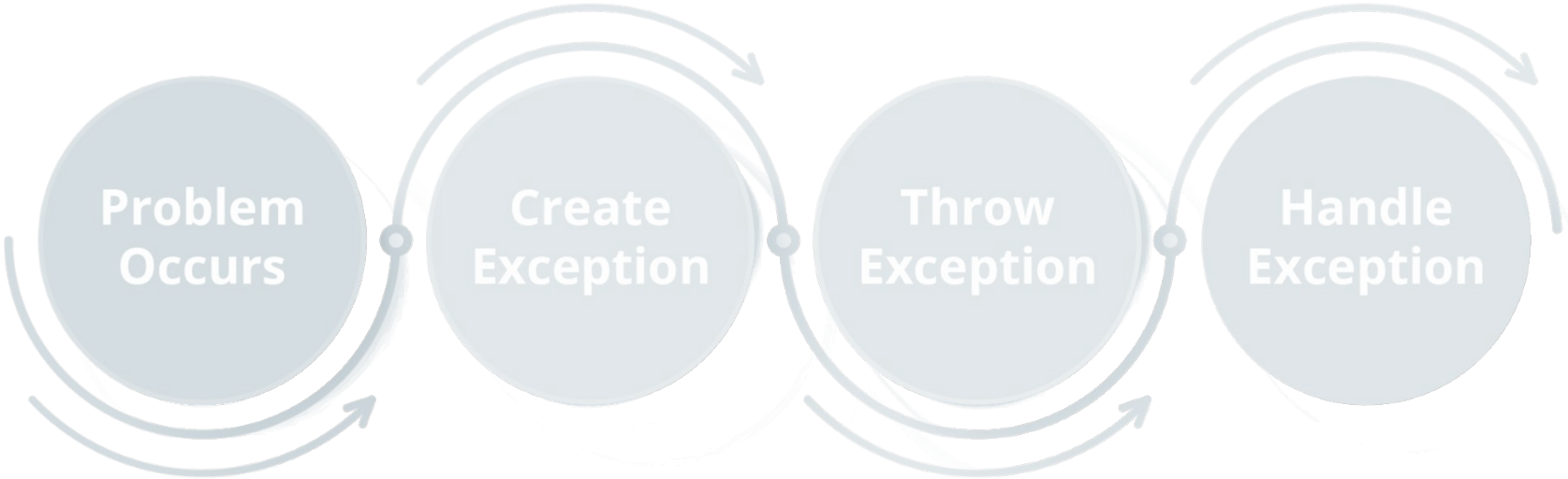


Aula 12

Tratamento de Exceções

Rafael Geraldeli Rossi

Introdução

- **Uma exceção é uma indicação de um problema que ocorre durante a execução de um programa**
- **Definição** (*Java Lessons* - Material Complementar): *"uma exceção é um evento que ocorre durante a execução de um programa que interrompe o fluxo normal da execução das instruções do programa"*
- O nome "exceção" significa que o problema não ocorre frequentemente

Introdução

```java
public class Teste {

    public int teste;

    public static void main(String args[]){

        int[] array = {0,5,8,7,10};

        for(int i=0;i<=5;i++){
            System.out.println(i + "- " + array[i]);
        }


    }

}
```

Localizar: Aqui | Anterior | Próximo

Saída ×

Console do Depurador × | Teste (run) ×

```
run:
0- 0
1- 5
2- 8
3- 7
4- 10
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at Teste.main(Teste.java:14)
/home/rafael/.cache/netbeans/8.1/executor-snippets/run.xml:53: Java returned: 1
FALHA NA CONSTRUÇÃO (tempo total: 1 segundo)
```

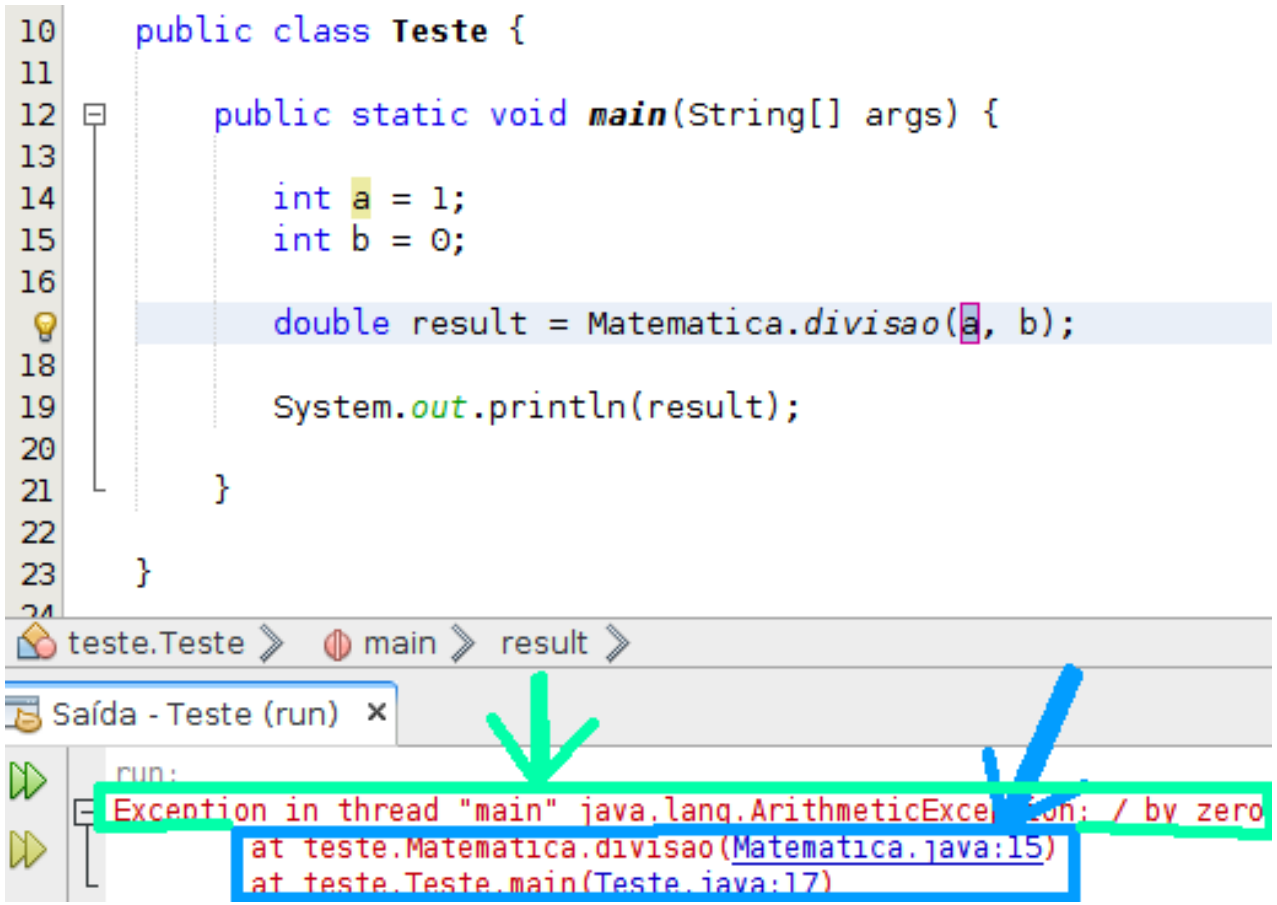
Introdução

- No decorrer deste curso já nos deparamos com alguns tipos de exceções:
  - `NullPointerException` → pode ocorrer quando uma referência `null` é utilizada onde se espera um objeto
  - `ArrayIndexOutOfBoundException` → ocorre quando é feita uma tentativa de acesso de um elemento fora dos limites de um `array`
  - `ArithmeticException: / by zero` → ocorre quando é feita um divisão entre inteiros por 0
  - `InputMismatchException` → ocorrem quando, por exemplo, o método `nextInt` da classe `Scanner` recebe uma `String` que não representa um inteiro válido

Introdução

- Ao ocorrer uma exceção, o Java exibe uma mensagem na tela contendo:
  - **O nome da exceção** → mensagem descritiva que indica o problema que ocorreu e a pilha de chamadas de método
  - **A pilha de chamadas de métodos (stack trace)** → cadeia de métodos que foram chamados no momento que ocorreu a exceção

Introdução


10 public class Teste {
11
12 public static void main(String[] args) {
13
14 int a = 1;
15 int b = 0;
16
double result = Matematica.divisao(a, b);
18
19 System.out.println(result);
20
21 }
22
23 }
teste.Teste
main
result
Saída - Teste (run)
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
at teste.Matematica.divisao(Matematica.java:15)
at teste.Teste.main(Teste.java:17)

Introdução

- Ao ocorrer uma exceção, e se essa exceção não for tratara, **o programa é encerrado**
- Porém, podemos **tratar as exceções → um programa pode continuar executando (em vez de encerrar) depois de lidar com um problema**
- O tratamento de exceções ajuda a **assegurar a robustez dos aplicativos**

Tratamento de Exceções

- Vale ressaltar que algumas exceções podem ser controladas com verificações que você mesmo pode programar:
  - Verificar ser será feita uma divisão por 0
  - Verificar se a *string* que o usuário digitou corresponde ao tipo de valor a qual ela será convertida
  - Acessar uma posição inválida de um vetor
  - Tentar acessar o conteúdo de um objeto que aponta para nulo
  - ...

Tratamento de Exceções

- Porém, algumas exceções são difíceis ou mesmo impossíveis de serem explicitamente programadas
  - Uma falha na comunicação da rede
  - Falha na renderização de uma interface gráfica
  - Falha ao executar uma impressão
  - Falha no disco ao ler uma arquivo
  - Tentar gravar dados em um arquivo no qual não se tem permissão de escrita
  - ...

Tratamento de Exceções

- Independente se podem ser evitadas via código ou não, as exceções que podem ocorrer no programa podem ser tratadas

Qual o efeito do tratamento de exceções?

Resposta: o objetivo principal do tratamento de exceções é manter o programa funcionando. Pode ser utilizado para exibir uma mensagem mais amigável ao usuário do que a mensagem de exceção do Java.

Tratamento de Exceções

- Quando uma exceção ocorre em um método, o método cria um objeto e manda esse objeto para o sistema (que poderá manipular esse objeto)
- Esse objeto, chamado de **objeto de exceção**, contém informações sobre o erro, incluindo seu tipo, e estado do programa quando o erro ocorreu
- Criar esse objeto e mandá-lo para o sistema é chamado de **lançar uma exceção**

Tratamento de Exceções

- O sistema irá procurar no método (vamos chamar de método A) que gerou a exceção pelo tratamento da exceção
- Se não houver tal tratamento no método que gerou a exceção, a exceção será lançada para o método B (método que chamou o método A)
- Se também não houver tratamento no método B, a exceção será lançada para o método C (método que chamou o método B, se houver), e assim por diante, até a exceção ser lançada para o método `main`
- Por fim, se não houver tratamento no método `main`, o sistema será encerrado

Tratamento de Exceções

- Lembrando que quando realizamos uma chamada de método, o sistema operacional coloca essa chamada em uma **pilha** (*stack*)
- Assim, o último método chamado estará no topo da pilha, a método que chamou o último método estará abaixo do topo.... e o método `main` estará no fundo da pilha
- Quando ocorre uma exceção e essa não é tratada, o Java exibe o rastreamento da pilha (*stack trace*) que irá exibir o caminho das chamadas de métodos até ocorrer a exceção

Tratamento de Exceções

```
1   public class Teste {
2
3       public int teste;
4
5       public static void main(String args[]){
6           medoto1();
7       }
8
9       public static void medoto1(){
10          metodo2();
11      }
12
13      public static void metodo2(){
14          int i = 10/0;
15      }
16
17  }
18
```

Localizar: Aqui | Anterior | Próximo

Teste > main >

Saída ×

Console do Depurador × | Teste (run) ×

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Teste.metodo2(Teste.java:14)
        at Teste.medoto1(Teste.java:10)
        at Teste.main(Teste.java:6)
/home/rafael/.cache/netbeans/8.1/executor-snippets/run.xml:53: Java returned: 1
FALHA NA CONSTRUÇÃO (tempo total: 1 segundo)
```

Incluindo código em um bloco `try`

- O conjunto de instruções no qual pode-se ocorrer uma exceção deve estar incluído em um **bloco** `try`
- Um bloco `try` consiste na palavra-chave `try` seguida por um bloco de código entre chaves ({ })

Exemplo de bloco `try`

```
try{
    ...
}
```

Capturando Exceções

- O bloco `try` deve ser seguido por um **bloco** `catch` caso deseje-se "capturar" a exceção e tratá-la
- Um **bloco** `catch` (também chamado de **cláusula** `catch`, **rotina de tratamento de exceção**, **gerenciador de exceção** ou **tratador de exceção**) captura (isto é, recebe) e trata uma exceção
- Um bloco `catch` inicia com a palavra-chave `catch` e é seguido por um parâmetro entre parênteses (chamado parâmetro de exceção) e um bloco de código entre chaves ({ })

Capturando Exceções

Exemplo de bloco try e bloco catch

```
try{
    ...
}catch(TipoDeExcecao variavelDeExcecao){
    ...
}
```

Capturando Exceções

- Cada bloco `catch` especifica, entre parênteses, um **parâmetro de exceção** que identifica o tipo de exceção (tipo de objeto de exceção) que se pode processar
- Quando ocorrer uma exceção em um bloco `try`, **o bloco `catch` que é executado é o primeiro cujo parâmetro corresponde ao tipo da exceção que ocorreu**
- O nome do parâmetro da exceção permite ao bloco `catch` interagir com um objeto de exceção capturado

Capturando Exceções

```
5   public class TesteExcecao {
6
7       public static void main(String[] args) {
8           Scanner teclado = new Scanner(System.in);
9           try{
10              int num1 = teclado.nextInt();
11              int num2 = teclado.nextInt();
12              double resultado = num1/num2;
13              System.out.println("Resultado inteiro da divisão é " + resultado);
14          }catch(ArithmeticException arithException){
15              System.err.println("Problema na operação matemática");
16              System.err.println(arithException.getMessage());
17          }catch(InputMismatchException inputException){
18              System.err.println("O valor que você digitou não é um inteiro!");
19              inputException.printStackTrace();
20          }
21      }
22
23  }
24
```

Bloco *try*

Capturando Exceções

```
5   public class TesteExcecao {
6
7       public static void main(String[] args) {
8           Scanner teclado = new Scanner(System.in);
9           try{
10              int num1 = teclado.nextInt();
11              int num2 = teclado.nextInt();
12              double resultado = num1/num2;
13              System.out.println("Resultado inteiro da divisão é " + resultado);
14          }catch(ArithmeticException arithException){
15              System.err.println("Problema na operação matemática");
16              System.err.println(arithException.getMessage());
17          }catch(InputMismatchException inputException){
18              System.err.println("O valor que você digitou não é um inteiro!");
19              inputException.printStackTrace();
20          }
21      }
22
23  }
24
```

Blocos *catch*

Capturando Exceções

```
5   public class TesteExcecao {
6
7       public static void main(String[] args) {
8           Scanner teclado = new Scanner(System.in);
9           try{
10              int num1 = teclado.nextInt();
11              int num2 = teclado.nextInt();
12              double resultado = num1/num2;
13              System.out.println("Resultado inteiro da divisão é " + resultado);
14          }catch(ArithmeticException arithException){
15              System.err.println("Problema na operação matemática");
16              System.err.println(arithException.getMessage());
17          }catch(InputMismatchException inputException){
18              System.err.println("O valor que você digitou não é um inteiro!");
19              inputException.printStackTrace();
20          }
21      }
22
23  }
24
```

Objetos de **Exceção**

Capturando Exceções

```
5   public class TesteExcecao {
6
7       public static void main(String[] args) {
8           Scanner teclado = new Scanner(System.in);
9           try{
10              int num1 = teclado.nextInt();
11              int num2 = teclado.nextInt();
12              double resultado = num1/num2;
13              System.out.println("Resultado inteiro da divisão é " + resultado);
            }catch(ArithmeticException arithException){
                System.err.println("Problema na operação matemática");
                System.err.println(arithException.getMessage());
            }catch(InputMismatchException inputException){
                System.err.println("O valor que você digitou não é um inteiro!");
                inputException.printStackTrace();
20          }
21      }
22
23  }
24
```

Interagindo com objetos de **Exceção**

Hierarquia de Exceções no Java

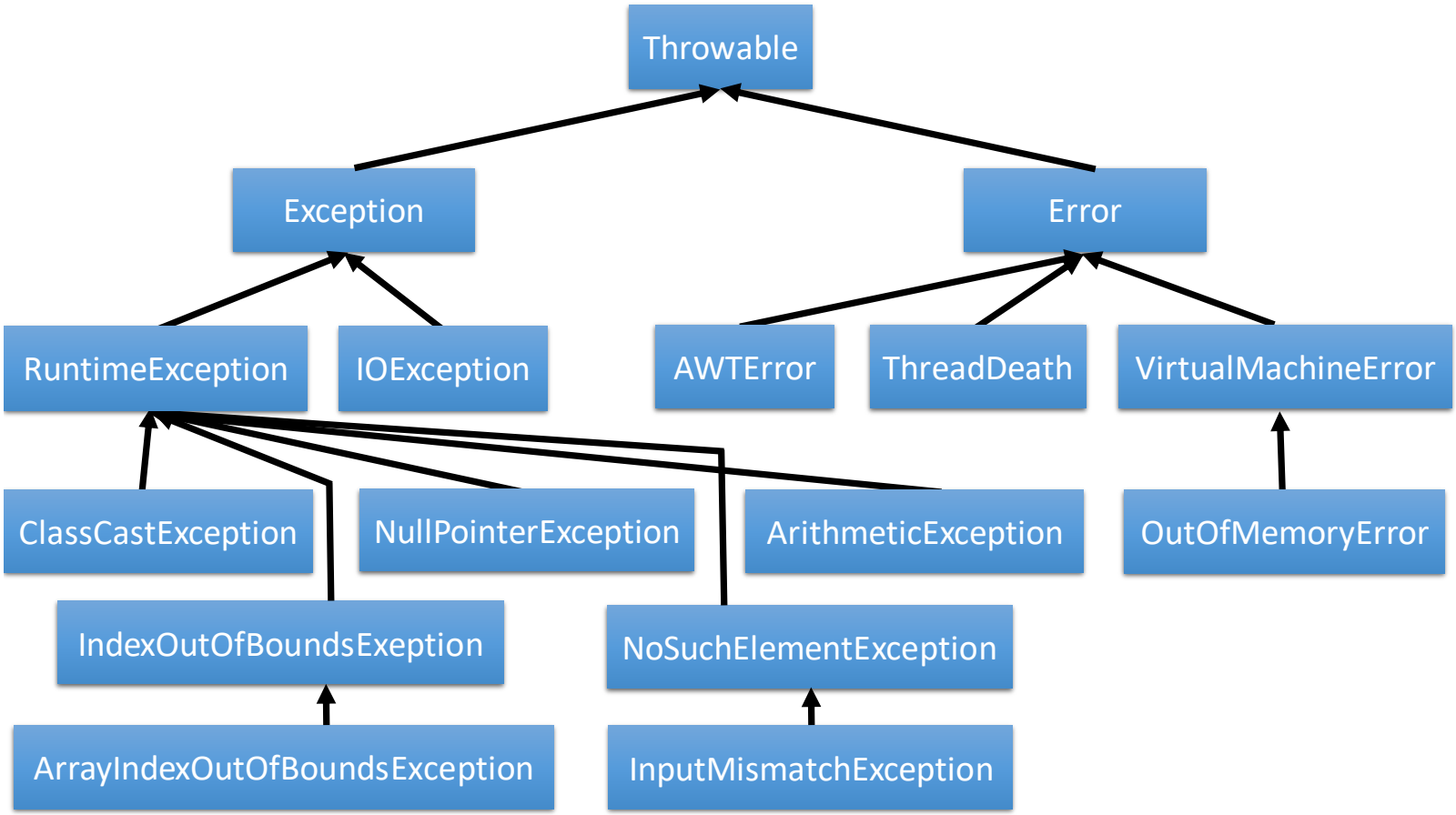
- **Todas as classes de exceção no Java herdam direta ou indiretamente da classe `Exception`, formando uma hierarquia de exceções**
- Pode-se estender a classe ou subclasses de `Exception` para criar suas próprias exceções
- A classe `Exception` herda da classe `Throwable` (subclasse de Object)
- **Só objetos `Throwable` podem ser usados com o mecanismo de tratamento de exceções**

Hierarquia de Exceções no Java

- A classe `Throwable` tem duas subclasses:
  - `Exception`: representam situações excepcionais que podem ocorrer em um programa Java e que podem ser capturados pelo aplicativo
  - `Error`: representam situações anormais que acontecem na JVM

- A hierarquia de exceções no Java contém centenas de classes (https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html)

Hierarquia de Exceções no Java



Capturando Exceções

- **Se um `catch` for escrito para capturar objetos de exceção de um tipo de superclasse, ele também pode capturar todos os objetos de subclasses dessa classe**
- Isso permite cláusula `catch` tratar erros relacionados com uma notação concisa e permite o processamento polimórfico das exceções relacionadas
- Com um único catch pode-se capturar mais de um tipo de exceção → os tipos de exceção devem ser separadas por um `pipe` ("|")

Capturando Exceções

```
1   public class Teste {
2
3       public static void main(String args[]){
4
5           try{
6
7           }catch(NullPointerException | ArithmeticException e){
8
9           }
10
11      }
12
13  }
```

Capturando Exceções

- **OBSERVAÇÃO 1:** obviamente pode-se fazer um tratamento específico para cada subclasse
- **OBSERVAÇÃO 2:** Se houver múltiplos blocos `catch` que correspondem a um tipo particular de exceção, somente o primeiro bloco `catch` correspondente executará na ocorrência de uma exceção
- **OBSERVAÇÃO 3:** não se pode criar dois blocos `catch` com o mesmo tipo de exceção

Capturando Exceções

```
2  import java.util.InputMismatchException;
3  import java.util.Scanner;
4
5  public class TesteExcecao {
6
7      public static void main(String[] args) {
8          Scanner teclado = new Scanner(System.in);
9
10         try{
11             System.out.println("Digite o primeiro número");
12             int num1 = teclado.nextInt();
13             System.out.println("Digite o segundo número");
14             int num2 = teclado.nextInt();
15             double resultado = num1/num2;
16             System.out.println("Resultado inteiro da divisão é " + resultado);
17         }catch(Exception e){
18             System.err.println("Deu erro!");
           }catch(ArithmeticException arithException){
20             System.err.println("Problema na operação matemática");
21             System.err.println(arithException.getMessage());
           }catch(InputMismatchException inputException){
23             System.err.println("O valor que você digitou não é um inteiro!");
24             teclado.nextLine();
               inputException.printStackTrace();
26         }
27
28     }
29
30 }
```

Um objeto de exceção de uma superclasse não pode aparecer antes de um objeto de exceção de uma subclasse

Capturando Exceções

- **OBSERVAÇÃO 4:** uma **exceção não capturada** é aquela para a qual não há nenhum bloco `catch` correspondente
- **OBSERVAÇÃO 5:** Vários tipos de tipos de exceções se encontram no pacote `java.lang`
- **OBSERVAÇÃO 6:** Várias exceções podem ser observadas em `https://docs.oracle.com/javase/8/docs/api/java/util/InputMismatchException.html`

Principais Métodos de um Objeto de Exceção

- Os principais métodos de um objeto de exceção são:
  - **public String getMessage()**: retorna uma mensagem sobre a exceção que ocorreu (essa mensagem é passada no construtor da classe `Throwable`)
  - **public void printStackTrace()**: retorna o *stack trace* impresso pelo método System.err
  - **public StackTraceElement[] getStackTrace()**: retorna um *array* contendo cada elemento do *stack trace*. O elemento no índice 0 representa o topo da pilha de chamadas e o último elemento representa o método mais ao fundo da pilha

Modelo de Término de Tratamento de Exceções

- **Se ocorrer uma exceção em um bloco `try`, o bloco `try` termina imediatamente e o controle do programa é transferido para o primeiro dos blocos `catch` (correspondente ao tipo de exceção)**

- Depois que a exceção é tratada, o controle do programa não retorna ao ponto de lançamento, uma vez que **o bloco `try` expirou** (inclusive se houver variáveis locais, estas são perdidas)

Modelo de Término de Tratamento de Exceções

- **O controle de execução retorna depois do último bloco** `catch`
- Isso é conhecido como **modelo de término de tratamento de exceções**
- **OBSERVAÇÃO:** algumas linguagens utilizam o **modelo de retomada de tratamento de exceções**, em que após uma exceção ser tratada, o controle é retomado logo depois do ponto de lançamento

Modelo de Término de Tratamento de Exceções

- No caso do Java, pode-se utilizar também o bloco `finally` para executar um conjunto de instruções:
  - Após a execução de um bloco `catch`, em caso da haver uma exeção lançada
  - Após executar o bloco `try` com sucesso
  - Caso haja uma exceção não tratada

- Um bloco `finally` é declarado utilizando a palavra-chave `finally` e o conteúdo do bloco é colocado entre chaves ({ })

Modelo de Término de Tratamento de Exceções

- O bloco ou cláusula `finally` é opcional
- Normalmente os blocos `try` são utilizados para a liberação de recursos ou para exibir uma mensagem padrão para todas as exceções capturadas
- **OBSERVAÇÃO 1:** pelo menos um **bloco** `catch` ou um **bloco** `finally` deve se seguir imediatamente ao bloco `try`

Modelo de Término de Tratamento de Exceções

Exemplo de blocos try, catch e finally

```
try{
    ...
}catch(TipoDeExcecao variavelDeExcecao){
    ...
}finally{
    ...
}
```

Tratando possíveis exceções ao ler e realizar a divisão de dois inteiros

```
16  public class TesteExcecao {
17
18      public static void main(String[] args){
19
20          Scanner scanner = new Scanner(System.in);
21          boolean continua = true;
22
23          do{
24              try{
                    System.out.print("Num1: ");
26                  int num1 = scanner.nextInt();
27                  scanner.nextLine();
28                  System.out.print("Num2: ");
29                  int num2 = scanner.nextInt();
30                  scanner.nextLine();
31                  int resultado = num1 / num2;
32                  System.out.print("Resultado: " + resultado + "\n");
33                  continua = false;
34              }catch(ArithmeticException ae){
35                  scanner.nextLine();
36                  System.err.println("Houve um erro no cálculo aritmético");
37                  ae.toString();
38              }catch(InputMismatchException ie){
                    scanner.nextLine();
40                  System.err.println("Houve ao converter a entrada do usuário em um valor inteiro");
41                  ie.toString();
42              }finally{
43                  System.out.println("Saindo do bloco try");
44              }
45          }while(continua == true);
46      }
47
48  }
```

Bloco try sem um bloco catch

```
1   public class Teste {
2
3       public int teste;
4
5       public static void main(String args[]){
6
7           try{
8               int i = 10/0;
9               System.out.println("Aqui 1");
10          }finally{
11              System.out.println("Aqui 2");
12          }
13
14      }
15
16  }
```

Localizar: Aqui Anterior Próximo

Saída ×

Console do Depurador × | Teste (run) ×

```
run:
Aqui 2
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Teste.main(Teste.java:11)
/home/rafael/.cache/netbeans/8.1/executor-snippets/run.xml:53: Java returned: 1
FALHA NA CONSTRUÇÃO (tempo total: 1 segundo)
```

Lançando Exceções com a Instrução `throw`

- **Você pode lançar suas próprias exceções utilizando a instrução `throw`, que é utilizada para indicar que uma exceção ocorreu**
- Uma instrução `throw` especifica um objeto a ser lançado
- Um operando de um `throw` pode ser qualquer classe derivada de `Throwable`

Lançando Exceções com a Instrução `throw`

- Um método que vá lançar uma exceção deve utilizar a palavra-chave `throws` e o tipo da exceção a frente da assinatura do método

```
10  public class Matematica {
11
12      public static int divisao(int a, int b) throws Exception {
13          if(a == 0 || b == 0){
14              throw new Exception();
15          }
16          return a / b;
17      }
18
19  }
```

Lançando Exceções com a Instrução throw

```
public void debitar(double valor) throws Exception{
    if(this.saldo >= valor){
        this.saldo -= valor;
        Transacao transacao = new Transacao("Debitou", valor);
        extrato.add(transacao);
    }else{
        throw new Exception("Saldo insuficiente");
    }

}
```

Lançando Exceções com a Instrução `throw`

- Quando se usa o `throws`, não precisamos tratar uma exceção no próprio método
- Se ocorrer uma exceção, essa será lançada para o método imediatamente abaixo na pilha de chamada de métodos
- O método que fez a chamada de outro método que gerou uma exceção, terá que fazer o tratamento do método ou lançar uma exceção para o próximo método da pilha
- **OBSERVAÇÃO:** quando se usa o `throws` estamos declarando que iremos lançar uma exceção

Lançando Exceções com a Instrução `throw`

- Pode-se especificar que um método poderá lançar mais de um tipo de exceção

- Para isso, deve-se separar os tipos de exceção por ","

```
public static void metodo2() throws NullPointerException, ArithmeticException{
}
```

Exceções Verificadas × Não Verificadas

- O Java distingue entre **exceções verificadas** e **exceções não verificadas**
- Essa distinção é importante → o compilador Java impõe um requisito "**capture-ou-declare**" (*catch-or-declare*) às exceções verificadas
- O tipo de uma exceção determina se a exceção é verificada ou é não verificada

Exceções Verificadas × Não Verificadas

- Todos os tipos de exceção que são subclasses diretas ou indiretas da classe `RuntimeException` (pacote `java.lang`) são exceções não verificadas → costumam ser causadas por deficiência no código do seu programa

- Todas as classes que herdam da classe `Exception` mas não da classe `RuntimeException` são consideradas exceções verificadas → tipicamente causadas por condições que não estão no controle do programa

Exceções Verificadas × Não Verificadas

- O compilador emitirá uma mensagem de erro indicando que a exceção deve ser capturada ou declarada caso um objeto sendo instanciado contenha um método que lance uma exceção verificada

- Ao contrário das exceções verificadas, o compilador Java não verifica o código para determinar se uma exceção não verificada é capturada ou declarada (em geral, pode-se impedir a ocorrência de exceções não verificadas pela codificação adequada)

Criando seu Próprio Tipo de Exceção

- Para criar sua própria exceção, basta estender um objeto de exceção

```
12  public class SaldoInsuficienteException extends Exception {
13
14      public SaldoInsuficienteException(){
15          super("Saldo Insuficiente");
16      }
17
18      public SaldoInsuficienteException(String mensagem){
19          super(mensagem);
20      }
21
22  }
```

Boas Práticas de Programação

- Alguns autores e programadores dizem que é uma péssima prática escrever blocos `catch` e lançar exceções considerando objetos de exceção do tipo `Exception`
- Isso causa um generalização do erro
  - Tratamento genérico para todas as exceções
  - Quem recebe a exceção não sabe o tipo de exceção que ocorre
- O bloco `finally` é geralmente usado para "limpar o programa", por exemplo, fechar os arquivos que foram abertos em um bloco `try`
- Outra regra de outro é sempre colocar uma mensagem explicativa na sua exceção

Vantagens

- Vantagens de se usar tratamento de exceções:
  1. Separar o código do gerenciamento de erro do código regular $\rightarrow$ toda a programação é feita em uma parte do código e todo o tratamento é feito em outra parte do código
  2. Propagar os erros para que sejam tratados em outros métodos $\rightarrow$ evitar programar o tratamento de exceção em todo método que é criado
  3. Possibilidade de agrupar tratamentos de exceções por tipos de objetos

Exercício

- Agora o Projeto Banco terá tratamento de exceções
  - Capturar exceções caso o usuário digite uma informação inválida (ex: uma palavra no lugar de um número)
  - Validar a transferência entre contas correntes utilizando tratamento de exceções
  - Criar a classe de exceção `SaldoInsuficiente`, a qual deve ser lançada nos métodos de saque e transferência

Exercício

- Agora o Projeto Banco terá tratamento de exceções:
  - Criar a classe de exceção `OperacaoNaoPermitida`, a qual deve ser lançada em todos os métodos que realizem uma operação bancária
  - Alterar os tipos de retorno dos métodos que retornam `boolean` para `void`
  - Fazer as alterações necessárias para o tratamento de Exceções na classe `Banco`

Material Complementar

- Professor Isidro Explica - Episódio 9 - Exceções

  `https://www.youtube.com/watch?v=7wbCL8klBlw`

- Exceções e controle de erros

  `https://www.caelum.com.br/apostila-java-orientacao-objetos/excecoes-e-controle-de-erros/`

- Tratando exceções em Java

  `https://www.devmedia.com.br/tratando-excecoes-em-java/25514`

Material Complementar

- Lesson: Exceptions

  https://docs.oracle.com/javase/tutorial/essential/exceptions/

- Exceções e controle de erros

  https://www.caelum.com.br/apostila-java-orientacao-objetos/excecoes-e-controle-de-erros/#11-4-outro-tipo-de-excecao-checked-exceptions

- Lidando com Exceptions

  http://blog.caelum.com.br/lidando-com-exceptions/?utm_source=Apostila_HTML&utm_campaign=FJ-11&utm_medium=referral

Imagem do Dia

Programação Orientada a Objetos
http://lives.ufms.br/moodle/

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br

Slides baseados em [Deitel and Deitel, 2010]

Referências Bibliográficas I

Deitel, P. and Deitel, H. (2010).
Java: How to Program.
How to program series. Pearson Prentice Hall, 8th edition.