Introdução: A programação orientado a objetos (POO) é um paradigma de desenvolvimento de software baseado no conceito de objetos, que representam entidades do mundo real e contêm dados (atributos) e comportamentos (métodos).

+ Origens

↳ Criada nos anos 1960, mas se popularizou nos anos 1990 com o crescimento da internet.

↳ Linguagens como SmallTalk, C++, Java e Python ajudaram a difundir o POO.

+ Vantagens da POO

↳ Modularidade - código organizado em classes e objetos reutilizáveis.

↳ Reutilização - código pode ser reaproveitado através da herança e composição

↳ Facilidade de manutenção: - Permite alteração sem grandes impactos no sistema.

↳ Escalabilidade - Ideal para sistemas complexos, pois facilita expansão.

Diferença entre Programação estruturada e POO

+ Programação Estruturada (Procedural)

- Código separado em funções
- Dados são manipulados separadamente por diferentes partes do código
- Exemplo em C

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

+ Programação Orientado a objetos (POO)

- Dados e métodos são agrupados em objetos
- Encapsula comportamento e dados dentro de classes
- Exemplo JAVA

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

Marcelo da Silva Mello
221125-1

+ Comparação

Característica	Programação Estruturada	Programação Orientada a objetos
Organização	Funções e variáveis separadas	Objetos que encapsulam dados e métodos
Manutenção	~~Baixa~~ Difícil	Mais fácil, pois objetos controlam seus dados
Reutilização	Baixa	Alta (herança e composição)
Flexibilidade	Baixa	Alta, pois classes podem se adaptar facilmente

◉ O que é um objeto ?

+ Um objeto é uma entidade que possui atributos e comportamentos

↳ Exemplo: Objeto "carro"

Atributo	Valor
cor	AZUL
modelo	SEDAN
ano	2023

✓ Métodos: acelerar(), frear(), ligar().

- Mensagens entre objetos

↳ Os objetos interagem enviando mensagens uns para os outros

```
JAVA
carro.ligar(); // o objeto 'carro' executa o método ligar()
```

◉ O que é uma classe ?

↳ Uma classe é um molde para criar objetos

+ Exemplo de classe em JAVA:

```
class carro {
    String cor;
    String modelo;
    int ano;

    void ligar() {
        System.out.println("Carro ligado!");
    }
}
```

+ Criando objetos a partir de classes

```
Carro meuCarro = new carro();
meuCarro.cor = "AZUL";
meuCarro.ligar(); // Saída: "carro ligado!"
```

∴ Agora meuCarro é um objeto baseado na classe carro.

Encapsulamento e ocultação de dados

+ "Proteger os dados do acesso direto e garantir que sejam manipulados corretamente!"

- Atributos privados (private) e métodos públicos (public)

```
class Pessoa {
    private String nome;
    Public void setNome(String n) { nome = n; }
    public String getNome() { return nome; }
}
```

√ vantagem: garante que os dados só possam ser alterados de forma controlada.

Herança

+ "Permite que uma classe herde características de outra, evitando repetição de código."

↳ Classe "mãe" (superclasse) → Classe "filho" (subclasse)

↳ A subclasse herda métodos e atributos da superclasse.

```
class Animal {
    void comer() { System.out.println("Comendo..."); }
}
class Cachorro extends Animal {
    void latir() { System.out.println("AU, AU!"); }
}
```

√ Uso

```
Cachorro c = new Cachorro();
c.comer(); // herdado de animal
c.latir(); // Específico de cachorro
```

+ Regra: Uma subclasse é um tipo da superclasse (Cachorro é um animal).

○ Polimorfismo

↳ "Objetos diferentes podem responder de formas diferentes ao mesmo comando."

- Exemplo JAVA

```
Classe Animal {
    Void fazerSom() { System.out.println("Som genérico"); }
}
Class Cachorro estende Animal {
    Void fazerSom() { System.out.println("latido"); }
}
Classe Gato estende Animal {
    Void fazerSom() { System.out.println("miau"); }
}
```

✓ Uso

```
Animal a1 = new Cachorro();
Animal a2 = new Gato();
a1.fazerSom(); // latido
a2.fazerSom(); // miau
```

+ Vantagens: Podemos tratar Cachorro e Gato como Animal, facilitando a expansão do código.

○ Composição

↳ "objetos podem conter outros objetos."

- Exemplo: Um carro contém um motor (JAVA)

```
Classe Motor {
    Void ligar() { System.out.println("motor ligado"); }
}
Class Carro {
    Motor motor = new Motor();
    Void ligar() { motor.ligar(); }
}
```

✓ Uso:

```
Carro carro = new Carro();
carro.ligar(); // "motor ligado"
```

○ Regra: Composição usa "tem um" (carro tem um motor).

Conclusão

- Os conceitos fundamentais da POO ajudam a criar código mais modular, reutilizável e fácil de manter.

- Encapsulamento - Protege dados e evita acesso indevido
- Herança - Permite reutilizar código de forma hierárquica
- Polimorfismo - Objetos diferentes podem responder de formas diferentes ao mesmo método
- Composição - Objetos podem ser formados por outros objetos.

A POO é amplamente utilizada em sistemas modernos, facilitando a construção de softwares escaláveis, seguros e eficientes.