

# 1a AVALIAÇÃO BIMESTRAL - 4SA - POO 2024 - DIA 02

- **Curso:** ENGENHARIA DE SOFTWARE
- **Série:** 4 S
- **Turma:** A
- **Turno:** NOTURNO
- **Professor(a):** JOÃO CHOMA NETO
- **Horário:** 21:00 – 22:30
- **Disciplina:** PROGRAMAÇÃO ORIENTADA A OBJETOS
- **BIMESTRE:** 1
- **VALOR:** 5,0 PONTOS

## INSTRUÇÕES PARA REALIZAÇÃO DA PROVA:

- É vedado, durante a prova, o porte e/ou o uso de aparelhos sonoros, fonográficos, de comunicação ou de registro eletrônico ou não, tais como: notebooks, celulares, tablets e similares.
- A prova é individual e sem consulta, é permitido o uso da disciplina.
- É obrigatória a permanência do acadêmico 50 MINUTOS em sala de aula após o início da prova
- Não será permitida a entrada na sala de aula após 10 minutos do início da prova.
- É obrigatória a assinatura da lista de presença impressa na qual constam RA, nome e curso.
- O valor de cada questão está ao lado da mesma.
- Em caso de qualquer irregularidade comunicar ao Professor ou fiscal de sala.
- Ao término da prova, levante o braço e aguarde atendimento do professor ou do fiscal.

E-mail \*

ra-23038173-2@alunos.unicesumar.edu.br

NOME DO ACADÊMICO

Antonio Ferreira de Lima

REGISTRO ACADÊMICO - inserir o - \*

23038173-2

PROBLEMA A SER RESOLVIDO

**Problema: Sistema de Gerenciamento de Entregas com Veículos de Diferentes Capacidades**  
Você foi contratado para desenvolver um sistema de gerenciamento de entregas para uma empresa de logística que opera com uma frota diversificada de veículos. Cada veículo tem uma capacidade específica, e o sistema deve gerenciar as entregas, otimizando o uso dos veículos de acordo com o volume e a prioridade das encomendas.

**Requisitos do Sistema:**

**1. Veículos:**

- A empresa utiliza diferentes tipos de veículos para realizar as entregas, e cada veículo tem uma capacidade máxima de carga (em peso e volume) que deve ser respeitada.
  - **Motocicletas:** Usadas para entregas leves e rápidas, com uma capacidade de até 30 kg e 1 m<sup>3</sup> de volume.
  - **Vans:** Podem carregar até 500 kg e 5 m<sup>3</sup> de volume.
  - **Caminhões:** Veículos pesados que suportam até 5000 kg e 30 m<sup>3</sup> de volume.

**2. Encomendas:**

- As encomendas podem ter diferentes tamanhos e prioridades. O sistema deve distribuir as encomendas entre os veículos de maneira eficiente, levando em conta:
  - **Peso e volume** da encomenda.
  - **Prioridade:** Algumas encomendas têm alta prioridade e devem ser entregues o mais rápido possível, utilizando veículos rápidos como motocicletas, se o tamanho permitir.
  - **Prazo de Entrega:** Encomendas com prazos mais apertados devem ser priorizadas na escolha do veículo e na rota.

**3. Distribuição e Otimização:**

- O sistema deve atribuir encomendas aos veículos disponíveis de acordo com suas capacidades.

**4. Gerenciamento de Frota:**

- O sistema deve gerenciar a disponibilidade dos veículos, rastrear o status de cada veículo (disponível, em rota, manutenção), e garantir que os veículos sejam usados de maneira eficiente.
- Veículos que estão em manutenção não podem ser atribuídos a entregas até que estejam disponíveis novamente.

**Requisitos Funcionais:**

- O sistema deve permitir:
  - Gerenciar os veículos da frota, atribuindo encomendas de acordo com o peso, volume e prioridade.
  - Gerenciar a disponibilidade dos veículos e controlar quando estão em manutenção ou em rota.
  - Gerar relatórios de entregas.

**Regras de Negócio:**

1. **Capacidade dos Veículos:** Cada veículo tem um limite de peso e volume que deve ser respeitado.
2. **Prioridade de Entregas:** Encomendas com alta prioridade devem ser atribuídas a veículos rápidos.
3. **Gestão de Frota:** O sistema deve impedir que veículos em manutenção sejam atribuídos a novas entregas.

**Requisitos para avaliação:**

1. **IMPLEMENTE** uma estrutura de classes que represente os diferentes tipos de eventos e as funcionalidades compartilhadas entre eles (2,0 pontos).
2. **ELABORE** um fluxo de execução no método **main** que represente o sistema (1,0 ponto).
3. **USE** boas práticas de orientação a objetos, incluindo herança E/OU encapsulamento E/OU polimorfismo (2,0 pontos).
4. Toda a lógica deve ser implementada.

**Pontos de Avaliação:**

- Estrutura e clareza do código.
- Aplicação correta dos conceitos de orientação a objetos (encapsulamento, herança, polimorfismo).
- Capacidade de implementar lógica de negócios.

Para apoiar a implementação segue casos de teste.

Não é obrigatório a utilização dos testes.

**Esses testes não cobrem o sistema por completo.**

```
import org.junit.jupiter.api.Test;
```

```
import static org.junit.jupiter.api.Assertions.*;
```

```
public class SistemaGerenciamentoEntregasTest {
```

```
    Motocicleta moto = new Motocicleta();
```

```
    Van van = new Van();
```

```
    Caminhao caminhao = new Caminhao();
```

```
    @Test
```

```
    public void testAdicionarVeiculo() {
```

```
        assertEquals("Motocicleta", moto.tipoVeiculo());
```

```
        assertEquals("Van", van.tipoVeiculo());
```

```
        assertEquals("Caminhão", caminhao.tipoVeiculo());
```

```
    }
```

```
    @Test
```

```
    public void testAtribuirEncomendaVeiculoCapacidade() {
```

```
        Encomenda encomendaMoto = new Encomenda(10, 0.5, "Alta");
```

```
        assertTrue(moto.podeCarregar(encomendaMoto.getPeso(), encomendaMoto.getVolume()));
```

```
        Encomenda encomendaVan = new Encomenda(400, 4, "Baixa");
```

```
        assertTrue(van.podeCarregar(encomendaVan.getPeso(), encomendaVan.getVolume()));
```

```
        Encomenda encomendaCaminhao = new Encomenda(4500, 25, "Média");
```

```
        assertTrue(caminhao.podeCarregar(encomendaCaminhao.getPeso(),
```

```
encomendaCaminhao.getVolume()));
```

```
    }
```

```
    @Test
```

```
    public void testAtribuirEncomendaVeiculoExcedeCapacidade() {
```

```
        Encomenda encomendaGrande = new Encomenda(35, 2, "Alta");
```

```
        assertFalse(moto.podeCarregar(encomendaGrande.getPeso(), encomendaGrande.getVolume()));
```

```
        Encomenda encomendaMuitoGrande = new Encomenda(5500, 32, "Baixa");
```

```
        assertFalse(caminhao.podeCarregar(encomendaMuitoGrande.getPeso(),
```

```
encomendaMuitoGrande.getVolume()));
```

```
    }
```

```
    @Test
```

```
    public void testVeiculoEmManutencao() {
```

```
        moto.colocarEmManutencao();
```

```
        assertFalse(moto.podeCarregar(10, 0.5));
```

```
van.colocarEmManutencao();  
assertFalse(van.podeCarregar(100, 1));
```

```
moto.marcarDisponivel();  
assertTrue(moto.podeCarregar(10, 0.5));
```

```
}  
}
```

## ESPAÇO 01 - PARA INCLUIR AS CLASSES \*

```
package org.example;
```

```
import java.awt.*;
```

```
public abstract class Veiculo {
    private boolean disponibilidade=true;
    private Encomenda EncoemndaAtual;

    public void setDisponibilidade(boolean disponibilidade) {
        this.disponibilidade = disponibilidade;
    }

    public boolean VerificarDisponibilidade(){
        return disponibilidade;
    }

    public abstract boolean PodeCaregar(Encomenda A);
    public boolean AtribuirEncomenda(Encomenda A){
        if(this.VerificarDisponibilidade()){
            if(PodeCaregar(A)){
                this.EncoemndaAtual = A;
                this.setDisponibilidade(false);
                System.out.println("Encoemnda atribuida");
                return true;

            } else {
                System.out.println("carga da encomenda excede limite do veiculo");
                return false;
            }
        } else {
            System.out.println("veiculo indisponivel");
            return false;
        }
    }
}
```

```
package org.example;
```

```
public class Van extends Veiculo{
    //omitindo constructor mais uma vez AAAAAAA
    @Override
    public boolean PodeCaregar(Encomenda A) {
        if(A.getPeso()<=500||A.getVolume()<=5){
```



```
        return true;
    }
    return false;
}
}
```

```
package org.example;
```

```
public class Motocicleta extends Veiculo{
//omitindo constructor dnv
    @Override
    public boolean PodeCaregar(Encomenda A) {
        if(A.getPeso()<=30||A.getVolume()<=1){
            return true;
        }
        return false;
    }
}
```

```
package org.example;
```

```
public class Caminhao extends Veiculo{
    //vou omitir o constructor

    @Override
    public boolean PodeCaregar(Encomenda A) {
        if(A.getPeso()<=5000||A.getVolume()<=30){
            return true;
        }
        return false;
    }
}
```

```
package org.example;
```

```
public class Encomenda {
    private int peso;
    private int volume;
    private boolean prioridade;

    public Encomenda(int peso, int volume, boolean prioridade) {
```

```

        this.peso = peso;
        this.volume = volume;
        this.prioridade = prioridade;
    }

    public int getPeso() {
        return peso;
    }

    public double getVolume() {
        return volume;
    }

    public boolean isPrioridade() {
        return prioridade;
    }

    public void setPeso(int peso) {
        this.peso = peso;
    }

    public void setVolume(int volume) {
        this.volume = volume;
    }

    public void setPrioridade(boolean prioridade) {
        this.prioridade = prioridade;
    }

    public void PrioridadeConfirma(){
        if(this.isPrioridade()){
            System.out.println("é prioridade");
            return;
        } else {
            System.out.println("não é prioridade");
            return;
        }
    }
}

```

```

package org.example;

```

```

public class Main {
    public static void main(String[] args) {
        Veiculo moto = new Motocicleta();
    }
}

```

```
Veiculo van = new Van();  
Veiculo cam = new Caminhao();
```

```
Encomenda encomenda1 = new Encomenda(10,1,false);  
Encomenda encomenda2 = new Encomenda(30,5,false);  
Encomenda encomenda3 = new Encomenda(5000,20,false);  
Encomenda encomenda4 = new Encomenda(40,10,true);
```

```
moto.AtribuirEncomenda(encomenda1);  
van.AtribuirEncomenda(encomenda3);  
cam.AtribuirEncomenda(encomenda2);  
van.AtribuirEncomenda(encomenda4);  
moto.AtribuirEncomenda(encomenda3);//moto agora ja esca carregando a encomenda1
```

```
encomenda1.PrioridadeConfirma();  
encomenda4.PrioridadeConfirma();
```

```
van.VerificarDisponibilidade();  
van.setDisponibilidade(false);  
van.AtribuirEncomenda(encomenda1);
```

```
}  
}
```

ESPAÇO 02 - PARA INCLUIR AS CLASSES

---

ESPAÇO 03 - PARA INCLUIR AS CLASSES

---

ESPAÇO 04 - PARA INCLUIR AS CLASSES

---

ESPAÇO 05 - PARA INCLUIR AS CLASSES

Este conteúdo não foi criado nem aprovado pelo Google.

Google Formulários