

Tratamento de Exceções

- O que são Exceções?



São erros que podem ocorrer durante a execução de um programa, causados por situações inesperadas, que rompem o fluxo normal de execução do programa.

Tipos de Exceções (apenas alguns exemplos)

Exceções e Descrição
ArithmeticException Erro aritmético, por exemplo divisão por zero
ArrayIndexOutOfBoundsException Índice do array fora dos limites.
ArrayStoreException Atribuição a um elemento do array de um tipo incompatível
ClassCastException Cast inválido
IllegalArgumentException Argumento ilegal usado na invocação do método
IndexOutOfBoundsException Algum tipo de índice está fora dos limites.
NegativeArraySizeException Array criado com tamanho negativo
NullPointerException Uso inválido de uma referência null
NumberFormatException

Tipos de Exceções em JAVA

Checked

- São verificadas em tempo de compilação
- Se dentro de um método existe código que lança uma exceção, então o compilador obriga a tratar essa exceção, ou a usar a cláusula throws

Unchecked

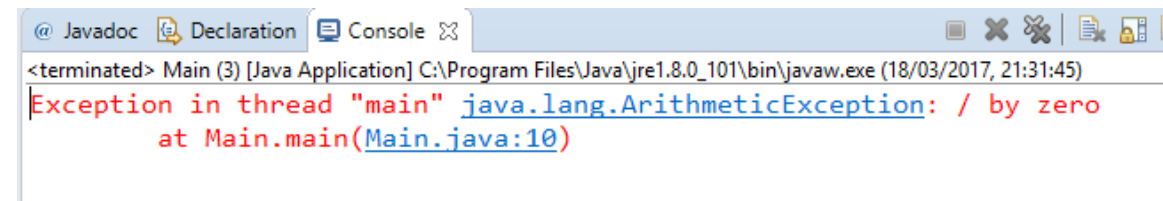
- Não são verificadas em tempo de compilação
- Tudo o que está debaixo das classes RuntimeException e Error são unchecked exceptions

Tratamento de Exceções

- Exemplo

```
public class Main {  
  
    public static void main(String[] args) {  
  
        int n1 = 4;  
        int n2 = 0;  
        int r;  
  
        r = n1/n2;  
  
        System.out.println("Resultado: "+ r);  
    }  
}
```




O programa termina de uma forma abrupta!



Como evitar?

R: Através de um correto tratamento das exceções de forma a tornar o programa robusto.

Tratamento de Exceções: Palavras Reservadas

- try
 - catch
 - finally
- 
- Define um bloco de tratamento de uma exceção
- throws  Declara que um metodo pode lançar uma ou mais exceções
 - throw  Lança uma exceção

Bloco de Tratamento de uma Exceção

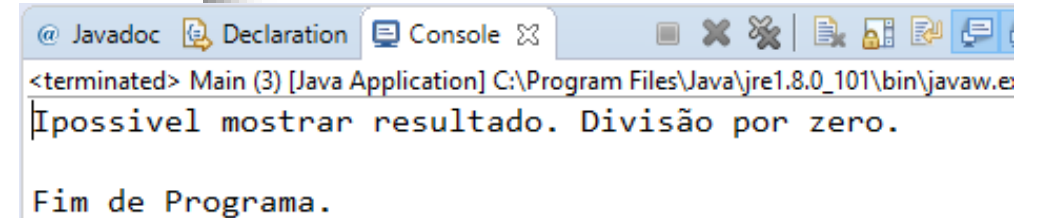
Sintaxe

```
try {  
    ...  
} catch (Excecao1 e1) {  
    ...  
} catch (Excecao2 e2) {  
    ...  
} finally {  
    ...  
}
```

Bloco de Tratamento de uma Exceção

Exemplo:

```
public class Main {  
  
    public static void main(String[] args) {  
  
        int n1 = 4;  
        int n2 = 0;  
        int r = 0;  
  
        try{  
            r = n1/n2;  
            System.out.println("Resultado: "+ r);  
        } catch (ArithmeticException e){  
            System.out.println("Ipossivel mostrar resultado. Divisão por zero.\n");  
        }  
  
        System.out.println("Fim de Programa.\n");  
    }  
}
```



```
<terminated> Main (3) [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.e  
Ipossivel mostrar resultado. Divisão por zero.  
  
Fim de Programa.
```

Bloco de Tratamento de uma Exceção

Uso do **finally**

O código que se colocar dentro do **finally**, executa sempre (quer seja gerada exceção ou não)

```
import java.io.FileWriter;
import java.io.IOException;
public class GravaArquivo2 {
    public static void main(String args[ ]) {
        FileWriter fw = null;
        try {
            fw = new FileWriter("teste.txt");
            fw.write(args[0]);
        } catch (IOException e) {
            System.out.println("Erro ao gravar arquivo!");
            System.out.println(e); // Imprime detalhes da Exceção.
        } finally {
            try{
                fw.close();
            }catch(IOException e){
                System.out.println("Erro ao fechar arquivo!");
                System.out.println(e); // Imprime detalhes da Exceção.
            }
        }
    }
}
```


Throws

- Declara que um método pode lançar uma ou mais exceções
- Um método pode lançar uma exceção se encontrar uma situação com a qual não sabe lidar
- Um metodo informa o compilador dos parametros que receber, do valor que retorna, e também dos erros que podem suceder durante a sua execução, através do **throws**.

Throws

Sintaxe

```
public void metodo( ) throws Excecao {  
    ...  
}
```

```
public void metodo( ) throws Excecao1, Excecao2 {  
    ...  
}
```

Throws

Exemplo

```
import java.io.FileWriter;

public class GravaArquivo {
    public static void grava(String texto) {
        FileWriter fw = new FileWriter("teste.txt");
        fw.write(texto);
        fw.close( );
    }
}
```



O compilador dá erro, porque exige que se declare a exceção **IOException** na clausula throws do método ou que a mesma seja tratada dentro do método.

Throws

Exemplo

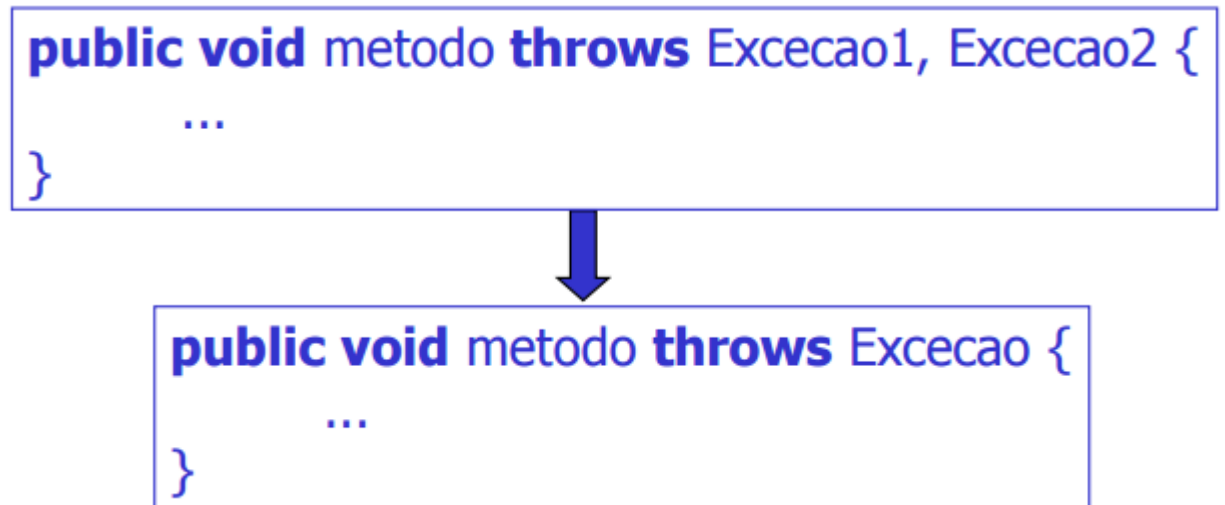
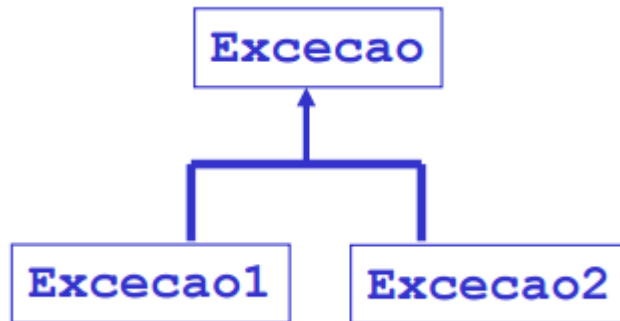
```
import java.io.FileWriter;  
  
public class GravaArquivo {  
    public static void grava(String texto) throws IOException {  
        FileWriter fw = new FileWriter("teste.txt");  
        fw.write(texto);  
        fw.close( );  
    }  
}
```



Neste caso o compilador já não dá erro, porque se está a usar a clausula throws

Lançar e Tratar Exceções pela SuperClasse

As exceções declaradas com throws podem ser superclasses das exceções realmente lançadas



Lançar e Tratar Exceções pela SuperClasse

ArrayIndexOutOfBoundsException e ArithmeticException são subclasses de Exception.

Tratamento pelas Subclasses

```
Scanner input = new Scanner(System.in);
int n1;
int n2;
System.out.println("Entre o numerador:\n");
n1 = input.nextInt();

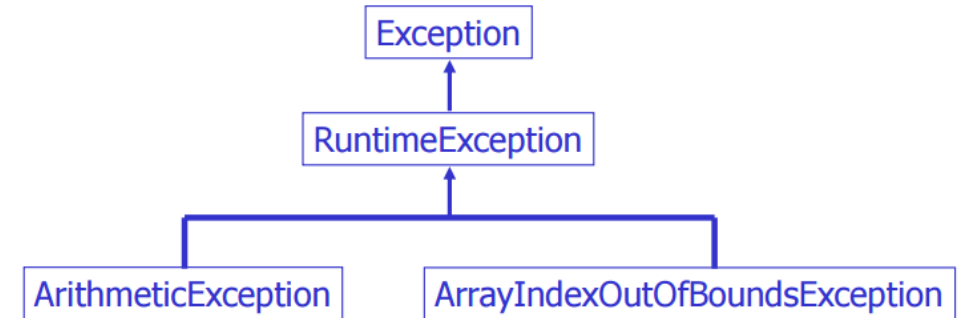
System.out.println("Entre o denominador:\n");
n2 = input.nextInt();

int array[] = new int[2];

try{
    array[0] = n1;
    array[1] = n2;
    array[3] = n1/n2;

    System.out.println("Resultado: "+ array[3]);
} catch (ArithmeticException e){
    System.out.println("Impossível mostrar resultado. Divisão por zero.\n");
}
catch (ArrayIndexOutOfBoundsException e){
    System.out.println("Erro. Índice de array inválido.\n");
}
```

Formadora: Carla Macedo



Tratamento pelas Superclasse

```
Scanner input = new Scanner(System.in);
int n1;
int n2;
System.out.println("Entre o numerador:\n");
n1 = input.nextInt();

System.out.println("Entre o denominador:\n");
n2 = input.nextInt();

int array[] = new int[2];

try{
    array[0] = n1;
    array[1] = n2;
    array[3] = n1/n2;

    System.out.println("Resultado: "+ array[3]);
} catch (Exception e){
    System.out.println("Erro ao executar o programa.\n");
}
```

throw

- **throw** é a palavra usada para lançar uma exceção.

Exemplo 1:

```
Exception e = new Exception("Mensagem de Erro!");  
throw e;
```

Exemplo 2:

```
throw new Exception("Mensagem de Erro!");
```

Exceções definidas pelo programador

- Um programa pode gerar um problema que não esteja descrito apropriadamente em nenhuma das classes de exceções
- O Java permite ao programador definir novas exceções como subclasses da classe Exception

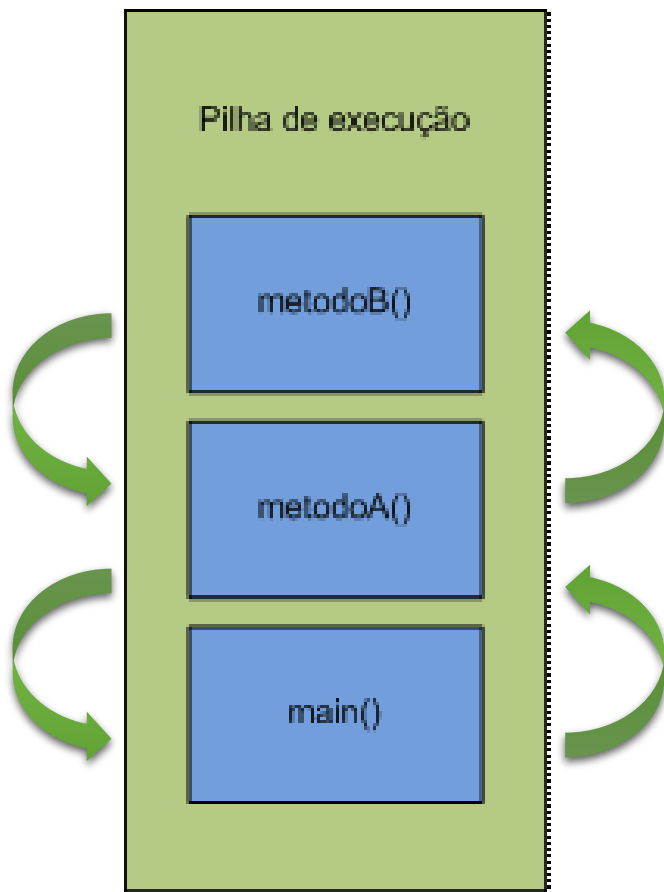
```
public class ExcecaoTextoInvalido extends Exception {  
    public ExcecaoTextoInvalido(String mensagem) {  
        super(mensagem);  
    }  
}
```


Exceções definidas pelo programador

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

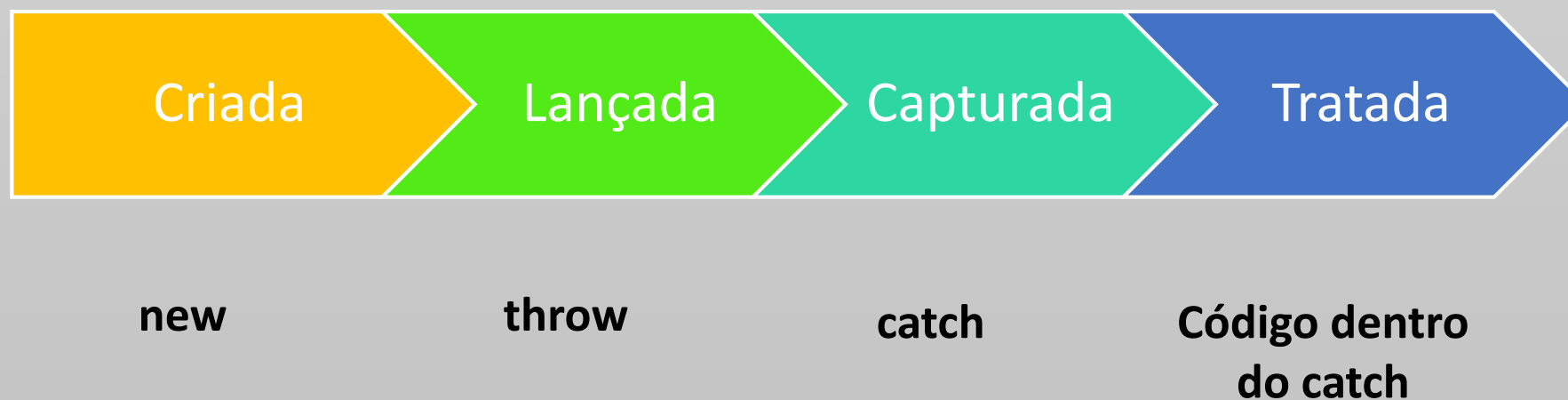
```
public static void grava(String texto) throws IOException, ExcecaoTextoInvalido {  
    FileWriter fw = null;  
    if (texto == null || texto.trim().equals("")) {  
        throw new ExcecaoTextoInvalido("Texto inválido: " + texto);  
    } else {  
        fw = new FileWriter("teste.txt");  
        fw.write(texto);  
    }  
}
```

Pilha de Invocações (Stack)



1. O método `main()` inicia e é colocado na Pilha
2. O método `main()` invoca o `metodoA()`. A sua execução é suspendida e o `metodoA()` é adicionado ao cimo da pilha
3. O `metodoA()` invoca o `metodoB()`. A sua execução é suspendida e o `metodoB()` é adicionado ao cimo da pilha
4. O `metodoB()` termina, e a execução retorna ao `metodoA()`. O `metodoB()` é eliminado do cimo da Pilha
5. O `metodoA()` termina, e a execução retorna ao `main()`. O `metodoA()` é eliminado do cimo da Pilha
6. O método `main()` termina é removido da pilha e programa finaliza.

Ciclo de Vida das Exceções

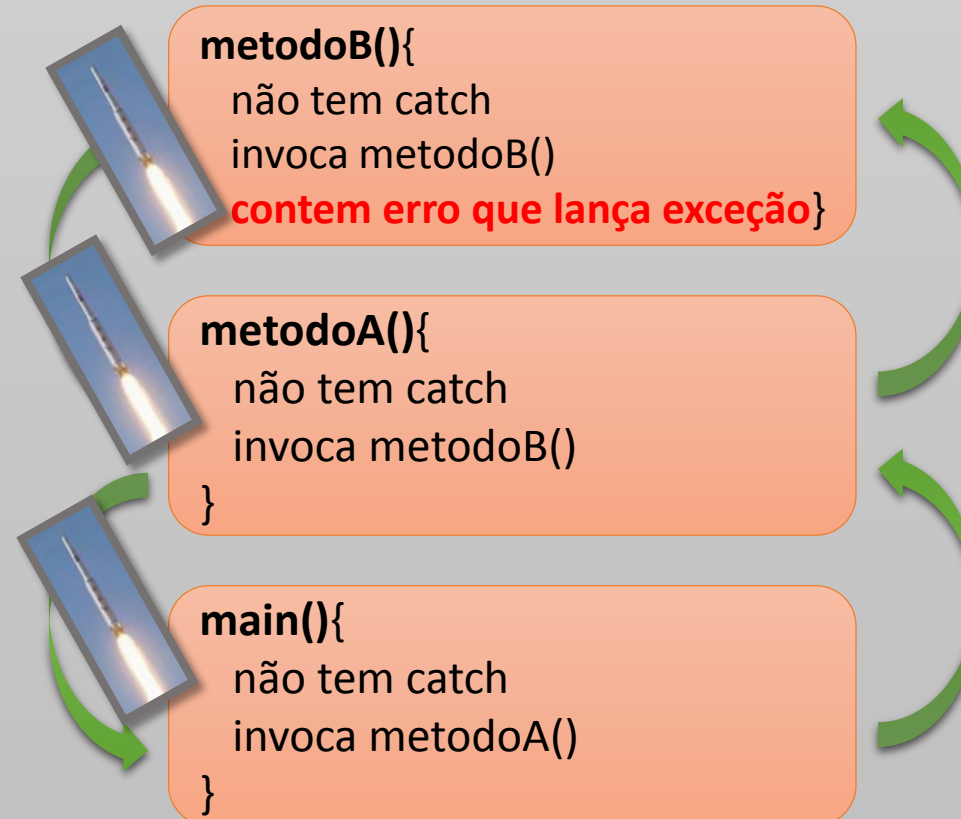


Propagação das Exceções



Toda a exceção
não tratada na
cláusula ***catch*** é
propagada na
pilha de execução

Exemplo 1

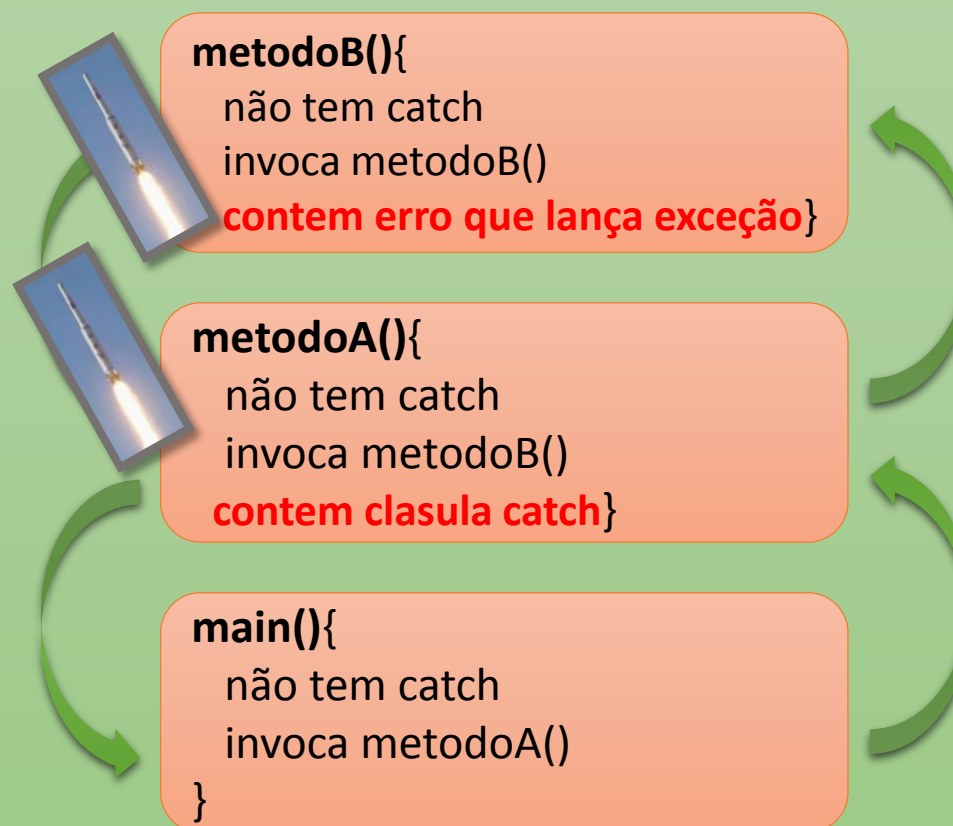


Propagação das Exceções



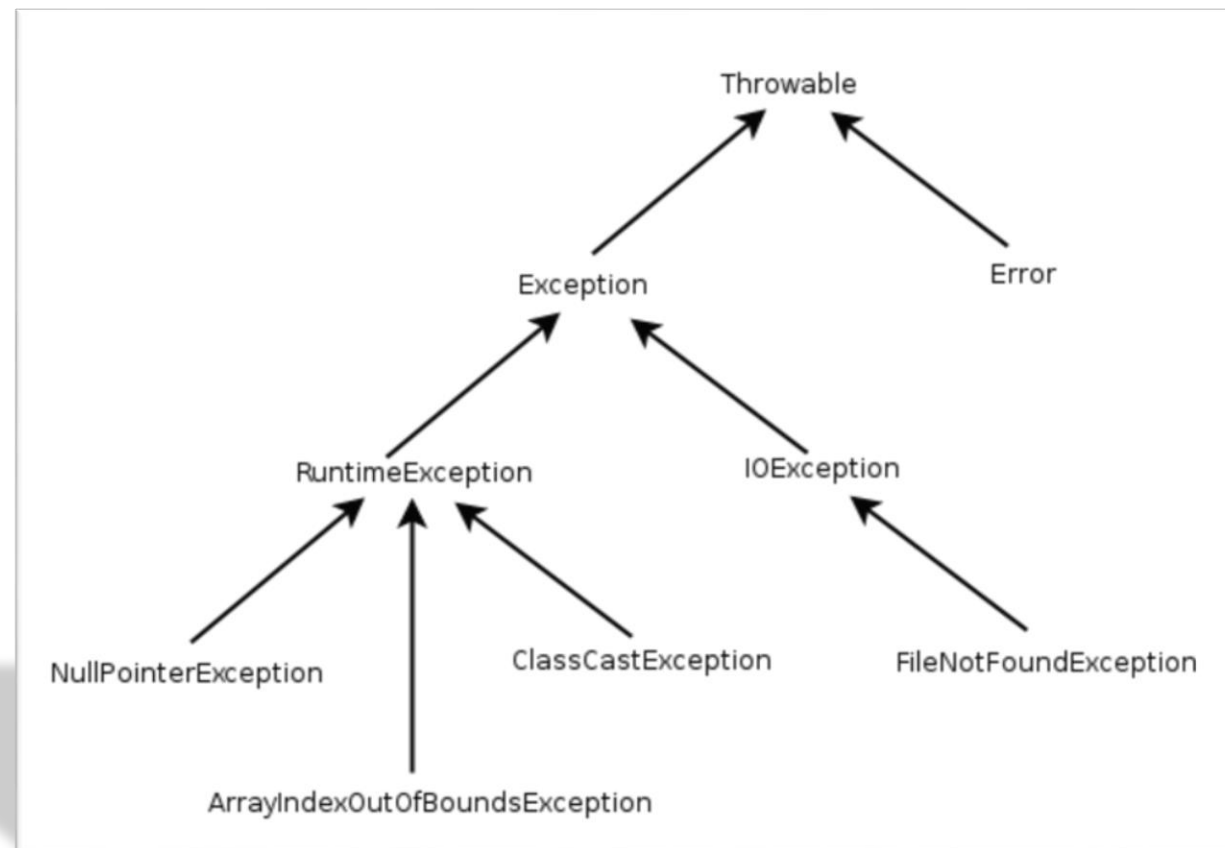
Toda a exceção
não tratada na
cláusula ***catch*** é
propagada na
pilha de execução

Exemplo 2



Hierarquia de Exceções em JAVA

Exemplos de algumas das mais significativas



Throwable Classe

Métodos

- **printStackTrace()**
Imprime o rastro da Pilha de Execução
- **getStackTrace ()**
Obtem informação sobre o rastro da pilha de execução
- **getMessage()**
Retorna uma string que corresponde à mensagem da exceção

Tratamento de Exceções – Boa Pratica

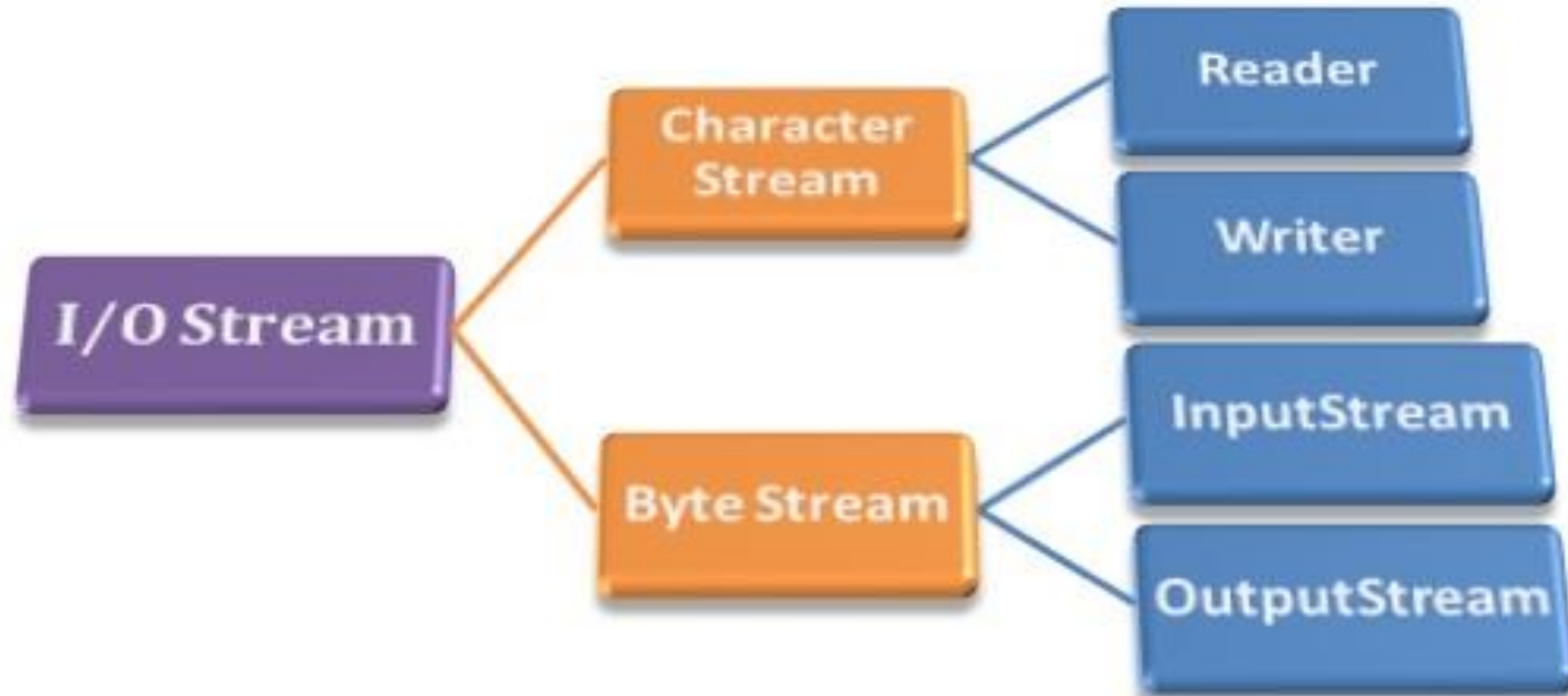
Regra Tratar ou Declarar



Todo método deve tratar todas as exceções verificadas fornecendo uma cláusula catch, ou então listar cada exceção verificada que não tiver recebido tratamento como uma exceção lançada (throws)

FIM DO TRAMENTO DE EXCEÇÕES

Input/Output



O que é uma *Stream*?



É um objeto que faz uma de duas acções: ou entrega dados a um destino (ecrã, ficheiro, etc), ou recebe dados de uma origem (teclado, ficheiro, etc).

Uma stream atua como um buffer entre a origem dos dados e o seu destino.

Input Stream

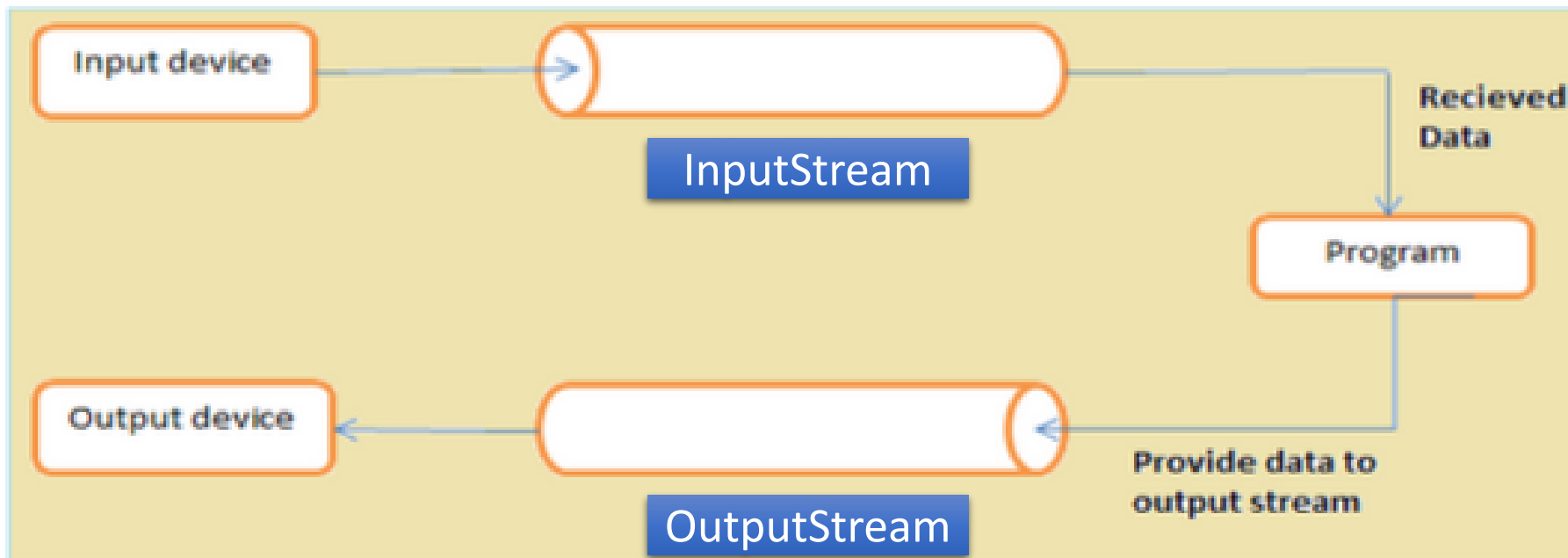
É uma stream que providencia input a um programa.

Exemplo: **System.in**

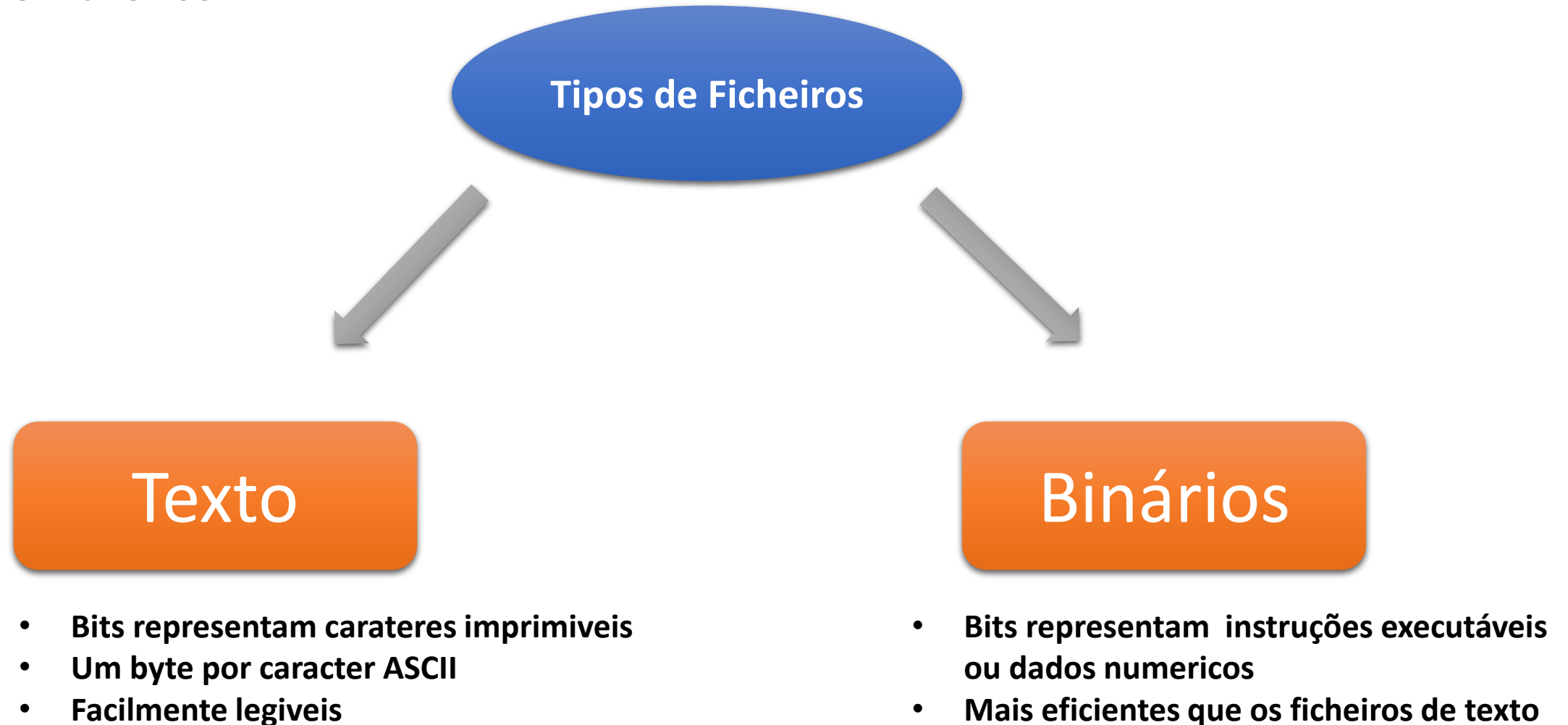
Output Stream

É uma stream que recebe output de um programa

Exemplo: **System.out**



Tipos de Ficheiros



Ficheiros

Todos os ficheiros têm dois nomes:

1. O nome da *stream* usado pelo Java

2. O nome do ficheiro usado pelo Sistema Operativo

Classes para Acesso a Ficheiros de Texto

- **Scanner**
 - Para leitura
 - Estabelece fluxo (interno) de entrada de ficheiro
- **PrintWriter**
 - Para escrita
 - Interface semelhante à System.out
 - Estabelece fluxo (interno) de saída para ficheiro
- **File**
 - Representa ficheiros

Exceções de Entrada com Scanner

IOException	Descrição
FileNotFoundException	Tentativa de estabelecimento de fluxo de entrada de um ficheiro inexistente

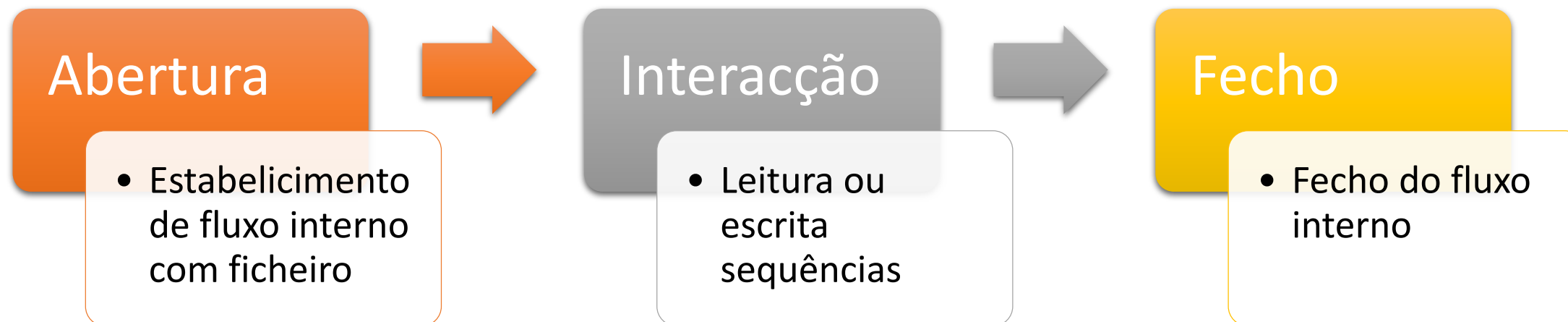
RuntimeException	Descrição
InputMismatchException	Tentativa de leitura de valor de tipo incompatível com conteúdo do ficheiro (e.g., int quando ficheiro contém letras)
NoSuchElementException	Tentativa de leitura quando o fluxo de entrada está esgotado
IllegalStateException	Tentativa de leitura quando o fluxo de entrada está fechado

Exceções de Saída com PrintWriter

IOException	Descrição
FileNotFoundException	Tentativa falhada de estabelecimento de fluxo de saída para ficheiro

RuntimeException	Descrição
NADA	

Sequência de Acesso a um Ficheiro



Exemplo de leitura: abertura

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import static java.lang.System.out;

...
try {
    final Scanner fileScanner = new Scanner(new File("My File.txt"));
    ...
} catch (final FileNotFoundException exception) {
    out.println("Ficheiro não encontrado!");
    ...
} ...
```

Exemplo de leitura: leitura e fecho

```
...
try {
    if (fileScanner.hasNextInt()) {
        final int numberOfCars = fileScanner.nextInt();
        ...
    }
    else {
        out.println("Ops! Não era suposto chegar aqui...");
    }
} finally {
    fileScanner.close();
}
```

Exemplo de escrita: abertura

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import static java.lang.System.out;

...
try {
    final PrintWriter fileWriter = new PrintWriter(new File("My new file.txt"));
    ...
} catch (final FileNotFoundException exception) {
    out.println("Error creating file. Sorry!");
    ...
}
...
```

Exemplo de escrita: escrita e fecho

```
...
try {
    fileWriter.println(20);
    ...
    if (fileWriter.checkError())
        out.println("Error writing to file.");
    } finally {
        fileWriter.close();
    }
    ...
```