## 💻 Exercício Prático: Sistema de Processamento de Pagamentos

### Contexto

Você foi contratado para desenvolver parte de um sistema de **processamento de pagamentos** para uma plataforma de e-commerce. A empresa deseja que o sistema seja **flexível** e **facilmente extensível**, de modo que novos tipos de pagamento possam ser adicionados sem a necessidade de alterar o código existente.

Atualmente, o sistema deve processar **pagamentos via cartão de crédito** e **pagamentos via Pix**, mas espera-se que, no futuro, seja necessário incluir outras formas de pagamento (como boleto, PayPal, criptomoedas etc.).

---

### Objetivos de Aprendizagem

Com este exercício, você deverá demonstrar:

- Uso de **interfaces** e **classes abstratas** para definir comportamentos genéricos;
- Uso de **polimorfismo** e **ligação dinâmica** para permitir que o sistema trate diferentes tipos de pagamento de forma uniforme;
- Aplicação dos princípios de **reuso**, **extensibilidade** e **design for change**.

---

### Especificação

1. **Interface Pagamento**
    - Declare uma interface chamada Pagamento com os seguintes métodos:
    - public interface Pagamento {
    - void autorizarPagamento(double valor);
    - void realizarPagamento(double valor);
    - }
2. **Classe Abstrata PagamentoBase**
    - Crie uma classe abstrata chamada PagamentoBase que **implemente a interface Pagamento**.
    - Essa classe deve conter atributos comuns a todos os tipos de pagamento, como:
    - protected String idTransacao;

- protected double valor;
- Inclua um **método concreto** gerarRecibo() que imprime informações básicas sobre o pagamento.
- Inclua **métodos abstratos** que cada tipo de pagamento precisará implementar:
- protected abstract void validarDados();

3. **Classes Concretas**
    - Crie duas classes concretas:
        - PagamentoCartaoCredito
        - PagamentoPix
    - Ambas devem **herdar de PagamentoBase** e **implementar** os métodos abstratos e da interface.
    - Em PagamentoCartaoCredito, simule a validação de número do cartão e autorização de limite.
    - Em PagamentoPix, simule a validação da chave Pix e a confirmação instantânea.
4. **Classe de Teste: ProcessadorDePagamentos**
    - Crie uma classe ProcessadorDePagamentos com um método processarPagamentos(List<Pagamento> listaPagamentos).
    - Esse método deve percorrer a lista e chamar:
    - pagamento.autorizarPagamento(valor);
    - pagamento.realizarPagamento(valor);
    - pagamento.gerarRecibo();
    - Note que **não deve haver if ou instanceof** — o comportamento deve ser definido **dinamicamente** via polimorfismo.
5. **Teste o sistema**
    - No método main, crie uma lista com diferentes tipos de pagamento:
    - List<Pagamento> pagamentos = new ArrayList<>();
    - pagamentos.add(new PagamentoCartaoCredito(...));
    - pagamentos.add(new PagamentoPix(...));

- Passe essa lista para o ProcessadorDePagamentos e observe a **ligação dinâmica**:
  o método correto será chamado **em tempo de execução**, conforme o tipo real do objeto.

---

### 💡 Extensão (Design for Change)

Sem alterar o código do ProcessadorDePagamentos, adicione uma nova classe:

- PagamentoCriptomoeda, por exemplo.
  Observe que ela poderá ser integrada facilmente ao sistema — basta implementá-la e adicioná-la à lista.
  Esse é o **benefício direto da ligação dinâmica e do polimorfismo**.