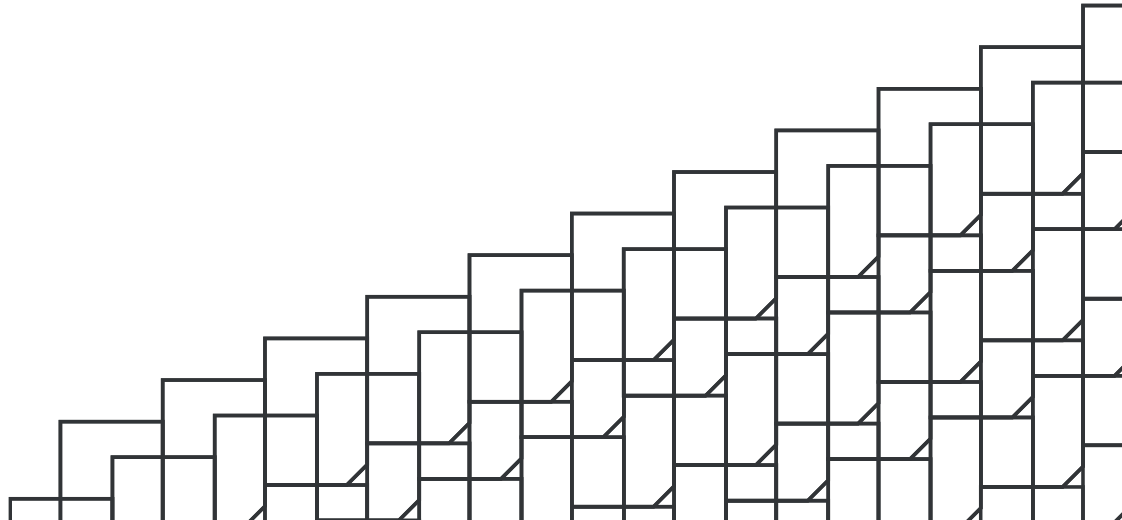
Herança

Estrutura de Dados II

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

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

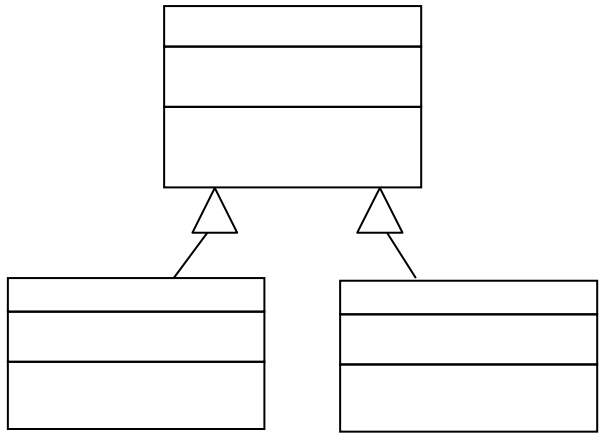
Prof. Dr. Ivan Carlos Alcântara de Oliveira
https://orcid.org/0000-0002-6020-7535

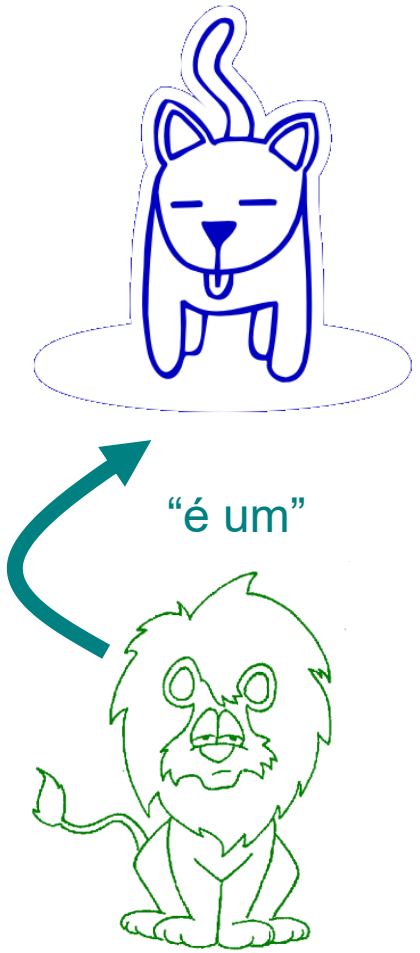


Herança

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

Possível representação:





Herança

A herança define uma hierarquia entre classes, na qual uma subclasse herda de uma ou mais superclasses.

(Baseado em um desenho clássico do livro do Grady Booch)

Herança

Permite especificar informações (atributos) e operações (métodos), herdáveis pelos descendentes (generalização)

- Características mais gerais 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

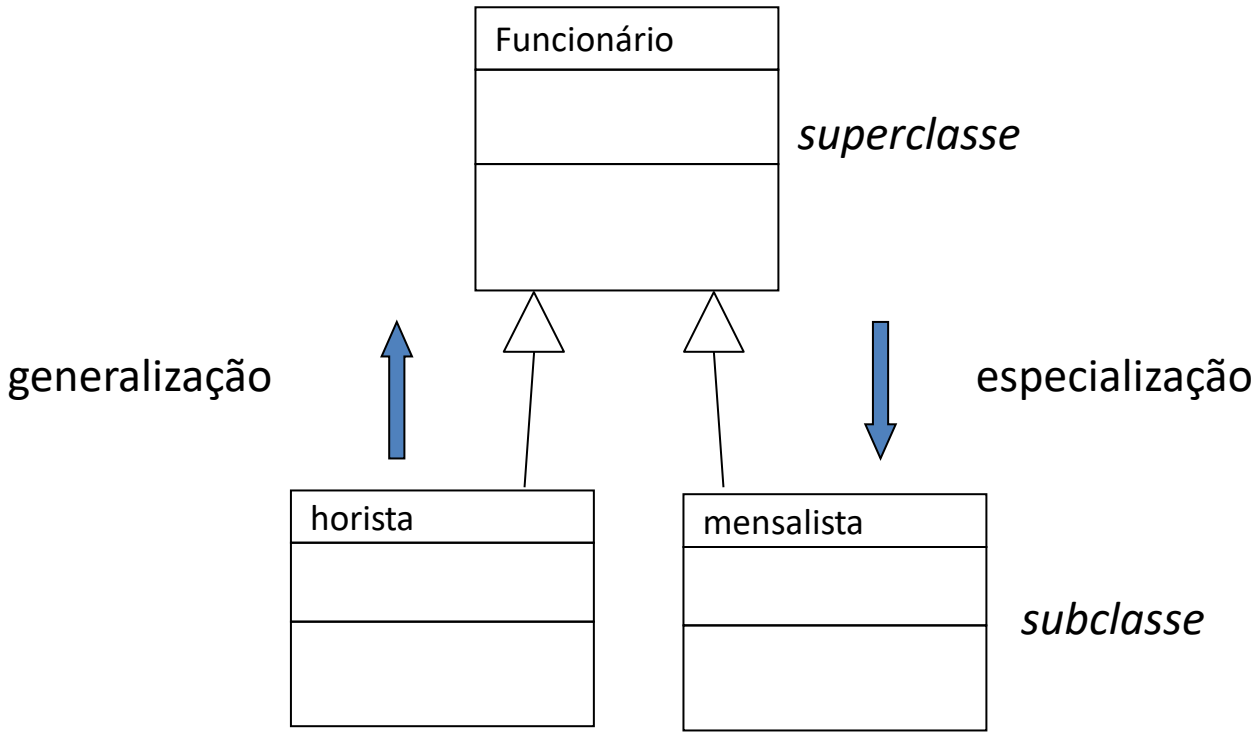
Possibilita que atributos e operações de uma classe sejam compartilhados:

- Na classe mais geral no qual se aplicam (superclasse)

- Pelas classes descendentes da classe (subclasse)

- Pelas instâncias da classe (forma de reuso)

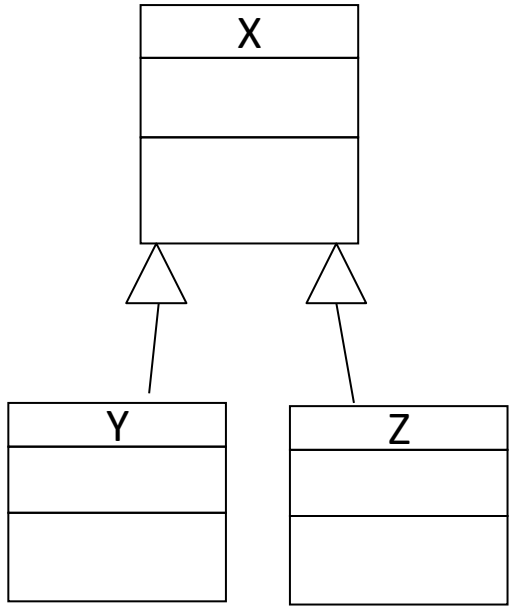
Herança


Funcionário
superclasse
generalização
especialização
horista
mensalista
subclasse

Herança

Supondo o modelo ao lado:

- As classes Y e Z são subclasses de X
- Y e Z herdam de X
- Os atributos e as operações de X são compartilhados por Y e Z, ou seja, torna-se parte de Y e Z
- Y e Z tem mais atributos e/ou operações que X



compartilhamento de código
menos redundância
programação incremental

Herança múltipla

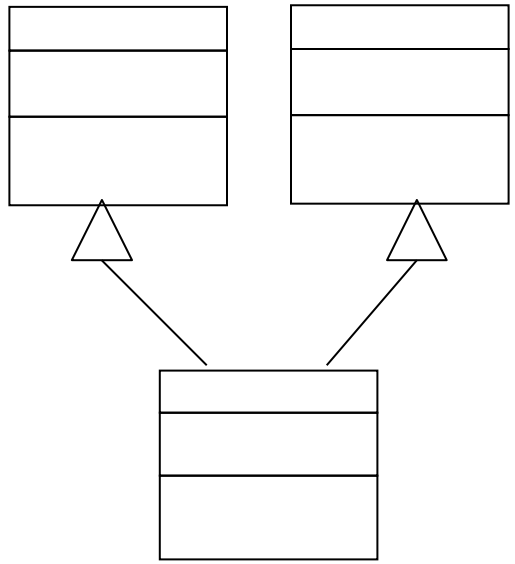
Uma mesma classe pode vir a herdar ao mesmo tempo mais de uma classe.

Observação:
- Reduz a compreensão da hierarquia de classes
- Exige tratamento de ambiguidades
- Tem uso controverso na comunidade de Orientação a Objetos

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

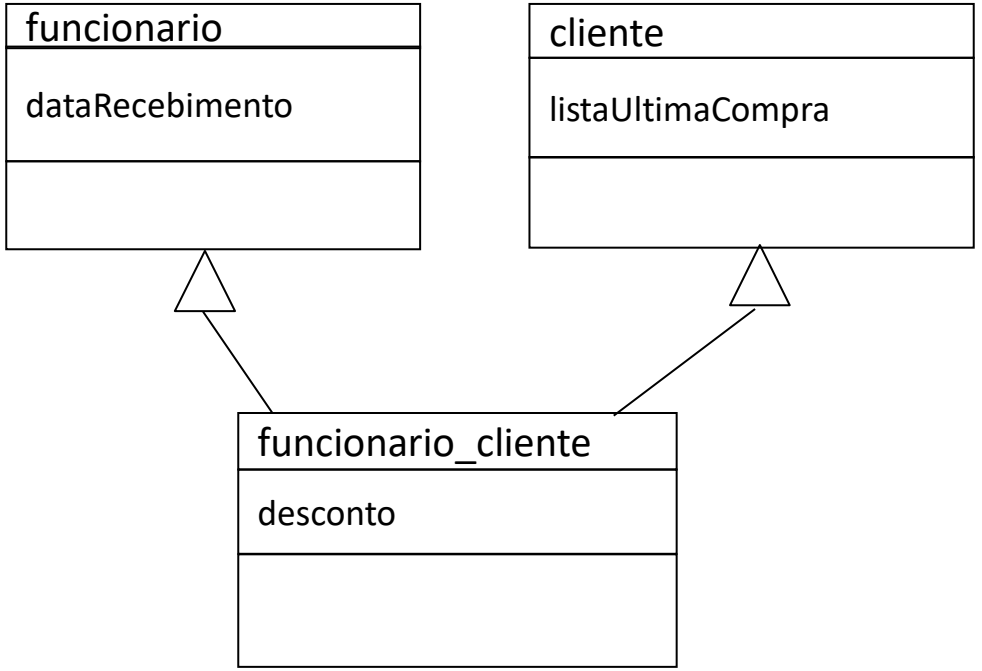
Herança múltipla

Possível forma de representação (UML):



Herança múltipla

Exemplo



Herança

- Herança é um relacionamento 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;

- Caso especial: Herança de interface ou por implementação é outro caso de relacionamento de herança;

Herança

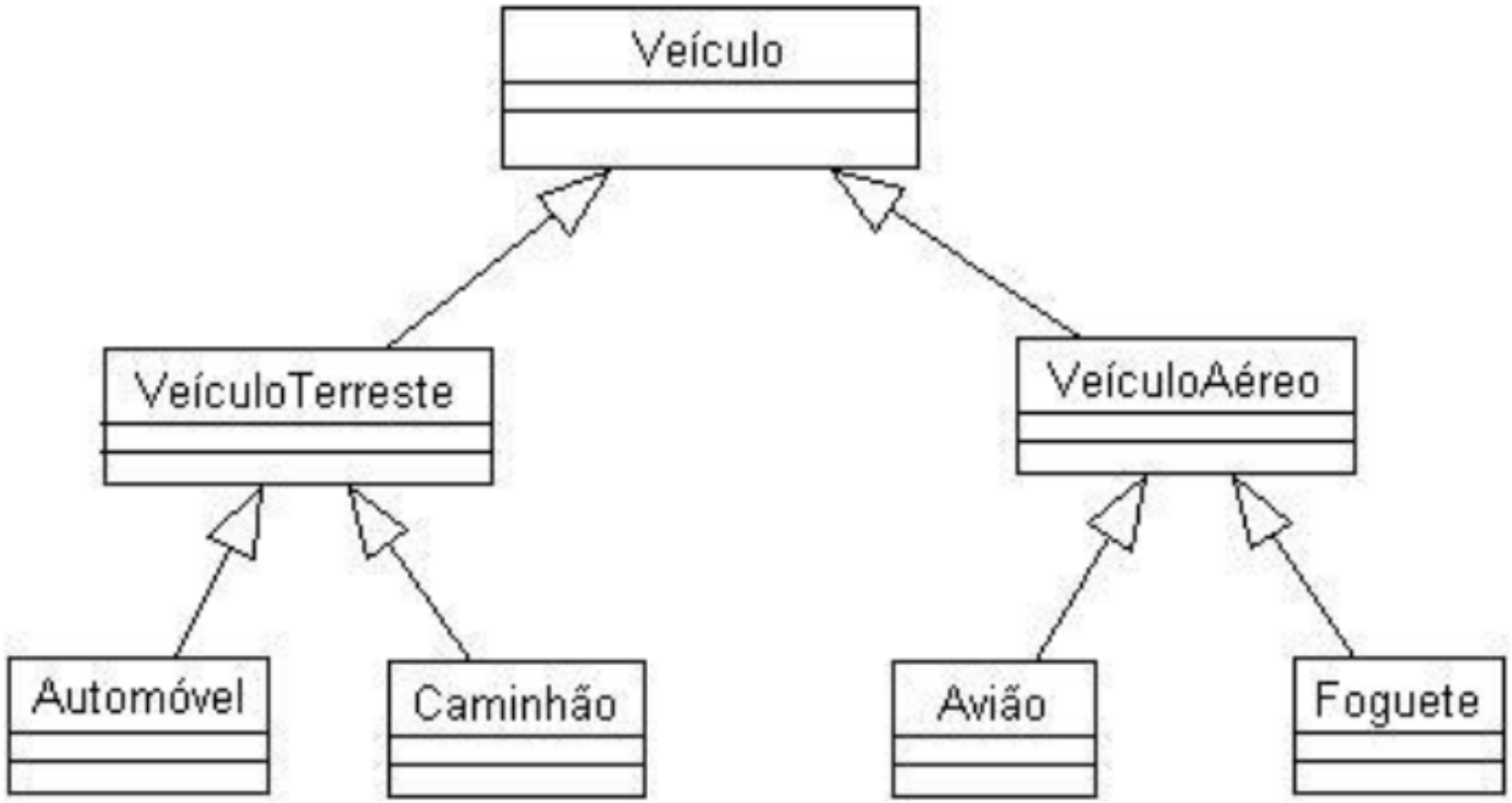
- Podemos identificar relacionamento de herança entre duas classes quando é possível afirmar que “*ClasseDerivada é um tipo de ClasseBase”* ou simplesmente *“ClasseDerivada é um ClasseBase”*;

- Em UML é possível usar um *diagrama de classes* para representar relacionamentos de herança entre classes (e outros tipos de relacionamentos que não sejam de herança, como *composição e agregação*; veremos futuramente);

- Num relacionamento de herança, as classes mais gerais ou abstratas (veremos futuramente) aparecem perto da raiz (parte superior) e as classes mais particulares ou específicas aparecerão nas folhas (parte inferior).

Herança

Representação UML



Herança

Três características fundamentais

- Uma *classe derivada* ***herdará as características da classe base (atributos/métodos);***

- A *classe derivada* ***poderá acrescentar (declarar) novos atributos e métodos que não existiam na classe base;***

- A *classe derivada* ***poderá redefinir (sobrescrever) métodos que foram declarados na classe base.***

Herança

Importância

- Aproveitar classes que já foram desenvolvidas, herdando seus atributos e métodos públicos (*public*) ou protegidos (*protected*) permitindo eficiência no desenvolvimento.

- Uso de pacotes de classes fornecidos por empresas e desenvolvedores terceiros (por exemplo: Microsoft, Sun Microsystems, etc.).

- A depuração de erros e manutenção é mais simples* usando classes e herança.

- A herança facilita a expansão de um sistema.

** Essa é uma afirmação subjetiva...*

Herança

Declarando herança em Java

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

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

Herança

Exemplo 1 - Java: herdando métodos públicos da classe JFrame

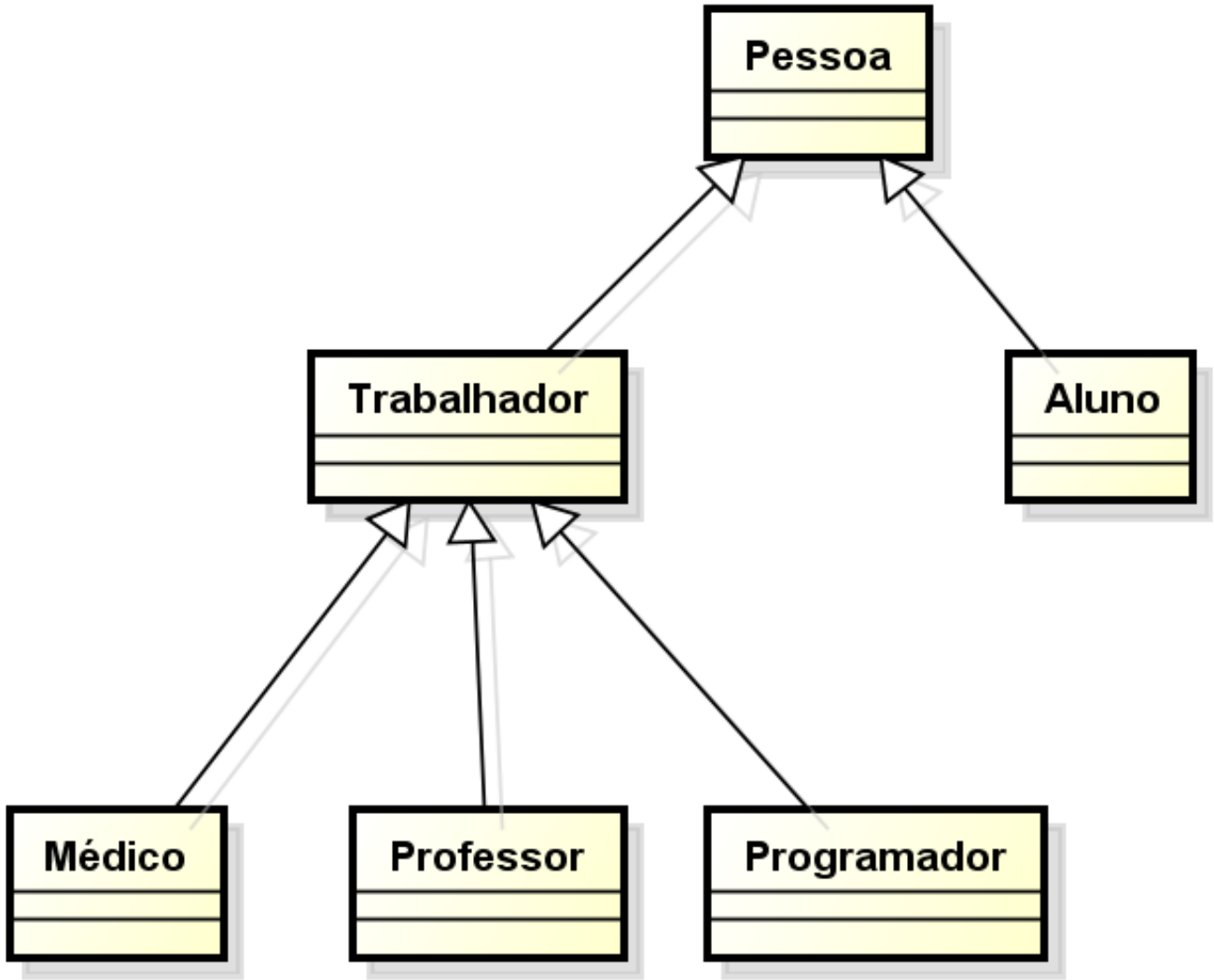
```
import javax.swing.*;

public class ExHerancaImportancia extends JFrame {
  public ExHerancaImportancia() { //construtor da classe }

  public static void main(String args[]) {
    ExHerancaImportancia janela = new ExHerancaImportancia();
    janela.setSize(430, 380);
    janela.setTitle("Uma janela com poucas linhas... Herança!");
    janela.setVisible(true);
  }
}
```

Herança

Exemplo 2 - UML



Herança

Exemplo 2 - Java

```
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

Encapsulamento e herança

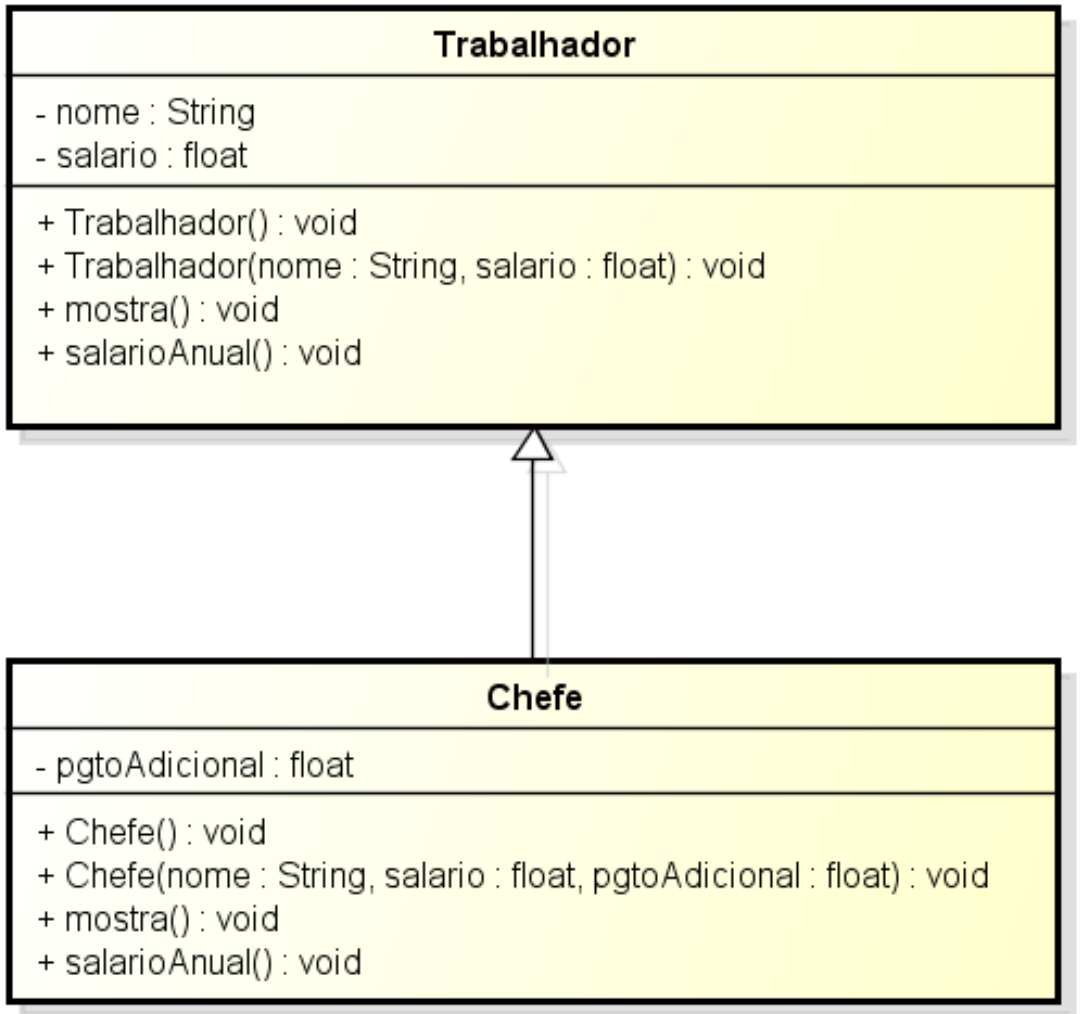
- Uma classe derivada pode acessar os atributos (dados membros) e métodos *públicos* (**public**) da classe base.

- Uma classe derivada pode acessar os atributos e métodos *protegidos* (**protected**) da classe base.

- Uma classe derivada **não** pode acessar os atributos e métodos *privados* (**private**) da classe base.

Herança

Exemplo 3 - Diagrama UML



Herança

Exemplo 3 - Java (arquivo Trabalhador.java)

```
import java.util.*;

public class Trabalhador {

  private String nome;
  private float salario;

  public Trabalhador() {
    nome = "Sem nome";
    salario = 0.0f;
  }


                                        // Continua no próximo slide...
```

Herança

Exemplo 3 - Java (arquivo Trabalhador.java)

```
// Continuação do slide anterior...

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

  public String getNome() {
    return nome;
  }

  public void setNome(String nome) {
    this.nome = nome;
  }                                          // Continua no próximo slide...
```

Herança

Exemplo 3 - Java (arquivo Trabalhador.java)

```
// Continuação do slide anterior...

  public float getSalario() {
    return salario;
  }

  public void setSalario(float salario) {
    this.salario = salario;
    if (this.salario < 0.0f) {
      this.salario = 0.0f;
    }
  }

                                        // Continua no próximo slide...
```

Herança

Exemplo 3 - Java (arquivo Trabalhador.java)

```
// Continuação do slide anterior...

  public void mostra() { // Ou @Override 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 * 13.0f + salario / 3.0f;
  }

} // fim da classe Trabalhador
```

Herança

Exemplo 3 - Java (arquivo Chefe.java)

```
public class Chefe extends Trabalhador {

  private float pgtoAdicional;

  public Chefe() {
    super();
    pgtoAdicional = 0.0f;
  }

  public Chefe(String nome, float salario, float pgtoAdicional) {
    // Executamos o construtor da classe base:
    super(nome, salario);
    setPgtoAdicional(pgtoAdicional);
  }                                          // Continua no próximo slide...
```

Herança

Exemplo 3 - Java (arquivo Chefe.java)

```
// Continuação do slide anterior...

  public float getPgtoAdicional() {
    return pgtoAdicional;
  }

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




                                        // Continua no próximo slide...
```

Herança

Exemplo 3 - Java (arquivo Chefe.java)

```
// Continuação do slide anterior...

  public float salarioAnual() {
    // Veja que foi usado o método herdado getSalario():
    return getSalario() * 13.0f + getSalario() / 3.0f
           + getPgtoAdicional() * 12.0f;

    // Também poderia escrever o seguinte código:
    // return super.salarioAnual() + getPgtoAdicional() * 12.0f;
  }



                                          // Continua no próximo slide...
```

Herança

Exemplo 3 - Java (arquivo Chefe.java)

```
// Continuação do slide anterior...

  public void mostra() {// Ou @Override public String toString() { ... }
    // Usado o método mostra() da classe base Trabalhador
    super.mostra();
    System.out.println(" --> Pgto. adicional mensal como chefe: R$"
                       + getPgtoAdicional());
  }

} // fim da classe Chefe



                                                  // Continua no próximo slide...
```

Herança

Exemplo 3 - Java (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();
                                          // Continua no próximo slide...
```

Herança

Exemplo 3 - Java (arquivo TrabChefe.java)

```
// Continuação do slide anterior...

    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();
  }
}
```

// Continua no próximo slide...

Herança

Exemplo 3 - Comentários

- 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.

Herança

Exemplo 3 - Comentários

- Quanto às três características básicas da herança:

1) A classe Chefe herda os atributos (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;

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

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

Herança

Exemplo 3 - Comentários

- A classe derivada poderá executar métodos idênticos da classe base, usando super.nomeDoMetodo();
    - No exemplo anterior usamos super.mostra();

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

Herança

Exemplo 3 - Comentários

- A classe derivada poderia executar outros métodos da classe base sem usar a palavra super, desde que o nome do método não exista na classe derivada. Este é o caso do 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 base.

Referências bibliográficas

1. LARMAN, C. Utilizando UML e Padrões: Uma Introdução a Analise e ao Projeto Orientados a Objetos. 2. ed. Porto Alegre: Bookman, 2005.
2. YOURDON, E. Analise e Projeto Orientados a Objetos: Estudos de Casos. São Paulo: Makron Books do Brasil, 1999.
3. WAZLAWICK, R. S. Analise e Projeto de Sistemas de Informação Orientados a Objetos. Rio de Janeiro: Campus, 2004.
4. Silveira, I. F., Notas de Aula da disciplina Análise e Programação Orientada a Objetos, 2007.
5. Souza, C., Modelagem de Casos de Uso. Notas de aula, 2005. Disponível em http://www2.ufpa.br/cdesouza/teaching.html.
6. Kasperavicius, L., Disciplina de Programação Orientada a Objetos, Curso de Tecnologia em Jogos Digitais, Notas de Aula, 2006.
7. Kasperavicius, L., Disciplina de Engenharia de Software, Curso de Bacharelado em Ciência da Computação, Notas de Aula, 2006.
8. Silva, L, Material da Disciplina Paradigmas de Linguagens de Programação, Disciplina Online, Ciência da Computação, 2010.
9. Ledon, M. F. P. Material da Disciplina Linguagens de Programação I, Ciência da Computação, Notas Aula, 2012.
10. Oliveira, I. C. A. Material da Disciplina Avaliação de Métodos de Análise e Projeto Orientados a Objetos, Curso de Pós-Graduação USP, Notas de Aula, 2009.
11. Site do prof. Newton. Disponível em: https://sites.google.com/site/profnewtonjava/exercicioheranca.


Universidade Presbiteriana
Mackenzie
150 anos
1870 - 2020


Faculdade de
Computação e Informática
FCI
50
ANOS