Exercício Prático:
Sistema de Notificações Corporativo

Contexto do Cenário

Você foi contratado para desenvolver o módulo de **Alertas** do backend de uma startup. O sistema precisa enviar notificações para os clientes sobre eventos críticos (ex: "Servidor caiu", "Pagamento aprovado").

Atualmente, o sistema envia apenas E-mails, mas o chefe de produto avisou que semana que vem vão querer enviar SMS, e no futuro, Push Notifications e WhatsApp.

Seu objetivo é criar uma arquitetura orientada a objetos que seja **flexível**, **segura** e **reutilizável**, aplicando os 4 pilares da POO.

Objetivos Técnicos

1. **Encapsulamento:** Proteger os dados sensíveis do usuário.
2. **Herança:** Criar uma base genérica para os canais de envio.
3. **Polimorfismo:** Permitir que o sistema troque de Email para SMS sem mudar o código principal.
4. **Composição:** Conectar o Usuário ao canal de envio dentro de um Sistema gerenciador.

Requisitos Detalhados

1. A Classe Usuario (O Cliente)

Esta classe representa quem receberá a notificação.

- **Atributos:**
  - nome: Público (pode ser acessado diretamente).
  - email: **Privado** (deve ser impossível acessar com usuario.email sem usar properties).
- **Métodos:**
  - Crie um **Getter** (@property) para ler o email.
  - Crie um **Setter** (@email.setter) para alterar o email.
  - **Regra de Negócio:** O Setter deve verificar se o novo email contém o caractere "@". Se não tiver, exiba um erro e não altere o valor.

2. A Classe CanalEnvio (A Base Abstrata)

Esta classe serve de molde para qualquer forma de comunicação.

- **Métodos:**
  - enviar(mensagem): Este método **não deve fazer nada** além de lançar um erro (raise NotImplementedError). Isso obriga qualquer programador que criar um novo canal a escrever sua própria lógica.

3. As Classes Email e SMS (As Implementações)

Estas classes herdam de CanalEnvio.

- **Herança:** Ambas devem ser subclasses de CanalEnvio.
- **Polimorfismo:** Ambas devem sobrescrever o método enviar(mensagem).
  - A classe Email deve imprimir: 📧 Enviando para servidor de email: [mensagem].
  - A classe SMS deve imprimir: 📱 Enviando para operadora telefônica: [mensagem].

4. A Classe SistemaAlerta (O Gerenciador)

Esta é a classe principal que usará **Composição**. Ela não herda de ninguém, ela *usa* os outros objetos.

- **Construtor (__init__):**
  - Deve receber dois parâmetros: um objeto usuario e um objeto canal. Salve-os em atributos self.
- **Método disparar(texto):**
  - Este método deve pegar o nome do usuário e chamar o método .enviar(texto) do canal configurado.

Roteiro de Testes (O que seu código deve rodar)

Para considerar o exercício concluído, seu script main deve realizar as seguintes operações com sucesso:

1. **Teste de Segurança:** Tente criar um usuário e mudar o email dele para uma string inválida (sem "@"). O sistema deve bloquear. Depois, mude para um email válido.
2. **Teste de Email:** Instancie um objeto Email, passe-o para o SistemaAlerta junto com o usuário e dispare uma mensagem.
3. **Teste de SMS (Polimorfismo):** Sem criar um novo usuário, instancie um objeto SMS. Crie um *novo* SistemaAlerta passando o *mesmo* usuário e o *novo* canal SMS. Dispare uma mensagem.
   - *Obs:* Note que você trocou a forma de envio sem precisar mudar nada na classe do usuário ou na lógica interna do sistema.

Dicas para a Solução

- Lembre-se que atributos privados em Python começam com __ (duplo sublinhado).
- Na **Composição**, você está passando *objetos inteiros* como argumentos para outra classe. Ex: sistema = SistemaAlerta(objeto_usuario, objeto_canal).
- No **Polimorfismo**, o SistemaAlerta não precisa saber se o canal é SMS ou Email. Ele só confia que o objeto passado tem um método .enviar().