**Pilares da POO (Programação Orientada a Objetos)**

A POO (Programação Orientada a Objetos) é um padrão de programação baseado no conceito de “objetos" e conjunto de objetos (classes), que podem conter atributos e métodos. Os objetos em questão são representações de entidades que existem no mundo real. Os atributos seriam as características dessa entidade, enquanto os métodos seriam o comportamento e/ou as ações que o objeto é capaz de realizar.

Por exemplo: Uma classe denominada “animal” tem um objeto denominado cachorro que poderia ter como atributos seu nome, raça e sexo. Já os métodos seriam latir, comer, passear.

A POO possui quatro pilares que fundamentam seu funcionamento (a abstração, o encapsulamento, herança e polimorfismo), e é de suma importância conhece-los para aprofundar-se na programação orientada a objetos. Portanto, vamos aprender um pouco sobre esses fundamentos.

Figura 1- Pilares da POO



<https://images.app.goo.gl/nEMgH9GKixMtW7oL8>

O pilar da abstração é definido como o processo de simplificar objetos complexos para que o código seja reduzido e possua apenas informações essenciais.

Podemos tomar como exemplo um objeto denominado carro, que possui atributos como marca, cor e modelo, e métodos como ligar, desligar e abastecer. O pilar da abstração faz com que o desenvolvedor não foque em exibir dados irrelevantes que possam confundir o usuário, mas sim, foque apenas em criar um código simples.

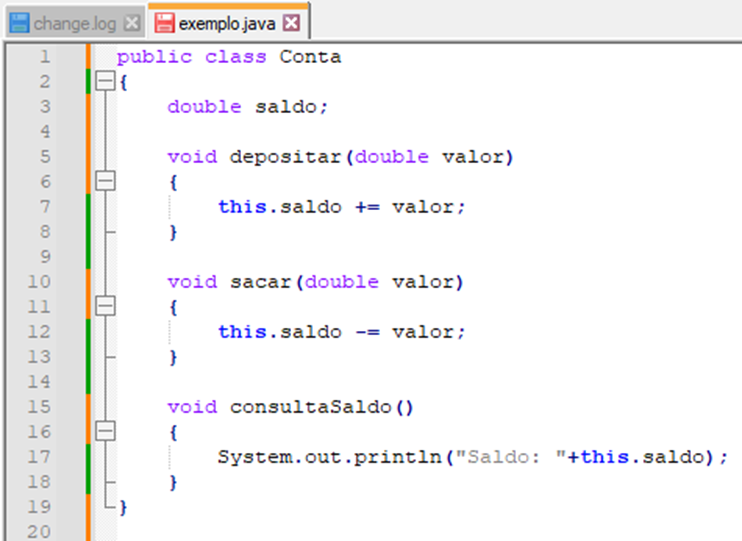
Lidar com questões como “Como o mecanismo do carro funciona?” podem atrasar a experiência do indivíduo no momento da compra, já que tudo que ele precisa saber é que o carro funciona.

Figura 2 – Esquema sobre abstração

<https://images.app.goo.gl/Ks4SppCsP9ENsSDf6>

Esse pilar pode ser importante e benéfico tanto para o programador quanto para o usuário, pois além possibilitar a criação de modelos que se parecem com o objeto da vida real, ele permite que sejam feitas implementações internas em um objeto sem que isso altere sua interface.

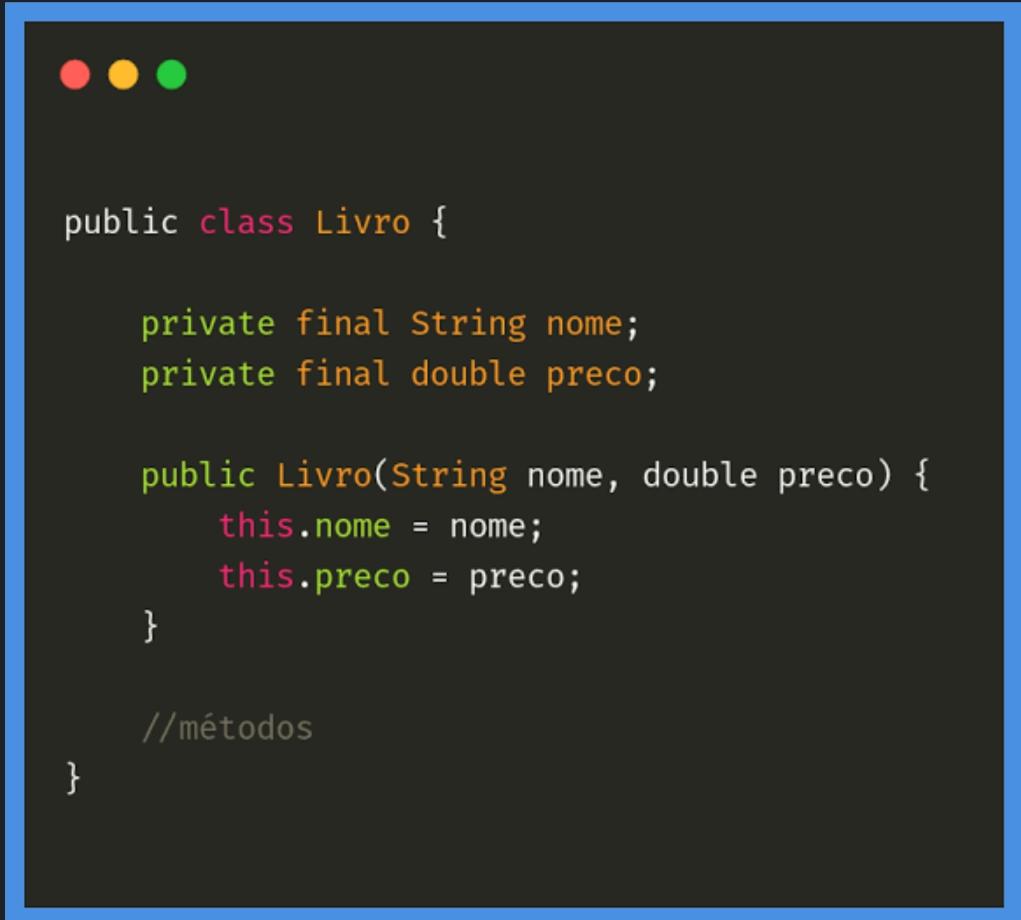
Figura 3 – Exemplo de abstração em Java.

****

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.estrategiaconcursos.com.br%2Fblog%2Fabstracao-encapsulamento-programacao-orientada-objetos%2F&psig=AOvVaw3TkCr_QgiHKSvv0CkmFwkg&ust=1723129232806000&source=images&cd=vfe&opi=89978449&ved=0CBEQjRxqFwoTCLC3hayS44cDFQAAAAAdAAAAABAE>

O segundo pilar da POO é o encapsulamento. O encapsulamento é o processo no qual o programador que está desenvolvendo um projeto utiliza de estruturas como o método, para ocultar e limitar o acesso aos dados do objeto, (suas funcionalidades), que não agregam valor algum para as características e que não deveriam ser acessadas e/ou manipuladas por outras classes.

Esse ato permite que o desenvolvedor faça alterações no código sem que isso altere o resto da programação. Dessa forma, um mesmo objeto pode ser reutilizado diversas vezes em diferentes partes do programa.

Figura 4 – Exemplo de encapsulamento em Java.

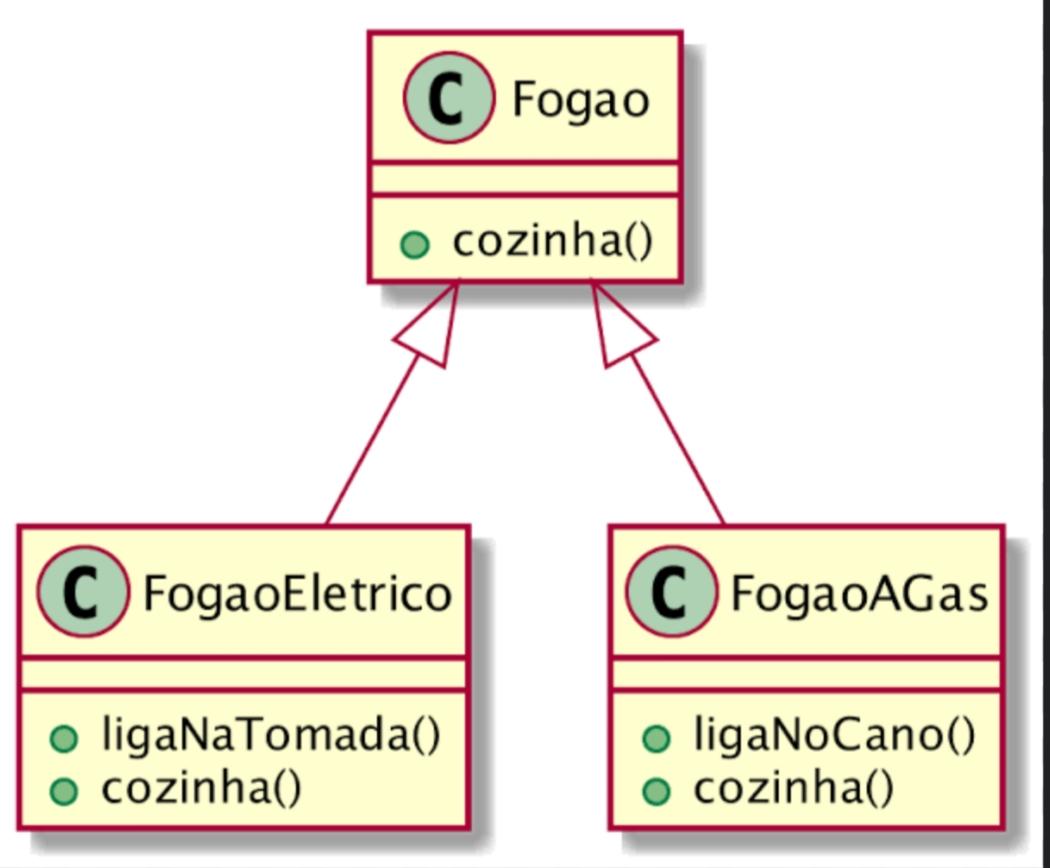
<https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.alura.com.br%2Fartigos%2Fassets%2Fcode%2Fo-que-e-encapsulamento.1721844099.png&tbnid=U6ln2731jFcE0M&vet=1&imgrefurl=https%3A%2F%2Fwww.alura.com.br%2Fartigos%2Fo-que-e-encapsulamento&docid=SN6xgPV-q6GmgM&w=844&h=768&hl=pt-BR&source=sh%2Fx%2Fim%2Fm1%2F4&kgs=07e78060a96516d4&shem=abme%2Ctrie>

Portanto, o encapsulamento é muito importante, pois fornece uma maior proteção aos dados de um objeto, garantindo assim, uma maior segurança e confidencialidade no processo de desenvolvimento do código.

O terceiro pilar, herança, constitui-se do conceito de haver uma classe já existente e ao criar outra, ao invés de construir o código desde o início, aquela classe existente é utilizada como base, compartilhando assim, os mesmos atributos e métodos.

Neste caso, as classes passam a ter um sistema de hierarquia entre si. A classe originária é chamada de classe base, e a que herda suas características é chamada de classe derivada. Desta maneira, as classes que derivam e recebem a herança não são imprevisíveis e precisam apenas definir as suas características individuais que a difere de sua classe base.

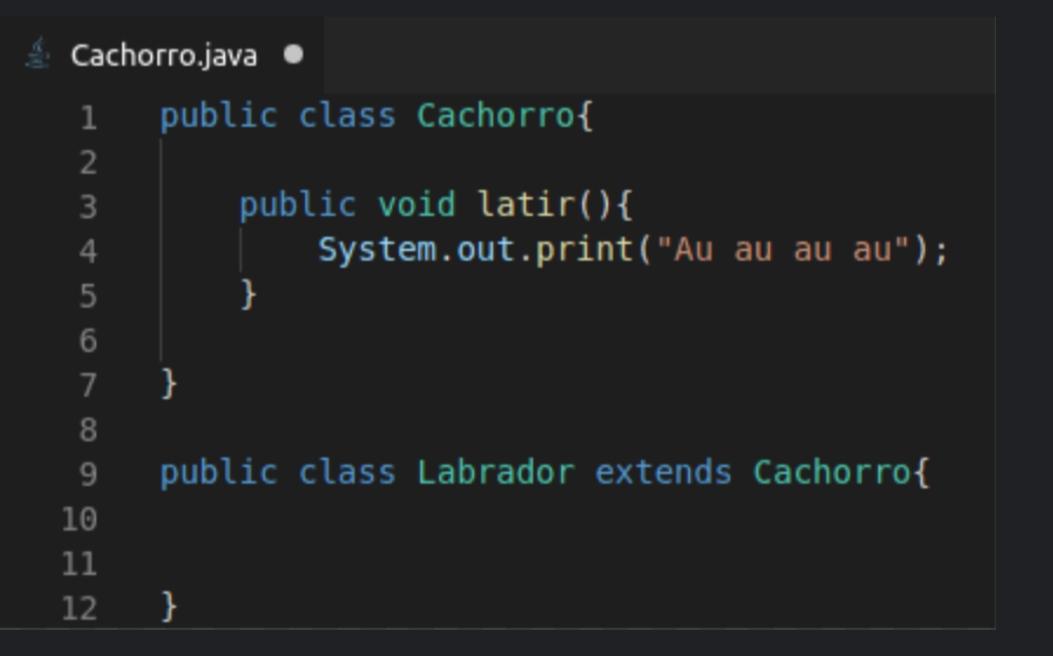
Figura 5 – Esquema sobre herança.



<https://images.app.goo.gl/gFnMssbfRgW9Aj3s8>

Outro exemplo seria: Uma classe derivada “profissionais” que herda as características da classe base “humanos”. Neste caso, não seria necessário que nessa classe derivada fosse questionado o sexo, idade e nome pois esses atributos já estão inseridos na classe “humanos”. Sendo assim, a classe derivada se preocuparia com coisas como a formação, o local que trabalha e etc.

Figura 6 – Exemplo em Java sobre herança.



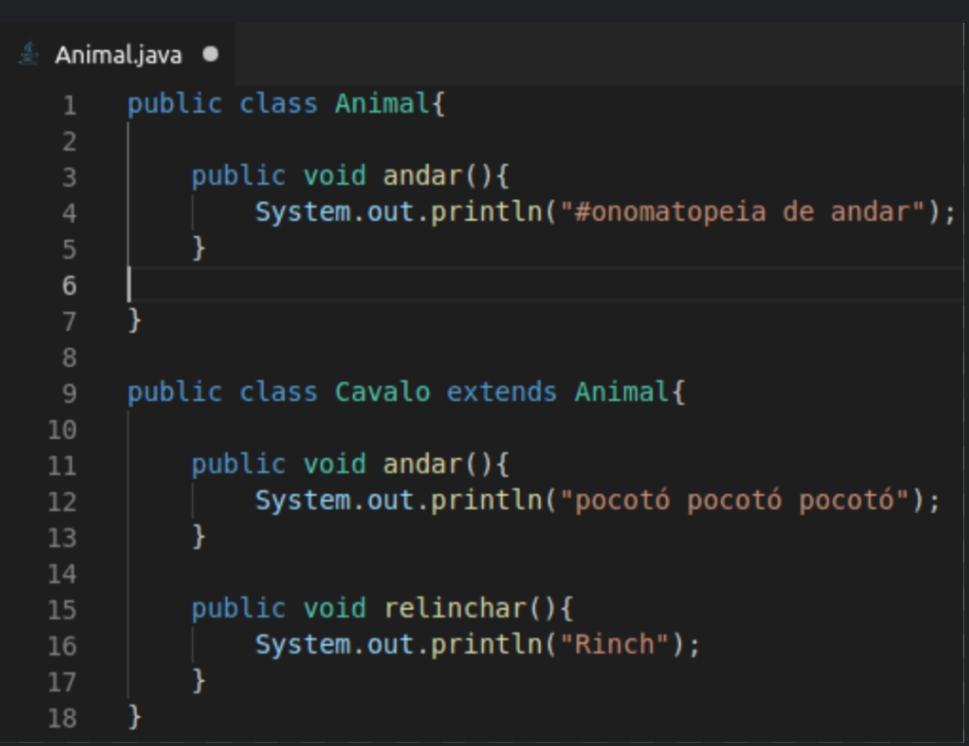
[https://www.google.com/imgres?imgurl=https%3A%2F%2Fmiro.medium.com%2Fv2%2Fresize%3Afit%3A425%2F1\*eUvKXT7ahhR8GCFyAEwAWA.png&tbnid=AQP9DcJetOmLuM&vet=1&imgrefurl=https%3A%2F%2Fmedium.com%2F%40phelipematheus%2Fjava-heran%25C3%25A7a-polimorfismo-755584b8cf9b&docid=WuxzeB9HVABj0M&w=425&h=261&hl=pt-BR&source=sh%2Fx%2Fim%2Fm1%2F4&kgs=36190c360e92b40a&shem=abme%2Ctrie](https://www.google.com/imgres?imgurl=https%3A%2F%2Fmiro.medium.com%2Fv2%2Fresize%3Afit%3A425%2F1*eUvKXT7ahhR8GCFyAEwAWA.png&tbnid=AQP9DcJetOmLuM&vet=1&imgrefurl=https%3A%2F%2Fmedium.com%2F%40phelipematheus%2Fjava-heran%25C3%25A7a-polimorfismo-755584b8cf9b&docid=WuxzeB9HVABj0M&w=425&h=261&hl=pt-BR&source=sh%2Fx%2Fim%2Fm1%2F4&kgs=36190c360e92b40a&shem=abme%2Ctrie)

O quarto e último pilar é chamado de polimorfismo. Um exemplo clássico e que torna mais fácil sua compreensão seria a necessidade de diversos personagens em um determinado jogo executarem os mesmos movimentos. O desenvolvimento do código seria mais complexo se houvesse a necessidade de programar individualmente vários personagens que no final acabariam executando os mesmos movimentos no jogo. Desse modo, o polimorfismo utiliza de classes abstratas para redefinir o comportamento de outra classe, ou seja, de certa forma o polimorfismo é um objeto capaz de assumir varias formas.

Para compreender melhor esse conceito é preciso saber o que é uma classe abstrata. Esse tipo de classe é como uma receita que tem os ingredientes e o modo de preparo. Os ingredientes seriam o esqueleto da programação em si e o modo de preparo seria as implementações sobre o personagem.

A interface também é um conceito importante a ser mencionado, pois é um tipo de classe abstrata que possui só o esqueleto da implementação (as implementações são representadas de forma muito simples). Sendo assim, qualquer objeto que esteja de acordo com a classe abstrata e a interface pode ser manuseado pela classe controladora, contando com a abstração.

Esse processo torna o código mais flexível e permite que implementações sejam feitas sem causar grandes alterações.

****Figura 7 – Exemplo em Java sobre polimorfismo.

<https://images.app.goo.gl/zytptpV3stmeFPZR8>

Diante do exposto, pode-se concluir que os pilares da Programação Orientada a Objetos (POO) são essenciais. Eles possibilitam ocultar os detalhes internos dos objetos, expondo apenas o necessário, o que facilita a manutenção e a proteção dos dados. A herança permite que uma classe herde propriedades e comportamentos de outra, promovendo a reutilização de código. Além disso, o polimorfismo faz com que métodos com o mesmo nome se comportem de maneira diferente, conforme o tipo de objeto.

Os fundamentos são importantes, pois permitem a criação de modelos simplificados que representam conceitos complexos, influenciando de forma direta no desenvolvimento de softwares, tornando-os robustos, flexíveis e reutilizáveis.

Atualmente, os pilares da Programação Orientada a Objetos são indispensáveis para um bom desenvolvedor, não apenas para a organização do código, a facilidade de manutenção e a expansão, mas também porque, ao se inter-relacionarem, os pilares contribuem para a obtenção de uma qualidade superior no programa.

**Perguntas frequentes:**

**O que são os pilares da POO e quais são eles?**

Existem quatro conceitos fundamentais da Programação Orientada a Objetos: o Encapsulamento, Herança, Polimorfismo e Abstração. Esses conceitos ajudam a estruturar e organizar o código de forma mais eficiente e modular.

**Como o encapsulamento melhora a segurança e a manutenção do código?**

**:** O encapsulamento é o princípio de esconder os detalhes internos do funcionamento de uma classe e expor apenas o que é necessário para o exterior. Isso ajuda a proteger os dados internos de alterações indesejadas e reduz a complexidade ao permitir que mudanças na implementação interna não afetem outras partes do código.

**Como a abstração facilita o design e a implementação de sistemas?**

A abstração é o processo de simplificar um sistema ao definir apenas as características essenciais e ocultar os detalhes complexos. Em POO, isso é feito por meio de classes e interfaces, permitindo que os desenvolvedores se concentrem nos aspectos importantes do sistema sem se perderem em detalhes. Isso ajuda a reduzir a complexidade e a melhorar a clareza do design.

**Referências bibliográficas:**

<https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos>

<https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_orientada_a_objetos>

<https://blog.grancursosonline.com.br/pilares-da-poo/>

<https://www.dio.me/articles/os-4-pilares-da-programacao-orientada-a-objetos-SSU4Q9>

<https://www.alura.com.br/artigos/o-que-e-encapsulamento>

<https://web.tecgraf.puc-rio.br/~marcio/cursos/oo/encapsul.htm#:~:text=1.,para%20as%20suas%20caracter%C3%ADsticas%20essenciais>.

<https://www.locaweb.com.br/blog/temas/codigo-aberto/o-que-e-encapsulamento/#:~:text=O%20encapsulamento%20permite%20que%20os,com%20os%20detalhes%20de%20implementa%C3%A7%C3%A3o>.

<https://www.devmedia.com.br/conceitos-e-exemplos-heranca-programacao-orientada-a-objetos-parte-1/18579>