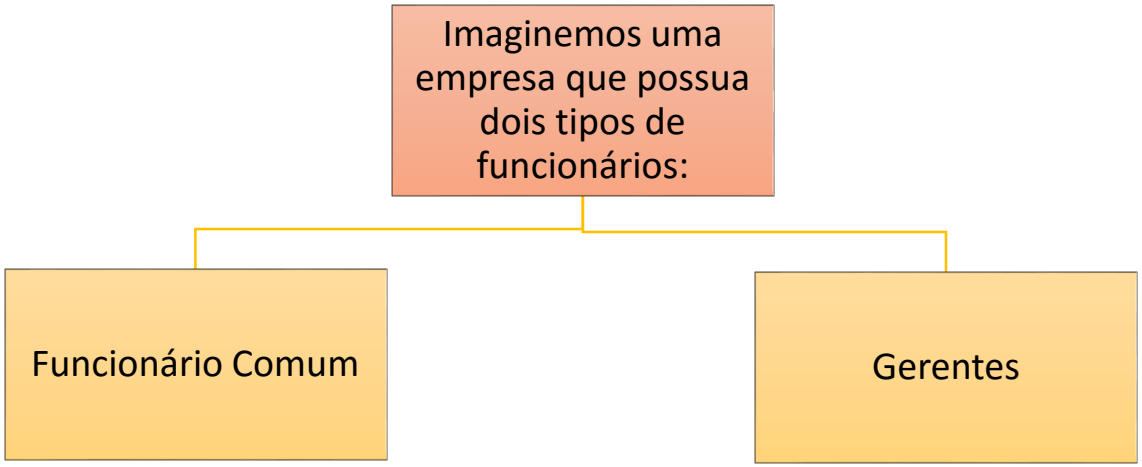
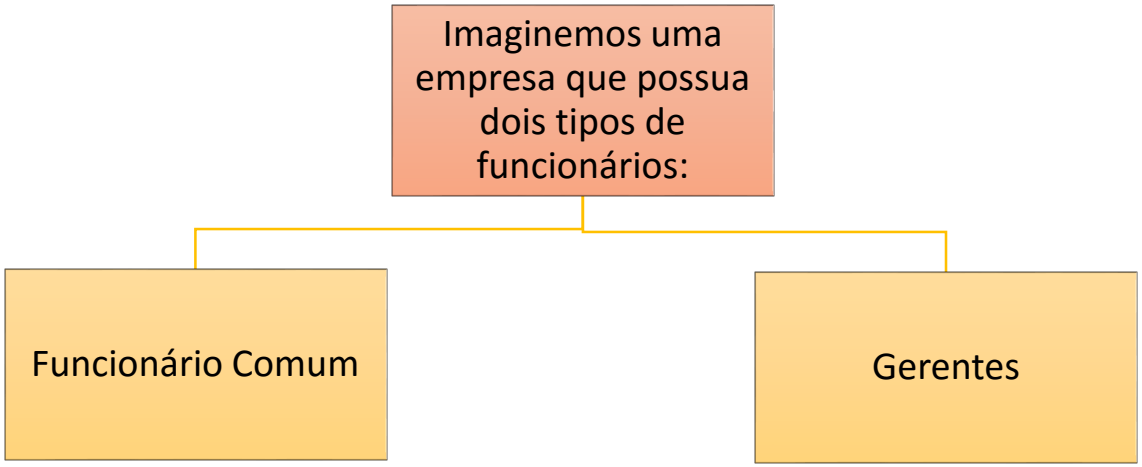
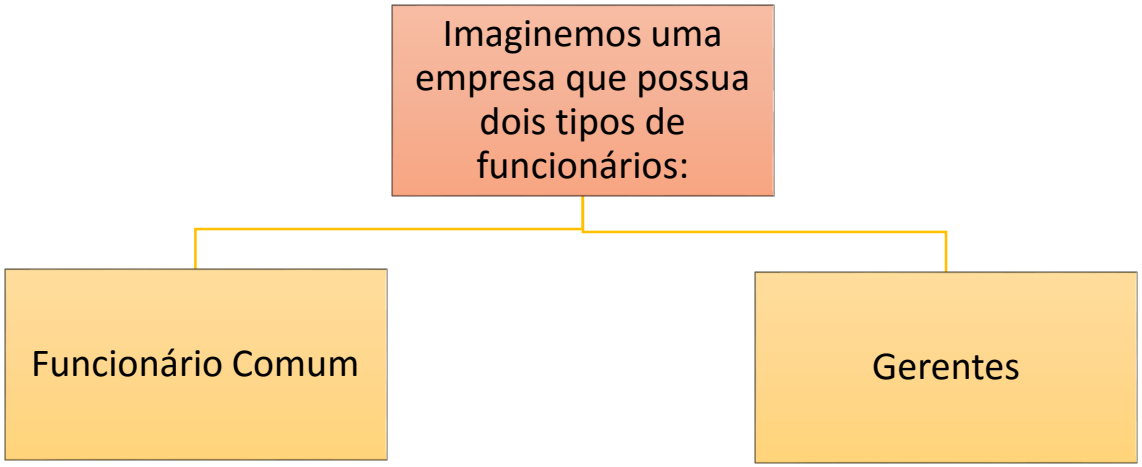
Programação Orientada a Objetos

Unidade 5 – Herança, reescrita e polimorfismo


Imaginemos uma empresa que possua dois tipos de funcionários:
Funcionário Comum
Gerentes

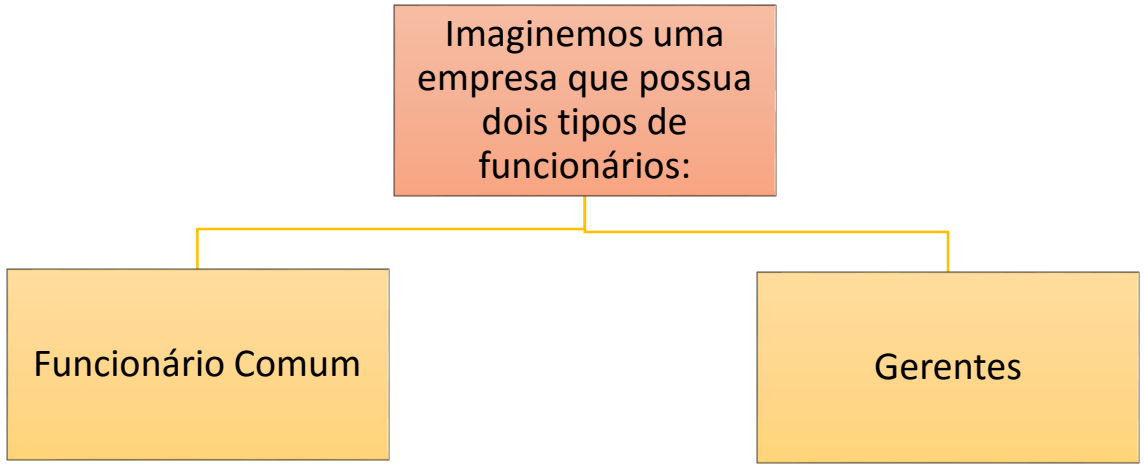



```java
public class Funcionario
{
    String nome;
    String cpf;
    double salario;
    // métodos devem vir aqui
}
```




```
public class Funcionario
{
    String nome;
    String cpf;
    double salario;
    // métodos devem vir aqui
}
```

```
public class Gerente
{
    String nome;
    String cpf;
    double salario;
    int senha;
    public bool Autenticar(int senha)
    {
        if (this.senha == senha)
        {
            Console.WriteLine("Acesso Permitido!");
            return true;
        }
        else
        {
            Console.WriteLine("Acesso Negado!");
            return false;
        }
    }
}
```




```
public class Funcionario
{
    String nome;
    String cpf;
    double salario;
    // métodos devem vir aqui
}
```

```
public class Gerente
{
    String nome;
    String cpf;
    double salario;
    int senha;
    public bool Autenticar(int senha)
    {
        if (this.senha == senha)
        {
            Console.WriteLine("Acesso Permitido!");
            return true;
        }
        else
        {
            Console.WriteLine("Acesso Negado!");
            return false;
        }
    }
}
```

Imaginemos uma empresa que possua dois tipos de funcionários:

Funcionário Comum

Gerentes

E se tivéssemos outro tipo de funcionário?

```
public class Funcionario
{
    String nome;
    String cpf;
    double salario;
    // métodos devem vir aqui
}
```

```
public class Gerente
{
    String nome;
    String cpf;
    double salario;
    int senha;
    public bool Autenticar(int senha)
    {
        if (this.senha == senha)
        {
            Console.WriteLine("Acesso Permitido!");
            return true;
        }
        else
        {
            Console.WriteLine("Acesso Negado!");
            return false;
        }
    }
}
```

Imaginemos uma empresa que possua dois tipos de funcionários:

Funcionário Comum

Gerentes

E se tivéssemos outro tipo de funcionário?

E se precisássemos adicionar uma nova informação para todos os funcionários?

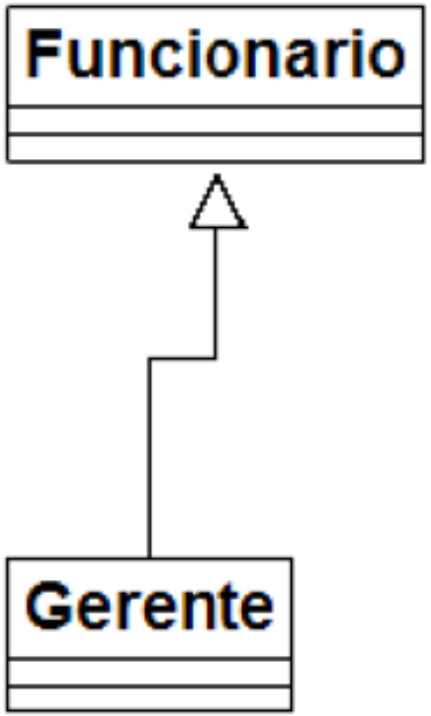
```
public class Funcionario
{
    String nome;
    String cpf;
    double salario;
    // métodos devem vir aqui
}
```

```
public class Gerente
{
    String nome;
    String cpf;
    double salario;
    int senha;
    public bool Autenticar(int senha)
    {
        if (this.senha == senha)
        {
            Console.WriteLine("Acesso Permitido!");
            return true;
        }
        else
        {
            Console.WriteLine("Acesso Negado!");
            return false;
        }
    }
}
```

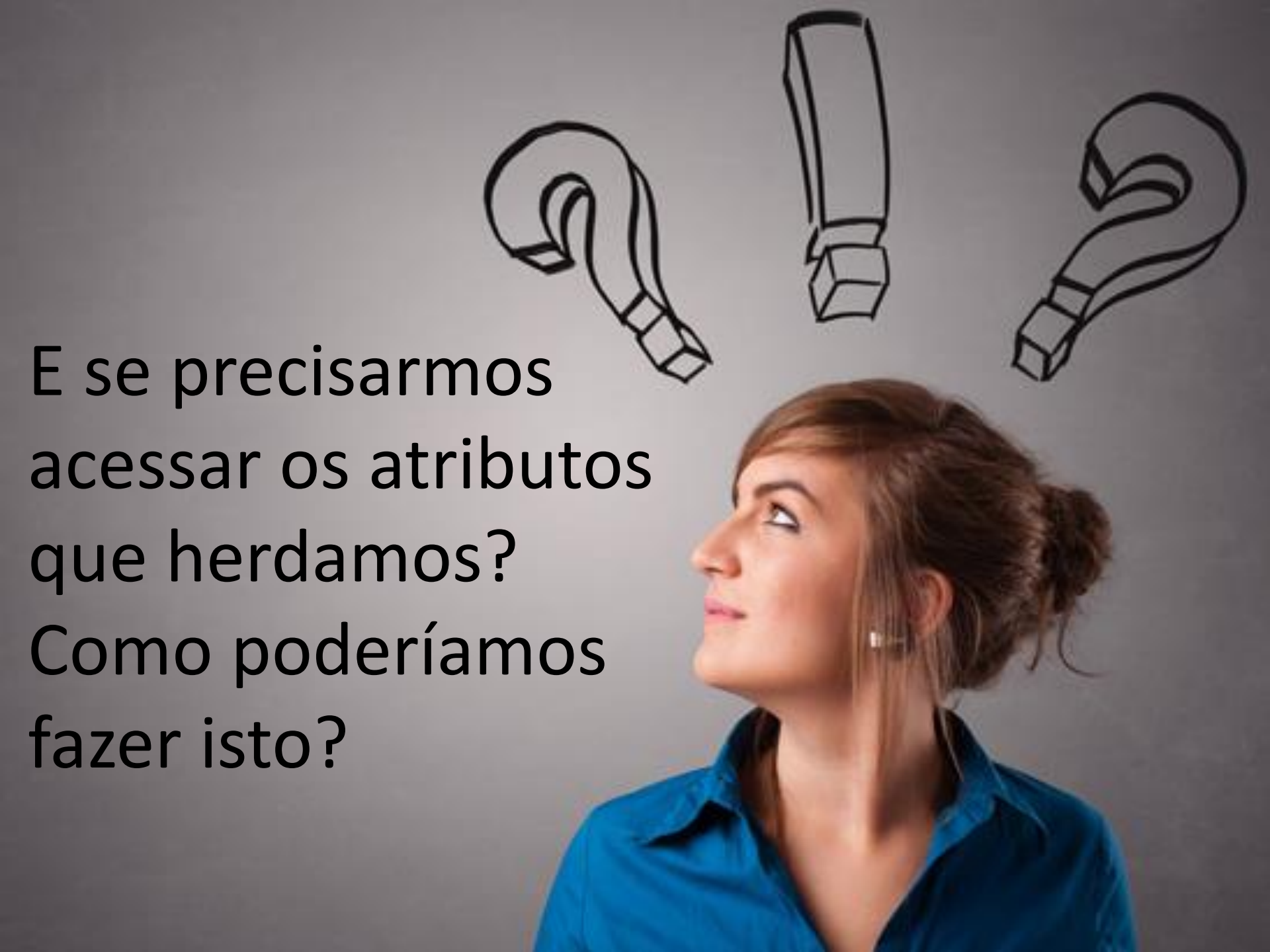
Existe um jeito, em C#, de relacionarmos uma classe de tal maneira que uma delas **herda** tudo que a outra tem. Isto é uma relação de classe mãe e classe filha.

```
public class Gerente : Funcionario
{
    private int _senha;

    public bool Autenticar(int senha)
    {
        if (this._senha == senha)
        {
            Console.WriteLine("Acesso Permitido!");
            return true;
        }
        else
        {
            Console.WriteLine("Acesso Negado!");
            return false;
        }
    }
}
```



No nosso caso, gostaríamos de fazer com que o **Gerente** tivesse tudo que um **Funcionário** tem, gostaríamos que ela fosse uma **extensão** de **Funcionário**. Fazemos isto através do símbolo “:”.


E se precisarmos
acessar os atributos
que herdamos?
Como poderíamos
fazer isto?

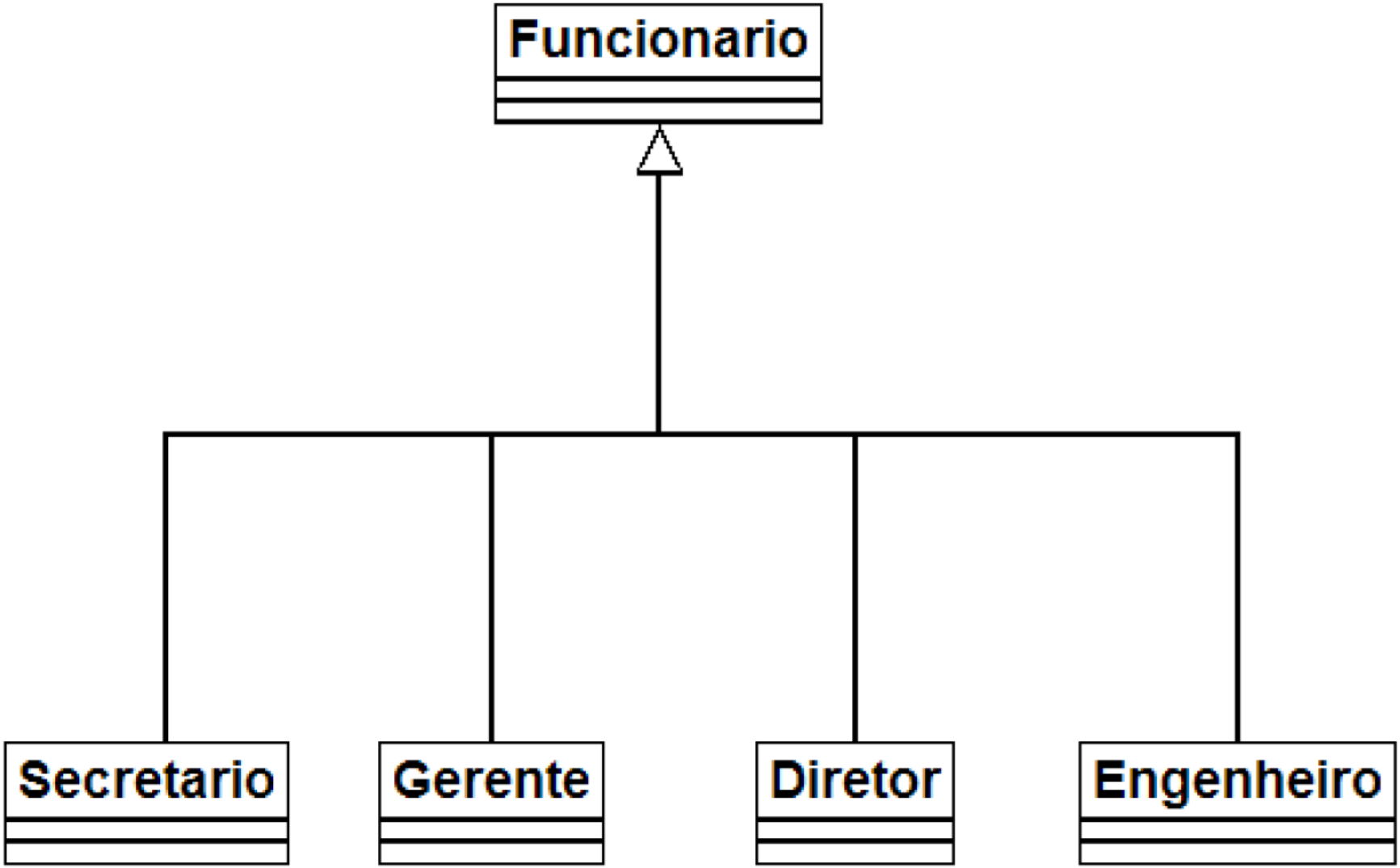
```
public class Funcionario
{
        protected string Nome;

        //atributos e métodos
}
```

```
public class Gerente : Funcionario
{
    public bool Autenticar(string senha)
    {
        if (_senha == senha)
        {
            Console.WriteLine("Usuário {0} foi autenticado", Nome);
            return true;
        }
        else
        {
            Console.WriteLine("Senha inválida!");
            return false;
        }
    }

}
```

Existe um outro modificador de acesso, o **protected**, que **fica entre o private e o public**. Um atributo protected **só pode ser acessado (visível) pela própria classe ou suas subclasses**.



Uma classe pode ter várias filhas, mas **pode ter apenas uma mãe**, é a chamada **herança simples do C#.**

Todo fim de ano, os funcionários da nossa empresa recebem uma bonificação: **Funcionários** comuns recebem **10%** do valor do salário e **Gerentes** recebem **15%** do valor do salário

```
public class Funcionario
{
    protected string Nome;

    private string _cpf;

    public double Salario;

    private DateTime _dataAdmissao;

    public double BuscarBonificacao()
    {
        return Salario * 0.10;
    }
}
```

Se deixarmos a classe **Gerente** como ela está, ela vai herdar o método **BuscarBonificacao**.

No C#, quando herdamos um método, podemos alterar seu comportamento. Podemos **reescrever**

Descomplicando Polimorfismo...

O que guarda uma variável do tipo **Funcionario**?
Uma referência para um **Funcionario**.

Descomplicando Polimorfismo...

O que guarda uma variável do tipo **Funcionario**?
Uma referência para um **Funcionario**.

Na **herança**, vimos que todo **Gerente é um Funcionario**, pois é uma extensão deste.

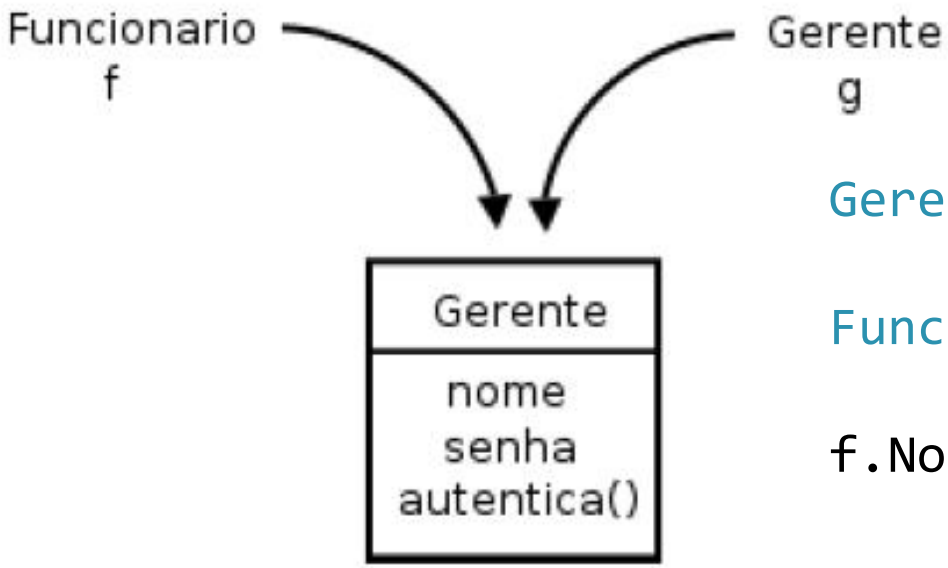
Descomplicando Polimorfismo...

O que guarda uma variável do tipo **Funcionario**?
Uma referência para um **Funcionario**.

Na **herança**, vimos que todo **Gerente é um Funcionario**, pois é uma extensão deste.

Podemos nos referir a um Gerente como sendo um Funcionario. Se alguém precisa falar com um Funcionario do banco, pode falar com um Gerente!
Porque? Pois **Gerente é um Funcionario**.

Descomplicando Polimorfismo...




```
Gerente g = new Gerente();

Funcionario f = g;

f.Nome = “João Vitor Moretto”;
```

Polimorfismo é a capacidade de um **objeto poder ser referenciado de várias formas**. Cuidado! polimorfismo não quer dizer que o objeto fica se transformando, muito pelo contrário, um objeto nasce de um tipo e morre daquele tipo, o que pode mudar é a maneira como nos referimos a ele

```
Gerente gerente = new Gerente();
Funcionario funcionario = gerente;
funcionario.Salario = 5000.0;

funcionario.BuscaBonificacao();
```

```
Gerente gerente = new Gerente();
Funcionario funcionario = gerente;
funcionario.Salario = 5000.0;

funcionario.BuscaBonificacao();
```

Qual é o retorno desse método?

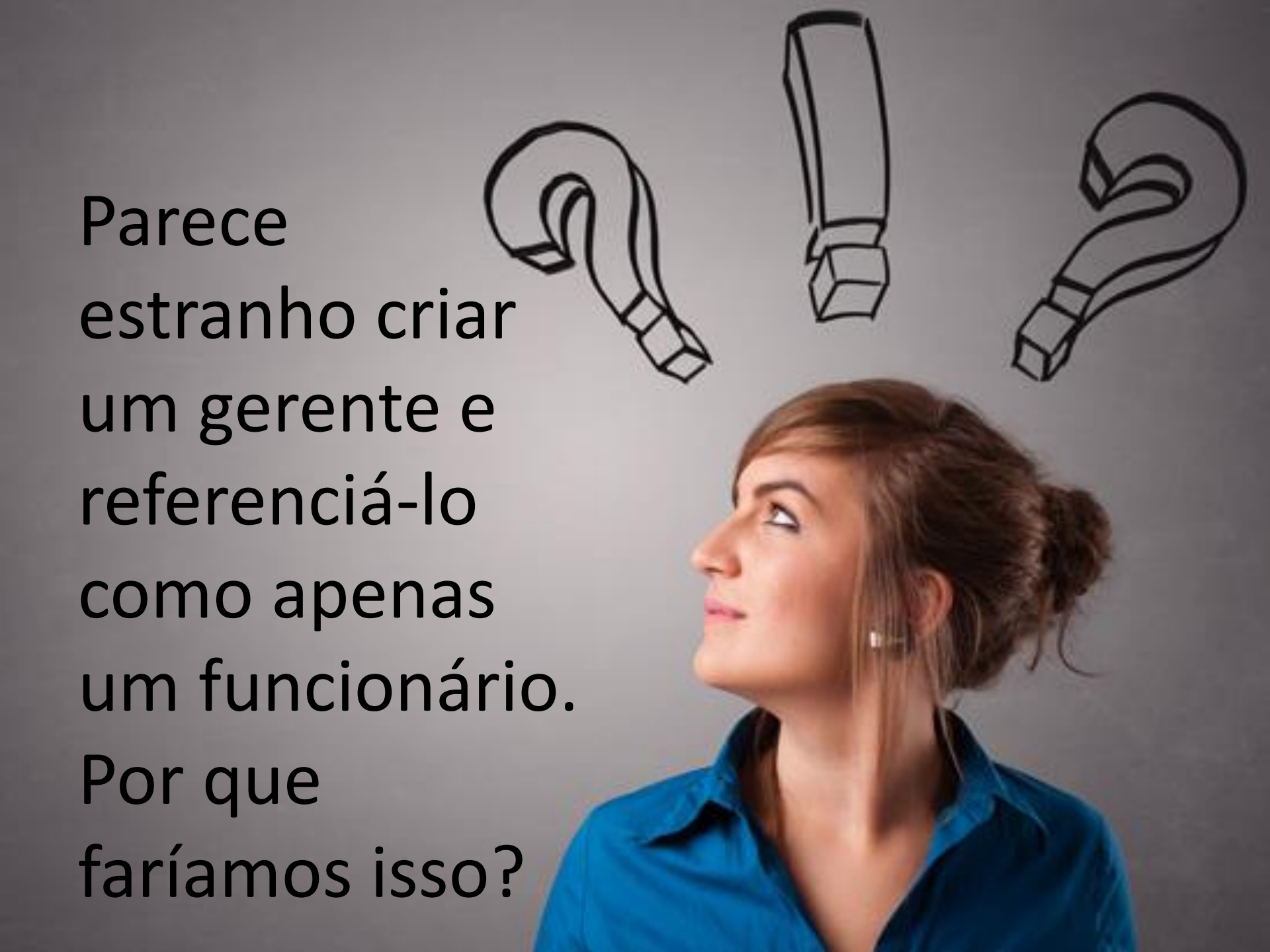
500 ou 750?

```
Gerente gerente = new Gerente();
Funcionario funcionario = gerente;
funcionario.Salario = 5000.0;

funcionario.BuscaBonificacao();
```

No C#, a **invocação de método sempre vai ser decidida em tempo de execução**. O C# vai procurar o objeto na memória e, aí sim, decidir qual método deve ser chamado, sempre relacionando com sua classe de verdade, e não com a que estamos usando para referenciá-lo.

Parece estranho criar um gerente e referenciá-lo como apenas um funcionário. Por que faríamos isso?



Como saber o valor gasto em bonificações de todos os funcionários?

```
public class ControladorDeBonificacao
{
    private double totalDeBonificacoes = 0;

    public void Registrar(Funcionario f)
    {
        this.totalDeBonificacoes += f.BuscarBonificacao();
    }

    public double BuscaTotalDeBonificacoes()
    {
        return this.totalDeBonificacoes;
    }
}
```

```
public class ControladorDeBonificacao
{
    private double totalDeBonificacoes = 0;

    public void Registrar(Funcionario f)
    {
        this.totalDeBonificacoes += f.BuscarBonificacao();
    }

    public double BuscaTotalDeBonificacoes()
    {
        return this.totalDeBonificacoes;
    }
}
```

E, em algum lugar da aplicação (ou no Main se for apenas para testes):

```
public class ControladorDeBonificacao
{
    private double totalDeBonificacoes = 0;

    public void Registrar(Funcionario f)
    {
        this.totalDeBonificacoes += f.BuscarBonificacao();
    }

    public double BuscaTotalDeBonificacoes()
    {
        return this.totalDeBonificacoes;
    }
}
```

E, em algum lugar da aplicação (ou no Main se for apenas para testes):

```
ControladorDeBonificacao controlador = new ControladorDeBonificacao();

Gerente funcionario1 = new Gerente();
funcionario1.Salario = 5000.0;
controlador.Registrar(funcionario1);

Funcionario funcionario2 = new Funcionario();
funcionario2.Salario = 1000.0;
controlador.Registrar(funcionario2);

Console.WriteLine(controlador.BuscaTotalDeBonificacoes());
```

Programação Orientada a Objetos

Unidade 5 – Herança, reescrita e polimorfismo