Exercício 1 — Controle de Dispositivos Inteligentes

Conceitos: Herança + Polimorfismo
Nível: Básico/Intermediário

Enunciado:

Você foi contratado para criar um sistema de automação residencial. Todos os dispositivos (luzes, TVs, ar-condicionado) devem ser controlados por comandos genéricos como **ligar** e **desligar**, mas cada dispositivo se comporta de forma diferente.

Requisitos:

1. Crie uma classe base Dispositivo com o método abstrato ligar() e desligar().
2. Crie as subclasses Lampada, Televisao e ArCondicionado que sobrescrevam esses métodos com mensagens diferentes.
3. Crie uma lista com diferentes dispositivos e, usando um laço, chame ligar() e desligar() para todos (polimorfismo).

Exercício 2 — Sistema de Transporte

Conceitos: Herança + Encapsulamento
Nível: Intermediário

Enunciado:

Crie um sistema para gerenciar veículos de transporte público. Há um controle de acesso ao número de passageiros (não pode ultrapassar a capacidade máxima).

Requisitos:

1. Classe base Transporte com atributos: _capacidade (protegido), __passageiros (privado).
2. Método embarcar(qtd) que só permite embarcar se houver espaço.
3. Subclasses: Onibus, Metro, Trem, cada uma com capacidade diferente.
4. Mostre, ao final, a quantidade de passageiros embarcados em cada transporte.

Exercício 3 — Loja de Instrumentos Musicais

Conceitos: Herança + Polimorfismo + Encapsulamento
Nível: Intermediário/Avançado

Enunciado:

Você está criando um sistema para gerenciar uma loja de instrumentos musicais. Cada instrumento pode ser tocado de forma diferente e tem um valor de venda que deve ser protegido.

Requisitos:

1. Classe base Instrumento com atributos:
    - nome (público)
    - __preco (privado)
    - método tocar() (abstrato)
    - método get_preco() e set_preco(valor) (com validação: preço deve ser > 0).
2. Subclasses: Violao, Bateria, Piano, cada uma sobrescrevendo tocar().
3. Crie uma função que receba uma lista de instrumentos e execute tocar() para cada um.
4. Tente acessar diretamente o preço fora da classe e explique o erro (encapsulamento).

Exercício 4 — Personagens de RPG

Conceitos: Herança + Encapsulamento + Polimorfismo
Nível: Avançado

Enunciado:

Em um jogo de RPG, há diferentes tipos de personagens (guerreiro, mago, arqueiro) com métodos de ataque diferentes e pontos de vida que devem ser protegidos contra manipulação direta.

Requisitos:

1. Classe Personagem com:
    - _nome (protegido)
    - __vida (privado, começa com 100)
    - método receber_dano(dano) que subtrai vida (mínimo 0)
    - método get_vida()
    - método abstrato atacar()
2. Subclasses:
    - Guerreiro: ataque com espada (dano 15)
    - Mago: lança magia (dano 20)
    - Arqueiro: dispara flecha (dano 10)
3. Simule um combate onde personagens atacam uns aos outros.
4. Tente forçar uma alteração do atributo __vida diretamente fora da classe e mostre o erro.

Exercício 5 — Sistema de Pagamentos

Conceitos: Polimorfismo puro + Encapsulamento
Nível: Intermediário

Enunciado:

Você está criando um sistema de pagamentos. Cada método de pagamento processa o valor de forma diferente.

Requisitos:

1. Classe Pagamento com método processar_pagamento(valor) (abstrato).
2. Subclasses:
    - PagamentoCartao: imprime "Pagamento de R$X processado com cartão".
    - PagamentoPix: imprime "Pagamento de R$X processado via Pix".
    - PagamentoBoleto: imprime "Boleto de R$X gerado com sucesso".
3. Crie uma lista de formas de pagamento e execute processar_pagamento() para cada uma.