Atividade

Objetivo geral

Desenvolver um entendimento dos pilares da Programação Orientada a Objetos (POO) através de atividades práticas, construindo conceitos de forma progressiva e integrando-os ao final.

Estrutura da atividade

A atividade será dividida em cinco partes, cada uma focando em um pilar específico da POO:

1. **Encapsulamento**
2. **Abstração**
3. **Herança**
4. **Polimorfismo**
5. **Composição**

Cada parte contém exercícios práticos para reforçar o aprendizado. Ao final, todas as partes servirá para construir um sistema completo.

Parte 1: Encapsulamento

Objetivo: Compreender como proteger os dados internos de uma classe e controlar o acesso a eles.

Atividade:

- Crie uma classe `ContaBancaria` com os seguintes atributos privados:
  - `_numero_conta`
  - `_titular`
  - `_saldo`
- Implemente métodos públicos para:
  - Depositar um valor na conta.
  - Sacar um valor da conta (apenas se houver saldo suficiente).
  - Consultar o saldo atual.

Instruções:

- Use prefixo `_` nos atributos para indicar que são privados.
- Garanta que o saldo nunca possa ser alterado diretamente de fora da classe.

Parte 2: Abstração

Objetivo: Aprender a modelar objetos do mundo real, focando nos atributos e comportamentos essenciais.

Atividade:

- Crie uma classe abstrata `Veiculo` com os seguintes métodos abstratos:
  - `acelerar()`
  - `frear()`
  - `mostrar_velocidade()`
- Os atributos devem incluir:
  - `marca`
  - `modelo`
  - `velocidade` (iniciada em 0)

Instruções:

- Utilize o módulo `abc` para implementar a classe abstrata.
- Não implemente os métodos na classe `Veiculo`.

Parte 3: Herança

Objetivo: Entender como reutilizar código através de classes derivadas.

Atividade:

- Crie duas classes que herdam de `Veiculo`:
  - `Carro`
  - `Moto`
- Implemente os métodos abstratos em cada classe:
  - `acelerar` aumenta a velocidade em 10 km/h para `Carro` e 15 km/h para `Moto`.
  - `frear` reduz a velocidade em 10 km/h, não permitindo valores negativos.

Instruções:

- Cada classe pode ter comportamentos específicos além dos herdados.

Parte 4: Polimorfismo

Objetivo: Praticar como objetos de diferentes classes podem ser tratados de forma uniforme.

Atividade:

- Crie uma função `teste_veiculo(veiculo)` que:
  - Chama `acelerar()` duas vezes.
  - Chama `mostrar_velocidade()`.
  - Chama `frear()` uma vez.
  - Chama `mostrar_velocidade()` novamente.
- Teste essa função com instâncias de `Carro` e `Moto`.

Instruções:

- Observe como a mesma função opera com objetos de classes diferentes.

Parte 5: Composição

Objetivo: Aprender a construir classes complexas utilizando outras classes como componentes.

Atividade:

- Crie uma classe `Motor` com o atributo `potencia`.
- Modifique a classe `Carro` para que tenha um atributo `motor` que é uma instância de `Motor`.
- Adicione um método na classe `Carro` que exiba a potência do motor.

Instruções:

- Inicialize o `Motor` dentro do construtor de `Carro`.
- Mostre como acessar atributos da classe composta.

Entrega

Para cada parte do trabalho, entregue os seguintes itens:

1. **Código-fonte comentado**: O código deve estar bem organizado e com comentários explicativos para facilitar o entendimento.
2. **Testes**: Inclua pequenos testes que comprovem o funcionamento correto do código.
3. **Explicação do pilar**: Forneça uma breve explicação sobre como o conceito (pilar) foi aplicado no código.
4. **Versionamento**: Caso tenha contexto do Git, crie um repositório no GitHub e forneça o link do seu repositório na atividade.

A estrutura dos arquivos deve ser organizada da seguinte forma:

`/nome_da_pasta/`, por exemplo: `/polimorfismo/script.py`.

Dicas

- **Pratique cada parte individualmente:** Certifique-se de entender bem cada conceito antes de avançar.
- **Teste frequentemente:** Escreva pequenos trechos de código para testar funcionalidades específicas.
- **Comente seu código:** Use comentários para explicar partes importantes ou complexas do seu código.
