Técnicas Avançadas de Programação

Desenvolvimento para a *Web* e Dispositivos Móveis

1º Ano, 2º Semestre

Joana Fialho
E-mail: jfialho@estgv.ipv.pt
Nuno Costa
E-mail: ncosta@estgv.ipv.pt
Carlos Simões
E-mail: csimoes@estgv.ipv.pt

Escola Superior de Tecnologia e Gestão de Viseu
2019-2020

Exceções

- Uma exceção é um erro que pode acontecer em tempo de processamento de um programa. Existem exceções causadas por erro de lógica e outras provenientes de situações inesperadas, como por exemplo um problema no acesso a um dispositivo externo, como uma controladora de disco, uma placa de rede, etc.

- Muitas destas exceções podem ser previstas pelo programador, e tratadas por meio de alguma rotina especial, para evitar que o programa seja cancelado de forma anormal.

- Dentro dos pacotes do Java existem alguns tipos de exceções já previamente definidas por meio de classes específicas. Estas exceções, pré estabelecidas, fazem parte dos pacotes do Java, e todas elas são sub-classes da classe ***Throwable***, que pertence ao pacote ***java.lang***.

- A detecção e controle de exceções, dentro de um programa Java, deve ser feita pelo programador através da estrutura ***try-catch-finally***, conforme descrito e mostrado mais à frente.

Exceções

Tratamento de Exceções

O tratamento de exceções Java é feito usando cinco palavras reservadas:

try , ***catch*** , ***throw*** , ***throws*** e ***finally***.

De forma resumida, o funcionamento é o seguinte:

As instruções do programa cujas exceções se pretendem monitorar são colocadas num bloco ***try***. Se uma exceção ocorrer dentro do bloco ***try***, ela será lançada. O código poderá capturar essa exceção usando ***catch*** e tratá-la de alguma forma. Exceções geradas pelo sistema são lançadas automaticamente pelo sistema de tempo de execução Java. Para lançar manualmente uma exceção é usada a palavra ***throw***. Em alguns casos, uma exceção que é lançada para fora de um método deve ser especificada como tal por uma diretiva ***throws***. Qualquer código que deva ser executado ao sair de um bloco ***try*** deve ser colocado num bloco ***finally***.

- Qualquer método cujo processamento possa provocar algum tipo de exceção indica esta condição por meio da palavra reservada ***throws***, como no exemplo a seguir:

public void writeInt(int valor) throws IOException

Exceções

```
try
{
    bloco de comandos que podem causar exceções
}
catch (ClasseDeExceção-1   erro1)
{
    Comandos para tratamento do erro ocorrido
}
catch (ClasseDeExceção-2  erro2)
{
    Comandos para tratamento do erro ocorrido
}
finally
{
    Comandos que devem ser executados tanto nas condições de processamento normal, como nas condições de erro
}
    Demais linhas de código do método
```

Se os comandos forem executados com sucesso, o fluxo de execução será o seguinte:

- execução dos comandos do bloco **try**
- execução do bloco **finally** (se existir)
- execução normal do resto do programa

Se ocorrer um erro1, o fluxo de execução será o seguinte:

- execução deste bloco **catch**
- execução do bloco **finally** (se existir)
- execução normal do resto do programa

Se ocorrer um erro-2, o fluxo de execução será o seguinte:

- execução deste bloco **catch**
- execução do bloco **finally** (se existir)
- execução normal do resto do programa

O bloco finally é opcional

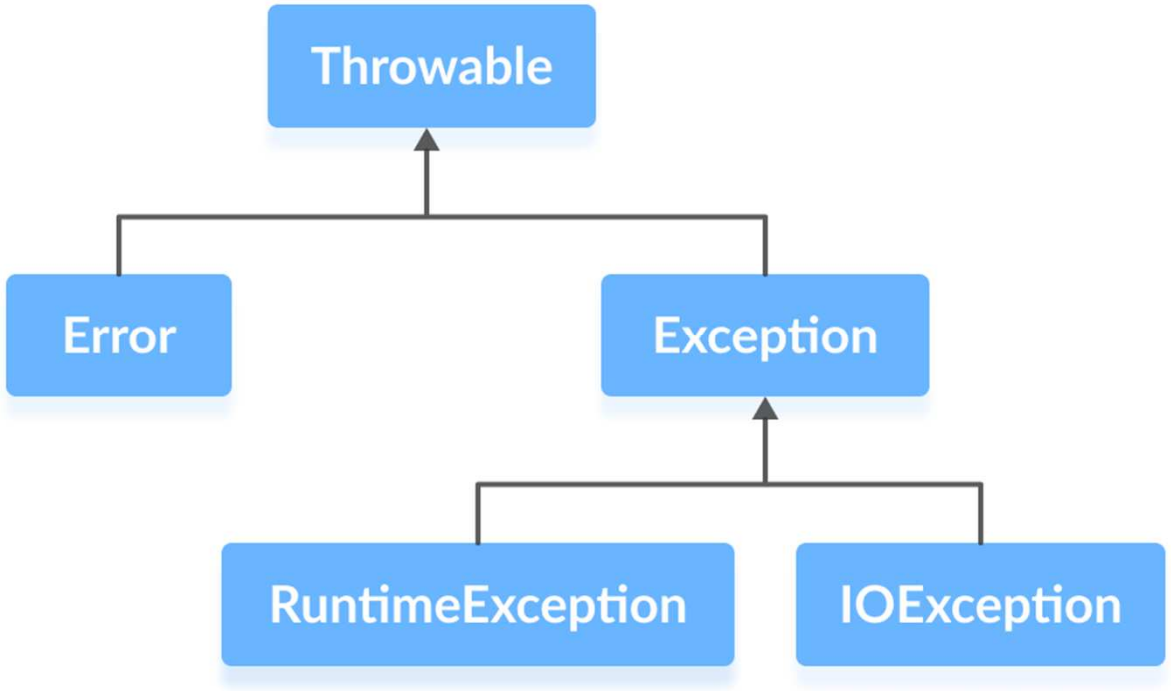
Exceções

Hierarquia de Exceções

Em Java, todas as exceções são representadas por classes e todas as classes de exceções são derivadas de uma classe chamada ***Throwable***. Logo, quando uma exceção ocorre em um programa, um objeto de algum tipo de classe de exceção é gerado. Há duas subclasses diretas de ***Throwable***: ***Exception*** e ***Error***. As exceções de tipo ***Error*** estão relacionadas a erros que não podemos controlar, como os que ocorrem na própria máquina virtual Java. Geralmente os programas não lidam com eles. Portanto, esses tipos de exceções não serão descritos aqui. Erros que resultam da atividade do programa são representados por subclasses de ***Exception***. Por exemplo, erros de divisão por zero, que excedem os limites do array e de I/O se enquadram nessa categoria. Em geral, os programas devem tratar exceções desses tipos. Uma subclasse importante de ***Exception*** é ***RuntimeException***, que é usada para representar vários tipos comuns de erros de tempo de execução.

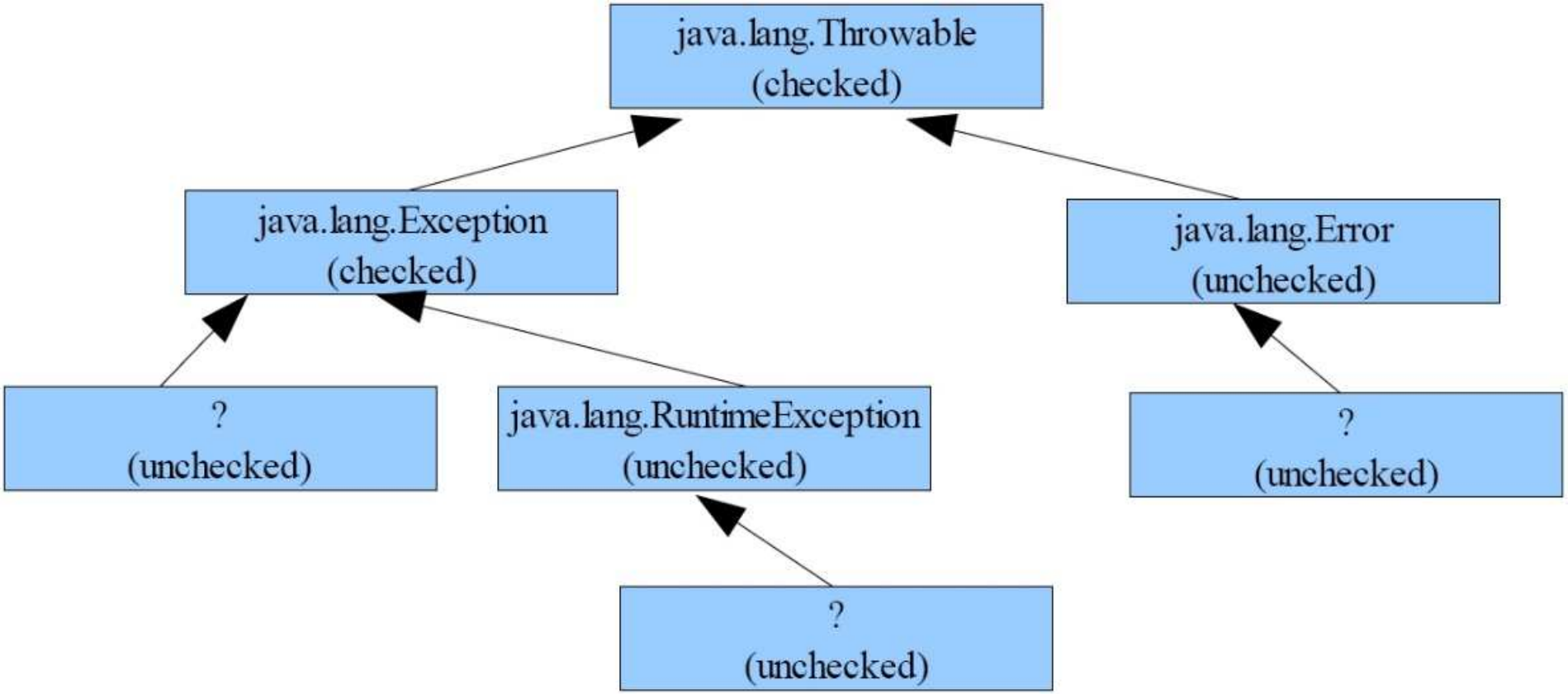
Exceções


Unchecked Exception
Checked Exception
Error
Exception Handling

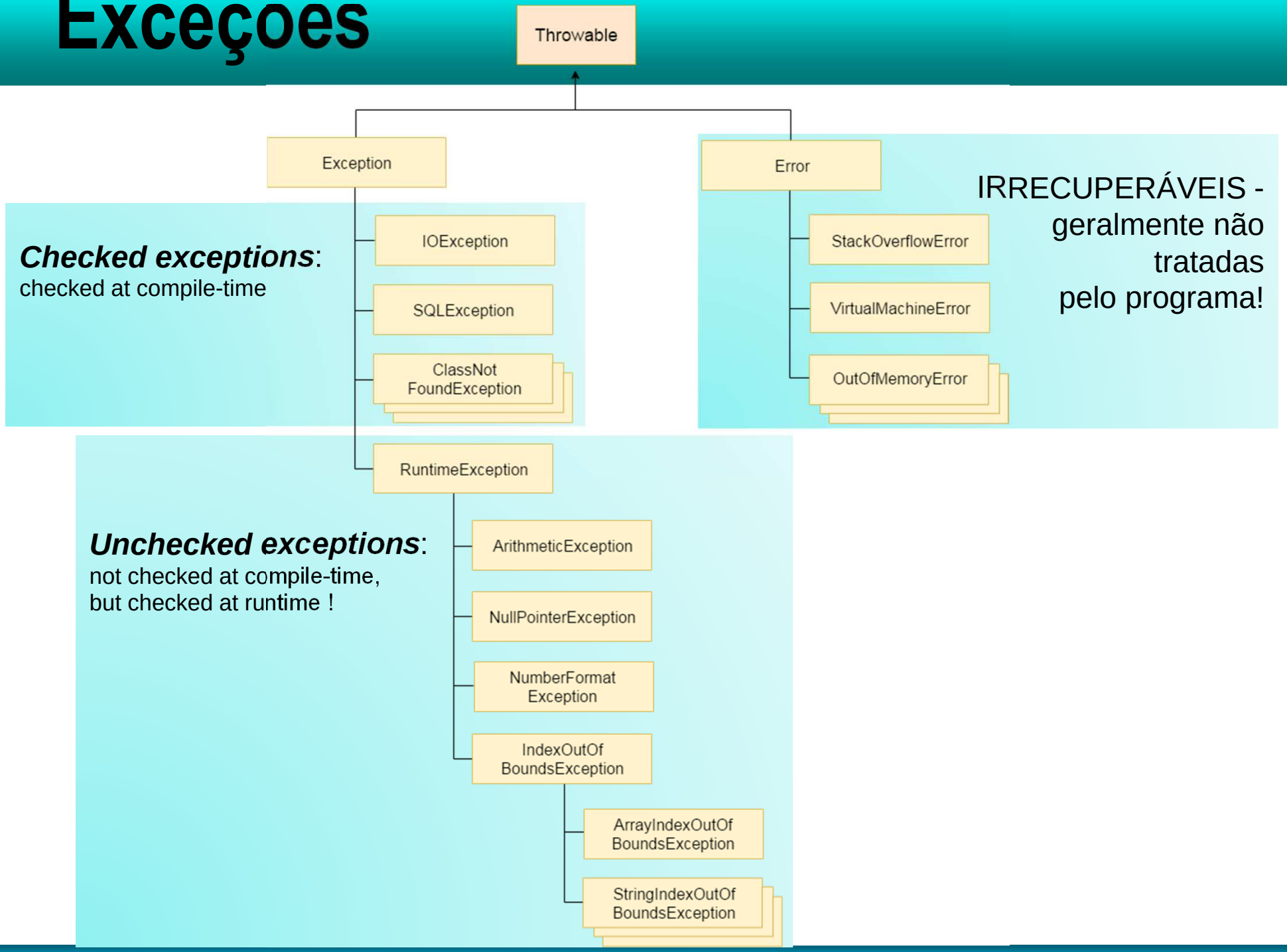

Throwable
Error
Exception
RuntimeException
IOException

Exceções

–


java.lang.Throwable
(checked)
java.lang.Exception
(checked)
java.lang.Error
(unchecked)
?
(unchecked)
java.lang.RuntimeException
(unchecked)
?
(unchecked)
?
(unchecked)

Exceções



Checked exceptions:
checked at compile-time

IRRECUPERÁVEIS - geralmente não tratadas pelo programa!

Unchecked exceptions:
not checked at compile-time,
but checked at runtime !

Exceções

Exemplo 1

```
public class run {
    public void metodo(int i) throws Exception // Declaração da exceção checked
    {
        if (i>3)
            throw new Exception("ficheiro não encontrado."); // lançamento
    }                           // da exceção, devido a uma situação anómala
    public static void main(String[] args) {
        run r = new run();
        try {
            r.metodo(4);
        }
        catch (Exception e) // captura da exceção que pode ser lançada por r.metodo()
        {
            System.out.println("erro: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Exceções

Exemplo 2 – As exceções *Runtime* pode ou não ser tratadas

```
public class run {
    public void metodo(int i)
    {
        if (i>3)
            throw new RuntimeException("índice fora dos limites.");
    }
    public static void main(String[] args) {
        run r = new run();
        r.metodo(4); // não é obrigatório que seja capturada a exceção
        try {            // as exceções Runtime podem ser capturadas
            r.metodo(6); // mas tal não é obrigatório
        }
        catch (RuntimeException e) {
            System.out.print("RuntimeException:" + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Exceções

Exemplo 3 – *finally*

```
i = 2;
System.out.println("antes do try-catch-finally");  // é executado
try {
    System.out.println("try 1");
    if (i>1)
        throw new FileNotFoundException("ficheiro não encontrado");
    System.out.println("try 2");  // NÃO É EXECUTADO
}
catch (FileNotFoundException e) {
    System.out.println("dentro FileNotFoundException");  // é executado
}
catch (Exception e)
{
    System.out.println("Dentro Exception"); // NÃO É EXECUTADO
}
finally
{
    System.out.println("dentro do finally");  // é executado
}
System.out.println("depois do try-catch-finally");  // é executado
```

Exceções

Exemplo 4 – exceção lançada num *catch*

```
i = 2;
System.out.println("antes do try-catch-finally"); // é executado
try {
    System.out.println("try 1");
    if (i>1)
        throw new FileNotFoundException("ficheiro não encontrado");
    System.out.println("try 2"); // NÃO É EXECUTADO
}
catch (FileNotFoundException e) {
    System.out.println("dentro FileNotFoundException"); // é executado
                           // um catch também pode lançar uma exceção
    if (i>1)
        throw new RuntimeException("Exceção lançada num catch");
    System.out.println("dentro FileNotFoundException 2"); // NÃO É
}                                                         // EXECUTADO
catch (Exception e)
{
    System.out.println("Dentro Exception"); // NÃO É EXECUTADO
}
Finally
{
    System.out.println("dentro do finally"); // é executado
}
System.out.println("depois do try-catch-finally"); // ==> NÃO é executado porque
                                    // foi lançada uma exceção dentro do bloco
                                    // catch e esta não foi ainda  // capturada
```

Exceções

Exemplo 5 – exceção lançada num *finally*

```
i = 2;
System.out.println("antes do try-catch-finally"); // é executado
try {
    System.out.println("try 1");
    if (i>1)
        throw new FileNotFoundException("ficheiro não encontrado");
    System.out.println("try 2"); // NÃO É EXECUTADO
}
catch (FileNotFoundException e) {
    System.out.println("dentro FileNotFoundException"); // é executado um catch
                                                        //  também pode lançar uma exceção
    if (i>1)
        throw new RuntimeException("Exceção lançada num catch");
    System.out.println("dentro FileNotFoundException 2"); // NÃO É EXECUTADO
}
catch (Exception e)
{
    System.out.println("Dentro Exception"); // NÃO É EXECUTADO
}
finally
{
    System.out.println("dentro do finally"); // é executado
    if (i>1)
        throw new RuntimeException("Exceção lançada num finally"); // ==> esta
                                                                   // exceção substitui a exceção do catch
    System.out.println("dentro do finally 2"); // Não é executado
}
System.out.println("depois do try-catch-finally"); // ==> NÃO é executado porque
                                                  // foi lançada a exceção dentro do finally
```

Exceções

Exemplo _