



8,90

## Avaliação A1

Nome: Ericksen G. M. Ilie Turma: T1: ☒ T2: ☐

### Orientações:

1. A prova pode ser feita a lápis, porém o professor se dará ao direito de não aceitar reclamações relativas à correção.
2. Coloque o seu nome nas folhas de respostas.
3. Manter celulares/dispositivos eletrônicos desligados!

### Boa sorte!

**Questão 1 – (1,0 pontos)** A respeito da linguagem de programação orientada a Objetos (Java) e o uso de Construtores, é correto afirmar que:

- 1.0
- a. O método conhecido como construtor é caracterizado como uma função que sempre retorna um valor diferente.
  - ☒ b. Um construtor é executado sempre que um objeto for criado.
  - c. Um construtor é uma variável de classe que armazena informações sobre um objeto criado.
  - d. Os construtores em Java não podem ser sobrecarregados.
  - e. O construtor default/padrão é aquele que recebe, no mínimo, um argumento como parâmetro.

**Questão 2 – (1,0 pontos)** A respeito de herança, qual das afirmativas abaixo está **incorreta**:

- 0.0
- a) Herança permite que uma classe herde os atributos e métodos de outra classe.
  - ☒ b) A reutilização de código não é uma característica da herança.
  - ☒ c) Através de uma classe derivada/subclasse é possível sobrescrever um método da classe pai/superclasse/classe base.
  - d) Na Herança, a classe que está herdando métodos/atributos de uma classe pai/superclasse pode ser chamada de classe derivada ou subclasse.



Questão 3 – (2,0 pontos) Assinale VERDADEIRO (V) ou FALSO (F) para as afirmativas abaixo:

2.0  
(V) `static` é um modificador de acesso que, quando precedido em um atributo, permite que o atributo seja compartilhado por todas as instância/objetos criados a partir dessa classe, já quando precedidos em métodos, mesmo que inexistindo instância/representação do objeto, permite que acessamos o método diretamente através do nome da classe.

(F) Para fazer uso exclusivamente de um método estático, se faz necessário instanciar/criar um objeto do tipo da classe onde o método foi definido.

(V) Quando se diz que uma classe "ContaCorrente" estende a classe "Conta", em programação orientada a objetos, estamos afirmando que, a classe "ContaCorrente" é uma subclasse da classe "Conta" e que a classe Conta seria uma Superclasse/Classe Pai.

(V) Métodos Getter e Setter, juntamente com modificadores de acesso como `private`, promovem o encapsulamento.

(V) A linha de código a seguir indica que estamos criando um objeto do tipo Conta e referenciando a variável `c1` ao endereço de memória onde encontra-se o objeto do tipo Conta. Código: `Conta c1 = new Conta()`.

(V) Por meio do Encapsulamento em programação orientada a objetos permite-se controlar acesso aos membros de uma classe, promovendo maior segurança e integridade dos dados.

(F) Caso uma variável seja definida com o modificador de acesso `private`, indica que outras classes não poderão acessá-la diretamente, com exceção das classes que herdam dessa classe.

(V) A JVM (Java Virtual Machine) é responsável por executar bytecode Java, proporcionando uma camada de abstração entre o código Java e o sistema operacional subjacente, permitindo que aplicações Java sejam executadas em diferentes plataformas sem modificação.

1.0  
Questão 4 – (1,0 pontos) Considerando o trecho de código abaixo, preencha as lacunas:

```
public class Funcionario{  
    private String nome;  
    private int idade;  
  
    public _____ getNome() {  
  
        _____ ;  
    }  
}
```

a) String

b) return this.nome



<pre>public _____ setNome(String nome) {     this.nome = nome; } public _____ getIdade() {     _____; } public _____ setIdade(int idade) {     this.idade = idade; } }</pre>	<p>c) <u>void</u> ✓</p> <p>d) <u>int</u> ✓</p> <p>e) <u>return this.idade</u> ✓</p> <p>f) <u>void</u> ✓</p>
--	---

1.5 **Questão 5 – (1,5 pontos)** Considerando o código abaixo, que as classes estão no mesmo pacote, e que cada classe é representada por um arquivo, indique o resultado da execução do programa.

```
public class Animal {  
    protected String nome;  
    public Animal(String nome) {  
        this.nome = nome;  
    }  
    public void emitirSom() {  
        System.out.println("O animal emite um som.");  
    }  
}
```

```
public class Cachorro extends Animal {  
    public Cachorro(String nome) {  
        super(nome);  
    }  
    @Override  
    public void emitirSom() {  
        System.out.println("O cachorro late.");  
    }  
    public void abanarRabo() {  
        System.out.println("O cachorro está abanando o rabo.");  
    }  
}
```



```
public class Teste {
    public static void main(String[] args) {
        Cachorro c1= new Cachorro ("Bobby");
        c1.emitirSom();
        c1.abanarRabo();
    }
}
```

a) A compilação falha porque o método `abanarRabo()` não é definido na classe `Animal`.

☒ b) A compilação é bem-sucedida e a saída é:

- ☐ O cachorro late.
- ☐ O cachorro está abanando o rabo.

c) A compilação falha porque o método `emitirSom()` não é definido na classe `Cachorro`.

d) A compilação falha porque o atributo `nome` não foi corretamente definido no construtor.

**Questão 6 – (2,5 pontos)** Defina um domínio qualquer, abstraia e implemente classes e subclasses que representam o domínio.

- a) **(0,5 pontos)** Utilize do conceito de Encapsulamento e comente a(s) linha(s) onde ocorre;
- b) **(0,5 pontos)** Utilize do conceito de Herança e comente a(s) linha(s) onde ocorre;
- c) **(0,5 pontos)** Utilize das boas práticas em escrita de nomenclaturas de classes e variáveis;
- d) **(0,4 pontos)** Deve haver no mínimo 3 (três) classes;
- e) **(0,4 pontos)** Deve haver no mínimo 1 (um) atributo por classe;
- f) **(0,2 pontos)** Deve haver no mínimo 2 (dois) métodos em cada classe;

**Questão 7 – (1,0 pontos)** Escreva um programa principal em java que represente ao menos um objeto do domínio da questão anterior, definindo seus atributos e apresentando em tela.

```
public class Teste{
    public static void main (String[] args){
```



Erickson G. Miller t1

- 6 -

```
public class Alimento {  
    protected String nome; // encapsulamento  
    private int calorias; // encapsulamento  
    private String codigo; // encapsulamento  
  
    public Alimento(String nome, String codigo) { // constructor  
        this.nome = nome;  
        this.codigo = codigo;  
    }  
  
    public void setName(String nome) { // setter  
        this.nome = nome;  
    }  
  
    public void setCalorias(int calorias) { // setter  
        this.calorias = calorias;  
    }  
  
    public void setCodigo(String codigo) { // setter  
        this.codigo = codigo;  
    }  
  
    public String getName() { // getter  
        return this.nome;  
    }  
  
    public int getCalorias() { // getter  
        return this.calorias;  
    }  
  
    public String getCodigo() { // getter  
        return this.codigo;  
    }  
}
```

- 6 -

```
public class Laticinio {  
public class Laticinio extends Alimento {  
    private int temperatura; // encapsulamento  
    public Laticinio (String nome, String codigo, int temperatura) {  
        super (nome, codigo);  
        this.temperatura = temperatura;  
    } // constructor ^^^  
    public void setTemperatura (int temperatura) { // setter  
        this.temperatura = temperatura;  
    }  
    public int getTemperatura () { // getter  
        return this.temperatura;  
    }  
}  
  
public class Bebida extends Alimento {  
    private String sabor; // encapsulamento  
    public Bebida (String nome, String codigo, String sabor) {  
        super (nome, codigo);  
        this.sabor = sabor;  
    } // constructor ^^^  
    public void setSabor (String sabor) { // setter  
        this.sabor = sabor;  
    }  
    public String void getSabor () { // getter  
        return this.sabor;  
    }  
    public void String demonstrar Sabor () {  
        System.out.println ("o sabor de " + this.nome + " e "  
    }  
}
```



- 7 -

public class Main {

public static void main (String[] args) {

Bevinda refri = new Bevinda ("Fanta", "abO", "lavasa");

refri.<sup>st</sup>calorias = 5500;

refri.demonstrar Sabor();

}

}

+ this.sabor);