**Escola e Faculdade Senai “Roberto Mange”**

**PBE– 1DS-TB**

**HELOISA MILITÃO DE SOUZA**

💻

**Programação Back-End:**

**Pesquisa – Herança e Polimorfismo**

**Dorival Alves da Silva Júnior**

**CAMPINAS**

**20 de maio de 2025**

**PROGRAMAÇÃO ORIENTADA AO OBJETOS**

A POO (Programação Orientada a Objetos) é um “molde” de programação que se baseia na utilização de objetos para representar elementos no mundo real e seus procedimentos. Esses objetos são modelados como indivíduos autônomos, pois possuem características únicas e comportamentos independentes. Ou seja, cada objeto é uma entidade única, com dados (atributos) e ações (métodos) específicos para ele.

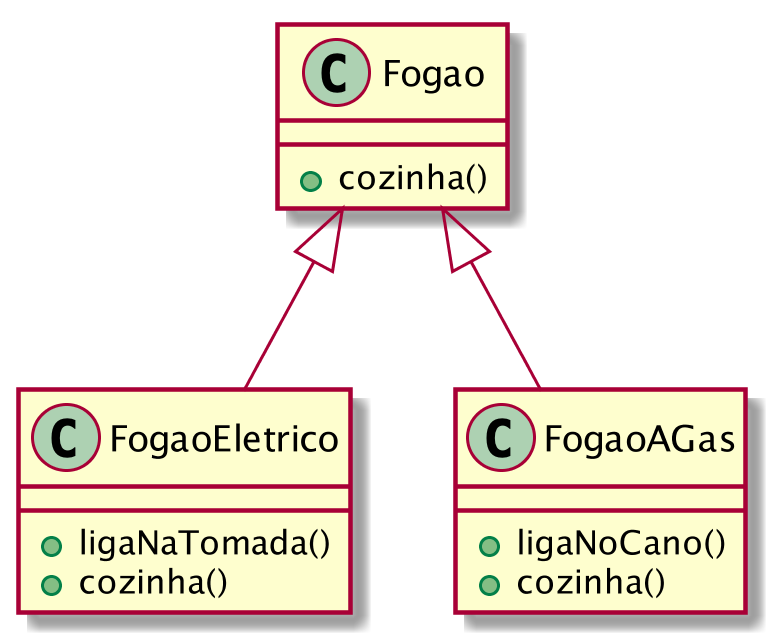
A POO é uma abordagem mais eficiente e intuitiva para modelagem de sistemas complexos. Antes dela, a programação era feita principalmente através de procedimentos, o que tornava o desenvolvimento de sistemas grandes e complexos uma tarefa difícil e confusa.

Portanto, a POO surgiu para resolver essas questões, disponibilizando uma maneira de representar objetos do mundo real como objetos em um código, com características e comportamentos específicos. Além disso, a POO oferece recursos de reutilização de código, encapsulamento, herança e polimorfismo, tornando a programação mais organizada e eficaz.

**VANTAGENS DA POO**

* **Maior facilidade de colaboração:** Permite que vários programadores trabalhem juntos em projetos de software de maneira mais eficiente;
* **Facilidade de manutenção:** Ajuda a organizar o código em unidades menores e mais fáceis de entender, tornando-o mais fácil de manter e corrigir;
* **Melhor capacidade de escalabilidade:** Ajuda a tornar o código escalável, o que é importante em projetos de software e de grande porte;
* **Melhor separação de responsabilidade:** Permite a distribuição das tarefas de maneira mais clara entre diferentes objetos;
* **Reutilização de código:** Permite a reutilização do código em vários lugares, o que economiza tempo e aumenta a eficiência.

**HERANÇA**



A herança é uma das bases da programação orientada a objetos e serve como uma ótima forma de reaproveitar código já existente. Em vez de escrever tudo do zero, existe a possibilidade de criar uma nova classe que "herda" características (como atributos e métodos) de outra. A classe que passa essas características adiante é chamada de superclasse (ou classe base), e a que recebe é conhecida como subclasse (ou classe derivada).

Na prática, isso significa que a subclasse começa com tudo que a superclasse tem — e ainda pode adicionar suas próprias funcionalidades ou até modificar o que veio herdado. Isso deixa o código mais organizado, mais fácil de manter e evita repetições desnecessárias.

Por exemplo, se você tem uma classe Animal com um método **fazer\_som()**, dá pra criar uma classe **Cachorro** que herda de **Animal** e usa esse mesmo método, mas também pode ter comportamentos específicos, como **abanar\_rabo()**.

* **Superclasse (Classe Base):** A classe cujos atributos e métodos são herdados por outras classes. É a classe “pai”.
* **Subclasse (Classe Derivada):** A classe que herda atributos e métodos da superclasse. É a classe “filha”.
* **Herança Simples:** Quando uma subclasse herda de uma única superclasse.
* **Herança Múltipla:** Quando uma subclasse herda de mais de uma superclasse. Nem todas as linguagens de programação suportam herança múltipla devido à sua complexidade.
* **Sobrescrita de Método:** A subclasse pode fornecer uma implementação específica de um método que já existe na superclasse.

**VANTAGENS**

***Reutilização de Código:***

* A herança permite que novas classes (subclasses) reutilizem código existente em outras classes (superclasses), evitando a duplicação de código e tornando o desenvolvimento mais eficiente.

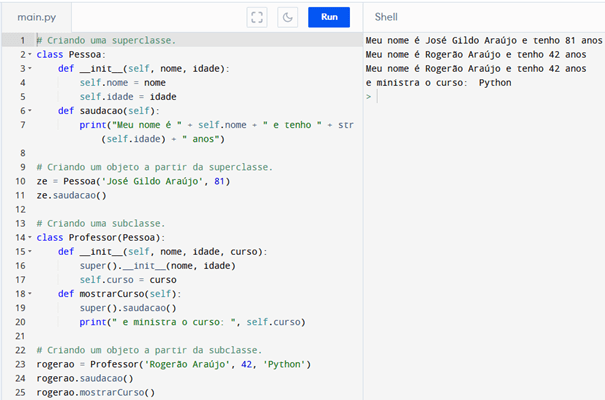
***Facilidade de Manutenção:***

* Mudanças na superclasse são automaticamente propagadas para as subclasses, o que simplifica a manutenção e a atualização do código.

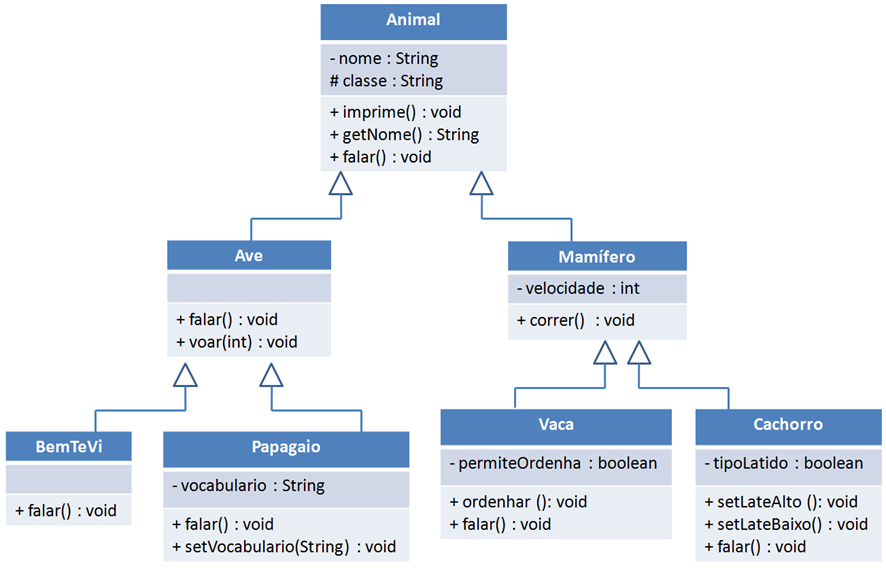
***Organização e Extensibilidade:***

* A herança permite criar uma estrutura hierárquica de classes, facilitando a organização do código e a extensibilidade do sistema, pois novas funcionalidades podem ser adicionadas sem alterar a lógica básica da superclasse.

**EXEMPLO PRÁTICO**



**POLIMORFISMO**



Polimorfismo é definido como um princípio a partir do qual as classes derivadas de uma única classe base são capazes de invocar os métodos que, embora apresentem a mesma assinatura, comportam-se de maneira diferente para cada uma das classes derivadas. Basicamente ele é um mecanismo por meio do qual selecionamos as funcionalidades utilizadas de forma dinâmica por um programa no decorrer de sua execução.

Com o Polimorfismo, os mesmos atributos e objetos podem ser utilizados em objetos distintos, porém, com implementações lógicas diferentes.

Por exemplo: podemos assumir que uma bola de futebol e uma camisa da seleção brasileira são artigos esportivos, mais que o cálculo deles em uma venda é calculado de formas diferentes.

Existem dois tipos principais de polimorfismo: o polimorfismo de sobrecarga e o polimorfismo de sobrescrita. O polimorfismo de sobrecarga ocorre quando uma classe possui vários métodos com o mesmo nome, mas com parâmetros diferentes. Já o polimorfismo de sobrescrita ocorre quando uma classe filha redefine um método da classe pai, implementando um comportamento específico para aquela classe.

**Polimorfismo estático:** ocorre em momento de compilação. O mesmo método é implementado várias vezes na mesma classe, com parâmetros diferentes. A escolha do método a ser chamado vai variar de acordo com o parâmetro passado.

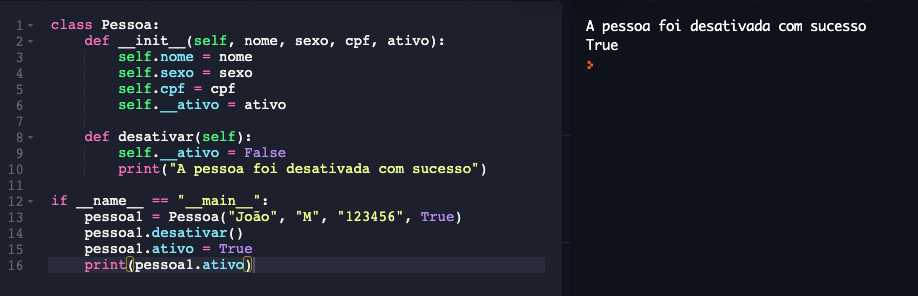
**Polimorfismo dinâmico:** ocorre em momento de execução. O mesmo método é implementado várias vezes nas subclasses derivadas, com os mesmos parâmetros. A escolha do método depende do objeto que o chama (e, consequentemente, da classe que o implementa).

**VANTAGENS**

O polimorfismo é uma das características mais poderosas e fundamentais da programação orientada a objetos, trazendo uma série de vantagens que contribuem para a criação de sistemas mais flexíveis, escaláveis e de fácil manutenção. Ele permite que diferentes classes compartilhem o mesmo nome de método, mas com comportamentos distintos, desde que estejam relacionadas por herança ou implementem a mesma interface. Isso significa que é possível escrever códigos mais genéricos e reutilizáveis, pois uma única função ou estrutura de controle pode operar sobre objetos de diferentes tipos sem precisar conhecer suas implementações específicas.

Essa capacidade de tratar diferentes objetos de forma uniforme promove um design mais limpo e modular, reduz a duplicação de código e facilita a implementação de novas funcionalidades, já que novos comportamentos podem ser adicionados simplesmente criando novas classes que estendam ou implementem as já existentes. Além disso, o polimorfismo fortalece o princípio do aberto/fechado da programação orientada a objetos, no qual as classes estão abertas para extensão, mas fechadas para modificação. Assim, ele contribui diretamente para a manutenção a longo prazo do sistema, permitindo que as mudanças ocorram de forma isolada, sem afetar outras partes do código que dependem de uma interface comum.

**EXEMPLO PRÁTICO**



**DIFERENÇAS E RELAÇÕES**

A herança e o polimorfismo são dois pilares centrais da programação orientada a objetos, e embora estejam diretamente relacionados, possuem diferenças claras em seus propósitos e funcionamento.

A herança é o mecanismo que permite que uma classe (chamada de subclasse ou classe derivada) herde atributos e métodos de outra classe (a superclasse ou classe base), promovendo o reuso de código e a organização hierárquica entre classes. Com ela, é possível criar novas classes a partir de classes existentes, estendendo ou especializando seu comportamento. Por exemplo, se temos uma classe Animal com um método falar(), uma classe Cachorro pode herdar de Animal e sobrescrever esse método para dizer "au au".

O polimorfismo, por sua vez, é a capacidade de um mesmo método ou interface assumir diferentes comportamentos, dependendo do objeto que o invoca. Isso é possível justamente por meio da herança ou da implementação de interfaces. O polimorfismo permite que possamos tratar diferentes objetos de maneira uniforme. Por exemplo, uma função que recebe um objeto do tipo Animal pode ser usada tanto com um Cachorro quanto com um Gato, desde que ambos herdem de Animal e sobrescrevam o método falar().

Portanto, a herança e o polimorfismo estão relacionados porque o polimorfismo depende, em muitos casos, da herança para funcionar. É por meio da herança que diferentes classes compartilham uma mesma interface ou estrutura de métodos, o que permite ao polimorfismo atuar, executando diferentes comportamentos conforme a classe específica do objeto. Em resumo, a herança estabelece a base comum, enquanto o polimorfismo permite a variação de comportamento sobre essa base, promovendo flexibilidade e extensibilidade no design dos sistemas.

**CONCLUSÃO**

Em conclusão, a herança e o polimorfismo são conceitos fundamentais da programação orientada a objetos e desempenham um papel muito importante no desenvolvimento de sistemas mais organizados, reutilizáveis e fáceis de manter. A herança permite que uma classe aproveite características de outra, facilitando o reaproveitamento de código e a criação de estruturas mais claras. Já o polimorfismo permite que diferentes classes utilizem um mesmo método de maneiras diferentes, trazendo mais flexibilidade para o código. Entender como esses dois conceitos se relacionam ajuda a criar programas mais eficientes e preparados para mudanças futuras, o que é essencial tanto em projetos simples quanto em sistemas mais complexos.

**REFERÊNCIAS**

FLORENTINO, F. S. **Programação orientada a objetos: Herança**. Disponível em: <https://dev.to/fabianoflorentino/programacao-orientada-a-objetos-heranca-1pc3>. Acesso em: 27 maio. 2025.

DE CAMARGO, W. B. **Polimorfismo: Programação Orientada a Objetos**. Disponível em: <https://www.devmedia.com.br/conceitos-e-exemplos-polimorfismo-programacao-orientada-a-objetos/18701>. Acesso em: 27 maio. 2025.]

COSTA, T. **Programação Orientada a Objetos(POO) - Herança**. Disponível em: <https://www.dio.me/articles/programacao-orientada-a-objetospoo-heranca>. Acesso em: 27 maio. 2025.

NEVES, C. S. F. **Minutos de Desenvolvimento de Sistemas: Herança e Polimorfismo na Programação Orientada a Objetos**. Disponível em: <https://www.estrategiaconcursos.com.br/blog/heranca-polimorfismo-programacao-orientada-objetos/>. Acesso em: 27 maio. 2025.