Tratamento de Exceções

Sumário

- Introdução;
- Tratamento de Exceções – Java;
- Hierarquia de Exceções – Java;
- Bloco `finally` - Java
- Bibliografia;

Introdução

- As exceções são na realidade erros durante a execução do programa;

- Esses erros são caudados por uma enormidade de circunstâncias, como:
  - Faltas de memória,
  - Impossibilidade de gravar;
  - Abrir um arquivo;
  - Atribuição de um valor impossível a um objeto;
  - Divisão por zero;

Introdução

- Se ocorrer um desses erros e não implementarmos o tratamento de exceções, o programa terminará abruptamente;

- *A solução é tratar exceções*:
  - Se uma função necessita enviar uma mensagem de erroa para a função chamadora, "dispara" um objeto representando o erro para fora dela.
  - Se a função chamadora não capturar e tratar o erro, o objeto irá para a chamada de nível superior e assim por diante, até que alguém capture o erro;

Tratamento de Exceções - Java

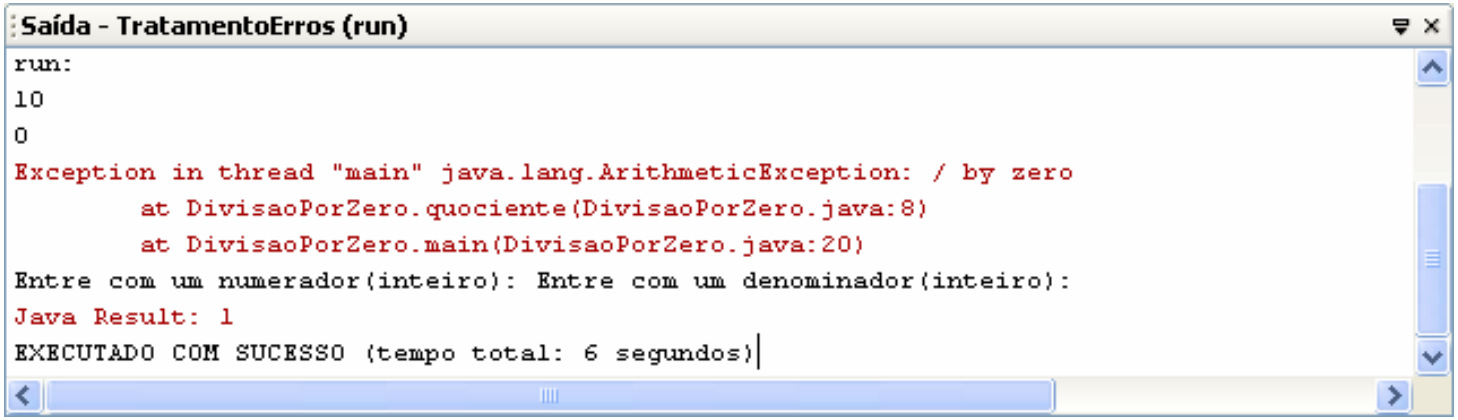
- Vejamos o tratamento de exceções no Java:

Bem-vindo | DivisaoPorZero.java

```
1  import java.util.Scanner;
2
3  public class DivisaoPorZero
4  {
5     // demonstra o lançamento de uma exceção quando ocorre uma divisão por zero
6     public static int quociente( int numerador, int denominador )
7     {
8        return numerador / denominador; // possivel divisão por zero
9     }
10
11    public static void main( String args[] )
12    {
13       Scanner scanner = new Scanner( System.in );
14
15       System.out.print( "Entre com um numerador(inteiro): " );
16       int numerador = scanner.nextInt();
17       System.out.print( "Entre com um denominador(inteiro): " );
18       int denominador = scanner.nextInt();
19
20       int resultado = quociente( numerador, denominador );
21       System.out.printf(
22          "\nResultado: %d / %d = %d\n", numerador, denominador, resultado );
23    } // fim de main
24 }
```

Tratamento de Exceções - Java

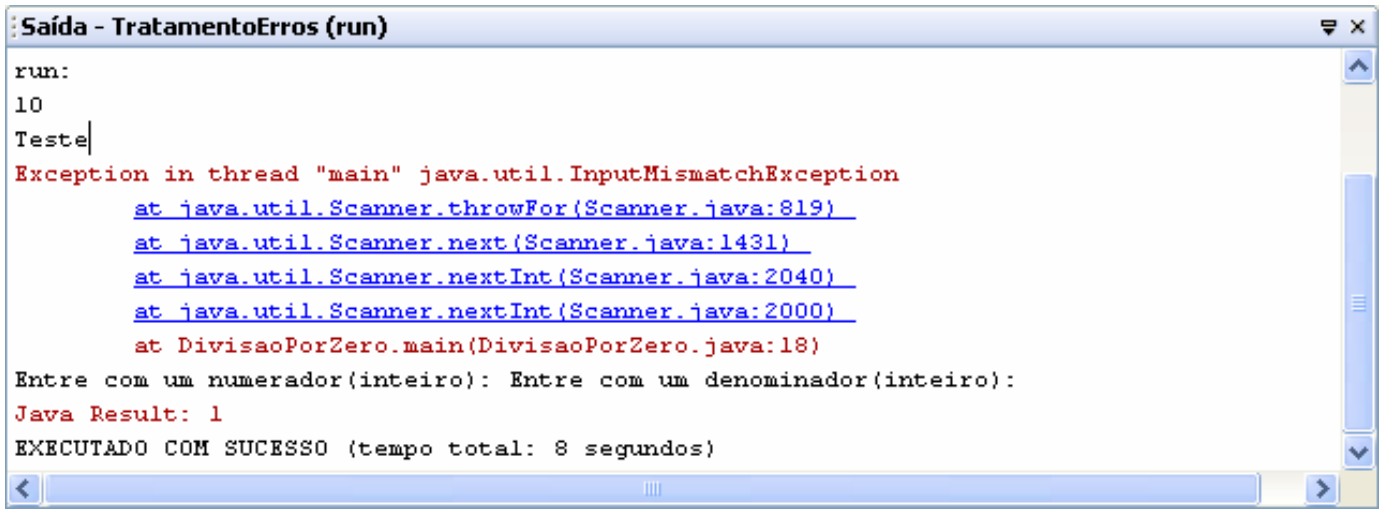
- Supondo uma entrada com 0 (zero) no denominador, vejamos a saída no console:



  - Exceção detectada no método main;

Tratamento de Exceções - Java

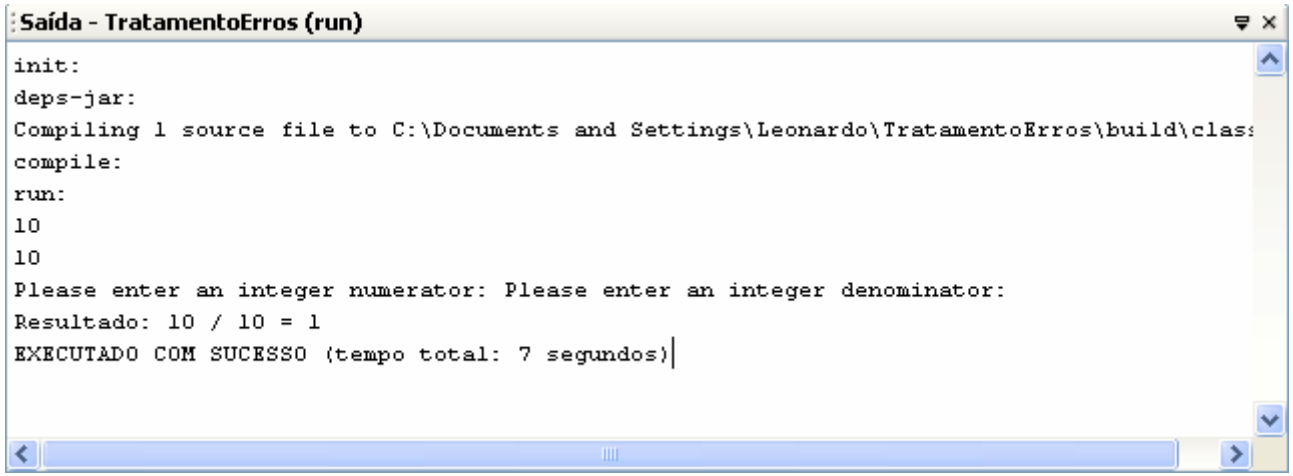
- Supondo uma entrada com "teste" (string) no denominador, vejamos a saída no console:



  - Exceção detectada no método main;

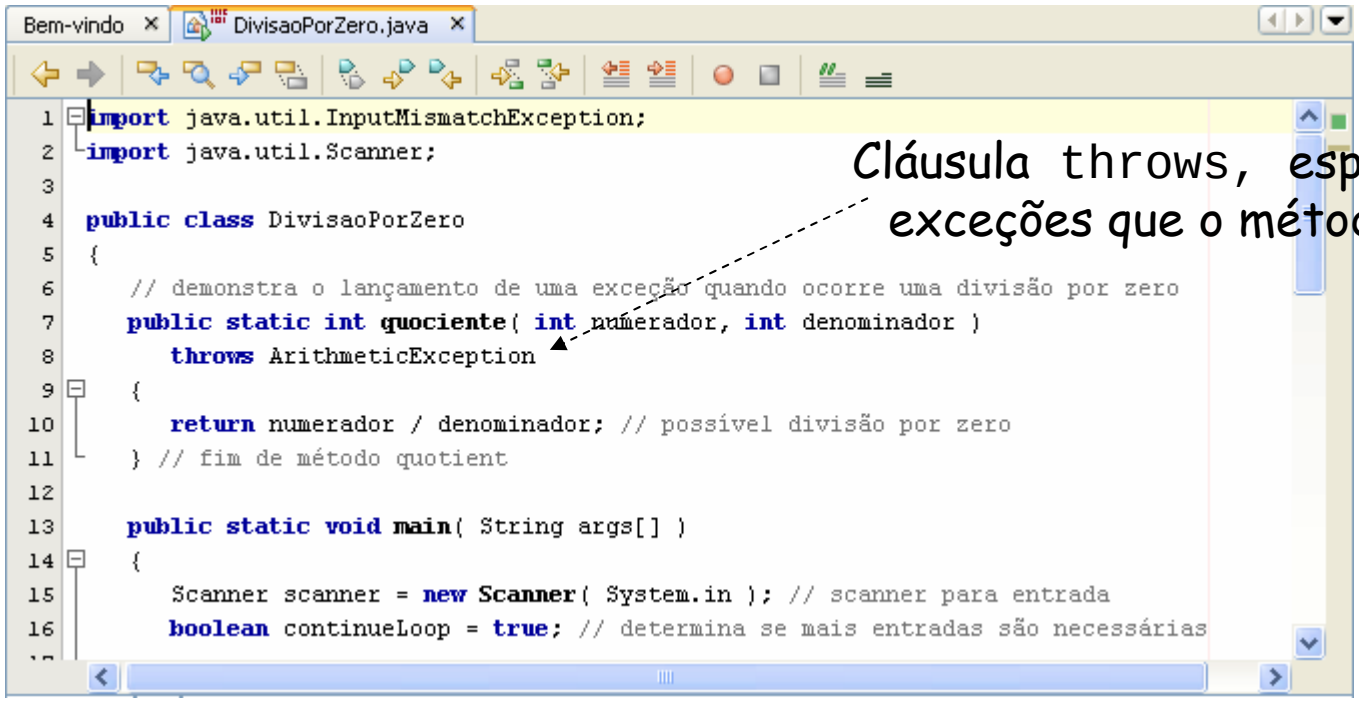
Tratamento de Exceções - Java

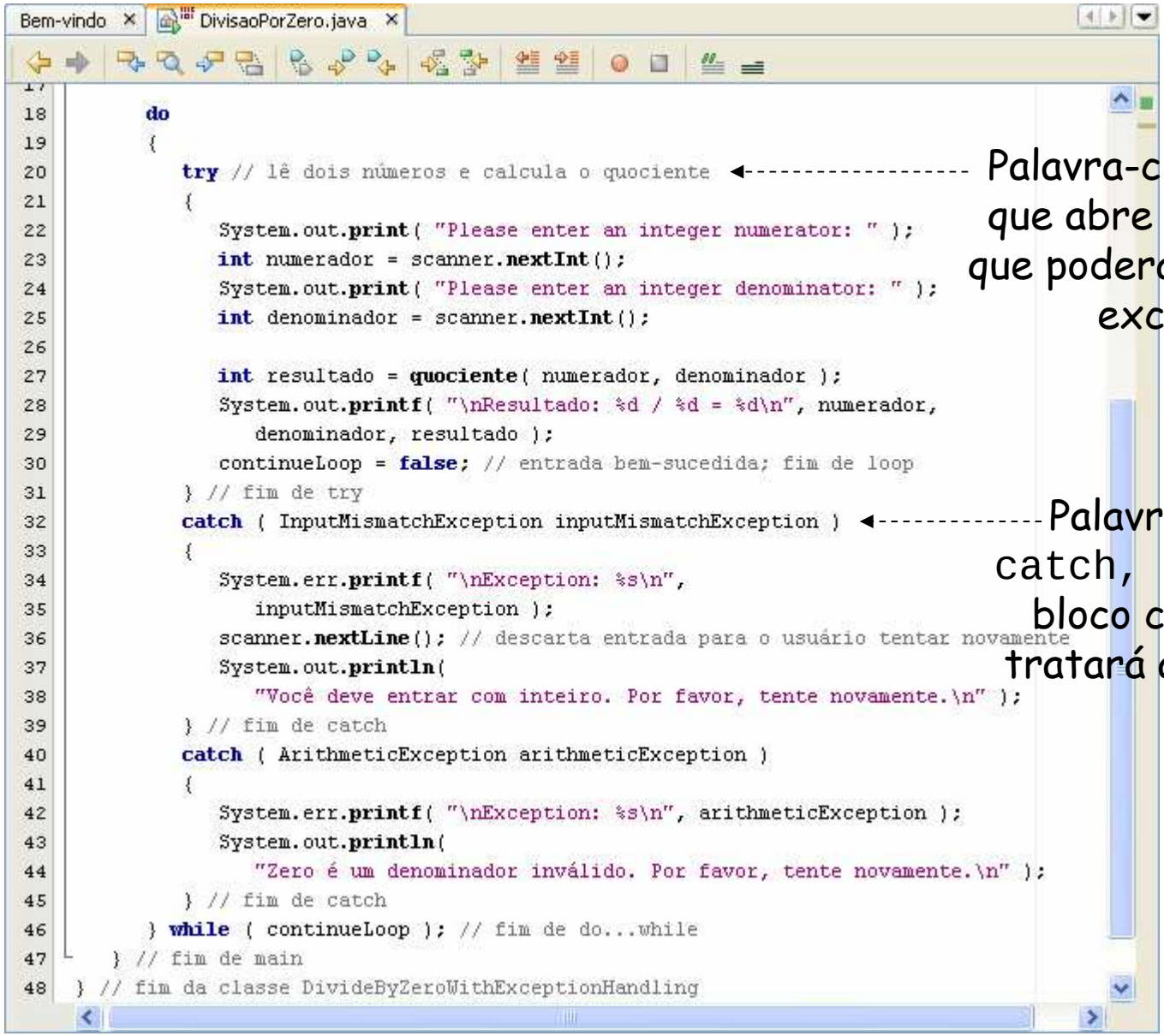
- É óbvio que o programa "roda" quando as entradas forem válidas, vejamos:



Tratamento de Exceções - Java

- Vejamos como tratar essas exceções para permitir que o programa conclua sua execução normalmente:





Palavra-chave `try`, que abre o bloco try que poderá lançar uma exceção.

Palavra-chave `catch`, que abre o bloco catch que tratará a execção.

Bem-vindo | DivisaoPorZero.java

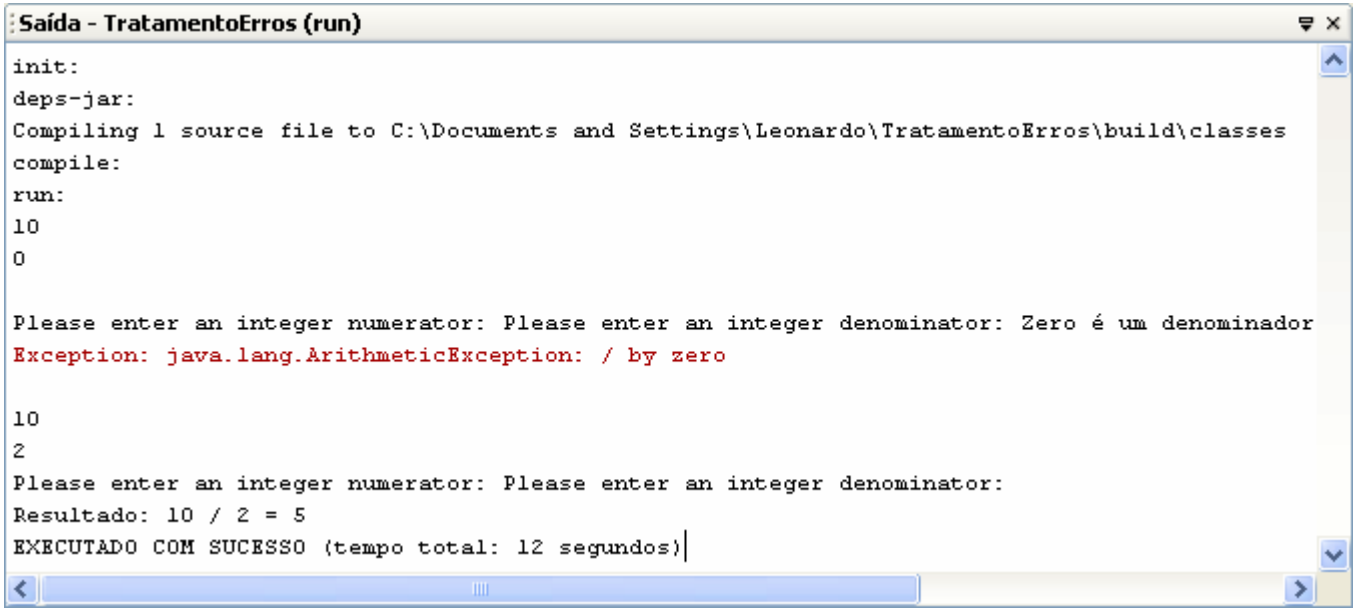
```
18       do
19       {
20          try // lê dois números e calcula o quociente
21          {
22             System.out.print( "Please enter an integer numerator: " );
23             int numerador = scanner.nextInt();
24             System.out.print( "Please enter an integer denominator: " );
25             int denominador = scanner.nextInt();
26
27             int resultado = quociente( numerador, denominador );
28             System.out.printf( "\nResultado: %d / %d = %d\n", numerador,
29                denominador, resultado );
30             continueLoop = false; // entrada bem-sucedida; fim de loop
31          } // fim de try
32          catch ( InputMismatchException inputMismatchException )
33          {
34             System.err.printf( "\nException: %s\n",
35                inputMismatchException );
36             scanner.nextLine(); // descarta entrada para o usuário tentar novamente
37             System.out.println(
38                "Você deve entrar com inteiro. Por favor, tente novamente.\n" );
39          } // fim de catch
40          catch ( ArithmeticException arithmeticException )
41          {
42             System.err.printf( "\nException: %s\n", arithmeticException );
43             System.out.println(
44                "Zero é um denominador inválido. Por favor, tente novamente.\n" );
45          } // fim de catch
46       } while ( continueLoop ); // fim de do...while
47    } // fim de main
48 } // fim da classe DivideByZeroWithExceptionHandling
```

O bloco try mostra o código que não pode ser executado se ocorrer uma exceção.

Um bloco catch inicia-se com a palavra-chave catch e é seguido por um parâmetro de exceção e um bloco entre chaves.

Tratamento de Exceções - Java

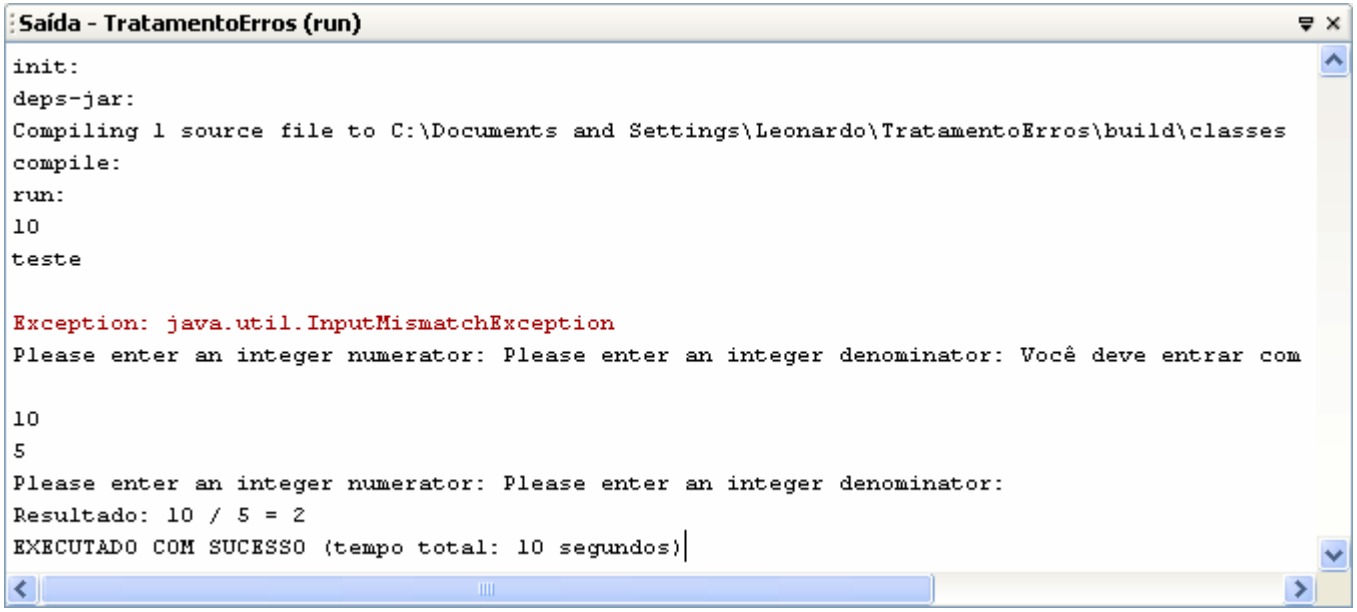
- Vejamos as saídas no console para o programa anterior com entrada 0 (zero) no denominador:



  - Divisão por zero. Disparou a exceção Aritmética;

Tratamento de Exceções - Java

- Vejamos as saídas no console para o programa anterior com entrada "teste" (string) no denominador:



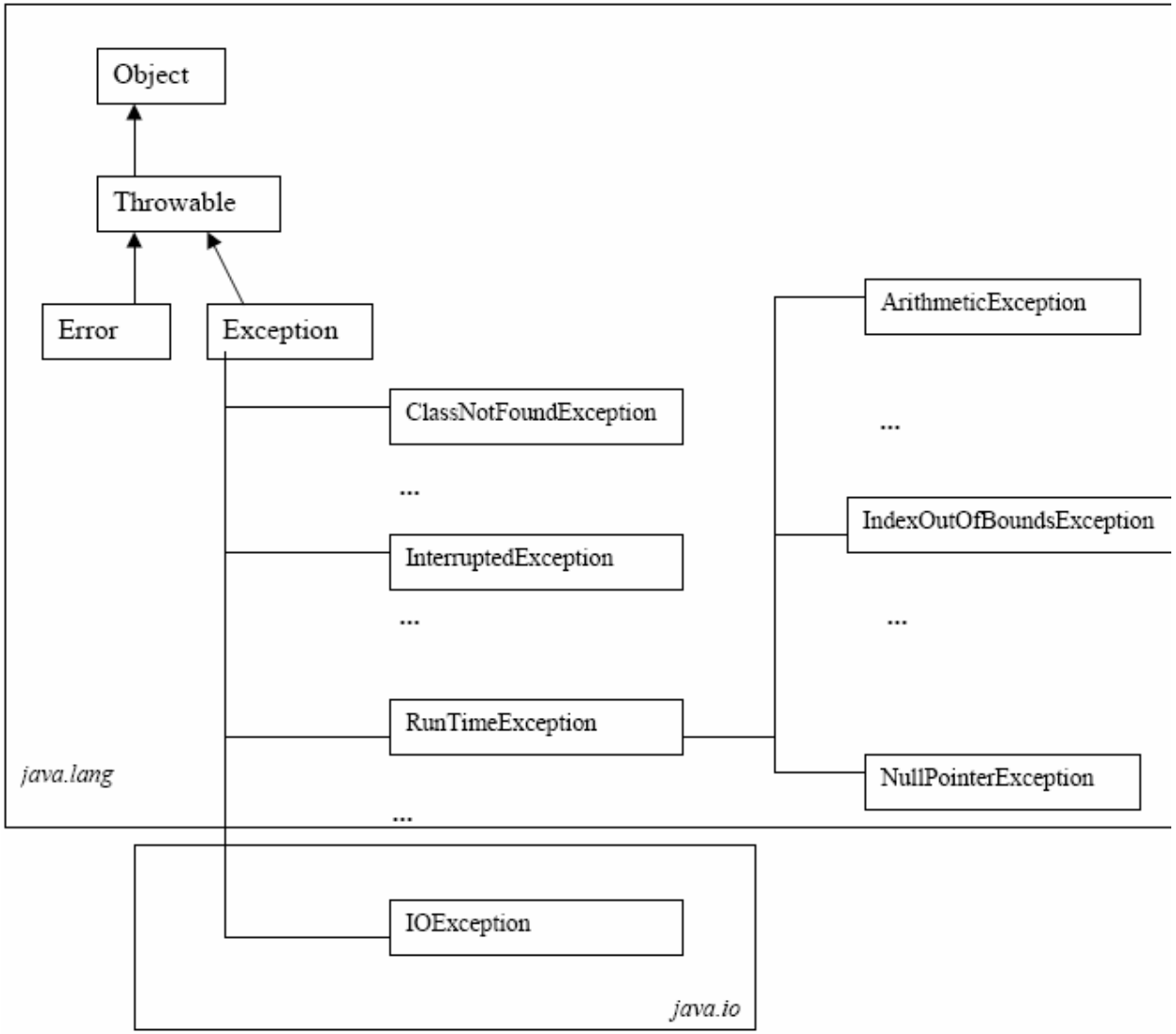
  - Entrada inesperada. Disparou a exceção de Entrada;

Tratamento de Exceções - Java

- Considerações:
  - A cláusula try (tentar) tenta executar um bloco de código;
  - A cláusula throws (jogar para fora) dispara as exceções do tipo Aritméticas ocorridas dentro do método. Ela aparece depois da lista de parâmetros do método e antes do corpo do método. A cláusula throws indica ao resto do programa que o método pode lançar uma `ArithmeticException`;
  - A cláusula cacth (pegar) captura e trata as exceções geradas pelo bloco try e deve seguir imediatamente o bloco try;

Hierarquia de Exceções - Java

- Todas as classes de exceção do Java herdam, direta ou indiretamente, da classe **`Exception`**.



Hierarquia de Exceções - Java

- Através da hierarquia de exceções é possível verificar exceções verificáveis pelo compilador e exceções não verificáveis:

- Exceções não verificáveis: são os objetos das classes RuntimeException, Error e respectivas sublcasses;
  - RuntimeException: Exceções cuja ocorrência é difícil de ser verificável pelo programador;
  - Error: Erros não recuperáveis;

Hierarquia de Exceções - Java

- Para cada Exceção verificável, o método onde essa exceção pode ocorrer deve:
  - Prever i tratamento da exceção (try – catch) ou
  - Lançar a exceção, através da cláusula throws para que seja tratada no método invocador ou em outro mais externo;

- Vejamos um exemplo:

Classe Bomba herdando a própria classe de Exceções (Exception)

```
1  public class Bomba extends Exception {
2      public Bomba (){
3          super();
4      }
5      public Bomba ( String s ){
6          super ( s );
7      }
8  }
```

Hierarquia de Exceções - Java

- Continuação do Exemplo Anterior, Classe Teste1:

```
1  public class Testel {
2      public static void explode () throws Bomba {
3          throw new Bomba();
4      }
5      public static void main (String []args) {
6          try {
7              explode();
8              }
9          catch ( RuntimeException e ) {
10             System.out.println ("RuntimeExplode"+ e.getMessage());
11         }
12         catch ( Bomba b ) {
13             System.out.println ( "Bomba" );
14         }
15     }// main
16 }//Testel
```

Indicativo de que o método explode dispara exceções do tipo Bomba;

Geração explicita de uma exceção definida ou não pelo programa-dor;

Bloco finally - Java

- O bloco **finally** é opcional e é colocado após o último bloco catch para tratar vazamento de recurso;

- O bloco **finally** será executado nas seguintes situações:
  - Se uma exceção for lançada no bloco try correspondente ou quaisque de seus blocos catch;
  - Se um bloco try fechar utilizando `return`, `break` ou `continue`;

- O bloco **finally** não será executado se o aplicativo fechar antes de um bloco `try` chamando o método **`System.exit`**;

Bloco finally - Java

- Como o bloco **finally** é quase sempre usado, sua finalidade maior é desalocar recursos(arquivos, conexões de banco de dados e de rede, etc) alocados em blocos try;

- O bloco **finally** consiste na palavra-chave **finally**, seguida pelo código entre chaves;

- Vejamos um exemplo em que o bloco **finally** é invocado;

Bloco finally - Java

- Exemplo:

```
1  public class Testel {
2      public static void explode () throws Bomba{
3          throw new NullPointerException();
4      }
5      public static void main (String []args) {
6          try {
7              explode();
8              }
9          catch ( Bomba e ) {
10             System.out.println ("Bomba");
11         }
12         finally {
13             System.out.println ("Exceção não capturada");
14         }
15     }
16 }
```

Bloco **finally**

Herança de Exceções - Java

- Um método que sobrepõe ("overrides") outro não pode declarar lançar mais exceções do que o método que é sobreposto;

```
1  class C1 {
2      public void m2() throws Exception {
3          System.out.println ("Método 2 da classe C1");
4      }
5  }
6  class C2 extends C1 {
8      public void m2() throws InterruptedException, ClassNotFoundException{
9          System.out.println ("Método 2 da classe C2");
10     }
```

Herança de Exceções - Java

- Considerações:
  - Cada exceção declarada em m2 da classe C2 tem que ser do mesmo tipo (classe) de uma exceção lançada em m2 de C1 ou de um seu subtipo (subclasse).
  - No método m2 da classe C1 tem que ser lançada a mesma exceção que em m2 de C2, ou uma exceção de uma sua superclasse;

Bibliografia

- Deitel, H. M. & Deitel, P. J. Java*: como programar,* Editora Bookman. 6ª ed. São Paulo: 2005.