# CURSO DE PROGRAMAÇÃO EM JAVA

Aula 15   
Tratamento de  
exceções

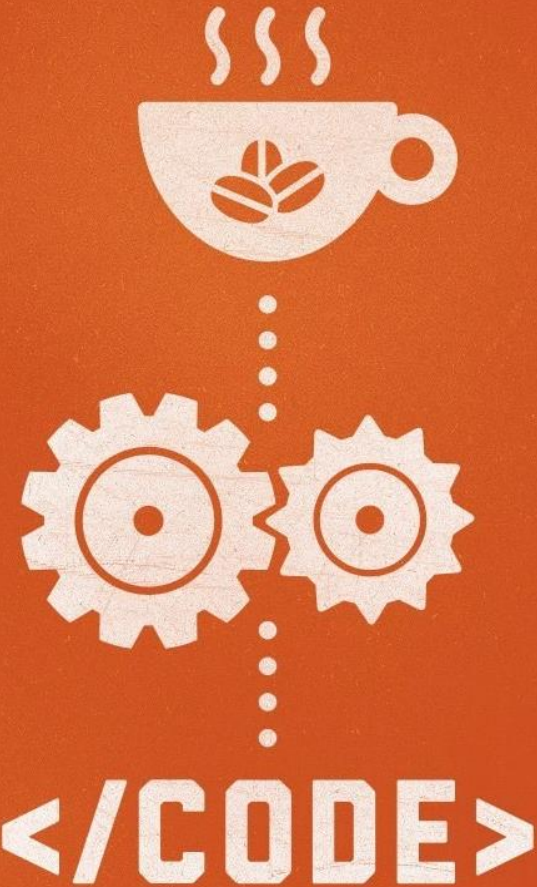
1.

# Tratamento de exceções

## Tratamento de exceções

Uma **exceção** é uma indicação de um **problema incomum** que ocorreu durante a **execução de um programa**. Com o **tratamento de exceções** é possível continuar a execução do programa (sem encerrá-lo) depois de lidar com o problema. Programas **robustos** e **tolerante a falhas** são aqueles que podem lidar com problemas à medida que surgem e continuar executando.

# PROGRAMMER



# Tratamento de exceções

- Exceções são:
  - Problemas em tempo de execução
  - Objetos criados a partir de classes especiais que são “lançados” quando ocorrem condições excepcionais
- Métodos podem capturar ou deixar passar exceções que ocorrerem em seu corpo
  - É obrigatório, para a maior parte das exceções, que o método declare quaisquer exceções que ele não capturar
- Mecanismo try-catch é usado para tentar capturar exceções enquanto elas passam por métodos

# Tratamento de exceções

- 1** Problemas de lógica de programação
  - Ex: limites do vetor ultrapassados, divisão por zero
  - Devem ser corrigidos pelo programador
- 2** Problemas devido a condições do ambiente de execução
  - Ex: arquivo não encontrado, rede fora do ar, etc.
  - Fogem do controle do programador mas podem ser contornados em tempo de execução
- 3** Problemas graves onde não adianta tentar recuperação
  - Ex: falta de memória, erro interno da JVM
  - Fogem do controle do programador e não podem ser contornados

# Tratamento de exceções

- Uma exceção é um tipo de objeto que sinaliza que uma condição excepcional ocorreu
  - A identificação (nome da classe) é sua parte mais importante
- Precisa ser instanciada com **new** e depois lançada com **throw**

```
IllegalArgumentException e =  
    new IllegalArgumentException("Erro!");  
throw e; // exceção foi lançada!
```

- A referência é desnecessária. A sintaxe abaixo é mais usual:

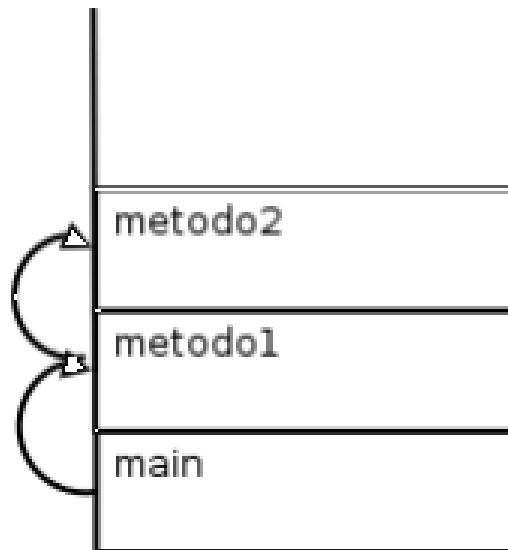
```
throw new IllegalArgumentException("Erro!");
```

# Exemplo

```
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");  
    }  
}
```

# Exemplo

Toda invocação de método é empilhada em uma estrutura de dados que isola a área de memória de cada um. Quando um método termina (retorna), ele volta para o método que o invocou. Ele descobre isso através da **pilha de execução** (*stack*)

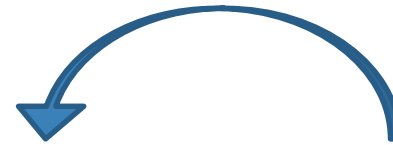




# Exemplo

Porém, o nosso metodo2 propositadamente possui um enorme problema: está acessando um índice de array indevido para esse caso; o índice estará fora dos limites da array quando chegar em 10!

```
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)
```



Essa é o conhecido  
**rastro da pilha** (*stacktrace*).

# Tratamento de exceções

- Uma exceção lançada interrompe o fluxo normal do programa:
  - O fluxo do programa segue a exceção
  - Se o método onde ela ocorrer não a capturar, ela será propagada para o método que chamar esse método e assim por diante
  - Se ninguém capturar a exceção, ela irá causar o término da aplicação
  - Se em algum lugar ela for capturada, o controle pode ser recuperado

# Exemplo

Vamos colocar o código que vai tentar (try) executar o bloco perigoso e, caso o problema seja do tipo `ArrayIndexOutOfBoundsException`, ele será pego (caught). Repare que é interessante que cada exceção no Java tenha um tipo... ela pode ter atributos e métodos.

Adicione um try/catch em volta do for, pegando `ArrayIndexOutOfBoundsException`. O que o código imprime?

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

# Exemplo

```
Console X
<terminated> Teste (1) [Java Application] /caelum/jdk1.5.0_07/bin/java (12/07/2006 4:50:20 PM)
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
```

# Exceções e métodos

- Uma declaração **throws** (observe o 's') é obrigatória em métodos e construtores que deixam de capturar uma ou mais exceções que ocorrem em seu interior

```
public void m() throws Excecao1, Excecao2 {...}  
public Circulo() throws ExcecaoDeLimite {...}
```

- **throws** declara que o método pode provocar exceções do tipo declarado (ou de qualquer subtipo)

- A declaração abaixo declara que o método pode provocar qualquer exceção (nunca faça isto)

```
public void m() throws Exception {...}
```

- Métodos sobrepostos não podem provocar mais exceções que os métodos originais

## Exemplo

Retire o try/catch e coloque ele em volta da chamada do metodo2.

```
System.out.println("inicio do metodo1");  
try {  
    metodo2();  
} catch (ArrayIndexOutOfBoundsException e) {  
    System.out.println("erro: " + e);  
}  
System.out.println("fim do metodo1");
```



```
Console X  
<terminated> Teste (1) [Java Application] /caelum/jdk1.5.0_07/bin/java (12/07/2006 4:56:54 PM)  
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 metodo1  
fim do main
```

# Exceções e métodos

```
public class RelatorioFinanceiro {  
    public void metodoMau() throws ExcecaoContabil {  
        if (!dadosCorretos) {  
            throw new ExcecaoContabil("Dados Incorretos");  
        }  
    }  
    public void metodoBom() {  
        try {  
            ... instruções ...  
            metodoMau();  
            ... instruções ...  
        } catch (ExcecaoContabil ex) {  
            System.out.println("Erro: " + ex.getMessage());  
        }  
        ... instruções ...  
    }  
}
```

*instruções que sempre  
serão executadas*

*instruções serão executadas  
se exceção não ocorrer*

*instruções serão executadas  
se exceção não ocorrer ou  
se ocorrer e for capturada*

## Relançar uma exceção

- Às vezes, após a captura de uma exceção, é desejável relançá-la para que outros métodos lidem com ela
- Isto pode ser feito da seguinte forma

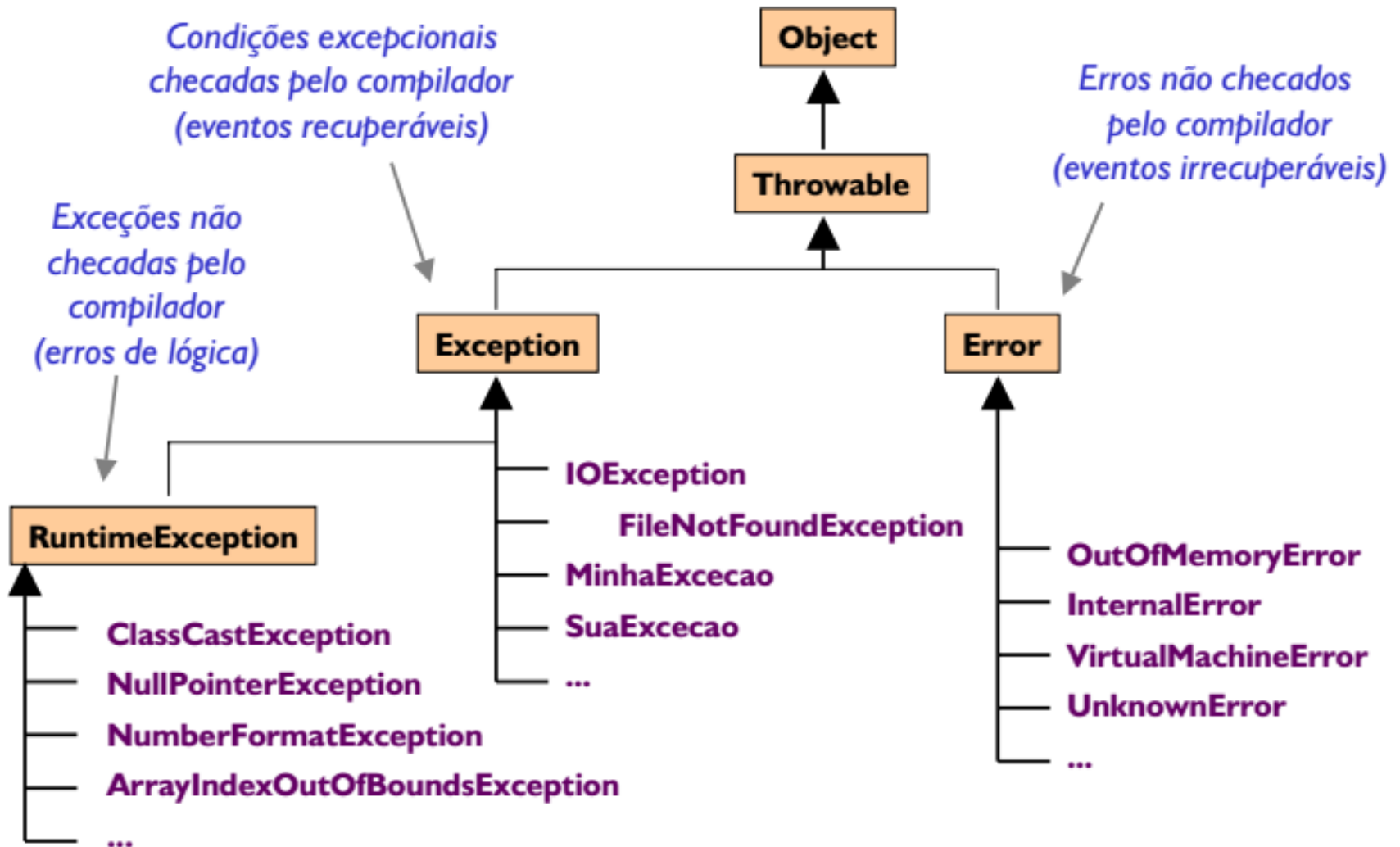
```
public void metodo() throws ExcecaoSimples {  
    try {  
        // instruções  
    } catch (ExcecaoSimples ex) {  
        // faz alguma coisa para lidar com a exceção  
        throw ex; // relança exceção  
    }  
}
```



2.

# Hierarquia de exceções em Java

# Hierarquia



# Erros dos tipos **throwables**

- ❑ **Unchecked:** Erros que acontecem fora do controle do programa, mas que devem ser tratados pelo desenvolvedor para o programa funcionar.
- ❑ **Checked Exception (Runtime) :** Erros que podem ser evitados se forem tratados e analisados pelo desenvolvedor. Caso haja um tratamento para esse tipo de erro, o programa acaba parando em tempo de execução (Runtime).
- ❑ **Error:** Usado pela JVM que serve para indicar se existe algum problema de recurso do programa, tornando a execução impossível de continuar.

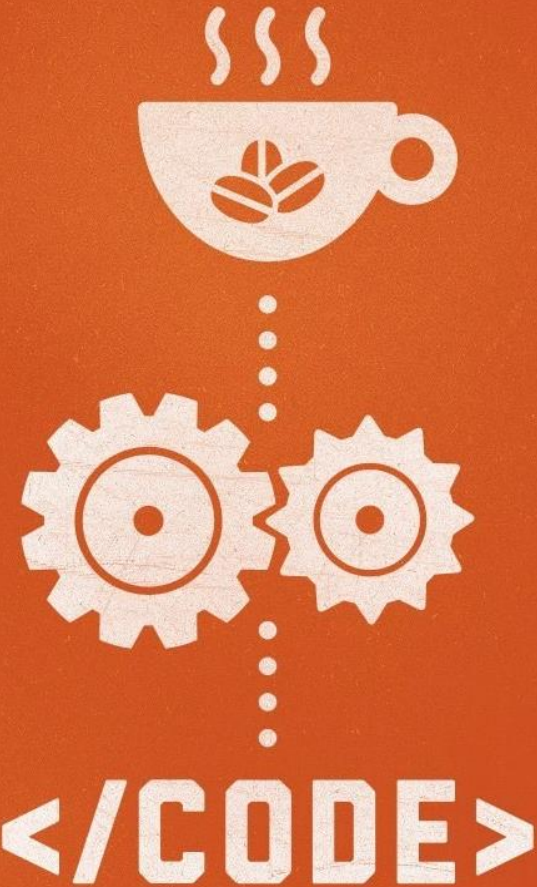
3.

Exceção: Checked Exceptions

## Exceção de *checked*

Exceção de *checked* **obriga** a quem chama o método ou construtor a tratar essa exceção, pois o compilador checará se ela está sendo devidamente tratada, diferente da anterior, conhecidas como *unchecked*.

# PROGRAMMER



# Exemplo

Um exemplo interessante é o de abrir um arquivo para leitura, onde pode ocorrer o erro do arquivo não existir (veremos como trabalhar com arquivos em outro capítulo, **não** se preocupe com isto agora):

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

O código acima não compila e o compilador avisa que é necessário tratar o `FileNotFoundException` que pode ocorrer:

```
Teste.java:3: unreported exception java.io.FileNotFoundException; must be caught  
or declared to be thrown  
        new java.io.FileReader("arquivo.txt");  
        ^  
1 error
```

# Exemplo

Para compilar e fazer o programa funcionar, temos duas maneiras que podemos tratar o problema. O primeiro, é **tratá-lo com o try e catch** do mesmo jeito que usamos no exemplo anterior, com uma array:

```
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");  
    }  
  
}
```

# Exemplo

A segunda forma de tratar esse erro, é delegar ele para quem chamou o nosso método, isto é, passar para a frente.

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



4.

Exceções encadeadas

# Exemplo

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

1. Com o try e catch:

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

2. Com o throws:

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

# Exemplo

3. Você pode, também, escolher tratar algumas exceções e declarar as outras no throws:

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

# Finally

- O bloco `try` não pode aparecer sozinho:
  - deve ser seguido por pelo menos um `catch` ou por um `finally`
  - O bloco `finally` contém instruções que devem se executadas independentemente da ocorrência ou não de exceções

```
try {  
    // instruções: executa até linha onde ocorrer exceção  
} catch (TipoExcecao1 ex) {  
    // executa somente se ocorrer TipoExcecao1  
}  
catch (TipoExcecao2 ex) {  
    // executa somente se ocorrer TipoExcecao2  
}  
finally {  
    // executa sempre ...  
}  
  
// executa se exceção for capturada ou se não ocorrer
```

5.

Declarando novos tipos de  
exceção

## Declarando novos tipos de exceção

- A não ser que você esteja construindo uma API de baixo-nível ou uma ferramenta de desenvolvimento, você só usará exceções do tipo (2)
- Para criar uma classe que represente sua exceção, basta estender `java.lang.Exception`:  

```
class NovaExcecao extends Exception {}
```
- Não precisa de mais nada. O mais importante é herdar de `Exception` e fornecer uma identificação diferente
- Bloco `catch` usa nome da classe para identificar exceções

## Declarando novos tipos de exceção

- Você também pode acrescentar métodos, atributos e construtores como em qualquer classe
- É comum criar a classe com dois construtores

```
class NovaExcecao extends Exception {  
    public NovaExcecao () {}  
    public NovaExcecao (String mensagem) {  
        super(mensagem);  
    }  
}
```

- Esta implementação permite passar mensagem que será lida através de `toString()` e `getMessage()`

# Declarando novos tipos de exceção

## ■ Construtores de Exception

- Exception ()
- Exception (String message)
- Exception (String message, Throwable cause) [Java 1.4]

## ■ Métodos de Exception

- String getMessage() – Retorna mensagem passada pelo construtor
- Throwable getCause() – Retorna exceção que causou esta exceção [Java 1.4]
- String toString() – Retorna nome da exceção e mensagem
- void printStackTrace() – Imprime detalhes (stack trace) sobre exceção





# DESAFIO

E aí, vamos praticar?

# Exercício 1

Implementar uma classe Conta uma classe banco contendo um array de contas com métodos para tratamento de exceções.

- ▷ Os métodos set das classes básicas de negócio lançarão exceções do tipo `ExcecaoDadoInvalido` quando o dado passado como parâmetro não for válido.
- ▷ O método `inserir` da classe `CadastroContas` deve lançar a exceção `ExcecaoRepositorio` quando não puder mais inserir contas no array e a exceção `ExcecaoElementoJaExistente` quando uma conta com um mesmo número já estiver cadastrada.
- ▷ O método `buscar` da classe `CadastroContas` deve lançar a exceção `ExcecaoElementoInexistente` quando a conta que se deseja buscar não estiver cadastrada.

# Obrigado!

## **Alguma pergunta?**

Você pode me contatar em:  
[ywassef@hotmail.com](mailto:ywassef@hotmail.com)