UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO
SCC0504/SCC0204 – Programação Orientada a Objetos
Prof. José Fernando Rodrigues Júnior

Lista 06 – Revisão Geral

➔ **Em todos os exercício, desenhe o correspondente Diagrama UML.**

1. (Classes/Objetos/Composição) Implemente uma aplicação em Java para uma loja virtual que realiza controle de estoque. Utilize as seguintes classes:

Classe Produto:

- id (inteiro único para cada produto)
- nome (String)
- preco (double)
- quantidade (int)
- vender(int quantidadeVendida) → reduz a quantidade do estoque se houver estoque suficiente
- repor(int quantidadeReposta) → incrementa a quantidade no estoque
- toString() → retorna os dados do produto

Classe Estoque:

- um atributo que seja uma lista (um array primitivo) contendo vários objetos da classe Produto
- adicionarProduto(Produto produto)
- removerProdutoPorId(int id)
- buscarProdutoPorNome(String nome)
- toString()→ exibe todos os produtos no estoque com suas quantidades atuais

Seu método main deve ser o seguinte:

```
Estoque estoque = new Estoque();
estoque.adicionarProduto(new Produto(1, "Mouse Gamer", 120.0, 15));
estoque.adicionarProduto(new Produto(2, "Teclado Mecânico", 250.0, 10));
estoque.adicionarProduto(new Produto(3, "Monitor LED", 900.0, 5));
Produto produtoBuscado = estoque.buscarProdutoPorNome("Mouse Gamer");
produtoBuscado.vender(3);
produtoBuscado.repor(2);
System.out.println(estoque);
```

2. (Herança/Abstração/Polimorfismo) Crie em Java um sistema de organização de documentos. O sistema deverá conter uma classe abstrata chamada Documento, com atributos comuns a todos os documentos (por exemplo, título, autor e tamanho em KB). Em seguida, crie subclasses especializadas que herdem de Documento, tais como: Texto (com número de palavras), Planilha (com quantidade de células), Apresentacao (com número de slides). Cada uma dessas subclasses deve sobrescrever o método abrir() da classe base, simulando comportamentos específicos para cada tipo de documento. Por fim, crie uma lista contendo vários documentos de tipos distintos e utilize o polimorfismo para exibir informações detalhadas e testar o método abrir() em cada objeto da lista.

Seu método main deve ser o seguinte:

```
List<Documento> biblioteca = new ArrayList<>();
biblioteca.add(new Texto("Relatório Anual", "Carlos Silva", 250, 1500));
biblioteca.add(new Planilha("Finanças", "Maria Souza", 120, 500));
biblioteca.add(new Apresentacao("Pitch Vendas", "João Pereira", 340, 15));

for (Documento doc : biblioteca) {
    doc.abrir();
    doc.formatar();
    System.out.println(doc);
}
```

3. (Interface/Polimorfismo) Crie uma interface chamada MetodoPagamento contendo dois métodos: realizarPagamento(double valor) e getTaxaPagamento(). Implemente três classes que utilizem essa interface com comportamentos específicos: CartaoCredito (com atributos como número do cartão e limite disponível), Pix (com atributo chavePix) e BoletoBancario (com atributos código de barras e data de vencimento). Utilize uma lista do tipo MetodoPagamento para armazenar objetos das diferentes implementações e demonstre o polimorfismo chamando o método realizarPagamento() em cada um deles.

Seu método main deve ser o seguinte:

```
List<MetodoPagamento> pagamentos = new ArrayList<>();
pagamentos.add(new CartaoCredito("1234-5678-9876-5432", 1000));
pagamentos.add(new Pix("usuario@pix.com"));
pagamentos.add(new BoletoBancario("23793", "2025-04-10"));

double valorCompra = 300;
double total = 0;

for (MetodoPagamento metodo : pagamentos) {
  metodo.realizarPagamento(valorCompra);
  total += metodo.getTaxaPagamento();
}

System.out.printf("Custo total com taxas: R$ %.2f%n", totalTaxas);
```

4.(Modelagem/Composição/Interface/Polimorfismo) Crie uma classe chamada NumeroComplexo contendo atributos para representar a parte real e imaginária. Essa classe deve conter métodos para realizar operações com outros números complexos:

- `double modulo()`
- `String toString()`
- `NumeroComplexo adicionar(NumeroComplexo)`
- `NumeroComplexo multiplicar(NumeroComplexo)`

Adicionalmente, defina uma interface chamada OperacaoComplexa com o método `executar(NumeroComplexo a, NumeroComplexo b)`. Implemente essa interface nas classes concretas `SomaComplexa`, e `MultiplicacaoComplexa`, cada uma realizando a operação correspondente.

Em seguida, crie uma classe chamada CalculadoraComplexa que utilize objetos do tipo OperacaoComplexa para executar as operações matemáticas, demonstrando claramente o uso de polimorfismo com interfaces.

Seu método main deve ser o seguinte:

```
NumeroComplexo c1 = new NumeroComplexo(3, 4);
NumeroComplexo c2 = new NumeroComplexo(1, -2);
System.out.println(c1);
System.out.println(c1.modulo());
System.out.println(c2);
System.out.println(c2.modulo());

CalculadoraComplexa calculadora = new CalculadoraComplexa();

calculadora.setOperacao(new SomaComplexa());
NumeroComplexo resultadoSoma = calculadora.calcular(c1, c2);
System.out.println("Soma: " + resultadoSoma);

calculadora.setOperacao(new MultiplicacaoComplexa());
NumeroComplexo resultadoMultiplicacao = calculadora.calcular(c1, c2);
System.out.println("Multiplicação: " + resultadoMultiplicacao);
```

5.(Interface/Ordenação) O algoritmo a seguir é capaz de ordenar objetos de qualquer classe, desde a classe implemente uma interface denominada Comparavel.

```
Interface Comparavel{
  public int comparaCom(Object outroObjeto);
}
```

O método comparaCom deve retornar:

- um número positivo se this for maior que outroObjeto;
- zero se forem iguais;
- um número negativo se this for menor que outroObjeto.

Algoritmo de ordenação BubbleSort:

```
public static void bubbleSort(Object[] array) {
        int n = array.length;
        boolean trocou;
        do {
            trocou = false;
            for (int i = 0; i < n - 1; i++) {
                Comparavel atual = (Comparavel) array[i];
                Comparavel proximo = (Comparavel) array[i + 1];
                if (atual.comparaCom(proximo) > 0) {
                    Object temp = array[i];
                    array[i] = array[i + 1];
                    array[i + 1] = temp;
                    trocou = true;
                }
            }
            n--;
        } while (trocou);
}
```

Implemente a interface, o algoritmo, e uma classe TimeFutebol.
No main, crie um array com 5 times de futebol e faça a ordenação deste array com o seguinte critério: um time antecede a outro time se ele tiver um valor maior para seu número de pontos ganhos = n_vitorias*3 + n_empates.