

INSTITUTO MAUÁ DE TECNOLOGIA



# Linguagens I

Herança

Profº. Tiago Sanches da Silva

# Exercícios

# Prática 2 - Exercício 1

- **Modelar** a classe Funcionarios de uma concessionaria.
- Crie um novo projeto: Concessionaria
- Inicie a implementação
- Discuta a solução com o professor se necessário
- RH deve conseguir acessar essas informações
  - (13º, comissão, férias, salario do mês)
- Comissão = 3% de todas as vendas
- salario do mês = salario base + horas extras + comissão

Funcionarios

# Prática 2 - Exercício 2

A concessionaria possui 2 cargos principais:

Vendedor:

- Recebe comissão por venda, como o funcionário do exercício passado. Mas agora armazena quem é o gerente responsável.
- Comissão: 3% de todas vendas feita pelo próprio vendedor.

Gerente de vendas:

- Pode vender, dar aumento a funcionários mas apenas se forem de sua responsabilidade.
- Possui uma senha para acesso ao sistema.
- Possui atributo que indica o número de vendedores que estão sob sua gerência.

Modele as duas classes.



Herança

# Repetindo o código

Vamos criar uma classe para funcionários de um banco:

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

# Repetindo o código

Além de um funcionário comum, há também outros cargos, como os gerentes. Os gerentes tem as mesmas informações que um funcionário comum, mas possuem informações adicionais e outras funcionalidades. Por exemplo, um gerente no nosso banco possui também uma **senha** numérica que permite o acesso ao sistema interno do banco.

```
public class Gerente {  
    private String nome;  
    private String cpf;  
    private double salario;  
    private int senha;  
    private int numeroDeFuncionariosGerenciados;  
  
    public boolean autentica(int senha) {  
        if (this.senha == senha) {  
            System.out.println("Acesso Permitido!");  
            return true;  
        } else {  
            System.out.println("Acesso Negado!");  
            return false;  
        }  
    }  
    // outros métodos  
}
```

# Repetindo o código

Compare:

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

public class Gerente {
    private String nome;
    private String cpf;
    private double salario;
    private int senha;
    private int numeroDeFuncionariosGerenciados;

    public boolean autentica(int senha) {
        if (this.senha == senha) {
            System.out.println("Acesso Permitido!");
            return true;
        } else {
            System.out.println("Acesso Negado!");
            return false;
        }
    }

    // outros métodos
}
```



# Repetindo o código

Se tivéssemos um outro tipo de funcionário que possui algumas características diferentes do funcionário comum, precisaríamos criar uma outra classe e copiar o código novamente!

Além disso, se um dia precisarmos adicionar uma nova informação para todos os funcionários, precisaremos passar por todas as classes de funcionário e adicionar esse atributo.

# Repetindo o código

Existe um jeito, em Java, 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**.

# Repetindo o código

No nosso caso, gostaríamos de fazer com que o Gerente tivesse tudo que um Funcionario tem, ou seja, gostaríamos que ela fosse uma **extensão** de Funcionario.

# Herança

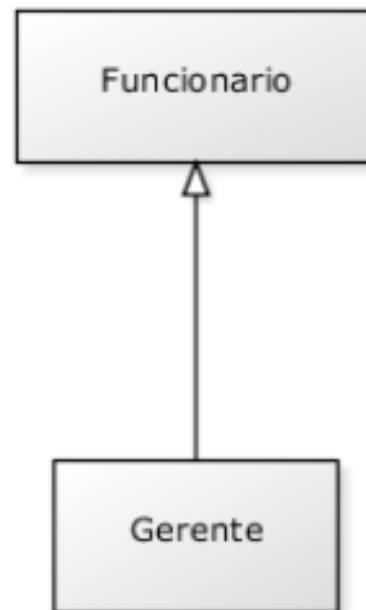
No Java fazemos isso com a palavra chave **extends**.

```
public class Gerente extends Funcionario {  
    private int senha;  
    private int numeroDeFuncionariosGerenciados;  
  
    public boolean autentica(int senha) {  
        if (this.senha == senha) {  
            System.out.println("Acesso Permitido!");  
            return true;  
        } else {  
            System.out.println("Acesso Negado!");  
            return false;  
        }  
    }  
    // outros métodos  
}
```

E como ficam os outros atributos?

# Herança

Em todo momento que criarmos um objeto do tipo Gerente, este objeto possuirá também os atributos definidos na classe Funcionario, pois um Gerente é um Funcionario:



Dizemos que a classe **Gerente herda** todos os atributos e métodos da **classe mãe**, no nosso caso, a **Funcionario**.

## Super e Sub classe

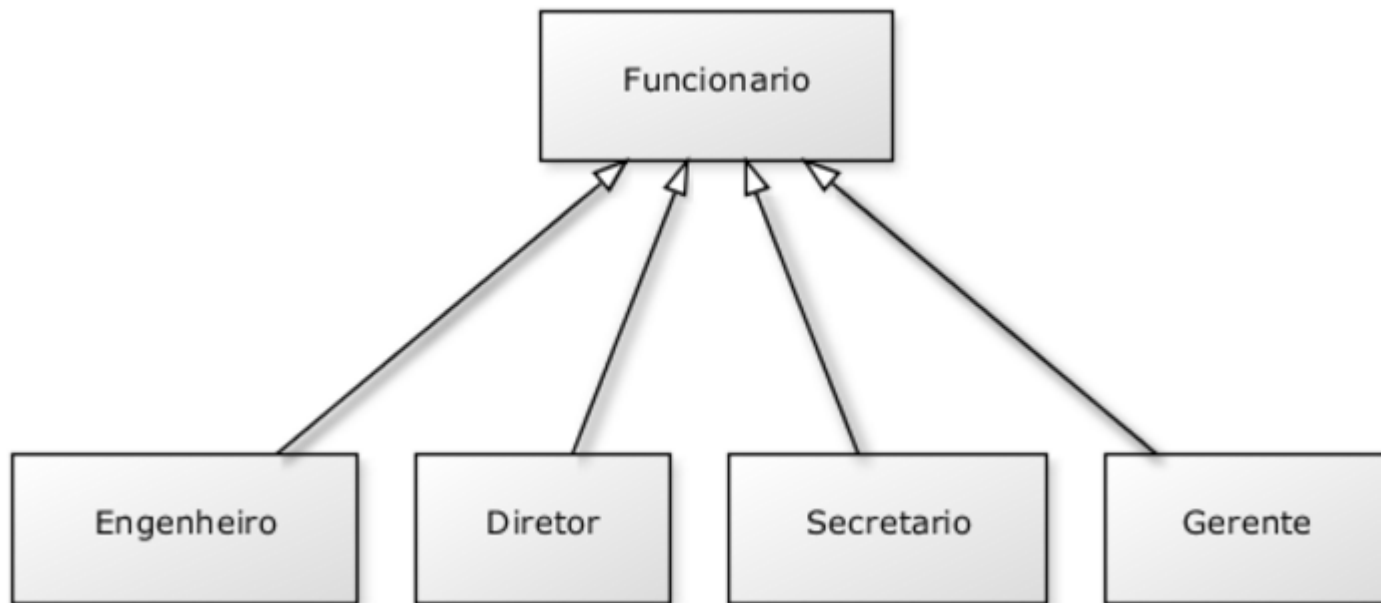
A nomenclatura mais encontrada é que **Funcionario** é a **superclasse** de **Gerente**, e **Gerente** é a **subclasse** de **Funcionario**. Dizemos também que todo **Gerente** é **um** **Funcionário**. Outra forma é dizer que **Funcionario** é classe **mãe** de **Gerente** e **Gerente** é classe **filha** de **Funcionario**.

Podemos ter uma classe **Diretor** que estenda **Gerente** e a classe **Presidente** pode estender diretamente de **Funcionario**.

Fique claro que essa é uma decisão de **negócio**. Se **Diretor** vai estender de **Gerente** ou não, vai depender se para você, **Diretor** é um tipo de **Gerente**.

# Herança

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



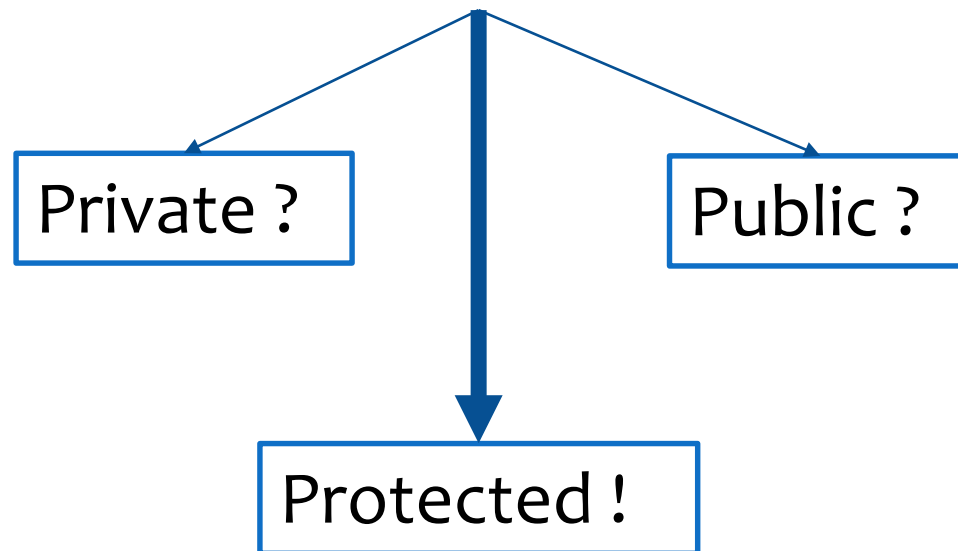


Verifiquem! Façam as alterações necessárias no exercício da concessionaria.

# Herança

A sub classe também herda os **atributos e métodos privados**, porém **não consegue acessá-los diretamente**.

E se precisamos acessar os atributos que herdamos?



## Protected !


Um atributo **protected** só pode ser acessado (visível) pela própria **classe** e por suas **subclasses**.

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

# Herança – Formalizando [1]

Herança é um mecanismo que permite que **características comuns** a diversas classes sejam fatoradas em uma classe base, ou **superclasse**. A partir de uma classe base, outras classes podem ser especificadas.

Cada classe derivada ou **subclasse** apresenta as características (estrutura e métodos) da superclasse e **acrescenta a elas o que for definido de particularidade para ela**.

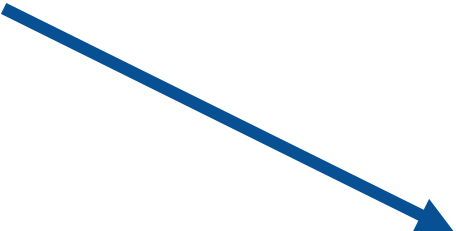


# Herança – Formalizando [2]

Herança é a capacidade de reutilizar código pela especialização de soluções genéricas já existentes.

```
// SuperClass.java  
public class SuperClass {  
    ...  
}
```

```
// SubClass.java  
public class SubClass extends SuperClass {  
    ...  
}
```



# Formas de Herança

1. **Extensão:** subclasse estende a superclasse, acrescentando novos membros (atributos e/ou métodos). A superclasse permanece inalterada, motivo pelo qual este tipo de relacionamento é normalmente referenciado como herança estrita.
2. **Especificação:** a superclasse especifica o que uma subclasse deve oferecer, mas não implementa nenhuma funcionalidade.
3. **Combinação de extensão e especificação:** a subclasse herda a interface e uma implementação padrão de (pelo menos alguns de) métodos da superclasse.

# Hierarquia de Classes

## Superclasse direta:

- Herdada explicitamente (um nível acima da hierarquia).

## Superclasse indireta:

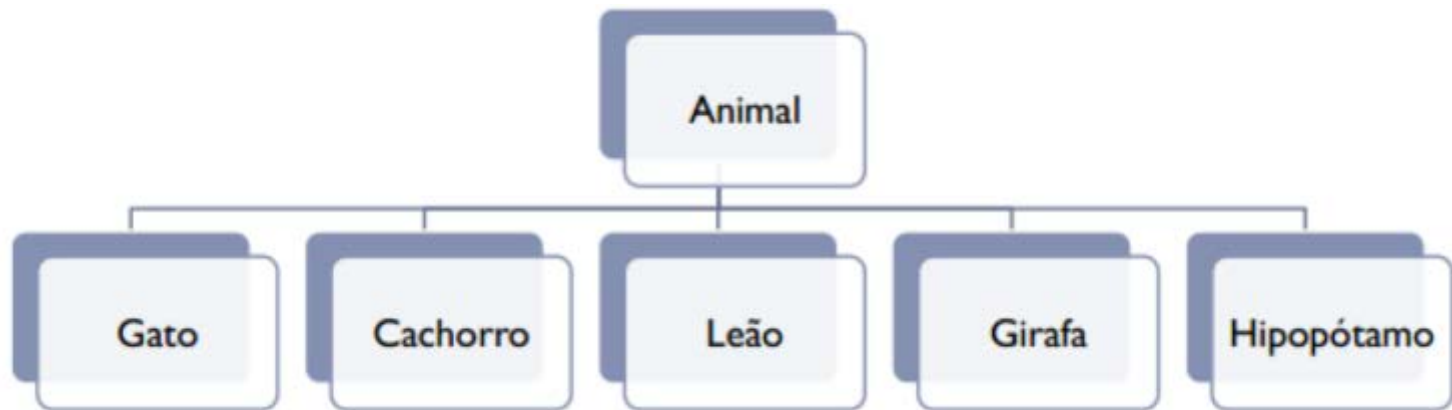
- Herdada de dois ou mais níveis acima da hierarquia.

Exercite sua capacidade de abstração.



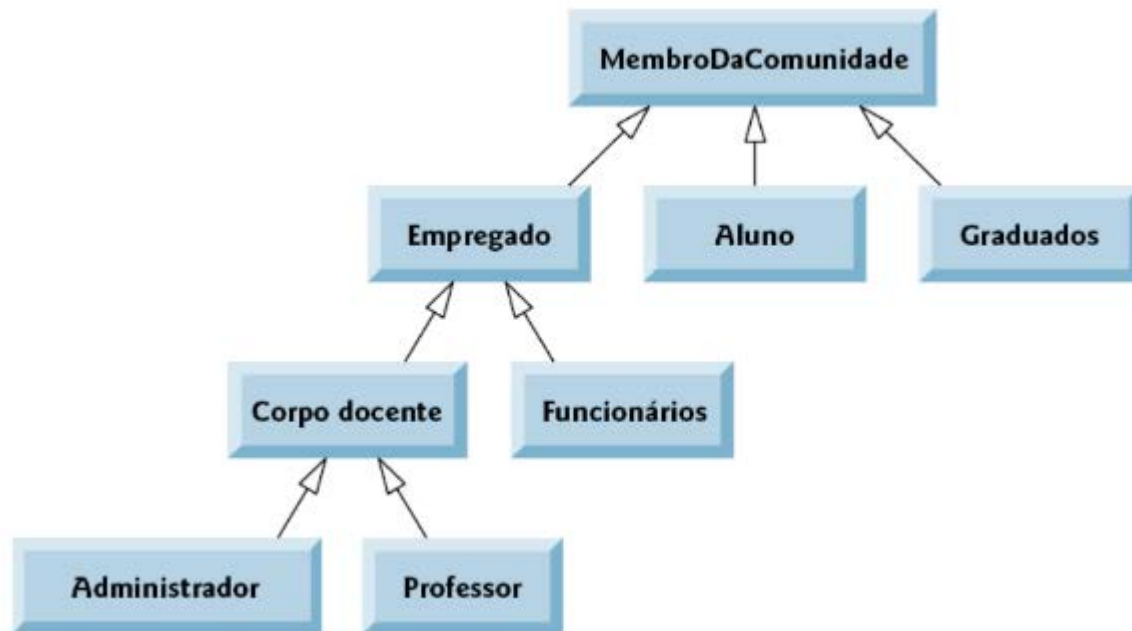
# Herança - Exemplos

O que é genérico e quais seriam as especializações? (atributos e métodos)



# Herança - Exemplos

O que é genérico e quais seriam as especializações? (atributos e métodos)



# Herança – Exercício

Como seria a linha de onde foi derivado o **celular**?

# Construtores e Herança

O construtor da superclasse, caso exista, sempre irá ser executado antes do construtor da subclasse. Pois podem existir variáveis da superclasse que precisam de inicialização.

# Construtores - super

E quando o construtor da superclasse espera alguns parâmetros?

Utilize super!

```
public class SuperClasse {  
    private int atributo;  
  
    public void SuperClasse ( int parametro ) {  
        this.atributo = parametro;  
    }  
}  
  
public class SubClasse extends SuperClasse {  
  
    public void SubClasse ( int parametroParaSuper ) {  
        super(parametroParaSuper);  
  
        // Continuação do construtor da sub classe  
    }  
}
```

# Visibilidade entre superclasse e subclasse

**private:** significa que a classe que herdou não possui aquele atributo?

**protected:** terei sempre que usar protected para classes que serão herdadas?

Como manter algum atributo **privado**, para não deixar a subclasse mexer diretamente sem um método de acesso?

Discussão em sala de aula:

- Vamos conversar sobre visibilidade e herança!

# Prática 2 - Exercício 3

Altere o software para que **Vendedor** e **Gerente** sejam subclasses de **Funcionario**.

E todo funcionário, sem exceção, possui um bônus extra mensal que depende do total de vendas de toda concessionaria.

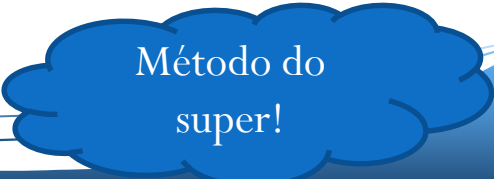
- 5% do valor de todas as vendas do mês dividido por todos os funcionários.

Vendedor

- Comissão: 3% de todas vendas feita pelo próprio vendedor adicionado do bônus extra mensal.

Gerente de vendas:

- Recebe comissão diferenciada: 5% de suas vendas + 25% do salario base + bônus extra mensal.



Método do  
super!

# Sobrescrita (Overriding)



Um dos mecanismos fundamentais na programação orientada a objetos é o conceito de sobrescrita (no termo em inglês, overriding) de métodos em classes derivadas.

A redefinição ocorre quando um método cuja assinatura já tenha sido especificada recebe uma nova definição (ou seja, um **novo corpo**) **em uma classe derivada**.

O mecanismo de sobrescrita, juntamente com o conceito de ligação dinâmica, é a chave para a utilização do **polimorfismo**.

**Aguarde as próximas aulas! 😊**



Perguntas?