

DEI - ISEP

# **Tratamento de Erros com Exceções**

Fernando Mouta



## Índice

1	Introdução .....	1
2	Benefícios das Exceções .....	1
3	Lançamento de uma Exceção .....	1
4	Bloco try - Região Guardada .....	2
5	Lista de Especificação das Exceções .....	3
6	Criação de Exceções .....	4
7	Exceções em Subclasses .....	9
8	Cláusula <i>finally</i> .....	9
9	Cláusula <i>catch</i> que Captura uma Exceção Lançada .....	11
10	Relançamento de uma Exceção ( <i>Rethrowing an Exception</i> ) .....	12
11	Exceções <i>RuntimeException</i> .....	13
12	Exemplos .....	14



## 1 Introdução

Idealmente os erros de um programa deviam ser apanhados em tempo de compilação porque código errado não deveria ser executado. Mas como nem todos os erros podem ser detetados em tempo de compilação, deve existir um formalismo de tratamento de erros em tempo de execução.

Em muitas linguagens o formalismo consiste no retorno de um valor especial ou na colocação de uma *flag* pela função na qual se verificou o erro, e o programa que chamou a função devia testar o valor retornado ou a *flag*, e determinar qual o problema.

Mas, a maior parte das vezes, os programas não testam as condições de erro. Se testassem todos os potenciais erros sempre que uma função era chamada o código tornar-se-ia ilegível.

## 2 Benefícios das Exceções

No ponto do programa onde ocorre um problema, conhecem-se as características do problema que ocorreu, mas em geral, nesse contexto (contexto corrente) não se tem informação suficiente para lidar com o problema. Num contexto mais alto, noutra parte do programa, existe a informação necessária para decidir o que fazer.

Outro benefício das exceções resulta num programa limpo de código de tratamento de erros. Não é necessário testar a ocorrência de um erro particular e tratá-lo em vários sítios de um programa. A exceção garante que o erro é tratado.

Deste modo o problema só é tratado num sítio. Além de se poupar código, separa-se código com o que se pretende fazer, do código de tratamento de erros.

## 3 Lançamento de uma Exceção

Uma exceção (condição excecional) é um problema que não permite a continuação do método ou *scope* em execução. Quando ocorre uma exceção não se pode continuar o processamento corrente porque não se tem informação necessária para tratar do problema no contexto corrente e relega-se o problema para um contexto mais alto, lançando uma exceção.

Quando se lança uma exceção (usa-se a palavra-chave **throw**), um objeto exceção é criado (através da chamado de um construtor da respectiva classe) e sai-se do método ou do scope corrente, sendo retornada a referência ao objeto exceção criado. O fluxo de execução corrente é parado e passa do contexto corrente para o local de tratamento da exceção para onde também é passada a referência do objeto exceção.

Exemplo de lançamento de uma exceção:

```
if (t == null) throw new NullPointerException();
```

Em todas as exceções *standard* (classes) há 2 construtores:

- um sem argumentos, e
- outro com 1 argumento do tipo *String*.

Normalmente lança-se uma exceção de uma classe diferente para cada tipo diferente de erro, para que no contexto mais amplo se possa determinar o que fazer (tomar a decisão apropriada) apenas através do conhecimento do tipo de objeto exceção.

Quando uma exceção é lançada o mecanismo de tratamento de exceções toma conta do controlo do fluxo do programa e termina a execução do método no qual a exceção é lançada, assim como os métodos dos quais este método foi chamado e a execução continua numa parte do programa que se destina a tratar exceções daquele tipo (*exception handler*).

## 4 Bloco try - Região Guardada

Para que *throw* não cause a saída de um método, deve existir dentro desse método um bloco para capturar a exceção.

Uma exceção lançada é capturada imediatamente a seguir a um *bloco try* por cláusulas denotadas pela palavra-chave ***catch***, designadas cláusulas *catch* ou cláusulas de tratamento de exceções (*exception handlers*).

Cada cláusula *catch* toma 1 e só 1 argumento de um tipo particular.

```
try {  
    // código que pode gerar exceções  
} catch ( Type1 id1 ) {  
    // tratamento de exceções do tipo Type1  
} catch ( Type2 id2 ) {  
    // tratamento de exceções do tipo Type2  
}
```

Normalmente o tratamento da exceção é baseado no tipo da exceção sem necessidade de usar o identificador, mas em qualquer dos casos o identificador tem de ser declarado.

As cláusulas *catch* devem aparecer diretamente depois do *bloco try*, tendo cada cláusula sempre 1 só argumento.

Este tratamento de exceções separa o código colocado num *bloco try* (código com o que se pretende fazer), do tratamento de erros colocado nas cláusulas onde se capturam e tratam as exceções (cláusulas *catch*).

Se uma exceção é lançada, o mecanismo de tratamento de exceções procura a primeira cláusula *catch* com um argumento do tipo da exceção. Se encontra executa o código dessa cláusula *catch* e sai.

## 5 Lista de Especificação das Exceções

Java obriga a declarar as exceções que um método lança na declaração do método depois da lista de argumentos. A especificação das exceções é feita através da palavra-chave **throws** seguida de uma lista de todos os tipos de potenciais exceções.

```
void método() throws Exceção1, Exceção2 {  
    . . .  
}
```

Se a especificação das exceções declaradas não estiver correta, o compilador deteta isso e informa que determinada exceção deve ser tratada, ou então indicada na especificação das exceções, significando que pode ser lançada do método.

Java contém uma classe chamada *Throwable* que descreve tudo o que pode ser lançado como exceção.

Throwable tem 2 subclasses:

- Error descreve erros de compilação e do sistema;
- Exception é o tipo de erros que podem ser lançados de qualquer método das classes de bibliotecas *standard* ou de métodos que escrevemos.

Todas as exceções que possam ocorrer são objetos de classes que são subclasses do tipo *Exception*. É possível capturar qualquer tipo de exceção capturando uma exceção do tipo base *Exception*.

```
catch ( Exception e ) {  
    System.out.println("Capturando a exceção" + e.getMessage() );  
}
```

Ao objeto exceção podem aplicar-se métodos da classe *Throwable*, ou ainda métodos da classe *Object* (superclasse de todas as classes).

Métodos da classe Throwable:

String getMessage()	retorna a mensagem de erro associada com a exceção.
String toString()	retorna uma descrição do objeto.
void printStackTrace()	imprime para o standard erro.
void printStackTrace(PrintStream)	imprime para um <i>stream</i> especificado (por exemplo <i>System.out</i> ).
printStackTrace()	mostra a sequência dos métodos invocados que levaram ao ponto onde a exceção foi lançada.

Método da classe Object:

getClass()	retorna um objeto pertencente à classe <u>Class</u> representando a classe desse objeto exceção. A este objeto retornado pode aplicar-se o método <i>getName()</i> , que retorna o nome da classe.
------------	--

## 6 Criação de Exceções

Podemos criar as nossas próprias exceções para denotar algum erro especial.

Para criar uma classe de exceções é necessário herdar de um tipo existente de exceções.

Mostramos, em seguida, um exemplo no qual o método *main()* chama o método *f()*, o qual chama o método *divide()*. O método *divide()* pode lançar uma exceção.

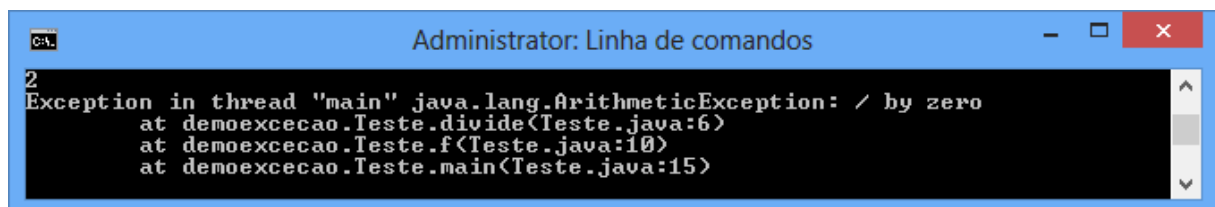
Apresentamos em seguida várias situações. Consideremos em todas essas situações, definida a seguinte classe *Excecao1*:

```
class Excecao1 extends Exception {  
    public Excecao1() { }  
    public Excecao1(String msg) {  
        super(msg);  
    }  
}
```

### 1. Sem tratamento das exceções.

```
public class Teste {  
    public static int divide(int a,int b) {  
        return a/b;  
    }  
    public static int f(int a,int b) {  
        return divide(a, b);  
    }  
    public static void main(String[] args) {  
        System.out.println( f(5, 2) );  
        System.out.println( f(3, 0) );  
        System.out.println( f(4, 1) );  
    }  
}
```

O programa dá erro de execução.





## 2. O método *divide()* pode lançar uma exceção, mas não é capturada nem declarada.

```
public class Teste {  
    public static int divide(int a,int b) {  
        if (b==0)  
            throw new Excecao1("Divisao por zero!");  
        return a/b;  
    }  
  
    public static int f(int a,int b) {  
        return divide(a, b);  
    }  
  
    public static void main(String[] args) {  
        System.out.println( f(5, 2) );  
        System.out.println( f(3, 0) );  
        System.out.println( f(4, 1) );  
    }  
}
```

Dá erro de compilação:

**error J0122:** Exception 'Excecao1' not caught or declared by 'int Teste.divide(int a, int b)'

## 3. O método *divide()* declara a exceção, mas o método *f()* não a captura nem a declara.

```
public class Teste {  
    public static int divide(int a,int b) throws Excecao1 {  
        if (b==0)  
            throw new Excecao1("Divisao por zero!");  
        return a/b;  
    }  
  
    public static int f(int a,int b) {  
        return divide(a, b);  
    }  
  
    public static void main(String[] args) {  
        System.out.println( f(5, 2) );  
        System.out.println( f(3, 0) );  
        System.out.println( f(4, 1) );  
    }  
}
```

Dá erro de compilação:

**error J0122:** Exception 'Excecao1' not caught or declared by 'int Teste.f(int a, int b)'

4. O método *divide()* e o método *f()* declaram a exceção, mas o método *main()* não a captura nem a declara.

```
public class Teste {
    public static int divide(int a,int b) throws Excecao1 {
        if (b==0)
            throw new Excecao1("Divisao por zero!");
        return a/b;
    }
    public static int f(int a,int b) throws Excecao1 {
        return divide(a, b);
    }

    public static void main(String[] args) {
        System.out.println( f(5, 2) );
        System.out.println( f(3, 0) );
        System.out.println( f(4, 1) );
    }
}
```

Dá erro de compilação.

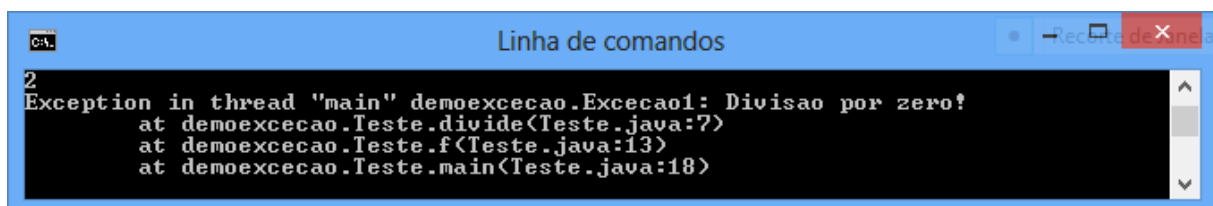
**error J0122:** Exception 'Excecao1' not caught or declared by 'void Teste.main(String[] args)'

5. Os métodos *divide()*, *f()* e *main()* declaram a exceção.

```
public class Teste {
    public static int divide(int a,int b) throws Excecao1 {
        if (b==0)
            throw new Excecao1("Divisao por zero!");
        return a/b;
    }
    public static int f(int a,int b) throws Excecao1 {
        return divide(a, b);
    }

    public static void main(String[] args) throws Excecao1 {
        System.out.println( f(5, 2) );
        System.out.println( f(3, 0) );
        System.out.println( f(4, 1) );
    }
}
```

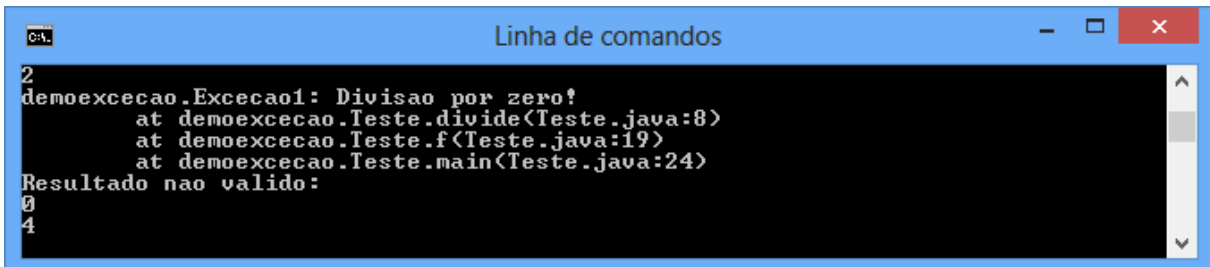
O programa compila mas dá erro de execução:



## 6. A exceção é capturada no método *divide()*.

```
public class Teste {  
    public static int divide(int a,int b) {  
        try {  
            if (b==0)  
                throw new Excecao1("Divisao por zero!");  
            return a/b;  
        } catch ( Exception e ) {  
            e.printStackTrace();  
            System.out.println("Resultado nao valido:");  
            return 0;  
        }  
    }  
  
    public static int f(int a,int b) {  
        return divide(a, b);  
    }  
  
    public static void main(String[] args) {  
        System.out.println( f(5, 2) );  
        System.out.println( f(3, 0) );  
        System.out.println( f(4, 1) );  
    }  
}
```

O método *main()* é completado:

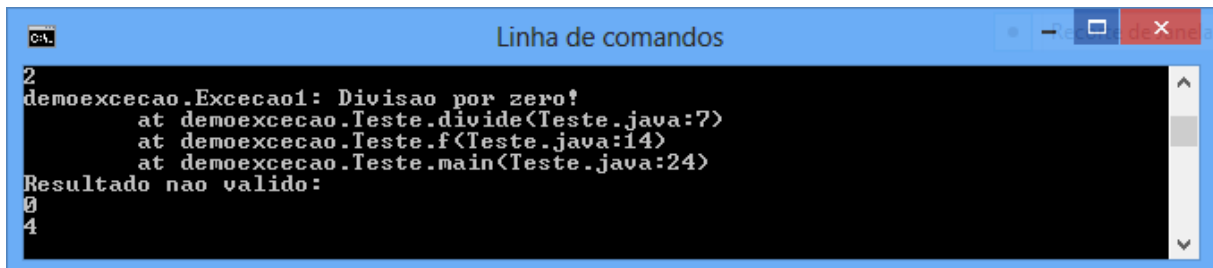


```
C:\> demoexcecao.Excecao1: Divisao por zero!  
    at demoexcecao.Teste.divide<Teste.java:8>  
    at demoexcecao.Teste.f<Teste.java:19>  
    at demoexcecao.Teste.main<Teste.java:24>  
Resultado nao valido:  
0  
4
```

## 7. A exceção é capturada no método *f()*.

```
public class Teste {  
    public static int divide(int a,int b)throws Excecao1 {  
        if (b==0)  
            throw new Excecao1("Divisao por zero!");  
        return a/b;  
    }  
  
    public static int f(int a,int b) {  
        try {  
            return divide(a, b);  
        } catch ( Exception e ) {  
            e.printStackTrace();  
            System.out.println("Resultado nao valido:");  
            return 0;  
        }  
    }  
  
    public static void main(String[] args) {  
        System.out.println( f(5, 2) );  
        System.out.println( f(3, 0) );  
        System.out.println( f(4, 1) );  
    }  
}
```

O método *main()* é completado:

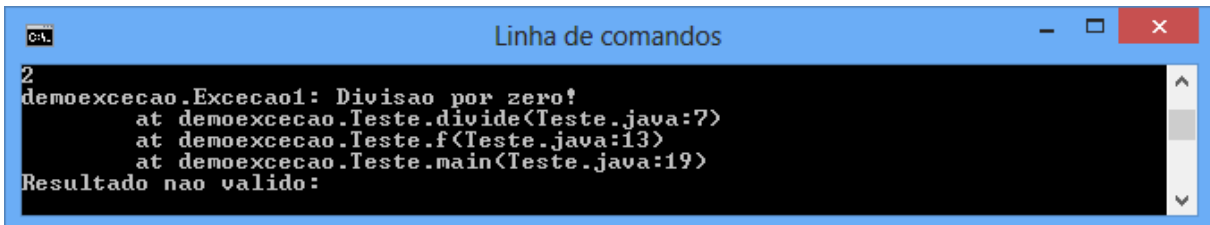


```
C:\> demoexcecao.Excecao1: Divisao por zero!  
    at demoexcecao.Teste.divide<Teste.java:7>  
    at demoexcecao.Teste.f<Teste.java:14>  
    at demoexcecao.Teste.main<Teste.java:24>  
Resultado nao valido:  
0  
4
```

## 8. A exceção é capturada no método *main()*.

```
public class Teste {  
    public static int divide(int a,int b) throws Excecao1 {  
        if (b==0)  
            throw new Excecao1("Divisao por zero!");  
        return a/b;  
    }  
  
    public static int f(int a,int b) throws Excecao1 {  
        return divide(a, b);  
    }  
  
    public static void main(String[] args) {  
        try {  
            System.out.println( f(5, 2) );  
            System.out.println( f(3, 0) );  
            System.out.println( f(4, 1) );  
        } catch ( Exception e ) {  
            e.printStackTrace();  
            System.out.println("Resultado nao valido:");  
            // return 0;  
        }  
    }  
}
```

O método *main()* não é completado, porque o lançamento da exceção causado pela invocação da 2ª instrução do método *main()* causa o abandono do código do bloco *try*:



```
2  
demoexcecao.Excecao1: Divisao por zero!  
    at demoexcecao.Teste.divide<Teste.java:7>  
    at demoexcecao.Teste.f<Teste.java:13>  
    at demoexcecao.Teste.main<Teste.java:19>  
Resultado nao valido:
```

## 7 Exceções em Subclasses

Quando se reescreve um método só se podem lançar exceções que foram especificadas no método da classe base. Deste modo código que funciona com a classe base também funcionará com qualquer objeto derivado da classe base incluindo exceções.

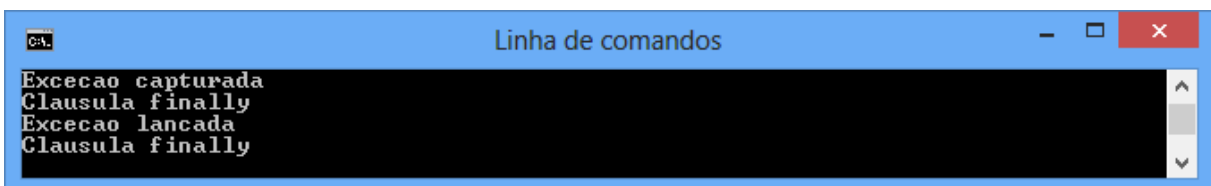
## 8 Cláusula *finally*

Por vezes há a necessidade de executar algum código, quer ocorra ou não uma exceção num *bloco try*, para colocar qualquer coisa no seu estado inicial, tal como um ficheiro aberto ou uma ligação a outro computador. Isso consegue-se usando uma cláusula *finally* no fim do tratamento das exceções.

```
try {  
    // . . . região guardada  
} catch ( Exceção1 e1 ) {  
    . . .  
} catch ( Exceção2 e2 ) {  
    . . .  
} finally {  
    . . .  
}
```

Exemplo:

```
public class Teste1 {  
    private static int count = 0;  
    public static void main (String[] args) {  
        while (true) {  
            try {  
                if (count++ == 0)  
                    throw new Exception();  
                System.out.println("Excecao lancada");  
            } catch ( Exception e ) {  
                System.out.println("Excecao capturada");  
            } finally {  
                System.out.println("Clausula finally");  
                if (count == 2) break;  
            }  
        }  
    }  
}
```



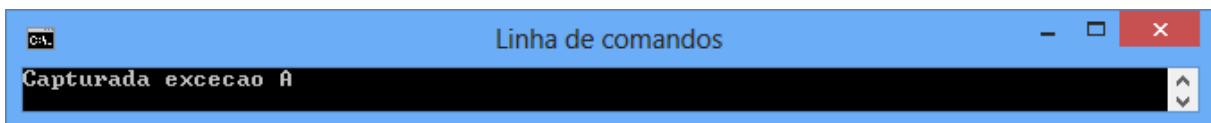
```
C:\>Excecao capturada  
Clausula finally  
Excecao lancada  
Clausula finally
```

## 9 Cláusula *catch* que Captura uma Exceção Lançada

Na determinação da cláusula *catch* que captura uma exceção lançada, o mecanismo de tratamento das exceções procura a primeira cláusula *catch* para a qual o tipo de exceção é o tipo ou subtipo do argumento da cláusula *catch*.

```
class A extends Exception { }
class B extends A { }
public class Teste2 {
    public static void main(String[] args) {
        try {
            throw new B();
        } catch (A a) {
            System.out.println("Capturada excecao A");
        }
    }
}
```

A exceção B é capturada pela cláusula *catch(A a)*.



O código seguinte dará erro de compilação porque a cláusula *catch (B b)* nunca será atingida:

```
class A extends Exception { }
class B extends A { }
public class Teste2 {
    public static void main(String[] args) {
        try {
            throw new B();
        } catch (A a) {
            System.out.println("Capturada excecao A");
        } catch (B b) {
            System.out.println("Capturada excecao B");
        }
    }
}
```

Mensagem de erro produzida:

**error J0102:** Handler for 'B' hidden by earlier handler for 'A'

## 10 Relançamento de uma Exceção (*Rethrowing an Exception*)

Para voltar a lançar uma exceção que foi capturada executa-se *throw referência*.

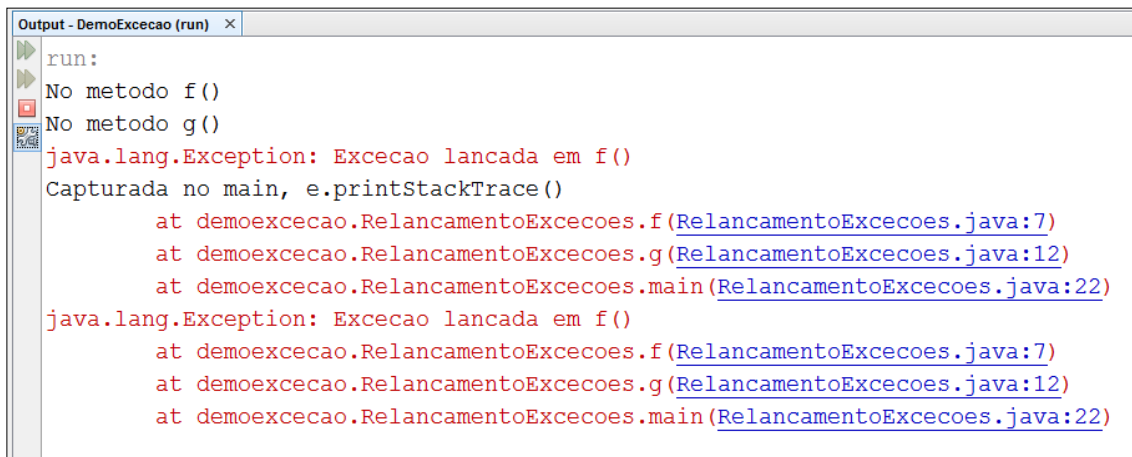
Uma exceção depois de capturada (por uma cláusula *catch*) pode voltar a ser relançada para ser tratada num contexto mais alto.

```
catch ( Exception e ) {  
    System.out.println("Exceção:");  
    e.printStackTrace();  
    throw e;  
}
```

Exemplo:

```
public class RelancamentoExcecoes {  
  
    public static void f() throws Exception {  
        System.out.println("No metodo f()");  
        throw new Exception("Excecao lancada em f()");  
    }  
  
    public static void g() throws Throwable {  
        try {  
            f();  
        } catch ( Exception e ) {  
            System.out.println("No metodo g()");  
            e.printStackTrace();  
            throw e;  
        }  
    }  
  
    public static void main(String[] args) throws Throwable {  
        try {  
            g();  
        } catch ( Exception e ) {  
            System.out.println("Capturada no main, e.printStackTrace()");  
            e.printStackTrace();  
        }  
    }  
}
```





```
run:
No metodo f()
No metodo g()
java.lang.Exception: Excecao lancada em f()
Capturada no main, e.printStackTrace()
    at demoexcecao.RelancamentoExcecoes.f(RelancamentoExcecoes.java:7)
    at demoexcecao.RelancamentoExcecoes.g(RelancamentoExcecoes.java:12)
    at demoexcecao.RelancamentoExcecoes.main(RelancamentoExcecoes.java:22)
java.lang.Exception: Excecao lancada em f()
    at demoexcecao.RelancamentoExcecoes.f(RelancamentoExcecoes.java:7)
    at demoexcecao.RelancamentoExcecoes.g(RelancamentoExcecoes.java:12)
    at demoexcecao.RelancamentoExcecoes.main(RelancamentoExcecoes.java:22)
```

## 11 Exceções *RuntimeException*

Java realiza verificações, em tempo de execução, de exceções da classe *RuntimeException*. Exceções deste tipo são automaticamente lançadas e não é necessário incluí-las nas especificações das exceções.

Todos os tipos de potenciais exceções têm de ser incluídos na declaração de um método (lista com a especificação das exceções) exceto exceções do tipo *RuntimeException*.

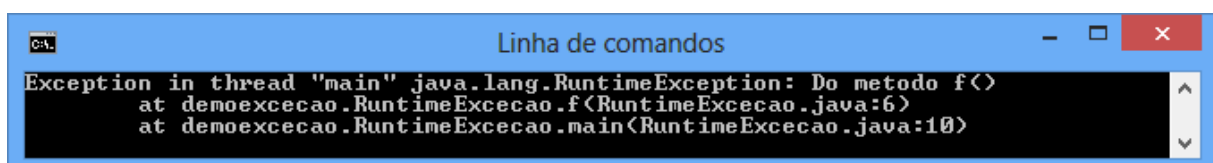
Exceções do tipo *RuntimeException* representam erros de programação. Estes erros tais como *NullPointerException* ou *ArrayIndexOutOfBoundsException* se não forem capturados são reportados na saída do programa auxiliando no processo de depuração (*debugging*).

Exceções do tipo *RuntimeException* ou de classes que herdam deste tipo não necessitam de pertencer à lista de especificação de exceções na declaração de um método.

Exemplo:

```
public class RuntimeExcecao {
    static void f() {
        throw new RuntimeException("Do metodo f()");
    }
    public static void main(String[] args) {
        f();
    }
}
```

Saída produzida:



```
Exception in thread "main" java.lang.RuntimeException: Do metodo f()
    at demoexcecao.RuntimeExcecao.f(RuntimeExcecao.java:6)
    at demoexcecao.RuntimeExcecao.main(RuntimeExcecao.java:10)
```

Se uma exceção do tipo *RuntimeException* é lançada e não é apanhada, na saída do programa *printStackTrace()* é chamado para essa exceção.

## 12 Exemplos

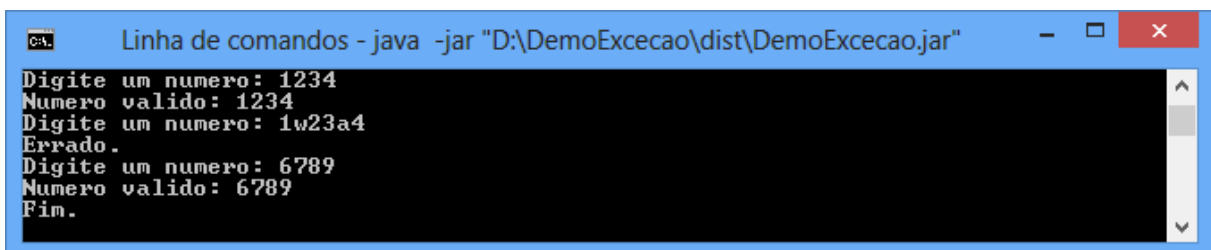
**Exemplo 1:** Programa que efetua 3 leituras de um número inteiro. Se a entrada não pode ser convertida num número, lança uma exceção que é capturada e imprime a mensagem "Errado."

```
import java.io.*;

class Numero1 {

    public static String lerString(String msg) throws IOException {
        DataInputStream din = new DataInputStream(System.in);
        System.out.print(msg);
        return din.readLine();
    }

    public static void main(String[] args) throws IOException {
        int n;
        for(int i=0; i<3; i++) {
            String s = lerString("Digite um numero: ");
            try {
                n = Integer.parseInt(s);
                System.out.println("Numero valido: " + n);
            } catch ( NumberFormatException nfe ) {
                //nfe.printStackTrace();
                System.out.println("Errado.");
            }
        }
        System.out.println("Fim.");
    }
}
```



```
ca. Linha de comandos - java -jar "D:\DemoExcecao\dist\DemoExcecao.jar"
Digite um numero: 1234
Numero valido: 1234
Digite um numero: 1w23a4
Errado.
Digite um numero: 6789
Numero valido: 6789
Fim.
```

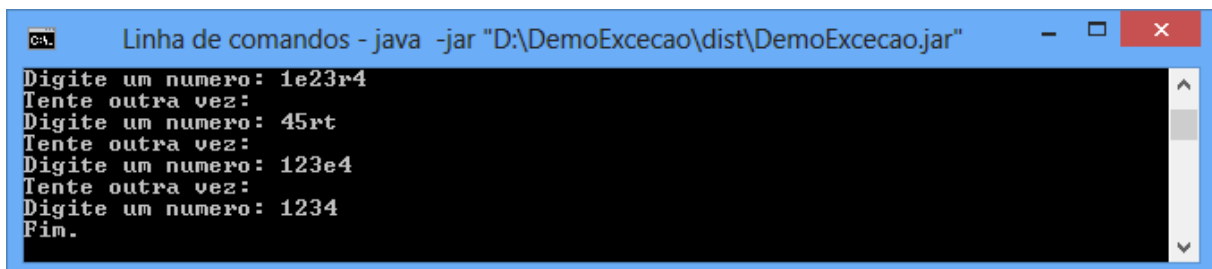
**Exemplo 2:** Programa que efetua a leitura de um número inteiro até a entrada poder ser convertida num número.

```
import java.io.*;

class Numero2 {

    public static String lerString(String msg) throws IOException {
        DataInputStream din = new DataInputStream(System.in);
        System.out.print(msg);
        return din.readLine();
    }

    public static void main(String[] args) throws IOException {
        boolean lido = false;
        int n;
        do {
            String s = lerString("Digite um numero: ");
            try {
                n = Integer.parseInt(s);
                lido = true;
            } catch ( NumberFormatException nfe ) {
                //nfe.printStackTrace();
                System.out.println("Tente outra vez:");
            }
        } while(!lido);
        System.out.println("Fim.");
    }
}
```



```
C:\> java -jar "D:\DemoExcecao\dist\DemoExcecao.jar"
Digite um numero: 1e23r4
Tente outra vez:
Digite um numero: 45rt
Tente outra vez:
Digite um numero: 123e4
Tente outra vez:
Digite um numero: 1234
Fim.
```