

Programação orientada a objetos.

- Classes e Objetos
- Métodos e Atributos
- Encapsulamento
- Herança
- Polimorfismo



```
expandVertical(true);  
sonal = new Table(scrolledComposite, SWT.BORDER { SWT.FULL_SELECTION);  
sonal.addSelectionListener(new SelectionAdapter() {  
    void widgetSelected(SelectionEvent e) {  
        PersonalContact [] mypersonalcontact = new PersonalContact ();  
        mypersonalcontact = myDatabaseConnection.getAllPersonalContact();  
        String[] Titles = {"Contact ID", "First Name", "Last Name", "Person  
        ScrolledComposite scrolledComposite = new ScrolledComposite(created  
        scrolledComposite.setBounds(325, 54, 319, 482);  
        scrolledComposite.setExpandHorizontal(true);  
        scrolledComposite.setExpandVertical(true);  
        loopIndex < Titles.length; loopIndex++) {  
            TableColumn(table_personal, SWT.NULL);
```

Prof. Gustavo Pires



Classes:

Um "molde" ou "modelo" que define a estrutura e o comportamento de um objeto.

Objetos:

Uma instância de uma classe. Cada objeto tem seus próprios dados, mas compartilha a mesma estrutura definida pela classe.



Atributos:

São as características ou propriedades de um objeto.

Métodos:

São as ações ou comportamentos que os objetos podem realizar.



Encapsulamento:

Princípio que controla o acesso aos atributos e métodos de uma classe, protegendo dados internos do objeto.

Atributos Privados: Usados para restringir o acesso direto a certos atributos. Em Python, isso é feito usando `_` ou `__`.

```
class Conta:
    def __init__(self, saldo):
        self.__saldo = saldo # Atributo privado
```



Herança:

Permite que uma classe herde atributos e métodos de outra classe (classe pai), promovendo a reutilização de código.

Polimorfismo:

Permite que métodos com o mesmo nome tenham diferentes comportamentos, dependendo da classe.



Vantagens da POO:

Reutilização de código: A herança e o encapsulamento promovem o reaproveitamento de lógica existente.

Modularidade: O código é dividido em partes menores e independentes, facilitando o desenvolvimento e a manutenção.

Facilidade de expansão: É simples adicionar novas funcionalidades sem alterar o código existente.

```
1 class Veiculo:
2     def __init__(self, marca, modelo):
3         self.marca = marca
4         self.modelo = modelo
5
6     def descricao(self):
7         return f"{self.marca} {self.modelo}"
8
9 class Carro(Veiculo):
10     def __init__(self, marca, modelo, portas):
11         super().__init__(marca, modelo)
12         self.portas = portas
13
14     def descricao(self):
15         return f"{super().descricao()} com {self.portas} portas"
16
17 # Criando objetos
18 veiculo = Veiculo("Honda", "Civic")
19 carro = Carro("Toyota", "Corolla", 4)
20
21 print(veiculo.descricao()) # Output: Honda Civic
22 print(carro.descricao()) # Output: Toyota Corolla com 4 portas
23
```