Universidade de São Paulo
Brasil

Aula 07 – Tratamento de Exceções

A forma certa de pegar erros

MAC0321 - Laboratório de Programação Orientada a Objetos

Professor: Marcelo Finger **(mfinger@ime.usp.br)**

Departamento de Ciência da Computação
Instituto de Matemática e Estatística

Tópicos

1. O que são e não são Exceções
2. Trechos críticos de código: Sintaxe try/catch.
3. Várias exceções
4. Lançamento de exceções: throw/throws
5. Classes de Exceção
6. Funções Lambda (sem nome) em Java
7. Testando exceções automaticamente

O que é (e não é) uma Excessão

Exceções

Em Java, a *ocorrência de erros* (exceções) durante a execução de um programa não necessariamente significa que o programa termina.

A linguagem já provê um mecanismo para **indicar trechos críticos** num programa e **se recuperar de eventuais exceções** ocorridas nestes trechos, sem parar a execução do programa

O Que Exceções NÃO SÃO

- Um conceito exclusivo de Java
- Um conceito relacionado à Orientação a Objetos
- Então o que são?
  - Um conceito de linguagens de programação, assim como if, while, =.

O Que Exceções NÃO SÃO

- Um conceito exclusivo de Java
- Um conceito relacionado à Orientação a Objetos
- Então o que são?
  - Um conceito de linguagens de programação, assim como if, while, =.

Sintaxe try/catch
Trechos críticos de código

Sintaxe para tratamento de exceções

```
try {
      ... // Trecho crítico do programa
}
catch(ExceptionType1 e1) {
     ... // Tratamento da exceção
}
...
catch(ExceptionTypeN eN) {
     ... // Tratamento da exceção
}
finally {  // Eliminação de precedência
           ...
}
```

Exemplo de trecho crítico

```
class conta{
   public static void main( String args[]){
      int den = 0;
      int num = 30;
      int res = num / den;   // Trecho crítico
    }
}
```

A execução do .class correspondente a esta classe provocaria a seguinte mensagem de erro:

```
...> java.lang.ArithmeticException: / by zero
     at ... conta ...
```

Tratamento do trecho crítico

```
class Conta{
   public static void main( String args[]){
        int den = 0;
        int num = 30, res;

        try{  // Indicador do trecho crítico
                res = num / den;
        }
        catch (ArithmeticException e) {
                //Recuperação da exceção
                System.err.println(“Dividiu por zero!”);
                res = 1;
        }
        ... // Continuação do programa
   }
}
```

Várias exceções
Seguidas ou aninhadas

Tratamentos de várias exceções seguidas

```
class Conta {
   public static void main(String args[]){
     int den, num;
     int c[] = {1};
          ...
     try{
               // Indicador do trecho crítico
               int res = num/den;
               c[999] = res;
     }
     catch(ArithmeticException e) {
               // Recuperação da exceção aritmética
               System.err.println("Dividiu por  zero!");
               res = 1;
     }
     catch(ArrayIndexOutOfBoundsException e){
                    // Recuperação da exceção de estouro de índice de vetor
          System.err.println("Estourou o  índice");
     }
     ... // Continuação do programa
  }
```

try aninhado dentro de try (tratamento local de exceções)

```
class Conta{
   public static void main(String args[]){
      int den, num, indice;
      int c[] = {1};
         ...
      try {  // Indicador do trecho crítico
          int res = num / den;
          try {
                c[indice]=res;
          }
          catch(ArrayIndexOutOfBoundsException e){
                // Recuperação:  estouro de índice do vetor
                System.out.println("Estourou o índice");
                indice = 0;
          }
        }
        catch(ArithmeticException e) {
                // Recuperação da exceção aritmética
                System.out.println("Dividiu por zero!");
                den = 1;
        }
        ... // Continuação do programa
   }
}
```

Tratamento com **finally**

Todo comando try exige, pelo menos, uma cláusula **catch** ou **finally**.

finally é executado antes que o **try**/**catch** termine o tratamento do erro, garantindo o tratamento após todo e qualquer **catch**.

Assim, **finally** tem precedência máxima sobre todos os outros tratamentos.

(O tratamento é feito em ordem **crescente** de precedência)

Exemplo com **finally**

```
class Conta{
        public static void main(String args[]){
                int den, num;
                int c[] = {1};
                ...
                try{  // Indicador do trecho crítico
                        int res = num / den;  // instrução “meio” burra
                        c[999] = res;
                }
                catch(ArithmeticException e) {
                        // Recuperação da exceção aritmética
                        System.out.println(“Dividiu por zero!”);
                }
                catch(ArrayIndexOutOfBoundsException e){
                        // Recuperação da exceção de estouro de índice do vetor
                        System.out.println(“Estourou o índice”);
                }
                finally{
                        den = 1;
                }
                // Qual o valor de den aqui???
        }
}
```

Lançamento de Exceções

Sintaxe `throw`/`throws`

Lançamento de exceções

Independente de outras exceções que possam ocorrer na execução, uma classe pode forçar o lançamento de exceções ou relançar uma determinada exceção.

Isto é feito através da palavra **throw**.

```
throw new <construtor da classe de exceção >(parâmetros );
throw <exceção>;
```

Exemplo com **throw**

```
class Exemplo{
        static void proc_exemplo(){
                try{
                        // Tenta lançar instância de uma exceção aritmética
                        throw new ArithmeticException(“Divide por zero”);
                }
                catch(ArithmeticException e){
                        System.err.println("Erro no método" + e.getMessage() );
                        throw e;  // Relança e para tratamento em outro escopo
                }
         }
        public static void main(String args[]) {
                try{
                        exemplo.proc_exemplo();
                }
                catch(ArithmeticException e) {
                        System.err.println(“Retratamento da exceção”);
                }
        }
}
```

Métodos que podem devolver exceções

Pode-se indicar os tipos de exceção que um método pode devolver utilizando-se a palavra reservada **throws**.

```
método(parâmetros)
    throws Exceção1, ..., ExceçãoN {
     ...
}
```

Isto **força** os trechos de programa, que utilizarem este método, a tratarem as exceções com try.

Exemplo com **throws**

```
class Erros{
        static void contas() throws ArithmeticException, ArrayIndexOutOfBoundsException {
                ...
                if (den == 0){
                        throw new ArithmeticException(“Denominador inválido”);
                }
                res = num / den;
                if (indice < 0 || indice >= c.length ){
                        throw new ArrayIndexOutOfBoundsException(“Indice > 0”);
                }
                c[indice] = res;
        }
        public static void main(String args[]) {
                try{
                        contas();
                }
                catch( ArithmeticException e){…}                        // Obrigatório
                catch( ArrayIndexOutOfBoundsException e){…}     // Obrigatório tb
        }
}
```

IME-USP

Classes Exceções
Por Herança

Criação de classes de exceções

Novas classes de exceções podem ser construídas extendo-se a classe **Exception**:

```
class <NovaClasse>  extends Exception {
        atributos
        ...
        construtores
        ...
        public String toString(){
                // O que imprimir quando a classe precisar ser impressa
                ...
        }
        Outros métodos...
}
```

Exemplo de nova classe de exceção

```
class ProprioErro extends Exception {
    private int aux;

    ProprioErro(int a){  //Construtor
        aux = a;
    }
    public String toString(){
        // devolve nome da classe e estado interno
        return("ProprioErro[" + aux + "]\n" + super.toString() );
    }
}
```

Utilização da nova classe

```
class Calculo{
        static void calcula(int a) throws ProprioErro {
                If (a > 10)
                        throw new ProprioErro(a); // Lança instância nova classe
                ...
        }
        public static void main(String args[]){
                try{
                calcula(1);             // Não gera exceção
                calcula(300);           // Gera exceção
                }
                catch(ProprioErro e){ // Tratamento da nova exceção
                        System.out.println(“Capturada “ + e);
                }
        }
}
```

IME-USP

Utilização da nova classe

```
class Calculo {
        static void calcula(int a)
                throws Proprioerro {
                if(a > 10)
                        throw new ProprioErro(a); // Lança instância nova classe
        ...
        }
        public static void main(String args[]){
                try{
                        calcula(1);             // Não gera exceção
                        calcula(300);           // Gera exceção
                }
                catch(ProprioErro e){           // Tratamento da nova exceção
                        System.out.println("Capturada " + e);
                }
        }
}
```

Como são usadas exceções?

- É uma boa forma de apontar os culpados
- Se um método recebe um parâmetro inesperado, pode lançar uma exceção
- Boa técnica de programação defensiva
- NOTA: Todo método que gera exceção DEVE SER TESTADO!!!

Uso de Pré- e Pós-condições

- Sempre garantir que o que é recebido ou devolvido por um método está de acordo com o contrato
- Se não estiver, uma exceção deve ser lançada

Desvio: Funções sem nome em Java (Funções Lambda)

Expressões Lambda em Java

- Uma **expressão lambda** é um pequeno bloco de código que recebe parâmetros e retorna um valor.
- Podem ser passados como parâmetros e retornados por métodos
- Transforma funções em tipos básicos
- Semelhantes aos métodos, mas não precisam de nome e podem ser implementadas diretamente no corpo de um método
- Presente na maioria das linguagens de programação
- Origem: Cálculo Lambda, teoria de computabilidade

$$\lambda x(xa)$$

IME-USP

Sintaxe em Java

A expressão lambda mais simples contém um único parâmetro:

parâmetro -> expressão

Mais de um parâmetro, corpo complexo:

(param$_1$,...., param$_k$) -> { <bloco de instruções> }

Se a expressão lambda precisar retornar um valor, o bloco de código deverá ter uma instrução **return**.

Uso de Expressões Lambda

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> numbers = new ArrayList<Integer>();
    numbers.add(5);
    numbers.add(9);
    numbers.add(8);
    numbers.add(1);
    numbers.forEach( (n) -> { System.out.println(n); } );
  }

}
```

`forEach` recebe como parâmetro uma função lambda `Integer -> void`

Interfaces Funcionais em Java

- Interface com uma única função.
- Funciona como o "tipo" de uma função

```
interface StringFunction {
  String run(String str);
}
```

StringFunction tem o paper de uma função do "tipo"

StringFunction: String -> String

Exemplo de Interface Funcionais

```
public class Xpto {
  public static void main(String[] args) {
    StringFunction exclama = (s) -> s + "!";
    StringFunction pergunta = (s) -> s + "?";
    printFormatted("Olá", exclama);
    printFormatted("Olá", pergunta);
  }
  public static void printFormatted(String str,
                                    StringFunction format) {
    String result = format.run(str);
    System.out.println(result);
  }

}
```

Testes Automáticos de Exceções

Exceções devem ser testadas

- Objetivo: garantir que a exceção é lançada quando devido
- Teste sucede quando exceção é lançada
- Falha se exceção não é lançada
- Usar AssertThrows(

  Exceção.class,

  Função Lambda que lança exceção

)

  Função lambda do tipo *() -> void*

  Sem parâmetros nem return

Classe sob teste

```
public class Fajuta {
        static void calcula(int a) throws ProprioErro {
                If (a > 10)
                        throw new ProprioErro(a); // Lança instância nova classe
        }
        public static void main(String args[]){
                try{
                        Fajuta.calcula(1);          // Não gera exceção
                        Fajuta.calcula(300);        // Gera exceção
                }
                catch(ProprioErro e){               // Tratamento da nova exceção
                        System.err.println("Capturada " + e);
                }
        }
}
```

Teste da exceção

```
class TestaFajuta {
        @Test
        void testeSemExceção() {
                try {
                        Fajuta.calcula(7);
                }
                catch (ProprioErro e) {
                        fail();
                }
        }
        @Test
        void testaProprioErro(){
                assertThrows(
                        ProprioErro.class,              // Classe da exceção testada
                        () -> {Fajuta.calcula(300);}    // Execução postergada void -> void
                );
        }
}
```

Lista de exercícios

No computador com o Eclipse

Entrega até o final do dia

MAC321
Lab POO

- Professor: Marcelo Finger

  E-mail: mfinger@ime.usp.br