Lista de Exercícios 02: Classes, Objetos e Encapsulamento

📚 ESTRUTURA PEDAGÓGICA DA LISTA

Esta lista contém **40 questões** organizadas de forma **incremental**, do básico ao avançado:

🎯 Tipos de Questões:

- **Questões 1-15:** Implementação Básica (Classes simples, atributos, métodos)
- **Questões 16-20:** Implementação com Encapsulamento (Getters, setters, validações)
- **Questões 21-25:** Correção de Código (Identificar e corrigir erros)
- **Questões 26-30:** Verdadeiro ou Falso (Conceitos teóricos)
- **Questões 31-35:** Análise de Código (Interpretar saída)
- **Questões 36-40:** Desafios Integrados (Sistemas completos)

📊 Progressão de Dificuldade:

```
BÁSICO (1-10) → INTERMEDIÁRIO (11-25) → AVANÇADO (26-35) → DESAFIOS (36-40)
```

💡 Como Estudar:

1. Leia cada questão com atenção
2. Tente resolver sozinho primeiro
3. Compile e teste seu código
4. Compare com a solução (pasta `solucoes/`)
5. Entenda os **conceitos** por trás de cada exercício

🟢 NÍVEL BÁSICO: Primeiros Passos com Classes

Questão 1: Classe Simples - Pessoa

Objetivo: Criar sua primeira classe com atributos públicos.

O que fazer:

- Crie uma classe chamada `Pessoa`
- Adicione dois atributos **públicos**: `nome` (String) e `idade` (int)
- Não precisa criar métodos ainda

Exemplo de uso esperado:

```
Pessoa p = new Pessoa();
p.nome = "João";
```

```
p.idade = 25;
System.out.println(p.nome + " tem " + p.idade + " anos");
```

Conceitos: Classe, atributos, tipos de dados

Questão 2: Criando Múltiplos Objetos

Objetivo: Entender que uma classe pode gerar vários objetos diferentes.

O que fazer:

- Use a classe `Pessoa` da Questão 1
- No método `main`, crie **três** objetos diferentes
- Atribua valores diferentes para cada objeto
- Imprima os dados de todas as pessoas

Saída esperada:

```
Maria tem 30 anos
Carlos tem 40 anos
Ana tem 22 anos
```

Conceitos: Instanciação, operador `new`, múltiplos objetos

Questão 3: Primeira Classe com Método

Objetivo: Adicionar comportamento (método) à classe.

O que fazer:

- Crie uma classe `Cachorro` com atributos públicos: `nome` (String) e `raca` (String)
- Adicione um método `void latir()` que imprime: `"Au au! Meu nome é [nome]"`
- Teste no `main` criando um cachorro e chamando o método

Exemplo de uso:

```
Cachorro dog = new Cachorro();
dog.nome = "Rex";
dog.raca = "Labrador";
dog.latir(); // Saída: Au au! Meu nome é Rex
```

Conceitos: Métodos de instância, `void`, chamada de métodos

Questão 4: Classe Produto Básica

Objetivo: Trabalhar com diferentes tipos de dados (String, double, int).

O que fazer:

- Crie uma classe `Produto` com atributos públicos:
  - `nome` (String)
  - `preco` (double)
  - `quantidade` (int)
- Crie um método `void exibirDados()` que imprime todas as informações do produto

Exemplo de saída:

```
Produto: Notebook
Preço: R$ 2500.00
Quantidade: 10
```

Conceitos: Múltiplos tipos de dados, formatação de saída

Questão 5: Método com Retorno

Objetivo: Entender a diferença entre `void` e métodos que retornam valores.

O que fazer:

- Use a classe `Produto` da Questão 4
- Adicione um método `double calcularValorTotal()` que retorna `preco * quantidade`
- Teste no `main` imprimindo o valor total em estoque

Exemplo de uso:

```
Produto p = new Produto();
p.nome = "Mouse";
p.preco = 50.0;
p.quantidade = 20;
double total = p.calcularValorTotal();
System.out.println("Valor total: R$ " + total); // Saída: Valor total: R$
1000.0
```

Conceitos: Métodos com retorno, tipo de retorno, palavra-chave `return`

Questão 6: Primeiro Construtor

Objetivo: Aprender a inicializar objetos com construtores.

O que fazer:

- Crie uma classe `Livro` com atributos públicos: `titulo` (String) e `autor` (String)

- Crie um **construtor** que recebe título e autor como parâmetros
- Teste criando um livro já com os dados inicializados

Exemplo de uso:

```
Livro livro = new Livro("1984", "George Orwell");
System.out.println(livro.titulo + " - " + livro.autor);
```

Conceitos: Construtores, inicialização de objetos, parâmetros

---

Questão 7: Classe Carro

Objetivo: Criar classe com múltiplos atributos e métodos.

O que fazer:

- Crie uma classe `Carro` com atributos públicos:
  - `marca` (String)
  - `modelo` (String)
  - `ano` (int)
- Adicione métodos:
  - `void acelerar()` → imprime "O carro está acelerando!"
  - `void frear()` → imprime "O carro está freando!"
  - `void exibirInfo()` → imprime marca, modelo e ano

Conceitos: Múltiplos métodos, organização de comportamentos

Questão 8: Classe com Cálculo - Círculo

Objetivo: Usar métodos para realizar cálculos matemáticos.

O que fazer:

- Crie uma classe `Circulo` com atributo público `raio` (double)
- Crie métodos:
  - `double calcularArea()` → retorna π × raio²
  - `double calcularCircunferencia()` → retorna 2 × π × raio
- Use `Math.PI` para o valor de π

Exemplo de uso:

```
Circulo c = new Circulo();
c.raio = 5.0;
System.out.println("Área: " + c.calcularArea());
System.out.println("Circunferência: " + c.calcularCircunferencia());
```

Conceitos: Operações matemáticas, classe `Math`, métodos de cálculo

---

Questão 9: Classe Retângulo

Objetivo: Praticar criação de classes com cálculos.

O que fazer:

- Crie uma classe `Retangulo` com atributos públicos: `base` (double) e `altura` (double)
- Crie um construtor que inicializa base e altura
- Crie métodos:
  - `double calcularArea()` → retorna base × altura
  - `double calcularPerimetro()` → retorna 2 × (base + altura)

Conceitos: Construtores com parâmetros, cálculos geométricos

Questão 10: Classe Funcionário Básica

Objetivo: Modelar conceitos do mundo real.

O que fazer:

- Crie uma classe `Funcionario` com atributos públicos:
  - `nome` (String)
  - `cargo` (String)
  - `salario` (double)
- Crie um método `void darAumento(double percentual)` que aumenta o salário
- Crie um método `void exibirDados()` que mostra todos os dados

Exemplo de uso:

```
Funcionario f = new Funcionario();
f.nome = "Pedro";
f.cargo = "Analista";
f.salario = 3000.0;
f.darAumento(10); // Aumenta 10%
f.exibirDados(); // Deve mostrar salário = 3300.0
```

Conceitos: Modelagem de objetos reais, métodos que modificam estado

---

🟡 NÍVEL INTERMEDIÁRIO: Encapsulamento e Validações

Questão 11: Introdução ao Encapsulamento

Objetivo: Entender por que usar `private` e criar getters/setters.

O que fazer:

- Transforme a classe `Pessoa` (Questão 1) usando **atributos privados**
- Crie métodos getters e setters para `nome` e `idade`
- No setter de idade, valide que não pode ser negativa

Código base para modificar:

```
public class Pessoa {
    // TORNAR PRIVADO
    public String nome;
    public int idade;

    // ADICIONAR GETTERS E SETTERS AQUI
}
```

Conceitos: Modificador `private`, getters, setters, validações básicas

---

Questão 12: Encapsulamento em Produto

Objetivo: Aplicar encapsulamento com validações.

O que fazer:

- Crie uma classe `Produto` com atributos **privados**:
  - `nome` (String)
  - `preco` (double)
  - `quantidade` (int)
- Crie getters para todos os atributos
- Crie setters com validações:
  - `setPreco`: não aceita valores negativos
  - `setQuantidade`: não aceita valores negativos
  - `setNome`: não aceita nome vazio

Exemplo de validação no setter:

```
public void setPreco(double preco) {
    if (preco < 0) {
        System.out.println("Erro: Preço não pode ser negativo!");
    } else {
        this.preco = preco;
    }
}
```

Conceitos: Validações, palavra-chave `this`, proteção de dados

Questão 13: Construtor com Encapsulamento

Objetivo: Combinar construtores com atributos privados.

O que fazer:

- Crie uma classe `ContaBancaria` com atributos **privados**:
  - `numeroConta` (String)
  - `titular` (String)
  - `saldo` (double)
- Crie um construtor que inicializa número da conta e titular (saldo inicia em 0)
- Crie **apenas getters** (não crie setSaldo)
- Crie métodos `depositar(double valor)` e `sacar(double valor)`
- O saque só pode ser feito se houver saldo suficiente

Exemplo de uso:

```
ContaBancaria conta = new ContaBancaria("12345", "Maria");
conta.depositar(1000);
conta.sacar(500);
System.out.println("Saldo: " + conta.getSaldo()); // Saída: Saldo: 500.0
```

Conceitos: Construtores, métodos de negócio, proteção de atributos

---

Questão 14: Classe Aluno com Notas

Objetivo: Criar classe com cálculos e encapsulamento.

O que fazer:

- Crie uma classe `Aluno` com atributos **privados**:
  - `matricula` (String)
  - `nome` (String)
  - `nota1` (double)
  - `nota2` (double)
- Crie construtor, getters e setters
- Crie método `double calcularMedia()` que retorna (nota1 + nota2) / 2
- Crie método `String verificarSituacao()` que retorna:
  - "Aprovado" se média >= 7
  - "Recuperação" se média >= 5 e < 7
  - "Reprovado" se média < 5

Conceitos: Métodos de cálculo, estruturas condicionais, regras de negócio

---

Questão 15: Classe Temperatura

Objetivo: Trabalhar com conversões e formatação.

O que fazer:

- Crie uma classe `Temperatura` com atributo privado `celsius` (double)
- Crie getters e setters
- Crie métodos:
  - `double paraFahrenheit()` → retorna (celsius × 9/5) + 32
  - `double paraKelvin()` → retorna celsius + 273.15
  - `void exibirConversoes()` → mostra temperatura em todas as escalas

Saída esperada:

```
Celsius: 25.0
Fahrenheit: 77.0
Kelvin: 298.15
```

Conceitos: Conversões, formatação de números, múltiplos cálculos

---

Questão 16: Sobrecarga de Construtores

Objetivo: Aprender a criar múltiplos construtores.

O que fazer:

- Crie uma classe `Produto` com atributos privados: `nome`, `preco`, `quantidade`
- Crie **três construtores**:
  1. Sem parâmetros (valores padrão)
  2. Com nome e preço (quantidade = 0)
  3. Com nome, preço e quantidade
- Use `this()` para chamar um construtor de outro

Exemplo de sobrecarga:

```java
public class Produto {
    private String nome;
    private double preco;
    private int quantidade;

    // Construtor 1: sem parâmetros
    public Produto() {
        this("Sem nome", 0.0, 0);
    }

    // Construtor 2: nome e preço
    public Produto(String nome, double preco) {
        this(nome, preco, 0);
    }

    // Construtor 3: completo
    public Produto(String nome, double preco, int quantidade) {
        this.nome = nome;
```

```
        this.preco = preco;
        this.quantidade = quantidade;
    }
}
```

Conceitos: Sobrecarga de construtores, palavra-chave `this()`, construtores encadeados

---

Questão 17: Classe com Validação de CPF

Objetivo: Criar validações mais complexas.

O que fazer:

- Crie uma classe `Cliente` com atributos privados: `nome`, `cpf`, `email`
- No `setCpf`, valide que o CPF tem exatamente 11 dígitos (apenas números)
- Remova pontos e traços antes de validar
- Se CPF inválido, não altere o atributo e mostre mensagem de erro

Dica de validação:

```
public void setCpf(String cpf) {
    // Remove pontos, traços e espaços
    String cpfLimpo = cpf.replaceAll("[^0-9]", "");

    if (cpfLimpo.length() == 11) {
        this.cpf = cpfLimpo;
    } else {
        System.out.println("CPF inválido! Deve conter 11 dígitos.");
    }
}
```

Conceitos: Expressões regulares, validação de dados, manipulação de strings

---

Questão 18: Classe Estoque

Objetivo: Criar classe com operações de adição e remoção.

O que fazer:

- Crie uma classe `Estoque` com atributos privados:
  - `nomeProduto` (String)
  - `quantidade` (int)
  - `estoqueMinimo` (int)
- Crie métodos:
  - `adicionar(int qtd)` → aumenta quantidade
  - `remover(int qtd)` → diminui quantidade (se houver estoque)
  - `boolean estaAbaixoDoMinimo()` → verifica se quantidade < estoqueMinimo

- `void verificarEstoque()` → mostra alerta se abaixo do mínimo

Conceitos: Métodos booleanos, validações de negócio, controle de estoque

---

Questão 19: Classe Data

Objetivo: Criar classe com validação de datas.

O que fazer:

- Crie uma classe `Data` com atributos privados: `dia`, `mes`, `ano` (int)
- No construtor e setters, valide:
  - Dia entre 1 e 31
  - Mês entre 1 e 12
  - Ano entre 1900 e 2100
- Crie método `String formatarData()` que retorna "dd/mm/aaaa"

Exemplo:

```
Data d = new Data(15, 8, 2024);
System.out.println(d.formatarData()); // Saída: 15/08/2024
```

Conceitos: Validações múltiplas, formatação, estruturas condicionais

---

Questão 20: Classe Funcionário com Benefícios

Objetivo: Criar classe com cálculos complexos.

O que fazer:

- Crie uma classe `Funcionario` com atributos privados:
  - `nome` (String)
  - `salarioBase` (double)
  - `temValeTransporte` (boolean)
  - `temValeAlimentacao` (boolean)
- Crie métodos:
  - `double calcularSalarioLiquido()` → salário base - descontos
  - Desconto de vale transporte: 6% do salário (se tiver)
  - Adicional de vale alimentação: + R$ 300 (se tiver)

Conceitos: Atributos booleanos, cálculos com condições, benefícios

---

🟡 NÍVEL INTERMEDIÁRIO AVANÇADO: Prática com Classes Simples

Questão 21: Classe Animal

Objetivo: Criar classe com atributos e métodos básicos.

O que fazer:

- Crie uma classe `Animal` com atributos privados:
  - `nome` (String)
  - `idade` (int)
  - `tipo` (String) - Ex: "Cachorro", "Gato"
- Crie um construtor que recebe nome e tipo (idade inicia em 0)
- Crie getters e setters para todos os atributos
- Crie um método `void fazerAniversario()` que aumenta a idade em 1
- Crie um método `void exibirInfo()` que mostra todos os dados

Teste no main:

```
Animal animal = new Animal("Rex", "Cachorro");
animal.setIdade(3);
animal.exibirInfo();
animal.fazerAniversario();
animal.exibirInfo();
```

Conceitos: Encapsulamento básico, métodos que modificam estado

---

Questão 22: Classe Calculadora Simples

Objetivo: Criar métodos com diferentes tipos de retorno.

O que fazer:

- Crie uma classe `Calculadora` (sem atributos)
- Crie métodos públicos:
  - `int somar(int a, int b)` → retorna a + b
  - `int subtrair(int a, int b)` → retorna a - b
  - `int multiplicar(int a, int b)` → retorna a × b
  - `double dividir(double a, double b)` → retorna a ÷ b (valide se b não é zero)

Teste no main:

```
Calculadora calc = new Calculadora();
System.out.println("Soma: " + calc.somar(10, 5));
System.out.println("Divisão: " + calc.dividir(10, 2));
```

Conceitos: Métodos estáticos vs instância, validações simples, tipos de retorno

---

Questão 23: Classe Nota Escolar

Objetivo: Trabalhar com validações e cálculos.

O que fazer:

- Crie uma classe `NotaEscolar` com atributos privados:
  - `disciplina` (String)
  - `nota` (double)
- Crie um construtor que recebe disciplina e nota
- No setter de nota, valide que está entre 0 e 10
- Crie método `String getConceito()` que retorna:
  - "Excelente" se nota >= 9
  - "Bom" se nota >= 7
  - "Regular" se nota >= 5
  - "Insuficiente" se nota < 5

Conceitos: Validações, estruturas condicionais, métodos de classificação

---

Questão 24: Classe ContadorSimples

Objetivo: Entender estado de objetos independentes.

O que fazer:

- Crie uma classe `ContadorSimples` com atributo privado:
  - `valor` (int) que inicia em 0
- Crie métodos:
  - `void incrementar()` → aumenta valor em 1
  - `void decrementar()` → diminui valor em 1
  - `void zerar()` → coloca valor em 0
  - `int getValor()` → retorna o valor atual

Teste criando 2 contadores:

```
ContadorSimples c1 = new ContadorSimples();
ContadorSimples c2 = new ContadorSimples();
c1.incrementar();
c1.incrementar();
c2.incrementar();
System.out.println("C1: " + c1.getValor()); // 2
System.out.println("C2: " + c2.getValor()); // 1
```

Conceitos: Independência entre objetos, estado do objeto

Questão 25: Classe Lampada

Objetivo: Trabalhar com atributos booleanos e estado.

O que fazer:

- Crie uma classe `Lampada` com atributos privados:

    - `ligada` (boolean) que inicia como false
    - `potencia` (int) em watts
- Crie um construtor que recebe a potência
- Crie métodos:
    - `void ligar()` → muda ligada para true
    - `void desligar()` → muda ligada para false
    - `String getEstado()` → retorna "Ligada" ou "Desligada"
    - `void mostrarInfo()` → mostra potência e estado

Conceitos: Atributos booleanos, métodos que alteram estado

🟣 QUESTÕES VERDADEIRO OU FALSO

Questão 26: Verdadeiro ou Falso - Conceitos Básicos

Objetivo: Testar compreensão teórica de POO.

Marque **(V)** para Verdadeiro ou **(F)** para Falso e **justifique sua resposta**:

( ) a) Atributos `private` podem ser acessados diretamente fora da classe.

( ) b) Métodos `void` não retornam nenhum valor.

( ) c) O construtor deve ter o mesmo nome da classe.

( ) d) Getters geralmente têm tipo de retorno `void`.

( ) e) É possível criar múltiplos objetos a partir de uma única classe.

( ) f) O construtor é chamado automaticamente ao usar o operador `new`.

Questão 27: V ou F - Encapsulamento

Marque **(V)** para Verdadeiro ou **(F)** para Falso e **justifique**:

( ) a) Encapsulamento significa esconder os detalhes internos de uma classe.

( ) b) Setters são usados para modificar atributos privados com segurança.

( ) c) É uma boa prática deixar todos os atributos como `public`.

( ) d) A palavra-chave `this` se refere ao objeto atual.

( ) e) Validações em setters ajudam a manter a integridade dos dados.

( ) f) Um atributo `private` pode ser acessado usando getters e setters.

Questão 28: V ou F - Construtores e Métodos

Marque **(V)** para Verdadeiro ou **(F)** para Falso e **justifique**:

( ) a) Um construtor pode ter tipo de retorno `void`.

( ) b) Se não criarmos nenhum construtor, Java cria um automaticamente.

( ) c) Construtores podem ser sobrecarregados (vários construtores com parâmetros diferentes).

( ) d) Um método pode ter o mesmo nome de um atributo da classe.

( ) e) A ordem dos parâmetros em um método é importante.

( ) f) Métodos podem modificar os atributos do objeto.

Questão 29: Complete o Código - Classe Livro Simples

Complete o código da classe `Livro` abaixo:

```
public class Livro {
    // 1. Declare atributos privados: titulo, autor, numPaginas
    _____________________________________
    _____________________________________
    _____________________________________

    // 2. Crie um construtor que recebe titulo e autor
    _____________________________________
    _____________________________________
    _____________________________________

    // 3. Crie um getter para titulo
    _____________________________________
    _____________________________________

    // 4. Crie um método void exibir() que mostra titulo e autor
    _____________________________________
    _____________________________________
}
```

Teste: Crie um objeto `Livro` e teste os métodos.

Questão 30: O que está errado? - Debugging

Analise o código abaixo e **identifique e corrija** os 3 erros:

```
public class Produto {
    private String nome;
    private double preco;

    // ERRO 1: Construtor está errado
    public void Produto(String nome, double preco) {
        nome = nome;        // ERRO 2: Falta this
```

```
        this.preco = preco;
    }

    // ERRO 3: Getter está errado
    public void getNome() {
        return nome;
    }
}
```

O que fazer:

- Liste os 3 erros
- Explique cada um
- Escreva o código corrigido

🟠 PRÁTICA COM MÚLTIPLOS CONCEITOS

Questão 31: Classe Hora

Objetivo: Criar classe com validações e formatação.

O que fazer:

- Crie uma classe `Hora` com atributos privados: `horas`, `minutos`, `segundos` (int)
- Crie um construtor que recebe os 3 valores
- Valide nos setters:
  - horas: 0 a 23
  - minutos: 0 a 59
  - segundos: 0 a 59
- Crie método `String formatar()` que retorna "HH:MM:SS" (use String.format("%02d", valor))

Teste:

```
Hora h = new Hora(14, 30, 45);
System.out.println(h.formatar()); // 14:30:45
```

Conceitos: Validações múltiplas, formatação de strings

Questão 32: Classe Termômetro

Objetivo: Trabalhar com conversões simples.

O que fazer:

- Crie uma classe `Termometro` com atributo privado: `temperaturaC` (double)
- Crie métodos:
  - `double getCelsius()` → retorna temperatura em Celsius

- `double getFahrenheit()` → retorna (C × 1.8) + 32
- `void setCelsius(double temp)` → define temperatura em Celsius
- `void setFahrenheit(double temp)` → converte F para C e armazena

Dica conversão F→C: C = (F - 32) / 1.8

Conceitos: Conversões, armazenamento em uma unidade padrão

Questão 33: Classe Estudante

Objetivo: Criar classe com cálculo de média simples.

O que fazer:

- Crie uma classe `Estudante` com atributos privados:
  - `nome` (String)
  - `nota1`, `nota2` (double)
- Crie construtor e getters/setters
- Crie método `double calcularMedia()` que retorna (nota1 + nota2) / 2
- Crie método `boolean foiAprovado()` que retorna true se média >= 6

Teste:

```
Estudante e = new Estudante("Ana", 7.5, 8.0);
System.out.println("Média: " + e.calcularMedia());
System.out.println("Aprovado: " + e.foiAprovado());
```

Conceitos: Métodos de cálculo, métodos booleanos

Questão 34: Classe Item de Compra

Objetivo: Trabalhar com cálculos de subtotal.

O que fazer:

- Crie uma classe `ItemCompra` com atributos privados:
  - `produto` (String)
  - `quantidade` (int)
  - `precoUnitario` (double)
- Crie construtor e getters/setters
- Crie método `double calcularSubtotal()` que retorna quantidade × preço
- Crie método `void exibirResumo()` que mostra produto, quantidade, preço e subtotal

Conceitos: Cálculos básicos, formatação de saída

Questão 35: Classe Ponto2D

Objetivo: Trabalhar com coordenadas e cálculo de distância.

O que fazer:

- Crie uma classe `Ponto2D` com atributos privados: `x`, `y` (double)
- Crie construtor e getters/setters
- Crie método `double calcularDistanciaOrigem()` que calcula a distância do ponto até a origem (0,0)
  - Fórmula: √(x² + y²)
  - Use `Math.sqrt()` e `Math.pow()`

Teste:

```
Ponto2D p = new Ponto2D(3, 4);
System.out.println("Distância: " + p.calcularDistanciaOrigem()); // 5.0
```

Conceitos: Matemática, uso da classe Math

🏆 PROJETOS FINAIS (Integradores e Opcionais)

Questão 36: Classe Jogo (Simples)

Objetivo: Criar classe que gerencia pontuação de um jogo.

O que fazer:

- Crie uma classe `Jogo` com atributos privados:
  - `nomeJogador` (String)
  - `pontos` (int)
  - `nivel` (int)
- Crie construtor que recebe apenas o nome (pontos e nível iniciam em 0 e 1)
- Crie métodos:
  - `void ganharPontos(int valor)` → adiciona pontos
  - `void subirNivel()` → aumenta nível em 1
  - `void exibirStatus()` → mostra nome, pontos e nível

Conceitos: Encapsulamento, métodos que modificam estado

Questão 37: Classe Vendedor (Comissão)

Objetivo: Calcular salário com comissão.

O que fazer:

- Crie uma classe `Vendedor` com atributos privados:
  - `nome` (String)
  - `salarioBase` (double)
  - `totalVendas` (double)

- Crie construtor e getters/setters
- Crie métodos:
  - `double calcularComissao()` → 10% do total de vendas
  - `double calcularSalarioTotal()` → salário base + comissão
  - `void exibirResumo()` → mostra todas as informações

Teste:

```
Vendedor v = new Vendedor("Carlos", 2000);
v.setTotalVendas(5000);
System.out.println("Salário Total: " + v.calcularSalarioTotal());
```

Conceitos: Cálculos percentuais, métodos de negócio

Questão 38: Classe Tamagotchi (Simplificado)

Objetivo: Simular bichinho virtual com estados simples.

O que fazer:

- Crie uma classe `Tamagotchi` com atributos privados:
  - `nome` (String)
  - `fome` (int) de 0 a 100
  - `felicidade` (int) de 0 a 100
- Crie construtor que recebe nome (fome = 50, felicidade = 50)
- Crie métodos:
  - `void alimentar()` → diminui fome em 20 (mínimo 0)
  - `void brincar()` → aumenta felicidade em 20 (máximo 100)
  - `String getEstado()` → retorna "Feliz", "Normal" ou "Triste" baseado na felicidade
  - `void exibirStatus()` → mostra nome, fome, felicidade e estado

Conceitos: Limites de valores, estados baseados em atributos

Questão 39: Classe Cofre (Poupança)

Objetivo: Simular um cofrinho com depósitos.

O que fazer:

- Crie uma classe `Cofre` com atributos privados:
  - `dono` (String)
  - `saldo` (double)
  - `meta` (double)
- Crie construtor que recebe dono e meta (saldo inicia em 0)
- Crie métodos:
  - `void depositar(double valor)` → adiciona ao saldo
  - `boolean atingiuMeta()` → retorna true se saldo >= meta

- `double faltaParaMeta()` → retorna quanto falta (meta - saldo)
- `void exibirProgresso()` → mostra saldo atual, meta e percentual atingido

Conceitos: Métodos booleanos, cálculos de progresso, formatação

---

Questão 40: Classe Turma (Desafio Final)

Objetivo: Integrar conceitos criando uma classe mais completa.

O que fazer:

- Crie uma classe `Turma` com atributos privados:
  - `disciplina` (String)
  - `professor` (String)
  - `numeroAlunos` (int)
  - `sala` (String)
- Crie construtor completo
- Crie getters e setters com validações:
  - numeroAlunos não pode ser negativo
  - disciplina e professor não podem ser vazios
- Crie métodos:
  - `void adicionarAluno()` → aumenta número de alunos em 1
  - `void removerAluno()` → diminui número de alunos em 1 (mínimo 0)
  - `boolean estaCheia()` → retorna true se >= 40 alunos
  - `void exibirInfo()` → mostra todas as informações

Teste:

```
Turma t = new Turma("POO", "Prof. Silva", 0, "Lab 01");
t.adicionarAluno();
t.adicionarAluno();
t.exibirInfo();
System.out.println("Turma cheia? " + t.estaCheia());
```

Conceitos integrados: Encapsulamento completo, validações, métodos de negócio, métodos booleanos

---

📌 RESUMO DOS CONCEITOS POR QUESTÃO

🟢 Nível Básico (Questões 1-10):

- Classes e objetos
- Atributos públicos
- Métodos simples (void e com retorno)
- Construtores básicos
- Tipos de dados (String, int, double)
- Cálculos matemáticos simples
- Formatação de saída

🟡 Nível Intermediário (Questões 11-20):

- Encapsulamento (private)
- Getters e setters
- Validações básicas
- Construtores com parâmetros
- Sobrecarga de construtores
- Palavra-chave `this`
- Métodos de negócio
- Validações complexas (CPF, datas)
- Atributos booleanos
- Cálculos com condições

🟡 Intermediário Avançado (Questões 21-25):

- Classes com múltiplos métodos
- Validações e classificações
- Independência entre objetos
- Estados de objetos
- Métodos que modificam estado

🟣 Conceitos Teóricos (Questões 26-30):

- Verdadeiro ou Falso sobre POO
- Completar código
- Identificar e corrigir erros
- Conceitos de encapsulamento
- Construtores e métodos

🟠 Prática Integrada (Questões 31-35):

- Validações múltiplas
- Conversões (temperatura, coordenadas)
- Formatação de strings
- Cálculos matemáticos (distância, média)
- Métodos booleanos

🏆 Projetos Finais (Questões 36-40):

- Sistemas simples mas completos
- Integração de conceitos
- Regras de negócio práticas
- Estados e comportamentos
- Cálculos percentuais
- Validações e limites

💡 DICAS PARA RESOLVER OS EXERCÍCIOS

1. **Leia com atenção** o enunciado completo
2. **Identifique** quais conceitos estão sendo testados
3. **Planeje** antes de codificar (quais atributos? quais métodos?)
4. **Compile frequentemente** para detectar erros cedo
5. **Teste** com diferentes valores
6. **Compare** com a solução (depois de tentar)
7. **Entenda** os conceitos, não apenas decore código

🎯 CRITÉRIOS DE CORREÇÃO

Cada questão será avaliada considerando:

- ✅ **Funcionamento correto** (30 pontos)
- ✅ **Encapsulamento adequado** (20 pontos)
- ✅ **Validações implementadas** (20 pontos)
- ✅ **Código organizado e legível** (15 pontos)
- ✅ **Nomenclatura adequada** (10 pontos)
- ✅ **Comentários explicativos** (5 pontos)

Total: 100 pontos por questão

Boa sorte! Qualquer dúvida, consulte o material de apoio ou o professor.