Programação orientada a objetos

Paradigma da programação

Ícaro Lisbôa Assunção
Instituto Federal da Bahia
Terceiro semestre

Camaçari - BA
18/12/2024

Existem oceanos

Assim como os sete mares que residem nosso planeta terra, existem oceanos dentro da linguagem de programação, e esses oceanos são paradigmas que induzem formas de se lidar e elaborar estruturas de códigos para resolver determinados problemas. Quando observamos linguagens de programação específicas, tais como: Python, C, C#, é possível ter compressão que cada uma possui uma forma específica de tratar a programação e como lida com seus paradigmas.

A linguagem Python por exemplo, é uma linguagem híbrida, a mesma utiliza os dois paradigmas da programação para lidar com seus modus operandi da programação, então a mesma tem por finalidade, trabalhar tanto por uma linguagem estruturada quanto uma linguagem orientada a objetos.

Dentro do paradigma de uma linguagem estruturada, possuímos características específicas que determinam sua estrutura básica, tais como:

Sequências: A sequência de uma linguagem estruturada, é a ordem de execução de cada função.

Condições: Sequências executadas a partir de condições específicas (if, else if, else).

Repetições: São laços que repetem padrões específicos sobre determinadas condições.

"Um exemplo clássico de uma linguagem estruturada, é o C, que detém esses princípios baseados em sua estrutura."

Essas estruturas básicas são responsáveis por manejar uma linguagem estruturada. A Programação Orientada a Objetos, surge com o intuito de ser uma forma a parte de lidar com essas questões orientadas.

A POO, surge com a principal finalidade de utilizar a abstração de objetos tangíveis para a programação, por isso, intitulado objetos, pois, são conceitos abstratos para definir um conceito tangível.

"O conceito de POO é baseado em classes e objetos abstratos"

Classes e Objetos

Como anteriormente explicitado, a Programação Orientada a Objetos, trata de forma inerente a questão de classes e objetos, toda a estrutura é baseada na manipulação de uma classe que detém na mesma objetos. Mas afinal, o que é uma classe? e o que é um objeto?

Classe:

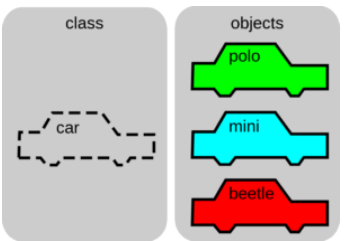
Uma classe pode ser vista como um molde/template, para um objeto, podemos ter como exemplo a classe “carro”, que possui elementos e características que definem a classe ser um carro.

```
class carro{

  public string Cor;
  public string Modelo;
  public int Ano;

}
```

Objeto:

Um objeto é uma instância da classe, ou seja, uma unidade daquela classe, que detém cada característica que a fazem pertencer àquela classe, então:

```
Carro carro1 new();
carro1.cor = “Red”;
carro1.Modelo = “Fiat”;
carro1.Ano = 1990;
```



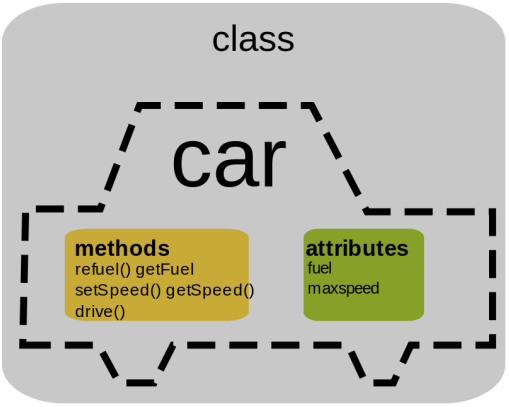
Pilares

Compreendido como funciona uma Linguagem Orientada a Objetos, vamos entender o conceito desse paradigma, que auxilia diretamente na organização de um código.

Encapsulamento

O primeiro pilar da POO, é o encapsulamento, que consiste em esconder detalhes internos de uma classe. Ainda com a analogia da classe “Carro”.

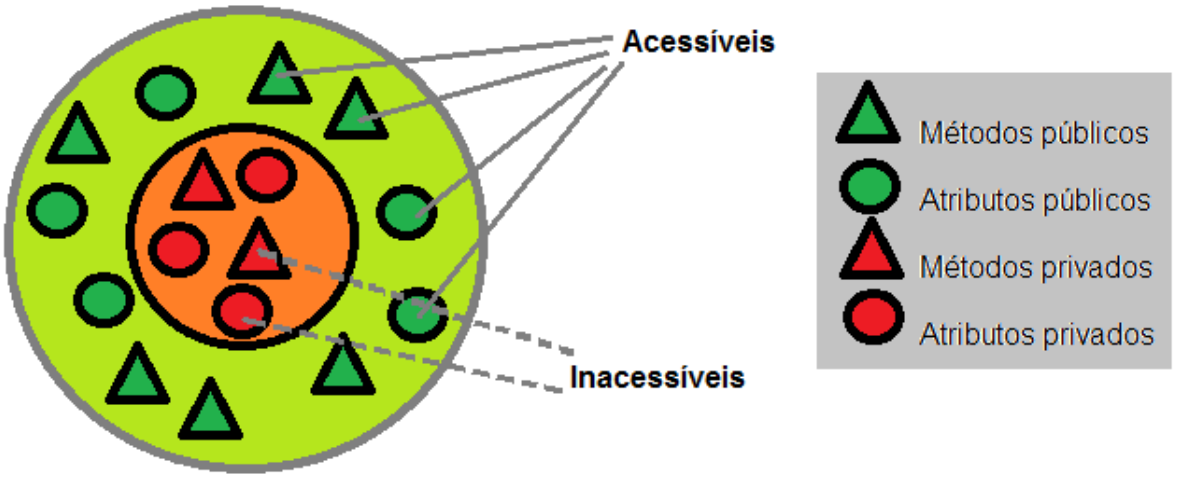
Como visto anteriormente, o carro possui determinados atributos e métodos, sendo atributos características, cor, modelo, e ano. E métodos, como aceleração, injeção eletrônica, e combustível.



Entretanto, caso esses atributos e métodos forem visíveis e modificáveis, como o mecanismo de aceleração, injeção eletrônica, isso pode dá liberdade indesejada para a realização de determinadas alterações, resultando de forma intrínseca questões de efeitos colaterais imprevisíveis. Nesse sentido, utilizemos um evento realístico para abordar o perigo desse conceito.

Suponhamos que um indivíduo não se encontre satisfeito com a velocidade do seu veículo, e o mesmo decide aumentar a aceleração do mesmo. O carro fica com uma aceleração maior, e de fato recorre a solicitação esperada, porém, de antemão pode e ocorre consequências imprevisíveis, como o funcionamento do veículo ser diminuído a curto prazo, ou até mesmo o veículo não funcionar ao tentar ligar o mesmo.

Dizemos então neste caso que o método de aceleração do seu carro não é visível por fora do próprio carro. E na POO, um atributo ou método que não é visível de fora do próprio objeto é chamado de “privado” e quando é visível, é chamado de “público”.



O conceito de encapsulamento pode ser entendido assim: quando queremos acelerar um carro, não precisamos entender exatamente como o motor funciona para realizar esse movimento. O que sabemos é que ao pisar no acelerador, o carro acelera, e o processo de aceleração é gerido internamente. A aceleração, neste caso, está "encapsulada", ou seja, o resultado desejado é exposto, mas os detalhes de como ele é alcançado são ocultos. Para o programa, importa apenas que a ação de acelerar seja realizada corretamente, sem a necessidade de saber os detalhes de sua execução.

O mesmo se aplica aos atributos. Por exemplo, não precisamos saber como o carro calcula a velocidade ou como o velocímetro mostra a velocidade. O importante é que o velocímetro forneça o valor correto, e esse cálculo é feito internamente ao carro. A manipulação dos atributos, como ler ou alterar seu valor, é feita através de métodos específicos, conhecidos como getters e setters, que controlam o acesso.

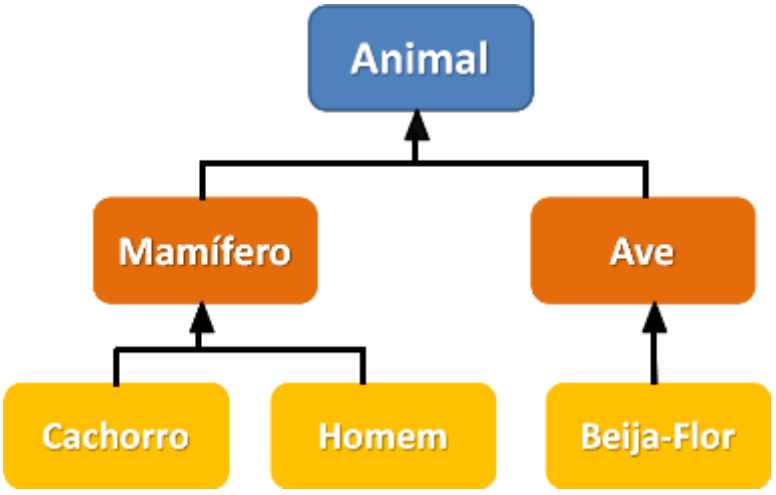
Esse tipo de encapsulamento ajuda a evitar o "vazamento de escopo", ou seja, previne que atributos ou métodos fiquem acessíveis a partes do programa que não deveriam ter acesso. Com isso, o uso de variáveis globais é evitado, e o código fica mais organizado, pois permite identificar de forma clara quem pode modificar o estado de uma variável em determinado momento, evitando confusões.

“Encapsulamento é esconder os detalhes internos de um objeto e mostrar apenas o necessário para que ele funcione corretamente.”

Herança

No exemplo anterior da classe “Carro”, imagine agora que você comprou um modelo específico, que, apesar de ser único, tem versões muito semelhantes, ou com pequenas modificações. Por exemplo, você comprou um Fiat Uno, mas existe uma versão desse modelo adaptada para enfrentar estradas de terra, com algumas diferenças nos atributos, mas mantendo as características principais.

Quando afirmamos que a classe A é um tipo de classe B, estamos dizendo que a classe A herda atributos e comportamentos da classe B, sendo a classe B a classe mãe da classe A, estabelecendo uma relação de herança entre elas. No caso do carro, podemos dizer que o Fiat Uno 2.0 é uma versão do Fiat Uno, em que as adaptações para estradas de terra representam uma especialização da versão original, mas o modelo básico permanece o mesmo. Esse é um exemplo claro de herança em programação orientada a objetos.

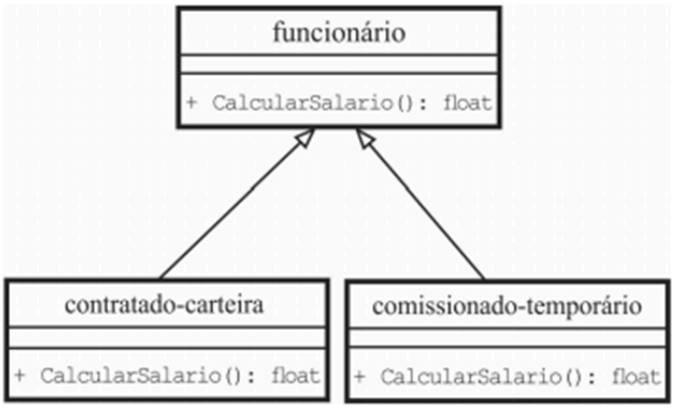



```
public class FiatUno extends Carro{

public FiatUno2.0 (mecanismo de adaptação da estrada de terra){
  String modelo = “FiatUno2.0”;
  super(modelo,  mecanismo de adaptação da estrada de terra);
  }
}
```

“Herança é um conceito da Programação Orientada a Objetos em que uma classe (filha) herda atributos e métodos de outra classe (mãe)”

Polimorfismo

Suponha que um dos motivos para você ter comprado um carro tenha sido a qualidade do sistema de som. No seu caso, a reprodução só é possível via rádio ou Bluetooth, enquanto no seu carro antigo, era feita apenas por cartão SD ou pendrive. Embora ambos os carros tenham o método "tocar música", a maneira como o som é reproduzido é diferente devido aos sistemas distintos. Isso é um exemplo de polimorfismo, onde dois objetos, pertencentes a classes diferentes, possuem o mesmo método, mas com implementações variadas. Ou seja, um método pode ter várias formas e implementações, mas sempre com o mesmo resultado. A palavra "polimorfismo" vem do grego, onde "poli" significa muitas e "morphos" significa forma.




```
public class Main{
  public static void main(String[] args) {
    Automovel moto = new Moto(“Yamaha XPT-100”, new
MecanismoDeAceleracaoDeMotos())
    Automovel carro = new Carro(“Honda Fit”, new
MecanismoDeAceleracaoDeCarros())

List<Automovel> listaAutomoveis = Arrays.asList(moto, carro);

for(Automovel automovel ; listaAutomoveis) {
   automovel.acelerar();
   automovel.acenderFarol();
  }
}
```

“Polimorfismo é o conceito em que um mesmo método pode ter diferentes implementações, permitindo que objetos de classes diferentes respondem de maneira específica à mesma ação.”

Referências

Você pode acessar um artigo sobre Programação Orientada a Objetos (POO) e seus conceitos essenciais, como polimorfismo, herança e encapsulamento, no seguinte link: Alura - POO: Programação Orientada a Objetos.