

Depois disto ouvi a voz do Senhor, que dizia:
A quem enviarei, e quem há de ir por nós?

Então disse eu: Eis-me aqui, envia-me a mim.

Isaías 6:8

Curso de Especialização em Tecnologia Java

LINGUAGEM DE PROGRAMAÇÃO JAVA I

- ▶ Prof: José Antonio Gonçalves
- ▶ zag655@gmail.com
- ▶ Ao me enviar um e-Mail coloque o “Assunto” começando: “pós2013_2+seu nome”

Ementa:

- **Orientação a Objetos em Java:** Classes, Objetos, Herança, Polimorfismo, Classes Abstratas, Interface;

Conteúdo já
visto

- **Exceções;**
- **Manipulação de Texto e Strings;**
- **Componentes básicos de interface gráfica;**
- **Tratamento de Eventos.**

Bibliografia:

DEITEL, H.; DEITEL, P. JAVA – Como Programar. 3.ed. Porto Alegre: Bookman, 2001.

ECKEL, B. Thinking in Java , 2nd edition, EUA: Prentice Hall, 2000.

HORSTMANN, C. Core Java – Advanced Features. EUA: Prentice Hall, 2000. Volume II.

HORSTMANN, C. Core Java – Fundamentals. EUA: Prentice Hall, 2000. Volume I.

Nestes Slides:

- **Orientação a Objetos em Java:**

Conceito e aplicação sobre Tratamento de Exceções;

Exceções

Exceções (*definição*)

- Trata-se de uma **indicação de um erro** que ocorre durante a execução de um programa;
- Chama-se **Exceção** justamente por ser algo (alguma situação) que foge a regra;

Exceções (*uso*)

–TRATAMENTO DAS EXCEÇÕES é um recurso de algumas linguagens de programação (incluindo Java) que nos ajuda a construir aplicações mais robustas e tolerantes as falhas.

- Colocando dentro do tratamento **operações passíveis de apresentar falhas**

Exceções (*estruturas de tratamento*)

Blocos:

- try (tentar)
- catch (capturar)
- finally (finalmente)

Exceções *(tratamento: composições possíveis)*

- try+catch;
- try + finally
- try + catch +finally

Não existe um bloco “try” sozinho.

Exceções *(exemplo de estrutura)*

try{

...código passível de apresentar falha...

(caso apresente falha esta disparará uma mensagem de erro a qual chamamos exceção, que, em tese, deverá ser “capturada” pelo catch() e, dentro deste bloco, a falha deve ser tratada)

}

catch(declaração de um objeto do tipo da exceção lançada pelo bloco try){

... código que tratará a exceção (se esta for lançada por uma operação contida no bloco try)...

}

finally{

...código executado independente do catch ser executado ...

(isto é, o que estiver aqui sempre será executado...)

}

Exceções (*estudo de caso*)

- Imagine que você quer propiciar uma interação do sistema com usuário através de uma entrada de dados
- O exemplo (código fonte) a seguir cria esta possibilidade, porém o **valor** de saída deste código sempre **será do tipo String**. Vejamos:

Exceções (*estudo de caso: entrada de dados*)

```
1. import java.io.BufferedReader;
2. import java.io.IOException;
3. import java.io.InputStreamReader;

4. public class Excecao1 {
5.     public static void main(String arg[]){
6.         BufferedReader memoria = new BufferedReader(new InputStreamReader(System.in));
7.         System.out.println("\n Entrada de Dados");
8.         String s = "";
9.         System.out.println("\nEntre com um valor: ");
10.        try{
11.            s = memoria.readLine();
12.        }
13.        catch(IOException erro1){
14.            System.out.println("\nErro de entrada de dados: "+erro1);
15.        }
16.        finally{
17.            System.out.println("\n Entrou no Finally");
18.        }
19.        System.out.println("\n O Valor de Entrada foi "+ s);
20.    }
21. }
```

Exceções (*estudo de caso: entrada de dados*)

- No exemplo (código fonte) anterior o valor de saída do método sempre será do tipo String.
- Mas se a necessidade for que este **valor seja de outro tipo**. Por exemplo do tipo inteiro (int)? Neste caso poderemos fazer uma conversão de tipos:

próximo slide, linha 19: **int numero = Integer.parseInt(s);**

- Teste o código a seguir, informando um número inteiro:

Exceções (*estudo de caso*)

```
1. import java.io.BufferedReader;
2. import java.io.IOException;
3. import java.io.InputStreamReader;

4. public class Excecao2 {
5.     public static void main(String arg[]){
6.         BufferedReader memoria = new BufferedReader(new InputStreamReader(System.in));
7.         System.out.println("\n Entrada de Dados");
8.         String s = "";
9.         System.out.println("\nEntre com um valor: ");
10.        try{
11.            s = memoria.readLine();
12.        }
13.        catch(IOException erro1){
14.            System.out.println("\nErro de entrada de dados: "+erro1);
15.        }
16.        finally{
17.            System.out.println("\n Entrou no Finally");
18.        }
19.        int numero = Integer.parseInt(s);
20.        System.out.println("\n O quadrado do valor de entrada eh: "+numero*numero);
21.    }
22. }
```

Exceções (*estudo de caso*)

- Mas... Se no código anterior (classe Exceção2) o usuário, **inadvertidamente, inserir uma letra** ao invés de número inteiro?

Teste seu código forçando o erro citado acima:

Exceções (*estudo de caso*)

- O fato de ter inserido uma letra, ao invés de um número, fez com que ocorressem **duas falhas** nas operações:

de conversão de tipos:

linha 19: **int numero = Integer.parseInt(s);**

E também no cálculo:

linha 20: `System.out.println("\n O quadrado do valor de entrada eh: "+numero*numero);`

Disparará uma exceção do tipo
“não-verificada”

Exceções (*estudo de caso*)

E apresentou as seguintes mensagens referentes ao erro:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "a"  
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)  
    at java.lang.Integer.parseInt(Integer.java:492)  
    at java.lang.Integer.parseInt(Integer.java:527)  
    at aulaspós_23_06_2012.Excecao2.main(Excecao2.java:19)
```

Nota:

Fique atento às mensagens de erros: Além de te mostrar o problema também te indica a solução. Neste caso nos mostrando que deverá tratar a exceção "`java.lang.NumberFormatException`" lançada pela operação contida na **linha 19**: `int numero = Integer.parseInt(s);`

Exceções (*estudo de caso*)

Deste exemplo podemos observar duas coisas:

- Neste caso, a “conversão de tipos” e “cálculos” **podem apresentar** falhas, logo **deveriam estar dentro do bloco try** (pode ser no mesmo que já existe neste código);
- Mesmo a operação de conversão e cálculo estarem dentro de um bloco try, devemos tratar o erro que estas operações, por ventura, venham a produzir. Para isso deveremos **criar um bloco catch que capture cada exceção lançada** caso este erro ocorra:

Exceções (*estudo de caso*)

```
1. import java.io.BufferedReader;
2. import java.io.IOException;
3. import java.io.InputStreamReader;

4. public class Excecao2 {
5.     public static void main(String arg[]){
6.         BufferedReader memoria = new BufferedReader(new InputStreamReader(System.in));
7.         System.out.println("\n Entrada de Dados");
8.         String s = "";
9.         System.out.println("\nEntre com um valor: ");
10.        try{
11.            s = memoria.readLine();
12.            int numero = Integer.parseInt(s);
13.            System.out.println("\n O quadrado do valor de entrada eh: "+numero*numero);
14.        }
15.        catch(IOException erro1){
16.            System.out.println("\nErro de entrada de dados: "+erro1);
17.        }
18.        catch(NumberFormatException erro2){
19.            System.err.println("\n Deve entrar com um número: erro2--> "+erro2);
20.        }
21.        finally{
22.            System.out.println("\n Entrou no Finally");
23.        }
24.        System.out.println("\n O Valor de Entrada foi "+ s);
25.    }
26. }
```

Podemos adicionar o método:
`printStackTrace()`.
Veremos a “**pilha de erros**” e por quais
classes estas se propagaram:
erro1. `printStackTrace()`;
erro2. `printStackTrace()`;

Exceções (*persistência do sistema*)

- Bem até aqui vimos como trata exceções(através dos blocos try/catch/finally).

- Mas, mesmo tratando as falhas, **nosso código anterior, ao ser executado, acaba interrompendo sua execução.**

Neste caso, para evitar essa parada, poderíamos colocar parte do nosso código **dentro de um laço:**

Exceções (*persistência do sistema*)

```
1. import java.io.BufferedReader;
2. import java.io.IOException;
3. import java.io.InputStreamReader;

4. public class Excecao2 {
5.     public static void main(String arg[]){
6.         BufferedReader memoria = new BufferedReader(new InputStreamReader(System.in));
7.         System.out.println("\n Entrada de Dados");
8.         String s = "";
9.         boolean continua = true;
10.        while(continua){
11.            System.out.println("\nEntre com um valor: ");
12.            try{
13.                s = memoria.readLine();
14.                int numero = Integer.parseInt(s);
15.                System.out.println("\n O quadrado do valor de entrada eh: "+numero*numero);
16.                continua = false; //provendo a condição de parada do laço
17.            }
18.            catch(IOException erro1){
19.                System.out.println("\nErro de entrada de dados: "+erro1);
20.            }
21.            catch(NumberFormatException erro2){
22.                System.err.println("\n Deve entrar com um número: erro2--> "+erro2);
23.            }

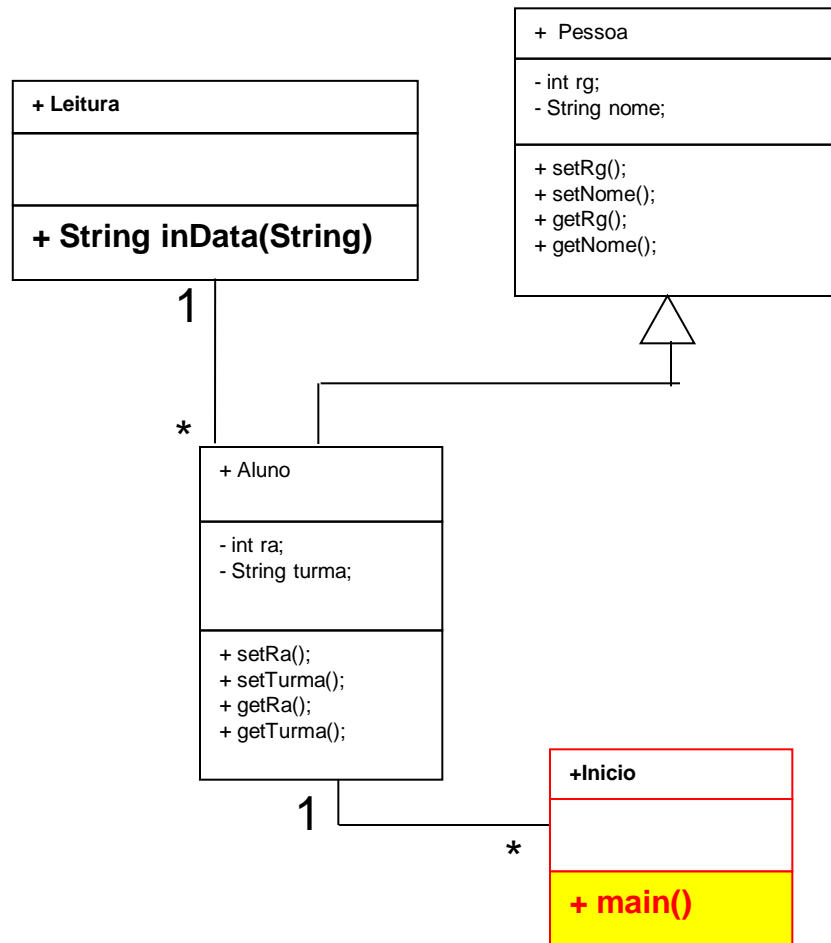
24.            finally{
25.                System.out.println("\n Entrou no Finally");
26.            }
27.        }

28.        System.out.println("\n O Valor de Entrada foi "+ s);
29.    }
30. }
```

Observe o código dentro do laço. Gerando assim um possível tratamento para o erro

Exceções (*estudo de caso*)

Observe o diagrama:



Exceções (*estudo de caso*)

Relembrando a **classe Leitura** (*recebe uma String e retorna uma String*)

```
1. import java.io.BufferedReader;
2. import java.io.IOException;
3. import java.io.InputStreamReader;

4. public class Leitura{
5.     public static String inData(String label){
6.         InputStreamReader c = new InputStreamReader(System.in);
7.         BufferedReader cd = new BufferedReader(c);
8.         System.out.print(label);
9.         String s = "";
10.        try{
11.            s = cd.readLine();
12.        }
13.        catch(IOException e){
14.            System.out.println("Erro de entrada");
15.        }
16.        return s;
17.    }
18. }
```

+ Leitura
+ String inData(String)

Exceções (*estudo de caso*)

Vamos criar a **classe Inicio** que teste o modelo

```
1. public class Inicio {  
2.     public static void main(String args[]){  
3.         Leitura l = new Leitura();  
4.         Pessoa pes = new Pessoa();  
5.         pes.setId(Integer.parseInt(l.inData("\nEntre com o ID <deve ser numero>: ")));  
6.         pes.setNome(l.inData("\n Entre com o nome: "));  
7.         System.out.println("\n ID....: "+pes.getId());  
8.         System.out.println("\n Nome..: "+pes.getNome());  
9.     }  
10. }
```

+Inicio
+ main()

Nota:

- Observe a linha 5 da **classe Inicio**.

- E se o usuário entrasse com uma letra? O que aconteceria?

R.: Ocorreria um erro e dispararia (na classe Inicio) a exceção: *NumerFormatException*.

-Como resolver?

-Uma das formas seria capturar e tratar esta exceção....

Exceções (*estudo de caso*)

Vamos a alterar a classe Inicio para tratar a exceção

```
1. public class Inicio {
2.     public static void main(String args[]){
3.         Leitura l = new Leitura();
4.         Pessoa pes = new Pessoa();
5.
6.         boolean testaInt = true;
7.         while(testaInt){
8.             try{
9.                 pes.setId(Integer.parseInt(l.inData("\nEntre com o ID <deve ser numero>: ")));
10.                testaInt = false;
11.            }
12.            catch(NumberFormatException e){
13.                System.out.println("\n O ID deve ser um numero inteiro");
14.            }
15.        }
16.
17.        pes.setNome(l.inData("\n Entre com o nome: "));
18.
19.        System.out.println("\n ID.....: "+pes.getId());
20.        System.out.println("\n Nome...: "+pes.getNome());
21.    }
22. }
```

+Inicio
+ main()

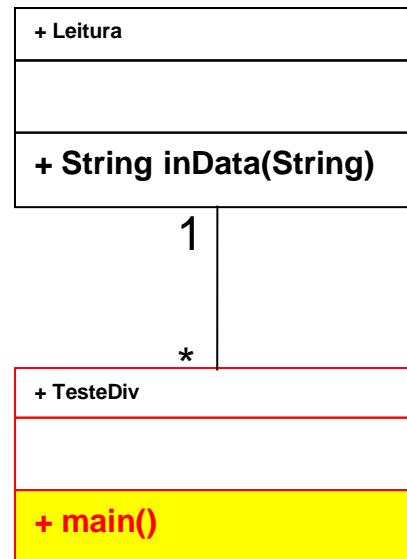
Exceções (*mais* estudo de caso)

Aplicação sobre uma operação de
divisão

Exceções (*estudo de caso*)

A **operação matemática de divisão** pode nos oferecer outro bom exemplo de uso do tratamento de exceções. Isto pelo fato de ter restrições:

- O divisor deve ser maior que 0 (zero). **Não existe divisão por 0 (zero):** dispara a exceção: **ArithmeticException: / by zero**



Exceções (*estudo de caso-DIVISÃO*)

Criando a classe **TesteDiv** para testar a exceção

```
1. public class TesteDiv {  
2.     public static void main(String arg[]){  
3.         Leitura l = new Leitura();  
4.         int numero = Integer.parseInt(l.inData("\nEntre com um numero: "));  
5.         int divisor = Integer.parseInt(l.inData("\nEntre com Divisor: "));  
6.         System.out.println("\n O resultador da divisão eh: "+numero/divisor);  
7.     }  
8. }
```

+TesteDiv

+ main()

Notas:

-A classe **Leitura** (*linha 3*) é nossa conhecida;

-**Teste** da classe **TesteDiv**:

-1º teste)_ Passe o valor 4 para *numero* e 2 para o *divisor*;

-2º teste)_ Passe o valor 4 para *numero* e **0 (zero)** para ***divisor***;

OBS.: Não se esqueça, como vimos em outros exemplos, as linhas 4 e 5 **também** deveriam ter seus tratamentos de exceção

Exceções (*estudo de caso-DIVISÃO*)

Vamos a **alterar** a classe TesteDiv **para tratar a exceção**

+TesteDiv
+ main()

```
1. public class TesteDiv {
2.     public static void main(String arg[]){
3.         Leitura l = new Leitura();
4.         boolean testeDiv = true;
5.         while(testeDiv){
6.             try{
7.                 int numero = Integer.parseInt(l.inData("\nEntre com um numero: "));
8.                 int divisor = Integer.parseInt(l.inData("\nEntre com Divisor: "));
9.                 System.out.println("\n O resultador da divisão eh: "+numero/divisor);
10.                testeDiv=false;
11.            }
12.            catch(ArithmeticException er_Div){
13.                System.err.println("\nNão existe divisão por zero: erro--> "+er_Div);
14.            }
15.            catch(NumberFormatException er_Num){
16.                System.err.println("\nO numero e/ou divisor devem ser numeros: erro--> "+er_Num);
17.            }
18.        }
19.    }
20. }
```

Exceções (*mais* estudo de caso)

Aplicação sobre uma operação
com *Array*

Exceções (*estudo de caso*)

Operação com Array:

- Não podemos “apontar” para um endereço que não exista no Array: O divisor deve disparar a exceção: **ArrayIndexOutOfBoundsException**:
- Veja e teste o código (**errado**):

```
1. public class TesteArray {  
2.     public static void main(String arg[]){  
3.         Leitura l = new Leitura();  
4.         int vetor[] = new int[5];  
5.         int valor = Integer.parseInt(l.inData("\n Informe o valor: " ));  
6.         int endereco = Integer.parseInt(l.inData("\n Informe o endereco do vetor: " ));  
7.         vetor[endereco]=valor;  
8.         System.out.println("\n Valor "+vetor[endereco]+" no endereco"+endereco);  
9.  
10.    }  
11. }
```

Nota:

Teste a classe TesteArray:

1º teste)_ Informe um valor para *valor* (linha 5), informe um endereço **válido** (≥ 0 e ≤ 4) para o vetor;
2º teste)_ Informe um valor para *valor* (linha 5), informe um endereço **inválido** (< 0 ou > 4) para o vetor;

Exceções (*estudo de caso*)

Operação com Array: **Corrigindo o código:**

```
1. public class TesteArray {
2.     public static void main(String arg[]){
3.         Leitura l = new Leitura();
4.         boolean testeArr = true;
5.         while(testeArr){
6.             try{
7.                 int vetor[] = new int[5];
8.                 int valor = Integer.parseInt(l.inData("\n Informe o valor: " ));
9.                 int endereco = Integer.parseInt(l.inData("\n Informe o endereco do vetor: " ));
10.                vetor[endereco]=valor;
11.                System.out.println("\n Valor "+vetor[endereco]+" no endereco "+endereco);
12.                testeArr=false;
13.            }
14.            catch(ArrayIndexOutOfBoundsException er_Array) {
15.                System.err.println("\n Endereço invalido para o Array: erro--> "+er_Array);
16.            }
17.            catch(NumberFormatException er_Num){
18.                System.err.println("\nO valor informado deve ser numero inteiro: erro--> "+er_Num);
19.            }
20.        }
21.    }
22. }
```


Exceções (*tipos*)

Verificada:

- Extende a classe Exception;

- **Verificada** durante a **compilação** :

Ex.: método `readLine()` da classe `BufferedReader` . Veja assinatura do método:

```
public String readLine() throws IOException
```

(<http://docs.oracle.com/javase/6/docs/api/> – em 20/06/2012)

- Obriga o programador a tratar a exceção. No caso do método acima:

throws: indica que este método poderá disparar uma exceção do tipo `IOException` (que deverá ser tratada)

Exceções (*tipos*)

Não Verificada:

- Extende a classe RuntimeException;

- Verificada durante a **execução**:

Ex.: Quando construímos uma expressão que dependerá de uma entrada de dados feita pelo usuário (logo, em tempo de execução) para sabermos se disparará ou não uma exceção: Observe a linha de código a seguir:

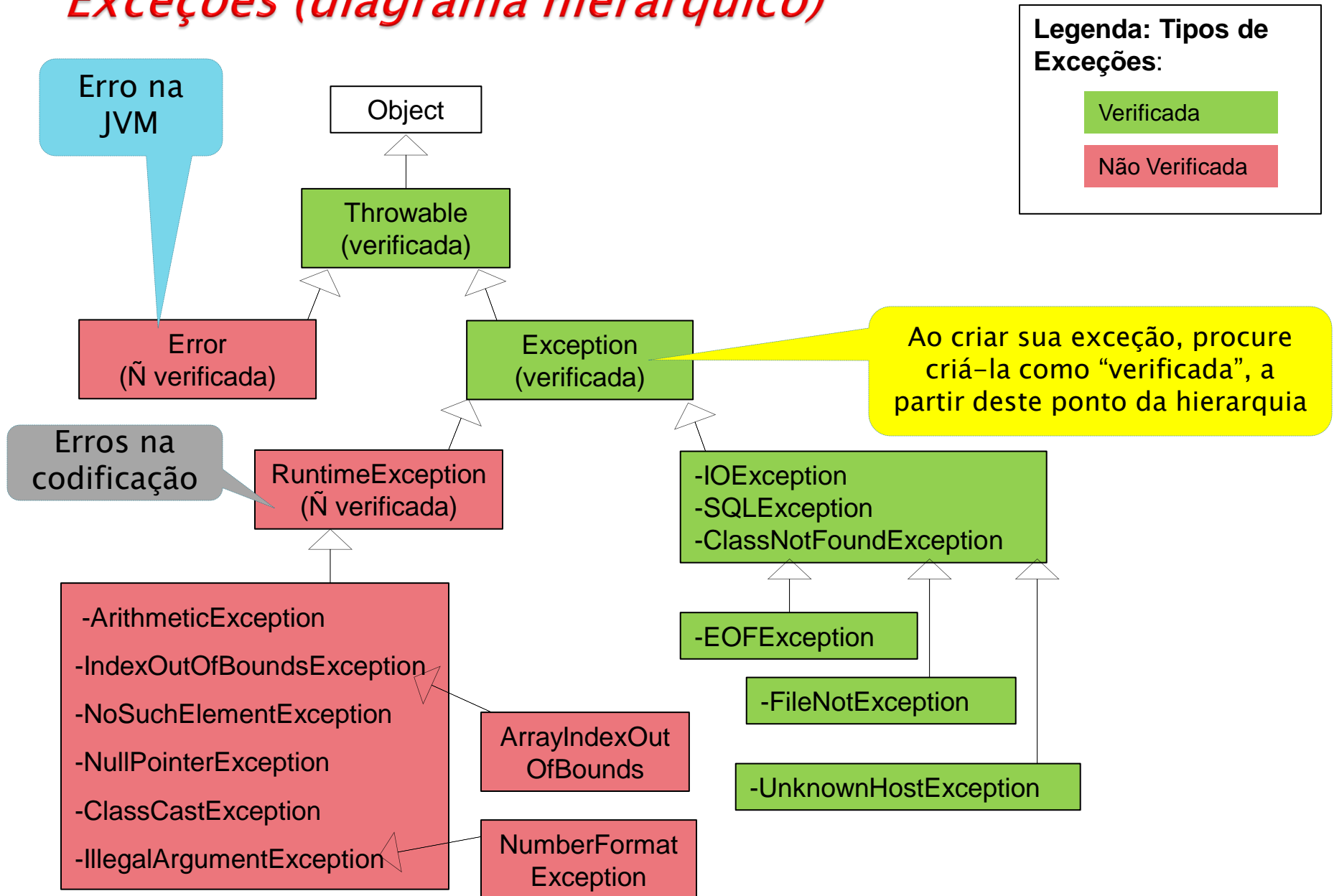
```
double resultado = numero/divisor;
```

Mas... E se o valor informado pelo usuário, para o *divisor*, for 0 (zero)?

Não existe divisão por 0.

Só descobriremos isso após o valor lançado pelo usuário, isto é, *em tempo de execução*

Exceções (diagrama hierárquico)



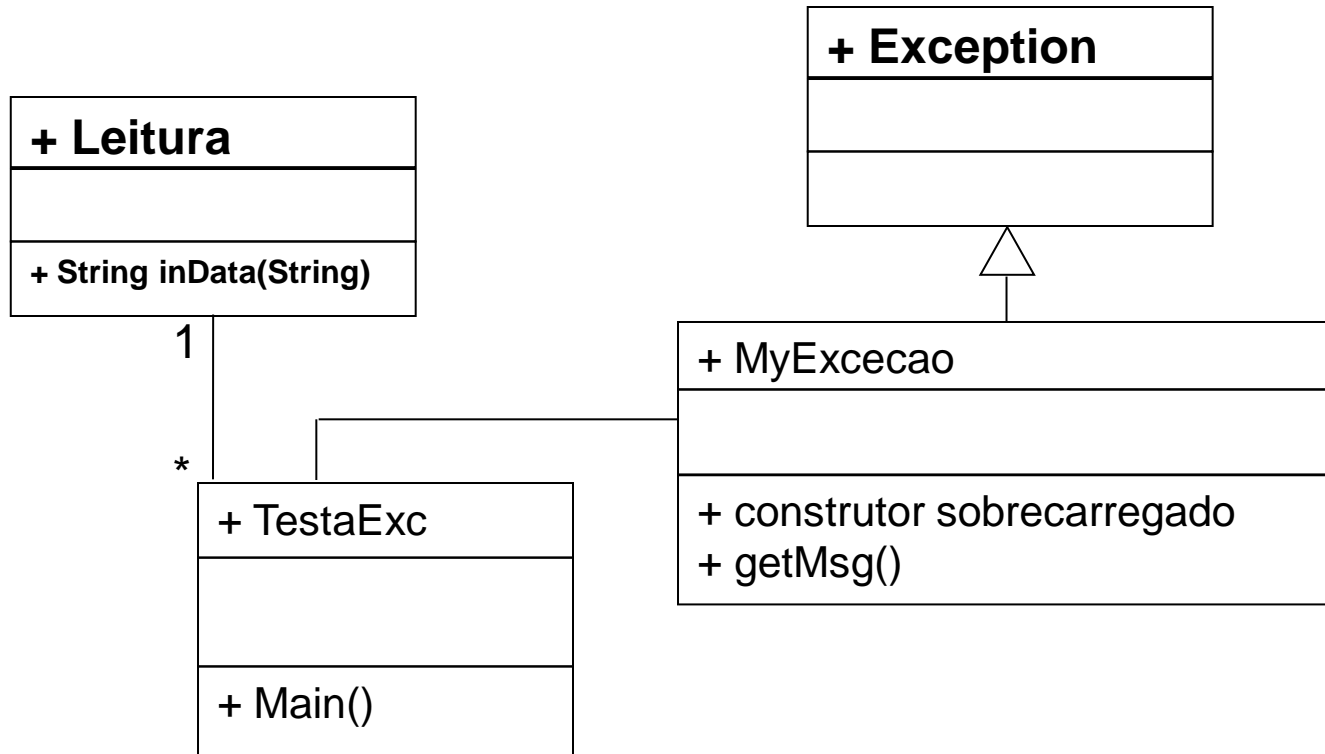
Exceções (*solução*)

De qualquer forma **devemos tratar** tanto as exceções **verificadas** quanto as **não-verificadas**

Exceções (*criando as nossas*)

Importante lembrar que, convencionalmente, criamos nossas exceções **extendendo a classe a classe Exception**. Dessa forma nossa exceção será do tipo verificada.

Exceções *(criando as nossas)*



Exceções (*criando as nossas*)

+ MyExcecao
+ construtor sobrecarregado + getMsg()

```
1. public class MyExcecao extends Exception{
2.     MyExcecao(int numero){ //construtor sobrecarregado
3.         System.out.println("\n Classe MyExcecao - Lancou valor Negativo");
4.     }
5.     public String getMsg(){
6.         return "Utilizou o metodo getMSg() da MyExcecao";
7.     }
8. }
```

Exceções (*criando as nossas*)

+ TestaExc
+ Main()

```
1. public class TestaExc{
2.
3.     public static void testaNumero(int numero) throws MyExcecao{
4.         if(numero>0) System.out.println("\n NUMERO POSITIVO: "+numero);
5.         else throw new MyExcecao(numero);
6.     }
7.
8.     public static void main(String arg[]){
9.         Leitura lei = new Leitura();
10.        int numero = Integer.parseInt(lei.inData("\n Entre com um valor: "));
11.        try{
12.            testaNumero(numero);
13.        }
14.        catch(MyExcecao me){
15.            System.err.println("\n Disparou minha excecao: "+me);
16.            System.out.println("\n Mensagem do getMsg() "+me.getMsg());
17.        }
18.    }
19. }
```


Exceções (*throws ou throw*)

throws: indica que aquele método poderá disparar uma exceção.

throw: usado para lançar a exceção propriamente dito.

Teste (o código anterior):

- Remova a cláusula “throws MyExcecao” (linha 3): tente compilar.

Resultado:

- Aparecerá mensagem informando que trata-se de uma exceção e que esta deverá ser tratada. E esta não se reportou à classe que a disparou.

Motivos:

- Durante a compilação o compilador detectou que na linha 5 (método testaNumero) é lançada uma **exceção do tipo verificada** (MyExcecao estende a classe Exception) logo este método deveria informar que pode e lança (**throw**) esta exceção.

Soluções:

- Tratar o erro aqui (try/catch);
- Informar ao método chamador que pode disparar a exceção (linha 5: *throws MyExcecao*) deixar que seja tratado neste (linhas: 10 a 16)