**Atividade Prática 2 – APPOO Prof. Gabriela Nunes Lopes 2025/2**

**Alunos:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1 (Aula 11)** Crie o diagrama UML e ó código Python de um sistema simples de processamento de pagamentos que pode lidar com diferentes métodos de pagamento, como cartão de crédito, PayPal e transferência bancária, utilizando polimorfismo.

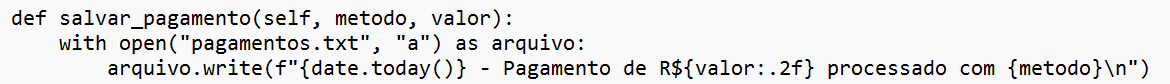
* Defina uma classe abstrata chamada MetodoPagamento que represente um método de pagamento genérico. Esta classe deve conter um método abstrato chamado processar\_pagamento() para processar um pagamento.
* Crie três subclasses:
  + CartaoCredito: Representa o pagamento com cartão de crédito.
  + PayPal: Representa o pagamento com PayPal.
  + TransferenciaBancaria: Representa o pagamento por transferência bancária.
* Implemente o método processar\_pagamento() em cada uma das subclasses para exibir uma mensagem informando que o pagamento foi processado com sucesso daquela subclasse específica, juntamente com o valor do pagamento.
* No programa principal, crie instâncias de cada tipo de método de pagamento e chame a função processar\_pagamento() para processar pagamentos de diferentes valores. Faça o cadastro e verificações de erros pelas classes ( não pelo main).

**2 (Aula 11)** Expanda o sistema de processamento de pagamentos adicionando a funcionalidade de salvar os pagamentos em um arquivo de texto. O sistema continuará a lidar com diferentes métodos de pagamento, como cartão de crédito, PayPal e transferência bancária, utilizando polimorfismo.

* Mantenha a classe abstrata MetodoPagamento e suas subclasses CartaoCredito, PayPal e TransferenciaBancaria conforme definidas anteriormente.
* Adicione um método salvar\_pagamento() em cada uma das subclasses para salvar as informações do pagamento em um arquivo de texto chamado pagamentos.txt. O método deve registrar a data do pagamento, o valor e o tipo de método de pagamento utilizado.
* Modifique a função processar\_pagamento() para chamar o método salvar\_pagamento() após processar o pagamento.

**Uma imagem contendo Texto

O conteúdo gerado por IA pode estar incorreto.**

****

**Texto

O conteúdo gerado por IA pode estar incorreto.**

**Exercício 3 (Aula 13) – Estatísticas de Jogador:** Use como base o exemplo da engenheira apresentado em sala. Você deve construir um código que compare desempenho entre jogadores de mesma função por meio de estatísticas. Para isso, sugere-se:

1 – Interface.py

Cria a classe EstatísticasInterface como método abstrato e cria o método abstrato de calcular estatísticas, como médias e máximos;

2 – Jogadores.py

Importa a interface para funcionar;

Calcula estatísticas necessárias para jogadores de cada posição. Por exemplo, para atacantes (classe Atacantes), calcula a média de gols por partida, e para goleiros (classe Goleiros), a média de defesas...

3 – Olheiro.py

Importa a interface e o módulo Type para funcionar

Na classe Olheiro, que recebe nome, crie o método da estatística calculada, que recebe o atributo do tipo EstatisticasInterface.

4 – Main.py

Importa Olheiro e as posições dos jogadores, por exemplo Atacantes e Goleiros;

Crie os objetos dos jogadores com seus atributos

Crie o objeto olheiro e chame os métodos para calcular as estatísticas do tipo de jogador

**Exercício 4 ( Aula 13) Playlist:** Crie um código e diagrama UML para fazer suas playlists de músicas favoritas (adicionar artista, nome da musica, álbum... a um arquivo txt que será sua playlist). Você é livre para fazer o programa como quiser, mas deve:

* Utilizar ao menos duas classes;
* Utilizar encapsulamento de alguma das variáveis;
* Utilizar get ou set em alguma parte do código;
* Utilizar interface para alguma das classes;
* Criar um arquivo Main separado que importe as classes e teste o código;
* Tipar todos os atributos e métodos;
* Deve ser possível poder mostrar a playlist para o usuário e/ou salvar num arquivo de texto;

Tente deixar o código o mais profissional possível, com análise de erros e boas práticas de programação.