

Aula 15 - Tratamento de Exceções (Erros)

Quando se cria programas de computador em Java, há possibilidade de ocorrer erros imprevistos durante sua execução, esses erros são conhecidos como exceções e podem ser provenientes de erros de lógica ou acesso a dispositivos ou arquivos externos.

O que são exceções (erros)?

As exceções ocorrem quando algo imprevisto acontece, elas podem ser provenientes de erros de lógica ou acesso a recursos que talvez não estejam disponíveis.

Alguns possíveis motivos externos para ocorrer uma exceção são:

- Tentar abrir um arquivo que não existe;
- Tentar fazer consulta a um banco de dados que não está disponível;
- Tentar escrever algo em um arquivo sobre o qual não se tem permissão de escrita;
- Tentar conectar em servidor inexistente;

1. Try..Catch

Para tratar as exceções em Java são utilizados os comandos **try** e **catch**.

O Try..Catch é uma estrutura com objetivo de capturar erros que podem ocorrer e tratá-los sem deixar “estourar” na tela.

```
try
{
    // bloco de instruções que podem dar erro
}
catch(tipo_excecao e)
{
    // bloco de instruções que irá tratar o possível erro
}
catch(tipo_excecao2 e)
{
    // bloco de instruções que irá tratar o possível erro
}
catch(tipo_excecao3 e)
{
    // bloco de instruções que irá tratar o possível erro
}
```

Onde:

try{ ... } - Neste bloco são introduzidas todas as linhas de código que podem vir a lançar uma exceção.

catch(tipo_excecao e) { ... } - Neste bloco é descrita a ação que ocorrerá quando a exceção for capturada. Esse bloco **não é executado**, a menos que aconteça uma exceção (erro) no bloco try.

Exemplo 01:

Sem tratamento de exceção (erro):

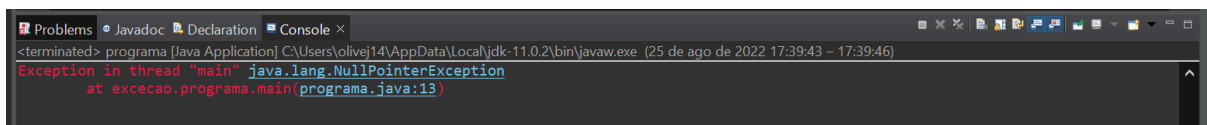
```
public static void main(String[] args) {
    Scanner ler = new Scanner(System.in);

    String nome = null;
    String novoNome = null;

    novoNome = nome.toUpperCase();

    System.out.println("Nome antigo: " + nome);
    System.out.println("Nome novo: " + novoNome);
}
```

Resultado:



Com tratamento de exceção (erro):

```
public static void main(String[] args) {
    Scanner ler = new Scanner(System.in);

    String nome = null;
    String novoNome = null;

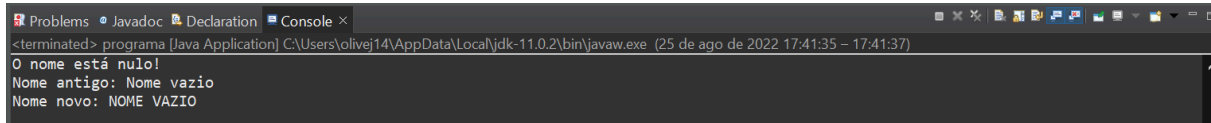
    try
    {
        novoNome = nome.toUpperCase();
    }
    catch (NullPointerException e)
    {
        System.out.println("O nome está nulo!");

        nome = "Nome vazio";
        novoNome = nome.toUpperCase();
    }

    System.out.println("Nome antigo: " + nome);
}
```

```
System.out.println("Nome novo: " + novoNome);  
}
```

Resultado:



```
<terminated> programa [Java Application] C:\Users\olivej14\AppData\Local\jdk-11.0.2\bin\javaw.exe (25 de ago de 2022 17:41:35 - 17:41:37)  
O nome está nulo!  
Nome antigo: Nome vazio  
Nome novo: NOME VAZIO
```

Exemplo 02:

```
package execucao;  
  
import java.util.InputMismatchException;  
import java.util.Scanner;  
  
public class programa {  
  
    public static void main(String[] args) {  
        Scanner ler = new Scanner(System.in);  
  
        int numero;  
        int resultado;  
  
        try  
        {  
            System.out.printf("Digite um número: ");  
            numero = ler.nextInt();  
  
            resultado = numero / 0;  
        }  
        catch(InputMismatchException e) {  
            System.out.println("É permitido apenas a digitação de números  
inteiros!");  
        }  
        catch(ArithmeticException e) {  
            System.out.println("Não é possível dividir por zero!");  
        }  
  
        System.out.println("Fim do programa!");  
    }  
}
```

Uma outra possibilidade é capturar qualquer erro possível, utilizando a classe Exception, veja:

```
public static void main(String[] args) {
    Scanner ler = new Scanner(System.in);

    int numero;
    int resultado;

    try
    {
        System.out.printf("Digite um número: ");
        numero = ler.nextInt();

        resultado = numero / 0;
    }
    catch(Exception e) {
        System.out.println("Erro na aplicação contate o adm do sistema!");
        System.out.println(e);
        System.out.println(e.getMessage());
    }

    System.out.println("Fim do programa!");
}
```

2. Try..Finally

O Try..Finally é uma estrutura com objetivo de executar um trecho de código mesmo que uma exceção aconteça. Ou seja, tudo o que estiver dentro do bloco finally será executado com certeza, independente que uma exceção aconteça ou não.

Ele pode ser utilizado com o try..catch, dessa forma:

```
try
{
    // bloco de instruções que podem dar erro
}
catch(tipo_excecao e)
{
    // bloco de instruções que irá tratar o possível erro
}
catch(tipo_excecao2 e)
```

```

{
    // bloco de instruções que irá tratar o possível erro
}
catch(tipo_excecao3 e)
{
    // bloco de instruções que irá tratar o possível erro
}
finally{
    // bloco de instruções que será executado mesmo que aconteça uma
    exceção tratada ou não tratada
}

```

Ou sozinho:

```

try
{
    // bloco de instruções que podem dar erro
}
finally{
    // bloco de instruções que será executado mesmo que aconteça uma
    exceção tratada ou não tratada
}

```

Exemplo:

```

public static void main(String[] args) {
    Scanner ler = new Scanner(System.in);

    int numero;
    int resultado;

    try
    {
        System.out.printf("Digite um número: ");
        numero = ler.nextInt();

        resultado = numero / 0;
    }
    catch(InputMismatchException e) {
        System.out.println("É permitido apenas a digitação de números
        inteiros!");
    }
    finally {

```

```
        System.out.println("Independente de tudo eu FUI executado :)");  
    }  
  
    System.out.println("Fim do programa!");  
}
```