Programação I

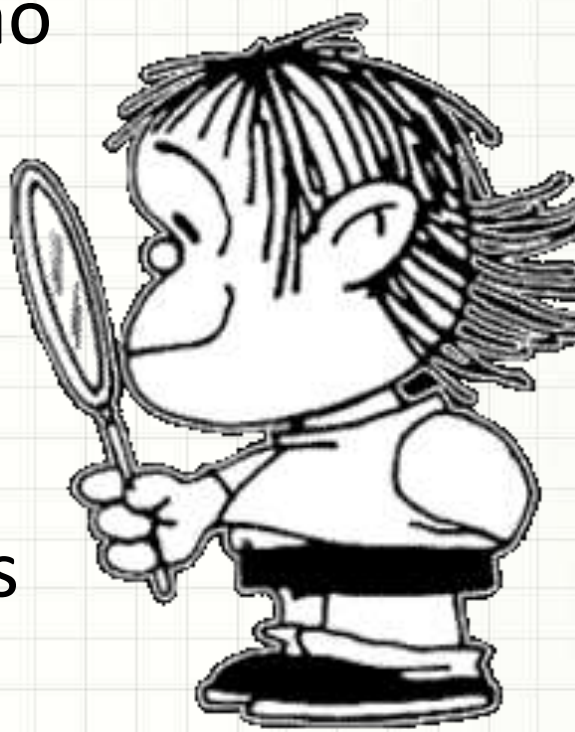
Herança e Polimorfismo

Prof. Dr. Daniel Caetano

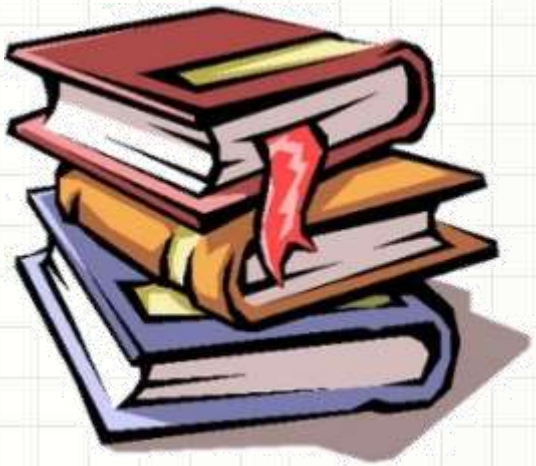
2017 - 1

Objetivos

- Compreender o conceito de herança na orientação a objetos
- Implementar herança para reuso de código e facilitar manutenção
- Compreender o conceito de polimorfismo
- Implementar polimorfismo aproveitando o potencial das linguagens orientadas a objetos



Material de Estudo



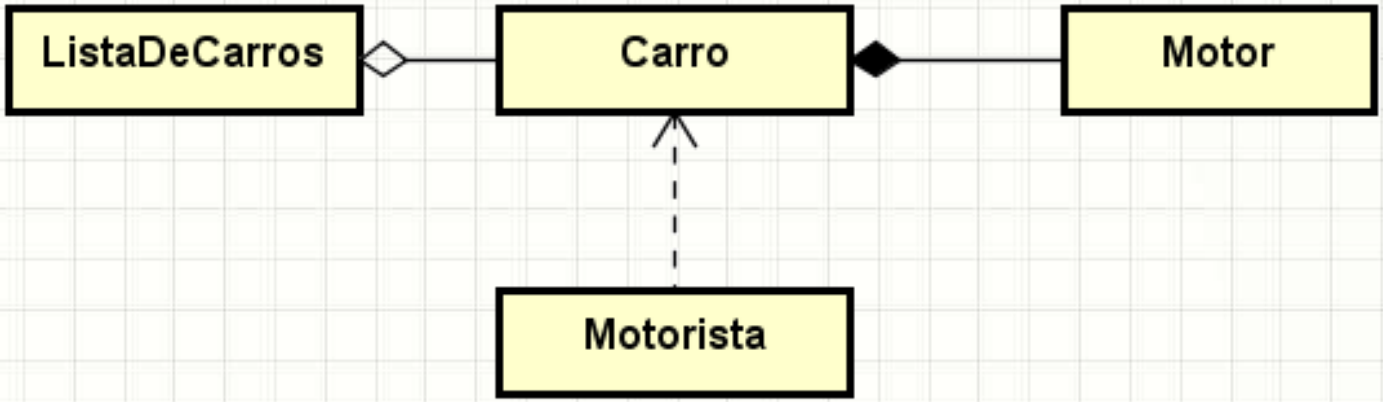
Material	Acesso ao Material
Apresentação	http://www.caetano.eng.br/ (Programação I – Aula 8)
Material Didático	Programação I –Págs 77 a 91

Retomando:

Relacionamentos de Objetos

Relacionamento de Objetos

- Existem três tipos de relações
  - Agregação: Carro → Lista de Carros em Oferta
  - Composição: Motor → Carro (venda de veículos)
  - Uso: Motorista → Carro



HERANÇA

O que é herança?

- Herança é uma das características mais úteis das classes... Mas o que é?
- O que é **herdar**?
- Receber algo que foi construído por outra pessoa

- Simplificadamente, em programação, herança é exatamente isso: receber algo que foi programado anteriormente

Como é essa tal de herança?

- Ao longo das aulas criamos uma classe para representar um **produto**
- Imagine agora que tenhamos de criar, para um sistema de uma livraria, uma classe para representar um **livro**
- No contexto da livraria, **o livro é um produto**!
- Isso significa que ele vai ter um código de identificação, um nome e uma descrição, como o nosso produto

Como é essa tal de herança?

- Temos duas alternativas:
  1. Criar uma classe chamada **Livro** e **<u>reescrever</u>** todo o código que já havíamos colocando na classe Produto
  2. Criar uma classe **Livro** e **<u>reaproveitar</u>** as funcionalidades que já estão disponíveis na classe Produto
- A opção dois pode ser feita justamente com a tal da **herança**

Herança em Java

- No Java, a herança funciona assim:
  1. Temos uma classe original, que chamamos de **superclasse** ou **classe pai**, da qual queremos herdar funcionalidades.
  2. E vamos criar uma classe derivada, que chamamos de **subclasse** ou **classe filha**, que irá **estender** as funcionalidades da classe pai

Herança em Java

- No Java, a herança funciona assim:
  1. Ter e **sup** s herdar fur
  2. E v namos de **subclasse** ou **classe filha**, que irá **extender** as funcionalidades da classe pai

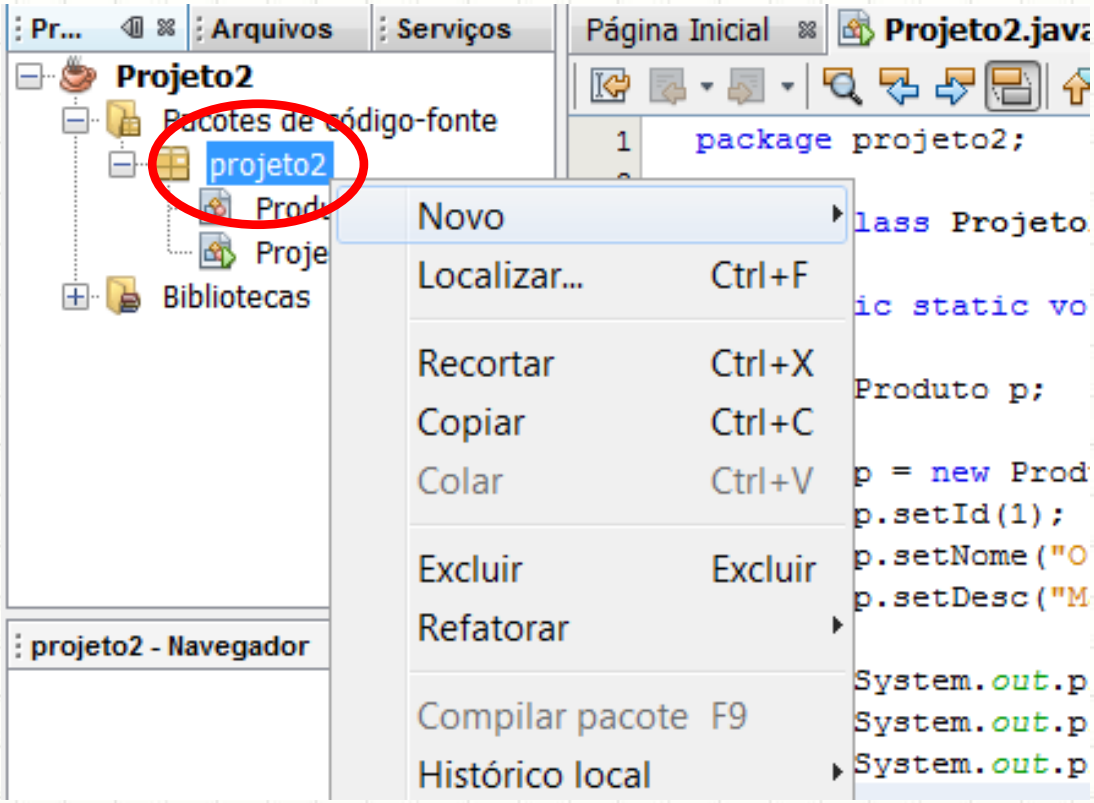
Apenas as características <u>públicas</u> e <u>protegidas</u> são herdadas...!

Herança em Java

- Observe que no processo de herança, **todas** as características <u>públicas</u> são herdadas
- Assim, no exemplo, um Livro não deixa de ser, também, um Produto... Todo objeto do tipo **Livro** será, ao mesmo tempo, um objeto do tipo **Produto**

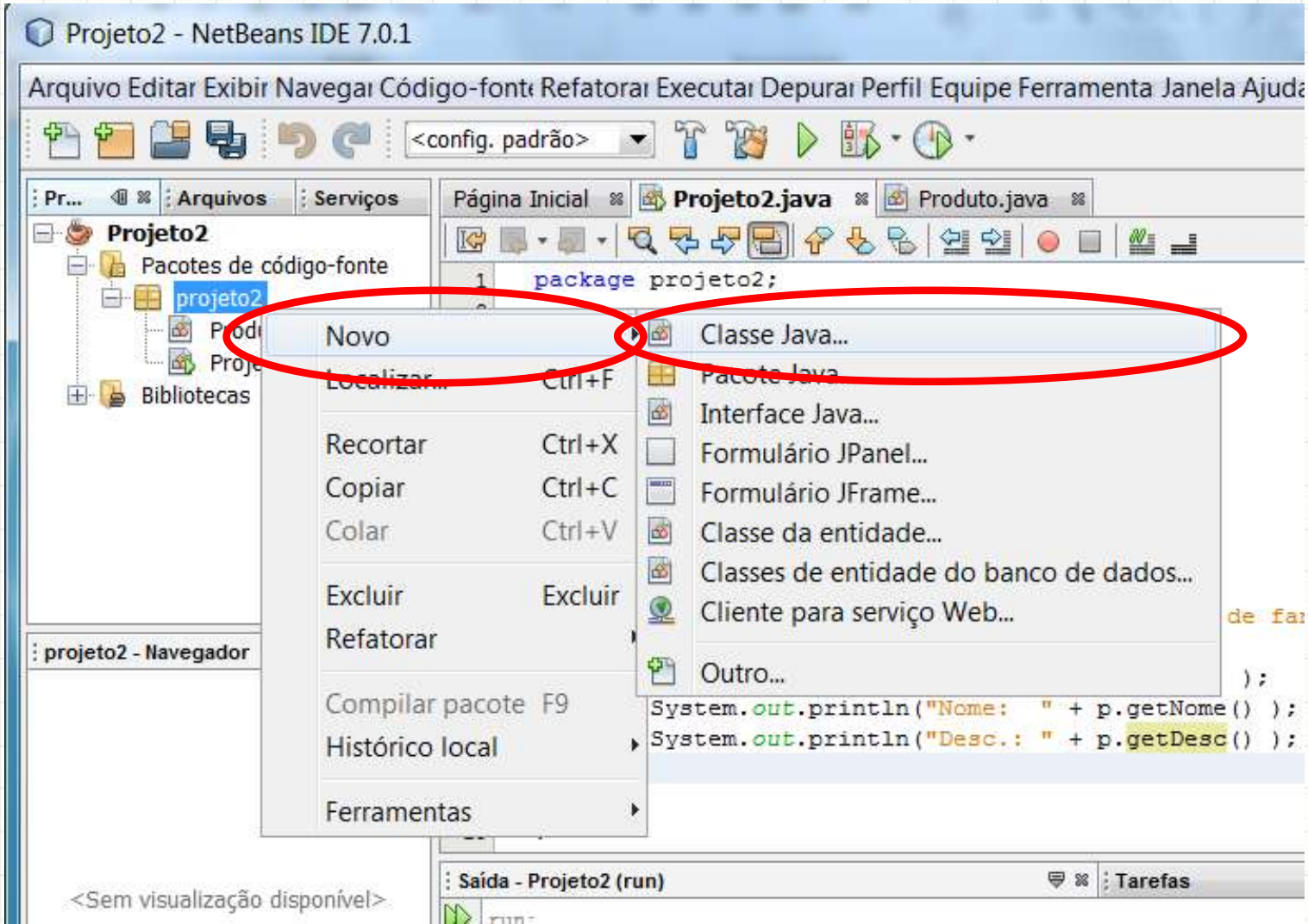
Definindo uma Classe Filha

- Para criar a nova classe, clique com o botão direito no ícone do pacote que tem o nome do projeto (**projeto2**)



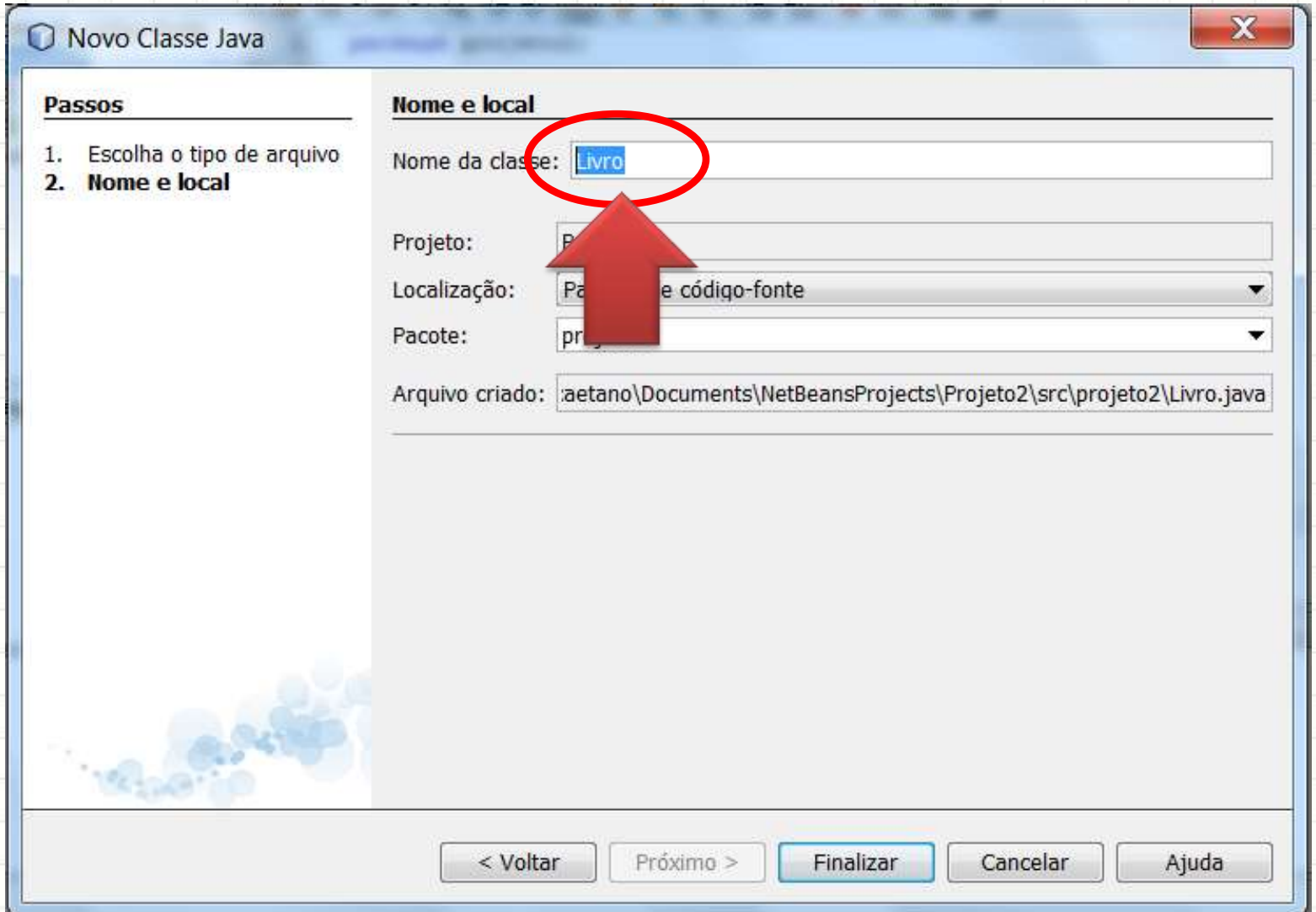
Definindo uma Classe Filha

- Agora selecione **novo** > **classe java**



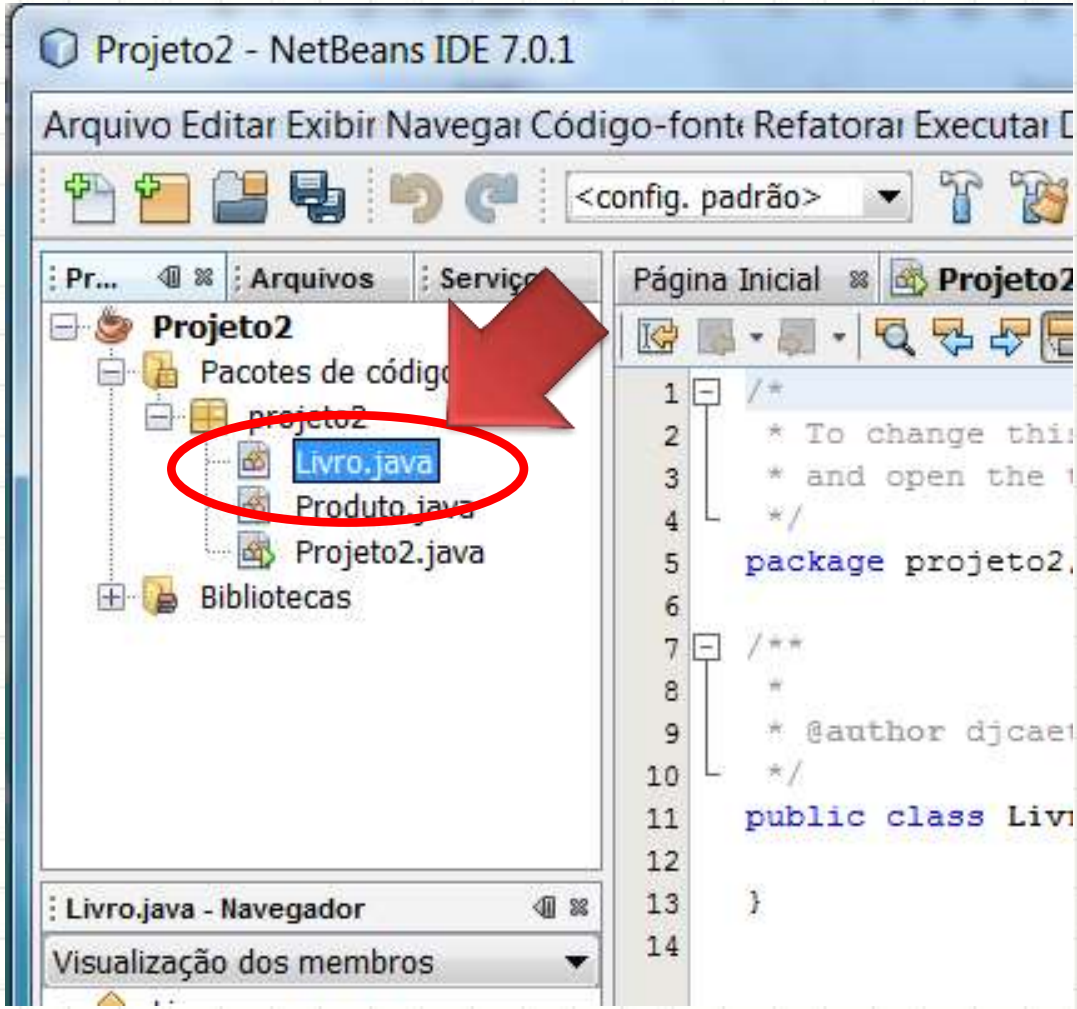
Definindo uma Classe Filha

- Agora dê um nome à classe: **Livro**



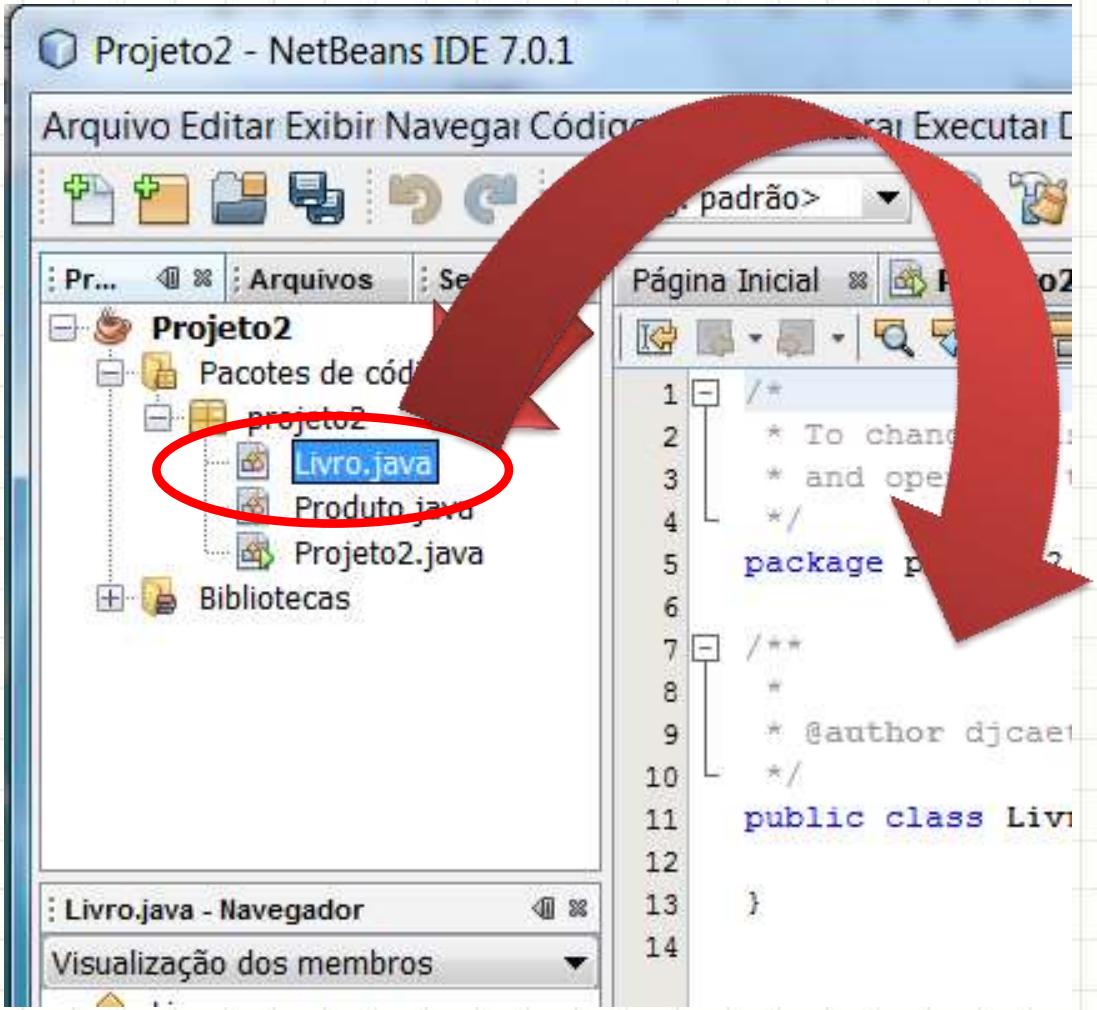
Definindo uma Classe Filha

- Observe a classe na área de projeto



Definindo uma Classe Filha

- Clique 2 vezes nesse ícone para ver o código



Informando quem é a Classe Pai

- Indique que esta classe **extends Produto** :

Página Inicial | Projeto2.java | Produto.java | Livro.java

```
1
2  package projeto2;
3
4  public class Livro extends Produto {
5
6  }
7
```

```
package projeto2;

public class Livro extends Produto {

}
```

Informando quem é a Classe Pai

- Indique que esta classe **extends Produto** :

Página Inicial | Projeto2.java | Produto.java | Livro.java

```
1
2   package projeto2;
3
4   public class Livro extends Produto {
5
6   }
7
```

Com isso já podemos usar a classe Livro como se fosse um Produto... Mas queremos adicionar o atributo **autor** ao Livro, não é?

Acrescentando atributos

- Acrescente o atributo **autor** como String:

Página Inicial | Projeto2.java | Produto.java | **Livro.java**

```
1
2   package projeto2;
3
4   public class Livro extends Produto {
5
6       private String autor;
7
8   }
9
```

Acrescentando atributos

- Acrescente o atributo **autor** como String:

Página Inicial | Projeto2.java | Produto.java | Livro.java

```
1
package projeto2;

public class Livro extends Produto {

    private String autor;

}
```

Criando os *getters* e *setters*

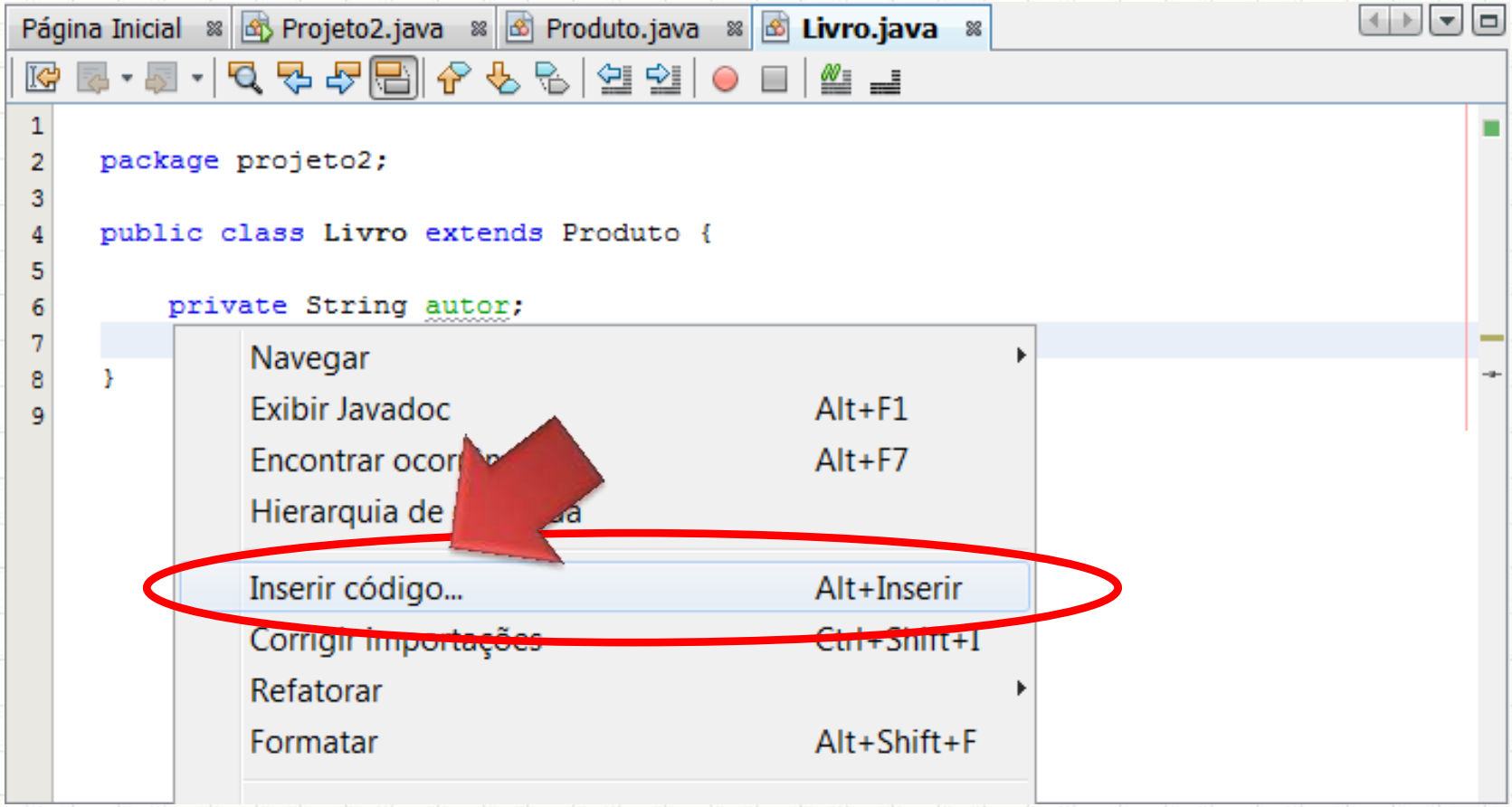
- Clique com o botão direito na área da classe

Página Inicial | Projeto2.java | Produto.java | **Livro.java**

```
1
2   package projeto2;
3
4   public class Livro extends Produto {
5
6       private String autor;
7
8   }
9
```

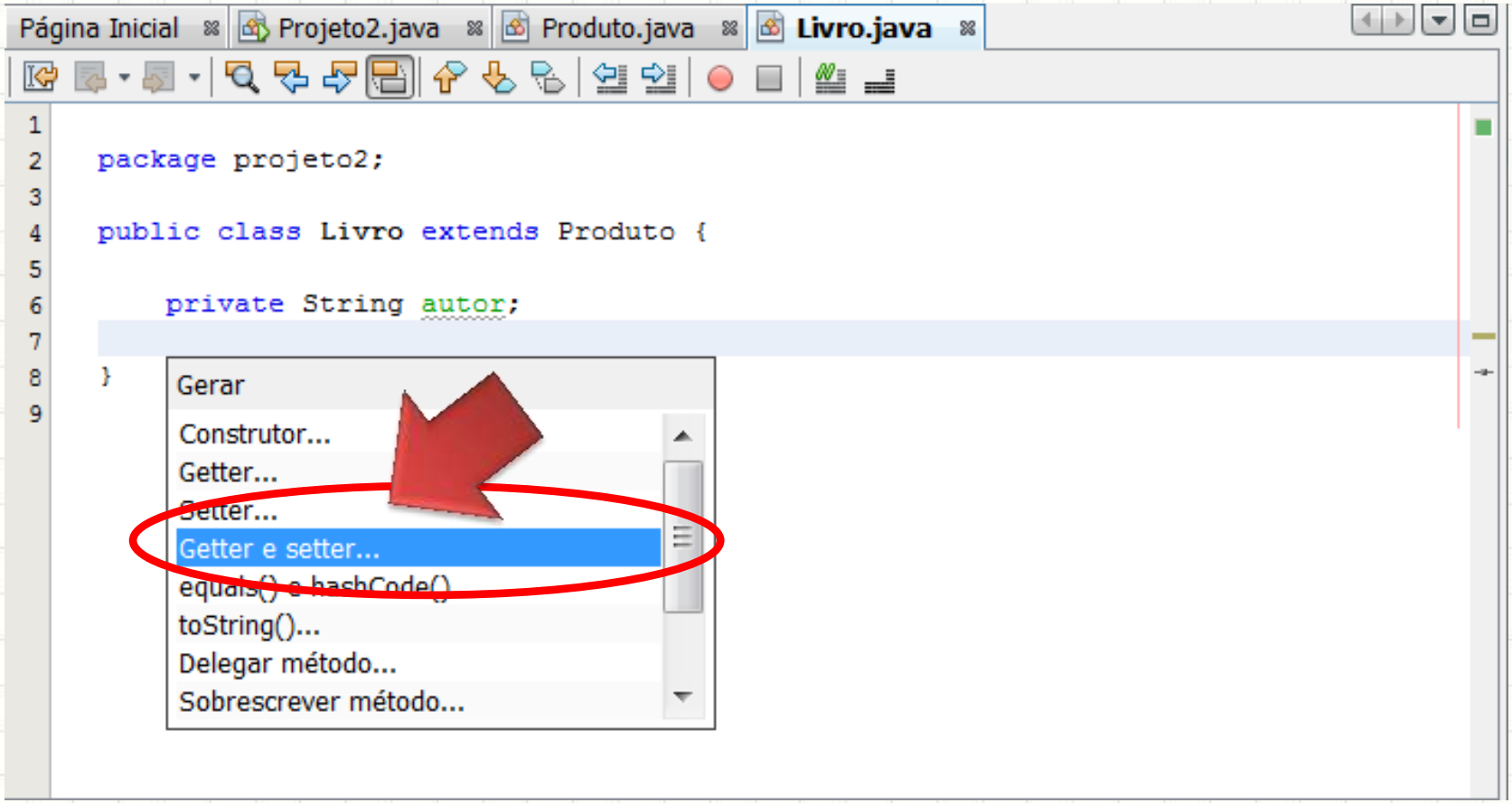
Criando os *getters* e *setters*

- Selecione **Inserir código...**



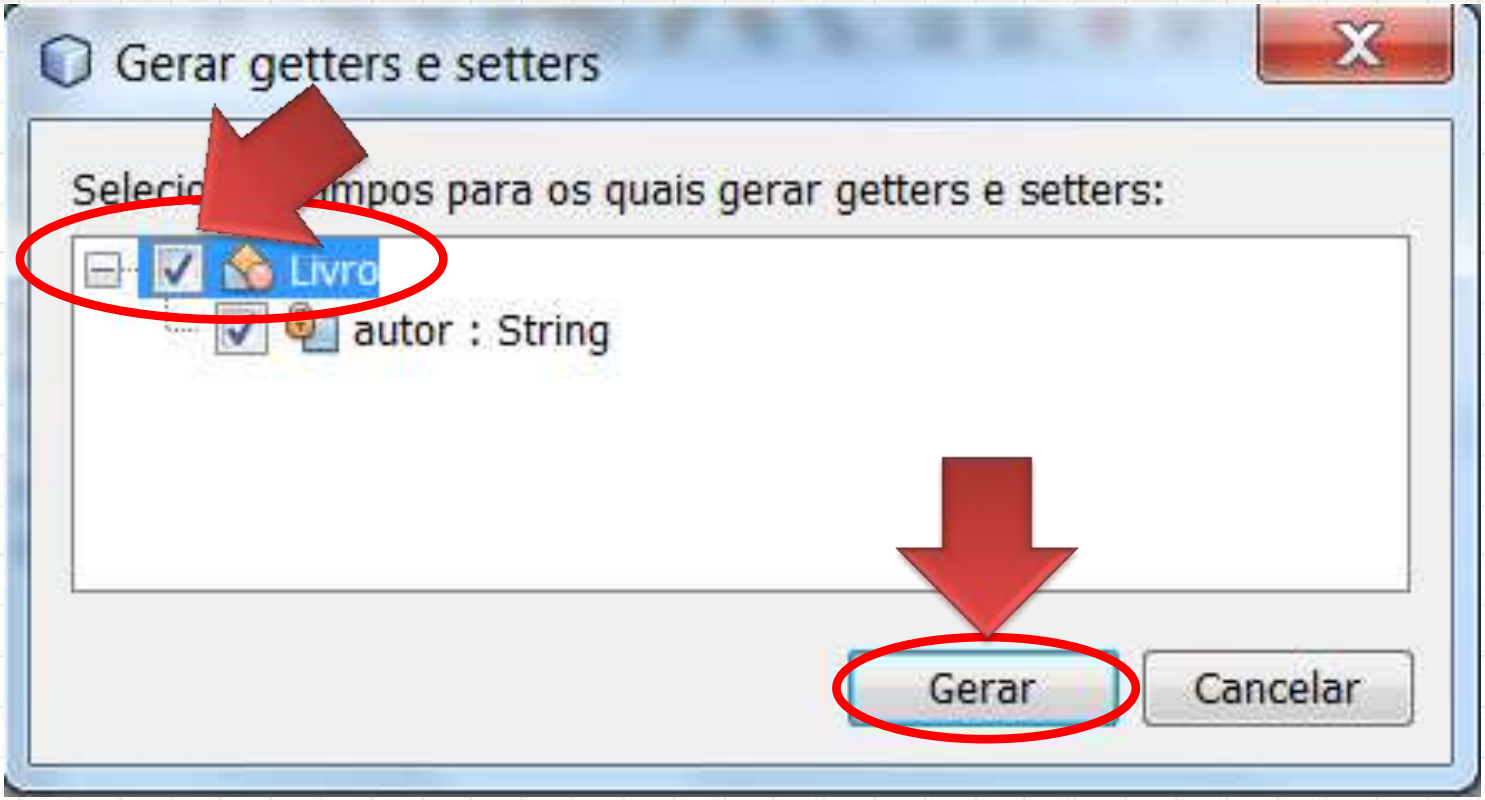
Criando os *getters* e *setters*

- Selecione a opção **Getter e setter...**



Criando os *getters* e *setters*

- E marque os atributos para os quais quer gerar *getters* e *setters*



Criando os *getters* e *setters*

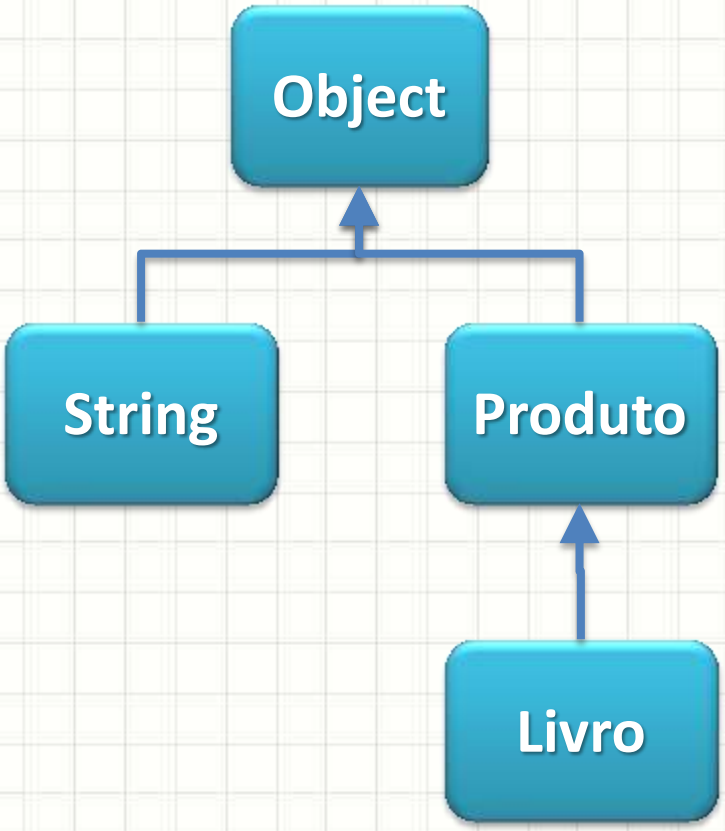
- O código será gerado automaticamente

Página Inicial | Projeto2.java | Produto.java | **Livro.java**

```
1
2   package projeto2;
3
4   public class Livro extends Produto {
5
6       private String autor;
7
8       public String getAutor() {
9           return autor;
10      }
11
12      public void setAutor(String autor) {
13          this.autor = autor;
14      }
15
16  }
17
```

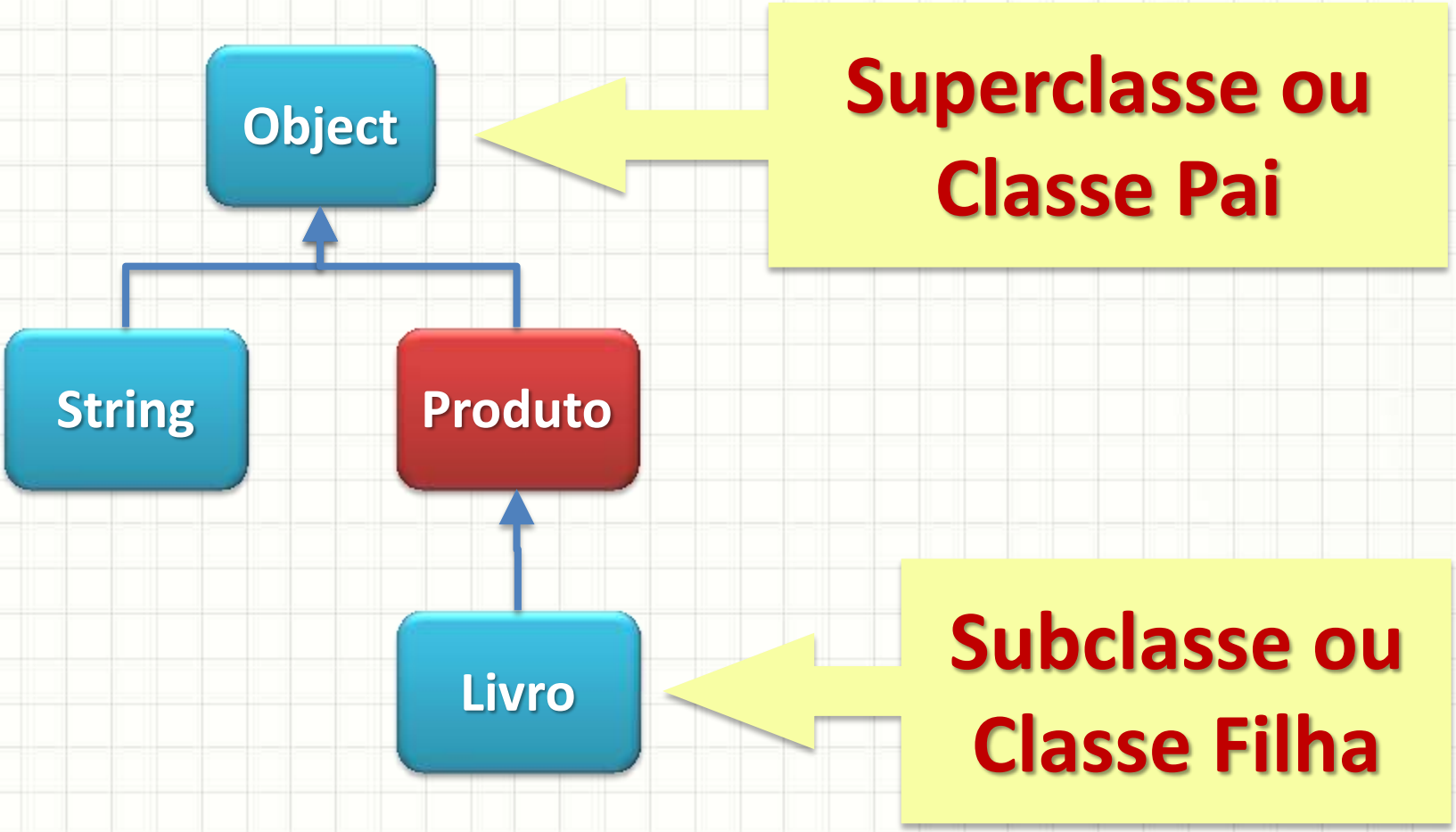
Hierarquia de Classes

- Object é “pai” de todas as classes do Java



Hierarquia de Classes

- Object é “pai” de todas as classes do Java



Herança de Métodos

- Para saber quais métodos estão disponíveis, basta digitar o nome da referência e adicionar um “.”:

```
Produto p;

p.
```

equals(Object o)	boolean
getClass()	Class<?>
getDesc()	String
getId()	int
getNome()	String
hashCode()	int
notify()	void
notifyAll()	void

```
is");
a literatura de fantasia.");
" + p.getId() );
" + p.getNome() );
" + p.getDesc() );
```

Herança de Métodos

- Para saber quais métodos ... eis, bas... da ... nar um ...

Métodos em negrito estão definidos nesta classe...

Métodos Herdados de Object

```
Produ...

p.|
```

```
equals ...ject o)  boolea...
getCla...s()        Clas...>
getDesc()            ...tring
getId()                 int
getNome()            String
hashCode()              int
notify()               void
notifyAll()            void
```

```
...is");
...a literatura de fantasia.");
... + p.getId() );
... + p.getNome() );
... + p.getDesc() );
```

Exemplo/Exercício

- Construa a Hierarquia ao lado:
  - Produto: tem **nome**
  - Livro: acrescenta **autor**

- **Experimente**
  a) Criar um produto
  b) Criar um livro
  c) Criar livro e colocar em uma ref. de produto
  d) Criar produto e colocar em uma ref. de livro

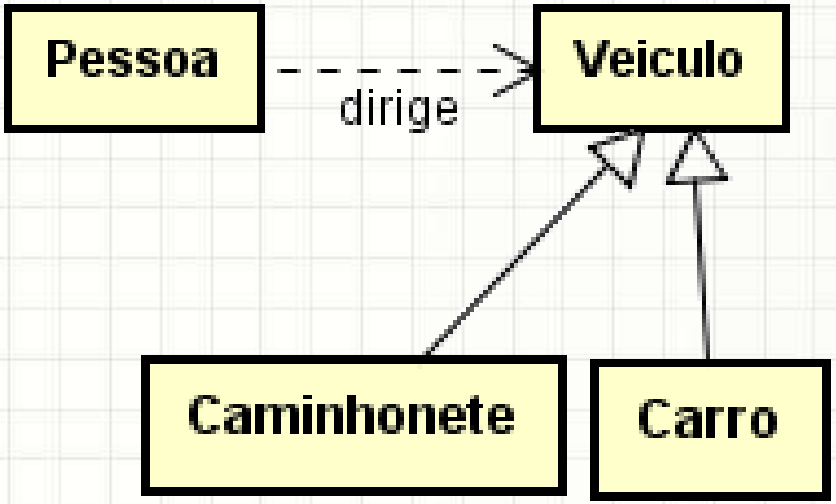
Para que misturar as bolas?

- Foi dito que pode-se guardar objetos de um tipo em uma referência de outro tipo
  - Desde que o tipo da referência seja *superclasse* da classe do objeto...
- Mas... para que serve guardar objetos de tipos diferentes do da referência?
- Pedido → Lista de Produtos **Polimorfismo**
  - Lista guarda qualquer subclasse de “Produto”
- E uma lista para “Object”, guarda o quê?

Polimorfismo e Sobrescrita de Métodos

Polimorfismo

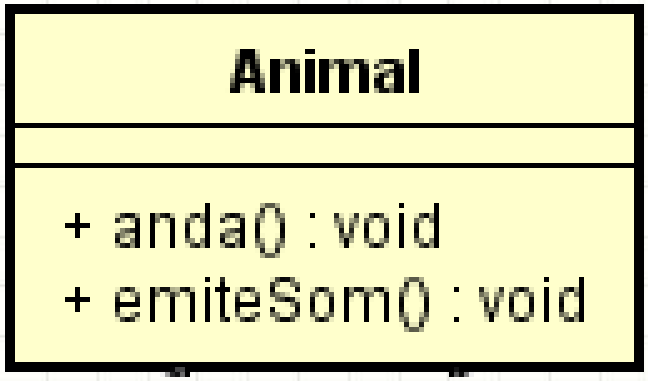
- Pessoa dirige Veículo
- Carro é Veículo
  - Pessoa dirige Carro
- Caminhonete é Veículo
  - Pessoa dirige Caminhonete



- Objeto pessoa não precisa “conhecer” o tipo específico do objeto

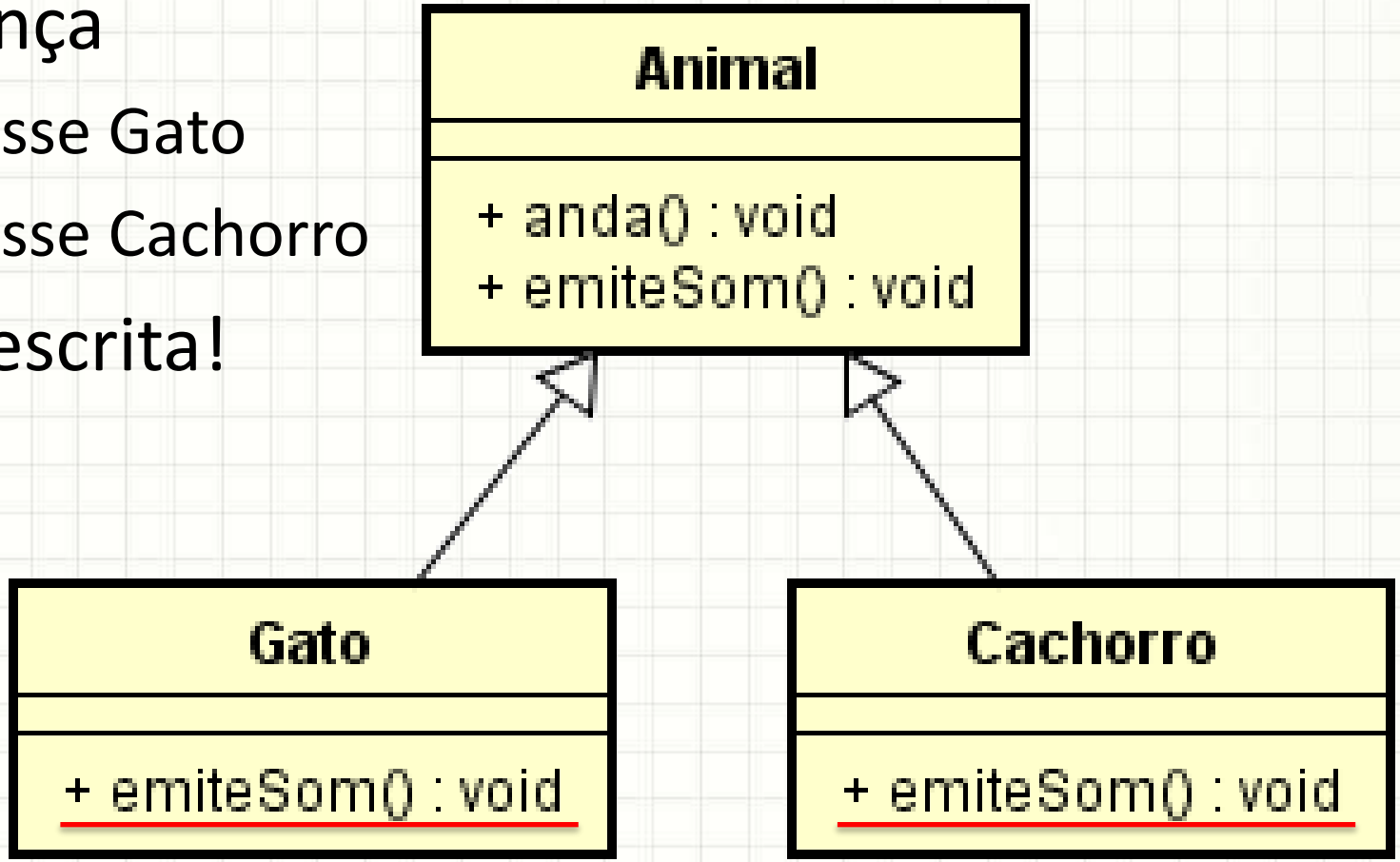
Polimorfismo

- Projeto: **Animais**
- Classe Animal



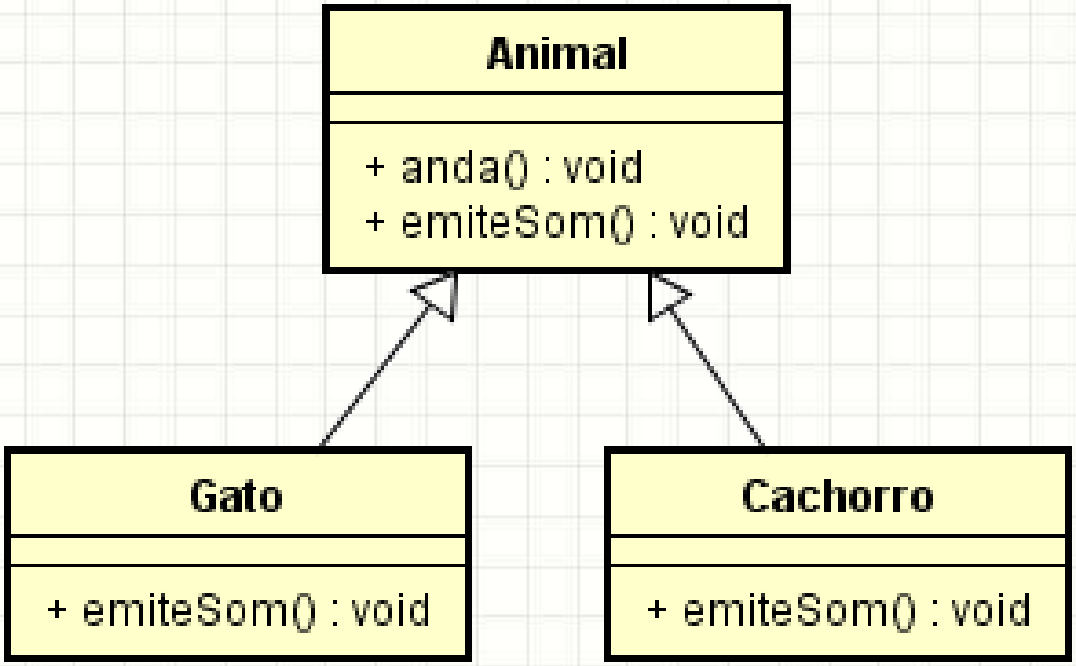
Polimorfismo

- Projeto: **Animais**
- Herança
  - Classe Gato
  - Classe Cachorro
- Sobrescrita!



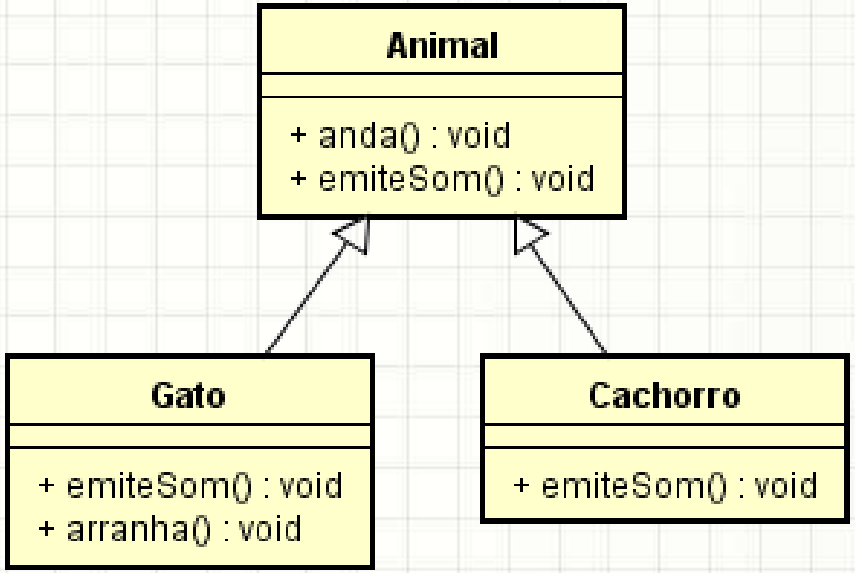
Sobrescrita (Override)

- Classe filha reimplementa...
  - Um método da classe pai
  - Em geral para executar algo diferente



Sobrescrita (Override)

- Como saber quais métodos posso executar?
  - Observe a estrutura:



- O tipo da referência
  - Animal: anda / emiteSom
  - Cachorro: anda / emiteSom
  - Gato: anda / emiteSom / arranha

Sobrescrita (Override)

- **Lembre-se**: para saber quais métodos estão disponíveis, basta digitar o nome da referência e adicionar um “.”:

```
Produto p;

p.
```

equals(Object o)	boolean
getClass()	Class<?>
getDesc()	String
getId()	int
getNome()	String
hashCode()	int
notify()	void
notifyAll()	void

```
is");
a literatura de fantasia.");
" + p.getId() );
" + p.getNome() );
" + p.getDesc() );
```

Sobrescrita (Override)

- **Lembre-se:** para saber o … tão disp… gita … ável e ac…

Métodos em negrito estão definidos nesta classe...

Métodos Herdados

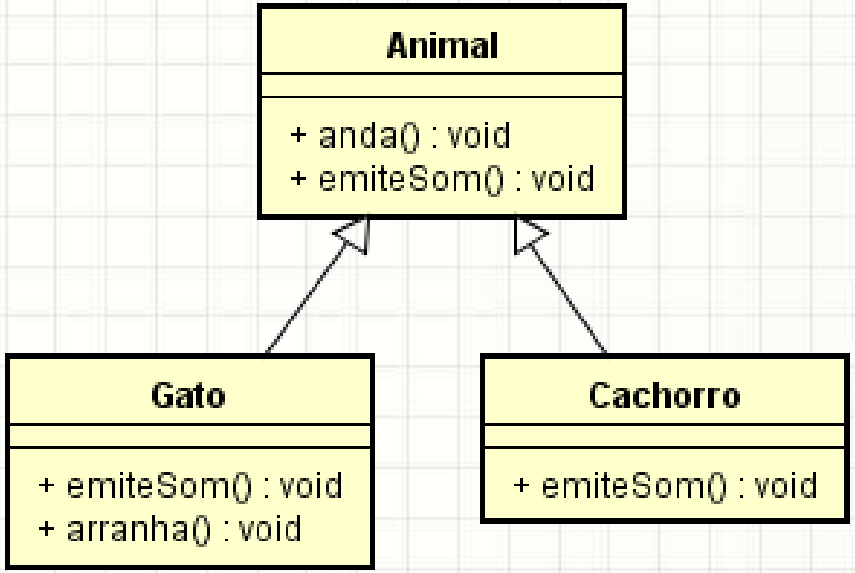
```
Produ…

p.|
```

```
equals…ject o)  boolea…
getCla…s()        Clas…>
getDesc()           …tring
getId()                int
getNome()           String
hashCode()             int
notify()              void
notifyAll()           void
```

```
…is");
…a literatura de fantasia.");
…" + p.getId() );
…" + p.getNome() );
…" + p.getDesc() );
```

Sobrescrita (Override)

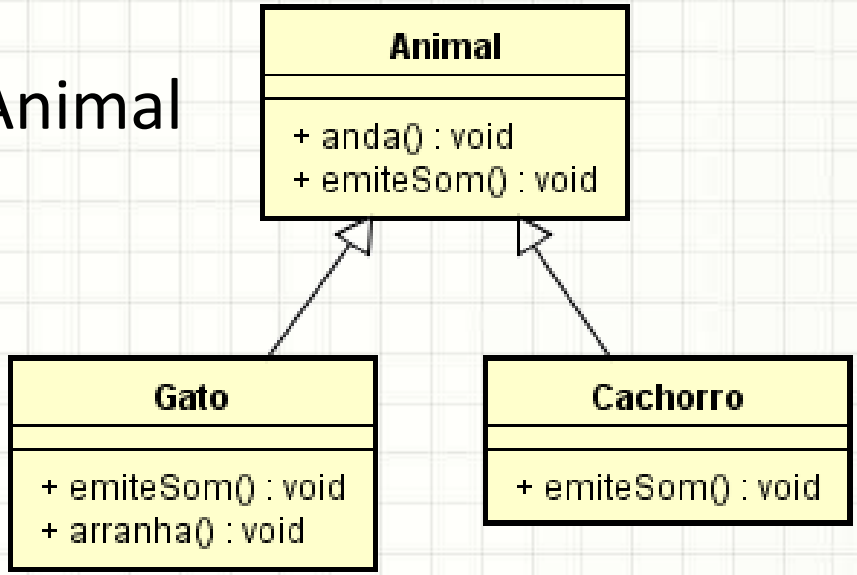
- Como saber qual versão do método será executada?
  - Observe a estrutura:
  - Referência: Animal
    - emiteSom()



- Para um objeto:
  - Animal: emiteSom do Animal
  - Cachorro: emiteSom do Cachorro
  - Gato: emiteSom do Gato

Sobrescrita (Override)

- E se um objeto Cachorro quiser executar um método do objeto pai que foi sobrescrito?
- Objeto Cachorro
  - Acionar emiteSom do Animal
  - Use a palavra **super**
  - **super**.emiteSom();



Tipos de Herança

Tipos de Herança

- Herança Simples x Herança Múltipla
- Herança Simples
  - Pessoa → Funcionario
- Herança Múltipla
  - Radio
  - Relógio

  } Rádio-Relógio

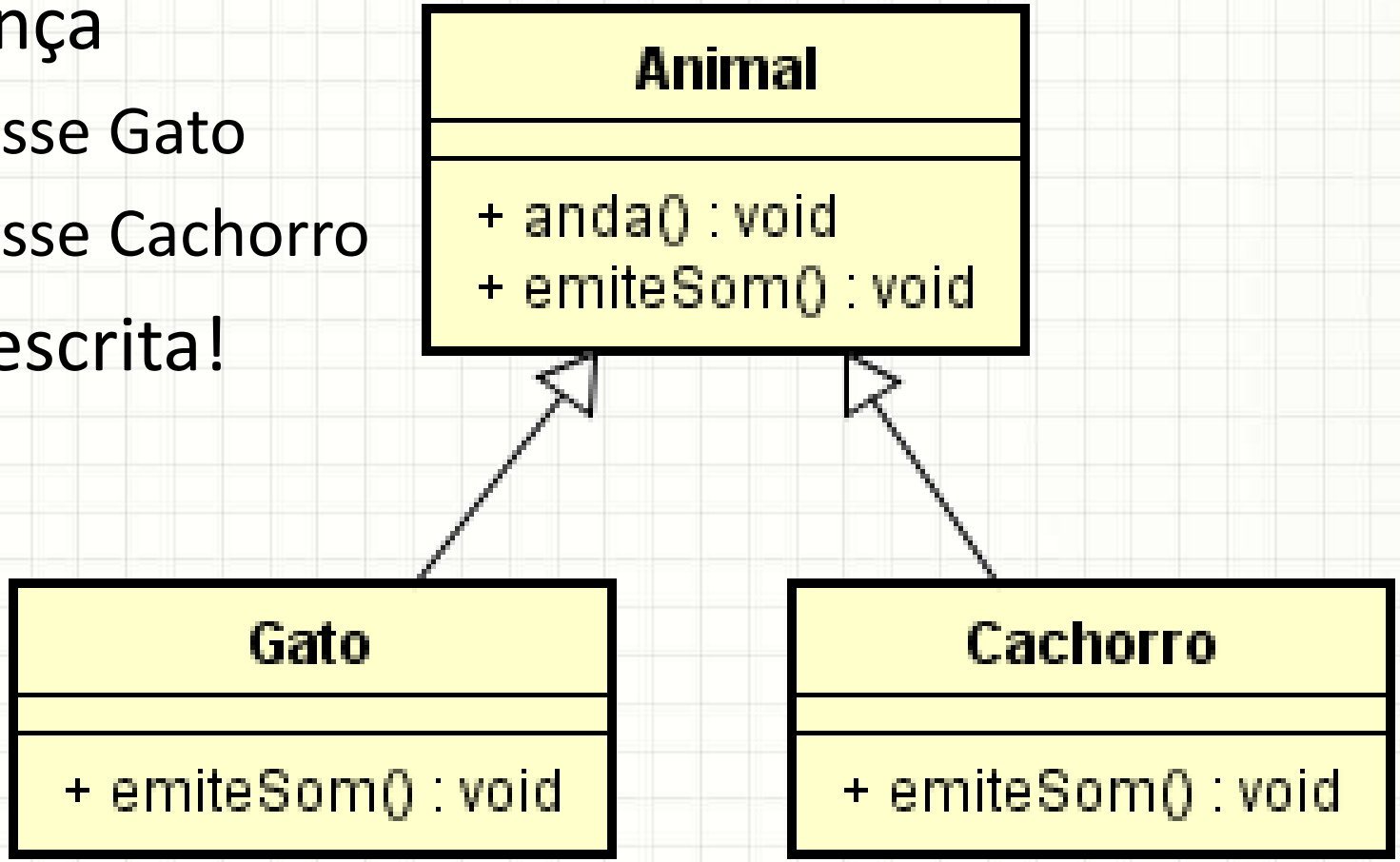
- Java não possui herança múltipla!

PERGUNTAS?

Parte Prática

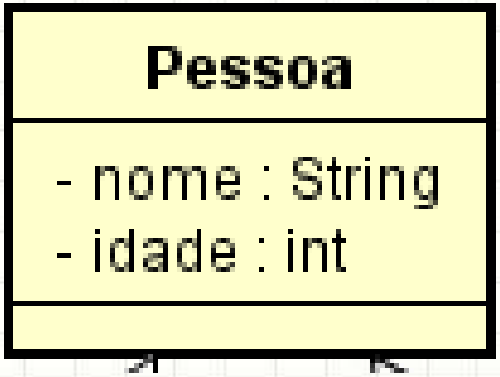
Experimentando

- Projeto: **Animais**
- Herança
  - Classe Gato
  - Classe Cachorro
- Sobrescrita!



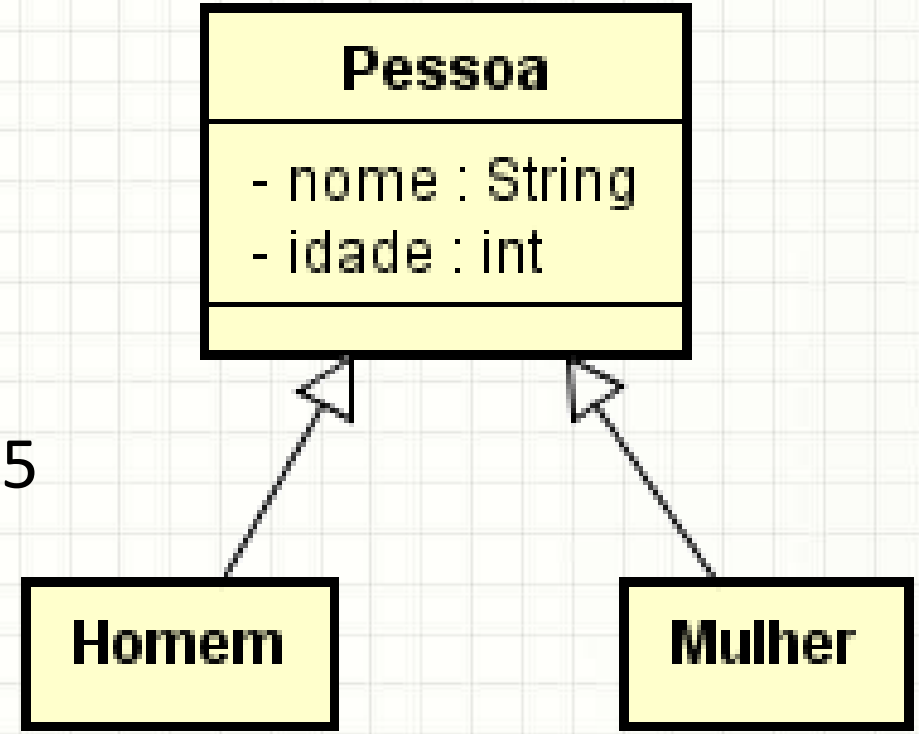
Exercício

- Projeto: **Pessoas**
- Classe Pessoa
  - String nome;
  - int idade;



Exercício

- Projeto: **Pessoas**
- Classe Homem
  - Idade mínima: 18
- Classe Mulher
  - Diz idade máxima: 35



Aplicação no Caso Prático

Avançando no CadProd

- Abra o projeto CadProd.
- Crie uma classe para cada tipo de produto
  - Livro: String autor
  - CD: String banda
  - DVD: Diretor
  - BD: Diretor e se é 3D ou não

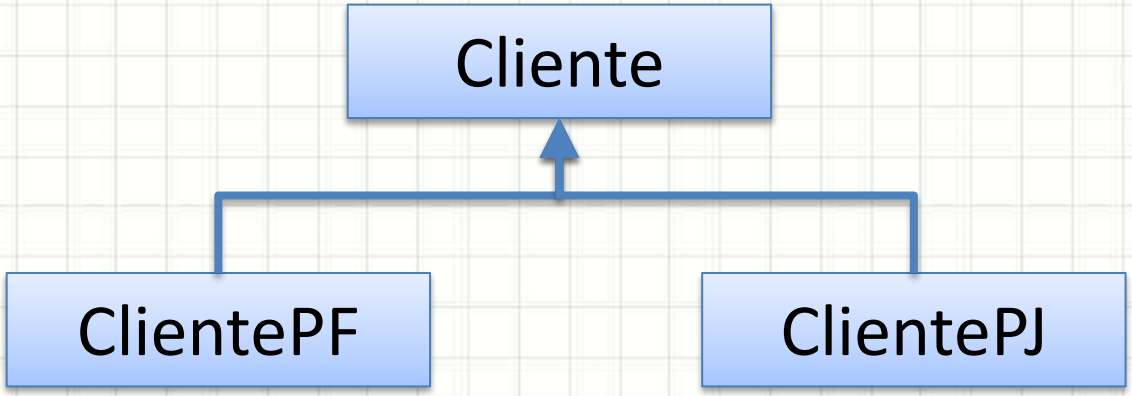
Avançando no CadProd

- Ainda no projeto CadProd
- Altere o programa principal de maneira que o preenchimento dos campos sejam parte dos objetos

PROJETO / ATIVIDADE

Projeto / Atividade

- Objetivo: criar cadastro de clientes simples
- Quem: Duplas especificadas
- A segunda entrega será 05/06
  - Listagem de clientes
  - Busca de clientes
  - Edição de clientes

Projeto / Atividade

- Hoje: Implementar a classe ListaDeClientes
  - Nos mesmos moldes da CadProd
  - Dois tipos de clientes:
    - Cliente Pessoa Física (cpf)
    - Cliente Pessoa Jurídica (cnpj)