Fase 1 — Heurística antes do código (mapa mental)

Problema escolhido
Queremos permitir que o sistema autentique o acesso do usuário utilizando diferentes métodos (biometria ou cartão/senha) de forma flexível, sem precisar alterar o cliente quando incluirmos novos métodos.

Quadro 1 — Procedural (onde surgem if/switch)

- Fluxo: recebe solicitação de acesso → if (modo == "biometria") então autentica via biometria → else autentica via cartão/senha → retorna sucesso/erro.
- Decisões embutidas no fluxo (ramificações por tipo de autenticação, horários ou permissões especiais).
- Sinais de dor: cada novo método de autenticação adiciona novos if/switch, aumentando casos de teste e risco de duplicação de código.

Quadro 2 — OO sem interface (quem encapsula o quê; o que ainda fica rígido)

- Encapsulamos cada método em classes concretas: BiometriaAuthenticator, CartaoSenhaAuthenticator.
- Um ControleAcesso orquestra o fluxo, mas ainda conhece as classes concretas ou mantém if/switch interno.
- Melhoras: coesão por método de autenticação; menos duplicação de lógica.
- Rigidêz remanescente: o cliente (ControleAcesso) continua decidindo qual classe usar; trocar método ou política exige alteração no cliente.

Quadro 3 — Com interface (contrato que permite alternar + ponto de composição)

- Contrato (o que): “autenticar usuário”.
- Implementações (como): BiometriaAuthenticator, CartaoSenhaAuthenticator (e futuras: TokenAuthenticator, RFIDAuthenticator).
- Ponto de composição: escolha do método via injeção de dependência ou fábrica externa ao cliente; política de seleção pode ser configurada separadamente (ex.: “horário noturno → biometria; emergência → cartão/senha”).
- Efeito: o cliente não muda quando adicionamos ou alternamos implementações; testes ficam mais simples usando dublês/mocks.
- 3 sinais de alerta previstos:
    1. Cliente muda ao trocar implementação (cheiro de acoplamento ao “como”).
    2. Ramificações espalhadas (if/switch por tipo ou política de autenticação em vários pontos do código).
    3. Testes lentos e frágeis (sem dublês, batendo em sensores físicos ou sistemas externos; difícil simular falhas/controlar cenários).