# Herança

Prof. Valdir Pedrinho de Tomin Junior, Eng.

Universidade Federal de Santa Catarina
Centro Tecnológico de Joinville
EMB5631 – Programação III

e-mail: valdir.pedrinho@ufsc.br

25 de setembro de 2018

# Introdução

## O que é herança?

- É uma forma de reutilização de software em que o programador cria uma classe que absorve dados e comportamentos de uma classe existente e os aprimora com novas capacidades;
- O programador deve designar que a nova classe herda os membros de uma classe existente.

## Definições

- **Classe Básica**: classe básica, classe pai ou superclasse é classe a existente;
- **Classe Derivada**: classe derivada, classe filha ou subclasse é classe que herdará os membros.

# Introdução

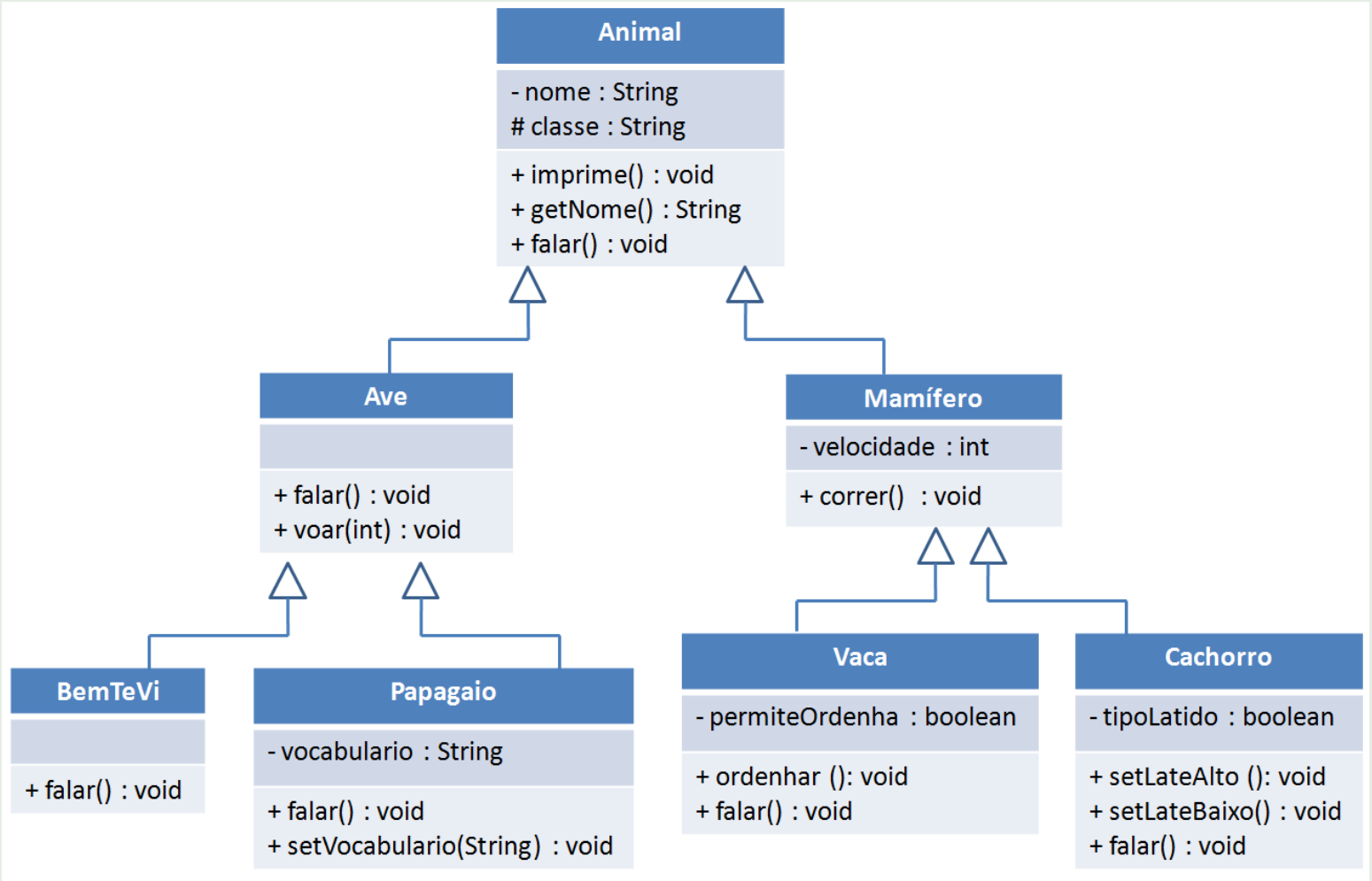
## Características

- Uma classe derivada representa um grupo mais especializado de objetos;
- Em geral, uma classe derivada contém comportamentos herdados de sua classe básica mais comportamentos adicionais;
- Uma classe derivada também pode personalizar comportamentos herdados da classe básica;
- É sinônimo de generalização, isto é, a classe base é a generalização de uma classe derivada.

## Os objetos são de que classe?

Cada objeto de classe derivada é, também, um objeto da sua classe básica

## Generalização – Diagrama UML



# Introdução

## Características

- Uma classe básica pode ter muitas classes derivadas;
- O conjunto de objetos representado por uma classe básica é, em geral, maior que o conjunto de objetos representados por qualquer de suas classes derivadas;

  **Exemplo**

  A classe básica Veiculo representa todos os veículos, incluindo carros, caminhões, barcos, aviões, bicicletas e assim por diante. A classe derivada Carro representa um subconjunto menor e mais específico de todos os veículos.

- Os relacionamentos de herança formam estruturas hierárquicas do tipo árvore. As classes derivadas são afiliadas das classes básicas.

# Implementação

## Sintaxe

```
NomeDaClasse: especificador_acesso ClasseBase
{
 // Implementação da classe
};
```

## Exemplo

Classes **Forma** e **Retangulo**

# Implementação

## Exercício – Implemente as classes abaixo

**EmpComSalarioBase** herda os membros de **EmpregadoComissionado**.

```
class EmpregadoComissionado
{
public:
    // Construtor, métodos
       set e get
    double rendimentos()
       const;
    void imprime() const;
protected:
    string nome;
    string sobrenome;
    double vendas;
    double taxaComissao;
}
```

```
class EmpComSalarioBase
{
public:
    // Construtor, métodos
       set e get
    double rendimentos()
       const;
    void imprime() const;
private:
    double salarioBase;
};
```

# Relações de Herança

## Controle de Acesso a Membros

| Escopo | Membro **public** | Membro **protected** | Membro **private** |
|---|---|---|---|
| No escopo da classe | permitido | permitido | permitido |
| Na classe derivada | permitido | permitido | negado |
| No escopo global | permitido | negado | negado |
| Função/Classe **friend** | permitido | permitido | permitido |

# Relações de Herança

## Tipos de Herança

| Membros da Classe Base | Herança pública | Herança protegida | Herança privada |
|---|---|---|---|
| **public** | Continuam **public** | Passam a ser **protected** | Passam a ser **private** |
| **protected** | Continuam **protected** | Continuam **protected** | Passam a ser **private** |
| **private** | Não podem ser acessados diretamente | Não podem ser acessados diretamente | Não podem ser acessados diretamente |

# Relações de Herança

## Resumo

| Especificador de acesso de membro de classe básica | Tipo de herança | | |
|---|---|---|---|
| | Herança `public` | Herança `protected` | Herança `private` |
| `public` | `public` na classe derivada.<br><br>Pode ser acessada diretamente por funções-membro, funções `friend` e funções não-membro. | `protected` na classe derivada.<br><br>Pode ser acessada diretamente por funções-membro e funções `friend`. | `private` na classe derivada.<br><br>Pode ser acessada diretamente por funções-membro e funções `friend`. |
| `protected` | `protected` na classe derivada.<br><br>Pode ser acessada diretamente por funções-membro e funções `friend`. | `protected` na classe derivada.<br><br>Pode ser acessada diretamente por funções-membro e funções `friend`. | `private` na classe derivada.<br><br>Pode ser acessada diretamente por funções-membro e funções `friend`. |
| `private` | Oculta na classe derivada.<br><br>Pode ser acessada por funções-membro e funções `friend` por meio das funções-membro `public` ou `protected` da classe básica. | Oculta na classe derivada.<br><br>Pode ser acessada por funções-membro e funções `friend` por meio das funções-membro `public` ou `protected` da classe básica. | Oculta na classe derivada.<br><br>Pode ser acessada por funções-membro e funções `friend` por meio das funções-membro `public` ou `protected` da classe básica. |

# Construtores e Destrutores

## Ordem de Chamada

- Instanciar um objeto de classe derivada inicia uma cadeia de chamadas de construtor em que o construtor de classe derivada, antes de realizar suas próprias tarefas, invoca o construtor da sua classe básica direta;
- O último construtor chamado nessa cadeia é o construtor da classe na base da hierarquia, cujo corpo na realidade termina de executar primeiro;
- Os destrutores são executados em ordem inversa.

## Sintaxe – Chamadas Explícitas

```
NomeDaClasse::NomeDaClasse(lista de parâmetros) :
    ClasseBase(parâmetros), atributo1(valor), ...,
    atributoN(valor){}
```

# Detalhes de Implementação

## Não são herdáveis

- Funções/Classes **friend**;
- Sobrecarga de operadores.

## Métodos com mesmo identificador

- Quando um método da classe derivada possui o mesmo identificador que o da classe base, sempre é chamado o método da classe derivada;
- O método da classe base só poderá ser chamado de forma explícita, através do operador de resolução de escopo (::).

## Exemplo

Classe **Paralelepipedo** que herda de **Retangulo**

# Implementação

## Exercício – Crie a classe **Imovel**

**Imovel** deve possuir um endereço e um preço.

- Crie uma classe **NovoImovel**, que herda **Imovel** e possui um adicional no preço. Crie métodos de acesso e impressão deste valor adicional;
- Crie uma classe **VelhoImovel**, que herda **Imovel** e possui um desconto no preço. Crie métodos de acesso e impressão para este desconto.

## Exercício – Crie uma classe chamada **Ingresso**

**Ingresso** deve possuir um valor e um método **imprimeValor**.

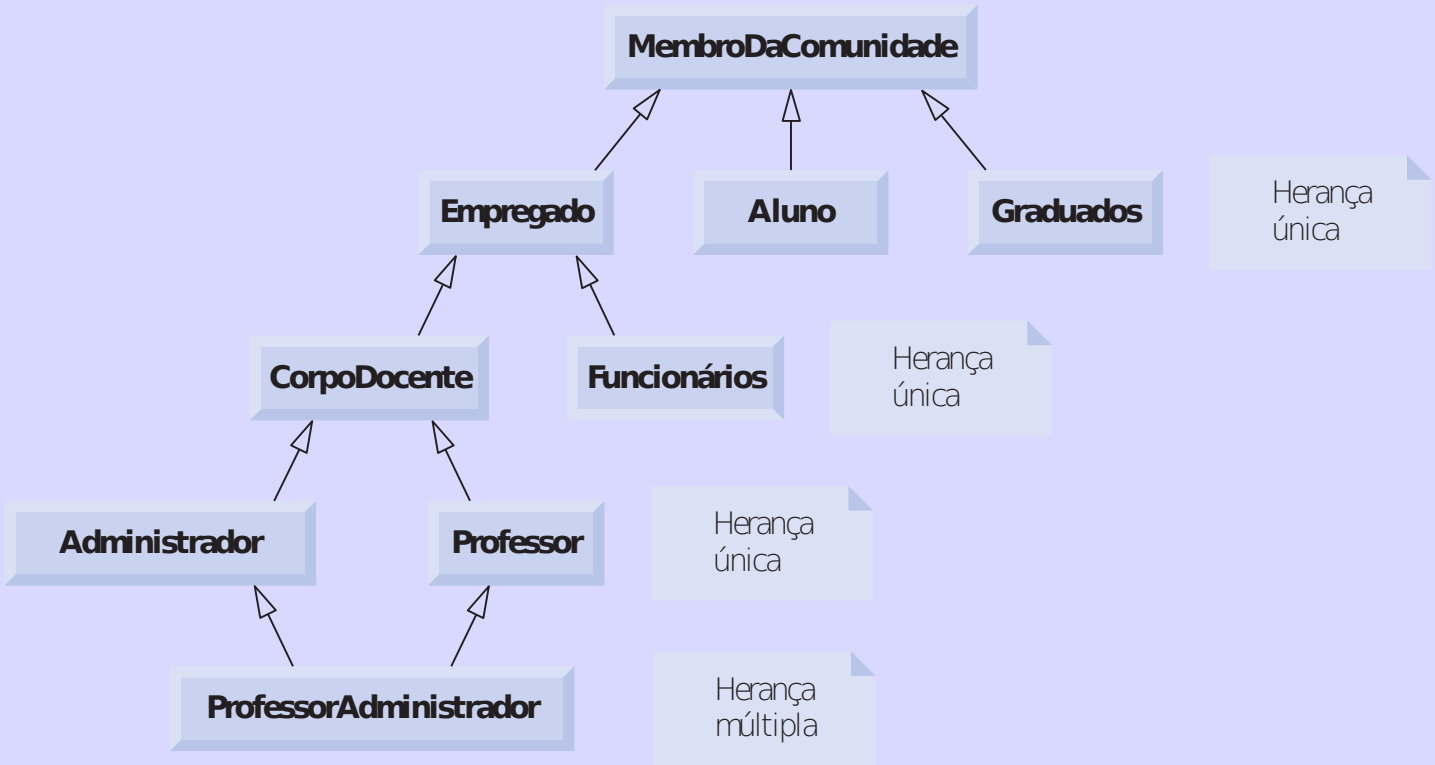
- Crie uma classe **VIP**, que herda **Ingresso** e possui um valor adicional. Sobrescreva o método **imprimeValor** para imprimir o valor do ingresso **VIP** (com o adicional incluído);
- Crie uma classe **CamaroteInferior** (que possui a localização do ingresso e métodos para acessar e imprimir esta localização) e uma classe **CamaroteSuperior**, que é ainda mais cara.

# Multiplicidade de Heranças

## A herança pode ser:

- **Simples**: a classe derivada deriva apenas de uma superclasse;
- **Múltipla**: a classe derivada deriva de várias superclasses.

## Diagrama UML



# Multiplicidade de Heranças

## Sintaxe

```
NomeDaClasse: especificador_acesso ClasseBase1,
    especificador_acesso ClasseBase2, ...,
    especificador_acesso ClasseBaseN
{
 // Implementação da classe
};
```

## Exemplo

Classes **Forma**, **Retangulo**, **CustoDeTinta**

# Veículos Rodoviários

## Exercício – Implemente a classe abaixo

```
class VeiculoRodoviario
{
public:
    VeiculoRodoviario();
    VeiculoRodoviario(int r, int p);
    // métodos set e get
private:
    int rodas;
    int passageiros;
}
```

# Veículos Rodoviários

## Exercício – Implemente as classe abaixo como herdeiras de **VeiculoRodoviario**

```
class Caminhao
{
public:
    Caminhao();
    Caminhao(int c, int r,
        int p)
    // métodos set e get
    void mostrar();
private:
    int carga;
};
```

```
enum tipo {car, van, vagao};
class Automovel
{
public:
    Automovel();
    Automovel(tipo t, int r,
        int p);
    void setTipo(tipo t);
    enum tipo getTipo();
    void mostrar();
private:
  enum tipo tipoCarro;
};
```

# Referências e Exercícios

## Livro Texto

Paul Deitel e Harvey Deitel.**C++: Como programar**. 5ªed. Ed. Prentice Hall Brasil. 2006.

## Conteúdo dessa Aula

- Capítulo 12
- Seção 24.6

## Exercícios do Livro

12.3, 12.4, 12.6, 12.7 e 12.9