Exercício 1 — Encapsulamento (Classe Produto)

Implemente a classe **Produto** com atributos privados **nome**, **preco** e **quantidadeEmEstoque**. Forneça getters e setters com validações: **preco** e **quantidadeEmEstoque** não podem ser negativos e **nome** não pode ser nulo ou vazio. Lance **IllegalArgumentException** em casos inválidos. Demonstre o uso criando instâncias, alterando valores válidos e tentando atribuições inválidas.

Exercício 2 — Encapsulamento com Validação de Regra (Desconto)

Estenda Produto com o método **aplicarDesconto(double porcentagem).** Permita apenas valores entre 0 e 50 (inclusive) e lance exceção (**IllegalArgumentException** ou **DescontoInvalidoException**) se a regra for violada. Mostre, em um **main** ou testes, o preço antes/depois do desconto e a reação a entradas inválidas.

Exercício 3 — Herança (Hierarquia de Funcionários)

Crie a classe base **Funcionario** com **protected String nome e protected BigDecimal salario** (com getters). Crie **Gerente** e **Desenvolvedor** que sobrescrevem **calcularBonus**(): 20% do salário para gerente e 10% para desenvolvedor. Garanta que salários sejam positivos. Em um programa, coloque diferentes funcionários em uma coleção do tipo **List<Funcionario>** e exiba o bônus de cada um.

Exercício 4 — Polimorfismo com Interface (IMeioTransporte)

Defina a interface **IMeioTransporte** com **acelerar()** e **frear().** Implemente **Carro**, **Bicicleta** e **Trem**, cada um com lógica própria de variação de velocidade e limites. No método principal, crie uma lista de **IMeioTransporte**, percorra e invoque **acelerar()/frear()** demonstrando polimorfismo. Trate operações inválidas com exceções apropriadas.

Exercício 5 — Abstração (Sistema de Pagamentos)

Implemente a classe abstrata **FormaPagamento** com **validarPagamento**() e **processarPagamento(BigDecimal valor).** Crie **CartaoCredito**, **Boleto** e **Pix** com validações específicas (ex.: número do cartão, formato de boleto, chave Pix). Simule o uso de cada forma por polimorfismo e trate erros de validação com exceções específicas (ex.: **PagamentoInvalidoException**).

Exercício 6 — Imutabilidade e Objetos de Valor (Carrinho de Compras)

Crie o objeto de valor imutável **Dinheiro** (valor **BigDecimal** e **enum Moeda**) com **equals**/**hashCode** coerentes. Modele **Produto**, **ItemCarrinho** e um **Carrinho** cuja lista de itens seja imutável: operações de adicionar/remover/aplicar cupom retornam

um novo carrinho. Valide **quantidades > 0**, proíba valores negativos e limite cupons a 30% com arredondamento bancário. Demonstre o fluxo completo em testes.

Exercício 7 — Generics (Repositório Genérico em Memória)

Defina **Identificavel** com **getId().** Crie **IRepository<T extends Identificavel, ID>** com **salvar, buscarPorId** (retorna **Optional<T>**), **listarTodos** e **remover**. Implemente **InMemoryRepository** com **Map**<**ID**, **T>,** garanta que **listarTodos** devolva cópia imutável e lance **EntidadeNaoEncontradaException** ao remover ID inexistente. Use com entidades como **Produto** e **Funcionario**.

Exercício 8 — Padrão Strategy (Cálculo de Frete com Lambdas)

Modele **CalculadoraFrete** com **calcular**(**Pedido**): **BigDecimal**. Crie estratégias **Sedex**, **Pac** e **RetiradaNaLoja** e permita injeção/troca da estratégia no **Pedido**. Acrescente uma estratégia promocional via lambda (frete grátis acima de X). Valide CEP/região e dispare exceções para CEP inválido. Mostre a troca de estratégia em tempo de execução.