Exceções

Instituto Federal do Espirito Santo – Ifes
Campus Colatina – Programação Orientada a Objetos I
Professor: Giovany Frossard Teixeira

Exceções

- Um método pode detectar uma falha mas não estar apto a resolver sua causa, devendo repassar essa função a quem saiba.

- Se introduzirmos o tratamento de falhas ao longo do fluxo normal de código, podemos estar comprometendo muito a inteligibilidade, ou seja, pode ficar muito complexo o entendimento do código.

Exceções

- Diz-se que uma exceção é *lançada* para sinalizar alguma falha.

- O lançamento de uma exceção causa uma interrupção abrupta do trecho de código que a gerou.

- O controle da execução volta para o primeiro trecho de código (na pilha de chamadas) apto a tratar a exceção lançada.

Exceções

- As linguagens OO tipicamente dão suporte ao uso de exceções.

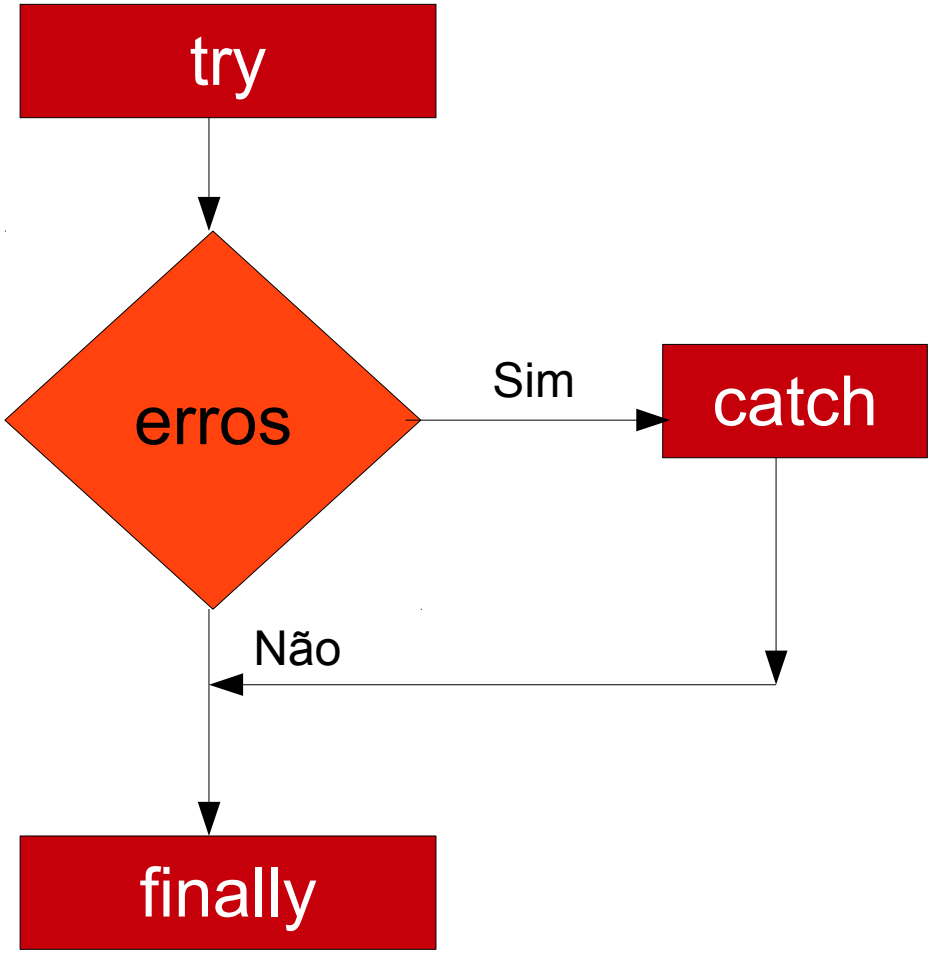
- Para usarmos exceções precisamos de saber:
  - Como definir uma exceção.
  - Como chamar uma exceção definida anteriormente.
  - Uma forma de tratar uma exceção chamada.

Exceções

- Java dá suporte ao uso de exceções:
  - As exceções em Java são definidas por classes.
  - São chamadas pelo comando *throw*.
  - São tratadas pelo comando *try-catch-finally*.

Comandos try-catch-finally

- O comando *try – catch- finally* funciona da seguinte forma:
  - Tenta-se executar o que o código no bloco try, senão ocorrer nenhum erro todo código do bloco é executado.
  - Se algum erro ocorrer um catch deverá ser acionado.
  - Independente do código ter sido executado com sucesso ou não o bloco finally e executado no final do processo.



Exemplo try-catch-finally

```
package excecoes;

public class Excecoes01 {

    public static void main(String[] args) {

        try
        {
            int a = Integer.parseInt(args[0]);

            int b = Integer.parseInt(args[1]);

            System.out.println("Divisao = " + ( a / b ) );
        }
        catch ( ArithmeticException erro )
        {
            System.out.println("Erro de divisão por zero");
        }
        catch ( ArrayIndexOutOfBoundsException erro )
        {
            System.out.println("Numero de argumentos  invalido");
        }
        catch ( NumberFormatException erro)
        {
            System.out.println("Digite apenas numeros inteiros");
        }
        finally
        {
            System.out.println("Final da execução !");
        }
    }
}
```

Comandos throw e thorws

- O comando **throw** é utilizado desejamos chamar uma exceção.
- O comando **throws** é utilizado quando queremos ignorar a chamada de uma exceção.

Chamando uma exceção

```
package excecoes;

public class Excecoes02 {

   public static void main(String[] args) {

      try

      {

         int a = 1;

         System.out.println("Como aprender ");

         if ( a == 1)

            throw new Exception("Minha excecao");

         System.out.println("será que chega aqui ");


      }

      catch ( Exception erro)

      {

         System.out.println(" a linguagem Java");

      }

   }

}
```

Ignorando a chamada de uma possível exceção

```
package excecoes;

import java.io.*;

public class Excecoes03 {

   public static void main(String[] args)
     throws IOException {

       BufferedReader dado;

       System.out.println("Entre com seu
       nome");

       dado = new BufferedReader( new
       InputStreamReader( System.in));

       System.out.println("Seu nome é" +
      dado.readLine());

   }

}
```

Obs. A linha:

```
dado = new BufferedReader( new
    InputStreamReader( System.in));
```

Pode gerar uma exceção do tipo IOException, mas devido ao uso do comando throws, essa exceção não será tratada pelo método main (na realidade a exceção seria disparada para o método que tivesse chamado o main, mas como ninguém chamou o main a exceção não será tratada).

Métodos getMessage e printStackTrace da classe Exception

- getMessage: retorna a String armazenada em uma exceção.

- printStackTrace: retorna o tipo da exceção gerada e em que linha do programa ocorreu o erro.

Exemplo – getMessage e printStackTrace

```
package excecoes;

public class Excecoes04 {

  public static void main(String[] args) {

      int x = 10, y = 0, z = 0;

      try

      {

         // gera uma exceção aritmética de
        //  divisão por zero

         z = x / y;

         z = 0;

      }

      catch (Exception erro)

      {

         // mostra a mensagem de erro

         System.out.println(erro.getMessage());

         // mostra a exceção e a linha onde
         // ocorreu o erro

         erro.printStackTrace();

      }

   }

}
```

Criando nossas próprias Exceções

- Para criamos exceções basta que a classe que estejamos implementando descenda da classe Exception.
- Podemos re-implementar os métodos definidos na hierarquia da classe Exception como por exemplo o método getMessage. Dando assim o comportamento de polimorfismo às classes descendentes.
- O exemplo dos próximos slides vai ilustrar essa possibilidade.

Exemplo – Definindo Exceções

```
package criando_excecoes;


public class ImparException
extends Exception{
    public String getMessage() {
        return "O número é ímpar";
    }
}
```

```
package criando_excecoes;


public class ParException
extends Exception {
    public String getMessage() {
        return "O número é par";
    }
}
```

Exemplo – Definindo Exceções

```
package criando_excecoes;


public class NumeroBicomposto {

    int par, impar;


    private void setPar(int numero) throws
    ImparException{

        if ( (numero % 2) != 0 ){

            throw new ImparException();

        }

        this.par = numero;

    }


    private void setImpar(int numero) throws
    ParException{

        if ( (numero % 2) == 0 ){

            throw new ParException();

        }

        this.impar = numero;

    }

    NumeroBicomposto( int par, int impar) throws
    ParException, ImparException{

        this.setPar(par);

        this.setImpar(impar);

    }

}
```

Exemplo – Definindo Exceções

```
package criando_excecoes;

public class UsaNumeroBiComposto {

   public static void main(String[] args) {

       NumeroBicomposto num1;

        try {

            num1 = new NumeroBicomposto(10, 16);

        } catch (ParException ex) {

            System.out.println(ex.getMessage());

        } catch (ImparException ex) {

            System.out.println(ex.getMessage());

        }

   }

}
```

Dúvidas?