Computação Orientada a Objetos

Tratamento de Exceções

Profa. Thienne Johnson

EACH/USP

Conteúdo

- Java, como programar
  - Deitel & Deitel

- Capítulo 14

Exceções

- **Uma exceção é uma indicação de um problema que ocorre durante a execução de um programa**

- O tratamento de exceções permite aos programadores criar aplicativos que podem resolver (tratar) exceções

- Está relacionado tanto ao tratamento de erros irrecuperáveis do sistema quanto de situações alternativas à seqüência típica de eventos;

Visão Geral

- Os programas costumam testar condições para determinar como a execução de um programa deve prosseguir

Realize uma tarefa
Se a tarefa anterior não tiver sido executada corretamente
Realize processamento de erro
Realize a próxima tarefa
Se a tarefa anterior não tiver sido executada corretamente
Realize processamento de erro

Exemplo 1

```
tente {
        cortar(cebola);
        panela.adicionar(cebola);
        cortar(tomate);
        panela.adicionar(tomate);
        panela.adicionar(Oleo.medida(colher));
    }
comer();
```

Exemplo 1 – cont.

```
tente {
        cortar(cebola);
        panela.adicionar(cebola);
        cortar(tomate);
        panela.adicionar(tomate);
        panela.adicionar(Oleo.medida(colher));
    }
imprevisto(CortarDedo e) {
            dedo.aplicar(curativo);
    }
comer();
```

Visão Geral

- Embora funcione, mesclar a lógica do programa com o tratamento de erros pode dificultar a leitura, modificação, manutenção e a depuração dos programas

Visão Geral

- O tratamento de exceções permite aos programadores remover da “linha principal” de execução do programa o código do tratamento de erros
  - aprimora a clareza do programa

- Os programadores podem escolher quais exceções serão tratadas:
  - todas as exceções
  - todas as exceções de um certo tipo
  - todas as exceções de um grupo de tipos relacionados (hierarquia de herança)

Visão Geral

- O tratamento de exceções reduz a probabilidade de que erros sejam neglicenciados

- Resultado: produtos de software mais robustos e tolerantes a falhas!

Conceitos básicos

- Diz-se que uma exceção é lançada (isto é, a exceção ocorre) quando um método detecta um problema e é incapaz de tratá-lo

- Quando uma exceção ocorre dentro de um método, o método cria um objeto do tipo exceção:
  - contém informação a respeito do evento, incluindo seu tipo e o estado do programa no momento da ocorrência.

Ex: Divisão por zero *SEM* tratamento de exceções

```
import java.util.Scanner;
public class DivideByZeroNoExceptionHandling
{
   // demonstra o lançamento de uma exceção quando ocorre uma divisão por zero

   public static int quociente( int numerador, int denominador )
   {
      return numerador / denominador; // possivel divisao por zero
   } // fim de método quociente

   public static void main( String args[] )
   {
      Scanner scanner = new Scanner( System.in ); // scanner para entrada

      System.out.print( "Entre com um numerador inteiro: " );
      int numerador = scanner.nextInt();
      System.out.print( " Entre com um denominador inteiro: " );
      int denominador = scanner.nextInt();

      int result = quociente( numerador, denominador );
      System.out.printf("\nResult: %d / %d = %d\n", numerador, denominador, result );
   } // fim de main
} // fim da classe DivideByZeroNoExceptionHandling
```

Ex: Divisão por zero *SEM* tratamento de exceções

- Execução 1: Divisão bem sucedida!

```
Entre com um numerador inteiro: 100
Entre com um denominador inteiro: 7

Result: 100/7 = 14
```

Ex: Divisão por zero *SEM* tratamento de exceções

- Execução 2: Usuário insere o valor 0 como denominador...

```
Entre com um numerador inteiro: 100
Entre com um denominador inteiro: 0
Exception in thread "main" java.lang.ArithmeticException: /by zero
at DivideByZeroNoExceptionHandling.quociente(DivideByZeroNoExceptionHandling.
   java:8)
at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.
   java:20)
```

Ex: Divisão por zero *SEM* tratamento de exceções

- Execução 2: Usuário insere o valor 0 como denominador...

```
Entre com um numerador inteiro: 100
Entre com um denominador inteiro: 0
Exception in thread "main" java.lang.ArithmeticException: /by zero
at DivideByZeroNoExceptionHandling.quociente(DivideByZeroNoExceptionHandling.
   java:8)
at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.
   java:20)
```

Rastreamento de pilha: mensagem que inclui

1) nome da exceção **(java.lang.ArithmeticException)**
2) o problema que ocorreu **(/by zero)**
3) o caminho de execução que resultou na exceção, método por método

Ex: Divisão por zero *SEM* tratamento de exceções

- Execução 2: Usuário insere o valor 0 como denominador...

```
Entre com um numerador inteiro: 100
Entre com um denominador inteiro: 0
Exception in thread "main" java.lang.ArithmeticException: /by zero
at DivideByZeroNoExceptionHandling.quociente(DivideByZeroNoExceptionHandling.
   java:8)
at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.
   java:20)
```

- A exceção foi detectada na linha 20 do método **`main`**
- Cada linha contém o nome da classe e o método seguido pelo nome do arquivo e da linha
- Subindo a pilha, vê-se que a exceção ocorre na linha 8, no método **`quociente`**

Ex: Divisão por zero *SEM* tratamento de exceções

- Execução 2: Usuário insere o valor 0 como denominador...

```
Entre com um numerador inteiro: 100
Entre com um denominador inteiro: 0
Exception in thread "main" java.lang.ArithmeticException: /by zero
at DivideByZeroNoExceptionHandling.quociente(DivideByZeroNoExceptionHandling.
   java:8)
at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.
   java:20)
```

A linha superior da cadeia de chamadas indica o ponto de lançamento – ponto inicial onde a exceção ocorre

Está na linha 8 do método **`quociente`**

Ex: Divisão por zero *SEM* tratamento de exceções

- Execução 3: Usuário insere a string “Hello” como denominador...

```
Entre com um numerador inteiro: 100
Entre com um denominador inteiro: Hello
Exception in thread “main” java.util.InputMismatchException
at java.util.Scanner.throwFor(Unknow Source)
at java.util.Scanner.next(Unknow Source)
at java.util.Scanner.nextInt(Unknow Source)
at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.
   java:18)
```

Ex: Divisão por zero *SEM* tratamento de exceções

- Execução 3: Usuário insere a string "Hello" como denominador...

```
Entre com um numerador inteiro: 100
Entre com um denominador inteiro: Hello
Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Unknow Source)
at java.util.Scanner.next(Unknow Source)
at java.util.Scanner.nextInt(Unknow Source)
at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling. java:18)
```

Informa a ocorrência de uma **`InputMismatchException`** (pacote **`java.util`**)

A exceção foi detectada na linha 18 do método **`main`**.

Ex: Divisão por zero *SEM* tratamento de exceções

- Execução 3: Usuário insere a string “Hello” como denominador...

```
Entre com um numerador inteiro: 100
Entre com um denominador inteiro: Hello
Exception in thread “main” java.util.InputMismatchException
at java.util.Scanner.throwFor(Unknow Source)
at java.util.Scanner.next(Unknow Source)
at java.util.Scanner.nextInt(Unknow Source)
at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling. java:18)
```

Subindo a pilha, nota-se que a exceção ocorre no método **`nextInt`** (pacote **`java.util`**)
No lugar do nome do arquivo e linha, aparece o texto **`Unknown Source`**
A JVM não tem acesso ao código-fonte no local da exceção

Ex: Divisão por zero *SEM* tratamento de exceções

- Nas execuções 2 e 3, quando as exceções ocorrem e os rastreamentos são exibidos, o programa também se fecha.

- Com tratamento de exceções: o programa pode continuar mesmo que uma exceção tenha ocorrido!

Incluindo código em um bloco *try*

- O bloco *try* inclui:
  - o código que pode lançar (*throw*) uma exceção
  - e o código que não deve ser executado se ocorrer uma exceção (ie, que deve ser pulado se uma exceção for lançada)

- Um bloco *try* consiste na palavra-chave try seguida por uma sequência de código entre chaves {}

- Este código, ou os métodos nele invocados, podem criar objetos derivados do tipo (classe) *Exception* sinalizando condições de exceção;

Exemplo – Bloco *try*

```
try // lê dois números e calcula o quociente
   {
      System.out.print( "Entre com um numerador inteiro: " );
      int numerador = scanner.nextInt();
      System.out.print( " Entre com um denominador inteiro: " );
      int denominator = scanner.nextInt();
      int result = quocient( numerador, denominador );
      System.out.printf( "\nResult: %d / %d = %d\n", numerador,
denominador, result );

 } // fim de try
```

O método **`nextInt`** lança uma exceção **`InputMismatchException`** se o valor lido não for um inteiro válido

Capturando exceções

- Um bloco *catch* (também chamado de *handler* de exceção) captura (ie, recebe) e trata uma exceção

- Um bloco *catch* inicia-se com a palavra-chave *catch* e é seguido por um parâmetro entre parênteses e um bloco de código entre chaves {}

- Pelo menos um bloco *catch* ou um bloco *finally* (discutido depois) deve se seguir imediatamente a um bloco *try*

Exemplo – Bloco *catch*

```
 catch ( InputMismatchException inputMismatchException ) {
      System.err.printf( "\nException: %s\n", inputMismatchException );
      scanner.nextLine(); // descarta entrada para o usuário tentar novamente
      System.out.println("Deve-se entrar com numeros inteiros. Tente de
novo.\n" );
   } // fim de catch

catch ( ArithmeticException arithmeticException )   {
      System.err.printf( "\nException: %s\n", arithmeticException );
      System.out.println("Zero é um denominador inválido.Tente de novo.\n" );
   } // fim de catch
```

O primeiro bloco trata uma exceção **`InputMismatchException`**

O segundo bloco trata uma exceção **`ArithmeticException`**

Capturando exceções

```
try{
  código que pode gerar exceções

}

catch (TipoDeExceção ref) {
   código de tratamento da exceção
}

catch (TipoDeExceção2 ref2) {
   código de tratamento da exceção
}
```

- Todo bloco *catch* especifica entre parênteses um parâmetro de exceção que identifica o tipo (classe) de exceção que o *handler* pode processar

Capturando exceções

- Quando ocorrer uma exceção em um bloco *try,* o bloco *catch* que será executado é aquele cujo tipo de parâmetro corresponde à exceção que ocorreu

- O nome do parâmetro de exceção permite ao bloco *catch* interagir com um objeto de exceção capturado
  - Ex: invocar o método **toString** da exceção capturada, que exibe informações básicas sobre a exceção

Erros comuns

- É um erro de sintaxe colocar código entre um bloco *try* e seus blocos *catch* correspondentes

- Cada bloco *catch* pode ter apenas um parâmetro
  - Especificar uma lista de parâmetros de exceção separados por vírgula é um erro de sintaxe

- É um erro de compilação capturar o mesmo tipo de exceção em dois blocos *catch* diferentes em uma única cláusula *try*

Fluxo de controle

- Se ocorrer uma exceção em um bloco *try*, este termina imediatamente e o controle do programa é transferido para o primeiro dos blocos *catch* seguintes em que o tipo do parâmetro de exceção corresponda ao da exceção lançada no bloco *try*

```
try{
  código que pode gerar exceções

}

catch (TipoDeExceção ref) {
   código de tratamento da exceção
}

catch (TipoDeExceção2 ref2) {
   código de tratamento da exceção
}
```

Fluxo de controle

- Após a exceção ser tratada, o controle do programa não retorna ao ponto de lançamento porque o bloco *try* expirou
  - as variáveis locais do bloco também foram perdidas

- Em vez disso, o controle é retomado depois do último bloco *catch*
  - Isso é conhecido como modelo de terminação de tratamento de exceções

Dica de prevenção de erros

- Com o tratamento de exceções, um programa pode continuar executando (em vez de encerrar) depois de lidar com o problema

- Isso ajuda a assegurar o tipo de aplicativos robustos que colaboram para o que é chamado de computação de missão crítica

Nomes de parâmetros

- Programadores Java costumam usar somente a letra "**`e`**" como o nome de parâmetros de exceção

- Parâmetros de exceção podem ser nomeados com base no seu tipo
  - Ex:

tipo	nome
`InputMismatchException`	`inputMismatchException`
`ArithmeticException`	`arithmeticException`

Nomes de parâmetros

- Usar um nome de parâmetro de exceção que reflita o seu tipo promove a clareza do programa
  - lembra ao programador o tipo da exceção em tratamento

Utilizando a cláusula *throws*

- A parte da declaração de método localizada na linha 2 é conhecida como uma cláusula *throws*

```
public static int quociente( int numerador, int denominador )
throws ArithmeticException
{
   return numerador / denominador; // possível divisão por zero
} // fim de método quociente
```

Utilizando a cláusula *throws*

- Uma cláusula *throws* especifica as exceções que um método lança
- Essa cláusula aparece depois da lista de parâmetros e antes do corpo do método

```
public static int quociente( int numerador, int denominador )
      throws ArithmeticException
   {
      return numerador / denominador; // possível divisão por
     zero
   } // fim de método quociente
```

Utilizando a cláusula *throws*

- A cláusula *throws* contém uma lista de exceções separadas por vírgulas que o método lançará se ocorrer algum problema
- Essas exceções podem ser lançadas por instruções no corpo do método ou por métodos chamados no corpo

```
public static int quociente( int numerador, int denominador )
      throws ArithmeticException
   {
      return numerador / denominador; // possível divisão por zero
   } // fim de método quociente
```

Utilizando a cláusula *throws*

- Um método pode lançar exceções das classes listadas em sua cláusula *throws* ou de suas subclasses

- Ex: adicionamos a cláusula *throws* a esse aplicativo para indicar ao resto do programa que o método **quociente** pode lançar uma **ArithmeticException**

```
public static int quociente( int numerador, int denominador )
      throws ArithmeticException
   {
      return numerador / denominador; // possível divisão por zero
   } // fim de método quociente
```

Utilizando a cláusula *throws*

- Os clientes do método **quociente** são informados de que o método pode lançar uma **ArithmeticException** e de que a exceção deve ser capturada

```
public static int quociente( int numerador, int denominador )
      throws ArithmeticException
   {
      return numerador / denominador; // possível divisão por
     zero
   } // fim de método quociente
```

Ex:Tratando `ArithmeticException` e `InputMismatchException`

- Utilizando o tratamento de exceções para processar quaisquer **`ArithmeticException`** e **`InputMismatchException`** que possam surgir no programa

- Se o usuário cometer um erro, o programa captura e trata (lida com) a exceção
  - Permite ao usuário tentar inserir a entrada novamente

Ex:Tratando ArithmeticException e InputMismatchException

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class DivideByZeroWithExceptionHandling
{
   // demonstra o lançamento de uma exceção quando ocorre uma divisão
     por zero
   public static int quociente( int numerador, int denominador )
      throws ArithmeticException
   {
      return numerador / denominador; // possível divisão por zero
   } // fim de método quociente
//continua...
```

Ex:Tratando ArithmeticException e InputMismatchException

```
public static void main( String args[] )
   {
      Scanner scanner = new Scanner( System.in ); // scanner para entrada
      boolean continueLoop = true; // determina se mais tentativas são
      necessárias
      do
      {
         try { // lê dois números e calcula o quociente
            System.out.print( "Entre com um numerador inteiro: " );
            int numerador = scanner.nextInt();
            System.out.print( " Entre com um denominador inteiro: " );
            int denominador = scanner.nextInt();
            int result = quociente( numerador, denominador );
            System.out.printf( "\nResult: %d / %d = %d\n", numerador,
               denominador, result );
            continueLoop = false; // entrada bem-sucedida; fim de loop
         } // fim de try
//continua...
```

Ex:Tratando ArithmeticException e InputMismatchException

```
      catch ( InputMismatchException inputMismatchException )
        {
           System.err.printf( "\nException: %s\n", inputMismatchException );
           scanner.nextLine(); // descarta entrada para o usuário tentar novam.
           System.out.println("Deve-se entrar com numeros inteiros.Tente de
    novo.\n");
        } // fim de catch

      catch ( ArithmeticException arithmeticException )
        {
          System.err.printf( "\nException: %s\n", arithmeticException );
          System.out.println("Zero e um denominador invalido.Tente de novo.\n");
        } // fim de catch
    } while ( continueLoop ); // fim de do...while
  } // fim de main
} // fim da classe DivideByZeroWithExceptionHandling
```

Ex:Tratando `ArithmeticException` e `InputMismatchException`

- Execução 2: Usuário insere o valor 0 como denominador...

```
Entre com um numerador inteiro: 100
Entre com um denominador inteiro: 0

Exception: java.lang.ArithmeticException: /by zero
Zero e um denominador invalido. Tente de novo.

Entre com um numerador inteiro: 100
Entre com um denominador inteiro: 7

Result: 100/7 = 14
```

Ex:Tratando `ArithmeticException` e `InputMismatchException`

- Execução 3: Usuário insere a string “Hello” como denominador...

```
Entre com um numerador inteiro: 100
Entre com um denominador inteiro: Hello

Exception: java.util.InputMismatchException
Deve-se entrar com numeros inteiros. Tente de novo.

Entre com um numerador inteiro: 100
Entre com um denominador inteiro: 7

Result: 100/7 = 14
```

Usos típicos

- Quando o sistema pode se recuperar do erro: o tratador de erro implementa o procedimento de recuperação

- Quando o sistema não pode se recuperar do erro mas desejamos encerrar o programa de forma "limpa"

- Em projetos grandes que exijam tratamento de erros uniforme

Exercício 1

- Que tipos de exceções podem ser capturadas pelo código abaixo ?
- Há aguma contra-indicação no seu uso ?

```
try {
}
catch (Exception e)   {
}
```

Exercício 2

- Há algo errado com o código abaixo ?
- Ele vai compilar ?

```
try {
}
catch (Exception e) {
}
catch (ArithmeticException a) {
}
```

Exercício 3: qual a saída deste programa?

```
class Exemplo {
 public static void main (String[] args)
 { try   { teste(); }
   catch ( Exceção x)
         { System.out.println("Tratamento 3"); }
 }

 static void teste() throws Exceção
 { try {
          try  {throw new Exceção(); }
          catch ( Exceção x)
                { System.out.println("Tratamento 1"); }
          throw new Exceção();

        }
   catch (Exceção x){

           System.out.println("Tratamento 2");
           throw new Exceção(); }
 }

}

class Exceção extends Exception {}
```

Exercício 4 – Tratar exceção

```
public class Excecao1 extends Object {
   public static void main(String args[]) {
        byte dado[] = new byte[10];
        System.out.println("Digite um número");
        System.in.read(dado);
   }
}
```

Exercício 5 – ArrayIndexOutOfBondsException

```
public class Exc2 extends Object {

   public static void main(String args[]) {
         int a[] = new int[2];
         System.out.println(a[4]);
   }
}
```

Exercício 6 – Tratar exceção

```
class TestThrow{
    static int hexChar2int(char c) throws Exception{
            if(c>='0'& c<='9') return c-'0';
            if(c>='A'& c<='F') return c-'A'+10;
            if(c>='a'& c<='f') return c-'a'+10;
            throw new Exception("caractere nao e hexadecimal!");
}
    public static void main(String[] args){
            int i=hexChar2int('G');
            System.out.println("Tudo ok! i iniciou com:"+i);
    }
}
```