Programação Orientada Objetos II

Nome: BRUNO DE LIMA SANTOS
Professor: José Eduardo Soares De Lima

Colégio ULBRA São Lucas

29/08/2024

RESUMO

Neste trabalho de pesquisa, você encontrará pesquisas relacionadas a Orientações Objetos, será explicado confeitos e finalidades básicos de Orientação objeto, A orientação a objetos é um paradigma de programação que se baseia na organização e estruturação do código em torno de objetos, que são representações de entidades do mundo real. Esses objetos possuem características (atributos) e comportamentos (métodos) que podem ser definidos e manipulados de forma independente. A orientação a objetos visa a modularidade, reutilização de código e facilita o desenvolvimento de sistemas complexos.Serão apresentados os seguintes conceitos fundamentais da programação orientada a objetos: classes,herança,atributos, métodos, polimorfismo , variáveis, constantes, objetos, sobrecarga, sobrecarga e encapsulamento. Cada um desses tópicos será explorado para fornecer uma compreensão abrangente dos princípios e práticas da programação orientada a objetos.

PALAVRAS CHAVES: **classes, herança, atributos, métodos, polimorfismo , variáveis, constantes, objetos, sobrecarga, sobreposição e encapsulamento.**

INTRODUÇÃO

Este trabalho de pesquisa visa fornecer uma compreensão abrangente dos princípios da programação orientada a objetos (POO). A programação orientada a objetos é um paradigma essencial para o desenvolvimento de software, focando na criação e manipulação de objetos que representam entidades do mundo real ou conceitual.

O objetivo geral deste estudo é analisar e descrever os conceitos fundamentais da POO, incluindo classes, herança, atributos, métodos, polimorfismo, variáveis,

constantes, objetos, sobrecarga, sobreposição e *encapsulamento. A pesquisa busca explorar como esses conceitos se inter-relacionam e contribuem para a construção de sistemas de software mais robustos e flexíveis.*

O problema central abordado é entender como a POO organiza e estrutura o código de forma eficiente, permitindo a reutilização e a manutenção do software. A importância deste estudo reside na capacidade da POO de resolver problemas complexos de programação através da abstração e da modularidade, facilitando o desenvolvimento e a gestão de grandes sistemas.

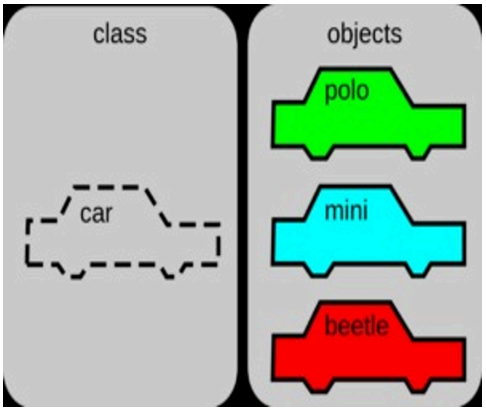
O trabalho está estruturado em seções que detalham cada conceito-chave da POO, começando com uma introdução geral à classe e à herança, seguida pela análise de atributos e métodos, e a exploração de polimorfismo, variáveis e constantes. A pesquisa também aborda o encapsulamento e a criação de objetos, destacando sua importância na construção de software orientado a objetos.

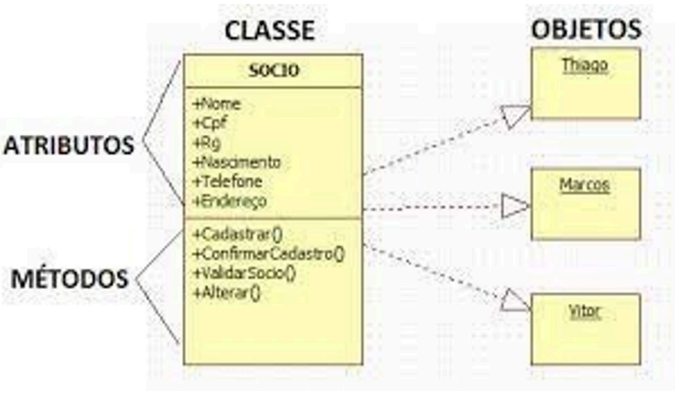
As principais etapas do trabalho são apresentadas a seguir:

1. *Conceito de Classes e Herança*
2. *Atributos e Métodos*
3. *Polimorfismo (*sobrecarga, sobrecaposição)
4. *Variáveis e Constantes*
5. *Encapsulamento*

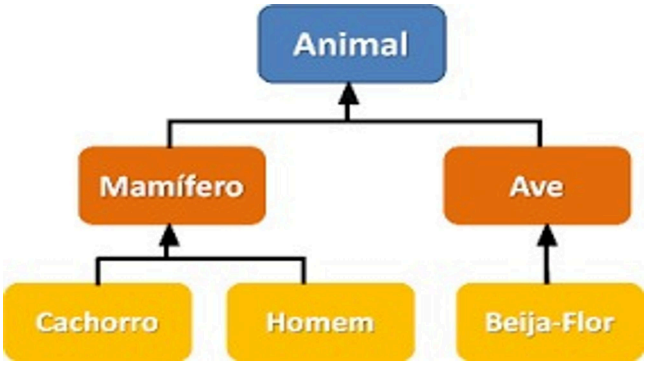
ADMINISTRAÇÃO

***Classes** são definições de tipos de dados que encapsulam atributos (propriedades) e métodos (comportamentos) relacionados. Uma classe serve como um modelo para criar objetos, que são instâncias concretas da classe.* Em POO, você define uma classe com variáveis e métodos, e então cria objetos a partir dessa classe. A classe define o que o objeto irá conter e como o objeto irá se comportar.





A **herança** é um mecanismo que permite que uma classe (subclasse) herde atributos e métodos de outra classe (superclasse), facilitando a reutilização de código e a extensão de funcionalidades, permitindo que as classes herdem campos e métodos de outras classes. A classe que herda os campos e métodos é chamada de subclasse, e a classe da qual a subclasse herda é chamada de superclasse. Usaremos de exemplo, a classe “Animal”, que tem como herança, “Mamífero e Ave”.



EXEMPLO:

```
class Superclasse {
    // Atributos e métodos da superclasse
}

class Subclasse extends Superclasse {
    // Atributos e métodos adicionais ou modificados da
subclasse
}
```

Exemplo Prático - Herança em Veículos:

```
class Veiculo {
    private int velocidade;

    public Veiculo() {
        this.velocidade = 0;
    }

    public void acelerar(int quantidade) {
        velocidade += quantidade;
```

```
    }

    public void frear(int quantidade) {
        velocidade -= quantidade;
    }
}

class Carro extends Veiculo {
    private int numeroDePortas;

    public Carro(int numeroDePortas) {
        this.numeroDePortas = numeroDePortas;
    }

    public int getNumeroDePortas() {
        return numeroDePortas;
    }
}

class Motocicleta extends Veiculo {
    public void empinar() {
        System.out.println("A motocicleta está empinando!");
    }
}
```

Polimorfismo, derivado do grego "muitas formas", é um dos pilares fundamentais da Programação Orientada a Objetos. Ele se refere à capacidade de uma única função ou método operar de maneiras diferentes dependendo do contexto ou dos objetos com os quais interage. Isso é alcançado através de métodos de sobrecarga e sobrescrita .

A **sobrescrita**, ou overriding, ocorre quando uma subclasse fornece uma implementação específica para um método que já está definido em sua superclasse. A implementação na subclasse é considerada uma "sobreescrita" da implementação na superclasse.

```
class Animal {    void som() {

      System.out.println("O animal faz um som");

    }

}
```

```
class Cachorro extends Animal {   @Override  void som() {
        System.out.println("O cachorro late");
    }
}
class Gato extends Animal {  @Override void som() {
     System.out.println("O gato mia");
    }
}
```

Sobrecarga, ou overloading, permite que uma classe tenha múltiplos métodos com o mesmo nome, mas com diferentes assinaturas. Isso permite que o mesmo método realize tarefas diferentes baseadas nos argumentos passados para ele.

```
class Calculadora {
    int somar(int a, int b) {
        return a + b;
    }

    double somar(double a, double b) {
        return a + b;
    }

    int somar(int a, int b, int c) {
        return a + b + c;
    }
}
```

Neste exemplo, o método somar é sobrecarregado para lidar com dois números inteiros, três números inteiros e dois números double.
Encapsulamento é a prática de esconder os detalhes de implementação de uma classe e apenas expor uma interface pública. Isso é feito para evitar que o estado do objeto seja alterado de maneira indesejada.O encapsulamento é um dos quatro princípios fundamentais da Programação Orientada a Objetos. Ele descreve a ideia de agrupar dados e os métodos que manipulam esses dados em uma única unidade, que é o objeto. Isso permite que o estado interno do objeto seja escondido do mundo exterior.

Os **métodos** acessores (getters) e modificadores (setters) são uma parte essencial do encapsulamento. Eles são usados para acessar e modificar, respectivamente, os atributos privados de uma classe.

Um **método acessor** é usado para obter o valor de um atributo privado. Ele retorna o valor do atributo ao chamador.

Um **método modificador** é usado para modificar o valor de um atributo privado. Ele aceita um valor como parâmetro e o atribui ao atributo.

Uma **variável** em programação funciona como uma "caixa" que armazena informações que podem ser modificadas enquanto o programa está em execução. Por exemplo, ao criar um jogo, você pode precisar registrar a pontuação dos jogadores. Para isso, você define uma variável chamada pontuação, onde o valor da pontuação será guardado.

Sempre que um jogador marca pontos, você pode acessar essa "caixa" no código e atualizar o valor armazenado. Assim, uma variável pode conter números, textos ou valores booleanos. Vale destacar que, embora uma variável possa mudar de valor,

ela só pode manter um valor de cada vez. A cada nova atribuição, o valor anterior é substituído pelo novo.

Agora, imagine que você tem uma caixa que armazena algo que nunca muda, como o número de dias em uma semana. Essa caixa representa uma constante. Uma **constante** é um valor que, uma vez definido, permanece inalterado durante a execução do programa, como constantes matemáticas ou endereços de memória. Para facilitar a identificação dessas constantes no código, geralmente usamos letras maiúsculas. A forma de definir constantes pode variar conforme a linguagem de programação: em algumas, elas são declaradas no início do programa; em outras, podem ser especificadas em módulos separados.

Utilizar constantes em um programa oferece uma grande vantagem: a facilidade de manutenção e ajuste do código. Ao empregar constantes para representar valores fixos, você pode modificar esses valores de forma centralizada, o que facilita a alteração de seu comportamento em todo o código.

CONSIDERAÇÕES FINAIS

O estudo da Programação Orientada a Objetos (POO) e fundamental para a construção de sistemas. Os conceitos abordados — **classes**, **herança**, **atributos**, **métodos**, **polimorfismo (sobrecarga e sobrescrita)**, **variáveis**, **constantes** e **encapsulamento** — formam a base sobre a qual a POO se estrutura. As classes fornecem um modelo para a criação de objetos, enquanto a herança permite a extensão e reutilização de código, promovendo uma hierarquia organizada. A definição de atributos e métodos é crucial para descrever e manipular o estado e o comportamento dos objetos. O polimorfismo, através da sobrecarga e sobrescrita, permite que métodos se comportam de diferentes formas, aumentando a flexibilidade do código.

Variáveis e constantes desempenham papéis distintos mas complementares: variáveis armazenam valores que podem mudar, enquanto constantes mantêm valores fixos, garantindo integridade e previsibilidade no código. O encapsulamento, por sua vez, é essencial para a proteção dos dados e a exposição controlada da funcionalidade de uma classe, melhorando a modularidade e a manutenção do software.

A prática e a compreensão desses conceitos permitem aos desenvolvedores criar sistemas mais modularizados, reutilizáveis e de fácil manutenção. A POO, com sua ênfase na abstração e na modularidade, facilita a resolução de problemas complexos e a gestão de grandes projetos de software.

Em conclusão, a programação orientada a objetos não apenas melhora a organização e a estruturação do código, mas também promove uma abordagem sistemática para o desenvolvimento de software. A correta aplicação desses conceitos é fundamental para criar soluções eficientes e escaláveis, destacando a importância da POO no cenário atual da programação.

REFERÊNCIAS

https://bead-edam-a70.notion.site/Programa-o-Orientada-a-Objetos-I-d658d71f7f584d15aee1f761a4b2f8d8#274a6a4205c74674aeee67a2adb657dd

https://www.devmedia.com.br/principais-conceitos-da-programacao-orientada-a-objetos/32285

https://www.dio.me/articles/orientacao-a-objetos-resumo-rapido

https://hub.asimov.academy/blog/variaveis-constantes-programacao/