Lista de Linguagem de Programação Orientada a Objetos

Linguagem de Programação Orientada a Objetos

FACOM – UFMS

Prof. Mariana Caravanti de Souza

1. Implemente em Java um pequeno sistema para armazenar a quantidade de acessos em um site
   a. Implemente uma classe ContadorDeVisitas que possua:
   - Um atributo **privado** total (número de visitas).
   - Um método público reiniciar() que zera o contador.
   - Um método público registrarVisita() que aumenta em 1 o número de visitas.
   - Um método público getTotal() que retorna o valor atual.

   b. Adicione ao ContadorDeVisitas um método registrarVisita(int quantidade) que aumente o contador em **quantidade** visitas, além do já existente que aumenta em apenas 1.
   c. Crie uma classe ContadorDeVisitasPremium que **herda** de ContadorDeVisitas e adicione um método registrarVisitaVIP(). Esse método deve aumentar em **10** o número de visitas.
2. Crie uma classe Coordenada2D que represente um ponto no plano cartesiano. A classe deve conter:
   a. Dois atributos **privados** x e y (números reais).
   b. Métodos getters e setters para os atributos.
   c. Um método mover() **sem parâmetros** que posicione o ponto na origem (0,0).
   d. Um método mover(double novoX, double novoY) que posiciona o ponto x e o ponto y considerando os valores informados.
   e. Um método mover(Coordenada2D outro), que recebe um objeto do tipo Coordenada2D e faz o objeto atual receber as mesmas posições do objeto recebido por parâmetro.
   f. Um método equals para verificar igualdade semântica entre dois pontos;
   g. Um método toString para escrever a posição atual do objeto;
   h. Um método que calcule a distância entre este ponto e outro ponto recebido como parâmetro.
3. Defina uma **interface FormaGeometrica**. A interface deve conter um **método pertence**(Coordenada2D ponto).

   Depois, implemente a classe **Reta** (y = ax + b) que:

   a) Possui atributos a e b (coeficientes da equação);

   b) **Implementa FormaGeometrica**;

c) Possui métodos getters e setters;

d) Possui toString para exibir a equação da reta;

e) Possui um método que, dado outra Reta, retorna o ponto de interseção ou null se forem paralelas.

4. Defina uma **interface OperacaoComplexa** que declare os métodos:
   a. somar(Complexo outro)
   b. subtrair(Complexo outro)
   c. multiplicar(Complexo outro)
   d. dividir(Complexo outro)
   e. Em seguida, crie a **classe Complexo** que:
   - Possui dois atributos privados: parteReal, parteImaginaria;
   - Possui métodos públicos getters e setters;
   - **Implementa** todos os métodos da interface OperacaoComplexa
   - Implementa equals para verificar igualdade semântica;
   - Implementa toString para exibir o número no formato "a + bi";
   - Fornece um método que calcule o **módulo** do número.
5. Implemente a classe Circulo, que deve:

   a) Ter atributos raio e centro (um objeto Coordenada2D);

   b) Fornecer métodos getters e setters para o raio;

   c) Possuir métodos inflar e desinflar que aumentem/diminuam o raio de um valor informado;

   d) Fornecer uma versão sobrecarregada de inflar e desinflar que alterem o raio em 1 unidade;

   e) Métodos mover para reposicionar o centro:
   - sem parâmetros → origem
   - com dois valores (x, y)
   - com outro ponto (Coordenada2D)
   - Calcular e retornar a área do círculo.
6. Implemente a classe Pais, com os atributos:
   - código ISO (ex.: "BRA")
   - nome
   - população

- área em km²
- lista de países vizinhos (máximo de 40).
- A classe deve fornecer:

a. Métodos getters e setters para todos os atributos;
b. Método equals que compara países pelo código ISO;
c. Método que verifique se outro país é vizinho;
d. Método que calcule a densidade populacional;
e. Método que, recebendo outro país, retorne os vizinhos em comum entre eles.

7. **Classe Continente**

   Implemente em Java uma classe Continente, que representa um agrupamento de países.

   A classe deve:

   a) Possuir atributo nome, conjunto de países (lista ou vetor);

   b) métodos getters e setters

   c) Método para adicionar um novo país a lista;

   e) Método que calcule o país que possui maior **população total** do continente;

   f) Método que calcule o país com maior **densidade populacional** do continente;

   h) Método que retorne o país com menor população;

   i) Método que retorne o país de maior dimensão territorial;

   j) Método que retorne o país de menor dimensão territorial;

   k) Método que calcule a razão entre a maior e a menor área territorial.

8. Crie uma classe Pessoa que representa uma pessoa em uma árvore genealógica.

   A classe deve:

   a) Possuir atributos nome, pai e mae (sendo pai e mae objetos do tipo Pessoa);

   b) Métodos setNome, setPai, setMae, além de getters correspondentes;

   c) Método equals que considera duas pessoas iguais se possuem o **mesmo nome e a mesma mãe**;

   d) Método que verifique se duas pessoas são irmãs;

   e) Método que verifique se uma pessoa é antecessora da outra (pai, mãe ou qualquer antepassado).

9. Implemente uma classe Conjunto que armazena Strings sem repetição.

   A classe deve:

   a) Método adicionar(String elemento) que só insere se não existir;

b) Método contem(String elemento) que verifica presença;

c) Método uniao(Conjunto outro) que devolve um novo conjunto com todos os elementos de ambos, sem repetição;

d) Método inter(Conjunto outro) que devolve um novo conjunto com os elementos em comum;

e) Método menos(Conjunto outro) que devolve um novo conjunto com os elementos que estão neste conjunto mas não no outro.

10. Implemente em Java uma hierarquia de classes que represente animais.
    a. Crie a classe **Animal** que possui:
    - Atributo nome (String).
    - Método emitirSom() (void), que apenas imprime "Som genérico de animal".
    - Método getNome() e setNome(String nome).

    b. Crie as subclasses:
    - **Cachorro**, que sobrescreve emitirSom() para imprimir "Au au!".
    - **Gato**, que sobrescreve emitirSom() para imprimir "Miau!".
    - **Vaca**, que sobrescreve emitirSom() para imprimir "Muuu!".

    c. Na classe de teste (Main), crie um **vetor de Animal** que armazena diferentes objetos (Cachorro, Gato, Vaca).
    d. Percorra o vetor e chame emitirSom() para cada animal, mostrando o funcionamento do **polimorfismo**.
11. Implemente em Java uma hierarquia de classes que represente veículos.
    a. Crie a classe **Veiculo** que possui:
    - Atributo modelo (String).
    - Método mover() (void), que apenas imprime "Veículo em movimento genérico".
    - Métodos getModelo() e setModelo(String modelo).

    b. Crie as subclasses:
    - **Carro**, que sobrescreve mover() para imprimir "O carro está dirigindo".
    - **Moto**, que sobrescreve mover() para imprimir "A moto está acelerando".
    - **Bicicleta**, que sobrescreve mover() para imprimir "A bicicleta está pedalando".

    c. Na classe de teste (**Main**), crie um vetor de **Veiculo** que armazena diferentes objetos (Carro, Moto, Bicicleta).

d. Percorra o vetor e chame mover() para cada veículo, mostrando o funcionamento do **polimorfismo**.

12. Implemente em Java uma hierarquia de classes que represente formas de pagamento.
    a. Crie a **interface** Pagamento que possui:
    - Método void pagar(double valor) , que apenas imprime "Pagamento genérico de R$<valor>".

    b. **Crie as classes que implementam Pagamento:**
    - **CartaoCredito**, que sobrescreve pagar(double valor) para imprimir "Pagamento de R$<valor> realizado no cartão <numeroCartao>".
    - **Boleto**, que sobrescreve pagar(double valor) para imprimir "Pagamento de R$<valor> realizado via boleto <codigoBoleto>".
    - **Pix**, que sobrescreve pagar(double valor) para imprimir "Pagamento de R$<valor> realizado via Pix <chavePix>".

    c. Na classe de teste (**Main**), crie um **vetor de Pagamento** que armazena diferentes objetos (CartaoCredito, Boleto, Pix).
    d. Percorra o vetor e chame pagar(valor) para cada forma de pagamento, mostrando o funcionamento do **polimorfismo**.