**Orientação a objetos**

**Paradigmas:** são abordagens de como resolver problemas, baseadas em uma teoria ou conjunto de princípios definidos. Se baseiam normalmente em alguma teoria matemática ou computacional.

**Paradigma Orientado a Objetos:** princípio de espelhar o mundo real através de uma estrutura de objetos com características e ações, que interagem uns com os outros.

Ex:

Objeto - Gato:

* sexo;
* idade;
* pelagem;
* status:
* castrado;
* vacinado;
* vermifugado;

Comportamentos:

* miar;
* comer;
* dormir;
* limpar;
* brincar;
* bolinha;
* laser;

Funções executadas dentro de um objeto, se refere ao comportamento do mesmo e são chamadas de métodos.

Não é prático criar um objeto para cada novo gato, além de ter de criar muitas variáveis, por isso se cria um modelo de objeto, rotulando o tipo de cada variável presente. Esse modelo chamamos de classe.

Ex:

const Gato = {

nome: stringNome,

nascimento: stringData,

pelagem: stringPelagem,

status: {

castrado: boolCastrado,

vacinado: boolVacinado,

vermifugado: boolVermifugado

},

}

[{

nome: “Abelhinha”,

nascimento: “22/11/2012”,

pelagem: “creme”,

status: {

castrado: true,

vacinado: true,

vermifugado: true

},

},

{

nome: “Abelhinha”,

nascimento: “22/11/2012”,

pelagem: “creme”,

status: {

castrado: true,

vacinado: true,

vermifugado: true

},

}]

Cada novo objeto criado a partir de um modelo de classe é chamado de instância.

**CLASSE:**

Modelo de objeto, contendo atributos e métodos. rotulando o tipo de cada variável presente

Atributo: características dos objetos.

Métodos: funções que representam comportamentos típicos do objeto.

**OBJETO:**

É um elemento que representa alguma entidade, da área de interesse do problema que está sendo analisado. Seus comportamentos e características buscam se espelhar no mundo real.

**HERANÇA**

Uma classe pode herdar os dados de outra classe, ou seja, a classe filha (subclasse) terá os mesmos atributos e métodos da classe pai (superclasse).

**ENCAPSULAMENTO:**

Serve para limitar o acesso dos métodos e atributos de uma classe, para evitar que alterações indesejadas ocorra. Para isso, delimita-se o acesso como público, privado ou protegido, de acordo com o nível de acesso desejado.

**POLIMORFISMO:**

A subclasse pode conter o mesmo método da superclasse, porem com parâmetros ou comportamentos diferentes.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**SOLID**

**S — Single Responsiblity Principle**(Princípio da responsabilidade única)

Cada classe tem apenas uma única responsabilidade. Por exemplo, uma classe usuário que é responsável pelos dados de um usuário não deve ser responsável por gerenciar pagamentos também, pois estaria com mais de uma responsabilidade, o ideal seria cria ruma nova classe.

**O — Open-Closed Principle**(Princípio Aberto-Fechado)

Objetos ou entidades precisam estar abertos para extensão, ou seja, caso necessário criar uma nova funcionalidade, uma nova entidade ou objeto deve ser criado a partir do mesmo, para que não altere o código da superclasse.

**L — Liskov Substitution Principle**(Princípio da substituição de Liskov)

Cada subclasse deve poder substituir a superclasse sem problemas, pois deve ter os mesmos métodos e atributos.

**I — Interface Segregation Principle**(Princípio da Segregação da Interface)

Cliente não deve ser forçado a depender de interface que ele não usa. Por exemplo, a classe juliana é vegetariana, ela não deve ser forçada a ter ou depender de alguma interface de carne.

**D — Dependency Inversion Principle**(Princípio da inversão da dependência)

* Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender da abstração.
* Abstrações não devem depender de detalhes. Detalhes devem depender de abstrações.