Encapsulamento

1. **Criação de uma Classe Simples**
   - **Objetivo**: Crie uma classe `Produto` que represente um produto de uma loja. A classe deve ter atributos privados `nome`, `preco` e `quantidade` em estoque. Implemente métodos públicos para acessar e modificar esses atributos. Garanta que o preço e a quantidade não possam ser negativos.

2. **Melhorando a Classe com Validação**
   - **Objetivo**: Extenda a classe `Produto` para incluir um método `aplicarDesconto(double porcentagem)` que reduz o preço do produto. Valide para garantir que o desconto não possa ser maior que 50%. Implemente a lógica que lança uma exceção se o desconto for inválido.

Herança

3. **Criação de uma Hierarquia de Classes**
   - **Objetivo**: Crie uma classe `Funcionario` com atributos `nome` e `salario`. Em seguida, crie duas subclasses: `Gerente` e `Desenvolvedor`. Adicione um método `calcularBonus` que retorna um valor diferente para cada tipo de funcionário (por exemplo, 20% do salário para `Gerente` e 10% para `Desenvolvedor`). Utilize `protected` para permitir que as subclasses acessem os atributos da classe base de forma segura.

4. **Sobrescrita de Métodos**
   - **Objetivo**: Na classe `Funcionario`, crie um método `trabalhar()`, que imprime uma mensagem genérica sobre o trabalho realizado. Nas subclasses `Gerente` e `Desenvolvedor`, sobrescreva esse método para especificar o tipo de trabalho realizado por cada um. Utilize a anotação `@Override` e explore como ela ajuda a garantir que a sobrescrita foi feita corretamente.

Polimorfismo

5. **Polimorfismo com Interfaces**
   - **Objetivo**: Crie uma interface `IMeioTransporte` com métodos `acelerar()` e `frear()`. Implemente essa interface em classes `Carro`, `Bicicleta` e `Trem`. No método principal, crie um array de `IMeioTransporte` e percorra-o chamando `acelerar()` e `frear()` para cada objeto. Utilize polimorfismo para que cada tipo de transporte implemente `acelerar()` e `frear()` de maneira diferente.

6. **Polimorfismo com Classes Abstratas**
   - **Objetivo**: Crie uma classe abstrata `Animal` com um método abstrato `emitirSom()`. Crie subclasses `Cachorro`, `Gato` e `Vaca`, cada uma implementando `emitirSom()` de maneira específica. Crie uma lista de `Animal` no método principal e adicione instâncias de cada subclasse. Itere sobre a lista e invoque o método `emitirSom()` para cada animal, demonstrando o polimorfismo.

Abstração

7. **Abstração em um Sistema de Pagamentos**
   - **Objetivo**: Crie uma classe abstrata `FormaPagamento` com métodos abstratos `processarPagamento(double valor)` e `validarPagamento()`. Crie classes concretas `CartaoCredito`, `Boleto` e `Pix` que herdam de `FormaPagamento` e implementam os métodos abstratos. Adicione lógica de validação específica para cada forma de pagamento e simule um sistema que utilize diferentes formas de pagamento.

8. **Sistema de Gestão de Funcionários**
    - **Objetivo**: Crie um sistema que gerencie diferentes tipos de funcionários (`Gerente`, `Desenvolvedor`, `Estagiario`). Cada tipo de funcionário deve ter uma maneira diferente de calcular o salário e o bônus. Utilize uma combinação de herança, polimorfismo e encapsulamento para estruturar as classes. Implemente um método `calcularFolhaPagamento` que itera sobre todos os funcionários e calcula o total de salários e bônus. Adicione novas funcionalidades, como a possibilidade de promover um funcionário, o que altera seu tipo e os cálculos de salário e bônus.