Programação Orientada a Objetos

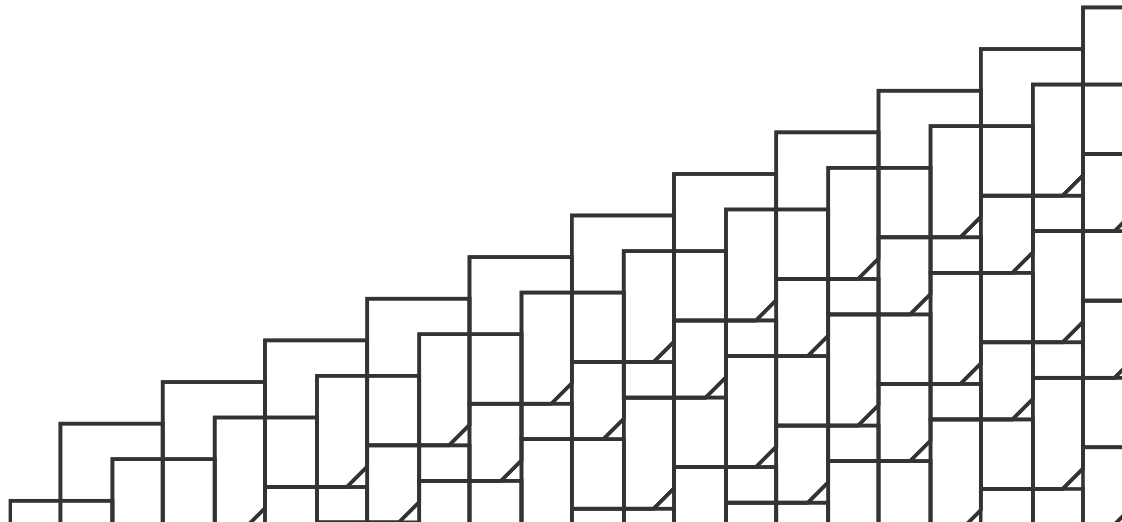
Introdução

Estrutura de Dados II

Prof. Me. André Kishimoto
http://lattes.cnpq.br/7395582872076146

Prof. Dr. Jean Marcos Laine
http://lattes.cnpq.br/4953261018941841

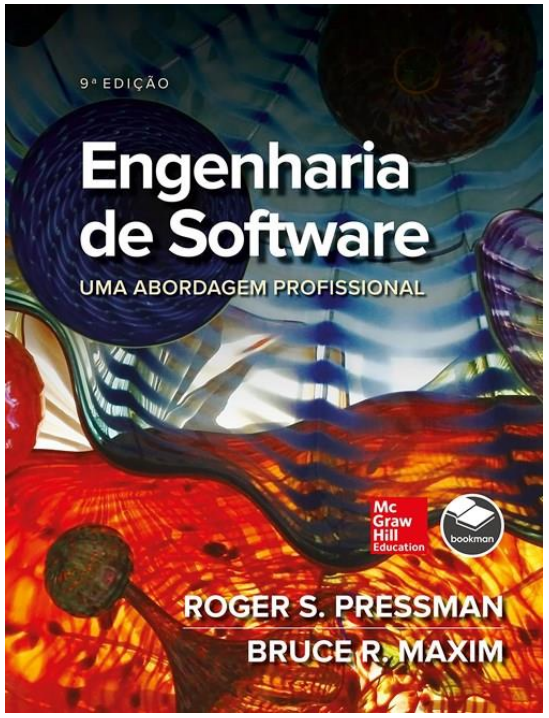
Prof. Dr. Ivan Carlos Alcântara de Oliveira
https://orcid.org/0000-0002-6020-7535



Programação Orientada a Objetos

É uma metodologia (*paradigma*) de programação adequada ao desenvolvimento de sistemas de grande porte, provendo modularidade e reusabilidade.

Na programação orientada a objetos, um programa é visto como um conjunto de objetos que trabalham juntos para realizar uma tarefa.



Programação Orientada a Objetos

- Na visão da orientação a objetos, o mundo é composto por diversos objetos que possuem um conjunto de características (atributos) e um comportamento bem definido.
- Todo objeto possui as seguintes características:
  - **Estado**: conjunto de propriedades de um objeto (valores dos atributos).
  - **Comportamento**: conjunto de ações possíveis sobre o objeto (métodos da classe que o objeto instancia).
  - **Unicidade**: todo objeto é único (possui um endereço de memória diferente).

Classe

- Modelo ou especificação para um conjunto de objetos
- É uma descrição genérica dos objetos individuais pertencentes a um dado conjunto
- A partir de uma classe é possível criar quantos objetos forem desejados.

Uma classe define as características e o comportamento de um conjunto de objetos.

- Exemplo: classe **Pessoa**

Objeto

- É uma **instância** de uma classe.
- Ele é capaz de armazenar estados por meio de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos.
- João, José e Maria, por exemplo, podem ser considerados objetos de uma classe Pessoa que tem como atributos **nome**, **CPF**, **data de nascimento**, entre outros, e métodos como **calcularIdade**.

Classe

Sintaxe:

<qualificador> **class** <nome_da_classe>{

<declaração dos atributos>

<declaração dos métodos>

}

Como funciona?

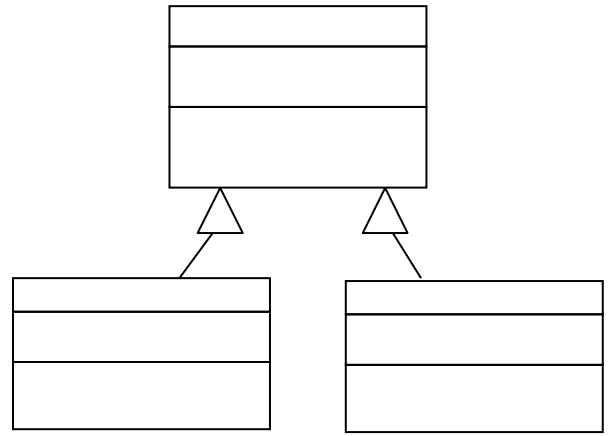
- <**qualificador**>: O qualificador de acesso determinará a visibilidade da classe. Pode ser **public** (classe pública) ou **private** (classe privada). Classes privadas só poderão ser visualizadas dentro de seu próprio pacote enquanto as públicas serão acessíveis por qualquer classe de qualquer pacote. Se o qualificador for omitido, a classe será privada por padrão.
- <**nome_da_classe**>: nome que identifica a classe. Há um padrão entre os programadores de sempre iniciarem os nomes de classes com letras maiúsculas. Mas, apesar de ser uma boa prática, seguir esse padrão não é uma obrigação.

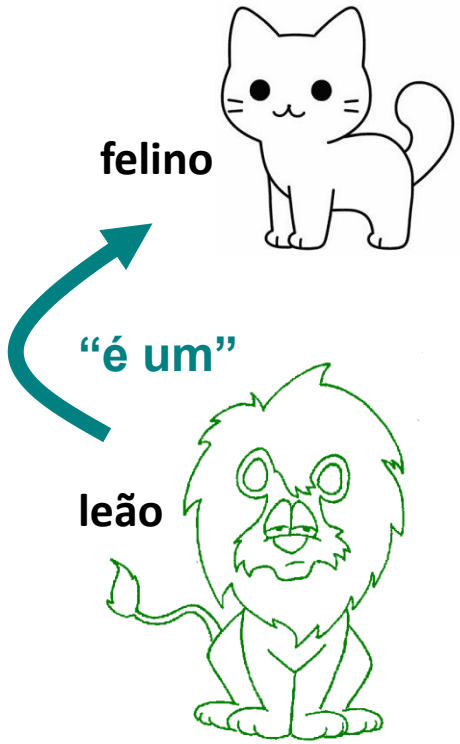
Herança

A *herança* (generalização/especialização) é um relacionamento *entre classes*, que existe quando há a relação "**é um**" entre duas ou mais classes.

A *herança* é uma forma de **reutilizar código** absorvendo membros de uma classe já existente: economia de tempo no desenvolvimento!

Possível representação:





Herança

Permite especificar atributos e operações herdáveis pelos descendentes (generalização)

- as características mais gerais ficam nas classes mais gerais

Criação de novas classes com base na alteração de classes existentes (especialização)

- somente as diferenças precisam ser descritas
- uma forma de programar incrementalmente

Herança

Possibilita que atributos e operações de uma classe sejam compartilhados:

- na classe mais geral no qual se aplicam
(superclasse ou classe base)

- pelas classes descendentes da classe
(subclasse ou classe derivada)

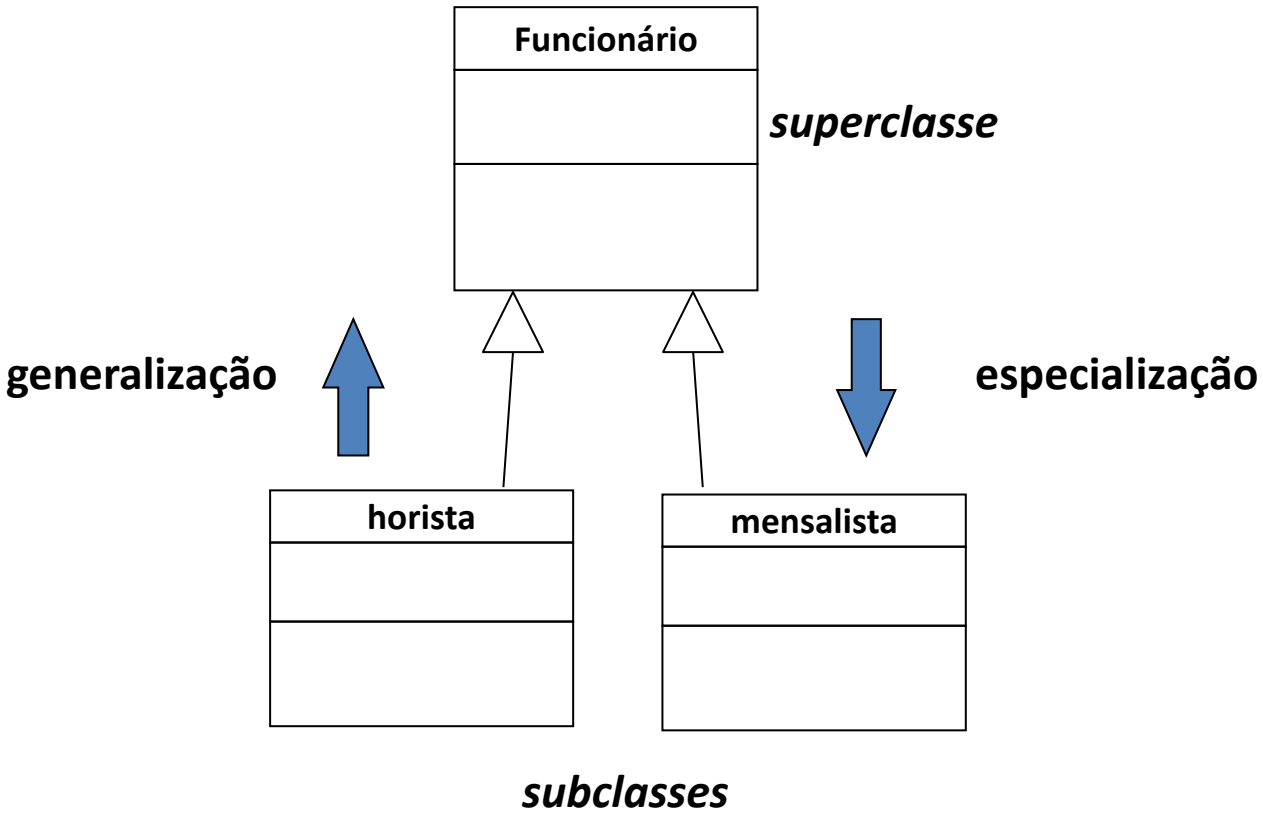
- pelas instâncias da classe
(forma de reuso)

Quadro 2.1 Exemplos de herança.

Superclasse	Subclasses
Aluno	AlunoDeGraducao, AlunoDePosGraduacao
Forma	Circulo, Triangulo, Retangulo, Esfera, Cubo
Financiamento	FinanciamentoDeCarro, FinanciamentoDeCasa
Empregado	CorpoDocente, Funcionario
ContaBancaria	ContaCorrente, ContaPoupanca

Fonte: adaptado de Deitel e Deitel (2010, p. 280).

Representação Gráfica de Herança



Herança Múltipla

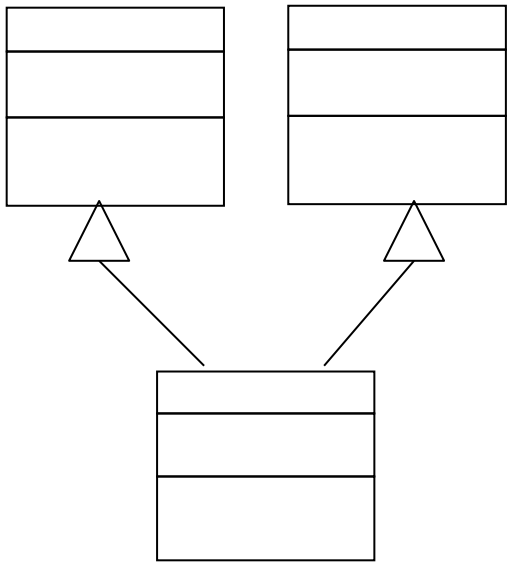
Uma mesma classe pode vir a herdar ao mesmo tempo de mais de uma classe.

Observação:
- reduz a compreensão da hierarquia de classes
- exige tratamento de ambiguidades
- tem uso controverso na comunidade de Orientação a Objetos
- C++ e Python são linguagens que suportam herança múltipla!

Atenção: **Java não suporta** herança múltipla para evitar problemas de ambiguidade! Veja este exemplo.

Herança Múltipla

Possível forma de representação:



Herança (Resumindo)

- Herança é um relacionamento (do tipo **é um**) entre duas classes;

- Uma destas classes será chamada de “classe base” (superclasse, classe pai, classe mãe) e a outra será chamada de “classe derivada” (subclasse, classe filha, classe herdeira);

- O relacionamento de herança permite representar generalização/especificação: uma classe base mais geral e uma classe derivada que seria mais específica ou particular;

Herança

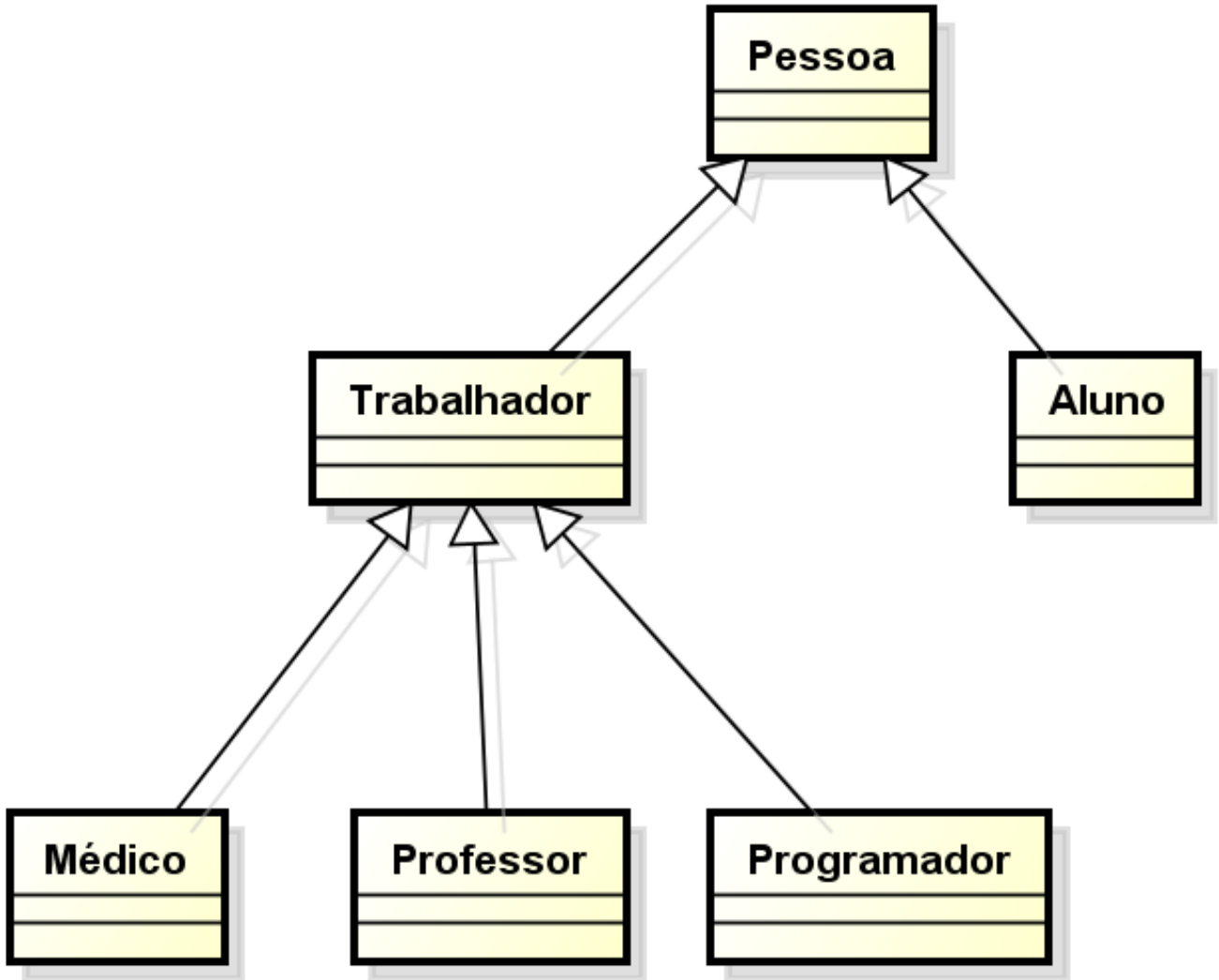
Declarando Herança - Java

```
class ClasseBase {
.....
}


class ClasseDerivada extends ClasseBase {
......
}
```

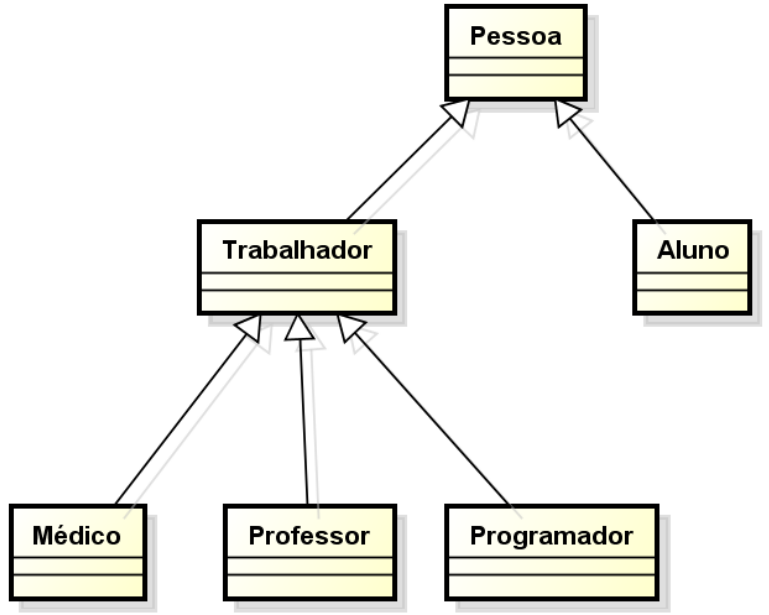
Herança

Exemplo



Herança

Exemplo




```
class Pessoa {
.....
}

class Trabalhador extends Pessoa {
.....
}

class Aluno extends Pessoa {
.....
}

class Medico extends Trabalhador {
.....
}
```

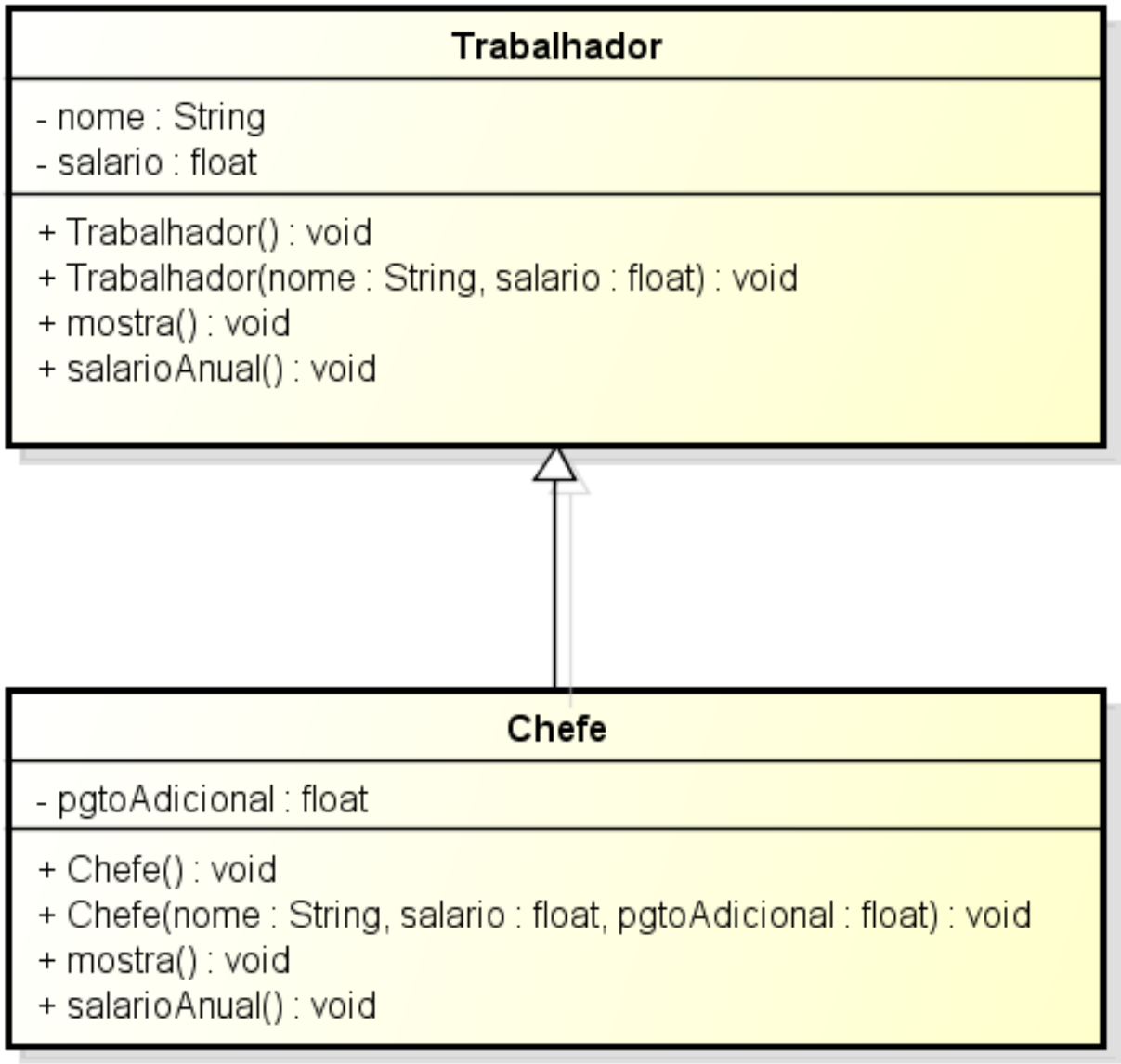
```
class Professor extends Trabalhador {
.....
}

class Programador extends Trabalhador
{
.....
}

Programador pr = new Programador();
```

Herança

Exemplo



Herança

Exemplo – Arquivo TrabChefe.java

```
import java.util.*;

class Trabalhador {
      private String nome;
      private float salario;

      public Trabalhador(){
            nome = "Sem nome";
            salario = 0;
      }
```

```
public Trabalhador(String nome, float salario){
      setNome(nome);
      setSalario(salario);
}

public String getNome(){
      return nome;
}

public void setNome(String nome){
      this.nome = nome;
}

public float getSalario(){
      return salario;
}

public void setSalario(float salario){
      this.salario = salario;
      if (this.salario<0) this.salario = 0;
}
```

(Construtor)

(gets e sets)

Herança

Exemplo – Arquivo TrabChefe.java

Herança

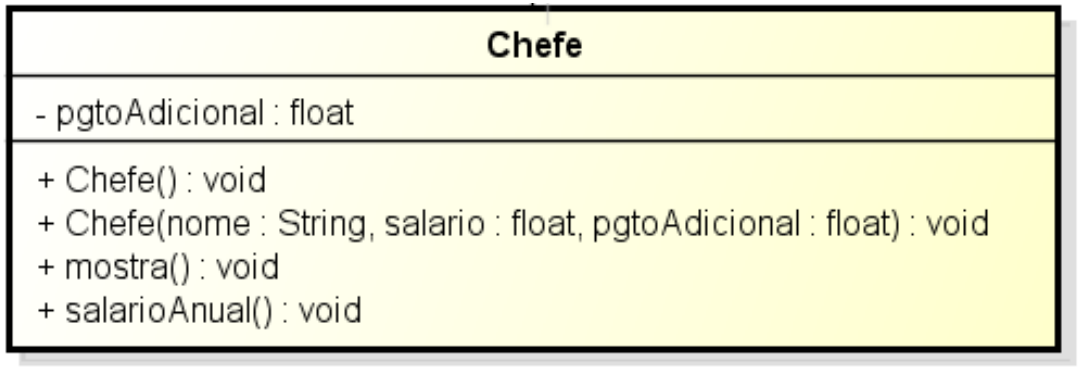
Exemplo – Arquivo TrabChefe.java

```
        public void mostra() //ou public String toString()
        {
                System.out.print(nome);
                System.out.print(", Salario mensal: R$" + salario);
                System.out.println(", Salario anual: R$" +
                salarioAnual());
        }

        public float salarioAnual()
        {
                return(salario * 13 + salario/3);
        }
} // fim da classe Trabalhador
```

Herança

Exemplo – Arquivo TrabChefe.java




```
class Chefe extends Trabalhador {
      private float pgtoAdicional;
      public Chefe(){
            //executamos o construtor da classe base:
            super("Sem nome", 0.0f);
            pgtoAdicional = 0.0f;
      }
      public Chefe(String nome, float salario, float pgtoAdicional){
            //executamos o construtor da classe base:
            super(nome, salario);
            setPgtoAdicional(pgtoAdicional); //outra forma usando set
      }
```

Herança

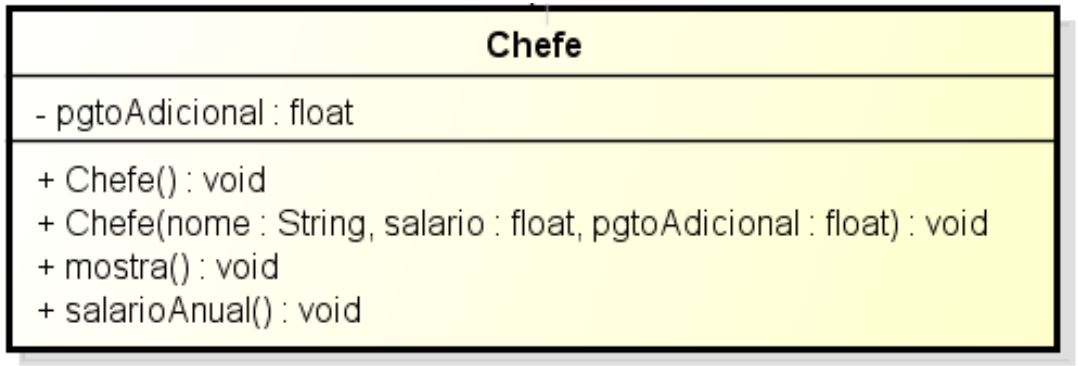
Exemplo – Arquivo TrabChefe.java

```
public float getPgtoAdicional(){
      return pgtoAdicional;
}

public void setPgtoAdicional(float pgtoAdicional){
      this.pgtoAdicional = pgtoAdicional;
}
```

Herança

Exemplo – Arquivo TrabChefe.java




```
public float salarioAnual() {
      //veja que foi utilizado o método herdado getSalario()
      return (getSalario() * 13 + getSalario()/3 + getPgtoAdicional() *
12);
      // Também poderia escrever o seguinte código
      // return super.salarioAnual() + getPgtoAdicional() * 12;
}

public void mostra() { //ou public String toString()
      // Utilizado o método mostra() da classe base Trabalhador
      super.mostra();
      System.out.println(" --> Pgto. adicional mensal como chefe: R$" +
getPgtoAdicional () ); }
} // fim da classe Chefe
```

Herança

Exemplo – Arquivo TrabChefe.java

```
public class TrabChefe { //classe principal para demonstração
	public static void main (String[] args) {
		System.out.println(new Date());
		System.out.println("\n");
		Trabalhador trabA = new Trabalhador();
		//agora será executado o método mostra()da classe Trabalhador:
		trabA.mostra();
		Trabalhador trabB = new Trabalhador("Joao Silva", 1500.75f);
		trabB.mostra();
		Trabalhador trabC = new Trabalhador("Ana Souza", 900f);
		trabC.mostra();
		Chefe chefe = new Chefe("Julio Moreira", 1000f, 100f);
		//agora, por se tratar de um objeto da classe Chefe, será
		//executado o método mostra()da classe Chefe:
		chefe.mostra();
	}
}
```

Herança

Exemplo – Comentário

- A declaração **class Chefe extends Trabalhador** estabelece um relacionamento de herança entre as classes Chefe e Trabalhador, sendo que Trabalhador é a classe base e Chefe é a classe derivada;

- Quanto às três características básicas da herança:

  ✓ A classe Chefe herda os atributos passivos (nome e salario) e métodos (construtores, getNome, getSalario, mostra etc.) da classe base Trabalhador; mas só poderá acessar os atributos/métodos públicos ou protegidos da classe Trabalhador, nunca poderá acessar diretamente os privados*;*

Herança

Exemplo – Comentário

✓A classe Chefe adiciona novos dados (pgtoAdicional) e métodos (dois construtores, getPgtoAdicional e setPgtoAdicional);

✓A classe Chefe **redefine** os métodos mostra() e salarioAnual() que já existiam na classe base Trabalhador.

- A classe derivada poderá executar métodos idênticos da classe base, utilizando **super.nomeDoMetodo( ); no exemplo anterior utilizamos super.mostra();**

- Para executar algum método construtor da classe base, utilizaríamos: **super( );** **ou super(listaDeParâmetros);**

Herança

Exemplo – Comentário

- A classe derivada poderia executar outros métodos da classe base, sem utilizar a palavra super, sempre que esse nome de método não exista na classe derivada; este é o caso de getSalario() no exemplo anterior;

- Observe que, por exemplo, trabC.mostra(); executará o método mostra da classe Trabalhador, mas no caso de chefe.mostra(); será executado o método mostra() da classe Chefe.

- A mesma coisa acontece com o método salarioAnual(), pois será executado o método de uma classe ou outra dependendo da instância do objeto que chama o método, porém esse método da classe Chefe não chama o método da classe pai.

PRATICANDO

Exercício

Crie um projeto Java, em alguma IDE disponível, e implemente uma classe **Animal** que obedeça à seguinte descrição:

- possua os atributos nome (String), comprimento (float), número de patas (int), cor (String), ambiente (String) e velocidade média (float)
- crie um método construtor que receba por parâmetro os valores iniciais de cada um dos atributos e atribua-os aos seus respectivos atributos.
- crie os métodos get e set para cada um dos atributos.
- crie um método dados, sem parâmetro e do tipo void, que, quando chamado, imprime na tela uma espécie de relatório informando os dados do animal.

Atenção: cada classe do projeto deve estar em um arquivo separado.

Crie uma classe **Peixe** que herde da classe Animal e obedeça à seguinte descrição:

- possua um atributo caracteristica(String)
- crie um método construtor que receba por parâmetro os valores iniciais de cada um dos atributos (incluindo os atributos da classe Animal) e atribua-os aos seus respectivos atributos.
- crie ainda os métodos get e set para o atributo caracteristica.
- crie um método dadosPeixe sem parâmetro e do tipo void, que, quando chamado, imprime na tela uma espécie de relatório informando os dados do peixe (incluindo os dados do Animal e mais a característica)

Crie uma classe **Mamifero** que herde da classe Animal e obedeça à seguinte descrição:

- possua um atributo alimento(String)
- crie um método construtor que receba por parâmetro os valores iniciais de cada um dos atributos (incluindo os atributos da classe Animal) e atribua-os aos seus respectivos atributos.
- crie ainda os métodos get e set para o atributo alimento.
- crie um método dadosMamifero sem parâmetro e do tipo void, que, quando chamado, imprime na tela uma espécie de relatório informando os dados do mamifero (incluindo os dados do Animal e mais o alimento).

Crie uma classe **TestarAnimais** que possua um método main para testar as classes criadas.

a) Crie um objeto camelo do tipo **Mamífero** e atribua os seguintes valores para seus atributos:

- Nome: Camelo
- Comprimento: 150 cm
- Patas: 4
- Cor: Amarelo
- Ambiente: Terra
- Velocidade: 2.0 m/s
- Alimento: carne

b) Crie um objeto **tubarao** do tipo Peixe e atribua os seguintes valores para seus atributos

- Nome: Tubarão
- Comprimento: 300 cm
- Patas: 0
- Cor: Cinzento
- Ambiente: Mar
- Velocidade: 1.5 m/s
- Característica: Barbatanas e cauda

Durante o teste do seu programa, chame os métodos para imprimir os dados de cada um dos objetos criados.

Anotações


Universidade Presbiteriana
Mackenzie
150 anos
1870 - 2020


Faculdade de
Computação e Informática
FCI
50
ANOS