

Herança e Polimorfismo

Mateus Raeder

Herança

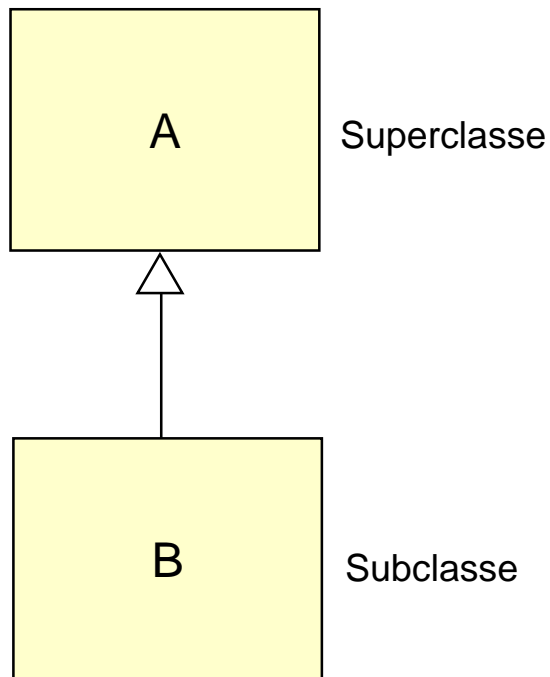
- ▶ **Ideia:** derivar uma nova classe a partir de uma outra já existente



Classe original: **SUPERCLASSE**
Classe derivada: **SUBCLASSE**

- **Por que o nome HERANÇA?**
 - Porque a **subclasse HERDA** os atributos e métodos da **superclasse**

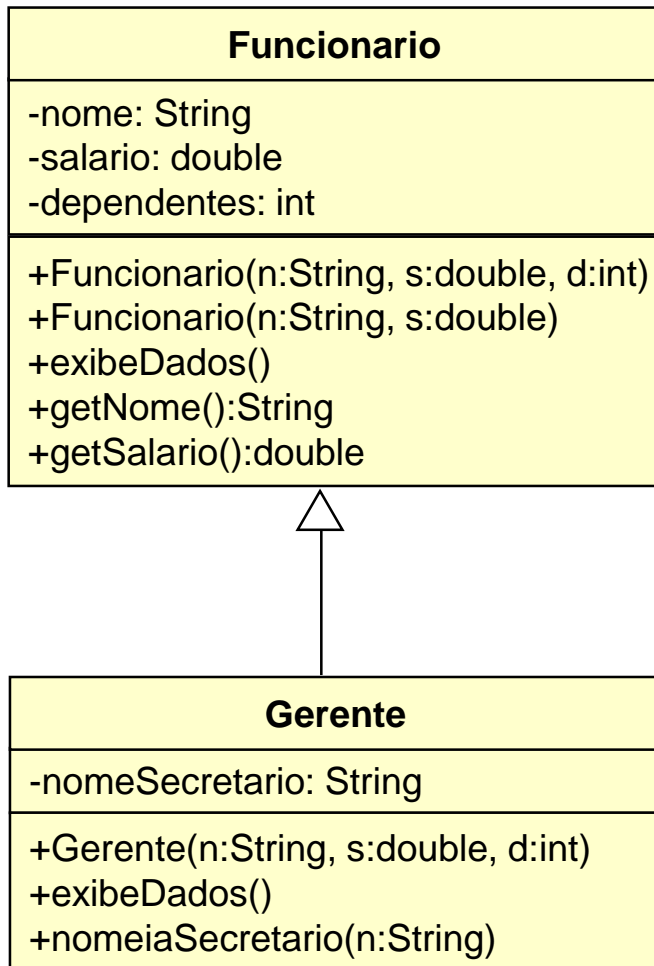
Herança



■ IMPORTANTE:

- todo objeto da subclasse B também É UM objeto da superclasse A
- a subclasse B apresenta características **ADICIONAIS** e **EXCLUSIVAS** em relação às da superclasse (por isso é mais especializada)

Herança



- **Gerente** é um tipo especial de Funcionário, com características adicionais e exclusivas
- Repare que o método `exibeDados()` é **SOBRESCRITO**, ou seja, **REDEFINIDO**
- O método `nomeiaSecretario` é **EXCLUSIVO** de Gerente

Herança

- **Mas como dizer que uma classe HERDA outra?**

- com a utilização da palavra reservada **extends**

- **Exemplo:**

```
public class Gerente extends Funcionario
{
    .
    .
    .
}
```

Herança

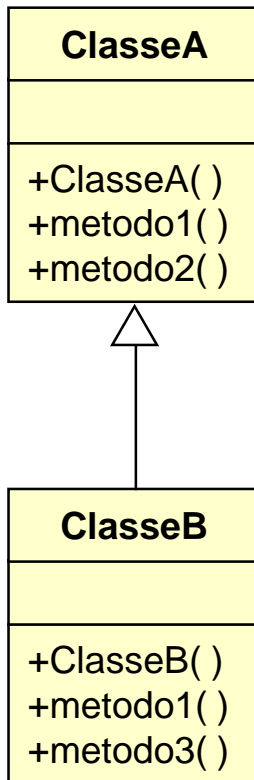
- **Ao implementar os métodos em uma subclasse, podemos:**
 - **sobrescrever métodos da superclasse:** é a redefinição de um método da superclasse, com a mesma assinatura (nome e parâmetros) e o mesmo tipo de retorno (*não confundir com **sobrecarga***). Ex.: `exibeDados()`
 - **herdar métodos da superclasse:** todo método da superclasse que não é sobrescrito, é herdado pela subclasse. Ex.: `getNome()`
 - **definir novos métodos:** são novos métodos que são exclusivos da subclasse, e não aparecem na superclasse. Ex.: `nomeiaSecretario(String n)`
- **Para os atributos, podemos:**
 - **herdar atributos da superclasse:** todo atributo da superclasse são herdados pela subclasse. Ex.: `nome`, `salario`
 - **definir novos atributos:** são novos atributos que são exclusivos da subclasse, e não aparecem na superclasse. Ex.: `nomeSecretario`

Herança

- Quando um objeto da subclasse chama um método, algumas situações podem ocorrer:
 - se o método foi **sobrescrito** pela subclasse, o da subclasse é acionado
 - se o método só existe na **superclasse**, o da superclasse será acionado
 - se o método é **exclusivo** da subclasse, este será acionado
 - se o método **não existe** nem na superclasse e nem na subclasse, ocorre um erro

Herança

- O que vai ocorrer em cada uma das linhas abaixo:



```
ClasseB b = new ClasseB();
```

```
b.metodo1();
```

```
b.metodo2();
```

```
b.metodo3();
```

```
b.metodo4();
```


Herança

■ Construtores da subclasse

- os construtores da superclasse não são herdados, logo, a subclasse deverá possuir seus próprios construtores para inicializar os seus atributos
- devemos, todavia, inicializar os atributos da classe que está sendo herdada, chamando algum construtor da superclasse com a utilização da chamada **super(...)**

```
public Gerente(String n, double s, int d, String nomeSecretario)
{
    super(n, s, d); //chama o construtor da superclasse
    this.nomeSecretario = nomeSecretario;
}
```

Herança

- **Quais atributos de sua superclasse uma subclasse pode acessar?**
 - os atributos do tipo `private` são acessíveis diretamente apenas para a classe que o possuir. Qualquer outra classe somente tem acesso a estes atributos utilizando os métodos de acesso públicos (GET, por exemplo).
 - Logo, suponha o método **exibeDados** na classe Gerente:

```
public void exibeDados()  
{  
    System.out.println("Nome: "+nome); //erro  
    System.out.println("Salário: "+salario); //erro  
    System.out.println("Dependentes: "+dependentes); //erro  
    System.out.println("Secretário: "+nomeSecretario);  
}
```

Herança

- **Quais atributos de sua superclasse uma subclasse pode acessar?**
 - porém, podemos chamar o método `exibeDados()` da superclasse:

```
public void exibeDados()  
{  
    super.exibeDados(); //acessa o método da superclasse  
    System.out.println("Secretário: "+nomeSecretario);  
}
```

- outra opção é definir os atributos da superclasse como **protected**. Um atributo **protected** é acessível à classe e às suas subclasses

Polimorfismo

- **Ideia:** realizar uma tarefa de formas diferentes (*poli* = muitas; *morphos* = formas)
- O objeto “assume a forma de outro”

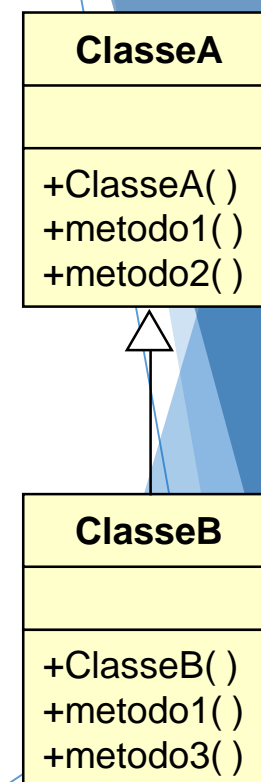
A instrução:

```
ClasseA a;
```

declara a como uma referência da superclasse ClasseA.

Regra 1: podemos atribuir um objeto da subclasse a uma referência da superclasse (*upcasting*)

```
a = new ClasseB(); //razoável, pois  
    ClasseB É UM ClasseA
```



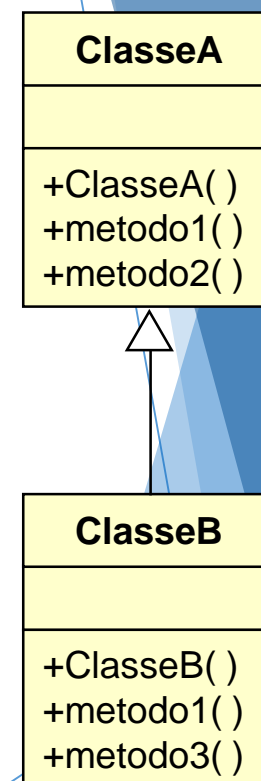
Polimorfismo

- Agora a, uma referência de superclasse, está apontando para um objeto de subclasse.
- Logo:

```
a.metodo1(); //chama metodo1() de ClasseB,  
           pois este foi sobrescrito  
a.metodo2(); //chama metodo2() de ClasseA  
a.metodo3(); //ERRO
```

tentativa de acessar através de uma referência de uma superclasse um método exclusivo da subclasse

Regra 2: uma referência de superclasse só reconhece membros da superclasse, mesmo que aponte para um objeto da subclasse



Polimorfismo

- Como acessar o metodo3() então?
- A resposta está no *downcasting*

Regra 3: a atribuição de um objeto de superclasse a uma referência de subclasse, sem coerção, não é permitida.

- Logo:

```
ClasseB b = a; //erro de compilação,  
               pois A não é um B
```

Devemos fazer a coerção (*casting*)

```
ClasseB b = (ClasseB) a;  
b.metodo3();
```

