Fatec São Caetano do Sul – Antônio Russo

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - ADS		
AVALIAÇÃO OFICIAL	**DISCIPLINA: Programação Orientada a Objetos**	**NOTA**
DATA: 08/10/2020 **☒ P1 ☐ P2 ☐ P3**	**TURMA: 4º ADS – Manhã** **PROFESSOR: MSc Flávio Viotti**	
ALUNO: Felix Petiz Bonilho	**RA:**1680481911002	
ALUNO: Caroline Bognar da Silva	**RA:**1680481911053	

INSTRUMENTO DE AVALIAÇÃO:

- **Os programas devem EXCLUSIVAMENTE serem confeccionados utilizando a linguagem Java**
- **Cada solução dos exercícios deverá ser colocada logo abaixo da questão respectiva**
- **Programas que contenham erro de compilação serão 100% desconsiderados, uma vez que você pode executar esses programas no Netbeans**
- **Dê preferência para resolver os exercícios utilizando os conteúdos ministrados nas aulas. Caso algum exercício seja resolvido com comandos que não foram discutidos em sala de aula, existe a possibilidade de o professor perguntar para algum aluno do grupo como aquele comando funciona, se o mesmo não souber responder, a questão será cancelada.**
- **Nas suas resposta use a cor VERMELHA**
- **Alunos em DP NÃO Presencial não podem fazer esta prova**
  A prova é composta por 12 questões

Gabarito para as questões de múltipla escolha (2,4 pontos):

Questão 1	Questão 2	Questão 3	Questão 4	Questão 5	Questão 6
a	a	b	c	c	d

1. Em relação ao uso de arrays na linguagem Java, avalie as afirmativas a seguir.

   I - Um array é um grupo de variáveis que contém valores todos do mesmo tipo.
   II - O primeiro elemento em cada array tem um índice um.
   III - Um arraylist é semelhante a um array, mas pode ser dinamicamente redimensionado.
   Das afirmativas acima, apenas:
   a. I e III estão corretas.
   b. I está correta.
   c. II está correta.
   d. III está correta.
   e. I e II estão corretas.

2. Dadas três classes, "Funcionário", "Mensalista", e "Horista", nestas três classes está definido o método "calcularFolhaPagamento()", sendo que para cada classe, a forma de cálculo implementada na referida função se dá de forma diferente. Isto é possível graças ao recurso de polimorfismo presente na programação orientada a objetos. Sendo assim, pergunta-se: Qual é o tipo de relacionamento entre as classes supracitadas?
   a. Generalização, pois métodos polimórficos apenas podem ser implementados neste tipo de relacionamento.
   b. Agregação, pois não é possível criar métodos polimórficos em outro tipo de relacionamento.
   c. Composição, pois métodos polimórficos apenas podem ser implementados neste tipo de relacionamento.

d. Generalização, sendo que polimorfismo é o recurso que permite que classes herdem atributos e métodos de outras classes.

3. Assinale a alternativa correta para: "Você precisa armazenar diversos nomes dentro de uma coleção. Esses nomes são de pessoas entrevistadas na rua. Sabendo que existem homônimos, não será permitido o armazenamento de dados repetidos. Para resolver essa situação de armazenagem você utilizaria qual tipo de classe de coleção?".
   a. ( ) Collection
   b. ( ) HashSet
   c. ( ) HashMap
   d. ( ) ArrayList
   e. ( ) LinkedList

4. Uma das características da programação orientada a objetos está relacionada com a proteção dos atributos internos dos objetos contra modificações diretas. As alterações dos atributos devem ocorrer por meio de métodos adequados, criados para acesso e modificação desses atributos. Essa característica é conhecida como

   a. polimorfismo.
   b. sobrecarga de operador.
   c. encapsulamento.
   d. herança.
   e. generalização.

5. Na linguagem Java, são modificadores para controle de acesso às variáveis e aos métodos de uma classe:
   a. protected e static.
   b. private e static.
   c. public e private.
   d. static e public.
   e. final e static.

6. Em POO (Programação Orientada a Objetos), dizer que a classe A estende a classe B é o mesmo que dizer que:

   a. as classes A e B são irmãs.
   b. a classe B é subclasse de A;
   c. a classe A é superclasse de B;
   d. a classe A é derivada de B;
   e. a classe B é derivada de A;

7. (1,0 ponto) Qual a finalidade dos seguintes termos:
   a. super. :
      A palavra super é usada para acessar os métodos e atributos da superclasse, de acordo com suas visibilidades, a partir de uma subclasse.

   b. Agregação :
      A agregação representa uma relação entre objetos parte-todo, ou seja, em que um objeto compõe, ao menos em parte, o outro, considerando que a não existência do objeto todo acarretara a não existência do objeto parte.

   c. Casting :
      Casting é quando uma variável de um tipo é moldada para outro, isto é, quando ocorre uma conversão entre tipos de variáveis que não ocorreria naturalmente.

   d. this() :
      A palavra 'this' seguida de um parênteses ('this()') é utilizada para acessar um dos métodos construtores da própria classe em que foi chamada.

8. (2,0 ponto) Crie uma classe chamada Ingresso que possui um valor em reais, um método construtor que recebe como parâmetro o valor e um método imprimeValor().

```
public class Ingresso {
    private float valReais;
```

```
    /**
     * Imprime na tela o valor do ingresso
     */
    public void imprimeValor() {
        System.out.println("R$" + valReais);
    }

    //Construtor
    public Ingresso(float valReais) {
        this.valReais = valReais;
    }

    //Getters & Setters
    public float getValReais() {
        return valReais;
    }

    public void setValReais(float valReais) {
        this.valReais = valReais;
    }
}
```

a. crie uma classe VIP que herda Ingresso e possui um valor adicional. Crie um método que retorne o valor de ingresso VIP (com o adicional incluído);

```
public class Vip extends Ingresso{
    private float valAdicional;

    /**
     * Retorna o valor do ingresso VIP
     * @return valor do igresso + valor adicional do ingresso VIP
     */
    public float valorVip() {
        return valAdicional + getValReais();
    }

    //Construtor
    public Vip(float valReais) {
        super(valReais);
    }

    //Getters & Setters
    public float getValAdicional() {
        return valAdicional;
    }

    public void setValAdicional(float valAdicional) {
        this.valAdicional = valAdicional;
    }
}
```

b. crie uma classe Normal que herda ingresso e que possui um método que imprime "Ingresso Normal";

```
public class Normal extends Ingresso{
    /**
```

```
     * Imprime na tela o valor total do ingresso normal
     */
    public void imprimeIngressoNormal() {
        System.out.println("Valor do Ingresso Normal:");
        imprimeValor();
    }

    //Construtor
    public Normal(float valReais) {
        super(valReais);
    }
}
```

c. crie uma classe chamada CamaroteInferior (que possui a localização do ingresso e métodos para acessar e imprimir esta localização) e uma classe CamaroteSuperior, que é mais cara (possui um valor adicional). Esta última possui um método para retornar o valor do ingresso. Ambas as classes herdam da classe VIP.

```
public class CamaroteInferior extends Vip{
    private String localizacao;

    /**
     * Imprime na tela a localização do camarote inferior
     */
    public void imprimeLocalizacao() {
        System.out.println("Localização: " + localizacao);
    }

    //Construtor
    public CamaroteInferior(float valReais) {
        super(valReais);
    }

    //Getters & Setters
    public String getLocalizacao() {
        return localizacao;
    }

    public void setLocalizacao(String localizacao) {
        this.localizacao = localizacao;
    }
}
```

```
public class CamaroteSuperior extends Vip{
    private float valAdicionalC;

    /**
     * Retorna o valor total do camarote superior
     * @return valor do ingresso VIP + valor adicional do camarote superior
     */
    public float valorIngresso() {
        return valAdicionalC + valorVip();
    }

    //Construtor
```

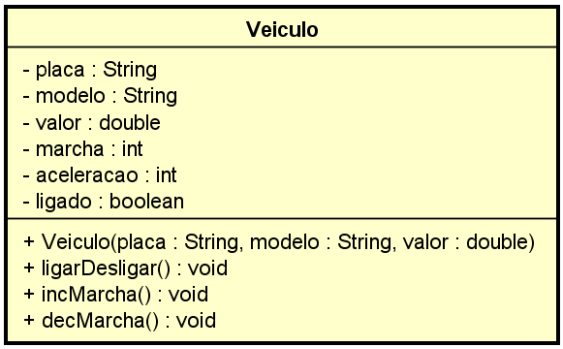
```
    public CamaroteSuperior(float valReais) {
        super(valReais);
    }

    //Getters & Setters
    public float getValAdicionalC() {
        return valAdicionalC;
    }

    public void setValAdicionalC(float valAdicionalC) {
        this.valAdicionalC = valAdicionalC;
    }
}
```

9. (2,0 ponto) Dada as classes para implementação a seguir:

No projeto implemente a classe Veiculo conforme a definição acima. Implemente todos os métodos, inclusive os getters e setters, conforme as definições abaixo.



a. Ao instanciar um veículo este deve ficar com ligado = false, aceleração = 0 e marcha = 0
b. Ao ligar deve ficar com aceleração = 1000 RPM
c. Ao desligar deve ficar com aceleração = 0 RPM
d. Ao trocar de marcha, deve incrementar ou decrementar marcha uma a uma, sendo o mínimo 0 (neutro) e o máximo 6(a) marcha
e. Somente permitir ligar ou desligar estando em marcha neutra, ou seja 0(zero)
f. Somente permitir trocar de marcha se o veículo estiver ligado

Exercícios 9a-f

```
public class Veiculo {
    private String placa, modelo;
    private double valor;
    private int marcha, aceleracao;
    private boolean ligado;

    //Métodos
    /**
     * Desliga o carro (ligado = False) caso estiver ligado (ligado = True) e o contrário,
     * sendo que este deve estar na marcha neutra (marcha = 0)
     */
    public void ligarDesligar() {
        if (marcha != 0) return;
        ligado = !ligado;
        if (ligado) {
            aceleracao = 1000;
        } else {
```

```
        aceleracao = 0;
    }
}

/**
 * Aumenta o valor da marcha em 1, com o limite máimo de 6
 */
public void incMarcha() {
    if (ligado && marcha < 6) marcha++;
}

/**
 * Diminui o valor da marcha em 1, com o limite mínimo de 0
 */
public void decMarcha() {
    if (ligado && marcha > 0) marcha--;
}

//Construtor
public Veiculo(String placa, String modelo, double valor) {
    this.placa = placa;
    this.modelo = modelo;
    this.valor = valor;
    ligado = false;
    aceleracao = 0;
    marcha = 0;
}

//Getters & Setters
public String getPlaca() {
    return placa;
}

public void setPlaca(String placa) {
    this.placa = placa;
}

public String getModelo() {
    return modelo;
}

public void setModelo(String modelo) {
    this.modelo = modelo;
}

public double getValor() {
    return valor;
}

public void setValor(double valor) {
    this.valor = valor;
}

public int getMarcha() {
    return marcha;
```

```
    }

    public void setMarcha(int marcha) {
        this.marcha = marcha;
    }

    public int getAceleracao() {
        return aceleracao;
    }

    public void setAceleracao(int aceleracao) {
        this.aceleracao = aceleracao;
    }

    public boolean isLigado() {
        return ligado;
    }

    public void setLigado(boolean ligado) {
        this.ligado = ligado;
    }
}
```

g. Criar um objeto da classe Veículo. O objeto deve ser inicializado usando o método construtor e os dados podem ser fornecidos via "hard code"
h. Permitir o usuário ligar, desligar, e fazer trocar de marcha.

Exercícios 9g-h

```
public static void main(String[] args) {
        Veiculo veiculo1 = new Veiculo("POO-2020", "ferrari", 999999.99);
        System.out.println("Placa: " + veiculo1.getPlaca() + "\nModelo: " +
veiculo1.getModelo()
                + "\nValor: " + veiculo1.getValor() + "\nLigado: " + veiculo1.isLigado()
                + "\nAceleração: " + veiculo1.getAceleracao() + "RPM\nMarcha: " +
veiculo1.getMarcha());

        veiculo1.ligarDesligar();
        System.out.println("\nLigado: " + veiculo1.isLigado() + "\nAceleração: " +
veiculo1.getAceleracao() + "RPM");

        veiculo1.incMarcha();
        System.out.println("\nMarcha: " + veiculo1.getMarcha());

        veiculo1.incMarcha();
        System.out.println("\nMarcha: " + veiculo1.getMarcha());

        veiculo1.ligarDesligar();
        System.out.println("\nLigado: " + veiculo1.isLigado() + "\nAceleração: " +
veiculo1.getAceleracao() + "RPM");

        veiculo1.decMarcha();
        System.out.println("\nMarcha: " + veiculo1.getMarcha());

        veiculo1.decMarcha();
        System.out.println("\nMarcha: " + veiculo1.getMarcha());
```

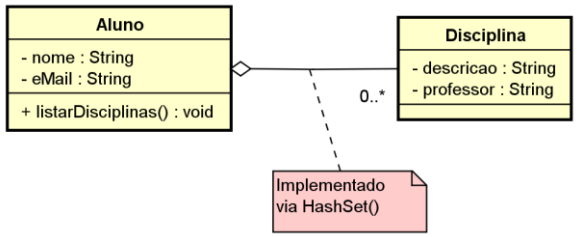
```
        veiculo1.decMarcha();
        System.out.println("\nMarcha: " + veiculo1.getMarcha());

        veiculo1.decMarcha();
        System.out.println("\nMarcha: " + veiculo1.getMarcha());

        veiculo1.ligarDesligar();
        System.out.println("\nLigado: " + veiculo1.isLigado() + "\nAceleração: " +
veiculo1.getAceleracao() + "RPM");

    }
```

10. (1,0 ponto) Assuma que as duas classes já estão criadas. Você precisa implementar somente o método listarDisciplinas(). Uma das regras do negócio é que cada aluno pode possuir diversas disciplinas. Utilize a listagem via Iteradores.




```
public void listarDisciplinas() {
        System.out.println("Disciplinas:");
                Iterator<Disciplina> it = listaDisciplinas.iterator();
                while (it.hasNext()) {
                    Disciplina aux = it.next();
                    System.out.println("\tDescrição: " + aux.getDescricao()
                            + "\n\tProfessor: " + aux.getProfessor() + "\n");
                }
}
```

11. (1,0 ponto) Dado as classes abaixo, implemente somente a classe Disciplina


```
public class Disciplina {
    private String descricao;
    private Professor prof;

    //Construtor
    public Disciplina() {
        prof = new Professor();
    }

    //Getters & Setters
    public String getDescricao() {
```

```
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public Professor getProf() {
        return prof;
    }

    public void setProf(Professor prof) {
        this.prof = prof;
    }
}
```

12. (0,6 ponto) Dado as classes abaixo, que contém erros, faça a correção da mesma:

```
public class ClasseA {
      private String param1;
      public ClasseA(String p) {
            param1 = p;
      }
      //..getters e setters
}

public class ClasseB extends ClasseA {
      private String param1;

      public ClasseB(String p) {
            super(p);
      }

      //..getters e setters
}
```