Disciplina: POO- Programação Orientada a Objetos

Herança
Polimorfismo

Profª. Drª. Giovana Angélica Ros Miola
giovana.miola@fatec.sp.gov.br

Centro Paula Souza

Fatec Presidente Prudente

Herança

- Herança é um relacionamento do tipo generalização/especialização, onde uma classe pode ser derivada (subclasse) ou outra mais geral (classe base)
- A classe derivada absorve todas as características fundamentais e adiciona outras novas características, de tal modo a torná-la mais especializada.

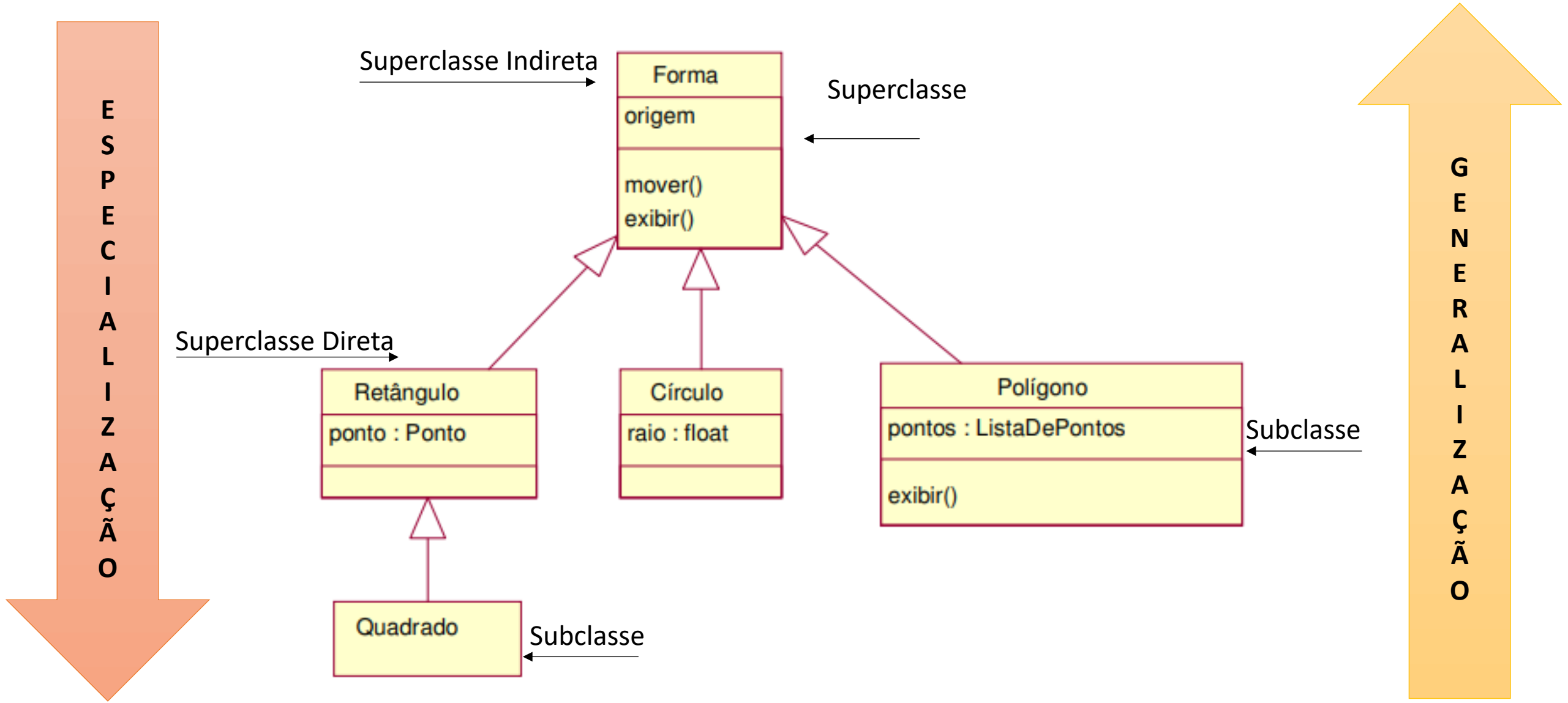
Herança

- Ocorre quando uma classe passa a herdar características (**atributos e métodos**) definidas em uma outra classe (base)
- Possibilita o **compartilhamento** ou o **reaproveitamento/extensão** (**reuso**) de recursos definidos anteriormente em outra classe, como atributos e métodos
- A classe original é chamada de superclasse, base, classe pai/mãe e a classe originada é denominada subclasse, classe derivada, classe filha(o)
- Implica um relacionamento "**é um**"
- Utiliza o caractere **:** para representar a herança (java – extends)

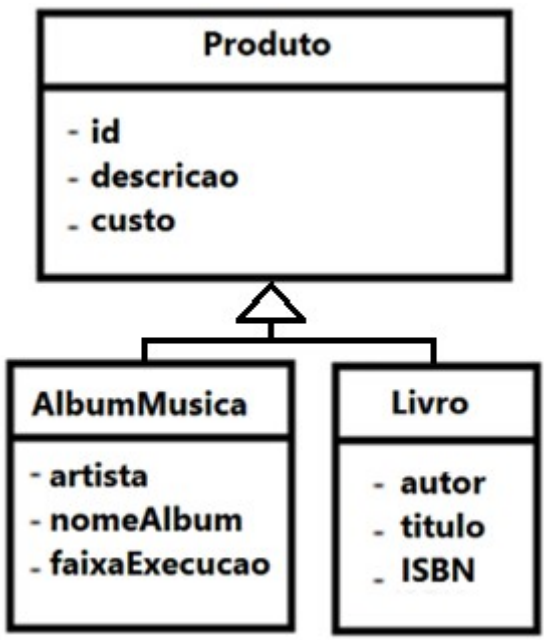
Herança

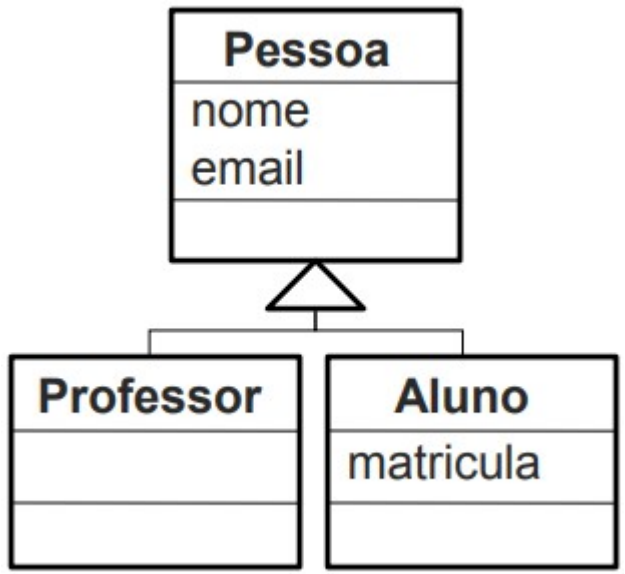
- A classe derivada é um subtipo da classe base
- A classe derivada pode ou não ter membros/atributos diferentes dos herdados
- **Generalização** - Obtém similaridades entre classes e define novas classes. As classes mais genéricas são as superclasses
- **Especialização** - Identifica atributos e métodos não correspondentes entre classes distintas colocando-os na subclasse

Herança


ESPECIALIZAÇÃO
GENERALIZAÇÃO
Superclasse Indireta
Forma
origem
mover()
exibir()
Superclasse
Superclasse Direta
Retângulo
ponto : Ponto
Círculo
raio : float
Polígono
pontos : ListaDePontos
exibir()
Subclasse
Quadrado
Subclasse

Herança- Exemplos





Herança

- Herança Simples
  - uma classe é derivada de **APENAS uma** superclasse
- Herança Múltipla
  - uma subclasse é derivada de mais de uma superclasse
  - C# não suporta este tipo de herança
- Problema
  - uma subclasse pode herdar **métodos** que ela não necessita ou deveria ter
- Solução
  - Subscrever (redefinir) o método da superclasse com a implementação mais adequada ao contexto da subclasse(polimorfismo)
- Observação
  - método da superclasse pode ser acessado a partir da subclasse precedendo o nome do método com a palavra-chave **base.**

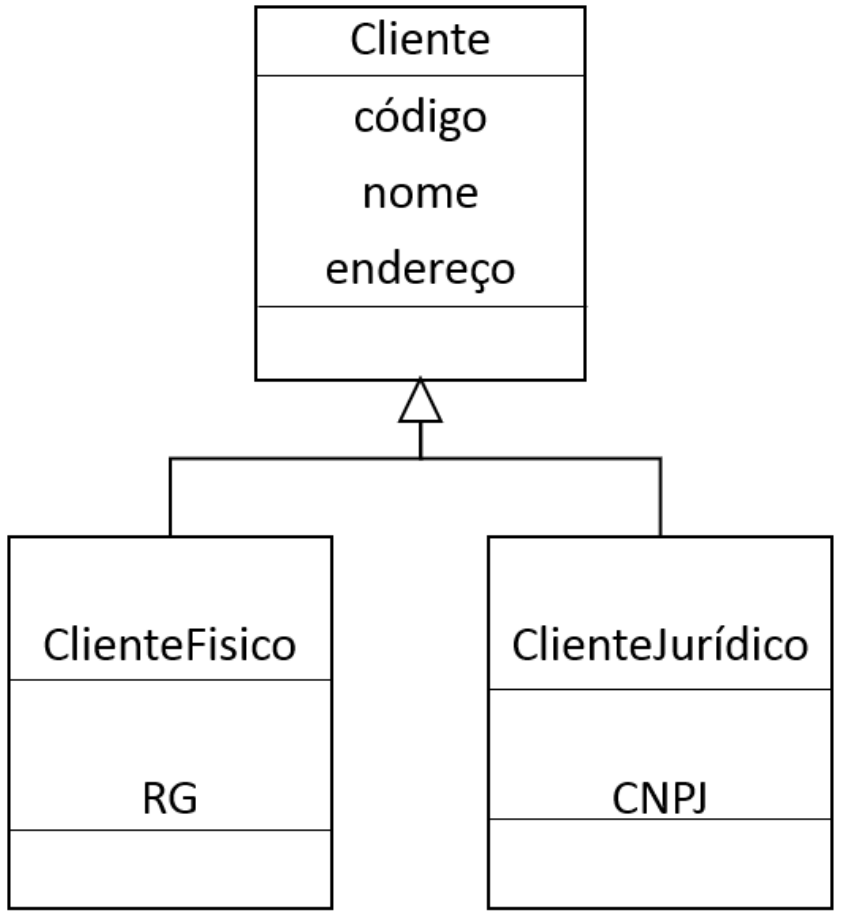
Modificador de acesso para Herança

- **protected**
  - Que fica entre o private e o public.
  - Um atributo protected só pode ser acessado (visível) pela própria classe **ou** suas subclasses dentro do projeto.
  - Apenas os atributos ou métodos da **superclasse** terão o protected.

SEM Reuso de Código
Implementação mais Longa

```
public class ClienteFisico
{
    private int codigo;
    private string nome;
    private string endereco;
    private string RG;
    //métodos
}
class ClienteJuridico
{
    private int codigo;
    private string nome;
    private string endereco;
    private string CNPJ;
    //métodos
}
```

Mudando para Herança



Implementando...

```
class Cliente
{
    protected int codigo;
    protected string nome;
    protected string endereco;
    public int Codigo
    {
        get { return codigo; }
        set { codigo = value; }
    }
    public string Nome
    {
        get { return nome; }
        set { nome = value; }
    }
    public string Endereco
    {
        get { return endereco; }
        set { endereco = value; }
    }
    public virtual void MostraDados()
    {
        Console.WriteLine("Código: " + codigo +
            "\tNome: " + nome +
            "\tEndereço: " + endereco);
    }
}
```

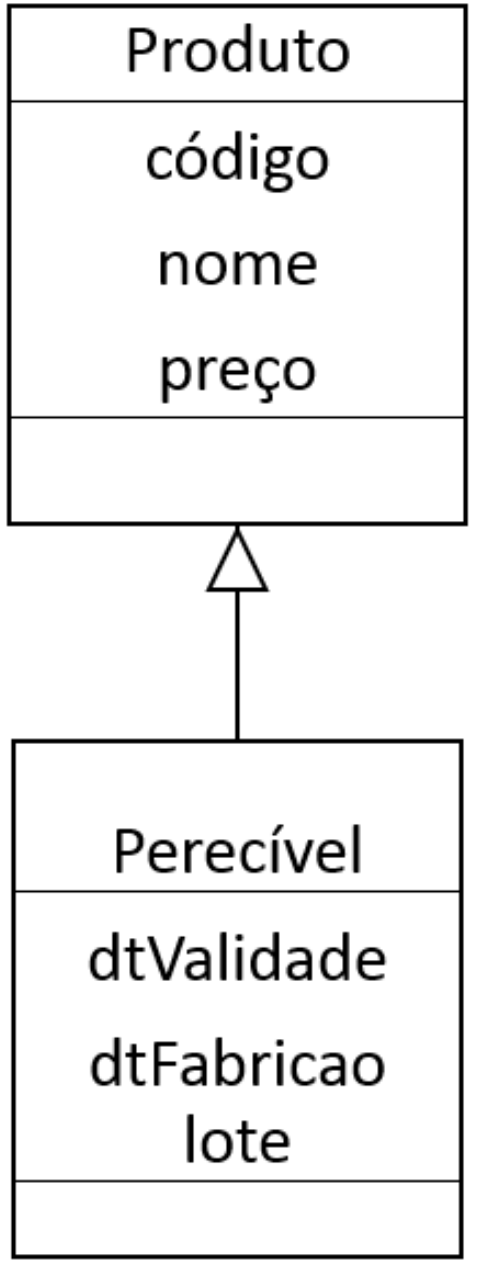
```
class ClienteFisico: Cliente
{
    private int rg;
    public int Rg
    {
        get { return rg; }
        set { rg = value; }
    }
    public override void MostraDados()
    {
        base.MostraDados();
        Console.WriteLine("\tRG: " +rg);
    }
}
```

```
class ClienteJuridico: Cliente
{
    private int cnpj;
    public int Cnpj
    {
        get { return cnpj; }
        set { this.cnpj = value; }
    }
    public override void MostraDados()
    {
        base.MostraDados();
        Console.WriteLine("\tCNPJ: " + cnpj);
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        //instância da classe base - superclasse
        Cliente c = new Cliente();
        c.Codigo = 56;
        c.Nome = "Bia";
        c.Endereco = "Rua Piauí, 10";
        c.MostraDados();
        //instância da subclasse - classe derivada
        ClienteFisico cf = new ClienteFisico();
        cf.Codigo = 59;
        cf.Nome = "Clara";
        cf.Endereco = "Rua Terezina,75";
        cf.MostraDados();
        //instância da subclasse - classe derivada
        ClienteJuridico cj = new ClienteJuridico();
        cj.Codigo = 1000;
        cj.Nome = "Empresa Esperança";
        cj.Endereco = "Rua Amazonas,79";
        cj.MostraDados();
        Console.ReadKey();
    }
}
```

Exercício

- Implementem as classes, métodos e instancias para a situação demonstrada no diagrama de classes.



Exercício sobre Herança

Crie a superclasse Funcionário(código, nome, salário)

- Crie uma subclasse Mensalista (qtdeHorasTrabalhadas)
- Crie outra subclasse Horista (qtdeHorasSemana)
- Criar o construtor de todas as classes.
- Na main() instanciar um objeto referente a cada classe, usando os construtores.

Polimorfismo

- Capacidade de um objeto responder a uma mensagem de várias maneiras
- É a possibilidade de uma ou mais classes responderem a mesma mensagem, cada uma de uma forma diferente.
- O modificador de acesso virtual indica que o método pode ser **sobrescrito** na classe derivada/subclasse
- Duas formas de polimorfismo:
  - **Polimorfismo com sobrecarga de métodos**
  - **Polimorfismo com herança e métodos virtuais**

Polimorfismo com sobrecarga de métodos
Polimorfismo **ESTÁTICO**

- Tipo mais simples de polimorfismo
- Resolvido pelo compilador em tempo de compilação
- **A sobrecarga ocorre dentro da mesma classe e**
- **Com quantidades de parâmetros diferentes.**

```
class Teste
{
    public void FazAlgo()
    {
        Console.WriteLine("Não tem parâmetro");
    }
    public void FazAlgo(string mensagem)
    {
        Console.WriteLine(mensagem);
    }
}
```

Exemplo de Polimorfismo com **sobrecarga de métodos**

```
class Pessoa
{
    string nome;
    string sexo;
    int idade;
    double salario;

    calcularSalario( ) // mensal
    { ... };

    calcularSalario(int qtdeHoras)
    { ... };

    calcularSalario(int qtdeHoras, string titularidade)
    { ... };

}
```

```
// ... Main...
f1.calcularSalario( );

f2.calcularSalario( 10 );

f3.calcularSalario( 23, “mestre”)
...
```

Polimorfismo com herança e métodos virtuais
Polimorfismo **DINÂMICO**

- A sobrescrita somente é possível em classes herdadas
- Substitui um método da superclasse na subclasse sobrescrevendo/ substituindo o mesmo método.
- Tipo mais complicado de polimorfismo, resolvido **dinamicamente** em tempo de execução
- Ocorre quando uma subclasse tem um método com a mesma assinatura da superclasse
- **Sobrescrita redefine o método, ocorre em classes separadas e**
- **Com mesma quantidade de parâmetros**

```
class Teste
{
    public virtual void FazAlgo()
    {
        Console.WriteLine("Classe base");
    }
}
// ...
class NovoTeste: Teste
{
    public override void FazAlgo()
    {
        Console.WriteLine("Classe derivada");
    }
}
```

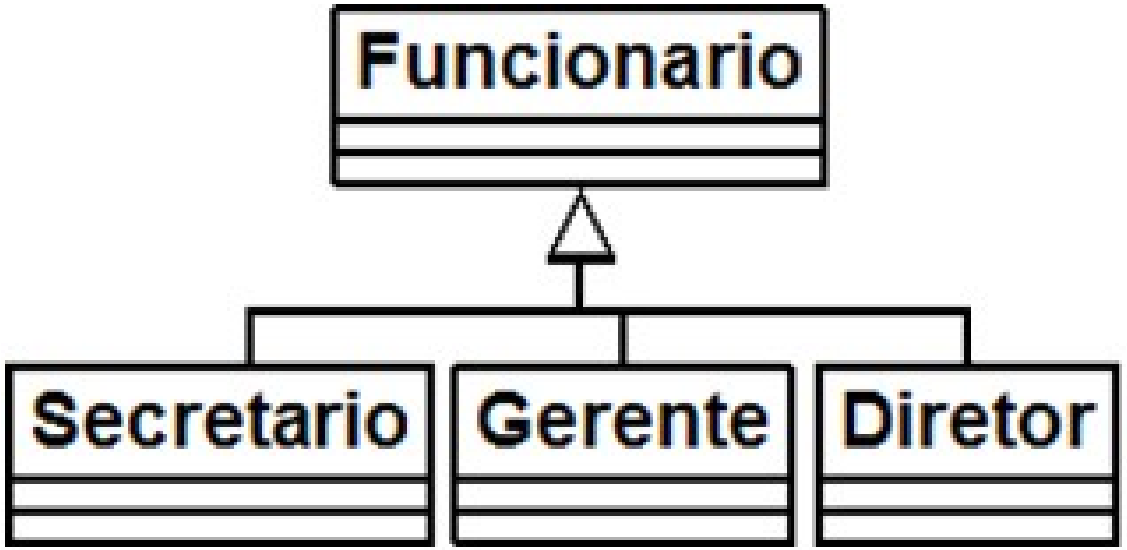
Exemplo de Polimorfismo com **sobrescrita de métodos**

```
class Pessoa{
    String nome;
    String sexo;
    double salario;

    virtual void calcularSalario( )
    { ... };
}

class Funcionario: Pessoa {
     int cargo;
     override void calcularSalario( )
     { ... };
     mostrarDados ( )
     { ... };
}
```

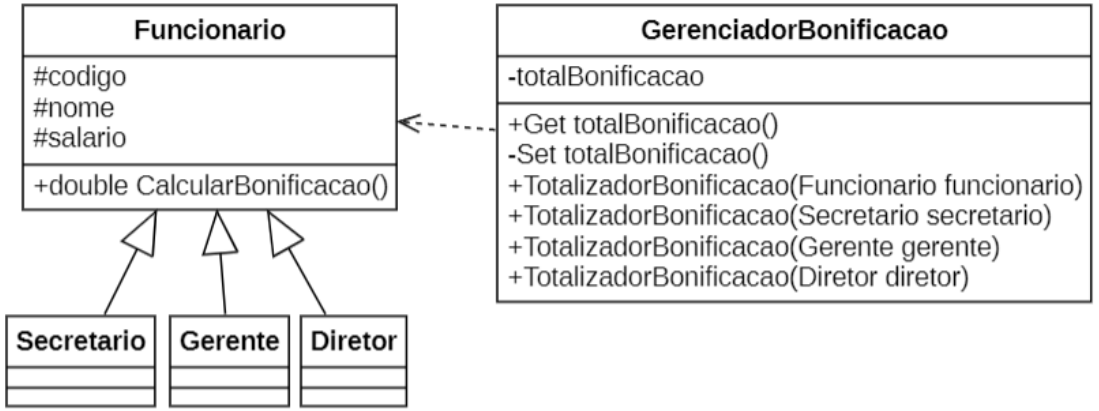
```
Pessoa P1 = new Pessoa()
P1. calcularSalario( );

Funcionario F1 = new Funcionario();
F1. calcularSalario( );
```

Exercício

- Todo fim de ano, **todos** os funcionários (código, nome e salário) do banco recebem uma bonificação de 10%.
- Os gerentes recebem 15%.
- Os diretores terão direito a mais R$ 1000,00.
- Implementem as classes, métodos e instancias para tal situação.



Exercício



- Complemente o projeto, referente ao slide anterior
- As classes Funcionário, Secretario, Gerente e Diretor já foram desenvolvidas.
- Implemente a classe GerenciadorBonificacao que, por meio da **associação de dependência (relacionamento fraco, ocorre por meio da passagem de parâmetro)**, com a Funcionário somará o totalBonificacao (atributo), utilizando o método TotalizadorBonificacao(*)
- * representa um parâmetro de um objeto de uma classe, a do Funcionário, do Secretario, do Gerente e do Diretor, ou seja, implemente 4 métodos, cada um com um parâmetro referente a cada uma das classes.
- Isto indicará o polimorfismo estático, cada vez com um objeto diferente (4 classes).
- Exemplo do método:

```
public void TotalizadorBonificacao(Diretor diretor){
        this.TotaldeBonificacao += diretor.CalcularBonificacao();}
```

- Este deverá ter a lógica de somar, por meio do atributo a bonificação de cada tipo de funcionário