Lista de Exercícios
Programação Orientada a Objetos

Prof. Pedro Gabriel Calíope Dantas Pinheiro

8 de outubro de 2025

1. Em um sistema de controle de acesso de um evento, é necessário contar o número de pessoas que entram em uma sala. Implemente uma classe `Contador` em Java, responsável por encapsular um valor que representa o total de ocorrências. A classe deve oferecer métodos para:

   a) Zerar o contador;

   b) Incrementar o valor;

   c) Retornar o valor atual do contador.

2. Em uma aplicação gráfica 2D, é necessário representar pontos no plano cartesiano para manipular objetos geométricos. Implemente uma classe `Ponto2D` que represente um ponto no plano cartesiano. A classe deve oferecer:

   a) Construtores:

      i) Padrão (sem parâmetros), criando o ponto na origem (0,0);

      ii) Com dois valores `double` para coordenadas x e y;

      iii) A partir de outro ponto existente.

   b) Métodos de acesso (`getter/setter`);

   c) Método `equals` para comparação semântica;

   d) Método que calcule a distância entre dois pontos.

3. Um engenheiro civil deseja criar um programa para analisar a interseção de retas em um plano. Crie uma classe que represente uma reta ($y = ax + b$), oferecendo:

   a) Construtores:

      i) Com coeficiente angular e coeficiente linear;

      ii) A partir de dois pontos distintos.

b) Métodos de acesso para os coeficientes;

c) Método para verificar se um ponto pertence à reta;

d) Método que, dada outra reta, retorne o ponto de interseção ou `null` se forem paralelas.

4. Um aplicativo de design gráfico precisa manipular formas geométricas básicas. Implemente uma classe `Circulo` no plano cartesiano, com:

   a) Construtor que receba o raio e o centro do círculo (como um ponto);

   b) Construtor que receba apenas o raio e posicione o círculo na origem;

   c) Métodos de acesso ao raio;

   d) Métodos `inflar()` e `desinflar()` para alterar o tamanho;

   e) Método que retorne a área do círculo.

5. Em um sistema de geolocalização, cada país deve ser representado com informações básicas. Crie uma classe `Pais` com atributos: código ISO, nome, população e dimensão (em km^2), além de uma lista de países vizinhos.

   a) Construtor que inicialize código ISO, nome e dimensão;

   b) Métodos de acesso (`getters/setters`);

   c) Método de igualdade semântica (mesmo código ISO);

   d) Método que retorne a densidade populacional (habitantes/km^2).

6. Em um sistema genealógico digital, é preciso representar pessoas e suas relações familiares. Implemente uma classe `Pessoa` que modele indivíduos de uma árvore genealógica, contendo:

   a) Construtores:

      i) Inicialize nome, pai e mãe;

      ii) Inicialize apenas o nome (pai e mãe nulos).

   b) Método de igualdade semântica (mesmo nome e mesma mãe).

7. Um sistema de cadastro precisa manipular conjuntos de elementos textuais (como categorias ou palavras-chave) sem repetições. Crie uma classe `Conjunto` de elementos `String` com:

   a) Método para adicionar elemento (sem repetição);

   b) Método para verificar se um elemento pertence ao conjunto;

   c) Método `uniao` que retorne novo conjunto com a união;

d) Método `inter` que retorne novo conjunto com a interseção;

e) Método `menos` que retorne novo conjunto com a diferença.

8. Um módulo de processamento numérico precisa manipular operações matriciais. Desenvolva uma classe `Matriz` que ofereça suporte às operações básicas de álgebra linear:

   a) Construtor para inicializar dimensões (linhas e colunas);

   b) Métodos de acesso aos elementos;

   c) Método para somar duas matrizes;

   d) Operações complementares:

      i) Comparação semântica;

      ii) Transposta;

      iii) Oposta;

      iv) Matriz nula;

      v) Verificação se é identidade;

      vi) Verificação se é diagonal;

      vii) Verificação se é singular;

      viii) Verificação se é simétrica;

      ix) Verificação se é anti-simétrica;

      x) Subtração;

      xi) Multiplicação;

      xii) Cópia da matriz.

9. Um sistema de locadora de veículos precisa representar diferentes tipos de veículos. Crie uma classe `Veiculo` com os atributos básicos `marca` e `modelo`. Em seguida:

   a) Encapsule os atributos;

   b) Crie métodos de acesso (`getters/setters`);

   c) Crie uma classe `Carro` que herde de `Veiculo` e adicione o número de portas;

   d) Crie uma classe `Moto` que herde de `Veiculo` e adicione as cilindradas;

   e) Sobrescreva `toString()` em cada subclasse para exibir informações completas.

10. Em um sistema de RH, é necessário calcular bônus de funcionários de diferentes cargos. Crie uma classe abstrata `Funcionario` com atributos privados `nome` e `salario`.

    a) Métodos de acesso (`getters/setters`);

    b) Método abstrato `calcularBonus()`;

c) Subclasses:

i) `Gerente`: bônus de 20%;

ii) `Desenvolvedor`: bônus de 10%.

d) Crie uma lista de funcionários e exiba o bônus de cada um, demonstrando o polimorfismo.

11. Um zoológico digital deseja modelar animais e seus comportamentos. Crie uma hierarquia de classes para representar animais:

a) Classe abstrata `Animal` com atributos `nome` e `idade`;

b) Métodos de acesso (`getters/setters`);

c) Método abstrato `emitirSom()`;

d) Classes `Cachorro` e `Gato` que implementam `emitirSom()`;

e) Demonstre polimorfismo armazenando diferentes animais em uma lista.

12. Um banco digital precisa modelar diferentes tipos de contas. Crie uma classe `ContaBancaria` com atributos privados `numero` e `saldo`.

a) Métodos de acesso (`getters/setters`);

b) Métodos `depositar()` e `sacar()`;

c) Classe `ContaCorrente`, que cobra taxa fixa em cada saque;

d) Classe `ContaPoupanca`, que adiciona o método `renderJuros(double taxa)`.

13. Uma escola precisa registrar o desempenho dos alunos em avaliações. Implemente uma classe `Boletim` que utilize uma coleção `ArrayList<Double>` para armazenar as notas de um aluno.

a) Método para adicionar notas;

b) Método que calcule a média aritmética;

c) Método que retorne a maior e a menor nota;

d) Exibição formatada dos resultados.

14. Em um sistema de cadastro de produtos, é necessário armazenar o estoque em memória. Crie uma classe `Estoque` que utilize um `HashMap<String, Integer>` para associar o nome do produto à sua quantidade.

a) Métodos para adicionar, remover e atualizar produtos;

b) Método que exiba todos os produtos e quantidades;

c) Método que retorne o produto com maior quantidade em estoque.

15. Um aplicativo de rede social precisa armazenar nomes de usuários sem permitir repetições. Crie uma classe `GerenciadorUsuarios` que utilize um `HashSet<String>` para guardar os nomes únicos.

    a) Método para adicionar e remover usuários;

    b) Método que exiba todos os usuários cadastrados;

    c) Teste o comportamento de não duplicação de elementos.

16. Um sistema de biblioteca precisa manter um catálogo de livros e permitir buscas eficientes. Crie uma classe `Biblioteca` que utilize um `HashMap<String, String>` para associar ISBN e título.

    a) Métodos para adicionar, remover e buscar livros;

    b) Método que exiba todos os livros cadastrados;

    c) Método que verifique se um título já existe no acervo.

17. Em um sistema de votação digital, cada eleitor pode votar apenas uma vez. Crie uma classe `UrnaEletronica` que utilize um `HashMap<String, String>` para registrar votos (CPF $\rightarrow$ candidato).

    a) Método para registrar votos sem duplicação;

    b) Método que exiba todos os votos;

    c) Método que calcule o total de votos por candidato.

18. Uma empresa de automação residencial está desenvolvendo um sistema para controle de dispositivos inteligentes, como lâmpadas, termostatos e sensores de presença. Cada dispositivo pode ser ligado e desligado, e alguns também realizam medições específicas como temperatura ou luminosidade. Além disso, há dispositivos compostos, como um "Painel Central", que coordena o comportamento de outros dispositivos.

    Implemente uma estrutura de classes em Java que atenda a esses requisitos, considerando reutilização de código, extensibilidade e flexibilidade na adição de novos tipos de dispositivos.

19. Um sistema de reservas de hospedagem deve gerenciar clientes e suas reservas de forma eficiente. Cada cliente possui um identificador único (como CPF) e pode realizar diversas reservas, cada uma com código, data e valor. O sistema deve permitir registrar reservas, consultar todas as reservas de um cliente e exibir o total de reservas ativas.

    Implemente uma solução orientada a objetos em Java que atenda a essas necessidades, considerando eficiência de busca, organização e integridade dos dados.