Programação orientada a objetos

- Como a maioria das atividades que fazemos no dia a dia, programar também possui modos diferentes de se fazer. Esses modos são chamados de paradigmas da programação e, entre eles, estão a programação orientada a objetos (POO) e a programação estruturada.

Estruturada

- Na programação estruturada, um programa é composto por três tipos básicos de estruturas:

1) Sequências: são os comandos a serem executados

2) Condições: if, else, switch, etc.

3) Repetições: for, while, do-while, etc.

- Essas estruturas são usadas para processar a entrada do programa, alterando os dados até que a saída esperada seja gerada.

- A principal característica é que o programa é tipicamente escrito em uma única rotina (ou função) podendo, é claro, ser quebrado em sub-rotinas. Mas o fluxo do programa continua o mesmo, como se pudéssemos copiar e colar o código das sub-rotinas diretamente nas rotinas que as chamam, de tal forma que, no final, só haja uma grande rotina que execute todo o programa.

- 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 consegue dizer de forma nativa que uma variável pode ser acessada por apenas algumas rotinas do programa.

Programação orientada a objetos

- Se baseia principalmente em dois conceitos chave: ***Classes e objetos***

- O que são?

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 híbrido, 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, 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".

- Dessa forma, uma classe é um conjunto de características e comportamentos que definem o conjunto de objetos pertencentes à essa classe. Repare que a classe em si é um conceito abstrato, como um molde, que se torna concreto e palpável através da criação de um objeto. Chamamos essa criação de ***instanciação da classe***, como se estivéssemos usando esse molde (classe) para criar um objeto.

Pilares da POO

- POO se baseia em quatro pilares fundamentais:

1) Abstração

- A abstração envolve a criação de modelos simplificados para representar objetos do mundo real. Ela permite que você se concentre nos aspectos relevantes de um objeto e ignore os detalhes desnecessários.

- Por exemplo, ao modelar um sistema de biblioteca, você pode criar uma classe Livro com propriedades como título, autor e número de páginas. Esses detalhes são abstraídos, permitindo que você trabalhe com livros de maneira mais eficiente.

2) Encapsulamento

- O encapsulamento é o conceito de ocultar os detalhes internos de uma classe e expor apenas uma interface pública.

- Você define métodos e propriedades como públicos, privados ou protegidos. Isso ajuda a proteger o estado interno de um objeto e a evitar acesso não autorizado.

- Por exemplo, uma classe ContaBancaria pode ter métodos públicos para depositar e sacar dinheiro, mas o saldo real é mantido privado.

3) Herança

- A herança permite que uma classe herde características (métodos e propriedades) de outra classe.

- Você pode criar uma nova classe (subclasse) com base em uma classe existente (superclasse). A subclasse herda os comportamentos da superclasse e pode adicionar ou modificar funcionalidades.

- Por exemplo, uma classe Carro pode herdar da classe Veiculo, aproveitando os métodos comuns a todos os veículos.

4) Polimorfismo

- O polimorfismo permite que objetos de diferentes classes sejam tratados de maneira uniforme

- Você pode usar uma interface comum para trabalhar com objetos de tipos diferentes. Isso promove a flexibilidade e reutilização de código.

- Por exemplo, uma interface Imprimivel pode ser implementada por várias classes (como Livro, Contrato e Revista). Cada classe fornece sua própria implementação do método imprimir, mas todas podem ser tratadas de forma polimórfica.

Interface

Muitos dos métodos dos carros são comuns em vários automóveis. Tanto um carro quanto uma motocicleta são classes cujos objetos podem acelerar, parar, acender o farol etc., pois são coisas comuns a automóveis. Podemos dizer, então, que ambas as classes "carro" e "motocicleta" *são* "automóveis".

Quando duas (ou mais) classes possuem comportamentos comuns que podem ser separados em uma outra classe, dizemos que a "classe comum" é uma *interface*, que pode ser "herdada" pelas outras classes. Note que colocamos a interface como "classe comum", que pode ser "herdada" (com aspas), porque uma interface não é exatamente um classe, mas sim um conjunto de métodos que todas as classes que herdarem dela devem possuir (implementar) - portanto, uma interface não é "herdada" por uma classe, mas sim *implementada*. No mundo do desenvolvimento de software, dizemos que uma interface é um "contrato": uma classe que implementa uma interface deve fornecer uma implementação a **todos** os métodos que a interface define, e em compensação, a classe implementadora pode dizer que ela é do tipo da interface. No nosso exemplo, "carro" e "motocicleta" são classes que implementam os métodos da interface "automóvel", logo podemos dizer que qualquer objeto dessas duas primeiras classes, como um Honda Fit ou uma motocicleta da Yamaha, são automóveis.

Um pequeno detalhe: uma interface não pode ser herdada por uma classe, mas sim implementada. No entanto, uma interface pode herdar de outra interface, criando uma hierarquia de interfaces. Usando um exemplo completo com carros, dizemos que a classe "Honda Fit Cross" herda da classe "Honda Fit", que por sua vez herda da classe "Carro". A classe "Carro" implementa a interface "Automóvel" que, por sua vez, pode herdar (por exemplo) uma interface chamada "MeioDeTransporte", uma vez que tanto um "automóvel" quanto uma "carroça" *são* meios de transporte, ainda que uma carroça *não seja* um automóvel.

Design Patterns

- Alguns problemas aparecem com tanta frequência em POO que suas soluções se tornaram padrões de designs de sistemas e modelagem de código orientado a objeto, a fim de resolvê-los. Esses padrões de projeto, ou design patterns, nada mais são do que formas padronizadas de resolver problemas comuns em linguagens orientadas a objetos.