

Linguagem JAVA

Exceções

(Livro *Big Java, Late Objects* – Capítulo 7)

- [Conceitos Básicos](#)
- [Captura e Tratamento de Exceções](#)
- [Lançamento de Exceções](#)
- [Bibliografia](#)

- [Conceitos Básicos](#)
- [Captura e Tratamento de Exceções](#)
- [Lançamento de Exceções](#)
- [Bibliografia](#)



- [Evento Excecional](#)
 - [Noção](#)
 - [Exemplos](#)
 - [Causas](#)
- [Mecanismo de Exceções](#)
- [Definição de Exceção](#)
- [Tipos de Exceção](#)
 - [Erro](#)
 - [Exceção](#)
- [Hierarquia de Classes](#)
 - [Erros e Exceções](#)
 - [Erros](#)
 - [Exceções](#)
- [Tipos de Classes de Exceção](#)
 - [Não-Runtime](#)
 - [Runtime](#)
- [Declaração de Classe de Exceção](#)
 - [Nativa](#)
 - [Definida pelo Programador](#)

- Noção

- É um acontecimento
- Ocorre
 - Tempo de execução do programa // run-time
- Provocado
 - Por erros de execução do programa

Exemplo 1

- Abertura de ficheiro inexistente

Exemplo de Programa

```
7 public class Main {  
8     public static void main(String[] args) throws FileNotFoundException {  
9         Scanner ficheiroDeDados = new Scanner(new File("c:/PPROG/Dados"));  
10    }  
11 }
```

Em Execução

- Ficheiro inexistente ⇒ Erro de execução ⇒ Evento exceccional FileNotFoundException ⇒ Programa termina abruptamente

Saída indica

- Tipo de evento exceccional
- Origem do evento // classe, método e linha
- Sequência de métodos envolvidos // ordem inversa da invocação
- Ponto de ocorrência do evento

```
Output - Excepciones (run)  
run:  
Exception in thread "main" java.io.FileNotFoundException: c:\PPROG\dados (The system cannot find the path specified)  
    at java.io.FileInputStream.open(Native Method)  
    at java.io.FileInputStream.<init>(FileInputStream.java:106)  
    at java.util.Scanner.<init>(Scanner.java:636)  
    at excepciones.Main.main(Main.java:9)
```

Tipo de Evento Exceccional

Origem do Evento (classe, método e linha)

- Exemplo 2
 - Índice de array fora dos limites
- Exemplo de Programa

```
5 public class Main {  
6     public static void main(String[] args) {  
7         Scanner ler = new Scanner(System.in);  
8  
9         String[] nomes = new String[2];  
10        for (int i = 0; i <= nomes.length; i++) {  
11            System.out.println("Indique um nome:");  
12            nomes[i] = ler.nextLine();  
13        }  
14    }  
15 }
```

- Em Execução
 - Índice fora dos limites \Rightarrow Erro de execução \Rightarrow Evento exceccional `ArrayIndexOutOfBoundsException` \Rightarrow Programa termina abruptamente
- Saída

```
Output - Excepcões (run)  
run:  
Indique um nome:  
Roberto  
Indique um nome:  
Luisão  
Indique um nome:  
Luis Filipe  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2  
    at excepcoes.Main.main(Main.java:12)
```

Tipo de Evento
Exceccional

Origem

▪ São Indesejados

- Terminação abrupta do programa
 - Pode provocar perda de trabalho

▪ Causas

- Erros de programação
 - Exemplos
 - Índice de array fora dos limites
 - Divisão por zero
 - Podem ser evitados
- Circunstâncias externas ao programa
 - Exemplos
 - Erros nos dados de entrada fornecidos // Ex: dados inválidos
 - Utilizador
 - Ficheiro // Ex: ficheiro inexistente
 - Rede
 - Limitações físicas do hardware // Ex: disco cheio e memória cheia
 - Erros de dispositivos // Ex: falha na rede e falha na impressora
 - Não podem ser evitados completamente

▪ Conclusão

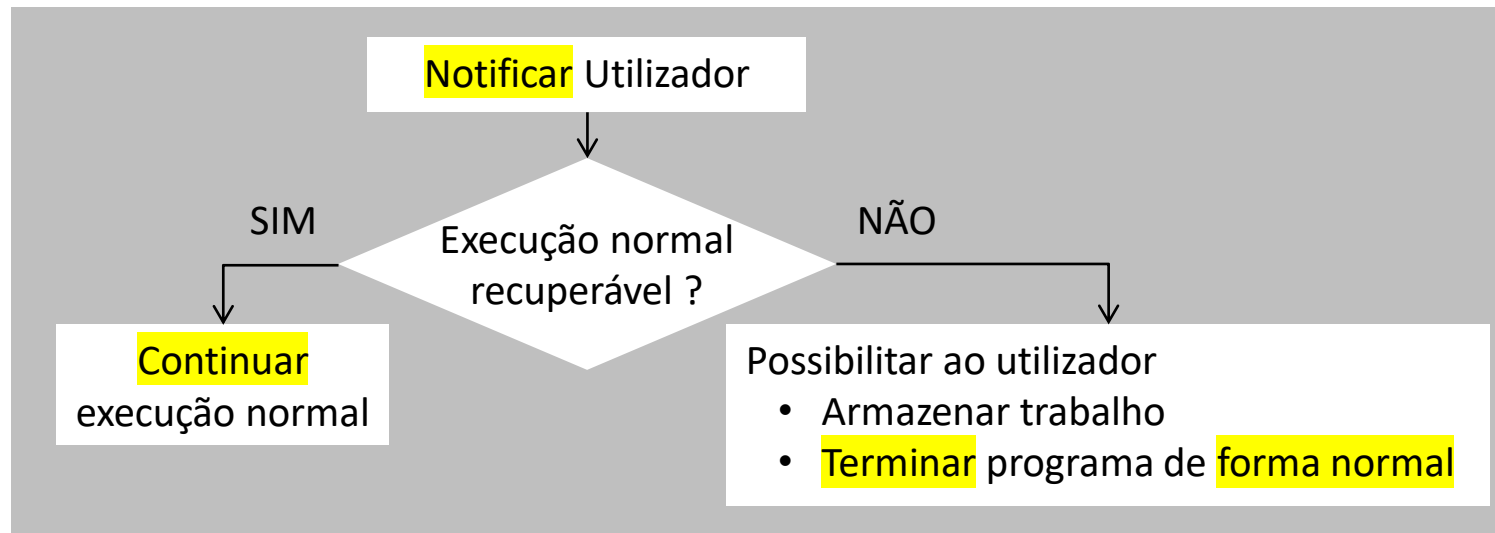
- Impossível evitar completamente ... eventos excepcionais.

- Solução Java

- Mecanismo de Exceções

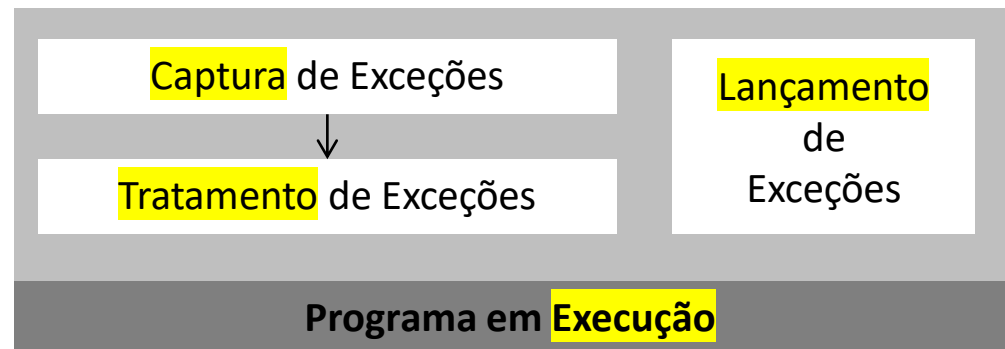
Objetivo

- Contribuir para criação de programas fiáveis // i.e., programas robustos
 - Permitindo aos programas em eventos excepcionais
 - Evitar terminação abrupta
 - Proceder da seguinte forma



Facilita aos Programas

- Captura e Tratamento de Exceções
- Lançamento de Exceções

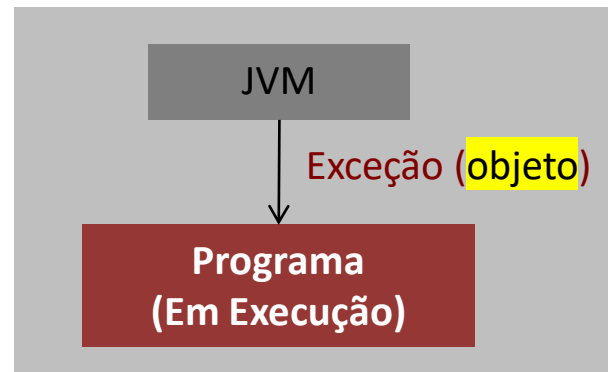


Exceção

- É um **evento** excepcional
 - Gerado** pela JVM em **tempo de execução** de um programa
 - Comunicado** a esse programa
 - Para indicar **ocorrência de erro** que provocou **interrupção** do fluxo de **execução normal**
 - Através de **objetos** da **Hierarquia de Classes de Erros e Exceções**

JVM (Java Virtual Machine)

- Executa programas Java



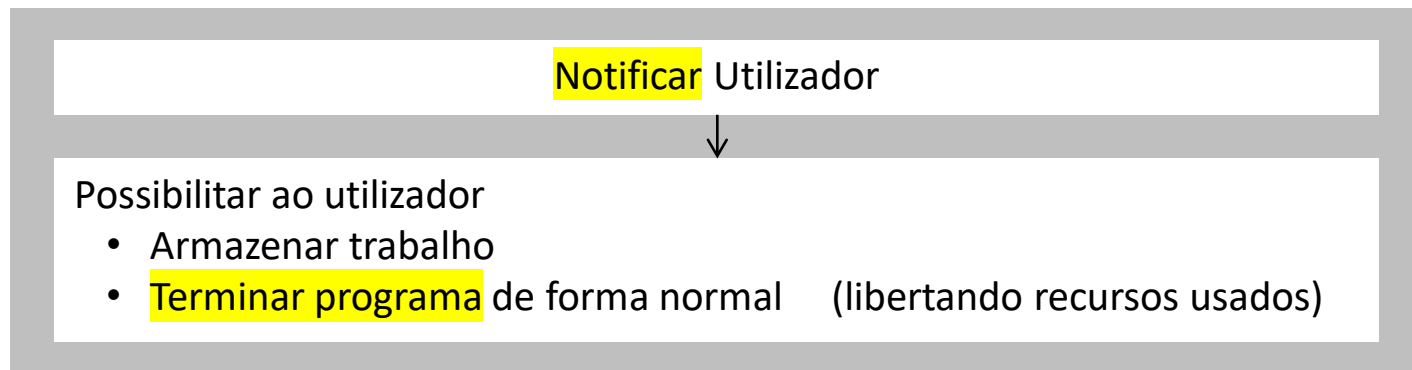
Tipos de Exceção

- Erro // causado por erro dentro da JVM; Ex: memória cheia
- Exceção // causado por erro fora da JVM; Ex: divisão por zero

Erro ≠ Exceção

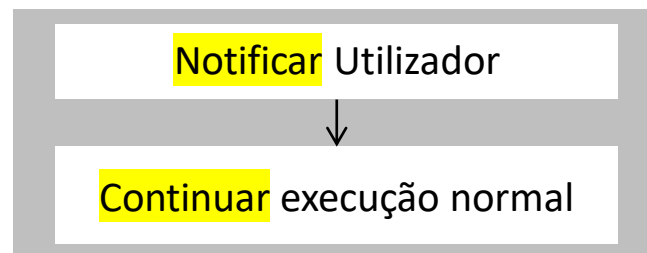
Erro

- Muito pouco frequente
- Programa tem muito pouco a fazer

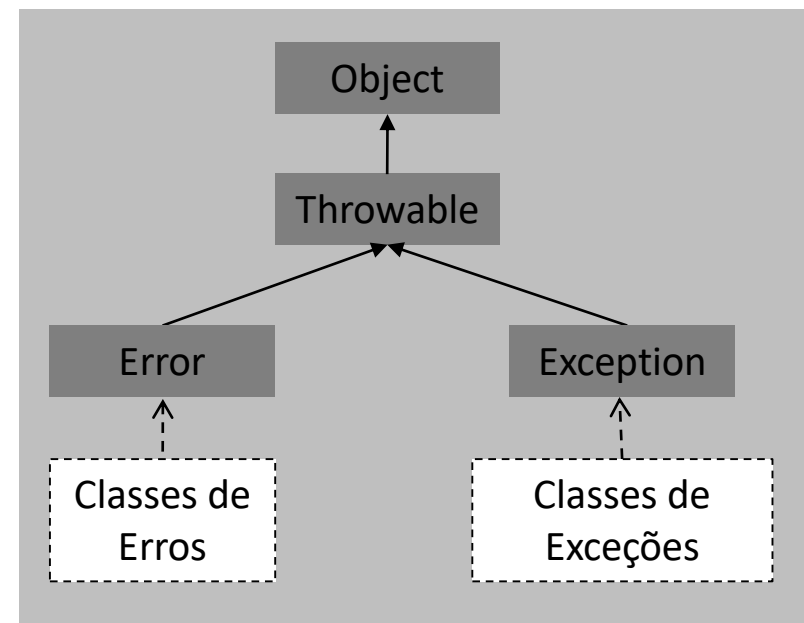


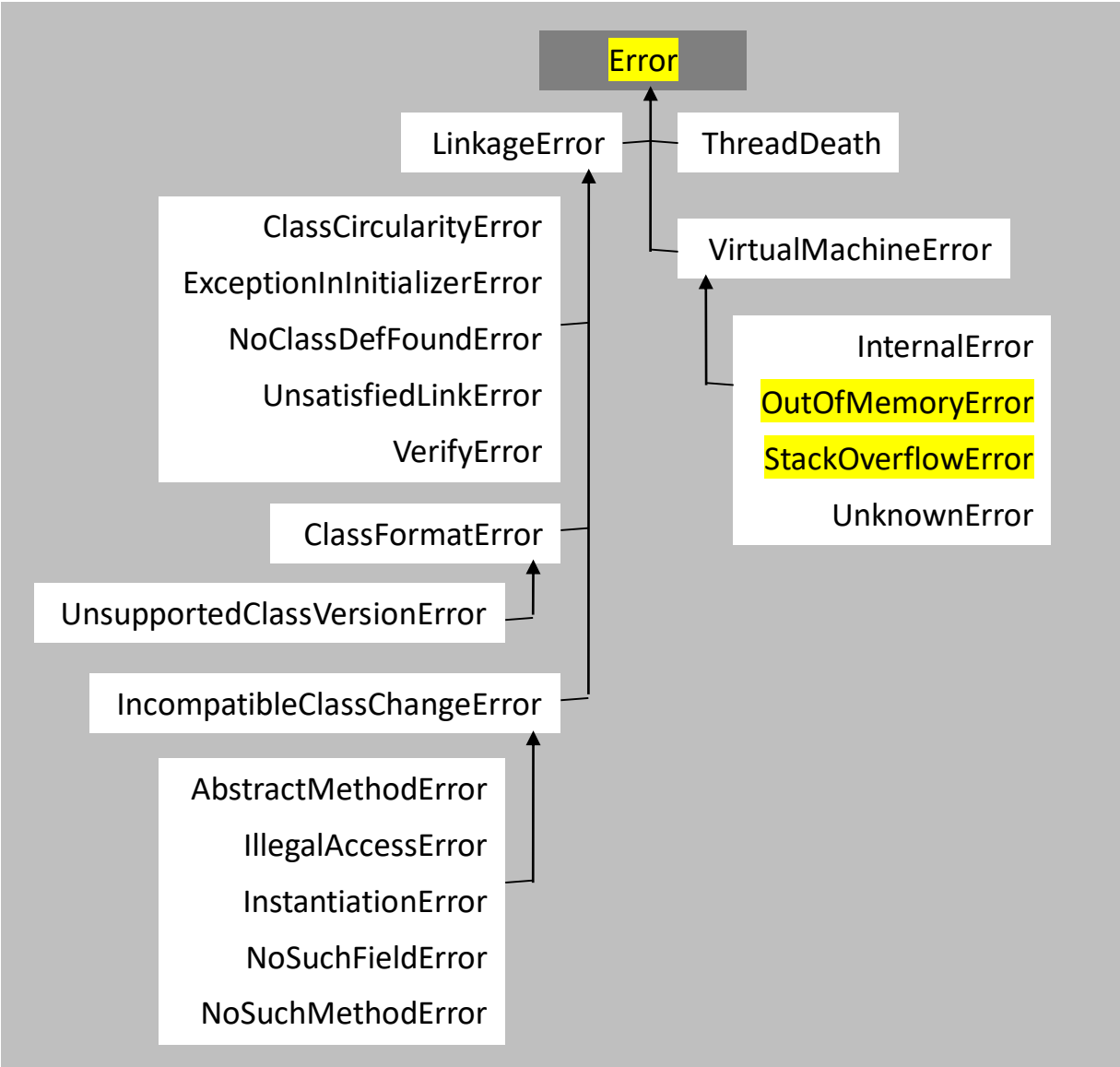
Exceção

- Mais frequente
- Pode permitir a programa recuperar funcionamento normal

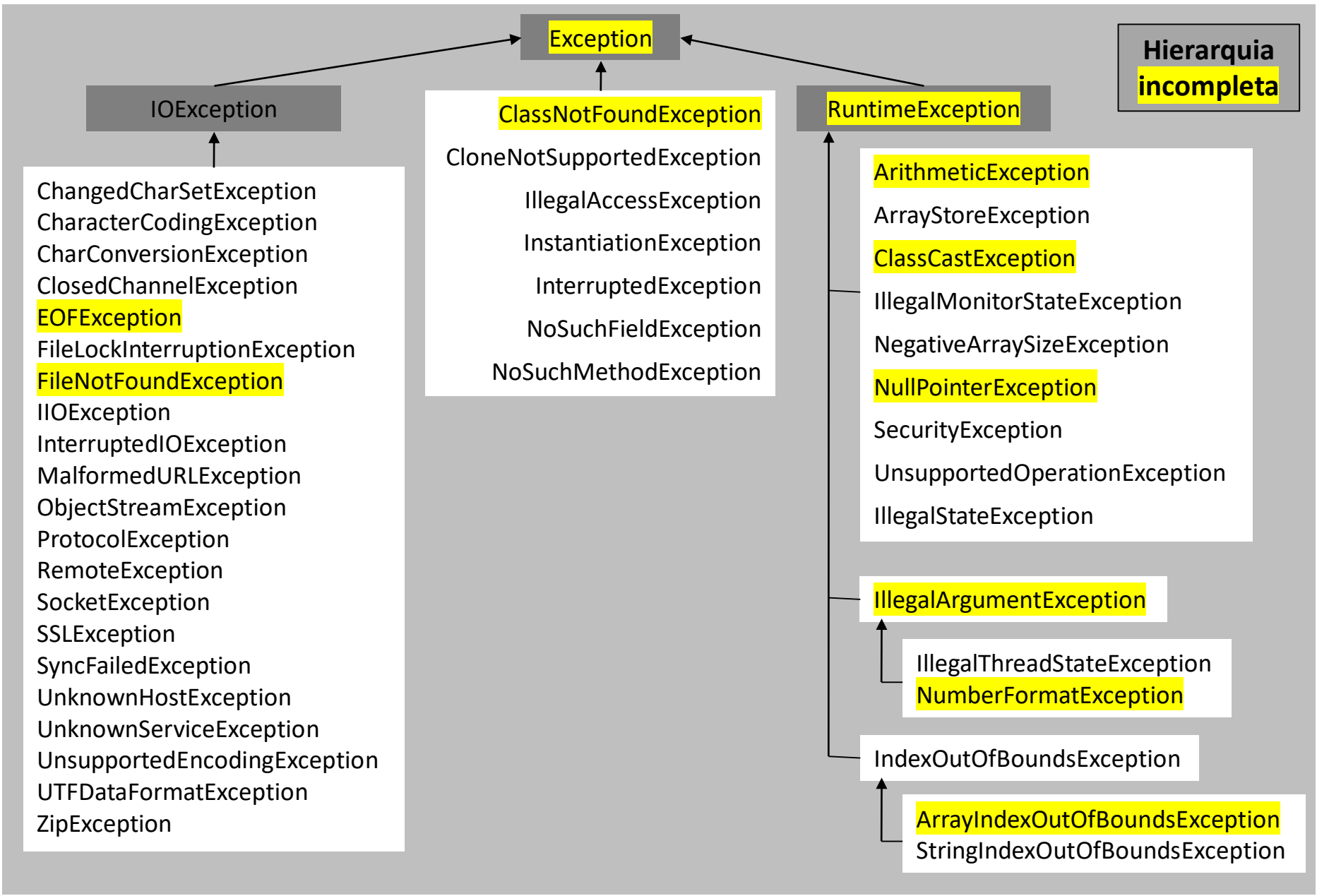


- **Classes de Erros**
 - Criam objetos para comunicação de erros
 - Pertencem à Hierarquia de Erros
- **Classes de Exceções**
 - Criam objetos para comunicação de exceções
 - Pertencem à Hierarquia de Exceções
- **Classe Throwable**
 - **Topo** das Hierarquias
 - Erros
 - Exceções
 - **Superclasse** de Erros e Exceções
- **Classe Error**
 - **Superclasse** de todas as classes de erros
- **Classe Exception**
 - **Superclasse** de todas as classes de exceções
 - Nativas
 - Próprias // definidas pelo programador

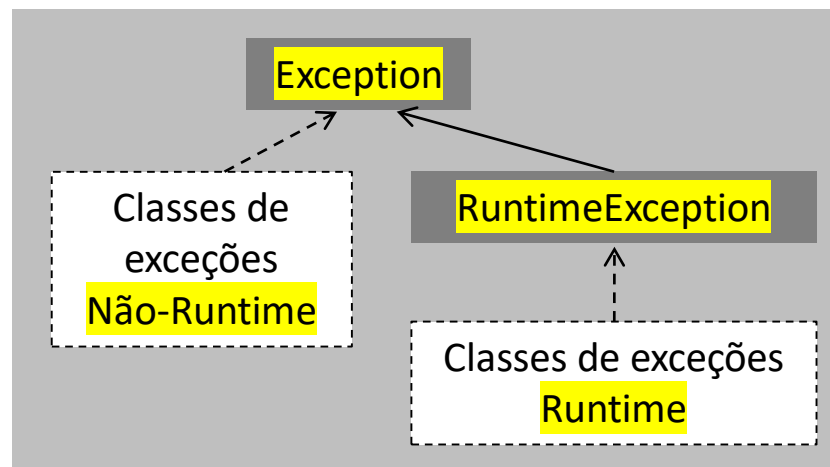




Hierarquia
incompleta



- Tipos de Classe de Exceção
 - Runtime (unchecked)
 - Não-Runtime (checked)



Em geral

- Causadas por circunstâncias externas ao programa
- Não são originadas por erros de programação
- Programa não pode evitar

Exemplo

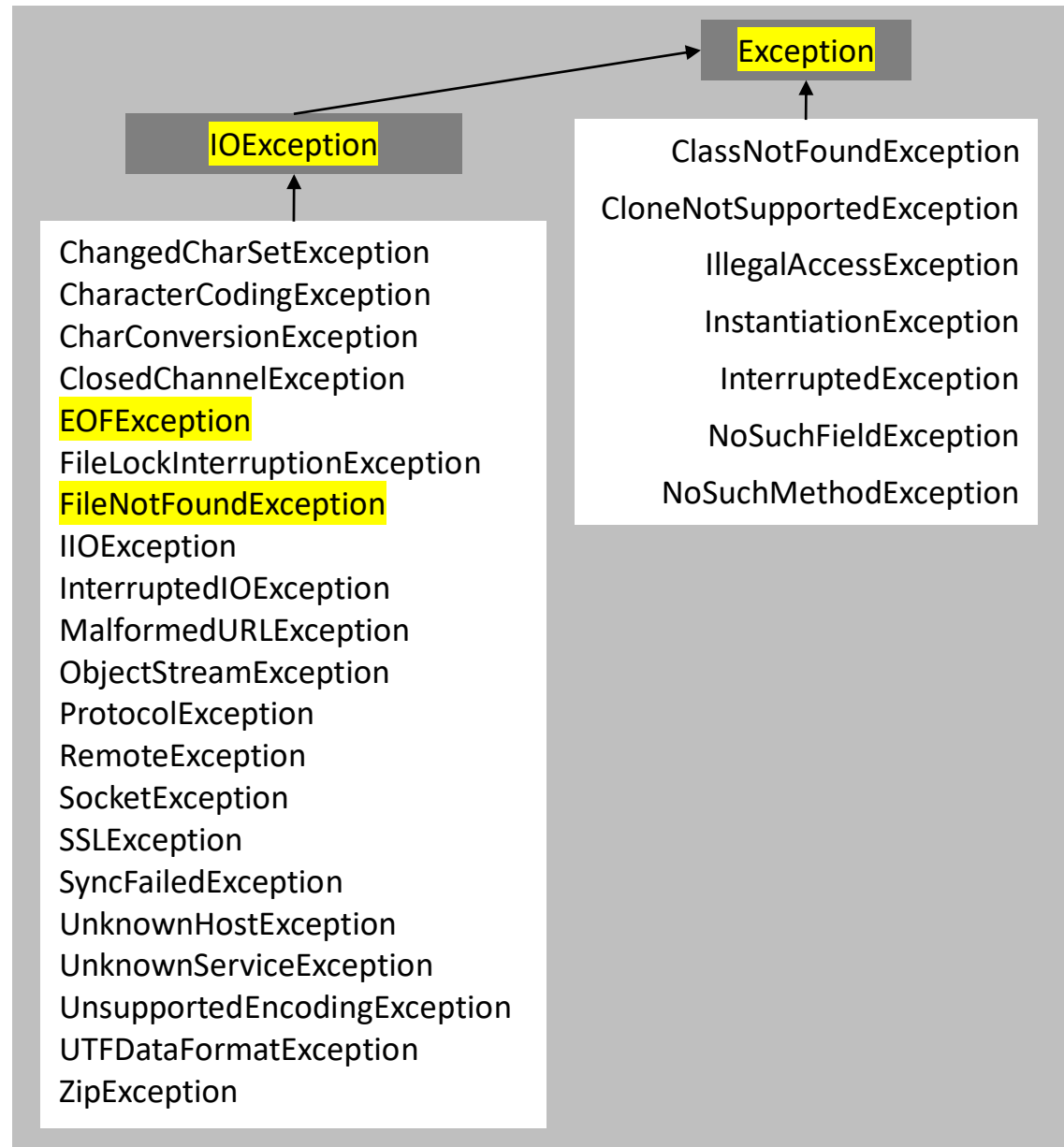
- Insucesso na abertura de ficheiro (erro de I/O)
 - FileNotFoundException

Têm de ser capturadas e tratadas pelos programas

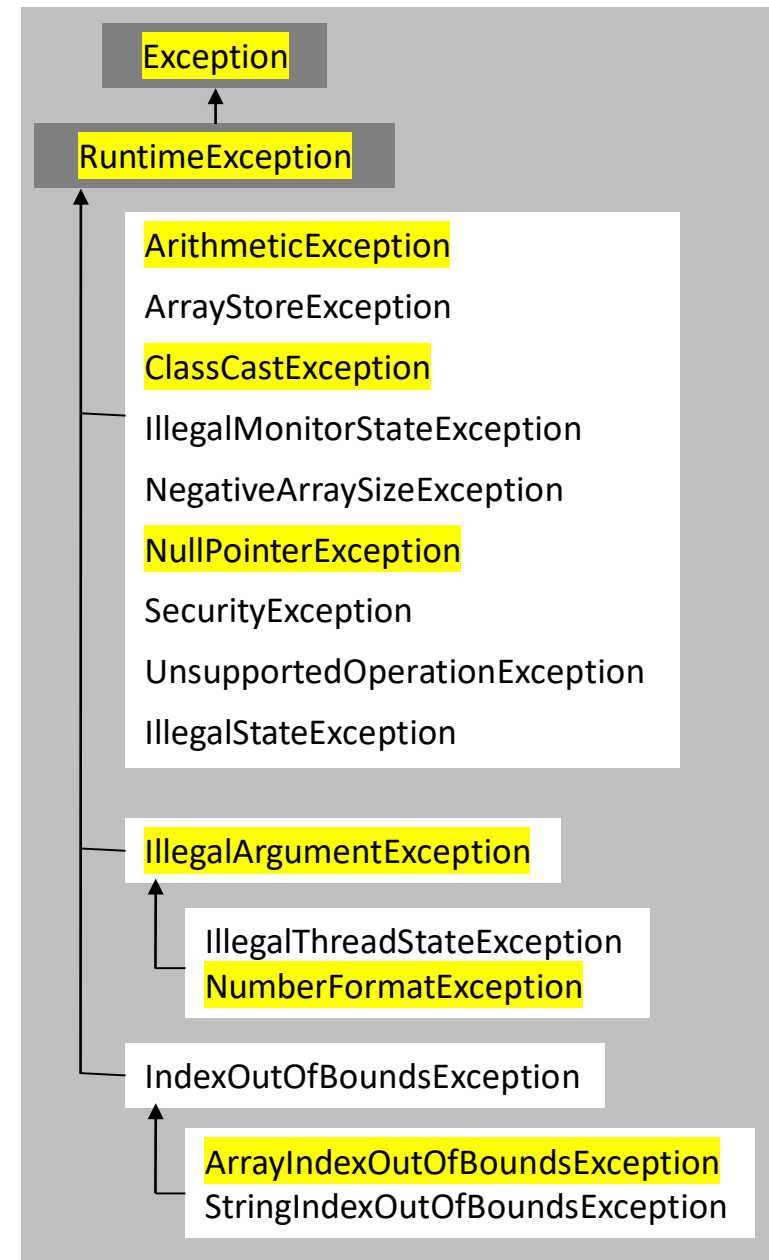
- Compilador verifica se o programa captura e trata este tipo de exceção

Designadas

- Exceções checked



- **Em geral**
 - Originadas por erros de programação
 - Podem ser evitáveis
- **Exemplos**
 - Divisão por zero
 - ArithmeticException
 - Índice de array fora dos limites
 - ArrayIndexOutOfBoundsException
 - Casting inválido
 - ClassCastException
- **Não têm de ser capturadas e tratadas pelos programas**
 - Compilador não verifica se programa captura e trata este tipo de exceção
 - Razões
 - Algumas exceções são muito difíceis de determinar pelo programador
 - Código ficaria ilegível
- **Designadas**
 - Exceções unchecked



- **Classes de Exceção**

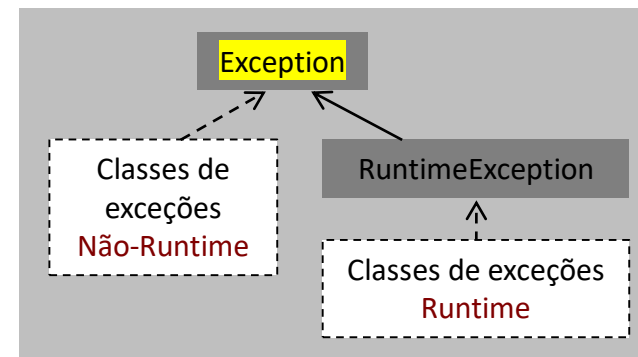
- São muito simples
- Possuem apenas construtores
 - Em geral, 2
 - Sem parâmetros
 - Com parâmetro tipo String // permite enviar informação da exceção
 - Invocam construtores da superclasse

- **Exemplos de classes de exceção nativas**

```
public class ArithmeticException extends RuntimeException {  
    public ArithmeticException() { super(); }  
    public ArithmeticException( String s ) { super(s); }  
}
```

```
public class ArrayIndexOutOfBoundsException extends IndexOutOfBoundsException {  
    public ArrayIndexOutOfBoundsException() { super(); }  
    public ArrayIndexOutOfBoundsException( String s ) { super(s); }  
    public ArrayIndexOutOfBoundsException( int index ) { super("Array index out of range: " + index); }  
}
```

- **Interesse**
 - Quando as classes de exceções **nativas não servem**
- **Requisito**
 - Classe pertencer à **Hierarquia de Exceções**
- **Tipo de Exceção**
 - **Runtime (unchecked)**
 - **Subclasse** de classes **Runtime**
 - Exemplo



```
4 public class ArgumentoNegativoOuNuloException extends IllegalArgumentException {
5
6     public ArgumentoNegativoOuNuloException() {
7         super("Atenção: Argumento Negativo ou Nulo!");
8     }
9
10    public ArgumentoNegativoOuNuloException(String mensagem) {
11        super(mensagem);
12    }
13
14 }
```

- **Não-Runtime (checked)**
 - **Subclasse** da classe **Exception**
 - Subclasse de qualquer classe de exceções Não-Runtime

- [Conceitos Básicos](#)
- [Captura e Tratamento de Exceções](#)
- [Lançamento de Exceções](#)
- [Bibliografia](#)



- [Introdução](#)
- [Cláusulas](#)
 - [try](#)
 - [catch](#)
 - [finally](#)
- [Estrutura Geral de Código](#)
 - Captura e Tratamento
- [Exemplos](#)

▪ Mecanismo de Exceções permite a Programa

1. Capturar exceções
2. Fazer tratamento adequado dessas exceções
 - Correção, se possível
3. Recuperar execução normal

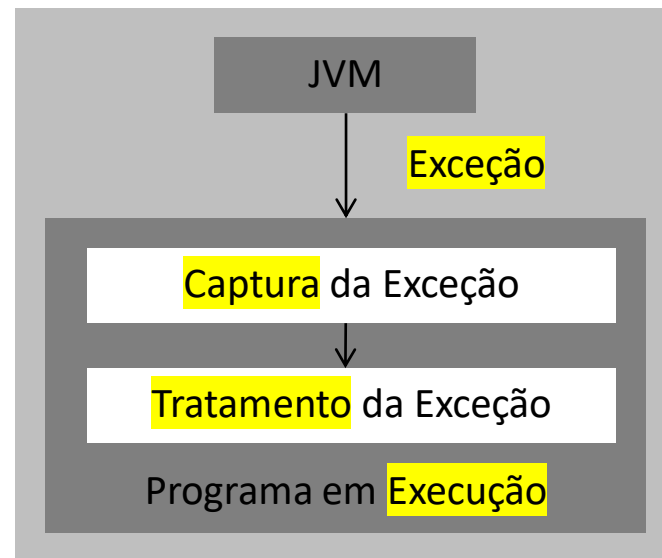
▪ Exemplo

- Permite a programa que executa divisão por zero
 - Corrigir resultado dessa operação
 - Continuar execução normal de seguida

▪ Programação

- Requer conhecimentos sobre:
 - Hierarquia de Classes de Exceções
 - Cláusulas de captura e tratamento
 - Estrutura geral do código de captura e tratamento

// define objetos de exceções a capturar



try { ... }

- Define
 - Bloco de código **susceptível de gerar** erros de execução

```
Scanner ler = new Scanner(System.in);  
try {  
    int x = Integer.parseInt( ler.next() );  
} catch ( NumberFormatException e ) {  
  
    System.out.println( e.getMessage() );  
  
    // alternativa: System.out.println( e );  
}
```

Retorna **informação** sobre a exceção capturada (**guardada** em e)

catch(Tipo_de_Exceção parâmetro){ ... }

- Define
 - Tipo de exceção** a capturar // classe de exceções
 - Parâmetro** recebe a exceção capturada // parâmetro = **objeto exceção**
 - Bloco de **código para tratar** esse tipo de exceção capturado // chamado **handler** da exceção

finally { ... }

- Opcional**
- Complementa **try**
- Usado sempre** que try seja executado
 - Independentemente da ocorrência de exceção em try

- Um bloco try, múltiplos catch e, opcionalmente, um finally // por esta ordem

```
try {  
    // código susceptível de gerar erros de execução  
  
} catch ( Tipo_exceção_1 parâmetro ) {  
    // código para tratar qualquer Tipo_exceção1 ou subtipo  
}  
...  
} catch ( Tipo_exceção_N parâmetro ) {  
    // código para tratar qualquer Tipo_exceçãoN ou subtipo  
}  
finally {  
    // código executado quando try é usado  
}
```

1 try

Múltiplos catch

- Para diferenciar tratamentos
- Exceção ocorrida em try é capturada pelo primeiro bloco catch do seu tipo ou de uma sua superclasse
- Primeiros mais específicos
- Últimos mais genéricos
- Tipo Exception captura tudo (tem de ser o último)

1 finally

- Opcional

- Torna programa legível
 - Separa claramente dois tipos de código: código normal e código excecional.

- Leitura de dados do utilizador simples e mensagem de erro com informação não enviada pela exceção

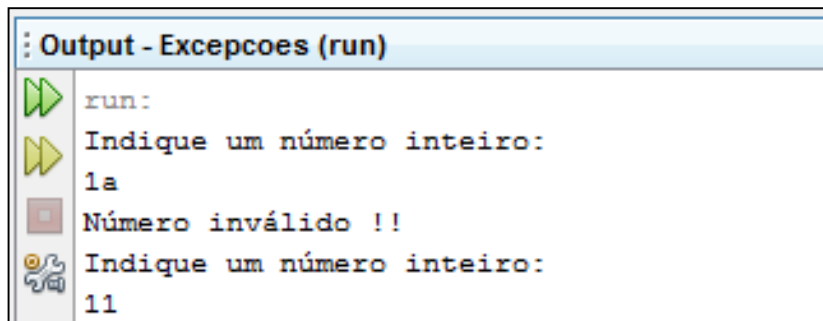
- Programa Fonte:

```
6 public class Main {  
7     public static void main(String[] args) {  
8         Scanner ler = new Scanner(System.in);  
9         boolean invalido = true;  
10        do {  
11            try {  
12                System.out.println("Indique um número inteiro:");  
13                int num = Integer.parseInt(ler.next());  
14                invalido = false;  
15            } catch (Exception e) {  
16                System.out.println("Número Inválido!!");  
17            }  
18        } while (invalido);  
19    }  
20 }  
21
```

Tipo Exception
captura qualquer
exceção

- Programa em execução

- Saída:



```
Output - Excepcões (run)  
run:  
Indique um número inteiro:  
1a  
Número inválido !!  
Indique um número inteiro:  
11
```

- Leitura de dados do utilizador simples e mensagem de erro com informação enviada pela exceção

- Programa Fonte:

```
6 public class Main {
7     public static void main(String[] args) {
8         Scanner ler = new Scanner(System.in);
9         boolean invalido = true;
10        do {
11            try {
12                System.out.println("Indique um número inteiro:");
13                int num = Integer.parseInt(ler.next());
14                invalido = false;
15            } catch (Exception e) {
16                System.out.printf("Exception %s %n", e.getMessage());
17            }
18        } while (invalido);
19    }
20 }
21
```

- Programa em execução

- Saída:

```
Output - Excepcoes (run)
run:
Indique um número inteiro:
1a
Exception For input string: "1a"
Indique um número inteiro:
1
```

mensagem de erro
veiculada pela exceção
capturada

- Diferentes Tratamentos das Exceções: ler dados do utilizador (corrigível) e ler ficheiro (não corrigível)

```
7 public class Main {  
8     public static void main(String[] args) {  
9         Scanner ficheiro=null, ler = new Scanner(System.in);  
10        boolean invalido = true;  
11        do {  
12            try {  
13                System.out.println("Indique um número inteiro:");  
14                int num = Integer.parseInt(ler.next());  
15                invalido = false;  
16                ficheiro = new Scanner(new File("c:/Dados"));  
17                while (ficheiro.hasNext()) {  
18                    System.out.println(ficheiro.nextLine());  
19                }  
20            } catch (FileNotFoundException e) {  
21                System.out.println("Ficheiro não encontrado!!");  
22            } catch (NumberFormatException e) {  
23                System.out.println("Número Inválido!!");  
24            } catch (Exception e) {  
25                System.out.println(e.getMessage());  
26            }  
27        } while (invalido);  
28    }  
29 }
```

`FileNotFoundException` é
não-runtime ⇒ obrigatório
tratamento local ou
lançamento

Saída no slide seguinte



- **Diferentes Tratamentos** das Exceções: ler dados do utilizador (corrigível) e ler ficheiro (não corrigível)

```
Output - Excepcoes (run)
run:
Indique um número inteiro:
1
Ficheiro não encontrado!!
```

```
Output - Excepcoes (run)
run:
Indique um número inteiro:
1a
Número Inválido!!
Indique um número inteiro:
1
Ficheiro não encontrado!!
```

- Captura e tratamentos em métodos não principais (main)

```
8 public class Main {
9
10     public static void main(String[] args) {
11         System.out.println(lerNomes());
12     }
13
14     private static String lerNomes() {
15         StringBuilder nomes = null;
16         try {
17             Scanner ficheiro = new Scanner(new File("c:/Nomes"));
18             nomes = new StringBuilder();
19             while (ficheiro.hasNext()) {
20                 nomes.append(ficheiro.nextLine());
21             }
22             nomes.toString();
23         } catch (FileNotFoundException e) {
24             System.out.println("Ficheiro não encontrado!!!");
25         } catch (Exception e) {
26             System.out.println(e.getMessage());
27         }
28         return nomes.toString();
29     }
30 }
```

`FileNotFoundException` é não-runtime
⇒ obrigatório tratamento local ou
lançamento

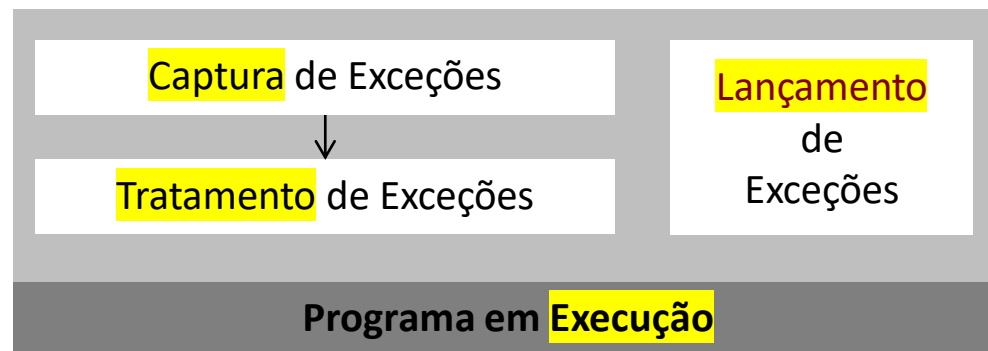
Exemplo do Lançamento na próxima
secção

- [Conceitos Básicos](#)
- [Captura e Tratamento de Exceções](#)
- [Lançamento de Exceções](#)
- [Bibliografia](#)

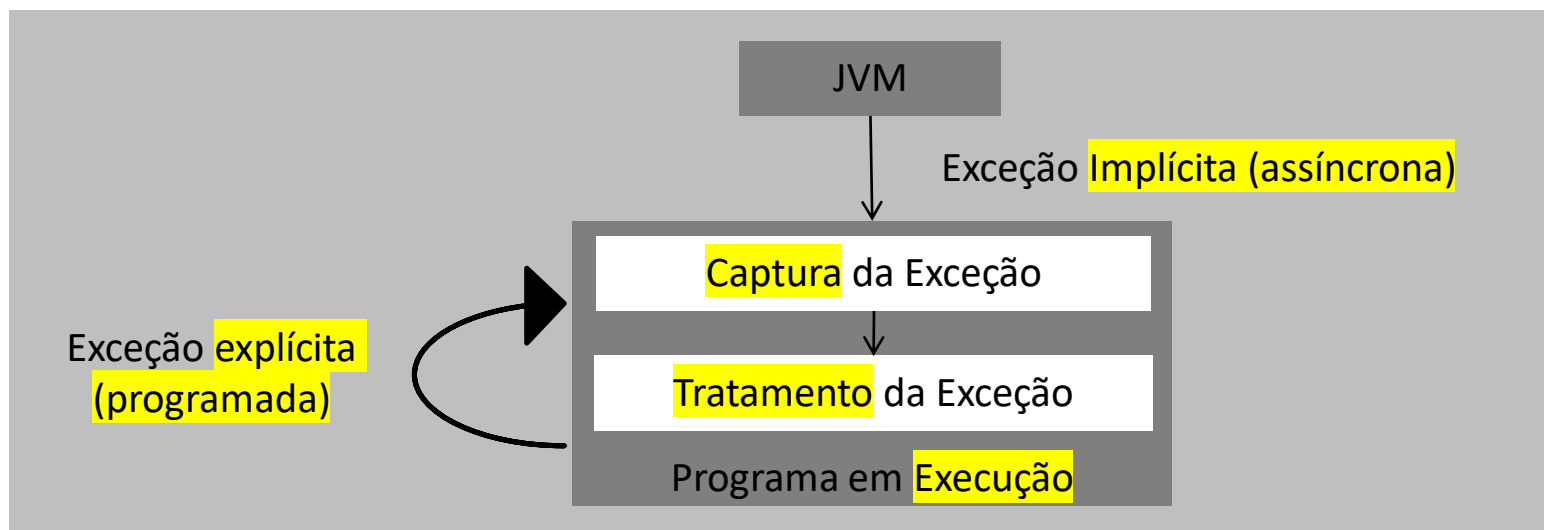


- [Introdução](#)
- [Tipos de Exceções \(Perspetiva do Lançamento\)](#)
 - [Implícitas](#)
 - [Explícitas](#)
- [Interesse](#)
- [Código](#)
 - [Instrução throw](#)
 - [Cláusula throws](#)
- [Procura de Tratamento de Exceção](#)
- [Relançamento de Exceções](#)

- Mecanismo de Exceções **permite** a programa
 - **Captura e Tratamento** de Exceções
 - **Lançamento** de Exceções



- **Na Perspetiva do Lançamento**
 - Exceções Implícitas
 - Exceções Explícitas
- **Exceções Implícitas**
 - Lançadas automaticamente pela JVM
 - Consideradas assíncronas
- **Exceções Explícitas**
 - Lançadas por código definido pelo programador
 - Consideradas programadas




- Permitir a programa provocar eventos de exceção **próprios** ("erros próprios")
 - Para **alertar explicitamente** programa para erro de execução

Exemplo da Classe Circulo

- Constrói objetos **diferentes dos pretendidos** quando dados fornecidos são inválidos
 - Criado circulo de raio 1 quando raio fornecido é nulo ou negativo

Solução indesejável
(círculos diferentes dos pedidos)



```
public class Circulo extends Figura {  
    4     private double raio;  
    5  
    6  
    7     public Circulo(double raio, Cor cor) {  
    8         super(cor);  
    9         setRaio(raio);  
   10     }  
   11  
   12     public void setRaio(double raio) {  
   13         this.raio = raio > 0 ? raio : 1;  
   14     }  
   15  
   16     public String toString() { ... }  
   17  
   18  
   19     public double area() { ... }  
   20  
   21  
   22  
   23 }
```

- Solução desejável**
 - Impedir construção de círculos nessas circunstâncias

- Instrução `throw`
- Cláusula `throws`

- Interesse

- Lançar explicitamente exceção dentro de método // qualquer tipo de exceção

- Sintaxe

throw instância de uma classe de exceções;

- Exemplo

```
public class Circulo extends Figura {  
  
    private double raio;  
  
    public Circulo(double raio, Cor cor) {  
        super(cor);  
        setRaio(raio);  
    }  
  
    public final void setRaio(double raio) {  
        if(raio<=0)  
            throw new IllegalArgumentException("Raio " +raio+ " é inválido!!" );  
        this.raio = raio;  
    }  
  
    public String toString() { ...3 lines }  
  
    public double area() { ...3 lines }  
}
```

define informação
sobre a exceção

- Uso
 - Distinguir os dois tipos de exceções
 - Runtime (unchecked)
 - Não-Runtime (checked)

- Método que lança explicitamente uma exceção checked
 - Obrigado a escolher entre:
 - Tratar localmente essa exceção

```
private static String lerNomes() {  
    try {  
        Scanner ficheiro = new Scanner(new File("c:/nomes.txt"));  
        StringBuilder nomes = new StringBuilder();  
        while (ficheiro.hasNext()) {  
            nomes.append(ficheiro.nextLine());  
        }  
        return nomes.toString();  
    } catch (FileNotFoundException excecacao) {  
        System.out.println("Ficheiro c:\nomes.txt não encontrado!!!");  
        return null;  
    }  
}
```

- Declarar (explicitamente) no cabeçalho (cláusula throws) que pode lançar essa exceção
 - Alerta claramente método invocador para capturar e tratar a exceção

```
private static String lerNomes() throws FileNotFoundException {  
    Scanner ficheiro = new Scanner(new File("c:/nomes.txt"));  
    StringBuilder nomes = new StringBuilder();  
    while (ficheiro.hasNext()) {  
        nomes.append(ficheiro.nextLine());  
    }  
    return nomes.toString();  
}
```

- Método que lança explicitamente uma exceção unchecked
 - Não é obrigado a:
 - Tratar localmente essa exceção
 - Declarar (explicitamente) no cabeçalho (cláusula throws) que pode lançar essa exceção
 - Exemplo: método setRaio

```
public class Circulo extends Figura {  
  
    private double raio;  
  
    public Circulo(double raio, Cor cor) {  
        super(cor);  
        setRaio(raio);  
    }  
  
    public final void setRaio(double raio) {  
        if(raio<=0)  
            throw new IllegalArgumentException("Raio " +raio+ " é inválido!!" );  
        this.raio = raio;  
    }  
  
    public String toString() { ...3 lines }  
  
    public double area() { ...3 lines }  
}
```

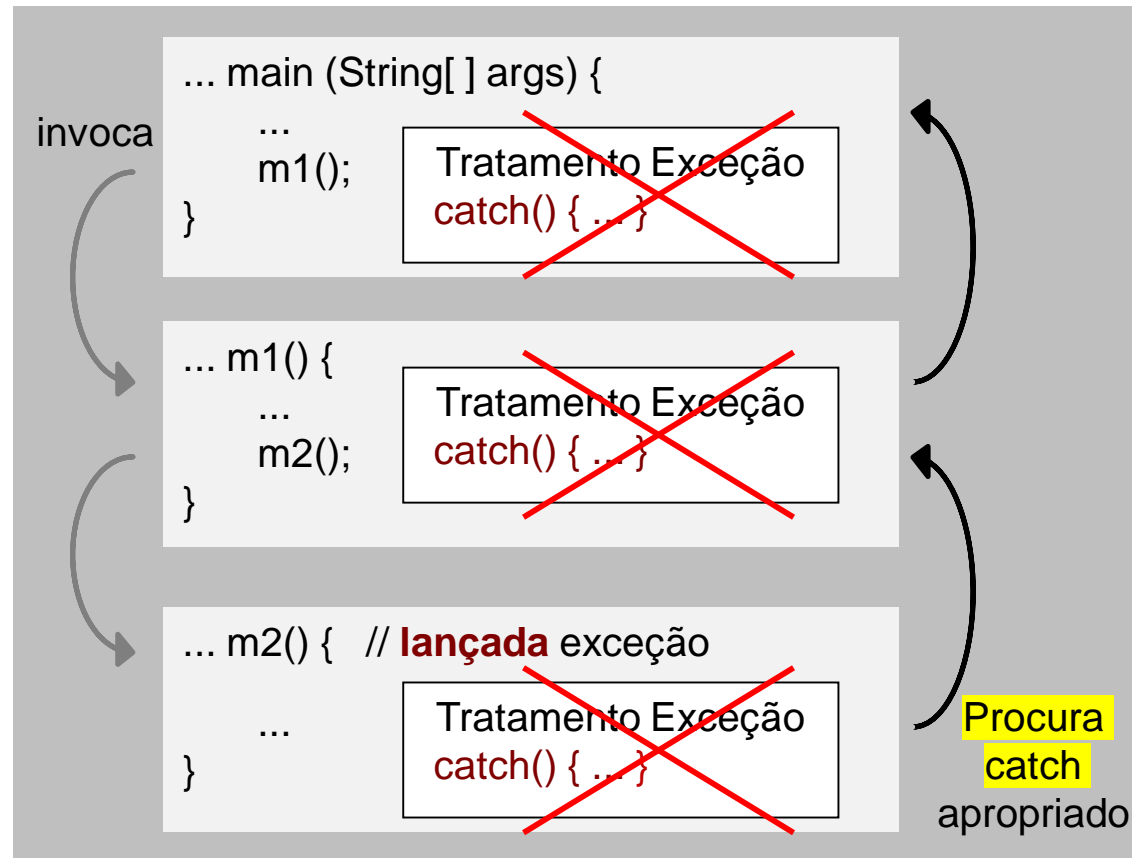

- **Usada**
 - Cabeçalho de método
- **Sintaxe**
... nomeMetodo(lista_parâmetros) throws exceção_1, ..., exceção_N { ... }
- **Declara**
 - Método susceptível de lançar, pelo menos, uma exceção
 - Lançamento pode ser
 - Explícito // através da instrução throw
 - Implícito // através de erro de execução
- **Declarar apenas**
 - Exceções checked e não tratadas localmente // unchecked não é obrigatório tratar

- **Exemplos**

```
public void push(E elem) throws StackFullException, InvalidTypeException { ... }
```

```
public void pop() throws EmptyStackException {  
    if (this.empty)  
        throw new EmptyStackException(); // constrói exceção e depois lança-a  
    else  
        numElem --;  
}
```

- Procura de catch (handler) após ocorrência de uma situação de exceção num método:
 1. Método lança (implícita/ou explícita/) **objeto exceção** contendo **informação** sobre essa exceção.
 2. Objeto exceção é passado à JVM.
 3. **JVM inicia a procura** de um **catch** capaz de **tratar** com essa exceção:
 - Seguindo a **ordem inversa da invocação** dos métodos que originaram a exceção, desde o ponto da ocorrência até encontrar um **catch** ou o método **main**
 - Usada a Stack de Execução
 5. Se catch for encontrado (i.e., se a exceção for capturada)
 - Exceção é tratada
- Senão
- Programa termina com um erro de execução



Exceção não tratada ⇒ Terminação **abrupta** do programa

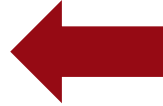
- **Catch** pode:
 - Lançar nova exceção
 - Relançar exceção capturada

- **Exemplo**

```
try {  
    // código onde ocorre exceção  
} catch (tipoExceção e) {  
    ...  
    throw e;                // relança exceção recebida pelo parâmetro e  
}
```

- **Interesse**
 - Quando o **catch** apenas pode realizar o **tratamento parcial** da exceção
 - **Tratamento restante** tem de ser efectuado fora do método que capturou a exceção
- **Muito vulgar**
 - Em **objetos gráficos**

- [Conceitos Básicos](#)
- [Captura e Tratamento de Exceções](#)
- [Lançamento de Exceções](#)
- [Bibliografia](#)



- <http://download.oracle.com/javase/tutorial/essential/exceptions/>
- Livros
 - JAVA5 e Programação por Objetos
 - Fernando Mário Martins
 - FCA (2006)
 - Core Java – Volume I – Fundamentals
 - Cay S. Horstmann and Gary Cornell
 - Sun Microsystems Press (2008)