UFRPE

Exceções

Richarlyson A. D'Emery
site: https://sites.google.com/site/profricodemery/mpoo
grupo: http://groups.google.com/group/**mpoo_uast**
email grupo: **mpoo_uast**@googlegroups.com
contato: rico_demery@yahoo.com.br

Sumário

- Objetivos
- Introdução
- A Classe Exception
- Captura e Tratamento
- try – catch - finally
- throws

Objetivos

- Entender tratamento de exceção e erro
- Implementar blocos **try**
- Levantando (**throw**) exceções
- Usar blocos **catch** para especificar *handlers*
- Usar blocos **finally** para liberar recursos
- Entender a hierarquia de exceções de java
- Criar exceções definidas pelo programador

Exceção

UFRPE

- Uma exceção é um evento não esperado
- Exceções podem ocorrer quando:
  - se tenta abrir um arquivo que não existe,
  - ocorrer problemas com a rede,
  - operandos estão fora do intervalo válido,
  - se tentar carregar uma classe que não existe,
  - índice do array fora dos limites,
  - overflow,
  - divisão por zero,
  - parâmetros inválidos,
  - falta de memória.

Exceção

- A classe Exception define erros menos graves em Java e que devem ser tratados
- Java permite um programa capturar:
  - todos os tipos de exceções,
  - todas as exceções de um determinado tipo,
  - ou alguns tipos de exceções.
- Vantagem
  - programador pode ver o processamento do erro próximo ao código gerador
  - determinar se o tratamento de erro está implementado e correto.
- Desvantagem
  - o código fica poluído de processamento de erro,
  - fica mais difícil de se concentrar na aplicação.
  - A aplicação pode ficar difícil de entender e manter.

Exceção

- A separação do código de tratamento de erro do programa principal esta consistente com as características de separabilidade da orientação a objetos.
- O programa fica mais robusto reduzindo a probabilidade de erros não capturados por um programa.
- Tratamento de Exceções servem para habilitar programas a tratarem os erros em vez de sofrer suas conseqüências.

Exceção

- Tratamento de Exceções trata
  - ***erros síncronos*** (divisão por zero), não são projetados para tratamento de
  - ***erros assíncronos*** (E/S de disco, chegada de mensagem da rede, etc), este são melhor tratados por outra técnica:
    - ***processamento de interrupções***.

Exceção

UFRPE

- Tratamento de Exceções são utilizados em situações em que o sistema pode se recuperar de um mal funcionamento causando uma exceção.
  - O procedimento de recuperação é chamado ***exception handler.***
- Um *exception handler* é utilizado em situações em que o mal funcionamento será tratado por um escopo diferente daquele em o mal funcionamento foi detectado.
- Exceções são objetos de classes derivadas da superclasse ***Exception***.

Exceção – Exemplo 1

UFRPE

- Índice do array fora dos limites,

```
public class Exemplo1{
   public static void main(String [] args) {
      int i = 0;
      String nomes [] = {"Bart Simpson", "Dr. Smith", "Rico D'Emery"};
      while (i<4){
        System.out.println(nomes[i]);
        i++;
      }
    }
}
```

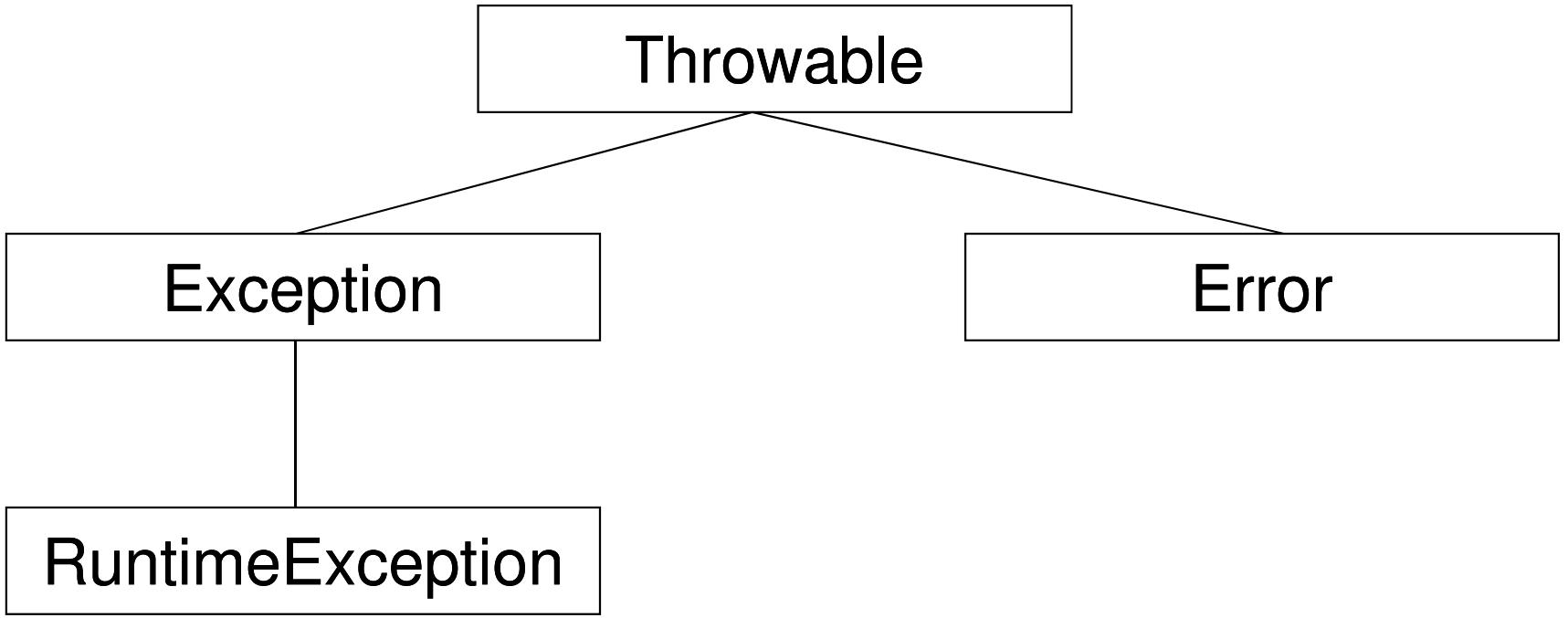
Exceções

- Se nenhum método capturar a exceção o programa termina e uma mensagem de erro é emitida.
  - Para o Exemplo 1

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 3
at Exemplo1.main(Exemplo1.java:10)
```

Classes de Exceções

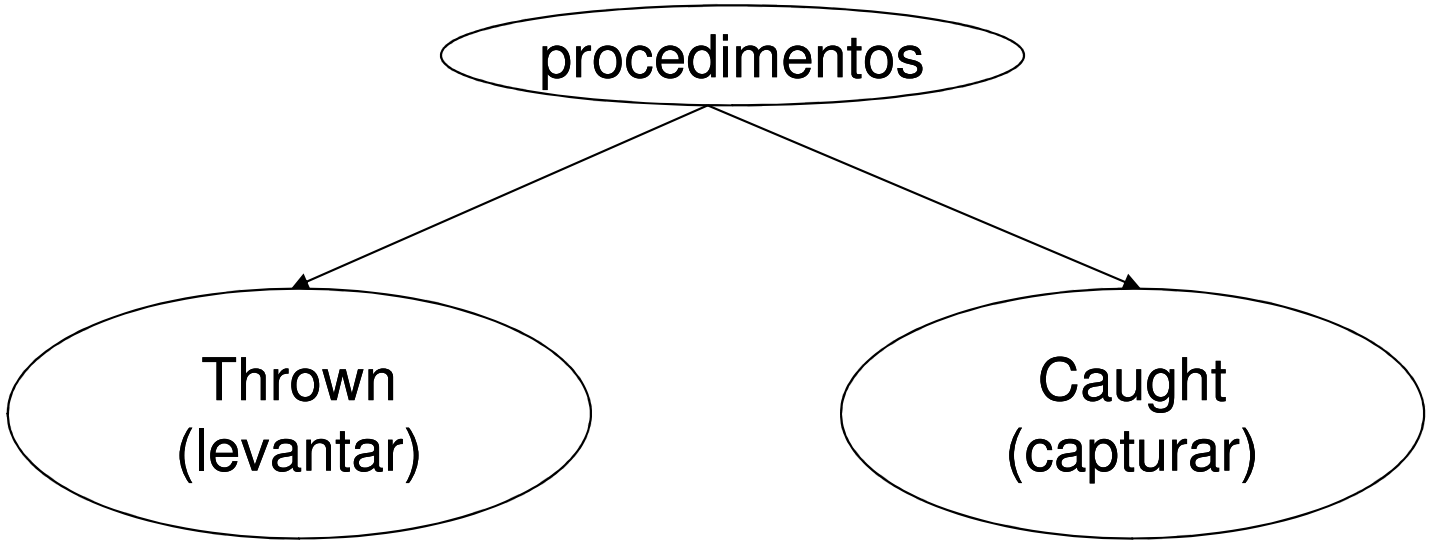
UFRPE



Tratamento de Exceção em Java

UFRPE

- Quando um método detecta um erro e não está apto a manuseá-lo o método *levanta uma exceção (throw).*
- Se existe um *exception handler*, para processar a exceção ela será capturada (*caught*) e tratada (*handled*).
- Exceções são eventos surpresas que ocorrem durante a execução de um programa.



Como Capturar e Tratar uma Exceção

- Para manipular uma possivel exceção use **try**.
- Para capturar e tratar exceções levantadas use **catch**.
- Para garantir que um bloco sempre seja executado use **finally**.

try – catch – finally


```
try{
   //código que pode levantar uma exceção
}
catch{
   //código que será executado se TipoDaExceção for
   levantada
}
finally{
   //código sempre executado
}
```

Tratando Exceções – Exemplo 2

UFRPE

- Tratando índice do array fora dos limites,

```
public class Exemplo2{
    public static void main(String [] args) {
        int i = 0;
        String nomes [] = {"Bart Simpson","“Dr. Smith","Rico D’Emery"};

        while (i<4){
            try{
                    System.out.println(nomes[i]);
             }
             catch(ArrayIndexOutOfBoundsException arrayIndexOutOfBoundsException){
                    System.out.println("Erro no Array!");
                    break;
             }

             finally{
                 System.out.println("Sempre mostrado!");
             }
             i++;
        }
    }
}
```

Declarar ou Tratar

UFRPE

- Trate as exceções usando try-catch- finally
- Declare que o código pode levantar mais de uma exceção.
- Exceções de *runtime* não precisam ser declaradas ou tratadas.

Exemplo 3

UFRPE

- Divisão por 0 (zero)

```
import java.util.Scanner;

public class Exemplo3 {
   public static void main(String [] args) {
        Scanner input = new Scanner( System.in );
        int resultado=0;
        System.out.print( "Informe o primeiro número: ");
        int numero1 = Integer.parseInt(input.nextLine());
        System.out.print( "Informe o segundo número: ");
        int numero2 = Integer.parseInt(input.nextLine());
        resultado = (numero1 / numero2);
        System.out.println( "O resultado é " + resultado);
   }
}
```

Exemplo 4

UFRPE

- Tratando a Divisão por 0

```
import java.util.Scanner;
public class Exemplo4 {
    public static void main(String [] args) {
          Scanner input = new Scanner( System.in );
          int resultado=0;

          System.out.print( "Informe o primeiro número: ");
          int numero1 = Integer.parseInt(input.nextLine());
          System.out.print( "Informe o segundo número: ");
          int numero2 = Integer.parseInt(input.nextLine());

          try{ resultado = (numero1 / numero2); }
          catch(ArithmeticException arithmeticException){
               System.out.print( "Não é possível divisão por zero");
          }
          System.out.println( "O resultado é " + resultado);
    }
}
```

Disparando uma exceção

UFRPE

- A instrução ***throw*** é executada para indicar que uma exceção ocorreu.
- O operando de um ***throw*** está em Throwable (java.lang)
- Classes de Throwable
  - Exception – ocorre durante a execução
    - Objeto chamado se *objeto de execução*
  - Error – não deveria ocorrer (sistema)

A Cláusula throws

- Use o throws na declaração do método.
- Um método pode levantar mais de uma exceção.
- Após a cláusula throws especifique a lista das possíveis exceções levantadas pelo método:

```
public void metodoProb() throws IOException,
        MalformedURLException{ }
```

Exemplo 5

UFRPE

```
//Exemplo5.java
import java.util.Scanner;

public class Exemplo5 {
    public static void main(String [] args) {
          Scanner input = new Scanner( System.in );
          float resultado=0;
          System.out.print( "Informe o primeiro número: ");
          float numero1 = Integer.parseInt(input.nextLine());
          System.out.print( "Informe o segundo número: ");
          float numero2 = Integer.parseInt(input.nextLine());
          try{
              resultado = quocient (numero1, numero2);
              System.out.println( "O resultado é " + resultado);
          }catch(Exception e){}
          finally{}
    }

    public static float quocient (float numer, float den) throws Exemplo5Exception{
          if (den==0)
              throw new Exemplo5Exception();
          else
              return (float) numer/den;
    }
}
```

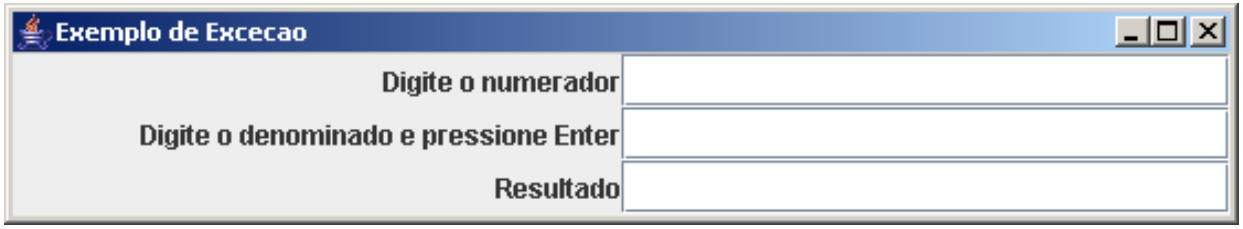
Exemplo 5 – cont.


```
//Exemplo5Exception.java
//Definicao da excecao
//Dipara uma exceçao quando se tenta  uma divisão por zero

public class Exemplo5Exception extends ArithmeticException{

   // contrutor de argumento é dispara erro default
   public Exemplo5Exception(){
        System.out.print( "Não é possível divisão por zero");
   }

   //contrutor com mensagem personalizada
   public Exemplo5Exception(String mensagem){
        super (mensagem);
   }
}
```

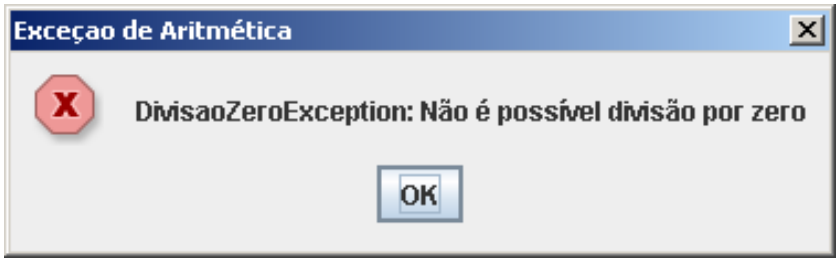
Exercício 1

UFRPE

- Utilize uma GUI para
  - Permitir a entrada de um numerador
  - Permitir a entrada de um denominador



  - Caso a divisão seja por zero exiba a mensagem :



Exercício 2

UFRPE

- Modifique o exercício 1 para:
  - Permitir a entrada de um número
  - Permitir a entrada de outro número
  - Permite escolher o tipo de operação
  - Os números poderão ser reais
  - Caso a divisão seja por zero exiba mensagem de erro

Exercício 3

UFRPE

- Modifique o exercício 2 da seguinte forma:
  - um método contendo o código com o tratamento da exceção.
  - Crie um segundo método que faça uma chamada ao primeiro.

Exercício 4

UFRPE

- Modifique o exercício 2 da seguinte forma:
  - O código com o tratamento da exceção deverá estar em uma outra classe.

UFRPE

FIM

Richarlyson D'Emery
rico_demery@yahoo.com.br