## **Pesquisa - Herança e polimorfismo - POO**

## **O que é Programação Orientada a Objetos?**

Programação orientada a objetos é um modelo de programação onde diversas classes possuem características que definem um objeto da vida real.

Esse modelo de programação foi criado com o intuito de aproximar o mundo real do mundo virtual.

### **Características da POO**

A POO trabalha utilizando classes, pois são nelas que são criados os comportamento dos objetos (métodos) e seus estados (atributos).

Um objeto é formado por um conjunto de propriedades (variáveis) e procedimentos (métodos).

### **Classes**

Uma classe é um modelo que define as características e comportamentos de um tipo de objeto, servindo como um guia para criar objetos específicos.

Define o que um objeto de um determinado tipo (por exemplo, um carro, um livro) pode ter (atributos) e o que ele pode fazer (métodos).

### **Objetos**

Objeto é uma entidade autônoma que encapsula dados e comportamentos. São utilizados para representar características e comportamentos de algo do mundo real.

Sua estrutura combina variáveis, funções e estruturas de dados.

Os objetos se dividem em duas categorias principais: propriedades e métodos.

* Propriedades:   
  Conhecidas como atributos, são elas que incluem as informações sobre o objeto, como por exemplo, as propriedades (atributos) de um carro seriam a sua cor, modelo, ano de fabricação e entre outros.
* Métodos:  
  Os métodos definem as operações que podem ser executadas no objeto, ou seja, suas ações. Os métodos de um carro, por exemplo, poderiam ser acelerar, frear, virar e entre outros.

## **Herança**

### Conceito teórico

A herança é um mecanismo que permite que uma classe herde características de outra, como atributos e métodos, incentivando o compartilhamento de funcionalidades, a reutilização de código e a criação de hierarquias.

### Vantagens

Por meio da herança é possível criar novas classes que herdam características de uma classe pai. Isso faz com que a necessidade de escrever um código semelhante seja eliminada, tornando o programa eficiente e consistente, além de deixar o código limpo.

Com a utilização desse pilar, é possível que a manutenção do código também se torne mais fácil, pois as alterações que são realizadas na classe pai são diretamente passadas para as classes filhas, fazendo com que a manutenção de item por item seja desnecessária.

Como é possível criar hierarquias de classes com a herança, o código se torna mais organizado.

### Exemplo prático - Python

class Veiculo:

def \_\_init\_\_(self, marca, modelo):

self.marca = marca

self.modelo = modelo

def acelerar(self):

print("Acelerando o veículo...")

class Carro(Veiculo):

def \_\_init\_\_(self, marca, modelo, cor):

super().\_\_init\_\_(marca, modelo)

self.cor = cor

def ligar\_radio(self):

print("Ligando o rádio do carro...")

### 

## **Polimorfismo**

### Conceito teórico

Polimorfismo significa que um mesmo método pode se comportar de maneiras diferentes dependendo do objeto que está o usando, ou seja, você pode utilizar o mesmo nome de método, mas a ação que ele irá realizar depende de qual das classes está chamando esse método.

É um pilar fundamental da programação orientada a objetos, pois oferece flexibilidade e uniformidade no tratamento de objetos.

### Vantagens

O polimorfismo permite que o código seja mais flexível e adaptável a novas situações e requisitos, pois é possível adicionar novas classes e comportamentos sem realizar alterações no código existente.

Com esse pilar, também é possível realizar alterações no comportamento de objetos sem a necessidade de alterar o corpo do código, tornando a manutenção muito mais fácil.

A simplicidade também é uma das vantagens do polimorfismo, pois é possível tratar objetos de diferentes classes como se fossem do mesmo tipo, o que torna o código mais fácil de ser compreendido.

Oferece uniformidade no tratamento de objetos, permitindo que todos sejam tratados de maneira uniforme ao utilizar a mesma interface ou método, simplificando o código, pois não há necessidade de saber o tipo específico do objeto para invocar um método.

### Exemplo prático - Python

class Animal:

def emitir\_som(self):

print("O animal emite um som.")

class Cachorro(Animal):

def emitir\_som(self):

print("O cachorro late: Au au!")

class Gato(Animal):

def emitir\_som(self):

print("O gato mia: Miau!")

animais = [Cachorro(), Gato()]

for animal in animais:

animal.emitir\_som()

### 

## **Diferenças e semelhanças entre os pilares**

Tanto a herança quanto o polimorfismo são pilares da programação orientada a objeto. Por mais que eles possam operar juntos, eles possuem algumas diferenças e semelhanças.

### Diferenças

A diferença entre os dois pilares está principalmente em seus objetivos, já que na herança, o objetivo é permitir que uma classe herde atributos e métodos de outra, e no polimorfismo, o objetivo é fazer com que os objetos respondam a chamada de métodos de maneiras diferentes.

### Semelhanças

Ambos os pilares permitem que haja uma reutilização de código existente, facilitando a criação e o tratamento de classes. Além disso, os dois pilares tornam o código fácil de ser adaptado, deixando-o mais flexível.

Também é possível enxergar semelhanças quando se trata de hierarquia de classes, já que a herança cria essa hierarquia e o polimorfismo trabalha dentro dela.

### 

## **Bibliografia:**

* <https://www.devmedia.com.br/programacao-orientada-a-objetos-e-programacao-estruturada/32813>
* <https://meuartigo.brasilescola.uol.com.br/informatica/programacao-orientada-objetos.htm>
* <https://ebaconline.com.br/blog/objeto-na-programacao-seo>
* <https://www.dio.me/articles/vantagens-da-programacao-orientada-a-objetos-poo>
* <https://adrianosantostreina.com.br/blog/poo-2o-pilar-heranca/>
* <https://adrianosantostreina.com.br/blog/poo-3o-pilar-polimorfismo/>