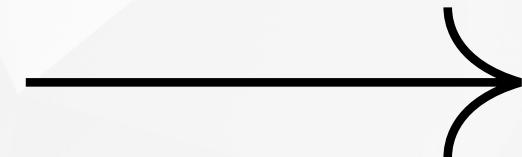


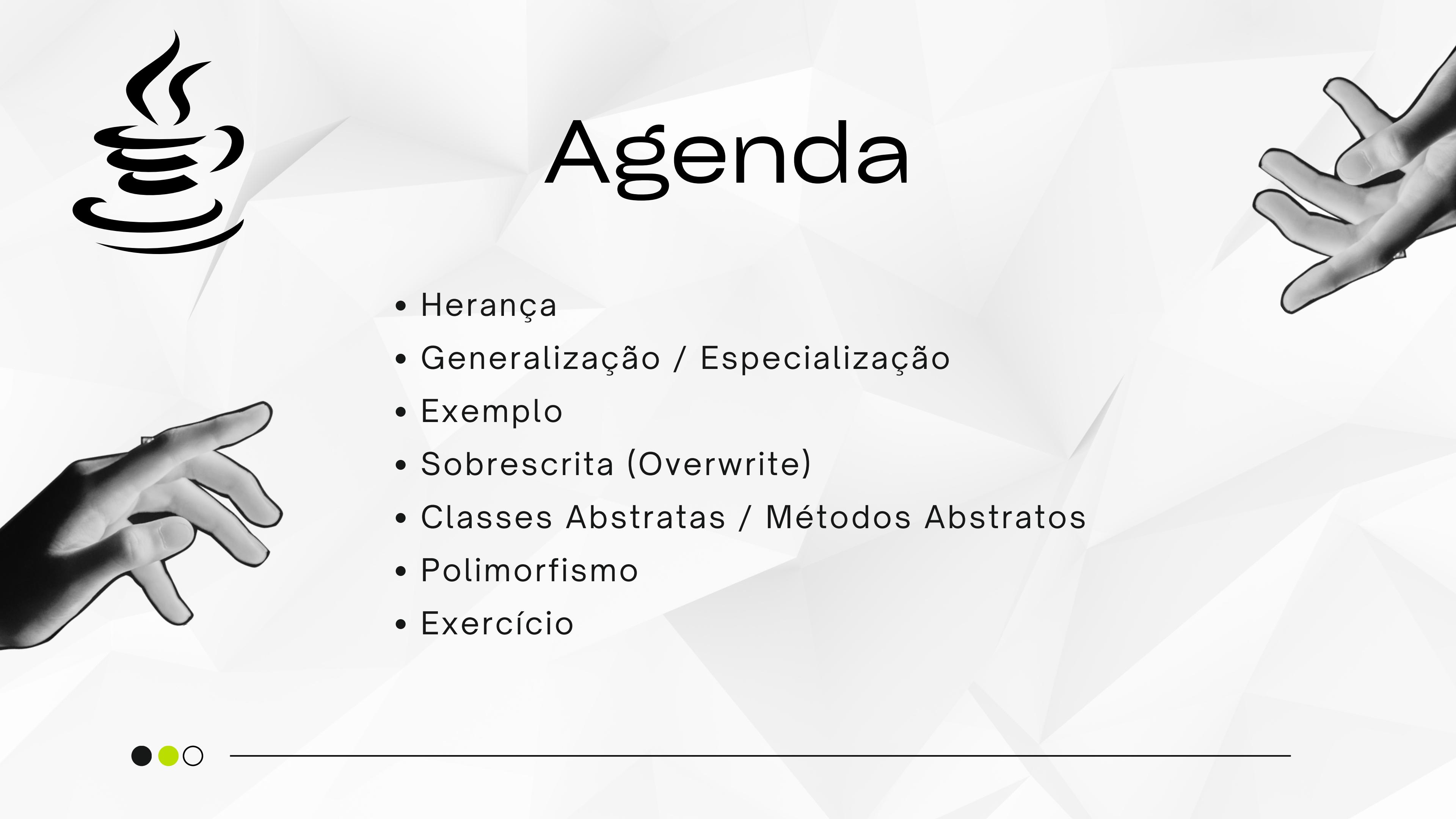
POO

Programação Orientada a Objetos

Material 006

Professor Maromo



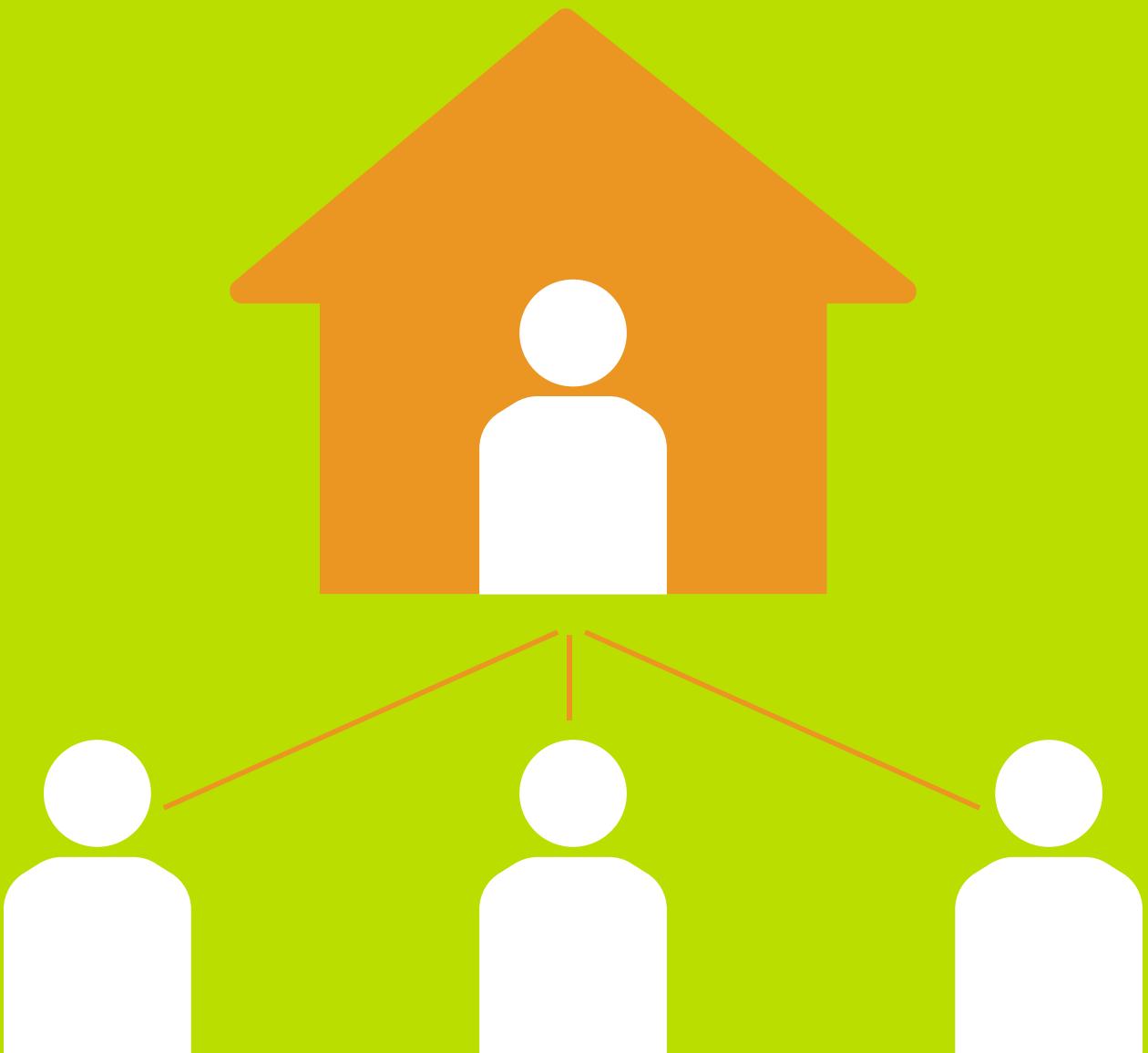


Agenda

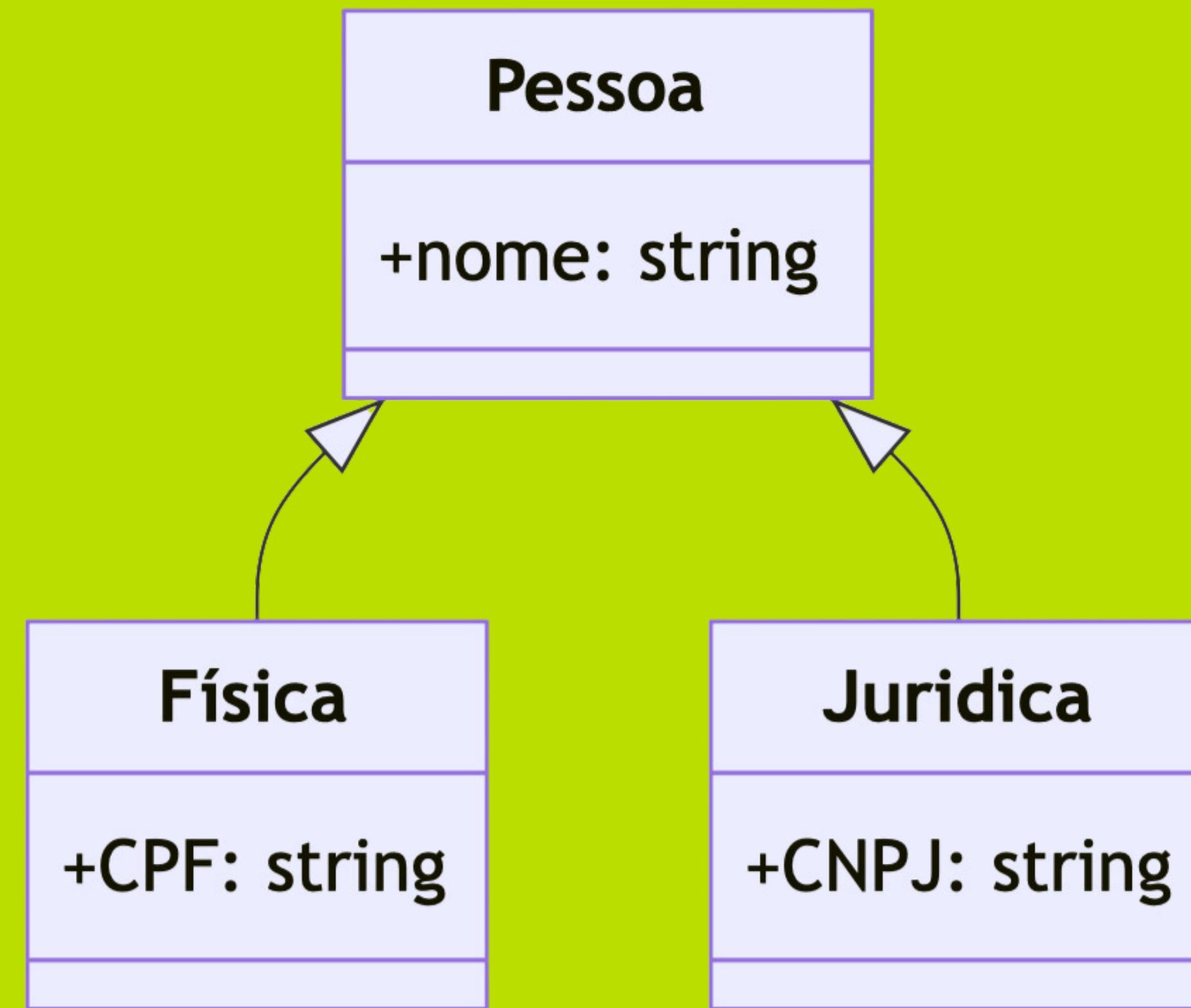
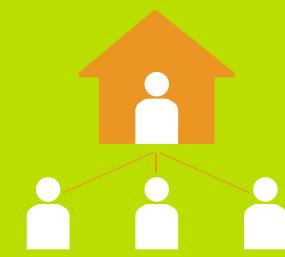
- Herança
- Generalização / Especialização
- Exemplo
- Sobrescrita (Overwrite)
- Classes Abstratas / Métodos Abstratos
- Polimorfismo
- Exercício

Herança

- Princípio da Programação Orientada a Objetos que permite que as classes compartilhem atributos e métodos comuns baseados em um relacionamento.
- A herança possibilita a aplicação de vários conceitos de orientação a objetos que não seriam viáveis sem a organização e estruturação alcançada por sua utilização.



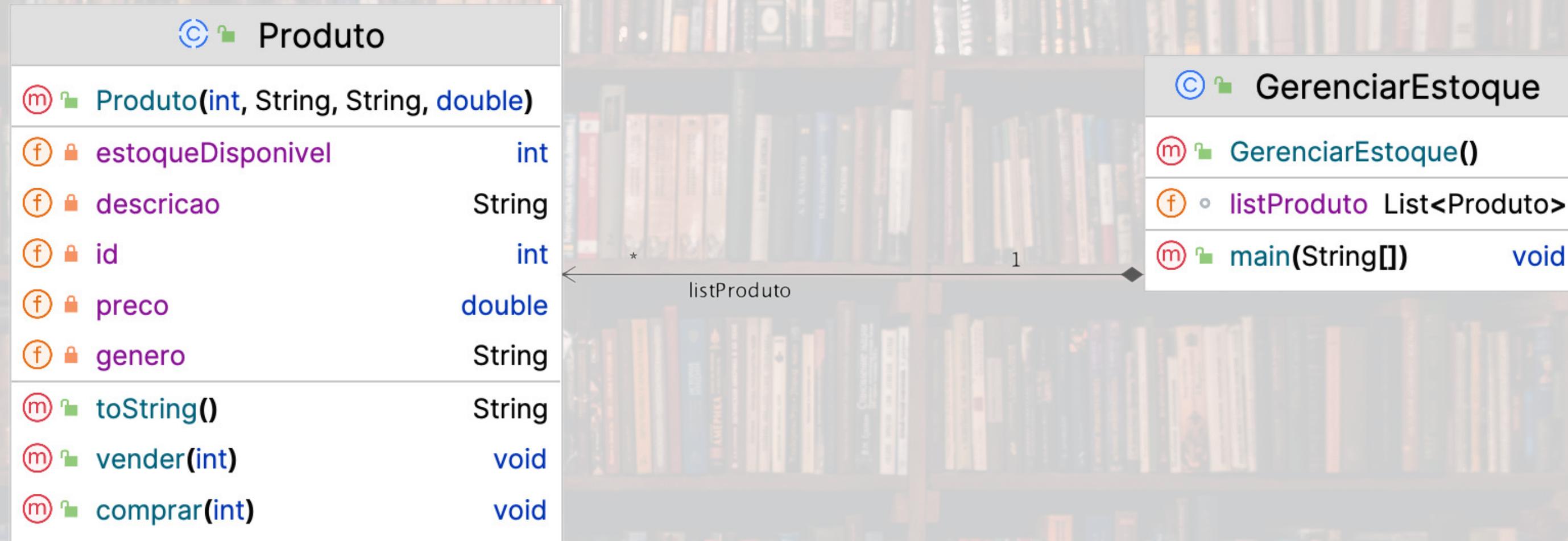
Herança – Representação UML



Generalização / Especialização

- É a 1^a abordagem a ser feita sobre **herança**.
- Possibilita a vantagem mais direta (reutilização de código).
- A **generalização** é o agrupamento de características (atributos) e regras (métodos) comuns em um modelo de sistema.
- A **especialização** é o processo inverso, é a definição das particularidades de cada elemento de um modelo de sistemas, detalhando características e regras específicas de um o objeto.

Projeto: Livraria



© Dvd	
(m) Dvd	(int, String, String, double, String, int, String)
(f) censura	String
(f) diretor	String
(f) duracao	int
(m) toString()	String

© Livro	
(m) Livro	(int, String, String, double, String, String, String)
(f) autor	String
(f) editora	String
(f) edicao	String
(m) toString()	String

extends

- A herança é definida na implementação da subclasse, ou seja, quanto a codificação, a superclasse não apresenta nenhuma diferença.
- O relacionamento de herança é gerado através do comando **extends** que é implementado na assinatura (cabeçalho) da subclasse indicando a sua superclasse.



Considerações sobre os Construtores

- Podemos resumir que a finalidade **dos construtores é inicializar os atributos de um objeto na sua instanciação.**
- Considerando **que uma subclasse herda os atributos (além dos métodos) de sua superclasse**, devemos portanto, **preparar os construtores da subclasse para inicializarem também os atributos herdados.**



Sobrescrita

(Overwrite ou Overriding)

- A sobrescrita ou reescrita de método **está diretamente relacionada com herança e é a possibilidade de manter a mesma assinatura de um método herdado e reescrevê-lo na subclasse.**
- Na chamada de um método sobrescrito Java considera, primeiro, a classe a partir da qual o objeto foi instanciado, se a superclasse possuir um método com a mesma assinatura este será descartado.



Projeto Livraria - sobrescrita

Consideremos o cenário que há três formas distintas de cálculo do Preço de Venda de um produto, a saber:

Produtos:

- Método calcularPrecoVenda()
 - Calcula 10% sobre o preço de custo e armazena no atributo precoVenda ($\text{precoVenda} = \text{precoCusto} * 1.1$). [No momento]

Livro:

- Método sobrescrito calcularPrecoVenda()
 - Calcula 15% sobre o preço de custo e armazena no atributo precoVenda ($\text{precoVenda} = \text{precoCusto} * 1.15$).

Dvd:

- Método sobrescrito calcularPrecoVenda(double cotacaoDolar)
 - Calcula 20% sobre o preço de custo e armazena no atributo precoVenda ($\text{precoVenda} = \text{precoCusto} * 1.20$).



© GerenciarEstoque	
(m) GerenciarEstoque()	
(f) listProduto List<Produto>	
(m) main(String[]) void	

© Produto	
(m) Produto(int, String, String, double)	
(f) estoqueDisponivel int	
(f) descricao String	
(f) id int	
(f) preco double	
(f) genero String	
(m) toString() String	
(m) vender(int) void	
(m) comprar(int) void	
(m) calcularPrecoVenda() void	

© Dvd	
(m) Dvd(int, String, String, double, String, int, String)	
(f) censura String	
(f) diretor String	
(f) duracao int	
(m) toString() String	
(m) calcularPrecoVenda() void	

© Livro	
(m) Livro(int, String, String, double, String, String, String)	
(f) autor String	
(f) edicao String	
(f) editora String	
(m) calcularPrecoVenda() void	
(m) toString() String	



**ABSTRATO OU
CONCRETO**



Classes abstratas e concretas

- Uma **classe abstrata** é desenvolvida para representar entidades e conceitos abstratos.
- A classe abstrata é sempre uma superclasse que **não possui instâncias**.
- Ela define um modelo para uma funcionalidade e fornece uma implementação incompleta (a parte genérica dessa funcionalidade) que é compartilhada por um grupo de classes derivadas (subclasses).
- Cada uma das classes derivadas completa a funcionalidade da classe abstrata adicionando um comportamento específico.
- Uma **classe concreta** é aquela que pode ser instanciada, ou seja, na prática manipulamos um objeto do seu tipo.

Métodos abstratos

- Uma **classe abstrata pode conter métodos concretos**, porém, um **método abstrato só pode ser definido em uma classe abstrata**.
- Esses métodos são implementados nas suas subclasses (concretas) com o objetivo de definir um comportamento (regras) específico.
- Um método abstrato define apenas a assinatura do método e, portanto, não contém código.
- Um método concreto, por sua vez, possui comportamento definido (código escrito).

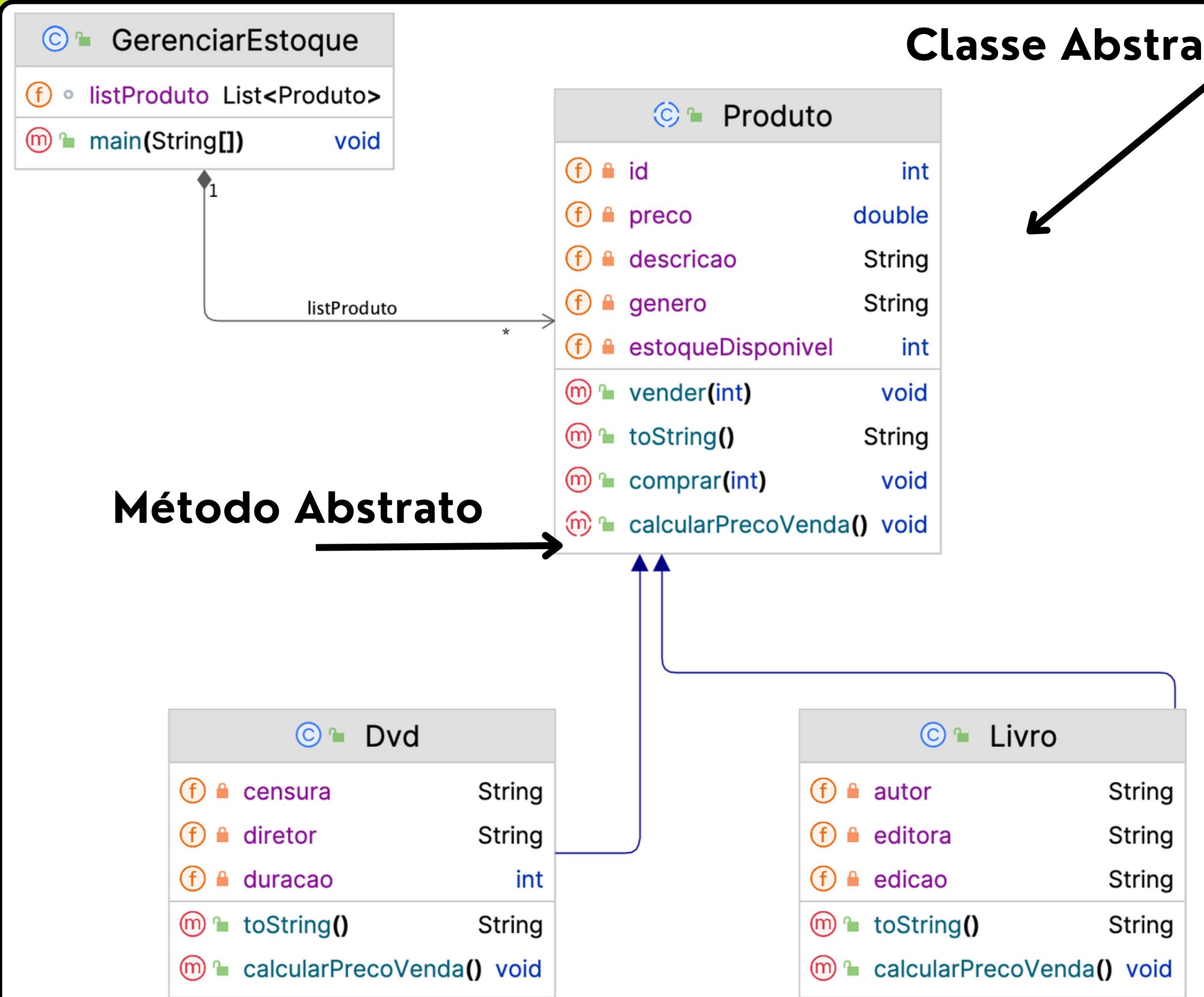
Métodos abstratos

- No exemplo da livraria, a classe **Produto** nunca será utilizada para **instanciar um objeto** porque os produtos efetivamente comercializados são **livros** e **dvds**.
- A **finalidade da classe Produtos é somente a generalização dos produtos**, portanto, conceitualmente ela deve ser definida como uma classe abstrata.
- Partindo desse princípio, o método **calcularPrecoVenda** também é um forte candidato a ser um método abstrato, porque, nesse exemplo, cada produto tem sua própria regra de cálculo.

Livraria



Classe Abstrata



Polimorfismo



- O termo **polimorfismo** é originário do grego e significa "muitas formas" (**poli** = muitas, **morphos** = formas).
- O polimorfismo permite que objetos de **diferentes subclasses** sejam tratados como objetos de uma **única superclasse**.
- É a possibilidade de manipular um objeto como sendo outro.
- O polimorfismo não quer dizer que o objeto se transforma em outro. Um objeto sempre será do tipo que foi instanciado o que pode mudar é a maneira como nos referimos a ele.

Polimorfismo



- Voltamos ao nosso projeto Livraria.
- Os produtos efetivamente comercializados, e que devem ser gerenciados, são **livro e dvd**.
- Contudo, a afirmação: “Livro e dvd são produtos” está correta.
- O conceito de polimorfismo **possibilita que tratemos um livro ou dvd como um produto** (pois, afinal de contas, eles são produtos), mas sem perdermos as suas características específicas porque apesar de serem produtos eles não deixam de ser um livro ou um dvd (especializações).



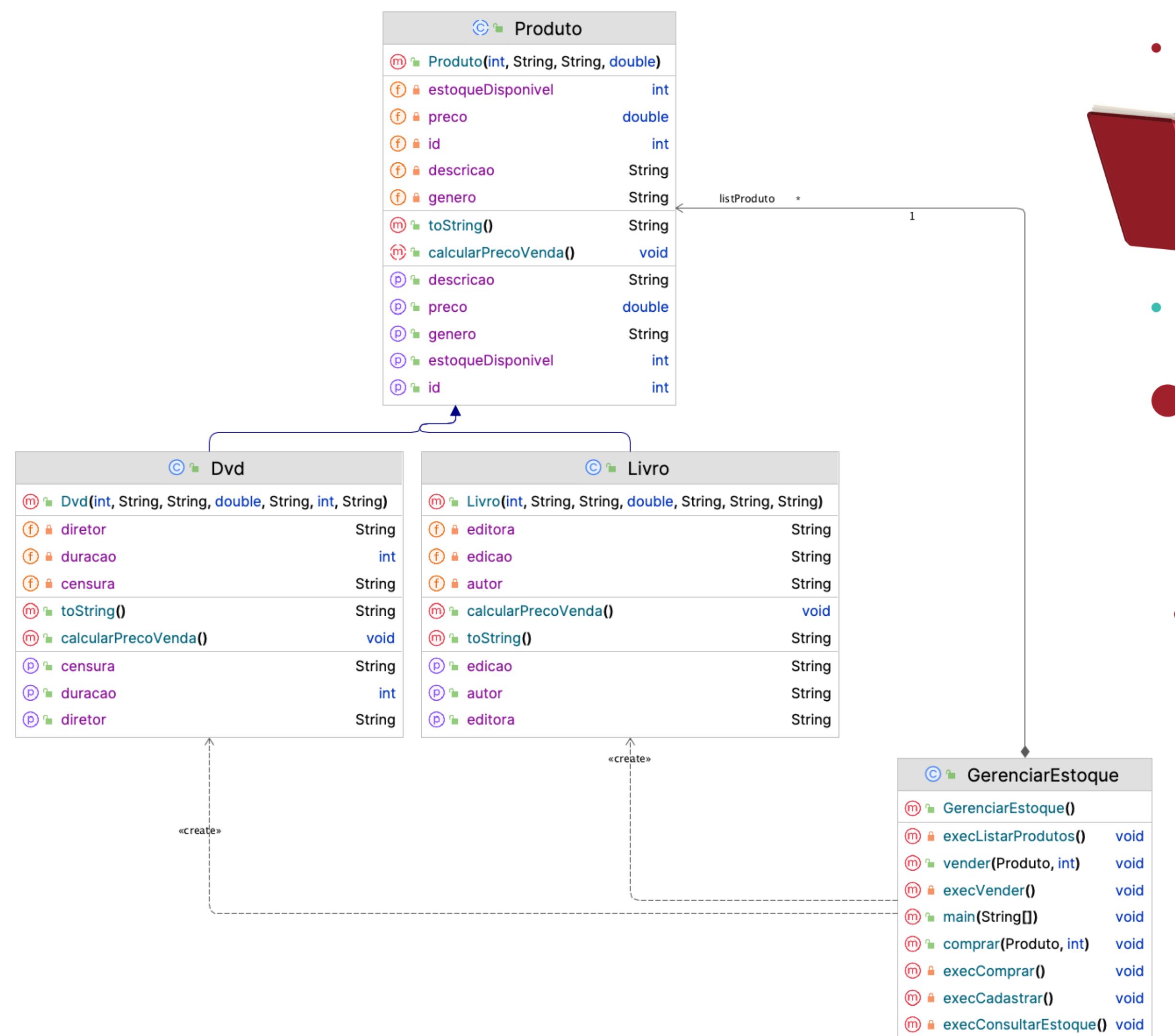
Nota sobre Polimorfismo

- Podemos definir métodos que reconheçam objetos através de suas superclasses mas que mantenham seus atributos e métodos implementados nas subclasses de origem.
- Nota:
 - Esse tipo de polimorfismo apresentado é o que conhecemos por polimorfismo utilizando-se da hierarquia de classes, em aula futura veremos o conceito sem a utilização de hierarquia, com o uso de Interface.

GerenciadorEstoque

©	GerenciarEstoque
m	GerenciarEstoque()
f	listProduto List<Produto>
m	vender(Produto) void
m	comprar(Produto) void
m	main(String[]) void

Os métodos **vender** e **comprar** respectivamente receber um objeto do tipo produto, como a classe Produto está definida como abstract, serão passado a ela as suas derivadas, ou seja, livro e dvd. Os métodos efetivam a venda ou compra independente do tipo de produto passado, desde que seja dos tipos herdados.



Modelagem Final

Como material de estudo, baixe o código disponível em:

Bônus

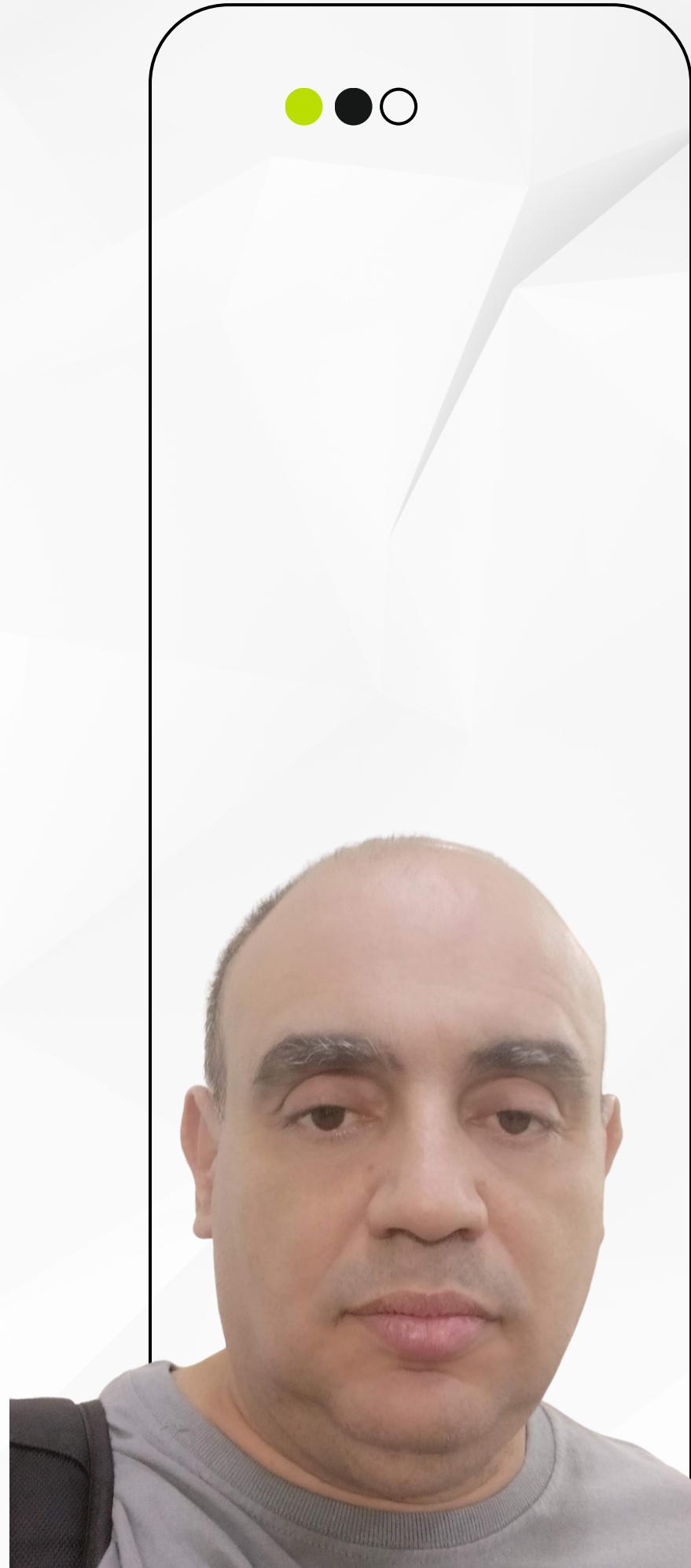
**maromo71/
projetolivraria**

Aula POO Herança

1 Contributor 0 Issues 0 Stars 0 Forks



maromo71/projetolivraria: Aula POO Herança
Aula POO Herança. Contribute to maromo71/projetolivraria development by creating an account on GitHub.
 GitHub



Obrigado

fim



Até a próxima aula



Referências Bibliográficas



- Mendes; **Java com Ênfase em Orientação a Objetos**, Novatec.
- Deitel; **Java, como programar** – 10º edição. Java SE 7 e 8
- Arnold, Gosling, Holmes; **A linguagem de programação Java** – 4º edição.
- Apostilas da Caelum.