

Linguagem de Programação Orientada a Objetos I

Tratamento de Exceções

Prof. Tales Bitelo Viegas

<https://fb.com/ProfessorTalesViegas>

The bottom of the slide features a decorative graphic consisting of several overlapping, wavy, horizontal bands. The colors transition from a light blue at the top, through a dark grey, to a light grey at the bottom. The waves create a sense of movement and depth.

Exceções

- ▶ São eventos que ocorrem durante a execução de um programa e quebram o fluxo normal de execução das instruções
- ▶ Indicam a ocorrência de erros ou condições excepcionais do programa

Exemplo 1

```
public class Divide{  
    public static void main(String[] args){  
        int dividendo = 10;  
        int divisor = 0;  
        System.out.println(dividendo/divisor);  
    }  
}
```

Exception in thread "main"
java.lang.ArithmeticException: / by zero



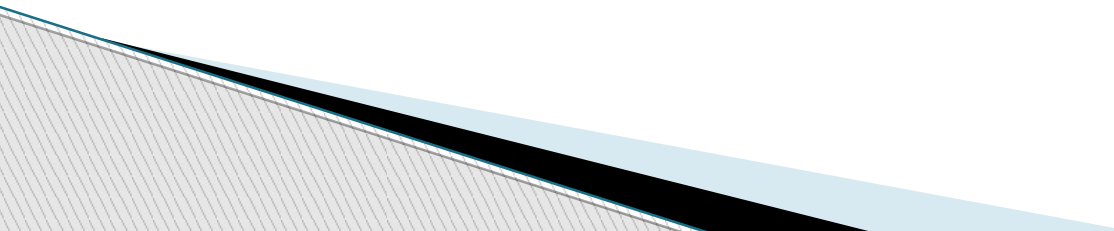
Exemplo 2

```
public class Vetor {  
    public static void main(String[] args) {  
        int[] vet = new int[3];  
        for (int i = 0; i <= 3; i++) {  
            vet[i] = 1;  
        }  
        for(int valor: vet){  
            System.out.println(valor);  
        }  
    }  
}
```

Exception in thread "main"

java.lang.ArrayIndexOutOfBoundsException: 3

Tipos de Exceções

- ▶ Erros aritméticos
 - ▶ Estouro de limite de array
 - ▶ Entrada de dados inválidos
 - ▶ Erros na manipulação de arquivos
 - ▶ Erros na comunicação com bancos de dados
 - ▶ Falhas na comunicação entre programas distribuídos
 - ▶ ...
- 

Introdução

- ▶ Erros de tempo de execução
 - 1. Erros de lógica de programação
 - ▢ Limites de vetores
 - ▢ Divisões por zero
 - 2. Erros devido a condições do ambiente de execução
 - ▢ Arquivo não encontrado
 - ▢ Rede fora do ar
 - 3. Erros graves, sem possibilidade de recuperação
 - ▢ Falta de memória
 - ▢ Erro interno da JVM
 - ▢ Falta de espaço em disco

Introdução

- ▶ Uma exceção interrompe o fluxo de execução do programa
- ▶ Este fluxo segue a exceção e se o método onde ela ocorrer não capturar, ela será propagada para o método que chamou este método e assim por diante
- ▶ Se não houver captura da exceção, ela irá causar o término do programa
- ▶ Mas se ela for capturada o controle pode ser recuperado

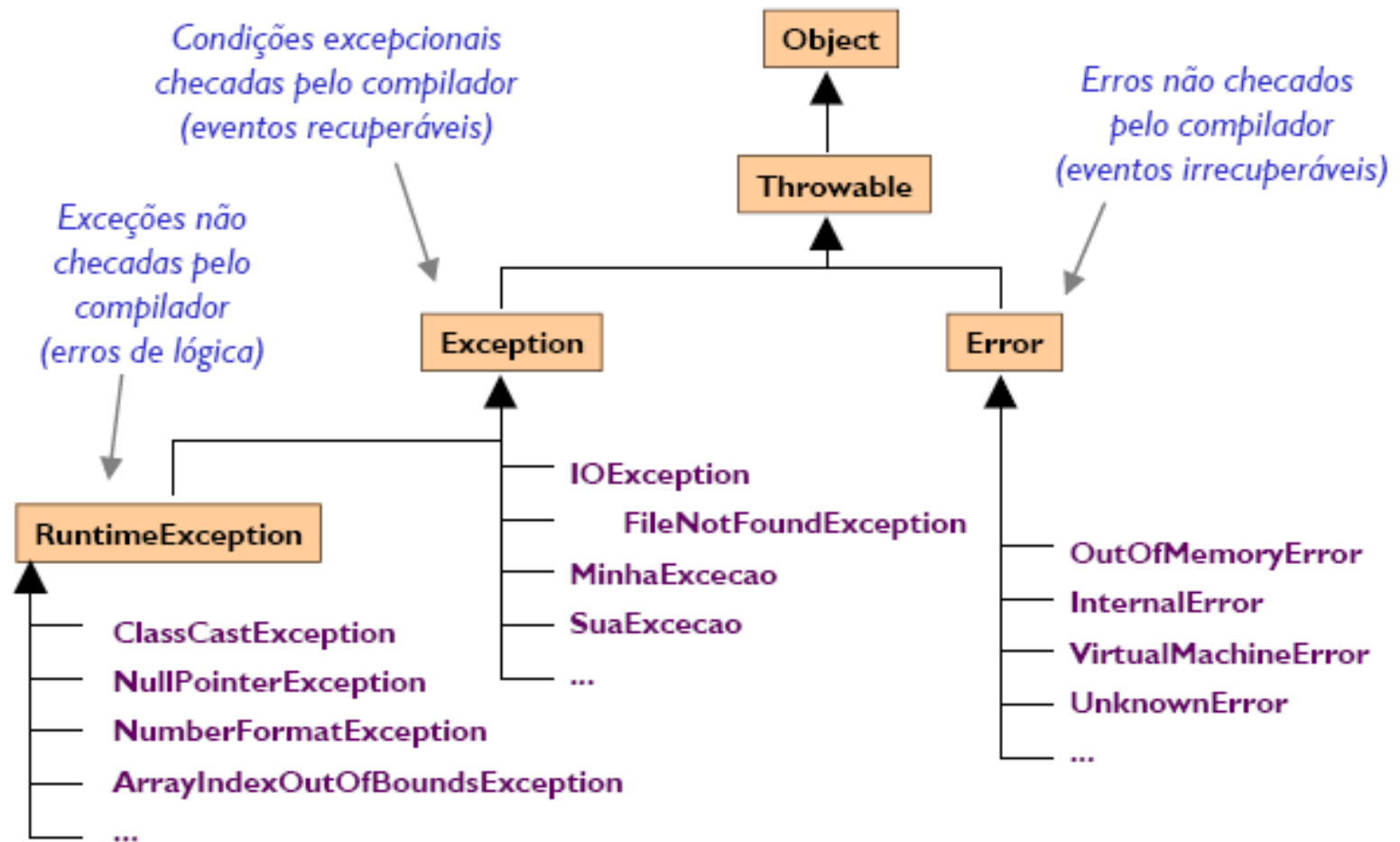
Classificação das Exceções

- ▶ Dois tipos de exceções:
 - Verificadas
 - Não verificadas
- ▶ Exceções verificadas
 - Ao chamar um método que lança uma exceção verificada, o programador deve dizer ao compilador o que está fazendo sobre a exceção se ela for lançada
 - Se devem a circunstâncias externas que o programador não pode evitar
 - Em geral, estendem a classe Exception
 - Exemplos: IOException, FileNotFoundException

Classificação das Exceções

- ▶ Exceções não verificadas
 - Em geral, estendem a classe RuntimeException ou Error
 - Exemplos: NumberFormatException, IllegalArgumentException, NullPointerException

Classificação das Exceções



Capturando uma exceção

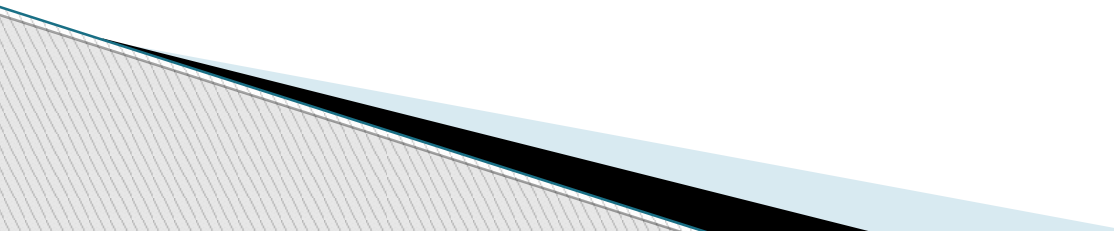
- ▶ Capturando um exceção
 - Um exceção pode/deve ser detectada e tratada
 - O código que potencialmente poderá gerar uma exceção é delimitada pelo bloco *try*
 - Cada bloco *try* contém uma ou mais chamadas de método que podem causar uma exceção e cláusulas *catch* para todos os tipos de exceção que o bloco *try* pode tratar

Usando try/catch/finally

```
try {  
    ...  
} catch (Excecao1 e1) {  
    ...  
} catch (Excecao2 e2) {  
    ...  
} finally {  
    ...  
}
```

Exemplo


```
public class Divide{  
    public static void main(String[] args){  
        int dividendo = 10;  
        int divisor = 0;  
  
        try {  
            System.out.println(dividendo/divisor);  
        } catch (ArithmeticException e){  
            System.out.println("Divisao por zero");  
        }  
    }  
}
```



Exemplo de Exceções 2

```
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;
```

```
public class TesteExcecoes1 {  
    public static void main(String[] args) {
```



```
        try {  
            BufferedReader in = new BufferedReader(new  
                InputStreamReader(System.in));  
            System.out.println("Quantos anos voce tem ?");  
            String inputLine = in.readLine();  
            int idade = Integer.parseInt(inputLine);  
            idade++;  
            System.out.println("No proximo ano voce tera: " + idade);
```

Capturando
uma exceção



```
        }  
        catch (IOException exception){  
            System.out.println("Erro de I/O: " + exception);  
        }
```

```
    }  
}
```

Exemplo de Exceções 2

- ▶ Considerações sobre o Exemplo 2:
 - Caso não ocorra uma exceção
 - ▢ As linhas de código dentro do bloco `try` são executadas normalmente
 - ▢ O bloco *catch* é ignorado
 - Caso ocorra a exceção:
 - ▢ As linhas de código dentro do bloco `try` são executadas até o ponto que gerou a exceção (as demais linhas são ignoradas)
 - ▢ As instruções do bloco `catch` são executadas

Exemplo de Exceções 3

```
import java.io.*;
public class TesteExcecoes2 {
    public static void main(String[] args) {
        try {
            BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
            System.out.println("Quantos anos voce tem ?");
            String inputLine = in.readLine();
            int idade = Integer.parseInt(inputLine);
            idade++;
            System.out.println("No proximo ano voce tera: " + idade);
        }
        catch (IOException exception){
            System.out.println("Erro de I/O: " + exception);
        }
        catch (NumberFormatException exception){
            System.out.println("A entrada nao eh um numero ! ");
        }
    }
}
```

Capturando
múltiplas exceções



Exemplo de Exceções 3

- ▶ Considerações sobre o Exemplo 3:
 - Caso não ocorra uma exceção
 - ▢ As linhas de código dentro do bloco `try` são executadas normalmente
 - ▢ Os blocos *catch* são ignorados
 - Caso ocorra a exceção:
 - ▢ As linhas de código dentro do bloco `try` são executadas até o ponto que gerou a exceção (as demais linhas são ignoradas)
 - ▢ As instruções do bloco `catch` correspondente são executadas

Exemplo de Exceções 3.1

```
import java.io.*;
public class TesteExcecoes31 {
    public static void main(String[] args) {
        try {
            BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
            System.out.println("Quantos anos voce tem ?");
            String inputLine = in.readLine();
            int idade = Integer.parseInt(inputLine);
            idade++;
            System.out.println("No proximo ano voce tera: " + idade);
        }
        catch (IOException|NumberFormatException exception){
            System.out.println("Aconteceu um erro: " + exception);
        }
    }
}
```

Capturando
múltiplas exceções



Exemplo de Exceções 4

```
import java.io.*;
public class TesteExcecoes4 {
    public static void main(String[] args) {
        try {
            BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
            System.out.println("Quantos anos voce tem ?");
            String inputLine = in.readLine();
            int idade = Integer.parseInt(inputLine);
            idade++;
            System.out.println("No proximo ano voce tera: " + idade);
        } catch (IOException exception){
            System.out.println("Erro de I/O: " + exception);
        }
        catch (NumberFormatException exception){
            exception.printStackTrace();
            System.exit(1);
        }
    }
}
```

Impressão da
cadeia de chamadas
de métodos que
leva a exceção



Exemplo de Exceções 4

- ▶ Considerações sobre o Exemplo 4:
 - Quando ocorre a exceção a cláusula *catch* pode analisar esse objeto para descobrir mais detalhes sobre a falha
 - Exemplo:
 - ▢ `exception.printStackTrace()`
 - ▢ Obtêm a impressão da cadeia de chamadas de método que leva à exceção

Exemplo de Exceções 5

```
import java.io.*;
public class TesteExcecoes5 {
    public static void main(String[] args) {
        try {
            BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
            System.out.println("Quantos anos voce tem ?");
            String inputLine = in.readLine();
            int idade = Integer.parseInt(inputLine);
            idade++;
            System.out.println("No proximo ano voce tera: " + idade);
        }
        catch (Exception exception){
            // detonamos as excecoes
            // na verdade elas nao foram tratadas
        }
    }
}
```

Detonando as
Exceções



Exemplo de Exceções 5

- ▶ Considerações sobre o Exemplo 5:
 - Quando ocorre a exceção a cláusula *catch* não determina com precisão qual exceção ocorreu
 - Muito utilizado para o compilador não reclamar da verificação de exceções (porém não é bom utilizar)
 - Este tratador é fictício
 - Já as exceções foram projetadas para transmitir relatórios do problema para um tratador competente

Exemplo de Exceções 6

```
import java.io.*;
public class TesteExcecoes6 {
    public static void main(String[] args) {
        try {
            BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
            System.out.println("Quantos anos voce tem ?");
            String inputLine = in.readLine();
            int idade = Integer.parseInt(inputLine);
            idade++;
            System.out.println("No proximo ano voce tera: " + idade);
        }
        catch (IOException exception){
            System.out.println("Erro de I/O: " + exception); }
        catch (NumberFormatException exception){
            System.out.println("A entrada nao eh um numero ! "); }
        finally{
            System.out.println("Eh sempre executado independentemente
de ocorrer uma excecao ! "); }
    } }
}
```

Cláusula
finally



Exemplo de Exceções 6

- ▶ Considerações sobre o Exemplo 6:
 - Uso da cláusula *finally*
 - Quando é necessário fazer algo no caso de ocorrência de qualquer exceção
 - O código da cláusula *finally* é executado sempre que o fluxo de código sai do bloco *try* de qualquer uma das maneiras:
 - ▢ Após completar a última instrução do bloco *try*;
 - ▢ Quando uma exceção foi lançada no bloco *try* que está sendo passado para esse chamador do método;
 - ▢ Quando uma exceção foi lançada no bloco *try* que foi tratado por uma das cláusulas *catch*

Gerando uma exceção

- ▶ No mecanismo de tratamento de exceções:
 - As exceções não podem ser negligenciadas
 - Elas podem ser tratadas por um tratador competente
- ▶ Detecção de condição de Erro
 - É necessário “lançar” (com throw) um objeto de exceção apropriado
 - **throw**: lança uma exceção e transfere o controle para um tratador para esse tipo de exceção
 - Procurar uma classe apropriada de exceção (a API possui muitas classes para sinalizar todos os tipos de condições excepcionais)

Lançando uma exceção

- ▶ Passos para lançar uma exceção:
 - Criar um objeto derivado da classe *Exception*
 - Lançar a exceção através da cláusula *throw*
 - Quando a exceção é lançada o método termina imediatamente
 - O mecanismo de tratamento de exceção repassa o objeto “*Exception*” para o manipulador de exceção apropriado
 - Sempre prefira escolher classes de exceção da própria API

Exemplo - lançando uma exceção

...

```
public class ContaBancaria {  
    public void sacar(double valor){  
        if (valor > saldo) {  
            IllegalArgumentException exception = new  
                IllegalArgumentException("O valor  
                    ultrapassa o saldo !");  
            throw exception;  
        }  
        saldo = saldo - valor;  
    }  
    ...  
}
```

...

Criando classes de exceção

- ▶ Projetando seus próprios tipos de exceção
 - Nenhum tipo de exceção-padrão descreve com a clareza um erro específico
 - Projetar o seu próprio tipo de exceção
 - Pode projetar suas próprias subclasses de tipos de exceção `Exception` ou `RuntimeException`
 - Exemplo: saque em uma conta bancária

Criando classes de exceção

- ▶ Exemplo: saque de uma conta bancária

...

```
public class ContaBancaria2 {  
    public void sacar(double valor){  
        if (valor > saldo) {  
            throw new InsufficientFundsException(  
                "Saque de " + valor + " excede o saldo de " +  
                saldo);  
        }  
        saldo = saldo - valor;  
    }  
    ...  
}
```

...

Criando classes de exceção

- ▶ Criando a classe de exceção:
 - Verificada ou não verificada ?
 - Estender Exception ou RuntimeException ?
 - Classe de exceção:
 - ▢ Costume fornecer dois construtores:
 - ▢ Um construtor *default*
 - ▢ Um construtor que aceita uma string de mensagem descrevendo a razão da exceção

Criando classes de exceção

- ▶ Classe `InsufficientFundsException`

```
public class InsufficientFundsException extends RuntimeException {  
  
    public InsufficientFundsException(){  
    }  
  
    public InsufficientFundsException(String reason){  
        super(reason);  
    }  
  
}
```

Delegando a Exceção

- ▶ Outra forma de tratar exceções é simplesmente “não tratá-las”.
- ▶ Neste caso, podemos informar para a JVM que, caso aconteça alguma exceção no método desenvolvido, quem deverá tratá-la é o código que o invocou
- ▶ Informamos isto através da palavra-chave **throws** na definição do método

Exemplo

...

```
public class ContaBancaria2 {  
    public void sacar(double valor) throws InsufficientFundsException{  
        if (valor > saldo) {  
            throw new InsufficientFundsException(  
                "Saque de " + valor + " excede o saldo de " +  
                saldo);  
        }  
        saldo = saldo - valor;  
    }  
    ...  
}  
...
```

