Paradigmas de Linguagens de Programação
Lista 4
Prof. Sergio d Zorzo

---

1. Explique a diferença entre encapsulamento e ocultação da informação
2. Explique a diferença entre coesão e acoplamento
3. Considere as duas classes Java abaixo

```
public class Carro1 {
  public double velocidade() {
    // Calcula e retorna a velocidade do carro
  }
  public Posicao posicaoAtual() {
    // Calcula e retorna a posicao atual do carro
  }
  private Posicao posicao;
}

public class Carro2 {
  public double velocidade() {
    // Calcula e retorna a velocidade do carro
  }
  public double latitudeAtual() {
    // retorna a latitude atual
  }
  public double longitudeAtual() {
    // retorna a longitude atual
  }
  private double latitude;
  private double longitude;
}
```

a. Em qual delas o princípio da ocultação da informação é melhor aplicado? Por que?
b. Qual das duas tem menor acoplamento? Por que?

4. Considere a seguinte classe Java

```
public class Carro {
  Posicao posicao;
  private void lePosicaoDoGPS() {
    // Lê sinal do GPS, calcula a posição e armazena o resultado no atributo
    // posicao
  }
  public Posicao getPosicaoAtual() {
    // Retorna a posicao atual do carro
  }
  public double getVelocidade() {
    // Calcula e retorna a velocidade do carro
  }
}
```

a. Como você implementaria o método getPosicaoAtual()? Que atributos seriam necessários?
b. Como você implementaria o método getVelocidade()? Que atributos seriam necessários?

5. Dê dois motivos para utilização de setters e getters, considerando os benefícios obtidos pelo princípio da ocultação da informação.

6. Considere o seguinte método em Java, que realiza uma busca de histórico de um cliente, em um sistema e-commerce. Considere que um log consiste de uma entrada no banco de dados, contendo: <id do cliente, data/hora, ação realizada>

```
Log[] buscarLogs(Cliente c) {
   // executa consulta no banco, buscando os logs do cliente
   // retorna todos em um array
}
```

Explique qual o resultado, em termos de acoplamento e coesão, se mudarmos o método para:

```
Log[] buscarLogs(int idCliente) {
   // executa consulta no banco, buscando os logs do cliente
   // retorna todos em um array
}
```

7. Quanto maior o número de dependências, maior o acoplamento de uma classe. Você concorda ou discorda? Por que?

8. Considere o seguinte código Java

```
class Log {
  public int idUsuario;
  public Date data;
  public String acao;
}
class Cadastro {
  int usuarioLogado;
  public void inserirCliente() {
     // inserir cliente no banco de dados
     Log log = new Log();
     log.idUsuario = idUsuario;
     log.data = new Date(); // recupera a data/hora atuais
     log.acao = "inserirCliente";
     Logger logger = new Logger();
     logger.gravarLog(log);
  }
}
class Logger {
  public void gravarLog(Log l) {
    // gravar l no banco de dados
  }
}
```

Modifique as classes acima para melhor aplicar o princípio da ocultação da informação e melhorar acoplamento e coesão.

9. O que é abstração de processos? E abstração de dados?

10. Que recursos a orientação a objetos oferece para aplicar a ocultação da informação?

11. Considere o programa a seguir

```
public String concatenaStrings1(String a, String b) {
  a += b;
  return b;
}

public StringBuffer concatenaStrings2(StringBuffer a,StringBuffer b) {
  a.append(b);
  return b;
}

public void principal() {
  String saudacao1 = "Alo";
  StringBuffer saudacao2 = new StringBuffer("Alo");

  String x = concatenaStrings1(saudacao1, saudacao1);
  StringBuffer y = concatenaStrings2(saudacao2, saudacao2);
  System.out.println(x);
  System.out.println(y);
}
```

a. Explique como a criação de apelidos nos procedimentos concatenaStrings1 e concatenaStrings2 prejudica a legibilidade do programa.
b. Explique porque x e y possuem valores diferentes ao final da execução do procedimento principal, mesmo que os procedimentos concatenaStrings1 e concatenaStrings2 aparentemente façam a mesma coisa.
c. Cite uma desvantagem da imutabilidade das Strings em Java

12. Qual a saída do programa abaixo?

```
class Posicao {
  int x, y;
}
…
Posicao p = new Posicao();
System.out.println(p.x);
System.out.println(p.y);
…
```

13. Modifique o programa anterior para que uma posição nunca possa assumir valores negativos.

14. Implemente um método que troca os valores de dois argumentos inteiros, um pelo outro.

15. Explique o que é e como deve ser usada a chamada this(...)

16. Explique o que é e como deve ser usada a chamada super(...)

17. Explique porque o programa abaixo não compila. O que deve ser alterado em “Classe” para que ele compile corretamente?

```
class SuperClasse {
   public SuperClasse(String valorInicial) {
   }
}
class Classe extends SuperClasse {
}
```

18. Explique porque o programa abaixo não compila. O que deve ser alterado em “Classe” para que ele compile corretamente e funcione da mesma forma abaixo?

```
class SuperClasse {
  private String valor;
  public SuperClasse(String valorInicial) {
    this.valor = valorInicial;
  }
}
class Classe extends SuperClasse {
  public Classe(String valorInicial) {
    valorInicial += "!";
    super(valorInicial);
  }
}
```

19. O que o modificador static faz quando usado com um atributo? Para que é utilizado?

20. O que o modificador static faz quando usado com um método? Para que é utilizado?

21. O que o modificador static faz quando usado com um bloco de código? Para que é utilizado?

22. O código abaixo compila? Por que?

```
class Qualquer {
  static int x;
  void mudar() {
    x += 2;
  }
}
```

23. O código abaixo compila? Por que?

```
class Qualquer {
  int x;
  static void mudar() {
    x += 2;
  }
}
```

24. Crie código Java que representa a seguinte situação:

“Uma pessoa possui nome e sobrenome, ambos do tipo String. Um livro possui autor, que é uma pessoa, título, que é uma String e número de páginas, que é um inteiro. Um autor pode ter vários livros, mas um livro tem somente um autor.”

25. Crie código Java que representa a seguinte situação:

“Um horário possui as horas, minutos e segundos (inteiros). Uma tarefa tem um horário de início e um horário de fim, além de um título (String) e um usuário que a solicitou (String).”

26. Crie código Java que representa a seguinte situação:

“Uma categoria possui nome, e subcategorias, que por sua vez também são categorias com suas próprias subcategorias, e assim por diante.”

27. Crie código java que representa a seguinte situação:

“Um arquivo possui nome e localização (Strings) e tamanho (inteiro). Um arquivo também possui anotações. Uma anotação possui um título (String) e uma data (String). Não podem existir anotações que não estejam associadas a um arquivo. Existem dois tipos de anotação: visual e textual. Anotações visuais possuem, além de título e data, uma lista de coordenadas (array de pares de inteiros). Anotações textuais possuem, além de título e data, um texto (String).”

28. Aponte exatamente os locais onde o código abaixo não compila. Como resolver cada um deles?

```
1:   class Numero {
2:   }
3:   class Real extends Numero {
4:     double valorReal;
5:     Real(double valorReal) { this.valorReal = valorReal; }
6:   }
7:   class Inteiro extends Numero {
8:     int valorInteiro;
9:     Inteiro(int valorInteiro) { this.valorInteiro = valorInteiro; }
10:  }
11:  class Teste {
12:    Numero somarNumerosInteiros(Numero um, Numero dois) {
13:      int result = um.valorInteiro + dois.valorInteiro;
14:      return new Inteiro(result);
15:    }
16:    Numero somarNumerosReais(Numero um, Numero dois) {
17:      double result = um.valorReal + dois.valorReal;
18:      return new Real(result);
19:    }
20:    void principal() {
21:      Inteiro i1 = new Inteiro(1);
22:      Inteiro i2 = new Inteiro(2);
23:      Inteiro i3 = somarNumerosInteiros(i1, i2);
24:      Real r1 = new Real(1.0);
25:      Real r2 = new Real(2.0);
26:      Real r3 = somarNumerosReais(r1, r2);
27:    }
28:  }
```

29. Modifique o código da classe “Teste” do exercício anterior, criando um novo procedimento: somarNumeros, que funciona para qualquer tipo de número, Real ou Inteiro.

30. Repita o exercício anterior, utilizando sobrescrita de métodos para obter o mesmo efeito. Utilize a seguinte classe abstrata como ponto de partida.

```
abstract class Numero {
  abstract Numero somar(Numero outro);
}
```

31. Uma pessoa tem nome, sobrenome e idade. Utilize a interface Ordenavel e o método de ordenação apresentados em aula, e faça ordenação de Pessoas em ordem decrescente de sobrenome, depois nome e depois idade. Em outras palavras, se duas pessoas tem o mesmo sobrenome, ordene pelo nome. Se possuem o mesmo sobrenome e nome, ordene pela idade.

32. Implemente o seguinte diagrama de classes utilizando Java, garantindo a semântica dos relacionamentos. Insira construtores, métodos set e get, além de métodos para manutenção dos relacionamentos de lista, onde julgar necessário.