

**Programação**  
**Programação III**

# **Tratamento de Exceções**

**Marco Veloso**

marco.veloso@estgoh.ipc.pt

# Agenda

## Tratamento de Excepções

### **Introdução à Linguagem Java**

- Paradigmas de Programação
- Linguagem Java

### **Programação Orientada a Objectos**

- Objectos
- Classes
- Heranças
- Polimorfismo

### **Tratamento de Excepções**

### **Estruturas de dados**

- Tabelas unidimensionais
- Tabelas multidimensionais
- Vectores
- Dicionários (Hashtables)
- Collections

### **Ficheiros**

- Manipulação do sistema de ficheiros
- Ficheiros de Texto
- Ficheiros Binários
- Ficheiros de Objectos
- Leitura de dados do dispositivo de entrada

# Definição de Excepção

## Tratamento de Exceções

Uma **excepção** (*exception*) é um sinal gerado pela máquina virtual de JAVA, durante a execução do programa, indicando a ocorrência de um **erro recuperável**

São exceções comuns a **tentativa de divisão por zero** ou **indexação para além dos limites de uma tabela**

Uma **excepção é sinalizada ou “lançada”** (*thrown*) a partir do ponto do código em que ocorreu, e diz-se “apanhada” (*caught*) no ponto do código o qual o controlo do fluxo do programa foi transferido

As exceções pode ser do tipo **implícito** (**automaticamente lançadas** pela máquina virtual), ou **explícito** (quando é o próprio **programador a declarar** através da instrução *throw*)

A classe *Excepção* (*Exception*) é uma subclasse da classe `java.lang.Throwable`, estando no topo da respectiva hierarquia. Assim, **todas as exceções são instancias** da respectiva classe de **Excepção**

# Exemplo de Excepção

## Tratamento de Exceções

Por exemplo, uma excepção devida a uma tentativa de **acesso a um índice inexistente numa tabela**, é instancia de **`ArrayIndexOutOfBoundsException`**,

```
java.lang.ArrayIndexOutOfBoundsException: 11
    at Teste.calculo(Teste.java:10)
    at Teste.main(Teste.java:4)
Exception in thread "main" Finished executing
```

classe que se posiciona na hierarquia:

```
java.lang.Object
    < Throwable
    < Exception
    < RuntimeException
    < IndexOutOfBoundsException
    < ArrayIndexOutOfBoundsException
```

A excepção, do exemplo apresentado, refere-se ao acesso a uma posição (11) de uma tabela com uma capacidade inferior, no método `calcula()` (linha 10 do código), invocado pelo método `main()` do programa (linha 4 do código).

Qualquer método que **contenha instruções potencialmente geradoras de erros** ou que chame **métodos que possam propagar erros** deve:

- **Declarar a possibilidade de propagar erros** (através da inclusão de **throws** no seu cabeçalho), **ou**
- **Tratar localmente as eventuais exceções**, utilizando **try . . . catch**

O tratamento de exceções deve ser feito em algum ponto do programa (o tratamento feito pelo sistema operativo consiste em terminar o programa abruptamente).

Como já foi visto anteriormente, as palavras reservadas **throws** **IOException** no **cabeçalho de um método** servem para indicar ao compilador que ele **pode gerar ou propagar um erro do tipo *IOException***.

Essa utilização pode ser evitada se **o método tratar internamente as exceções** (erros) que possam ocorrer, evitando assim a sua propagação

Para isso **enquadram-se as instruções** potencialmente geradoras de exceções **num bloco *try*** ao qual se segue um ou mais blocos ***catch*** (também denominados por *exception handlers*) que indicam o procedimento a seguir caso o erro ocorra.

# Bloco try ... catch

## Tratamento de Exceções

O bloco **try** ... **catch** tem a seguinte sintaxe:

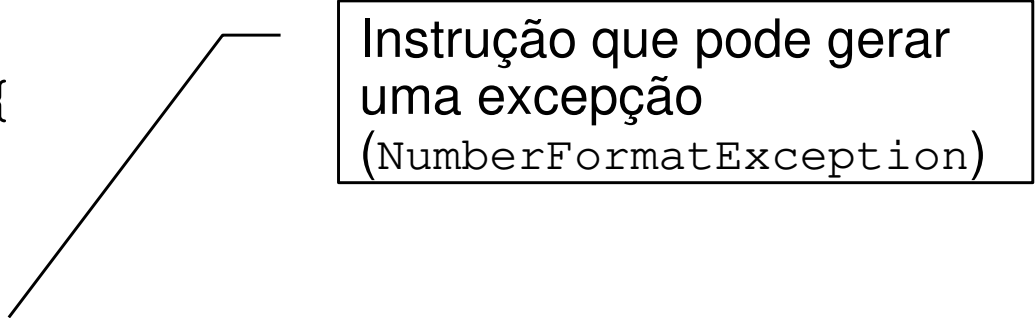
```
try {  
    //Código que faz a acção desejada, mas pode gerar um erro  
}  
catch (ExceptionType1 e1) {  
    //Instruções a executar se ocorrer uma excepção do tipo  
    //ExceptionType1  
}  
catch (ExceptionType2 e2) {  
    //Instruções a executar se ocorrer uma excepção do tipo  
    //ExceptionType2  
}
```

# Aplicação do bloco try ... catch

## Tratamento de Exceções

### Exemplo:

```
public static int readInt() {  
    while (true) {  
        try {  
            return Integer.valueOf(readString()).intValue();  
        } catch (NumberFormatException e) {  
            System.out.println ("!!! Not an integer !!!");  
        }  
    }  
}
```



Instrução que pode gerar uma exceção  
(NumberFormatException)

As associações **try ... catch** podem ser usadas numa relação de 1 para  $n$ , ou seja, **1 bloco try para  $n$  blocos catch**, pois o **mesmo bloco de código pode gerar várias exceções distintas, que podem ser tratadas em separado.**



# Blocos try ... catch ... finally

## Tratamento de Exceções

A cláusula **finally** também pode ser introduzida no bloco, fazendo sentido ser utilizada como **complemento à cláusula try**, permitindo a **execução de um conjunto de instruções após a execução do bloco try**.

A cláusula **finally** apenas é executada desde que pelo menos **uma instrução do bloco try** tenha sido executado e quer tenha ocorrido, ou não, uma exceção.

Usualmente esta cláusula está relacionado com **operações que devem ser realizadas sobre dispositivos lógicos ou físicos**, tais como, por exemplo, fechar um conjunto de ficheiros, encerrar acessos a bases de dado, libertar recursos, etc.

# Blocos try ... catch ... finally

## Tratamento de Exceções

O bloco **try ... catch ... finally** tem a seguinte sintaxe:

```
try {  
    //Código que faz a acção desejada, mas pode gerar um erro  
  
} catch (ExceptionType1 e1) {  
    //Instruções a executar se ocorrer uma excepção do tipo  
    //ExceptionType1  
} catch (ExceptionType2 e2) {  
    //Instruções a executar se ocorrer uma excepção do tipo  
    //ExceptionType2  
} [ finally {  
    //Execução Final  
} ]
```

# Lançamento de Exceções

## Tratamento de Exceções

Por vezes é necessário **“lançar” explicitamente uma exceção no meio da execução de um código**, ou seja alterar para situação de erro ocorrida na execução do código

Para o efeito, basta **recorrer à cláusula `throw` em conjunto com a criação de uma instancia particular da exceção** pretendida usando o operador **`new`**

A linguagem JAVA requer que **qualquer método que possa provocar a ocorrência de uma exceção normal**, ou seja, que possua uma declaração **`throw`**, **faça localmente o tratamento dessa exceção numa cláusula `catch`**, ou **declare explicitamente que pode lançar tal exceção**, embora não trate localmente

No último caso, no cabeçalho de tal método **devem ser explicitamente declaradas as exceções lançadas** através da definição de uma cláusula **`throws`**.

# Aplicação das cláusulas *throws* e *throw*

## Tratamento de Exceções

A situação seguinte, uma **situação potencialmente geradora de uma exceção**, recorrendo à implementação do método `pop()` da classe `Stack`, já que a operação não é aplicável a uma pilha vazia. Esta situação **pode gerar uma exceção mas não sendo tratada localmente**.

```
public void pop() throws EmptyStackException {  
    if (this.empty())  
        throw new EmptyStackException();  
    else  
        numElem -= 1;  
}
```

Lançamento da  
exceção

Declaração da exceção  
(`EmptyStackException`)  
que pode ser lançada  
quando o método é  
invocado

Como o **tratamento da exceção gerada não é feito localmente, é obrigatório a declaração `throws` no cabeçalho do método**, sendo delegada a responsabilidade de tratamento da exceção.

# Aplicação das cláusulas *throws* e *throw*

## Tratamento de Exceções

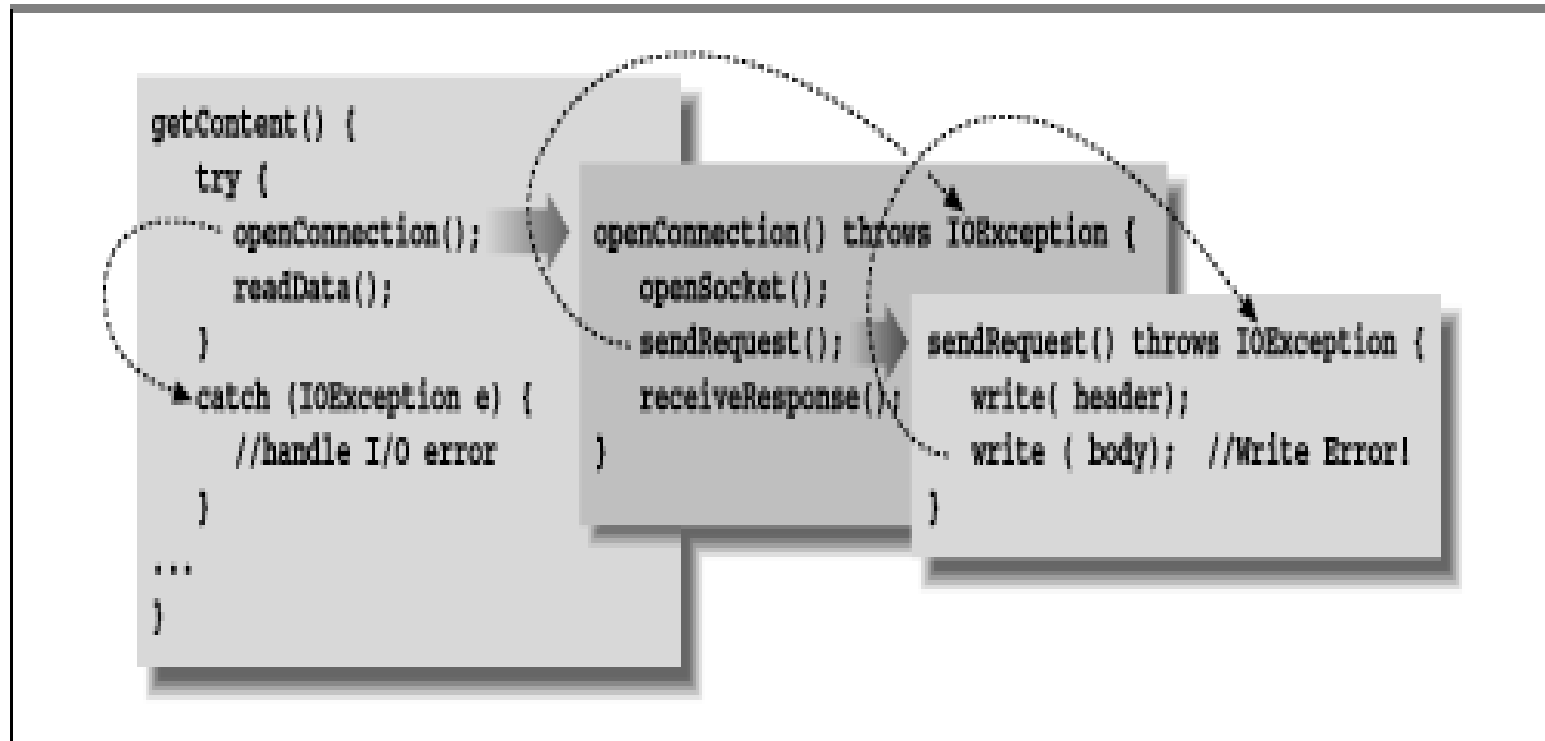
Na eventualidade da **geração de várias exceções distintas**, que não se pretende tratar localmente, pode-se **declarar uma lista de exceções**, separadas por vírgula, tal como apresentado no exemplo seguinte:

```
public void push() throws FullStackException,  
                  IllegalArgumentError,  
                  ArrayIndexOutOfBoundsException {  
  
    //... Código do método  
}
```

# Aplicação das cláusulas *throws* e *throw*

## Tratamento de Exceções

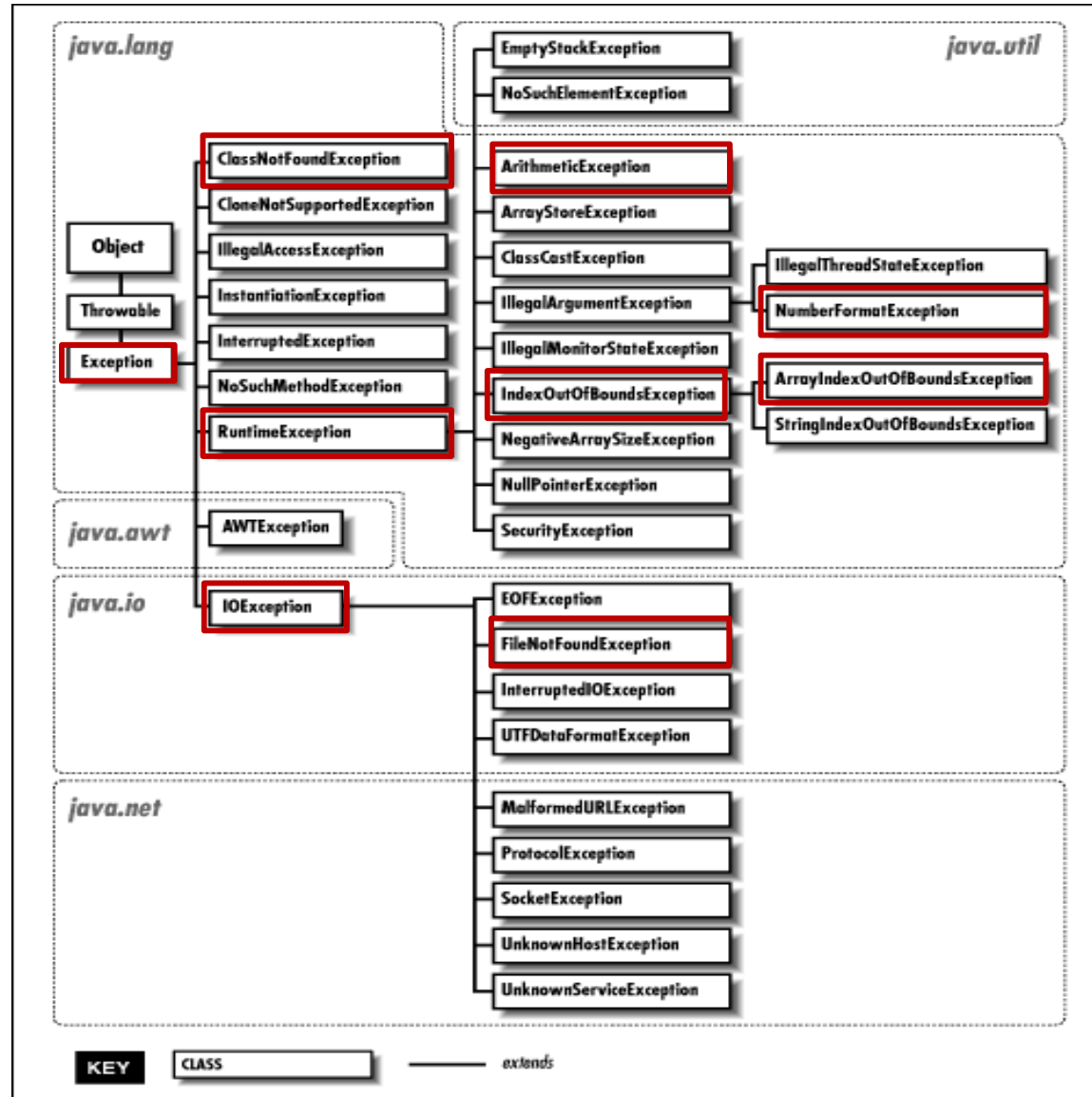
Lançamento (*throws*) de uma exceção, lançada na invocação de um método:



O método `openConnection()`, ao invocar o método `sendRequest()` pode receber uma exceção do tipo `IOException`, no entanto delega o seu tratamento a quem o tenha invocado (*throws* `IOException`).

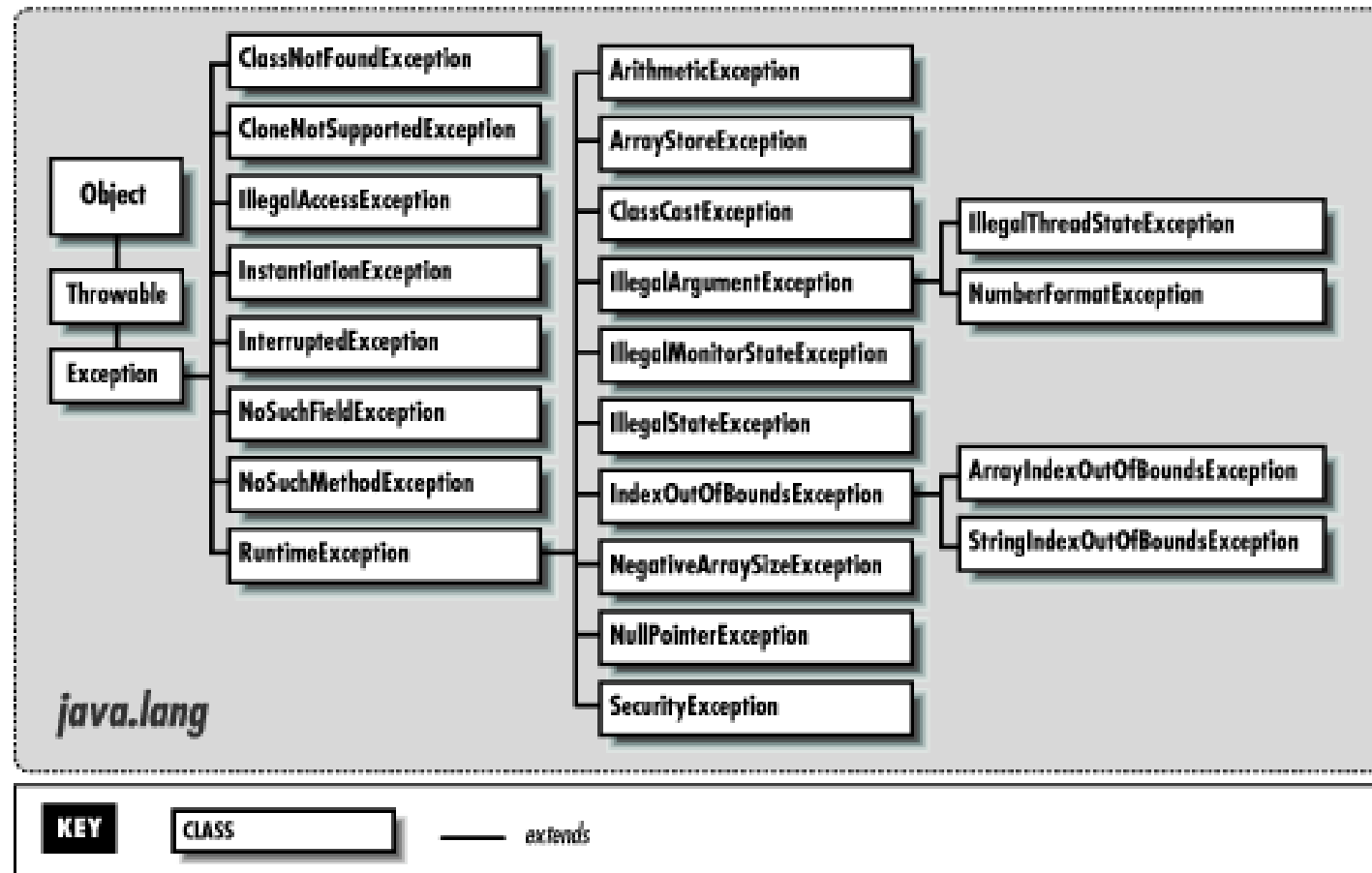
# Hierarquia de classes de Exceções

## Tratamento de Exceções



# Classes de Exceções: package *java.lang*

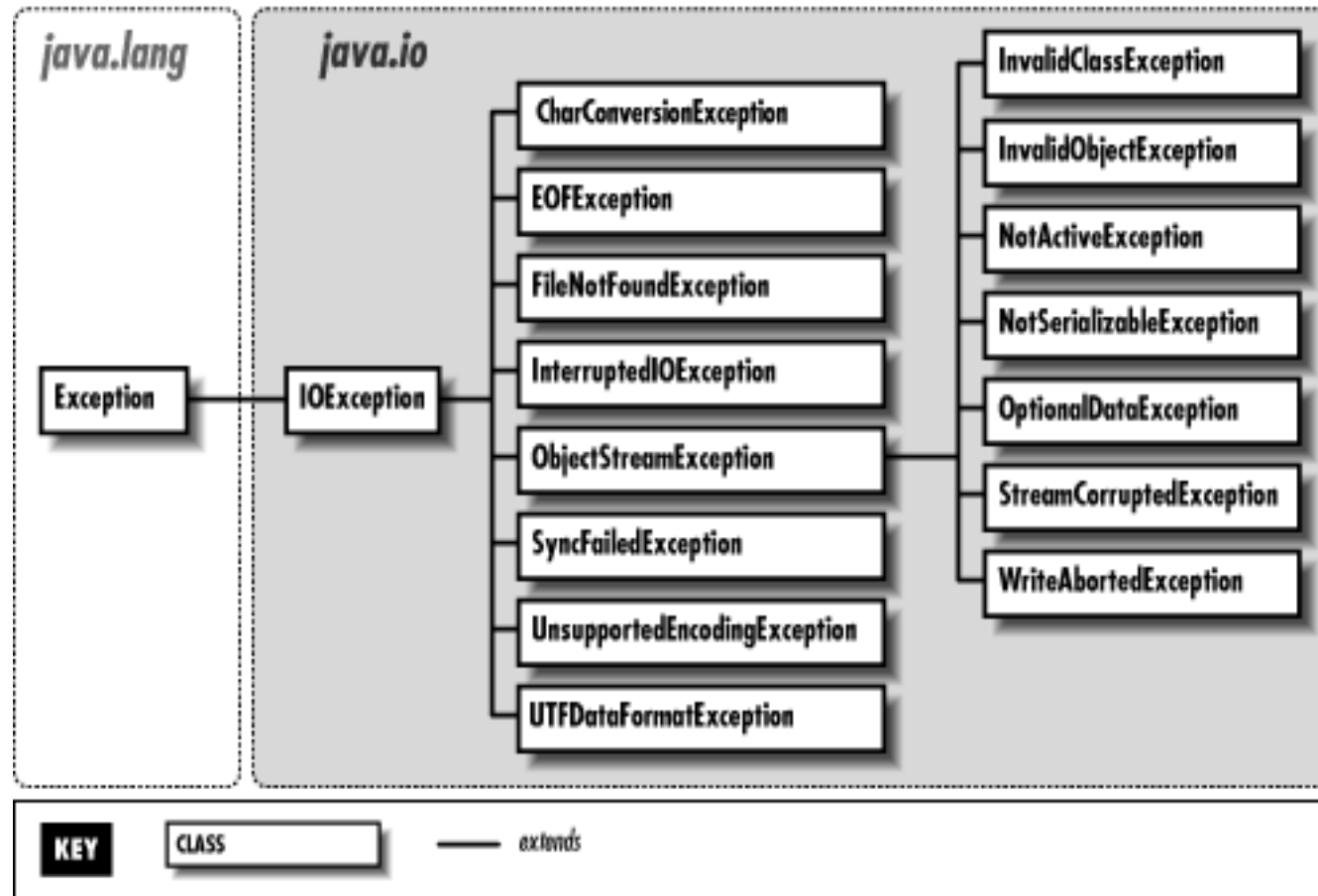
## Tratamento de Exceções





# Classes de Exceções: package *java.io*

## Tratamento de Exceções



# Declaração de Exceções

## Tratamento de Exceções

Sendo as exceções particulares não mais do que **instancias das diversas classes de excepção já existentes em JAVA**, a criação pelo próprio programador de novas classes exceções consiste, via reutilização, em:

**definição dos identificadores** das suas classes particulares de excepção;  
correcta declaração do **posicionamento destas novas classes na hierarquia de classes de excepção** que já estão definidas em JAVA.

Assim, o código típico para a criação de uma nova classe de excepção, que se designa de **NovaExcepcao**, herdando a implementação da definição da classe **Exception**:

```
public class NovaExcepcao extends Exception {  
    NovaExcepcao()           { super(); }  
    NovaExcepcao(String s) { super(s); }  
}
```

Definindo-se apenas dois construtores de excepção, usando as referências `super()` e `super(s)` para invocação do código dos construtores da superclasse.

**"*Java in a Nutshell*"**, 4ª Edição, Capítulo 3 "*Object-Oriented Programming in Java*"

David Flanagan

O'Reilly, ISBN: 0596002831

**"*Thinking in Java*, "**, 4ª Edição, Capítulo 3 "*Controlling Program Flow*"

Bruce Eckel

Prentice Hall, ISBN: 0131872486

**"*The Java Tutorial – Essential Classes: Exceptions*"**

Java Sun Microsystems

<http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html>

**"Fundamentos de Programação em Java 2"**, Capítulo 10 "*Ficheiros*"

António José Mendes, Maria José Marcelino

FCA, ISBN: 9727224237

**"Java 5 e Programação por Objectos"**, Capítulo 9 "*Excepções*"

F. Mário Martins

FCA, ISBN: 9727225489