

POO - PROGRAMAÇÃO ORIENTADA A OBJETOS

Erros e Exceções

Rodrigo R Silva

Instituto Federal de Educação, Ciência e Tecnologia Sul-Rio-Grandense
Campus Bagé

Nesta Aula Veremos...

1 Introdução

2 Erros e Exceções

Introdução

Conceito

- Uma exceção é um erro que pode ocorrer durante a execução de um programa.
- Java possibilita o tratamento de forma estruturada e controlada de possíveis erros de execução.
- A tecnologia Java oferece um mecanismo que automatiza o tratamento de erros.
- Em linguagens mais antigas o tratamento de erros deveria ser realizado de forma manual pelo programador.

Erros e Exceções

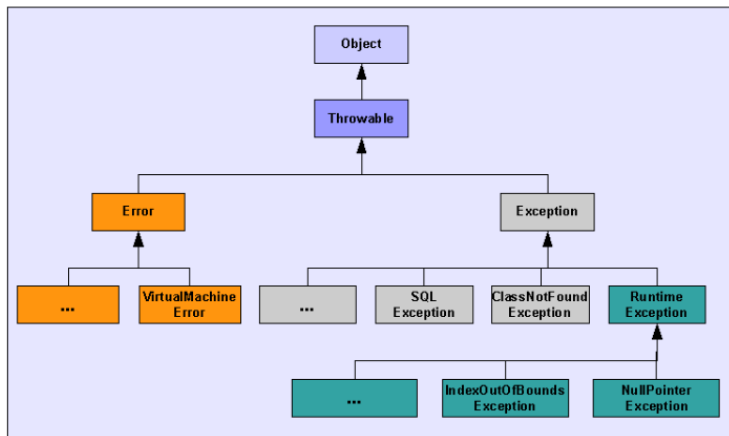
Tratador de exceções

- Trata-se de um bloco de código que é executado automaticamente quando um erro ocorre.
- Dessa forma, a chamada a métodos não necessita de um tratamento manual do sucesso ou falha na execução.
- O tratador de exceções se encarrega de processar o possível erro.
- Java define exceções padrão para erros mais comuns:
- Ex: divisão por zero, tentativa de acesso a índices fora do intervalo de um array.

Hierarquia de exceções

- Em Java, todas as exceções são representadas por classes as quais são especializações de Throwable.
- Dessa forma, quando algum erro ocorre um objeto de alguma classe de exceção é criado.
- Há duas especializações de Throwable, são elas:
- Exception: representam erros decorrentes da atividade de um programa. Ex: entrada e saída, índices de arrays, divisão por zero.
- Error: são erros que não podemos controlar, como falhas na máquina virtual Java, são independentes da execução do programa.

Hierarquia de exceções



Sentido de leitura

The screenshot shows an IDE console window with the following text:

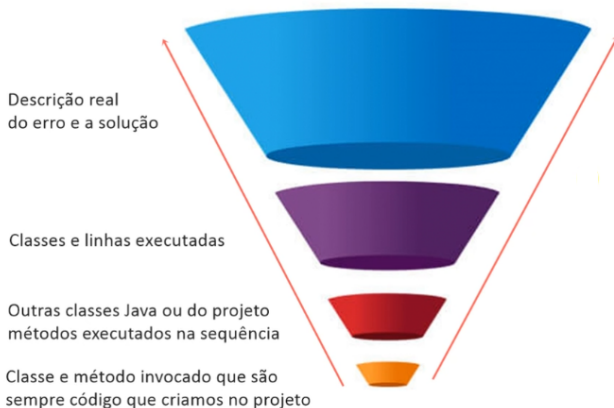
```
<terminated> PrimeiraClasseJava [Java Application] C:\Program Files\Java\jdk-9.0.4\bin\javaw.exe (22 de jun de 2019 16:47:55)  
Exception in thread "main" java.lang.NullPointerException  
    at cursojava.classes.Diretor.autenticar(Diretor.java:64)  
    at cursojava.classesauxiliares.FuncaoAutenticacao.autenticar(FuncaoAutenticacao.java:12)  
    at cursojava.executavel.PrimeiraClasseJava.main(PrimeiraClasseJava.java:26)
```

Annotations and their corresponding text boxes:

- Classe do nosso sistema que é o ponto inicial do erro.** (Points to `PrimeiraClasseJava` in the stack trace)
- Método no nosso sistema onde executa e é a entrada inicial do erro.** (Points to `main` in the stack trace)
- Métodos e classes internas do nosso código.** (Points to `PrimeiraClasseJava` in the stack trace)
- Identificação e descrição do exato do erro e ponto final.** (Points to `java.lang.NullPointerException`)
- Classes e linha exata que ocorreu o erro durante a execução do código.** (Points to `FuncaoAutenticacao.java:12`)

A vertical red arrow on the right side of the console window is labeled "Sentido de leitura" (Direction of reading), pointing upwards.

Funil de exceções



Sempre verifique qual o ponto inicial causador do problema, após isso avalie as linhas e para a descrição do erro tem sempre uma solução direta ou indireta sendo o caminho.

Erros mais comuns para iniciantes

- 1 `NullPointerException` : Erro número 1 do Java, quer dizer que existe um variável ou referência nula.
- 2 `NumberFormatException` : Converter string em um número sendo que a mesma está fazendo.
- 3 `FileNotFoundException` : Carregar um arquivo, mas não foi encontrado no caminho informado.
- 4 `ClassCastException` : Converter um objeto B para A sendo que não são do mesmo tipo ou sem relação.
- 5 `ClassNotFoundException` : Carregar uma classe que não foi encontrada.

Tratamento de exceções

- O tratamento de exceções em Java é realizado por cinco palavras chaves, são elas:
- **try, catch, throw, throws e finally.**
- Formam uma estrutura de comandos onde o uso de uma implica o uso de outra.
- Funcionamento:
- As instruções que desejamos controlar ficam no interior de um bloco **try**. Para captura de exceções usamos um bloco **catch**. Se desejarmos lançar manualmente uma exceção usamos **throw**. Casos onde a exceção deve ser tratada fora do método usamos **throws**. Todo código que deve ser executado ao sair de um bloco **try** deve ser colocado em um bloco **finally**.

Exemplo

- Dentro do TRY fica o código que pode ocorrer uma exceção e CATCH é o ponto de caída do código após a interrupção. A classe EXCEPTION tem todas as informações do erro.
- Toda exceção interrompe um bloco e um fluxo de processo.

```
try {  
  
    /*Código de regra de negócio - processos do sistema*/  
    /*Por exemplo realização de uma venda*/  
  
} catch (Exception e) {  
    e.printStackTrace(); /*IMPRIME A PILHA DE ERRO NO CONSOLE*/  
  
}
```

Ignorando exceções

- O tratamento de exceções evita o encerramento anormal do programa.
- Se um programa não tratar exceções a máquina virtual Java se encarrega disso.
- Entretanto, interrompe a execução do programa e mostra uma lista de chamadas de métodos que levaram até a causa da exceção.
- Isso até é útil para o programador, mas confuso para o usuário da aplicação.

Ignorando exceções

```
1 package classe.executavel;
2
3 public class ClasseExcecoes {
4
5     public static void main(String[] args) {
6
7         String nomes[] = {"Fulano", "Ciclano", "Beltrano"};
8         System.out.println("Nome: "+nomes[3]);
9
10    }
11
12 }
13
```

Problems @ Javadoc Declaration Console x

<terminated> ClasseExcecoes [Java Application] /home/rodrigo/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.lin
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
at classe.executavel.ClasseExcecoes.main(ClasseExcecoes.java:8)

Tratando várias exceções

- Podemos adicionar várias cláusulas catch a um bloco try, tratando várias exceções ao mesmo tempo.

```
3 public class ClasseExcecoes {  
4  
5 public static void main(String[] args) {  
6  
7     int v1[] = {5,8,9,6,5,1,4};  
8     int v2[] = {2,0,4,0,8};  
9     for(int i = 0; i < v1.length; i++) {  
10        try {  
11            System.out.println(v1[i] + "/" + v2[i] + "=" + v1[i]/v2[i]);  
12        } catch (ArithmeticException e) {  
13            System.out.println("Erro de divisão por Zero - "+e);  
14        } catch (ArrayIndexOutOfBoundsException e) {  
15            System.out.println("Índice fora do intervalo do array -"+e);  
16        }  
17    }  
18 }
```

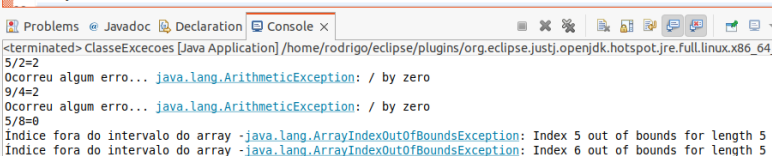
Problems @ Javadoc Declaration Console x

<terminated> ClasseExcecoes [Java Application] /home/rodrigo/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_5/2=2
Erro de divisão por Zero - [java.lang.ArithmeticException](#): / by zero
9/4=2
Erro de divisão por Zero - [java.lang.ArithmeticException](#): / by zero
5/8=0
Índice fora do intervalo do array - [java.lang.ArrayIndexOutOfBoundsException](#): Index 5 out of bounds for length 5
Índice fora do intervalo do array - [java.lang.ArrayIndexOutOfBoundsException](#): Index 6 out of bounds for length 5

Exceções de subclasses

- Para captura tanto de exceções de uma superclasse como de subclasses, primeiro devemos tratar a exceção da subclasse.

```
3 public class ClasseExcecoes {
4
5     public static void main(String[] args) {
6
7         int v1[] = {5,8,9,6,5,1,4};
8         int v2[] = {2,0,4,0,8};
9         for(int i = 0; i < v1.length; i++) {
10             try {
11                 System.out.println(v1[i] + "/" + v2[i] + "=" + v1[i]/v2[i]);
12             } catch (ArrayIndexOutOfBoundsException e) {
13                 System.out.println("Índice fora do intervalo do array -" + e);
14             } catch (Exception e) {
15                 System.out.println("Ocorreu algum erro... " + e);
16             }
17         }
18     }
```



Problems @ Javadoc Declaration Console x

<terminated> ClasseExcecoes [Java Application] /home/rodrigo/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_5/2=2
Ocorreu algum erro... [java.lang.ArithmeticException](#): / by zero
9/4=2
Ocorreu algum erro... [java.lang.ArithmeticException](#): / by zero
5/8=0
Índice fora do intervalo do array -[java.lang.ArrayIndexOutOfBoundsException](#): Index 5 out of bounds for length 5
Índice fora do intervalo do array -[java.lang.ArrayIndexOutOfBoundsException](#): Index 6 out of bounds for length 5

Bloco finally

- Sempre é executado, ocorrendo erros ou não.

```
public class ClasseExcecoes {  
    public static void main(String[] args){  
        int v1[] = {5,8,9,6,5,1,4};  
        int v2[] = {2,2,4,7,8,9};  
        for(int i = 0; i < v1.length; i++) {  
            try {  
                System.out.println(v1[i]+" / "+v2[i]+"="+v1[i]/v2[i]);  
            }catch (Exception e) {  
                StringBuilder saida = new StringBuilder();  
                e.printStackTrace();  
                System.out.println("Mensagem: "+e.getMessage());  
  
                for(int j = 0; j < e.getStackTrace().length; j++) {  
                    saida.append("\nClasse de erro: "+e.getStackTrace()[j].getClassName());  
                    saida.append("\nMétodo de erro: "+e.getStackTrace()[j].getMethodName());  
                    saida.append("\nMensagem de erro: "+e.getMessage());  
                    saida.append("\nLinha de erro: " + e.getStackTrace()[j].getLineNumber());  
                }  
                JOptionPane.showMessageDialog(null, saida);  
            }finally {  
                JOptionPane.showMessageDialog(null, "Ok, executou o finally!");  
            }  
        }  
    }  
}
```

Criando classes de Exceções

- Podemos criar nossas próprias classes de exceção.
- Para isso, devemos estender Exception, vejamos:

```
public class NumeroInvalidoException extends Exception{  
    private int numero;  
  
    public NumeroInvalidoException(int numero) {  
        this.numero = numero;  
    }  
  
    public String toString() {  
        return "O número (" + this.numero + ") informado é inválido";  
    }  
}
```

Lançando uma exceção

- Após definirmos o código de tratamento da exceção em uma classe, podemos utilizá-la em aplicativos.
- Para lançar uma exceção usamos **throw**.

```
 7 public class ClasseExcecoes {  
 8  
 9     public static void main(String[] args) {  
10  
11         int nro = Integer.parseInt(JOptionPane.showInputDialog("Informe um valor inteiro [1-10]: "));  
12         try {  
13             if(nro < 1 || nro > 10) {  
14                 throw new NumeroInvalidoException(nro);  
15             }  
16         } catch (Exception e) {  
17             System.out.println(e);  
18         }  
19     }  
20 }
```

Problems @ Javadoc Declaration Console ×

<terminated> ClasseExcecoes [Java Application] /home/rodrigo/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x
0 número (-1) informado é inválido

Lançando mas tratando fora

- Podemos lançar uma exceção em um método, mas deixar seu tratamento para o método que invocou.
- Para isso, usamos **throws**.

```
6
7 public class ClasseExcecoes {
8
9     public static void main(String[] args) throws NumeroInvalidoException{
10
11         int nro = Integer.parseInt(JOptionPane.showInputDialog("Informe um valor inteiro [1-10]: "));
12         if(nro < 1 || nro > 10) {
13             throw new NumeroInvalidoException(nro);
14         }
15     }
16 }
```

Problems @ Javadoc Declaration Console ×

<terminated> ClasseExcecoes [Java Application] /home/rodrigo/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux:
Exception in thread "main" O número (11) informado é inválido
at classe.executavel.ClasseExcecoes.main(ClasseExcecoes.java:13)

OBRIGADO!