Lista 3 - Lógica Aplicada I

A lógica de programação orientada a objetos em Python representa uma evolução natural na forma de pensar e desenvolver sistemas. Enquanto a programação estruturada foca na sequência de comandos e funções, a orientação a objetos (POO) organiza o código em torno de entidades chamadas objetos, que reúnem dados (atributos) e comportamentos (métodos) de maneira integrada.

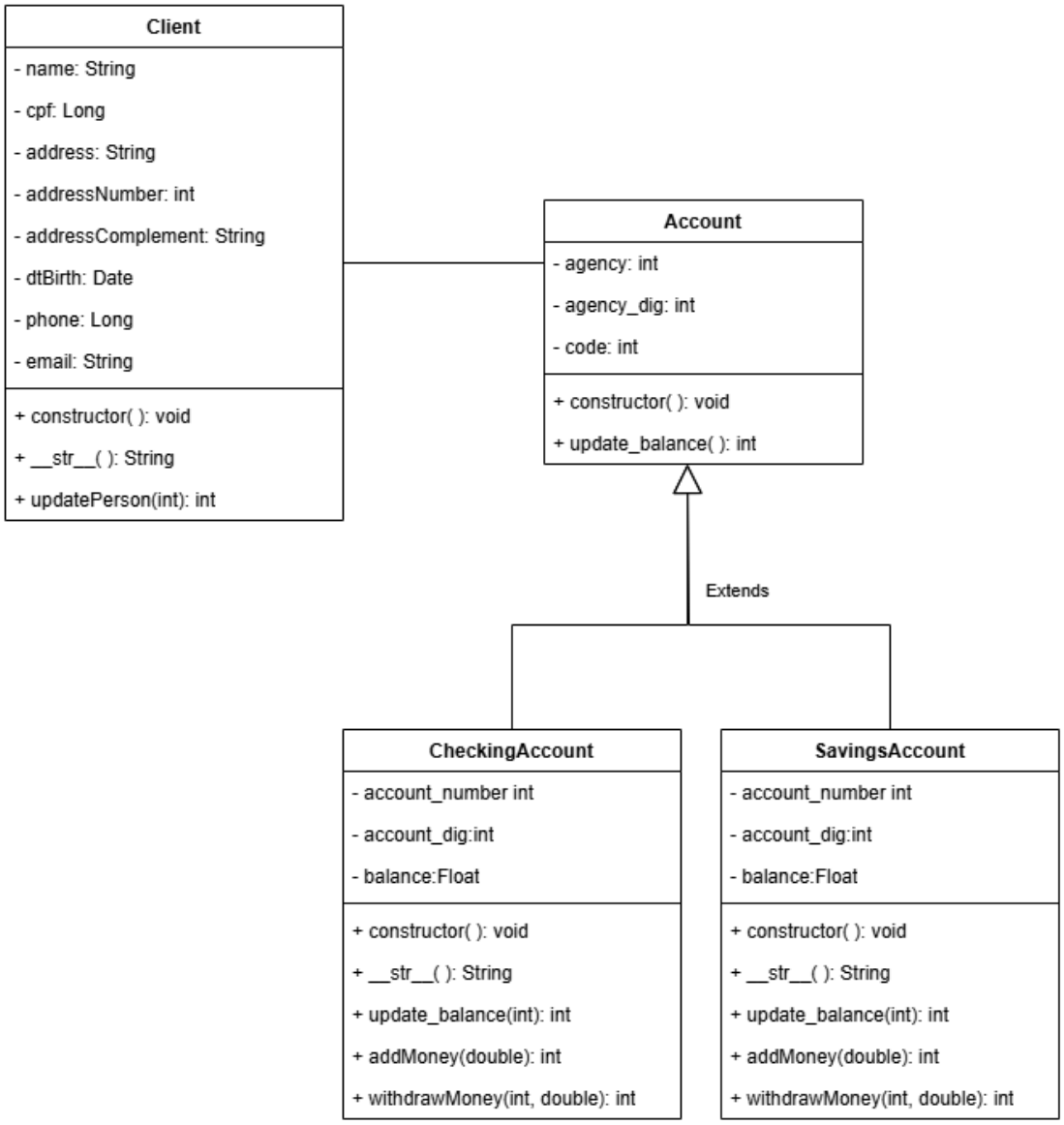
Em Python, a POO é aplicada de forma simples e poderosa, permitindo criar classes que funcionam como moldes para objetos do mundo real. Por exemplo, uma classe `Pessoa` pode conter atributos como `nome` e `idade`, além de métodos como **`falar()`** ou **`andar()`**. Essa estrutura torna o código mais modular, reutilizável e fácil de manter, facilitando o trabalho em projetos de maior complexidade.

Além disso, conceitos fundamentais como encapsulamento, herança e polimorfismo permitem criar hierarquias e comportamentos especializados, tornando o programa mais flexível e extensível. Em ambientes educacionais e profissionais, aprender lógica de programação com base em Python orientado a objetos ajuda o desenvolvedor a pensar de forma mais abstrata e organizada, fortalecendo a base para o desenvolvimento de aplicações modernas, desde sistemas web até inteligência artificial e jogos digitais.

Em síntese, a lógica de programação orientada a objetos em Python une clareza, simplicidade e poder expressivo, sendo uma das abordagens mais eficazes para compreender como transformar problemas do mundo real em soluções computacionais elegantes.

De acordo com o texto acima, resolva as questões a seguir de acordo com o paradigma de orientação à objetos:

1. **Seja o diagrama de classes UML a seguir:**



Implemente as classes Person (Pessoa), Account (Conta), CheckingAccount (ContaCorrente) e SavingsAccount (Poupança) utilizando o paradigma da orientação à objetos com Python.

2. **Considere a questão 1 e adicione a classe especializada ContaSalario ao modelo UML e ao programa em Python.**

3. **Modele uma classe base chamada Animal para representar características e comportamentos comuns a diferentes tipos de animais. Em seguida, utilizando os princípios da orientação a objetos em Python, crie duas classes especializadas: Cachorro e Gato, que herdem os atributos e métodos da classe Animal.**

★ Use UML para modelar as classes e depois implemente em python.

Cada classe deve conter:

- **Atributos:** nome, idade e som característico (por exemplo, "latido" ou "miado");
- **Métodos:** um método para exibir informações do animal e outro para emitir o som correspondente.

Por fim, crie objetos das classes Cachorro e Gato, demonstrando o uso da herança e a sobrescrita de métodos (polimorfismo).

4. **Modele uma classe base chamada `Veiculo` para representar características e comportamentos comuns a diferentes tipos de veículos. Em seguida, utilizando os princípios da orientação a objetos em Python, crie duas classes especializadas: `Carro` e `Moto`, que herdem os atributos e métodos da classe `Veiculo`.**

   ★ Use UML para modelar as classes e depois implemente em python.

   Cada classe deve conter:

   - **Atributos**: marca, modelo e ano;
   - **Métodos**: um método para exibir as informações do veículo e outro método `mover()` que descreva a forma de locomoção de cada tipo (por exemplo, “O carro está rodando nas quatro rodas” e “A moto está rodando em duas rodas”).

   Por fim, crie objetos das classes `Carro` e `Moto`, demonstrando o uso da herança e da sobrescrita de métodos (polimorfismo).

5. **Crie uma aplicação em Python orientado a objetos que modele diferentes tipos de televisores, utilizando os conceitos de classe abstrata e interface.**

   ★ Use UML para modelar as classes e depois implemente em python.

   Crie uma classe abstrata chamada `Televisor`, que possua os atributos:

   - `marca`
   - `Modelo`
   - `tamanho`

E os métodos:

- `exibir_informacoes()` — método concreto que mostra as informações básicas da TV;
- `ligar()` — método abstrato, que deve ser implementado nas subclasses;
- `Desligar()` — método abstrato, que deve ser implementado nas subclasses;
- `NumeroCanais()` — método abstrato, que deve ser implementado nas subclasses.
- `aumentarCanais()` — método abstrato, que deve ser implementado nas subclasses.
- `diminuirCanais()` — método abstrato, que deve ser implementado nas subclasses.
- `aumentarVolume()` — método abstrato, que deve ser implementado nas subclasses.
- `diminuirVolume()` — método abstrato, que deve ser implementado nas subclasses.

Crie uma interface (simulada em Python por meio de uma classe abstrata) chamada `Conectividade`, contendo o método abstrato:

- `conectar_internet()`.

Crie duas classes concretas, `SmartTV` e `TVComum`, que herdem de `Televisor` e implementem a interface `Conectividade`.

- O método `ligar()` deve ser sobrescrito para cada tipo (por exemplo, “Ligando a Smart TV com sistema operacional Android TV” e “Ligando a TV comum com controle remoto padrão”).

- O método `conectar_internet()` deve exibir uma mensagem indicando o tipo de conexão (por exemplo, “Conectando via Wi-Fi” na `SmartTV`, e uma mensagem informando “Esta TV não possui conexão com a internet” na `TVComum`).

Por fim, crie objetos das classes `SmartTV` e `TVComum`, demonstrando o uso dos métodos herdados e implementados.