Fernanda de Oliveira Nunes 1C 2º semestre

**Pilares da Programação Orientada a Objeto**

****

Programação Orientada a Objetos (POO) é um paradigma de programação que trata a computação como uma interação entre objetos. A POO se concentra na criação de objetos que possuem características e comportamentos próprios. No caso da orientação a objetos, temos métodos que são aplicados aos dados de cada objeto. Essencialmente, os procedimentos e métodos são iguais, sendo diferenciados apenas pelo seu escopo. A programação orientada a objetos traz outros pontos que acabam sendo mais interessantes no contexto de aplicações modernas. Como o desempenho das aplicações não é uma das grandes preocupações na maioria das aplicações (devido ao poder de processamento dos computadores atuais), a programação orientada a objetos se tornou muito difundida. Essa difusão se dá muito pela questão da reutilização de código e pela capacidade de representação do sistema muito mais perto do que veríamos no mundo real. Através da criação de objetos que combinam dados (atributos) e comportamentos (métodos), a POO possibilita a construção de sistemas mais organizados, flexíveis e reutilizáveis.

As classes são como moldes que especificam quais atributos e métodos um objeto pode ter. Os objetos podem interagir entre si por meio de mensagens, o que permite a comunicação e a colaboração entre eles.

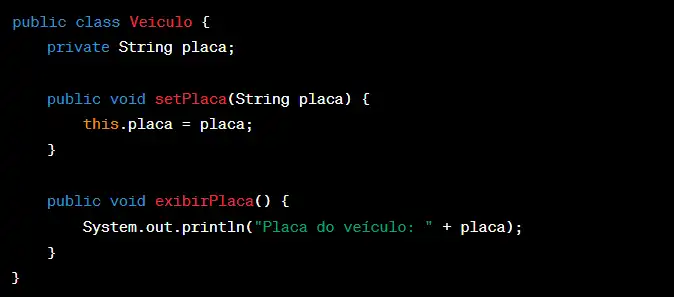


* Encapsulamento

Se trata de um dos elementos que adicionam segurança à aplicação em uma programação orientada a objetos pelo fato de esconder as propriedades, criando uma espécie de caixa preta.

A maior parte das linguagens orientadas a objetos implementam o encapsulamento baseado em propriedades privadas, ligadas a métodos especiais chamados getters e setters, que irão retornar e setar o valor da propriedade, respectivamente. Essa atitude evita o acesso direto a propriedade do objeto, adicionando uma outra camada de segurança à aplicação.

Para fazermos um paralelo com o que vemos no mundo real, temos o encapsulamento em outros elementos. Por exemplo, quando clicamos no botão ligar da televisão, não sabemos o que está acontecendo internamente. Podemos então dizer que os métodos que ligam a televisão estão encapsulados.



(Neste exemplo, temos a classe Veiculo que possui um atributo privado chamado placa. O encapsulamento é aplicado ao controlar o acesso direto a esse atributo e fornecer métodos públicos para interagir com ele. O método setPlaca é usado para definir a placa do veículo, enquanto o método exibirPlaca é usado para mostrar a placa na saída. Essa abordagem protege o atributo placa de modificações indesejadas e centraliza o acesso a ele por meio de métodos controlados.)

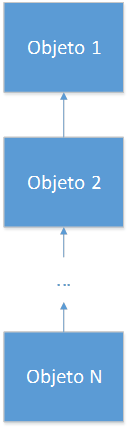
* Herança

Essa característica otimiza a produção da aplicação em tempo e linhas de código.

Para entendermos essa característica, vamos imaginar uma família: a criança, por exemplo, está herdando características de seus pais. Os pais, por sua vez, herdam algo dos avós, o que faz com que a criança também o faça, e assim sucessivamente. Na orientação a objetos, a questão é exatamente assim, como mostra a Figura 2. O objeto abaixo na hierarquia irá herdar características de todos os objetos acima dele, seus “ancestrais”. A herança a partir das características do objeto mais acima é considerada herança direta, enquanto as demais são consideradas heranças indiretas. Por exemplo, na família, a criança herda diretamente do pai e indiretamente do avô e do bisavô.



(A herança é exemplificada pela relação entre uma classe base Veiculo e suas subclasses Carro e Motocicleta. A classe base possui atributos e métodos comuns, que são herdados pelas subclasses.)

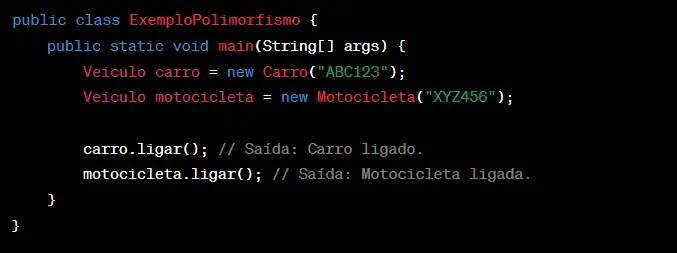


* Polimorfismo

O polimorfismo consiste na alteração do funcionamento interno de um método herdado de um objeto pai.

Como um exemplo, temos um objeto genérico “Eletrodoméstico”. Esse objeto possui um método, ou ação, “Ligar()”. Temos dois objetos, “Televisão” e “Geladeira”, que não irão ser ligados da mesma forma. Assim, precisamos, para cada uma das classes filhas, reescrever o método “Ligar()”.

Com relação ao polimorfismo, valem algumas observações. Como se trata de um assunto que está intimamente conectado à herança, entender os dois juntamente é uma boa ideia.



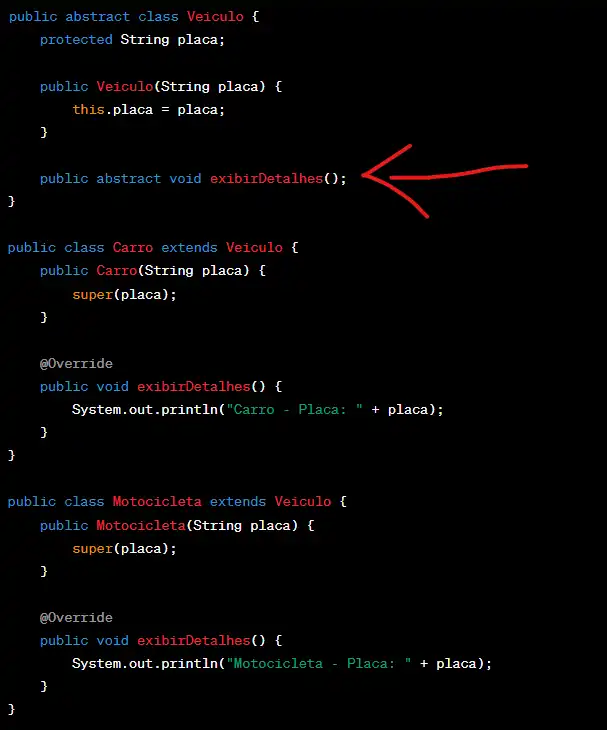
(O polimorfismo é demonstrado quando objetos das classes Carro e Motocicleta (subclasses de Veiculo) são tratados de forma genérica, chamando o método ligar() que é sobrescrito em cada subclasse.)

* Abstração

O primeiro ponto é darmos uma identidade ao objeto que iremos criar. Essa identidade deve ser única dentro do sistema para que não haja conflito. Na maior parte das linguagens, há o conceito de pacotes (ou namespaces). Nessas linguagens, a identidade do objeto não pode ser repetida dentro do pacote, e não necessariamente no sistema inteiro. Nesses casos, a identidade real de cada objeto se dá por ..

A segunda parte diz respeito a características do objeto. Como sabemos, no mundo real qualquer objeto possui elementos que o definem. Dentro da programação orientada a objetos, essas características são nomeadas propriedades. Por exemplo, as propriedades de um objeto “Cachorro” poderiam ser “Tamanho”, “Raça” e “Idade”.

Por fim, a terceira parte é definirmos as ações que o objeto irá executar. Essas ações, ou eventos, são chamados métodos. Esses métodos podem ser extremamente variáveis, desde “Acender()” em um objeto lâmpada até “Latir()” em um objeto cachorro.



(Neste exemplo, estamos lidando com a hierarquia de classes relacionada a veículos. A classe abstrata Veiculo serve como uma classe base para as subclasses Carro e Motocicleta. A classe abstrata Veiculo tem um construtor que recebe a placa do veículo, enquanto a subclasse Carro e Motocicleta estendem a classe base. Cada subclasse implementa o método abstrato exibirDetalhes, que é responsável por exibir os detalhes específicos de cada tipo de veículo, juntamente com sua placa.)

<https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>

[https://awari.com.br/os-4-pilares-da-programacao-orientada-a-objetos-guia-completo-para-iniciantehttps://www.dio.me/articles/os-4-pilares-da-programacao-orientada-a-objetos-Y0CN7Gs/](https://awari.com.br/os-4-pilares-da-programacao-orientada-a-objetos-guia-completo-para-iniciantes/)

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

<https://www.youtube.com/watch?v=dXZRgW-X2ls>

<https://youtu.be/8VcZkAYygoo?si=GOpXxEny3S7L-gl7>

<https://youtu.be/x9aLELLVzjQ?si=g_PQFqz4JZaJZsC->