Fabio de Moura Camargo Neto
github.com/bio353

Introdução à POO

Introdução

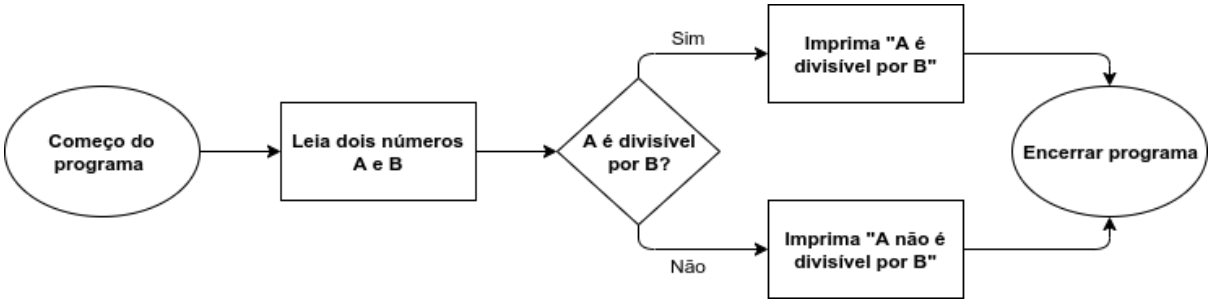
Definição de POO

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 de programação e, entre eles, estão a **programação orientada a objetos** (**POO**) e a programação estruturada. Quando começamos a utilizar linguagens como `Java`, `C#`, `Python` e outras que possibilitam o paradigma orientado a objetos, é comum errarmos e aplicarmos a programação estruturada achando que estamos usando recursos da orientação a objetos.

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)

A diferença principal é que na programação estruturada, um programa é tipicamente escrito em uma única rotina (ou função) podendo, é claro, ser quebrado em subrotinas. Mas o fluxo do programa continua o mesmo, como se pudéssemos copiar e colar o código das subrotinas 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 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.

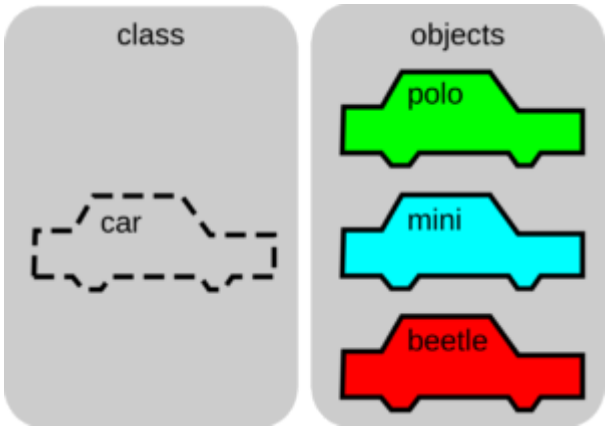
A programação orientada a objetos surgiu como uma alternativa a essas características da programação estruturada. O intuito da sua criação também foi o de aproximar o manuseio das estruturas de um programa ao manuseio das coisas do mundo real, daí o nome "objeto" como uma algo genérico, que pode representar qualquer coisa tangível.

Esse novo paradigma se baseia principalmente em dois conceitos chave: **classes** e **objetos**. Todos os outros conceitos, igualmente importantes, são construídos em cima desses dois.

Princípio da Abstração

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

5. 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.
6. 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.
7. 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".

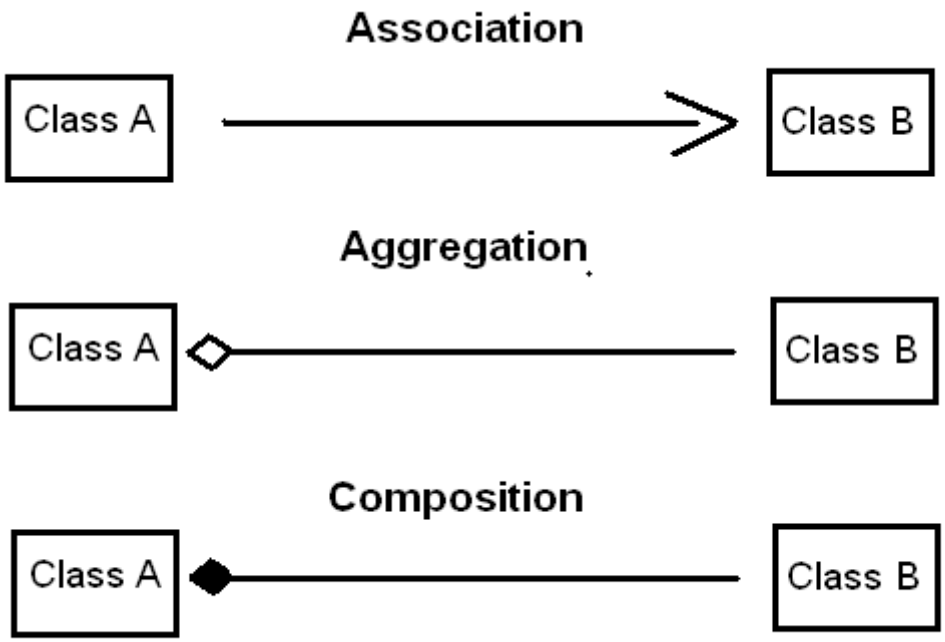


8. Assim, abstraindo um pouco a analogia, uma classe é um conjunto de características e comportamentos que definem o conjunto de objetos pertencentes à essa classe.
9. Repare que a classe em si é um conceito abstrato, como um molde, que se torna concreto e palpável através da criação de um objeto. Chamamos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

```
public class Carro {
    Double velocidade;
    String modelo;

    public Carro(String modelo) {
        this.modelo = modelo;
        this.velocidade = 0;
    }

    public void acelerar() {
        /* código do carro para acelerar */
    }
```

```
    public void frear() {
        /* código do carro para frear */
    }

    public void acenderFarol() {
        /* código do carro para acender o farol */
    }
}
```

Relacionamento entre Classes



▼ **Associação**

Ocorre quando tenho uma relação qualquer com outro objeto. Aqui, o objeto é instanciado fora da classe. Geralmente, não há uma relação do tipo “todo” e “parte” muito bem definida.

```
public class Pessoa {
    public void usaCelular(Celular celular) {
        /* código da função */
    }

    /* restante do código da classe */
}

public class Celular {
    /* código da classe */
}
```

▼ **Agregação**

Ocorre quando um objeto que cria uma copia de outro para si. Por exemplo, uma pessoa tem um marca-passo dentro de seu corpo, mas o objeto foi construído fora do corpo dela para depois ser colocado dentro.

```
public class Pessoa {
    MarcaPasso marcaPasso;

    /* restante do código da classe */
}

public class MarcaPasso {
    /* código da classe */
}
```

▼ **Composição**

Ocorre quando um objeto tem outro objeto intrincicamente em si. Por exemplo, uma pessoa tem coração. Por isso, o objeto é instanciado diretamente no escopo da classe.

```
public class Pessoa {
    Coracao coracao = new Coracao();

    /* restante do código da classe */
}

public class Coracao {
    /* código da classe */
}
```

Agradecimentos Especiais

Obrigado à alura por me poupar tempo com esse artigo maravilhoso e ao Brunão da programação por me ajudar com o material de associação, agregação e composição.