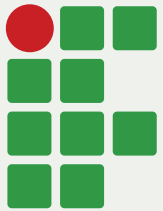
Linguagem de Programação 1

Tratamento de Exceção

Prof. Fábio José Rodrigues Pinheiro

Novembro de 2021



INSTITUTO FEDERAL
Catarinense
Campus Videira

Tratamento de Exceções

Um simples programa Java

```
1  import java.util.Scanner;
2  public class Principal{
3    public static void main(String args[]){
4      int[] vetor = new int[10];
5      Scanner teclado = new Scanner(System.in);
6
7      System.out.print("Entre com o número: ");
8      int numero = teclado.nextInt();
9
10     System.out.print("Em qual posição ficará?: ");
11     int posicao = teclado.nextInt();
12
13     vetor[posicao] = numero;
14   }
15 }
```

- O código acima é seguro? Executará sempre sem problemas?

Tratamento de Exceções

Exceção

Evento que indica a ocorrência de algum problema durante a execução do programa

Tratamento de Exceções

Exceção

Evento que indica a ocorrência de algum problema durante a execução do programa

Tratamento de exceções

Permite aos programas **capturar** e **tratar erros** em vez de deixá-los ocorrer e assim sofrer com as consequências

- Utilizado em situações em que o sistema pode recuperar-se do mau funcionamento que causou a exceção

Em Java

- Em Java, o **tratamento de exceções** foi projetado para situações em que um método detecta um erro e é incapaz de lidar com este
- Quando um erro ocorre é criado um **objeto de exceção**
  - Contém informações sobre o erro, incluindo seu tipo e o estado do programa quando o erro ocorreu.

Desenvolvendo códigos com tratamento de exceção

- O primeiro passo para tratar exceções é colocar todo o código que possa vir a disparar uma exceção dentro de um bloco **try...catch**
- As linhas dentro do bloco **try** são executadas sequencialmente
  - Se ocorrer uma exceção, o fluxo de execução passa automaticamente para um bloco **catch**
  - Se não ocorrer exceção, então o fluxo de execução passa para a próxima linha após os blocos **catch**

```
1 try{
2   // instruções que possam vir a disparar uma exceção
3 }catch(Tipo da excecao){
4   // instruções para lidar com a exceção gerada
5 }
6 System.out.println("continuando o programa");
7
```


Exemplo 1: Tipo misturado (int *vs* String)

```
1  public static void main(String[] args){
2      Scanner ler = new Scanner(System.in);
3      int a, b;
4
5      try{
6          a = ler.nextInt();
7          b = ler.nextInt();
8
9          double res = (double) a / b;
10
11         System.out.println(a + " dividido por " + b + " = " + res);
12
13     }catch(Exception e){
14         System.err.println("Ocorreu o erro: " + e.toString());
15     }
16     System.out.println("Fim do programa");
17 }
```


Determinando o tipo da exceção

- Para cada bloco **try** é possível ter um ou mais blocos **catch**
  - Cada bloco **catch** é responsável por tratar um tipo específico de exceção
- No exemplo anterior, o bloco **catch** capturava a exceção mais genérica possível em Java
  - Capturava objetos da `classe Exception`
- Em Java existem diversas outras classes para exceções, todas herdam da Exception
  - `ClassNotFoundException, ArithmeticException, FileNotFoundException, ...`

Determinando o tipo da exceção

- Para cada bloco **try** é possível ter um ou mais blocos **catch**
  - Cada bloco **catch** é responsável por tratar um tipo específico de exceção
- No exemplo anterior, o bloco **catch** capturava a exceção mais genérica possível em Java
  - Capturava objetos da `classe Exception`
- Em Java existem diversas outras classes para exceções, todas herdam da Exception
  - `ClassNotFoundException, ArithmeticException, FileNotFoundException, ...`

Sequência de blocos catch

Deve-se colocar a captura de exceções específicas antes das exceções mais genéricas

Capturando exceções específicas

```
1  public static void main(String[] args){
2      int[] numeros = new int[10];
3      Scanner teclado = new Scanner(System.in);
4    try{
5      System.out.print("Entre com o número: ");
6      int numero = teclado.nextInt();
7      System.out.print("Em qual posição ficará?: ");
8      int posicao = teclado.nextInt();
9
10     numeros[posicao] = numero;
11
12     }catch(java.util.InputMismatchException e){
13         System.err.println("Erro: Valores nao inteiros. ");
14     }catch(java.lang.ArrayIndexOutOfBoundsException e){
15         System.err.println("Erro: estouro de limite de vetor ");
16     }catch(Exception e){
17         System.err.println("Ocorreu o erro: " + e.toString());
18     }
19     System.out.println("Fim do programa");
20 }
```

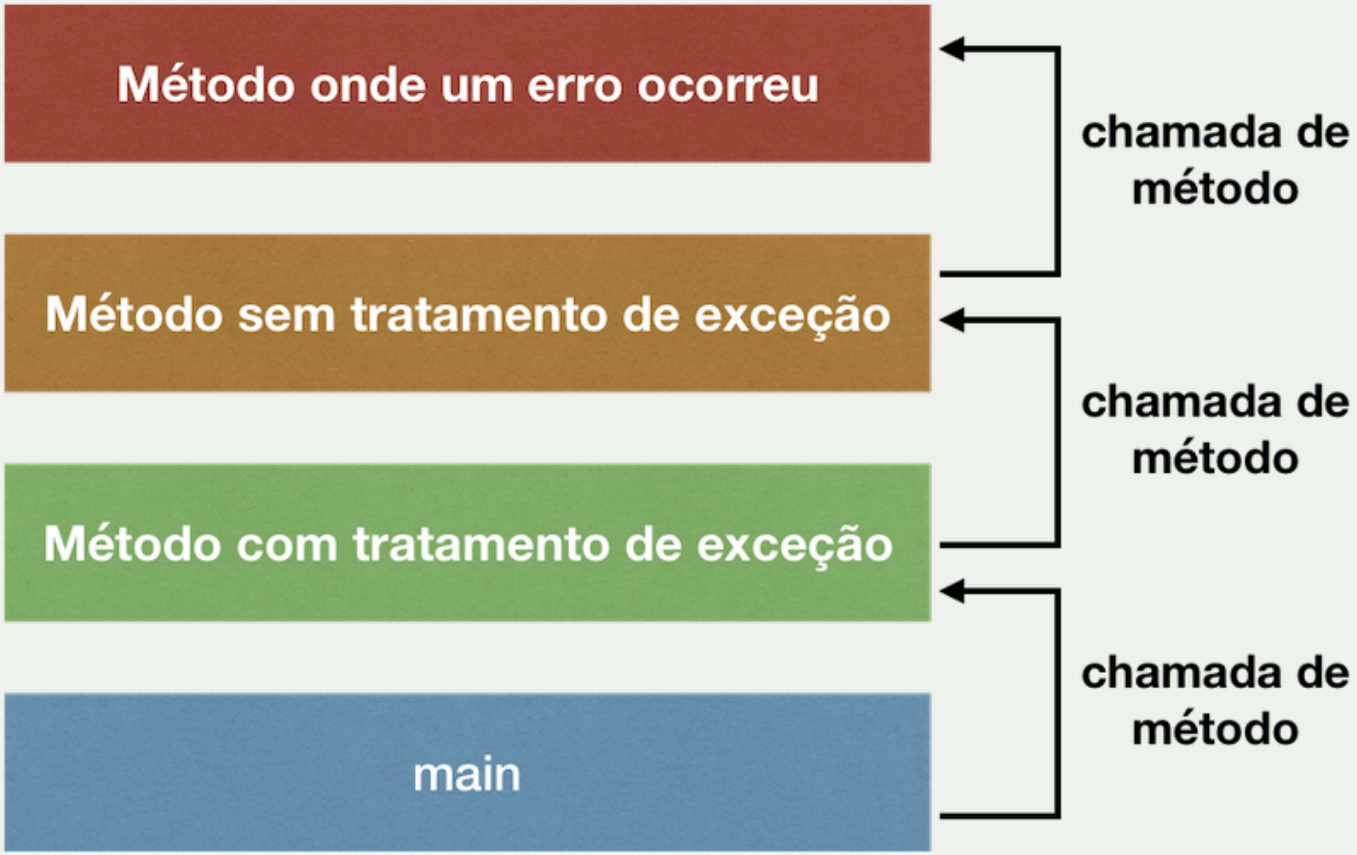

Bloco finally

- As linhas dentro do bloco **finally** sempre serão executadas, independente de ocorrer exceção ou não
- Códigos dentro dos blocos **catch** só serão executados somente se for lançada alguma exceção
- As linhas dentro do bloco **finally** sempre serão executadas, mesmo se houver instruções `return`, `continue` ou `break` dentro do bloco **try**
- O bloco **finally** é o local ideal para liberar recursos que foram adquiridos anteriormente

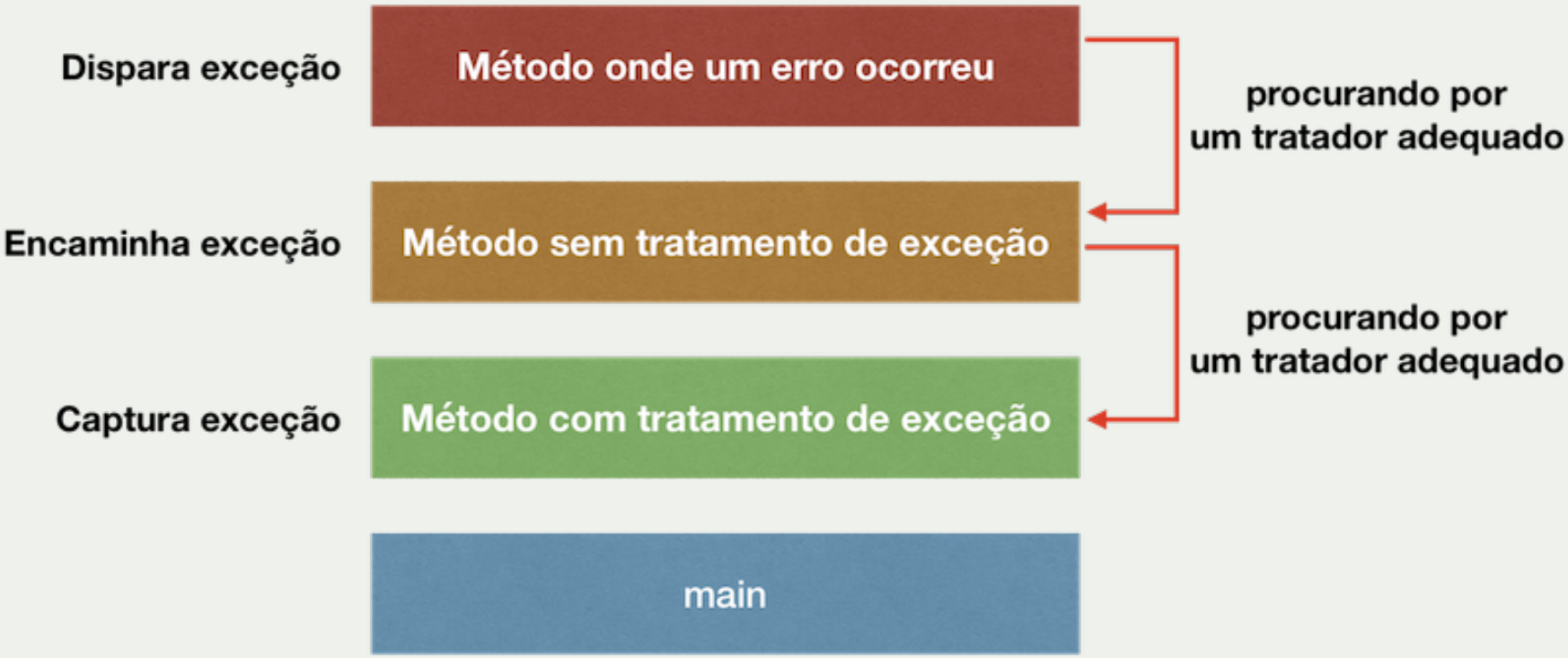
Bloco finally – execução mesmo diante de uma instrução return

```
1  public int lerTeclado(){
2    Scanner teclado = new Scanner(System.in);
3    do{
4      try{
5        System.out.print("Entre com um número: ");
6        int num = teclado.nextInt();
7        return num;
8      }catch(Exception e){
9        System.err.println("Erro: " + e.toString());
10     }finally{
11       System.out.print("Sempre será executada");
12     }
13     System.out.print("Se o return for executado, então essa linha não
       será");
14   }while(true);
15 }
```


Disparando e capturando exceções



Disparando e capturando exceções



Encaminhando e disparando exceção

- Encaminhando exceções para o método que o invocou

```
1 public void escreverArquivoNoDisco() throws IOException {
2 .....
3 }
```


Exercício

```
1  public class Exercicio{
2    private Scanner ler = new Scanner(System.in);
3
4    public int lerNumero(){
5      System.out.print("Entre com um numero: ");
6      return ler.nextInt();
7    }
8
9    public double divisao(int a, int b){
10     return (double) a / b;
11   }
12 }
```

- Garanta que o *lerNumero* irá progredir se o usuário não entrar com um *int*
- Garanta que o `divisao` dispare uma exceção para o método que o invocou se **b** for igual a zero

Agora é Java!

Classe MaskFormatter

- A classe `MaskFormatter` é usada para formatar e editar Strings
- A máscara indica quais são os caracteres válidos que podem estar contidos na String

#	Qualquer número
U	Qualquer caractere e todos serão convertidos para maiúsculo
L	Qualquer caractere e todos serão convertidos para minúsculo
A	Qualquer caractere ou número
?	Qualquer caractere
*	Qualquer coisa
H	Qualquer hexadecimal (0-9, a-f ou A-F)

```
1 MaskFormatter mask = new MaskFormatter("(##) #####-####");
```

- O construtor dispara uma exceção do tipo `ParseException`

Classe MaskFormatter – tratando a exceção

```
1  public String formata(String mascara, String valor){
2    MaskFormatter mask = null;
3    String resultado = "";
4    try {
5      mask = new MaskFormatter(mascara);
6      mask.setValueContainsLiteralCharacters(false);
7      mask.setPlaceholderCharacter('_');
8      resultado = mask.valueToString(valor);
9    } catch (ParseException e) {
10     e.printStackTrace();
11   }
12   return resultado;
13 }
14 public static void main(String[] args) {
15   Principal p = new Principal();
16   System.out.println(p.formata("(##) #####-####", "49998765432"));
17 }
```


Classe MaskFormatter – encaminhando a exceção

```
1  public String formata(String m, String v) throws ParseException {
2    MaskFormatter mask = null;
3    String resultado = "";
4    mask = new MaskFormatter(m);
5    mask.setValueContainsLiteralCharacters(false);
6    mask.setPlaceholderCharacter('_');
7    resultado = mask.valueToString(v);
8    return resultado;
9  }
10 public static void main(String[] args) {
11   Principal p = new Principal();
12   try {
13     System.out.println(p.formata("(##) #####-####", "49998765432"));
14   } catch (ParseException e) {
15     e.printStackTrace();
16   }
17 }
```