SENAI - ROBERTO MANGE

CAUÃ OLIVE BARBOSA

“Programação Orientada a Objetos”

Pesquisa

CAMPINAS/SP
2025

Sumário

Paradigmas

Paradigmas de programação são modos de decidir resolver determinado problema por meio de programação, assim como quando no nosso dia a dia, temos diversas formas de fazer uma mesma tarefa ou atividade.

As duas mais comuns são:

Programação Estruturada:

Esse paradigma é geralmente formado por códigos em um único bloco e com ênfase em sequência, decisão e iteração (Sub-rotinas, laços de repetição, condicionais e, estruturas em bloco), e foi impulsionados pelas vantagens práticas que o paradigma oferece, e também pelo **"Teorema do Programa Estruturado".**

O **Teorema do Programa Estruturado** afirma que **todo programa pode ser construído a partir da combinação de três estruturas básicas: sequência, seleção e repetição**. A programação estruturada é uma **abordagem de desenvolvimento de software que depende da organização lógica e ordenada de instruções** para resolver um problema.

Como dito antes, na programação estruturada, um programa é composto por **três tipos básicos de estruturas:**

- **Sequências:** são os comandos a serem executados
- **Condições:** sequências que só devem ser executadas se uma condição for satisfeita (exemplos: if-else, switch e comandos parecidos)
- **Repetições:** sequências que devem ser executadas repetidamente até uma condição for satisfeita (for, while, do-while etc)

Além disso, o **acesso às variáveis não possuem muitas restrições** na programação estruturada. Em linguagens fortemente baseadas nesse paradigma, **restringir o acesso à uma variável se limita a dizer se ela é visível ou não dentro de uma função** (ou módulo, como no uso da palavra-chave static, na linguagem C, mas não se consegue dizer de forma nativa que uma variável pode ser acessada por apenas algumas rotinas do programa. O **contorno** para situações como essas **envolve práticas de programação danosas ao desenvolvimento do sistema**, como o **uso excessivo de variáveis globais**.

Vale lembrar que variáveis globais são usadas tipicamente para manter estados no programa, marcando em qual parte dele a execução se encontra.

Apesar de ter sido sucedida pela POO, a PE ainda é muito influente pois grande parte das pessoas ainda aprende programação através dela. Para a resolução de problemas simples e diretos, a programação estruturada é bastante eficiente (talvez mais eficiente que a POO). Além disso, por exigir formas de pensar relativamente complexas, a POO até hoje ainda não é bem compreendida ou usada pela maioria.

Programação Orientada a Objetos:

A programação orientada a objetos tem o **propósito principal de aproximar o mundo lógico da programação e o mundo em que vivemos**. À vista disso, ela parte do **princípio de que tudo é objeto** — isso mesmo, tudo o que existe são os objetos.

No entanto, **eles necessitam ser criados** (tal qual no mundo real). Para isso, eles precisam de uma espécie de fôrma, de um invólucro, algo que possa lhes dar ao menos seu aspecto inicial: um ponto de partida!

Imagine que você comprou um carro recentemente e decide modelar esse carro usando programação orientada a objetos. O seu carro tem as características que você estava procurando: um motor 2.0 hybrid, azul escuro, quatro portas, câmbio automático etc. Ele também possui comportamentos que, provavelmente, foram o motivo de sua compra, como acelerar, desacelerar, acender os faróis, buzinar e tocar música. Podemos dizer que o carro

novo é um objeto, onde suas características são seus atributos (dados atrelados ao objeto) e seus comportamentos são ações ou métodos.

Seu **carro é um objeto** seu mas na loja onde você o comprou existiam vários outros, muito similares, com quatro rodas, volante, câmbio, retrovisores, faróis, dentre outras partes. Observe que, **apesar do seu carro ser único** (por exemplo, possui um registro único no Departamento de Trânsito), **podem existir outros com exatamente os mesmos atributos, ou parecidos, ou mesmo totalmente diferentes, mas que ainda são considerados carros**. Podemos dizer então que **seu objeto pode ser classificado** (isto é, seu objeto pertence à uma classe) como um carro, e que seu carro nada mais é que uma instância dessa classe chamada "carro".

Esse quadro completa nosso comparativo de POO: **todas essas características individuais dos objetos são os seus atributos, que pertencem somente a eles**. Já **suas funcionalidades** (caso tenha alguma) **são os seus métodos**. Por exemplo, acelerar e frear são funcionalidades de um carro.

São essas quatro características — **objetos, classes, atributos e métodos** — que definem o paradigma de programação orientada a objetos

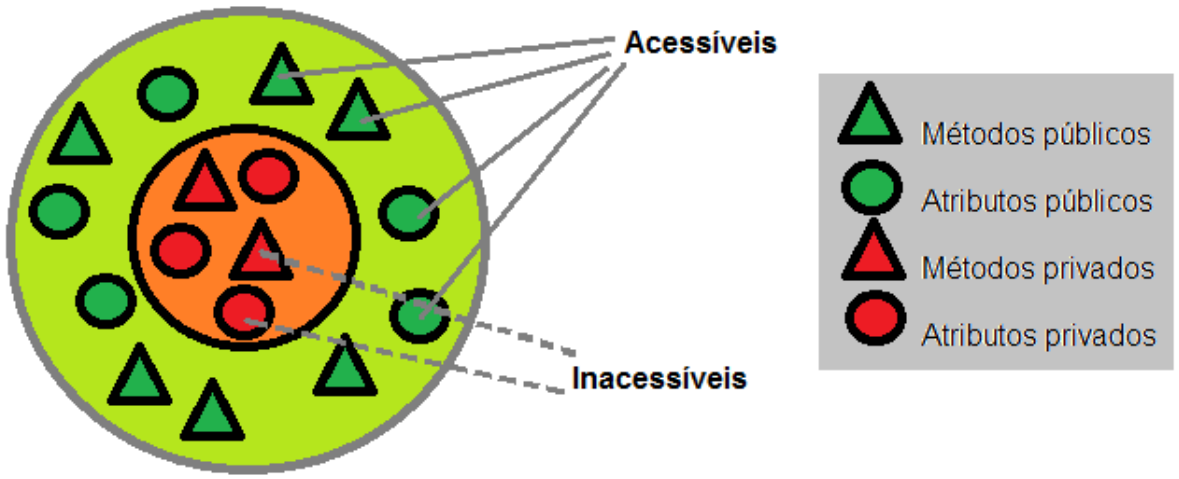
Pilares

Encapsulamento

Ainda usando a analogia do carro, sabemos que ele possui atributos e métodos, ou seja, características e comportamentos. Os métodos do carro, como acelerar, podem usar atributos e outros métodos do carro como o tanque de gasolina e o mecanismo de injeção de combustível, respectivamente, uma vez que acelerar gasta combustível.

No entanto, se alguns desses atributos ou métodos forem facilmente visíveis e modificáveis, como o mecanismo de aceleração do carro, isso pode dar liberdade para que alterações sejam feitas, resultando em efeitos colaterais imprevisíveis. Nessa analogia, uma pessoa pode não estar satisfeita com a aceleração do carro e modifica a forma como ela ocorre, criando efeitos colaterais que podem fazer o carro nem andar, por exemplo.

Dizemos, nesse caso, que o método de aceleração do seu carro não é visível por fora do próprio carro. 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".



Mas então, como sabemos como o nosso carro acelera? É simples: não sabemos. Nós só sabemos que para acelerar, devemos pisar no acelerador e de resto o objeto sabe como executar essa ação sem expor como o faz. Dizemos que a aceleração do carro está encapsulada, pois sabemos o que ele vai fazer ao executarmos esse método, mas não sabemos como - é na verdade, não importa para o programa como o objeto o faz, só que ele o faça.

O mesmo vale para atributos. Por exemplo: não sabemos como o carro sabe qual velocidade mostrar no velocímetro ou como ele calcula sua velocidade, mas não precisamos saber como isso é feito. Só precisamos saber que ele vai nos dar a velocidade certa. Ler ou alterar um atributo encapsulado pode ser feito a partir de getters e setters (colocar referência).

Esse encapsulamento de atributos e métodos impede o chamado vazamento de escopo, onde um atributo ou método é visível por alguém que não deveria vê-lo, como outro objeto ou classe. Isso evita a confusão do uso de variáveis globais no programa, deixando mais fácil identificar em qual estado cada variável vai estar a cada momento do programa, já que a restrição de acesso nos permite identificar quem consegue modificá-la.

Herança

Como o próprio nome diz, trata-se de uma relação de receber algo pré-existente. No caso da POO, a herança é um evento que ocorre entre classes. A doadora é chamada de classe-mãe. Já a classe que herda é chamada de filha.

Quando ocorre uma herança, a classe-filha herda as características da classe-mãe. Isso é bastante útil para um reaproveitamento de código, pois não seria necessário refazer algo que já existe. Parte-se de um ponto e se desenvolvem novos métodos.

Polimorfismo

Vamos dizer que um dos motivos de você ter comprado um carro foi a qualidade do sistema de som dele. Mas, no seu caso, digamos que a reprodução só pode ser feita via rádio ou bluetooth, enquanto que no seu antigo carro, podia ser feita apenas via cartão SD e pendrive. Em ambos os carros está presente o método "tocar música" mas, como o sistema de som deles é diferente, a forma como o carro toca as músicas é diferente. Dizemos que o método "tocar música" é uma forma de polimorfismo, pois dois objetos, de duas classes diferentes, têm um mesmo método que é implementado de formas diferentes, ou seja, um método possui várias formas, várias implementações diferentes em classes diferentes, mas que possuem o mesmo efeito ("polimorfismo" vem do grego poli = muitas, morphos = forma).

Abstração

Abstrair algo significa esconder os detalhes da implementação dentro de algo – às vezes um protótipo, às vezes em uma função. Portanto, quando você chama a função, não precisa entender exatamente o que ela está fazendo.

Um exemplo claro do conceito de abstração seria o funcionamento de um carro. Quando acionamos ele para ligar, não precisamos saber quais passos ele faz para colocar o motor em funcionamento. Quando acionamos o freio, não precisamos saber todos os mecanismos que são acionados para fazer o carro frear. Apenas sabemos o que cada objeto ou função do carro produz como resultado.

Voltando para a codificação, se você tivesse que entender cada função em uma base de código grande, você nunca codificaria nada, pois levaria meses para terminar de ler e entender a lógica de tudo isso.

Contudo, abstraindo certos detalhes, você é capaz de criar uma base de código reutilizável, simples de entender e facilmente alterável.