



## Java e Orientação a Objetos

### Capítulo: Tratamento de exceções

<https://devsuperior.com.br>

Prof. Dr. Nelio Alves

1

## Discussão inicial sobre exceções

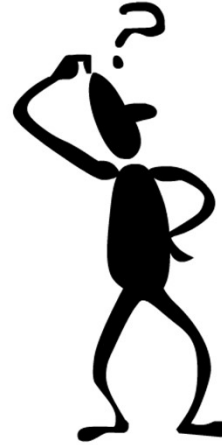
<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

2

Atenção: você não vai compreender  
tudo desta aula ainda.

Mas tudo ficará claro com os  
exemplos práticos nas próximas aulas



3

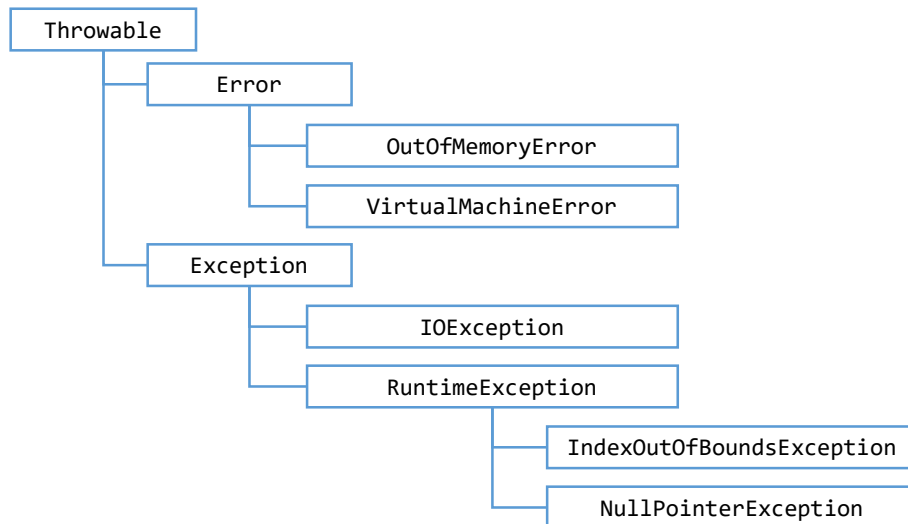
## Exceções

- Uma exceção é qualquer condição de erro ou comportamento inesperado encontrado por um programa **em execução**
- Em Java, uma exceção é um objeto herdado da classe:
  - `java.lang.Exception` - o compilador obriga a tratar ou propagar
  - `java.lang.RuntimeException` - o compilador não obriga a tratar ou propagar
- Quando lançada, uma exceção é propagada na pilha de chamadas de métodos em execução, até que seja capturada (tratada) ou o programa seja encerrado

4

## Hierarquia de exceções do Java

<https://docs.oracle.com/javase/10/docs/api/java/lang/package-tree.html>



5

## Por que exceções?

- O modelo de tratamento de exceções permite que erros sejam tratados de forma consistente e flexível, usando boas práticas
- Vantagens:
  - Delega a lógica do erro para a classe responsável por conhecer as regras que podem ocasionar o erro
  - Trata de forma organizada (inclusive hierárquica) exceções de tipos diferentes
  - A exceção pode carregar dados quaisquer

6

# Estrutura try-catch

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

7

## Estrutura try-catch

- Bloco try
  - Contém o código que representa a execução normal do trecho de código que **pode** acarretar em uma exceção
- Bloco catch
  - Contém o código a ser executado caso uma exceção ocorra
  - Deve ser especificado o tipo da exceção a ser tratada (upcasting é permitido)
- Demo

8

## Sintaxe

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

9

```
package application;

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

public class Program {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        try {
            String[] vect = sc.nextLine().split(" ");
            int position = sc.nextInt();
            System.out.println(vect[position]);
        }
        catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Invalid position!");
        }
        catch (InputMismatchException e) {
            System.out.println("Input error");
        }

        System.out.println("End of program");

        sc.close();

    }
}
```

10

# Pilha de chamadas de métodos

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

11

```
package application;

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

public class Program {

    public static void main(String[] args) {

        method1();

        System.out.println("End of program");

    }

    public static void method1() {
        System.out.println("***METHOD1 START***");
        method2();
        System.out.println("***METHOD1 END***");
    }

    public static void method2() {
        System.out.println("***METHOD2 START***");
        Scanner sc = new Scanner(System.in);

        try {
            String[] vect = sc.nextLine().split(" ");
            int position = sc.nextInt();
            System.out.println(vect[position]);
        }
        catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Invalid position!");
            e.printStackTrace();
            sc.next();
        }
        catch (InputMismatchException e) {
            System.out.println("Input error");
        }
        sc.close();
        System.out.println("***METHOD2 END***");
    }
}
```

12

# Bloco finally

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

13

## Bloco finally

- É um bloco que contém código a ser executado independentemente de ter ocorrido ou não uma exceção.
- Exemplo clássico: fechar um arquivo, conexão de banco de dados, ou outro recurso específico ao final do processamento.

Sintaxe:

```
try {  
    }  
catch (ExceptionType e) {  
    }  
finally {  
    }
```

14

```
package application;

import java.io.File;
import java.io.IOException;
import java.util.Scanner;

public class Program {

    public static void main(String[] args) {

        File file = new File("C:\\temp\\in.txt");
        Scanner sc = null;
        try {
            sc = new Scanner(file);
            while (sc.hasNextLine()) {
                System.out.println(sc.nextLine());
            }
        } catch (IOException e) {
            System.out.println("Error opening file: " + e.getMessage());
        } finally {
            if (sc != null) {
                sc.close();
            }
        }
    }
}
```

15

## Criando exceções personalizadas

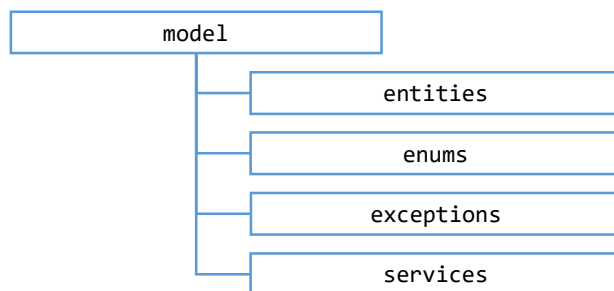
<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

16



## Sugestão de pacotes "model"



17

## Problema exemplo

Fazer um programa para ler os dados de uma reserva de hotel (número do quarto, data de entrada e data de saída) e mostrar os dados da reserva, inclusive sua duração em dias. Em seguida, ler novas datas de entrada e saída, atualizar a reserva, e mostrar novamente a reserva com os dados atualizados. O programa não deve aceitar dados inválidos para a reserva, conforme as seguintes regras:

- Alterações de reserva só podem ocorrer para datas futuras
- A data de saída deve ser maior que a data de entrada

Reservation
- roomNumber : Integer - checkin : Date - checkout : Date
+ duration() : Integer + updateDates(checkin : Date, checkout : Date) : void

18

## Examples

Room number: **8021**  
 Check-in date (dd/MM/yyyy): **23/09/2019**  
 Check-out date (dd/MM/yyyy): **26/09/2019**  
 Reservation: Room 8021, check-in: 23/09/2019, check-out: 26/09/2019, 3 nights

Enter data to update the reservation:  
 Check-in date (dd/MM/yyyy): **24/09/2019**  
 Check-out date (dd/MM/yyyy): **29/09/2019**  
 Reservation: Room 8021, check-in: 24/09/2019, check-out: 29/09/2019, 5 nights

Room number: **8021**  
 Check-in date (dd/MM/yyyy): **23/09/2019**  
 Check-out date (dd/MM/yyyy): **21/09/2019**  
 Error in reservation: Check-out date must be after check-in date

19

## Examples

Room number: **8021**  
 Check-in date (dd/MM/yyyy): **23/09/2019**  
 Check-out date (dd/MM/yyyy): **26/09/2019**  
 Reservation: Room 8021, check-in: 23/09/2019, check-out: 26/09/2019, 3 nights

Enter data to update the reservation:  
 Check-in date (dd/MM/yyyy): **24/09/2015**  
 Check-out date (dd/MM/yyyy): **29/09/2015**  
 Error in reservation: Reservation dates for update must be future dates

Room number: **8021**  
 Check-in date (dd/MM/yyyy): **23/09/2019**  
 Check-out date (dd/MM/yyyy): **26/09/2019**  
 Reservation: Room 8021, check-in: 23/09/2019, check-out: 26/09/2019, 3 nights

Enter data to update the reservation:  
 Check-in date (dd/MM/yyyy): **24/09/2020**  
 Check-out date (dd/MM/yyyy): **22/09/2020**  
 Error in reservation: Check-out date must be after check-in date

20

## Resumo da aula

- Solução 1 (muito ruim): lógica de validação no programa principal
  - Lógica de validação não delegada à reserva
- Solução 2 (ruim): método retornando string
  - A semântica da operação é prejudicada
    - Retornar string não tem nada a ver com atualização de reserva
    - E se a operação tivesse que retornar um string?
  - Ainda não é possível tratar exceções em construtores
  - Ainda não há auxílio do compilador: o programador deve "lembrar" de verificar se houve erro
  - A lógica fica estruturada em condicionais aninhadas
- Solução 3 (boa): tratamento de exceções

<https://github.com/acenelio/exceptions1-java>

21

## Resumo da aula

- Cláusula throws: propaga a exceção ao invés de trata-la
- Cláusula throw: lança a exceção / "corta" o método
- Exception: compilador obriga a tratar ou propagar
- RuntimeException: compilador não obriga
- O modelo de tratamento de exceções permite que erros sejam tratados de forma consistente e flexível, usando boas práticas
- Vantagens:
  - Lógica delegada
  - Construtores podem ter tratamento de exceções
  - Possibilidade de auxílio do compilador (Exception)
  - Código mais simples. Não há aninhamento de condicionais: a qualquer momento que uma exceção for disparada, a execução é interrompida e cai no bloco catch correspondente.
  - É possível capturar inclusive outras exceções de sistema

22

## Exercício de fixação

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

23

## Exercício de fixação

Fazer um programa para ler os dados de uma conta bancária e depois realizar um saque nesta conta bancária, mostrando o novo saldo. Um saque não pode ocorrer ou se não houver saldo na conta, ou se o valor do saque for superior ao limite de saque da conta. Implemente a conta bancária conforme projeto abaixo:

Account
- number : Integer - holder : String - balance : Double - withdrawLimit : Double
+ deposit(amount : Double) : void + withdraw(amount : Double) : void

24

## Examples

Enter account data

Number: **8021**

Holder: **Bob Brown**

Initial balance: **500.00**

Withdraw limit: **300.00**

Enter amount for withdraw: **100.00**

New balance: 400.00

Enter account data

Number: **8021**

Holder: **Bob Brown**

Initial balance: **500.00**

Withdraw limit: **300.00**

Enter amount for withdraw: **400.00**

Withdraw error: The amount exceeds withdraw limit

25

Enter account data

Number: **8021**

Holder: **Bob Brown**

Initial balance: **500.00**

Withdraw limit: **300.00**

Enter amount for withdraw: **800.00**

Withdraw error: The amount exceeds withdraw limit

Enter account data

Number: **8021**

Holder: **Bob Brown**

Initial balance: **200.00**

Withdraw limit: **300.00**

Enter amount for withdraw: **250.00**

Withdraw error: Not enough balance

<https://github.com/acenelio/exceptions2-java>

<https://www.youtube.com/watch?v=AiluJqJ0r8A>

26