Índice

[P.O.O Avançado 2](#_tdw6y3kiw6o)

[Herança 2](#_fwop92vddjjw)

[Comandos 3](#_lrludye5s1wr)

[Links e afins 4](#_v49ndfn8hwjl)

[Dúvidas 4](#_huow9eiz5of6)

[Temporários 5](#_6y3h84mt7ums)

# 

**Assuntos: JAVA Programação orientada a objetos avançada, Equilíbrio vida pessoal e profissional, Comunicação no ambiente de trabalho.**

**Abertura Alunos:**

Reflexão diária: Psicologia positiva

Eloisa Marin e Fernanda Sasso.



# [P.O.O Avançado](https://docs.google.com/document/u/0/d/1byRiY8KIIzpSHjUCLV2IdJPUmYC1PTYZz5wX7D8cASA/edit)

(Clique no título para ser direcionado as anotações de P.O.O)

## Herança

Objetos auxiliam a caracterizar classes.

Herança é uma relação entre uma superclasse e uma subclasse. O que ocorre é que a superclasse tem suas devidas características, e a subclasse recebe essas características podendo ter as suas próprias individuais também, sem influenciar outras subclasses.

**Exemplo:**

| **Subclasse aluno** | **Superclasse pessoa** | **Subclasse professor** |
| --- | --- | --- |
| **nome** | **⬅ nome ➡** | **nome** |
| **idade** | **⬅ idade ➡** | **idade** |
| **genero** | **⬅ gênero ➡** | **genero** |
| r.a | **\_\_** | titulo |
| curso | **\_\_** | materia |
| anoGraduacao | **\_\_** | salario |

Nesse exemplo, ambos aluno e professor vão ter as características da superclasse pessoa, mas, ambos tem também suas características individuais, sem influenciar uns aos outros.

**Herança simples:** Capacidade de uma subclasse de herdar características de uma superclasse.

**Herança múltipla:** Capacidade de uma subclasse de herdar características de múltiplas superclasses.

**(Java NÃO implementa herança múltipla, para isso se usa Interface)**

Cada subclasse pode ser uma candidata a ser superclasse de uma nova subclasse posteriormente.

Em um código, a palavra **super** se refere a uma superclasse.

* Indica a chamada ao construtor da superclasse
* Se a **superclasse** não tiver o construtor padrão, e a **subclasse** não referenciar o construtor da **superclasse** explicitamente, dará erro.

Em um relacionamento de herança:

* É necessário apenas indicar as diferenças entre superclasse e subclasse, o reuso acontecerá automaticamente.
* Um dos primeiros motivos para se usar herança é a necessidade de redefinir métodos.

Ao referenciar uma superclasse, ao criar o construtor, você pode simplesmente gerar o construtor pelo source, e ele gera os atributos da superclasse na subclasse.

### Comandos

**modificador class *nomeSuperclasse* {**

***tudo que você quiser que exista na superclasse***

**}**

Código para se criar uma superclasse

**modificador class *nomeSubclasse* extends *nomeSuperclasse* {**

***tudo que deseja ter na subclasse***

**}**

Código para se criar uma subclasse

**extends** Código que indica que a classe atual é subclasse de alguma classe.

**@Override** Código para indicar que terá uma substituição de um código herdado nesta classe.

## Polimorfismo

Polimorfismo é a habilidade de diferentes classes responderem a um mesmo método de diferentes formas.

Isso dá autonomia à própria classe de retornar funções de acordo com as suas funcionalidades.

Isso significa que um mesmo comando faz coisas diferentes para cada objeto, de acordo com sua necessidade.

Para isso, você cria métodos com o MESMO nome, mas com atributos e comandos diferentes, e ao chamar o método em outra classe, você poderá escolher qual versão usar.

**Exemplo:**

**public tipo *nomeMétodo* (tipo *nomeVariavel*) {**

**Comando que você deseja**

**}**

**public tipo *nomeMétodo* (tipo *nomeVariavel*, tipo *nomeVariavel2*) {**

**Comando que você deseja**

**}**

Nesse exemplo, tipo se refere a um tipo de variável (String, float, int, boolean, etc).

Também, nomeMetodo e nomeVariavel se referem ao nome que você deseja dar ao seu método e variável respectivamente.

Perceba que no método 2 é inserido uma variável a mais, pois apesar de terem o mesmo nome, os métodos têm comandos e necessidades diferentes.

Quando você for chamar o método, preste atenção em verificar qual deles você está chamando, de acordo com as variáveis que você determinou.

## Interface

O Java não permite herança dupla. Ou seja, ele não permite que uma classe seja derivada de mais de uma classe.

Para contornar isso, usamos o interface.

No interface, um objeto implementa as características de outro objeto.

Ele representa uma classe sem atributos de entrada de dados.

Recomenda-se o uso de no máximo 3 interfaces.

Faz-se o uso de métodos apenas assinados, ou seja, sem o corpo. Estes devem ser obrigatoriamente implementados nas subclasses que os utilizam.

**Regras**

1. Uma interface não pode ser instanciada diretamente. Mas podemos criar uma referência a uma interface, que pode apontar para um objeto de qualquer um de seus tipos derivados, implementando-o.
2. Uma interface não pode ser declarada com a palavra-chave final.
3. Não se pode ter variáveis de instância. Se declararmos uma variável em uma interface, ela deve ser inicializada no momento da declaração.
4. Uma classe que implementa uma interface deve fornecer suas próprias implementações de todos os métodos definidos da interface.
5. Quando implementamos um método de interface, ele deve ser declarado como público. Não se pode reduzir a visibilidade de um método de interface durante a substituição.
6. Também pode ser declarado com corpo vazio.
7. Uma interface de nível superior pode ser pública ou padrão, com o modificador abstract em sua definição. Portanto, uma interface declarada com private, protected ou final gerará um erro.
8. Todos os métodos não padrão definidos em uma interface são abstratos e públicos por padrão. Portanto, um método definido com private, protected, ou final em uma interface irá gerar um erro.
9. Se você adicionar qualquer novo método na interface, todas as classes concretas que implementam essa interface deem fornecer implementações para o método recém-adicionado, porque todos os métodos na interface são abstratos por padrão.

## Classes abstratas

Classes que não podem ser instanciadas, mas podem ser derivadas. Geralmente utilizadas somente para derivação de novas classes.

Identificamos uma classe abstrata pelo modificador **abstract** na sua assinatura.

Abstract é um modificador sem acesso em java, que é aplicável para classes, interfaces e métodos.

Classe abstrata é uma classe incompleta, que depende de subclasses para sua implementação. Ou seja, é essencial a criação de ao menos uma subclasse.

# Links e afins

**Cookbook MD11, P.O.O parte 02, Herança e Polimorfismo:**

<https://github.com/conteudoGeneration/cookbook_java_fullstack/blob/main/01_java/11.md>

# 

# Dúvidas

# Temporários







