



# Tratamento de Exceções

---

Richarlyson A. D'Emery

site: <https://sites.google.com/site/profricodemery/mpoo>

grupo: [http://groups.google.com/group/mpoo\\_uast](http://groups.google.com/group/mpoo_uast)

email grupo: [mpoo\\_uast@googlegroups.com](mailto:mpoo_uast@googlegroups.com)

contato: [rico\\_demery@yahoo.com.br](mailto:rico_demery@yahoo.com.br)

# Sumário

---



- Objetivos
- Introdução
- A Classe Exception
- Captura e Tratamento
- try – catch - finally
- throws

# Objetivos

---

- Entender tratamento de exceção e erro
- Implementar blocos **try**
- Levantando (**throw**) exceções
- Usar blocos **catch** para especificar *handlers*
- Usar blocos **finally** para liberar recursos
- Entender a hierarquia de exceções de java
- Criar exceções definidas pelo programador

# Exceção

---

- Uma exceção é um evento não esperado
- Exceções podem ocorrer quando:
  - se tenta abrir um arquivo que não existe,
  - ocorrer problemas com a rede,
  - operandos estão fora do intervalo válido,
  - se tentar carregar uma classe que não existe,
  - índice do array fora dos limites,
  - overflow,
  - divisão por zero,
  - parâmetros inválidos,
  - falta de memória.

# Exceção

---

- A classe Exception define erros menos graves em Java e que devem ser tratados
- Java permite um programa capturar:
  - todos os tipos de exceções,
  - todas as exceções de um determinado tipo,
  - ou alguns tipos de exceções.
- Vantagem
  - programador pode ver o processamento do erro próximo ao código gerador
  - determinar se o tratamento de erro está implementado e correto.
- Desvantagem
  - o código fica poluído de processamento de erro,
  - fica mais difícil de se concentrar na aplicação.
  - A aplicação pode ficar difícil de entender e manter.

# Exceção

---

- A separação do código de tratamento de erro do programa principal esta consistente com as características de separabilidade da orientação a objetos.
- O programa fica mais robusto reduzindo a probabilidade de erros não capturados por um programa.
- Tratamento de Exceções servem para habilitar programas a tratarem os erros em vez de sofrer suas conseqüências.

# Exceção

---

- Tratamento de Exceções trata
  - **erros síncronos** (divisão por zero), não são projetados para tratamento de
  - **erros assíncronos** (E/S de disco, chegada de mensagem da rede, etc), este são melhor tratados por outra técnica:
    - ***processamento de interrupções.***

# Exceção

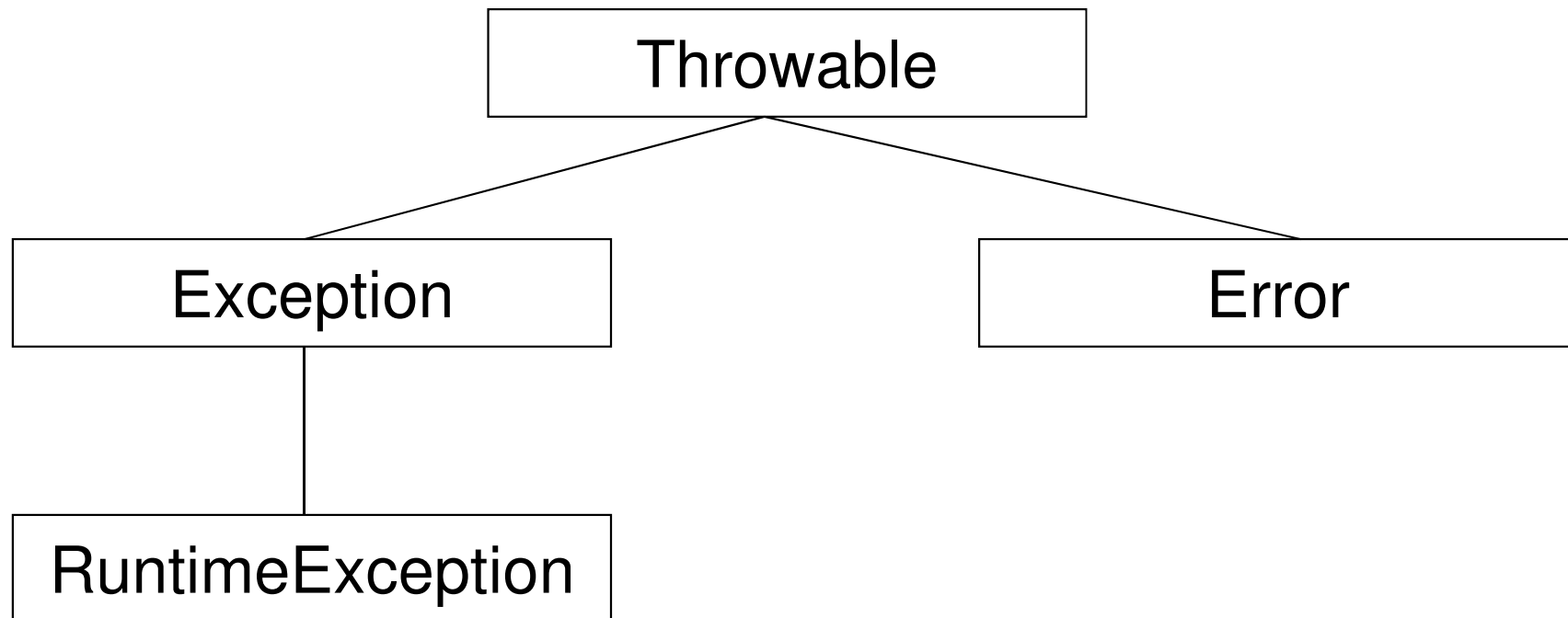
---

- Tratamento de Exceções são utilizados em situações em que o sistema pode se recuperar de um mal funcionamento causando uma exceção.
  - O procedimento de recuperação é chamado ***exception handler***.
- Um *exception handler* é utilizado em situações em que o mal funcionamento será tratado por um escopo diferente daquele em o mal funcionamento foi detectado.
- Exceções são objetos de classes derivadas da superclasse ***Exception***.



# Classes de Exceções

---



# Class Exception

docs.oracle.com/javase/10/docs/api/java/lang/Exception.html

OVERVIEW MODULE PACKAGE **CLASS** USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY NESTED | FIELD | CONSTR | METHOD DETAIL FIELD | CONSTR | METHOD

SEARCH:

## Class Exception

java.lang.Object  
  java.lang.Throwable  
    java.lang.Exception

**All Implemented Interfaces:**  
Serializable

**Direct Known Subclasses:**  
AbsentInformationException, AclNotFoundException, ActivationException, AgentInitializationException, AgentLoadException, AlreadyBoundException, ApplicationException, AttachNotSupportedException, AWTException, BackingStoreException, BadAttributeValueExpException, BadBinaryOpValueExpException, BadLocationException, BadStringOperationException, BrokenBarrierException, CardException, CertificateException, ClassNotLoadedException, CloneNotSupportedException, DataFormatException, DatatypeConfigurationException, DestroyFailedException, ExecutionControl.ExecutionControlException, ExecutionException, ExpandVetoException, FontFormatException, GeneralSecurityException, GSSEException, IllegalClassFormatException, IllegalConnectorArgumentsException, IncompatibleThreadStateException, InterruptedException, IntrospectionException, InvalidApplicationException, InvalidMidiDataException, InvalidPreferencesFormatException, InvalidTargetObjectTypeException, InvalidTypeException, InvocationException, IOException, JAXBException, JMEException, JShellException, KeySelectorException, LambdaConversionException, LastOwnerException, LineUnavailableException, MarshalException, MidiUnavailableException, MimeTypeParseException, MimeTypeParseException, NamingException, NoninvertibleTransformException, NonInvertibleTransformException, NotBoundException, NotOwnerException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, ReflectiveOperationException, RefreshFailedException, RemarshalException, RuntimeException, SAXException, ScriptException, ServerNotActiveException, SOAPException, SQLException, StringConcatException, TimeoutException, TooManyListenersException, TransformerException, TransformException, UnavailableServiceException, UnmodifiableClassException, UnsupportedAudioFileException, UnsupportedCallbackException, UnsupportedFlavorException, UnsupportedLookAndFeelException, URIReferenceException, URISyntaxException, UserException, VMStartException, XAException, XMLParseException, XMLSignatureException, XMLStreamException, XPathException

```
public class Exception
extends Throwable
```

# Class Exception

docs.oracle.com/javase/10/docs/api/java/lang/Exception.html

OVERVIEW MODULE PACKAGE **CLASS** USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

SEARCH:

## Constructor Summary

**Constructors**

Modifier	Constructor	Description
	<code>Exception()</code>	Constructs a new exception with null as its detail message.
	<code>Exception(String message)</code>	Constructs a new exception with the specified detail message.
	<code>Exception(String message, Throwable cause)</code>	Constructs a new exception with the specified detail message and cause.
protected	<code>Exception(String message, Throwable cause, boolean enableSuppression, boolean writableStackTrace)</code>	Constructs a new exception with the specified detail message, cause, suppression enabled or disabled, and writable stack trace enabled or disabled.
	<code>Exception(Throwable cause)</code>	Constructs a new exception with the specified cause and a detail message of <code>(cause==null ? null : cause.toString())</code> (which typically contains the class and detail message of cause).

## Method Summary

**Methods declared in class java.lang.Object**

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

**Methods declared in class java.lang.Throwable**

`addSuppressed, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, getSuppressed, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

# Exceção – Exemplo 1

---

- Índice do array fora dos limites,

```
public class Exemplo1{  
    public static void main(String [] args) {  
        int i = 0;  
        String nomes [] = {"Bart Simpson", "Dr. Smith", "Rico D'Emery"};  
        while (i<4){  
            System.out.println(nomes[i]);  
            i++;  
        }  
    }  
}
```

# Exceções

---

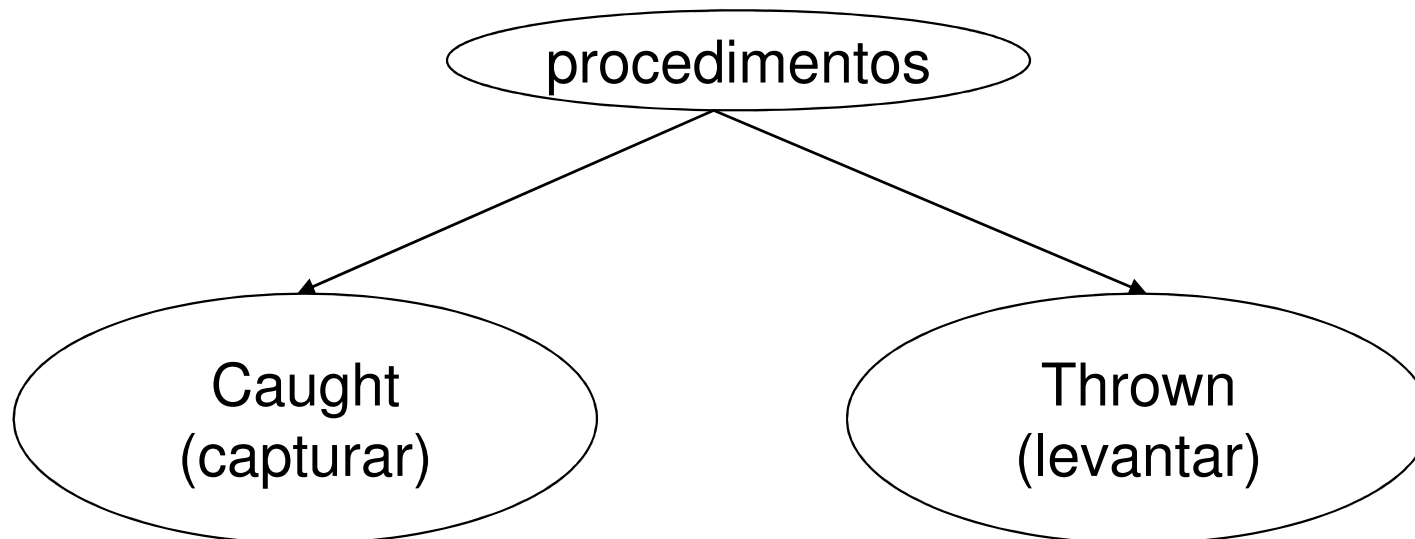
- Se nenhum método capturar a exceção o programa termina e uma mensagem de erro é emitida.
  - Para o Exemplo 1

```
Exception in thread "main"  
java.lang.ArrayIndexOutOfBoundsException: 3  
at Exemplo1.main(Exemplo1.java:10)
```

# Tratamento de Exceção em Java

---

- Quando um método detecta um erro e não está apto a manuseá-lo o método *levanta uma exceção (throw)*.
- Se existe um *exception handler*, para processar a exceção ela será capturada (*caught*) e tratada (*handled*).
- Exceções são eventos surpresas que ocorrem durante a execução de um programa.



# Como Capturar e Tratar uma Exceção

---

- Para manipular uma possível exceção use **try**.
- Para capturar e tratar exceções levantadas use **catch**.
- Para garantir que um bloco sempre seja executado use **finally**.



# try – catch – finally

---

```
try{  
    //código que pode levantar uma exceção  
}  
catch{  
    //código que será executado se TipoDaExceção for  
    levantada  
}  
finally{  
    //código sempre executado  
}
```



# Tratando Exceções – Exemplo 1

---

- Tratando índice do array fora dos limites,

```
public class Exemplo1{
    public static void main(String [] args) {
        int i = 0;
        String nomes [] = {"Bart Simpson", "Dr. Smith", "Rico D'Emery"};

        while (i<4){
            try{
                System.out.println(nomes[i]);
                i++;
            }
            catch(ArrayIndexOutOfBoundsException arrayIndexOutOfBoundsException){
                System.out.println("Erro no Array!");
                break;
            }
        }
    }
}
```

# Tratando Exceções – Exemplo 1

- Tratando índice do array fora dos limites,

```
public class Exemplo1{
    public static void main(String [] args) {
        int i = 0;
        String nomes [] = {"Bart Simpson", "Dr. Smith", "Rico D'Emery"};

        while (i<4){
            try{
                System.out.println(nomes[i]);
                i++;
            }
            catch(ArrayIndexOutOfBoundsException arrayIndexOutOfBoundsException){
                System.out.println("Erro no Array!");
                break;
            }
        }
    }
}
```

**Mas, e o finally?**

# Tratando Exceções – Exemplo 1

- Tratando índice do array fora dos limites,

```
public class Exemplo1{
    public static void main(String [] args) {
        int i = 0;
        String nomes [] = {"Bart Simpson", "Dr. Smith", "Rico D'Emery"};

        while (i<4){
            try{
                System.out.println(nomes[i]);
                i++;
            }
            catch(ArrayIndexOutOfBoundsException arrayIndexOutOfBoundsException){
                System.out.println("Erro no Array!");
                break;
            }

            finally{
                System.out.println("Sempre mostrado!");
            }
        }
    }
}
```

# Exemplo 1 - Análise

---

Observe o que acontece nesses cenários:

■ Situação 1:

```
...
while (i<10){
    try{
        System.out.println(nomes[i]);
        i++;
        System.out.println(i);
    }
    catch(ArrayIndexOutOfBoundsException arrayIndexOutOfBoundsException){
        System.out.println("Erro de acesso a elemento inexistente no Array!");
        //break;
    }
    finally{
        System.out.println("Sempre mostrado!");
    }
}
...
```

# Exemplo 1 - Análise

---

Observe o que acontece nesses cenários:

- Situação 2:

...

```
while (i<10){  
    try{  
        System.out.println(i);  
        System.out.println(nomes[i]);  
        i++;  
    }  
    catch(ArrayIndexOutOfBoundsException arrayIndexOutOfBoundsException){  
        System.out.println("Erro de acesso a elemento inexistente no Array!");  
        //break;  
    }  
    finally{  
        System.out.println("Sempre mostrado!");  
    }  
}
```

...

# Exemplo 1 - Análise

---

Observe o que acontece nesses cenários:

■ Situação 3:

...

```
while (i<10){  
    try{  
        System.out.println(nomes[i]);  
        i++;  
    }  
    catch(ArrayIndexOutOfBoundsException arrayIndexOutOfBoundsException){  
        System.out.println("Erro de acesso a elemento inexistente no Array!");  
        //break;  
    }  
    finally{  
        System.out.println(i);  
        System.out.println("Sempre mostrado!");  
    }  
}
```

...

# Declarar ou Tratar

---

- Trate as exceções usando try-catch- finally
- Declare que o código pode levantar mais de uma exceção.
- Exceções de *runtime* não precisam ser declaradas ou tratadas.

# Exemplo 2

---

- Divisão por 0 (zero)

```
import java.util.Scanner;

public class Exemplo2 {
    public static void main(String [] args) {
        Scanner input = new Scanner( System.in );
        int resultado=0;
        System.out.print( "Informe o primeiro número: ");
        int numero1 = Integer.parseInt(input.nextLine());
        System.out.print( "Informe o segundo número: ");
        int numero2 = Integer.parseInt(input.nextLine());
        resultado = (numero1 / numero2);
        System.out.println( "O resultado é " + resultado);
    }
}
```



# Tratando Exceções – Exemplo 2

---

## ■ Tratando a Divisão por 0

```
import java.util.Scanner;
public class Exemplo2 {
    public static void main(String [] args) {
        Scanner input = new Scanner( System.in );
        int resultado=0;

        System.out.print( "Informe o primeiro número: ");
        int numero1 = Integer.parseInt(input.nextLine());
        System.out.print( "Informe o segundo número: ");
        int numero2 = Integer.parseInt(input.nextLine());

        try{
            resultado = (numero1 / numero2);
        }
        catch(ArithmeticException arithmeticException){
            System.out.print( "Não é possível divisão por zero");
        }
        System.out.println( "O resultado é " + resultado);
    }
}
```

# Tratando Exceções – Exemplo 2

## ■ Tratando a Divisão por 0

```
import java.util.Scanner;
public class Exemplo2 {
    public static void main(String [] args) {
        Scanner input = new Scanner( System.in );
        int resultado=0;

        System.out.print( "Informe o primeiro número: " );
        int numero1 = Integer.parseInt(input.nextLine());
        System.out.print( "Informe o segundo número: " );
        int numero2 = Integer.parseInt(input.nextLine());

        try{
            resultado = (numero1 / numero2);
        }
        catch(ArithmeticException arithmeticException){
            System.out.print( "Não é possível divisão por zero");
        }
        System.out.println( "O resultado é " + resultado);
    }
}
```

Está no lugar Correto?

# Tratando Exceções – Exemplo 2

## ■ Tratando a Divisão por 0

```
import java.util.Scanner;
public class Exemplo2 {
    public static void main(String [] args) {
        Scanner input = new Scanner( System.in );
        int resultado=0;

        System.out.print( "Informe o primeiro número: ");
        int numero1 = Integer.parseInt(input.nextLine());
        System.out.print( "Informe o segundo número: ");
        int numero2 = Integer.parseInt(input.nextLine());

        try{
            resultado = (numero1 / numero2);
            System.out.println( "O resultado é " + resultado);
        }
        catch(ArithmeticException arithmeticException){
            System.out.print( "Não é possível divisão por zero");
        }
    }
}
```

# Tratando Exceções – Exemplo 2

- Tratando a Divisão por 0

```
import java.util.Scanner;
public class Exemplo2 {
    public static void main(String [] args) {
        Scanner input = new Scanner( System.in );
        int resultado=0;

        System.out.print( "Informe o primeiro número: ");
        int numero1 = Integer.parseInt(input.nextLine());
        System.out.print( "Informe o segundo número: ");
        int numero2 = Integer.parseInt(input.nextLine());

        try{
            resultado = (numero1 / numero2);
            System.out.println( "O resultado é " + resultado);
        }
        catch(ArithmeticException arithmeticException){
            System.out.print( "Não é possível divisão por zero");
        }
    }
}
```

Mas....

Existe algum outro problema envolvendo Exceção?

# Tratando Exceções – Exemplo 2

## ■ Tratando a Divisão por 0

```
import java.util.Scanner;
public class Exemplo2 {
    public static void main(String [] args) {
        Scanner input = new Scanner( System.in );
        int resultado=0;

        System.out.print( "Informe o primeiro número: ");
        int numero1 = Integer.parseInt(input.nextLine());
        System.out.print( "Informe o segundo número: ");
        int numero2 = Integer.parseInt(input.nextLine());

        try{
            resultado = (numero1 / numero2);
            System.out.println( "O resultado é " + resultado);
        }
        catch(ArithmeticException arithmeticException){
            System.out.print( "Não é possível divisão por zero");
        }
    }
}
```

← ? ←

# Tratando Exceções – Exemplo 2

## ■ Tratando a Divisão por 0

```
import java.util.Scanner;
public class Exemplo2 {
    public static void main(String [] args) {
        Scanner input = new Scanner( System.in );
        int resultado=0;

        System.out.print( "Informe o primeiro número: " );
        int numero1 = Integer.parseInt(input.nextLine());
        System.out.print( "Informe o segundo número: " );
        int numero2 = Integer.parseInt(input.nextLine());

        try{
            resultado = (numero1 / numero2);
            System.out.println( "O resultado é " + resultado);
        }
        catch(ArithmeticException arithmeticException){
            System.out.print( "Não é possível divisão por zero");
        }
    }
}
```

**Qual o Problema de Exceção?**

# Tratando Exceções – Exemplo 2

- Tratando a Divisão por 0

```
import java.util.Scanner;
public class Exemplo2 {
    public static void main(String [] args) {
        Scanner input = new Scanner( System.in );
        int resultado=0;

        System.out.print( "Informe o primeiro número: ");
        int numero1 = Integer.parseInt(input.nextLine());
        System.out.print( "Informe o segundo número: ");
        int numero2 = Integer.parseInt(input.nextLine());

        try{
            resultado = numero1 / numero2;
            System.out.println( "O resultado é " + resultado);
        }
        catch(ArithmeticException arithmeticException){
            System.out.print( "Não é possível divisão por zero");
        }
    }
}
```

**Qual o Problema de Exceção?**

**Analisando:**

**número1 = 10 ou dez ou ten**

**número2 = 0 ou zero**

# Tratando Exceções – Exemplo 2

## ■ Tratando a Divisão por 0

```
import java.util.Scanner;
public class Exemplo2 {
    public static void main(String [] args) {
        Scanner input = new Scanner( System.in );
        int resultado=0;
```

```
        System.out.print( "Informe o primeiro número: " );
```

```
        int numero1 = Integer.parseInt(input.nextLine());
```

```
        System.out.print( "Informe o segundo número: " );
```

```
        int numero2 = Integer.parseInt(input.nextLine());
```

```
        try{
```

```
            resultado = numero1 / numero2;
```

```
            System.out.println( "O resultado é " + resultado );
```

```
        }
```

```
        catch(ArithmeticException arithmeticException){
```

```
            System.out.print( "Não é possível divisão por zero." );
```

```
        }
```

```
    }
```

```
}
```

**Qual o Problema de Exceção?**

**Analisando:**

**número1 = 10 ou dez ou ten**

**número2 = 0 ou zero**

**Solução:**

**Tratar *NumberFormatException***



# Tratando Exceções

- Mas **try** admite vários **catch**?

```
try{
    numero1 = Integer.parseInt(input.nextLine());
    System.out.print( "Informe o segundo número: ");
    numero2 = Integer.parseInt(input.nextLine());
    resultado = (numero1 / numero2);
    System.out.println( "O resultado é " + resultado);
}
catch(NumberFormatException n){
    System.out.println("O número informado não é válido");
    n.printStackTrace();
}
catch(ArithmeticException a){
    System.out.println("Não é possível divisão por zero");
    a.printStackTrace();
}
```

# Tratando Exceções

---

- Cuidado:

```
try{  
    }  
catch(Exception e){  
    }
```

Ou

```
try{  
    }  
catch(NumberFormatException n){  
    }  
catch(ArithmeticException a){  
    }
```

# Exemplo 3

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.UnknownHostException;

//URLReader
public class Exemplo3{
    public static void main(String [ ] args){
        try{
            URL url = new URL("http://www.google.com.br");
            BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));

            String inputLine;
            while ((inputLine = in.readLine()) != null)
                System.out.println(inputLine);
        }
        catch(UnknownHostException e){
            System.out.println("Erro de comunicação de Host!");
            System.out.println(e.getMessage() + "\n");
            e.printStackTrace();
        }
        catch (MalformedURLException e) {
            System.out.println("Erro de fomrulação de URL!");
            e.printStackTrace();
        }
        catch (IOException e) {
            System.out.println("Erro de leitura de arquivo");
            e.printStackTrace();
        }
    }
}
```

# Disparando uma exceção

---

- A instrução *throw* é executada para indicar que uma exceção ocorreu.
- O operando de um *throw* está em Throwable (java.lang)
- Classes de Throwable
  - Exception – ocorre durante a execução
    - Objeto chamado se *objeto de execução*
  - Error – não deveria ocorrer (sistema)

# A Cláusula throws

---

- Use o throws na declaração do método.
- Um método pode levantar mais de uma exceção.
- Após a cláusula throws especifique a lista das possíveis exceções levantadas pelo método:

```
public void metodoProb() throws IOException,  
    MalformedURLException{ }
```

# Exemplo 4

```
//Exemplo4.java
import java.util.Scanner;

public class Exemplo4 {
    public static void main(String [] args) {
        Scanner input = new Scanner( System.in );
        float resultado=0;
        //Assumindo que o usuário digite números válidos
        System.out.print( "Informe o primeiro número: ");
        float numero1 = Integer.parseInt(input.nextLine());
        System.out.print( "Informe o segundo número: ");
        float numero2 = Integer.parseInt(input.nextLine());
        try{
            resultado = quocient (numero1, numero2);
            System.out.println( "O resultado é " + resultado);
        }catch(Exception e){}
        finally{}
    }

    public static float quocient (float numer, float den) throws Exemplo4Exception{
        if (den==0)
            throw new Exemplo4Exception();
        else
            return (float) numer/den;
    }
}
```

## Exemplo 3 – cont.

---

```
//Exemplo4Exception.java
//Definicao da excecao
//Dispara uma exceção quando se tenta uma divisão por zero

public class Exemplo4Exception extends ArithmeticException{

    // construtor de argumento é dispara erro default
    public Exemplo4Exception(){
        System.out.print( "Não é possível divisão por zero");
    }

    //construtor com mensagem personalizada
    public Exemplo4Exception(String mensagem){
        super (mensagem);
    }
}
```

# Exemplo 4 - Análise

- Experimente usar:

```
public Exemplo4Exception(String mensagem){  
    super (mensagem);  
}
```

- Em:

```
//Exemplo4.java  
import java.util.Scanner;  
  
public class Exemplo4 {  
    public static void main(String [] args) {  
        Scanner input = new Scanner( System.in );  
        float resultado=0;  
        //Assumindo que o usuário digite números válidos  
        System.out.print( "Informe o primeiro número: ");  
        float numero1 = Integer.parseInt(input.nextLine());  
        System.out.print( "Informe o segundo número: ");  
        float numero2 = Integer.parseInt(input.nextLine());  
        try{  
            resultado = quocient (numero1, numero2);  
            System.out.println( "O resultado é " + resultado);  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
        finally{}  
    }  
  
    public static float quocient (float numer, float den) throws Exemplo4Exception{  
        if (den==0)  
            throw new ("Erro: denominador zero");  
        else  
            return (float) numer/den;  
    }  
}
```



# Métodos

- printStackTrace, getStackTrace e getMessage
- Experimente usar o catch

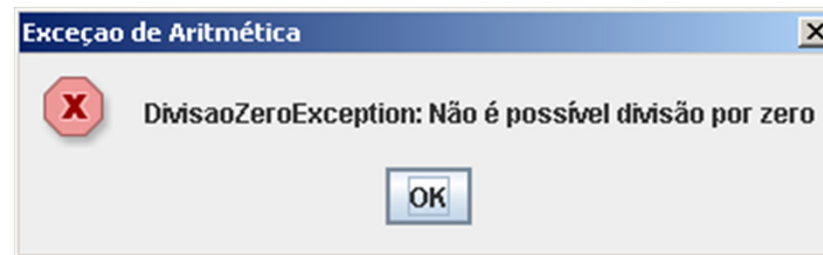
```
catch(Exception e){  
    //Usando o Construtor: Exemplo4Exception(String mensagem)  
    System.err.println(e.getMessage());  
    //obtem informações de rastreamento da pilha  
    StackTraceElement[] traceElements = e.getStackTrace();  
    System.out.println("\n ---Descrição da pilha de Exceção ---");  
    for(StackTraceElement element:traceElements){  
        System.out.println(element.getClassName());  
        System.out.println(element.getFileName());  
        System.out.println(element.getLineNumber());  
        System.out.println(element.getMethodName());  
    }  
    System.out.println("\n ---DESCREVENDO A EXCEÇÃO---");  
    //Imprime a exceção:  
    e.printStackTrace();  
}
```

# Exercício 1

- Utilize uma GUI para
  - Permitir a entrada de um numerador
  - Permitir a entrada de um denominador



- Caso a divisão seja por zero exiba a mensagem :



# Exercício 2

---

- Modifique o exercício 1 para:
  - Permitir a entrada de um número
  - Permitir a entrada de outro número
  - Permite escolher o tipo de operação
  - Os números poderão ser reais
  - Caso a divisão seja por zero exiba mensagem de erro

# Exercício 3

---

- Modifique o exercício 2 da seguinte forma:
  - um método contendo o código com o tratamento da exceção.
  - Crie um segundo método que faça uma chamada ao primeiro.

# Exercício 4

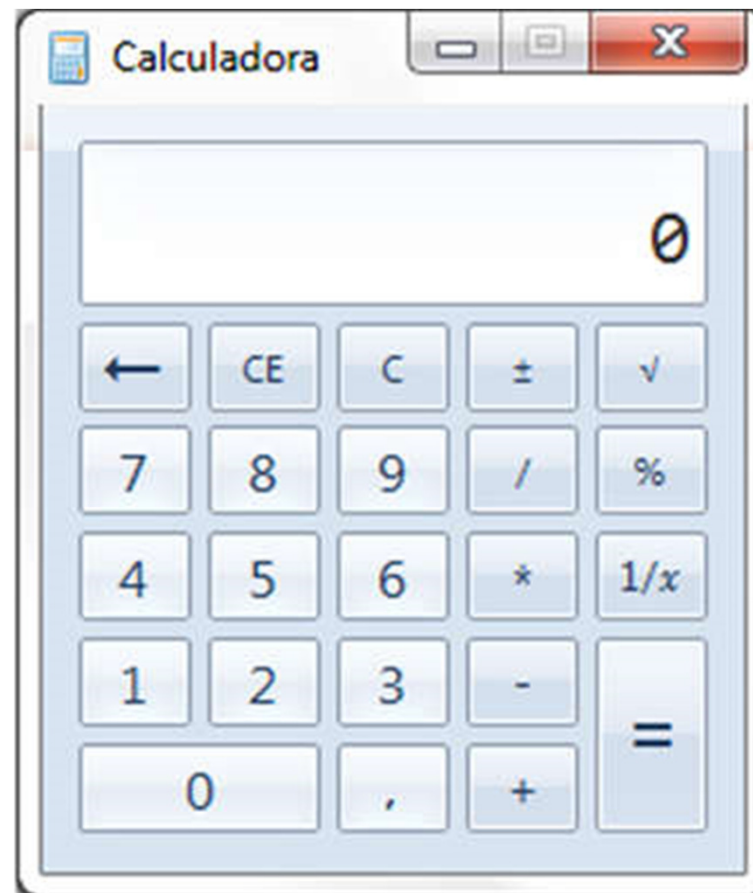
---

- Modifique o exercício 2 da seguinte forma:
  - O código com o tratamento da exceção deverá estar em uma outra classe.

# Exercício 5

---

- Implemente a calculadora abaixo. Faça uso de try/catch e throw/throws.





# FIM

---

Richarlyson D'Emery  
rico\_demery@yahoo.com.br