

Linguagem de Programação

ADS – Análise e Desenvolvimento de Sistemas
Prof. Vagner Macedo

Aula 15
Tratamento de Exceções

Definição

- Evento que ocorre durante a execução de um programa, que interrompe o fluxo normal de instruções.

Funcionamento

- Quando uma exceção ocorre em um método, o método cria um objeto que contém informações para manipular a exceção:
 - Tipo de Exceção;
 - Estado do programa;
 - Mensagem;
 - Exceção Ligada.
- Ao disparar uma exceção, o sistema procura na pilha de chamada de métodos o primeiro método que previu e capturou essa exceção.

Pilha de Chamada de Métodos (Call Stack)

- Essa pilha representa as chamadas internas de métodos do programa
- Quando estamos depurando um programa na IDE, podemos visualizar essa pilha em um ponto de interrupção através da janela “Pilha de Chamadas”.

The screenshot shows an IDE interface with the Main.java file open. The code defines a Main class with four methods: main, método1, método2, and método3. Each method contains a call to System.out.println("Passou pelo ..."). The Call Stack window at the bottom shows the current stack trace:

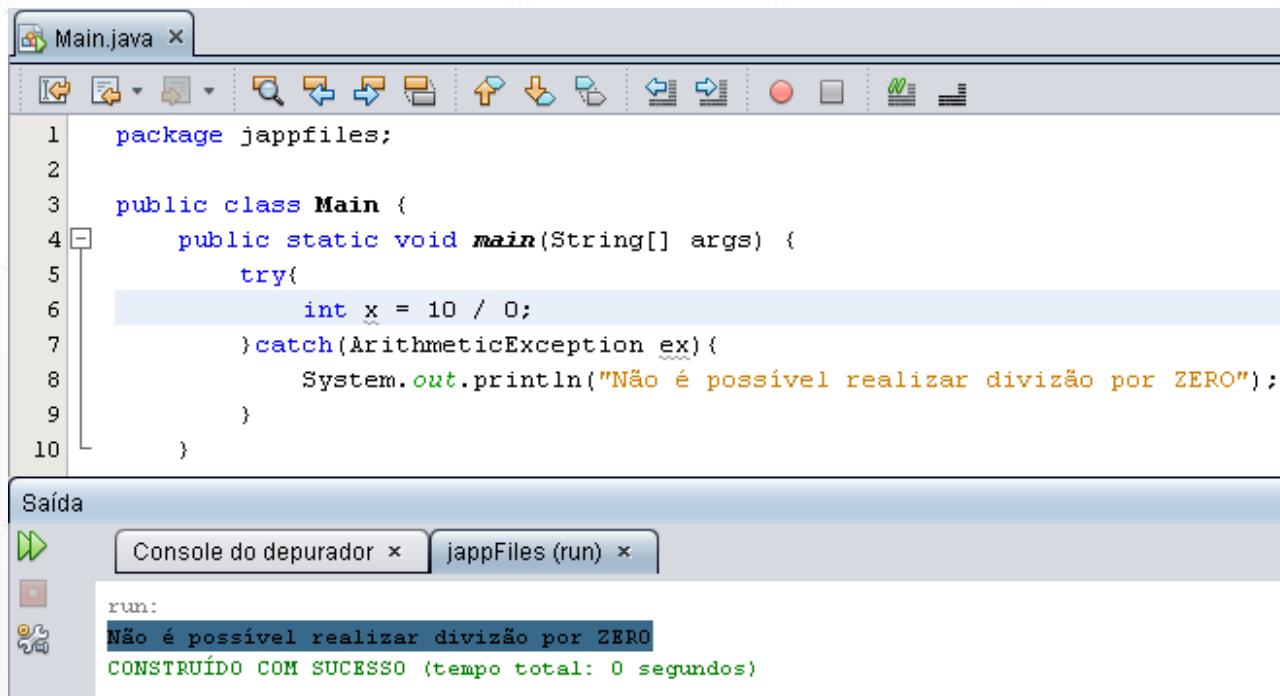
Nome
Main.método3:17
Main.método2:14
Main.método1:10
Main.main:6

Manipulador de Exceção

- Para capturar uma exceção, utilizamos um manipulador de exceção
- Ao encontrar na pilha de chamadas um método com um manipulador de exceção, o sistema passa ao manipulador o objeto criado na exceção
- Se nenhum manipulador de exceção for encontrado na pilha de chamadas de métodos, a aplicação é encerrada

A instrução try-catch

- A instrução permite capturar uma exceção e executar um bloco de instruções caso ela aconteça



The screenshot shows a Java code editor and a terminal window. The code editor displays Main.java with the following content:

```
1 package jappfiles;
2
3 public class Main {
4     public static void main(String[] args) {
5         try{
6             int x = 10 / 0;
7         }catch(ArithmeticsException ex){
8             System.out.println("Não é possivel realizar divizão por ZERO");
9         }
10    }
}
```

The terminal window below shows the output of the run command:

```
run:
Não é possivel realizar divizão por ZERO
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Sintaxe do try-catch

```
try{  
    //instruções que devem ser monitoradas, pois  
    //podem disparar uma exceção  
}  
catch(Exception e){  
    //instruções que devem ser executadas se  
    //ocorrer uma exceção  
}  
finally{  
    //instruções que devem ser executadas sempre  
}
```

O bloco “catch”

- Podemos utilizar vários blocos “catch” para o mesmo “try”.

The screenshot shows an IDE interface with a code editor and a console window. The code editor displays a Java file named Main.java with the following content:

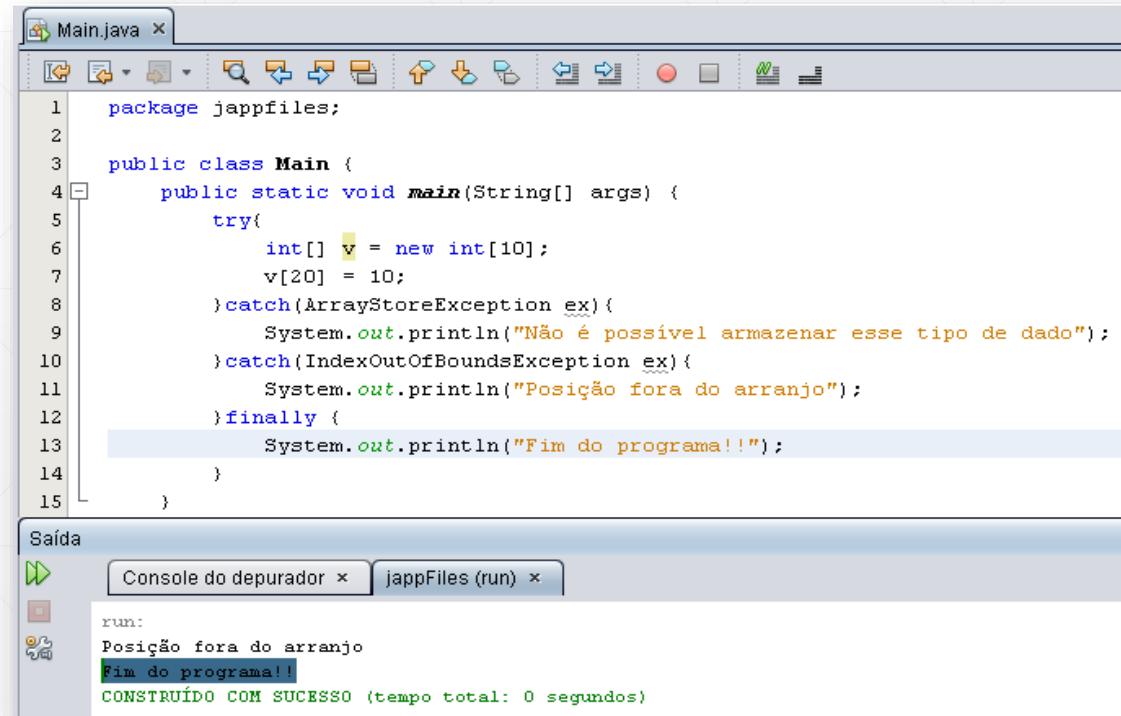
```
1 package jappfiles;
2
3 public class Main {
4     public static void main(String[] args) {
5         try{
6             int[] v = new int[10];
7             v[20] = 10;
8         }catch(ArrayStoreException ex){
9             System.out.println("Não é possível armazenar esse tipo de dado");
10        }catch(IndexOutOfBoundsException ex){
11            System.out.println("Posição fora do arranjo");
12        }
13    }
14}
```

The line `v[20] = 10;` is highlighted in yellow. The console window below shows the output of the program:

```
run:
Posição fora do arranjo
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

O bloco “finally”

- O bloco “finally” é utilizado para que tenhamos instruções que sempre devem ser executadas, ocorrendo exceção ou não.



The screenshot shows a Java code editor window titled "Main.java" and a "Saída" (Output) window. The code in Main.java is as follows:

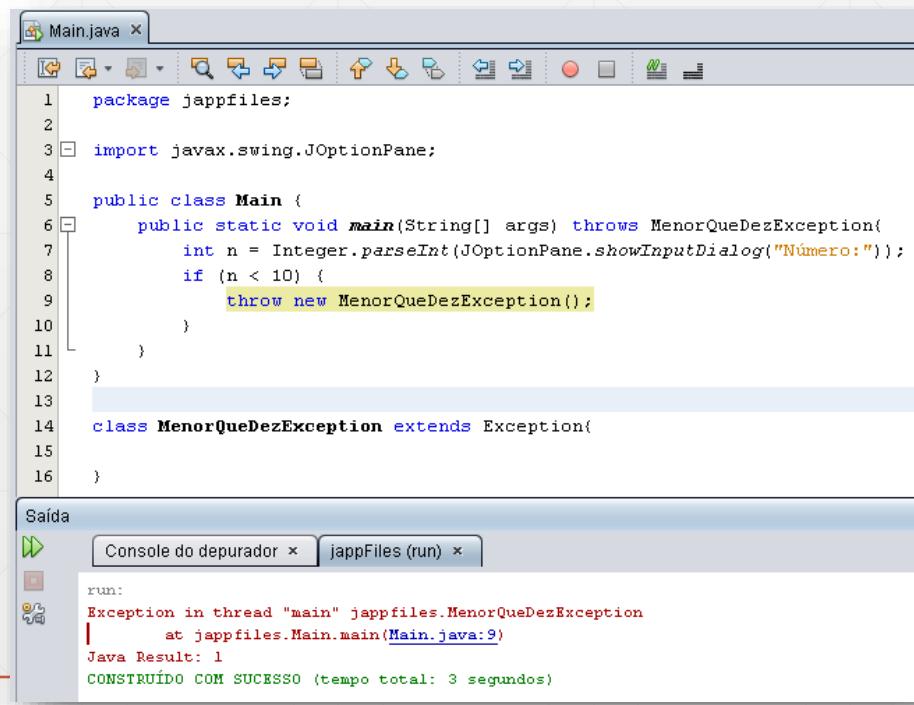
```
1 package jappfiles;
2
3 public class Main {
4     public static void main(String[] args) {
5         try{
6             int[] v = new int[10];
7             v[20] = 10;
8         }catch(ArrayStoreException ex){
9             System.out.println("Não é possível armazenar esse tipo de dado");
10        }catch(IndexOutOfBoundsException ex){
11            System.out.println("Posição fora do arranjo");
12        }finally{
13            System.out.println("Fim do programa!!!");
14        }
15    }
}
```

The "finally" block is highlighted with a light blue background. In the "Saída" window, the output of the program is shown:

```
run:
Posição fora do arranjo
Fim do programa!!!
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Disparando exceções

- Se quisermos “forçar a barra” e disparar uma exceção propositalmente no programa, podemos utilizar o comando “throws” como uma instrução.



The screenshot shows an IDE interface with a code editor and a terminal window. The code editor displays Main.java:

```
1 package jappfiles;
2
3 import javax.swing.JOptionPane;
4
5 public class Main {
6     public static void main(String[] args) throws MenorQueDezException{
7         int n = Integer.parseInt(JOptionPane.showInputDialog("Número:"));
8         if (n < 10) {
9             throw new MenorQueDezException();
10        }
11    }
12 }
13
14 class MenorQueDezException extends Exception{
15 }
```

The terminal window below shows the output of running the program:

```
run:
Exception in thread "main" jappfiles.MenorQueDezException
|   at jappfiles.Main.main(Main.java:9)
Java Result: 1
CONSTRUÍDO COM SUCESSO (tempo total: 3 segundos)
```

Obrigado!

