AULA 03: Polimorfismo, Exceções e Collections

Agradecimentos ao material do Profº **Bruno Nogueira**

Apresentação do Curso

Conteúdo

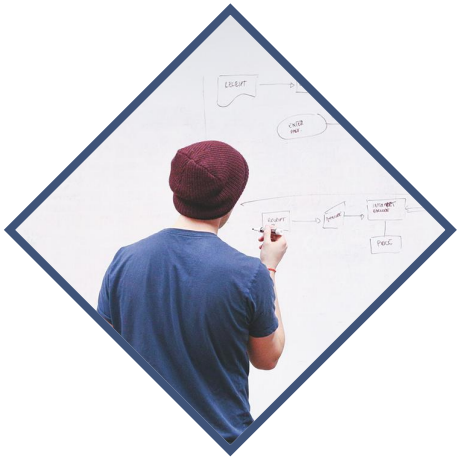
Polimorfismo

Collections

Tratamento de exceções

www.facom.ufms.br
https://hackatruck.com.br/

Lecionadores



Kaio Mitsuharu Lino Aida

kaiomudkt@gmail.com

Mateus Ragazzi Balbino

mateusragazzi.b@gmail.com

Acadêmicos de Sistemas de Informação

Mário de Araújo Carvalho

mariodearaujocarvalho@gmail.com

Acadêmico de Ciência da Computação.

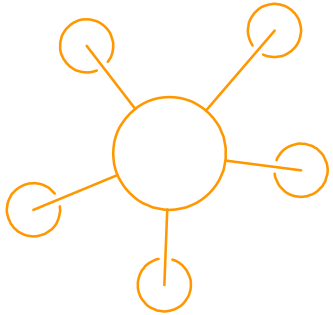
3

Aula

Polimorfismo

Orientação a Objetos

Polimorfismo



Polimorfismo

- **Poli** = Muitas; **Morphos** = Formas
- Em POO, um objeto pode ter **muitas formas**

Polimorfismo

Polimorfismo é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar **métodos que têm a mesma identificação, assinatura, mas comportamentos distintos**, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse.

Polimorfismo

- Fortemente associado à herança
- Primeiro, precisamos entender que um objeto de uma **subclasse pode ser utilizado como um objeto da superclasse**, mas não o contrário

Polimorfismo

- **Um objeto de uma subclasse tem TODOS os atributos e métodos de um objeto da superclasse.** Logo, todas as funcionalidades oferecidas pela superclasse (mais genérica) também são oferecidas pela subclasse (mais específica)

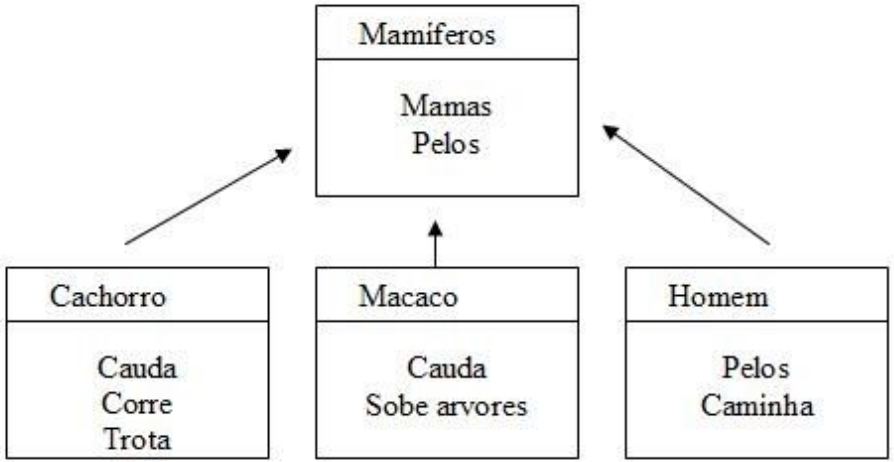
Polimorfismo

- O contrário, entretanto, não é verdade. A **subclasse pode criar novas funcionalidades que não estarão presentes na superclasse** ou alterar o comportamento das funcionalidades da superclasse. Logo, um objeto da superclasse não pode ser utilizado onde espera-se um objeto da subclasse

Polimorfismo - UML Classe - Herança

Herança

Polimorfismo

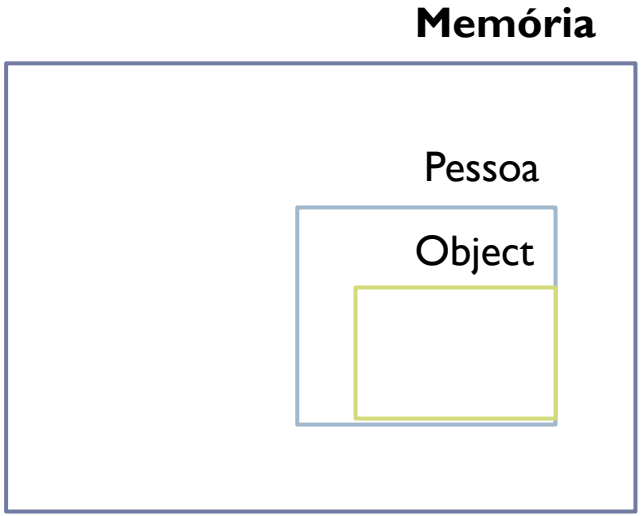


Polimorfismo

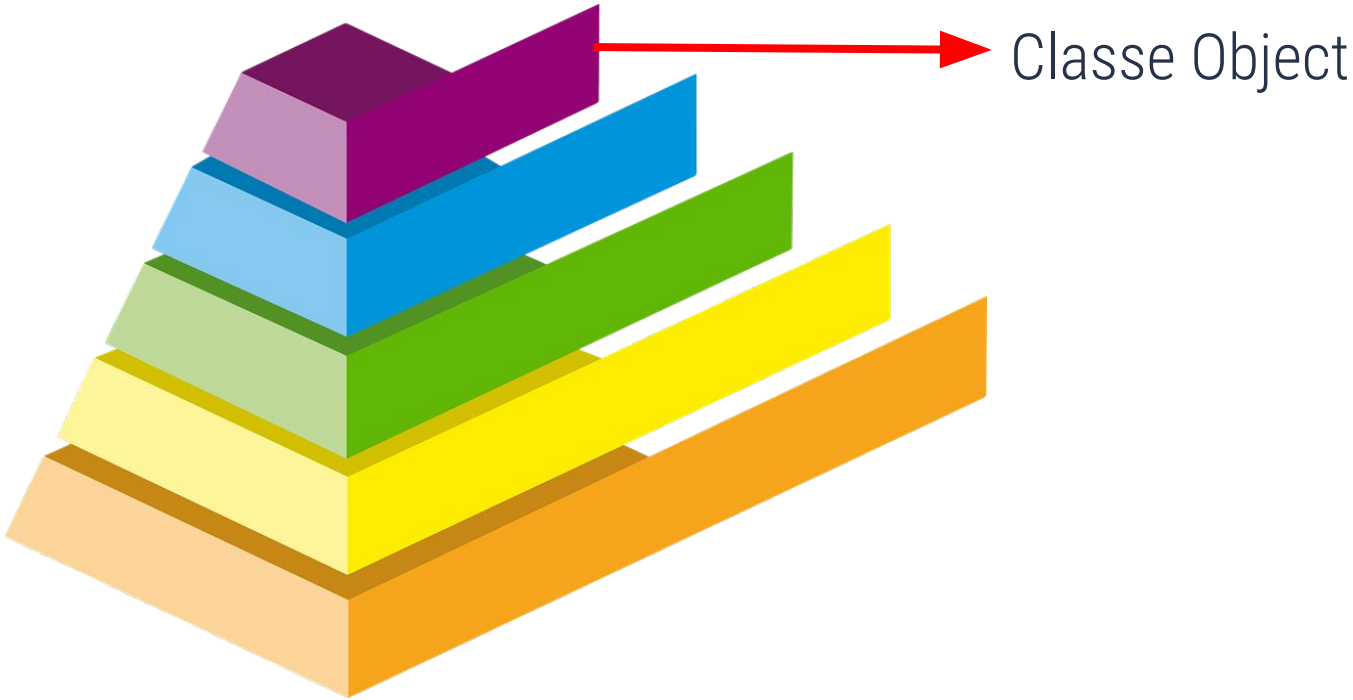
- Object o1 = new **Object**();
  - //o1 pode referenciar qualquer subtipo
- Object o2 = new **String**();
- Object o3 = new **Pessoa**();

Polimorfismo

Neste exemplo a variável de referência **o3** só irá conseguir enxergar a parte referente a classe **Object** da instância criada



Polimorfismo



Polimorfismo

```
public class Pessoa {
    private String nome;
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

Polimorfismo

```
public class Funcionario {
    private double salario;
    public double getSalario() {
        return salario;
    }
    public void setSalario(double salario) {
        this.salario = salairo;
    }
}
```

Polimorfismo

Funciona. É um objeto do tipo Funcionario e tem o método getSalario()

```
public class Teste {
    public static void main(String[] args) {
        Funcionario f1 = new Funcionario();
        Pessoa      f2 = new Funcionario();
        f1.getNome();
        f2.getNome();
    }
}
```

Polimorfismo

```
public class Teste {
    public static void main(String[] args) {
        Funcionario f1 = new Funcionario();
        Pessoa      f2 = new Funcionario();
        f1.getNome();
        f2.getNome();
        f1.getSalario();
        f2.getSalario();
    }
}
```

Erro de compilação.
É um objeto do tipo Pessoa e não tem o método getSalario()

Tipos de Polimorfismo

- Sobrecarga de métodos
- Sobrescrita de métodos

Sobrecarga de Métodos

- Compile time polymorphism ou static binding
- Podemos ter **vários métodos dentro de uma mesma classe com o mesmo nome, mas assinaturas (retorno e parâmetros) diferentes**
  - Devem ter um número diferente de parâmetros
  - Se tiverem o mesmo número de parâmetros, estes devem ser de tipos diferentes (o nome dos parâmetros não importa)

Sobrecarga de Métodos

```
public class Aluno {
    ....
    public Aluno(String nome) {
        ....
    }
    public Aluno(String nome, String RGA) {
        ....
    }
}
```

Sobrescrita de Métodos

- Acontece na herança, quando a subclasse sobrepõe/**sobrescreve o método original**
- Provêm uma nova implementação para um método existente e visível da superclasse
- Também chamado de Runtime polymorphism ou dynamic binding
- Método é escolhido se dá em tempo de execução e não mais em tempo de compilação

Sobrescrita de Métodos

- A escolha do método será chamado depende do tipo do objeto que recebe a mensagem
- Método de sobrescrito possui a mesma assinatura do método da classe herdada

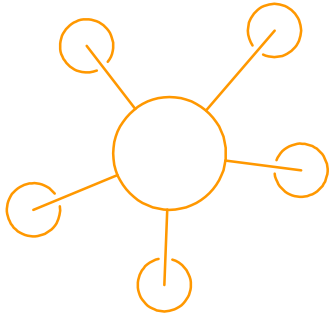
Sobrescrita de Métodos

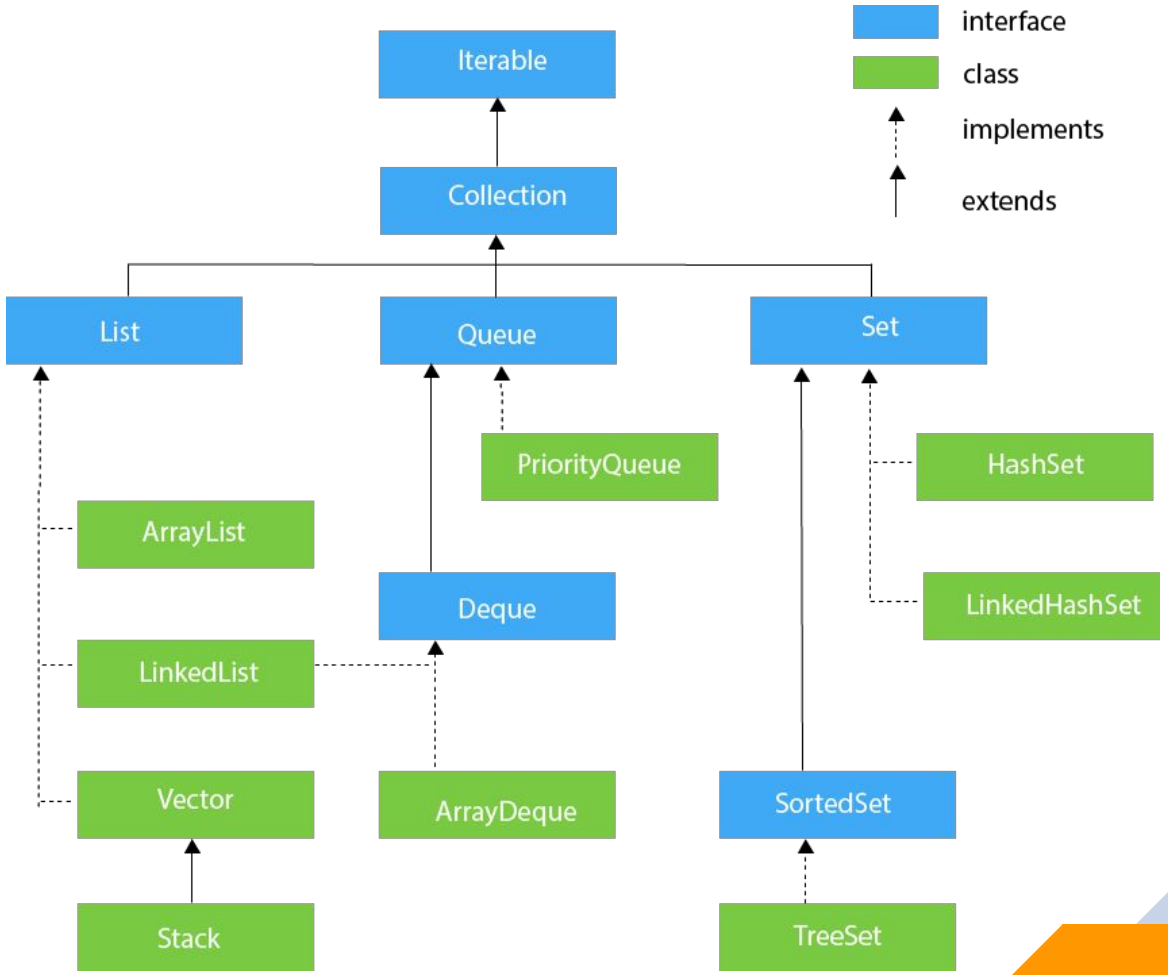
```
public class Aluno extends Pessoa {
    ....
    @Override
    public void imprimir() {
        System.out.print(this.nome);
        System.out.print(this.RGA);
    }
}
```

```
public class Pessoa {
    ....
    public void imprimir() {
        System.out.print(this.nome);
    }
}
```

Orientação a Objetos

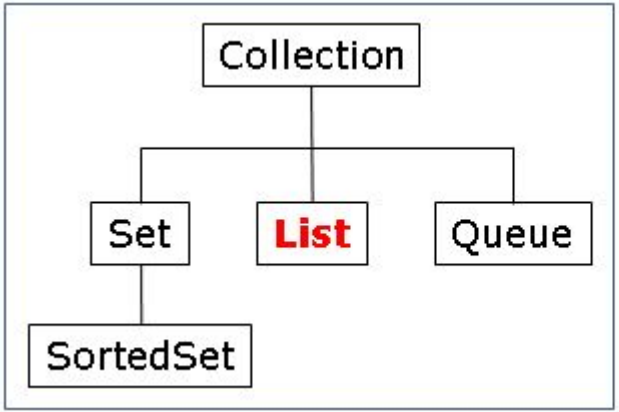
Collections




interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

List - Listas

- Especialização de Collection
- Vetor dinâmico
- Possui métodos:
  - add()
  - get()
  - remove()
  - clear()



Queue - Fila

- Especialização de Collection
- Fila first-in first-out
- Possui métodos:
  - add()
  - peek()
  - poll()
  - remove()

Set - Conjunto

- Especialização de Collection
- Conjunto de objetos
- Não possui ordem e duplicação de objetos
- Possui métodos:
  - add()
  - get()
  - remove()

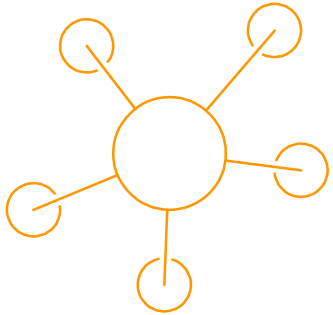
Lista e polimorfismo

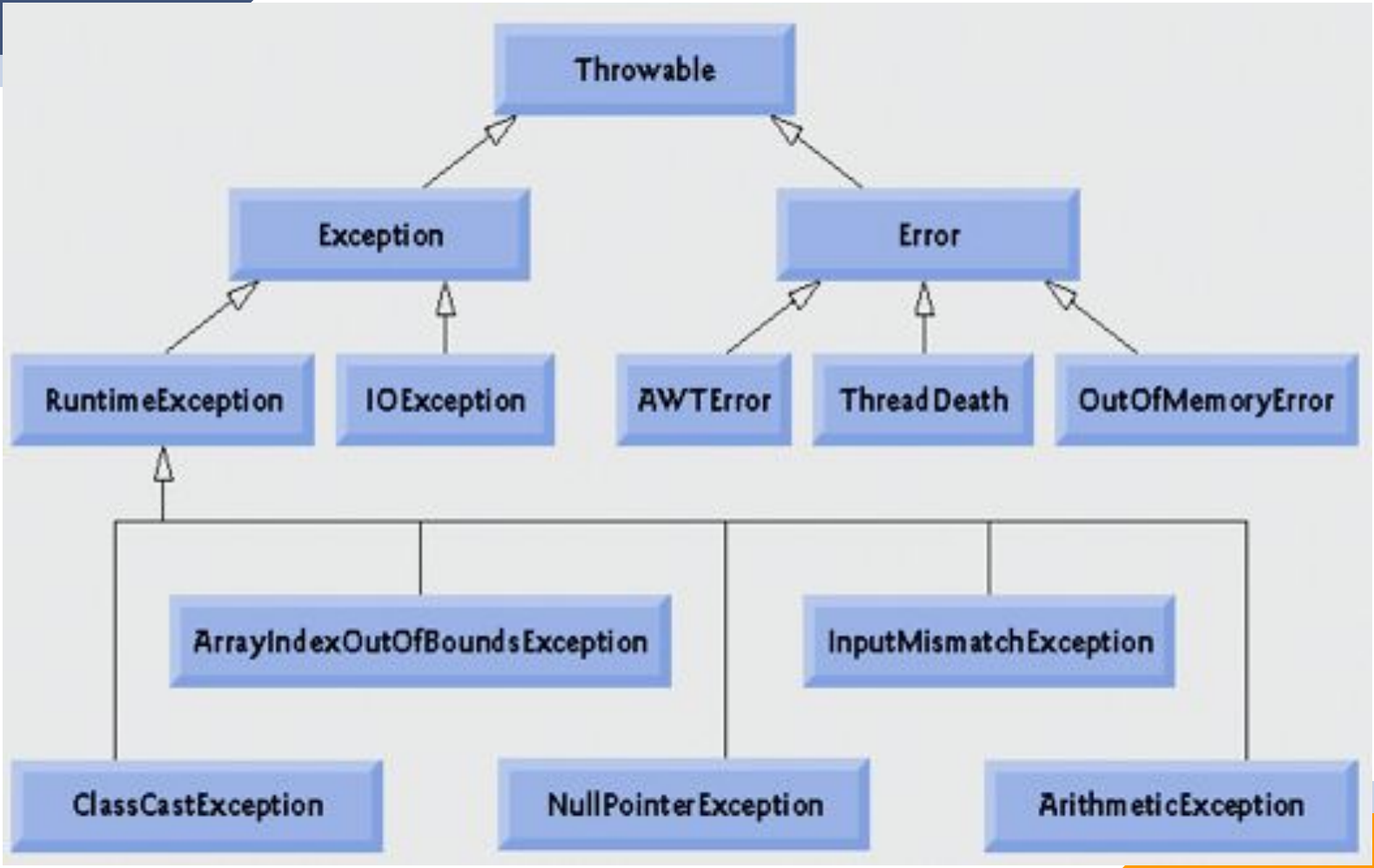
```
List carros;
carros = new ArrayList();
carros = new LinkedList();

Set setor;
setor = new HashSet();
setor = new LinkedHashSet();
```

Orientação a Objetos

Exceptions




Throwable
Exception
Error
RuntimeException
IOException
AWTError
ThreadDeath
OutOfMemoryError
ArrayIndexOutOfBoundsException
InputMismatchException
ClassCastException
NullPointerException
ArithmeticException

O que é?

- Situações inesperadas que ocorrem em tempo de execução
  - Pelo usuário: caracteres inválidos, divisão por zero, etc.
  - Pelo código: posição inválida de um vetor, manipulação de objeto null, etc.

Como funciona?

- Java difere exceções de erros
- Exceções (exceptions): são falhas inerentes ao seu programa
  - Podem ser contornadas, isto é, não implicam necessariamente na interrupção do seu programa
  - ArrayIndexOutOfBoundsException, InputMismatchException
- Erros (errors):são falhas inerentes ao ambiente
  - Não podem ser contornadas pelo programa e implicam na interrupção do mesmo
  - OutOfMemoryError, AWTError, etc

Lançamento de exceções

- Exceções são lançadas por trechos de código ou pelo ambiente Java quando detectam algum tipo de situação inesperada
- Diretiva **throw**

```
public void imprimePosicaoVetor(String [] vet, int pos){
        if(pos < 0 || pos >= vet.length)
                throw new ArrayIndexOutOfBoundsException();
        else
                System.out.println(vet[pos]);
}
```

Captura de exceções

- Quando uma exceção é lançada, ela deve ser capturada e tratada. Caso contrário,seu programa será encerrado
  - Blocos try,catch e finally

Exemplo exceção não tratada

Exceções podem ser capturadas e tratadas imediatamente no momento em que são geradas ou podem ser propagadas

```
public void imprimePosicaoVetor(String [] vet, int pos){
        if(pos < 0 || pos >= vet.length)
                throw new ArrayIndexOutOfBoundsException();
        else
                System.out.println(vet[pos]);
}
```

Exemplo exceção tratada

Exceções podem ser capturadas e tratadas

```
public void imprimePosicaoVetor(String [] vet, int pos){
        try {
                if(pos < 0 || pos >= vet.length)
                        throw new ArrayIndexOutOfBoundsException();
                else
                        System.out.println(vet[pos]);
        }
        catch (ArrayIndexOutOfBoundsException ex)
                ex.printStackTrace();
}
```

Exemplo exceção tratada

```
public void imprimePosicaoVetor(String [] vet, int pos) throws
ArrayIndexOutOfBoundsException {
        if(pos < 0 || pos >= vet.length)
                throw new ArrayIndexOutOfBoundsException();
        else
                System.out.println(vet[pos]);
}
```

Propagada – força quem chamou o método a capturar e tratar da maneira mais adequada

Mensagens

- Basicamente, é possível utilizar três formas de imprimir as mensagens de erro
  - printStackTrace(): imprime a pilha de chamadas de métodos e a numeração das linhas em que ocorreram as chamadas dos métodos
  - getMessage(): retorna uma String com a mensagem a ser exibida quando a exceção for lançada. Esta mensagem pode ser passada como parâmetro do construtor da exceção
  - toString(): retorna uma String com o nome e a descrição da exceção

Tipos de mensagens

```
catch (ArrayIndexOutOfBoundsException ex){
        System.err.println(ex.getMessage());
}
catch (ArrayIndexOutOfBoundsException ex){
        ex.printStackTrace();
}
```

Dúvidas?

Kaio Mitsuharu Lino Aida
kaiomudkt@gmail.com
Mateus Ragazzi Balbino
mateusragazzi.b@gmail.com
Mário de Araújo Carvalho
mariodearaujocarvalho@gmail.com