

1a AVALIAÇÃO BIMESTRAL - 4SA - POO 2024

- Curso: ENGENHARIA DE SOFTWARE
- Série: 4 S
- Turma: A
- Turno: NOTURNO
- Professor(a): JOÃO CHOMA NETO
- Horário: 19:00 – 20:30
- 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 git 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 *

lucas.kleindearaujo@gmail.com

NOME DO ACADÊMICO

Lucas Klein

REGISTRO ACADÊMICO - inserir o - *

23350021-2

PROBLEMA A SER RESOLVIDO

Problema: Sistema de Gerenciamento de Vagas de Estacionamento em um Shopping

Você foi contratado para desenvolver um sistema de gerenciamento de vagas de estacionamento para um shopping que deseja otimizar a utilização de suas vagas e fornecer um serviço mais eficiente para seus clientes.

Requisitos do Sistema:**1. Tipos de Vagas:**

- O shopping possui diferentes tipos de vagas:
 - **Vaga Regular:** Vaga comum para veículos de pequeno porte.
 - **Vaga VIP:** Reservada para clientes VIP que possuem cartão de fidelidade.
 - **Vaga para Veículos Elétricos:** Exclusiva para veículos que necessitam de carregamento.
 - **Vaga para Deficientes:** Exclusiva para pessoas com deficiência.

2. Gerenciamento de Entrada e Saída:

- O sistema deve registrar a entrada e a saída dos veículos, associando-os ao tipo de vaga utilizado.
- O tempo de permanência no estacionamento deve ser calculado e o valor a ser cobrado deve seguir as seguintes regras:
 - **Vaga Regular:** Tarifa de R\$ 10,00 por hora.
 - **Vaga VIP:** Tarifa de R\$ 5,00 por hora para clientes VIP.
 - **Vaga para Veículos Elétricos:** Tarifa de R\$ 12,00 por hora, com uma tarifa adicional pelo uso do carregador elétrico.
 - **Vaga para Deficientes:** Gratuita, mas requer comprovação de documentação.

3. Reservas de Vagas:

- Clientes VIP podem reservar vagas VIP com antecedência.
- Veículos elétricos podem reservar vagas com carregador elétrico.
- Vagas regulares não podem ser reservadas, sendo ocupadas por ordem de chegada.

4. Gerenciamento de Fidelidade:

- Clientes VIP têm acesso a benefícios como tarifas reduzidas.

Requisitos Funcionais:

- O sistema deve permitir:
 - Registrar entrada e saída de veículos, associando o tipo de vaga utilizada e calculando o tempo de permanência.
 - Gerenciar reservas de vagas VIP e para veículos elétricos.
 - Implementar diferentes comportamentos para os tipos de vaga, sem duplicação de código (aplicando polimorfismo e herança).

Regras de Negócio:

1. **Vaga VIP:** Somente clientes VIP podem reservar essas vagas, e eles têm acesso a tarifas reduzidas.
2. **Vaga para Deficientes:** Gratuita, mas exige comprovação de documentação no momento da entrada.
3. **Vaga para Veículos Elétricos:** Aplica-se uma tarifa adicional para o uso do carregador, além da tarifa de estacionamento.

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

class EstacionamentoTest {

    Estacionamento estacionamento = new Estacionamento();

    @Test
    void testClienteVIPodeReservarVagaVIP() {
        Cliente clienteVIP = new Cliente(true);
        Vaga vagaVIP = new VagaVIP();

        boolean resultadoReserva = estacionamento.reservarVaga(vagaVIP, clienteVIP);
        assertTrue(resultadoReserva);
    }

    @Test
    void testClienteNaoVIPNaoPodeReservarVagaVIP() {
        Cliente clienteNaoVIP = new Cliente(false);
        Vaga vagaVIP = new VagaVIP();

        boolean resultadoReserva = estacionamento.reservarVaga(vagaVIP, clienteNaoVIP);
        assertFalse(resultadoReserva);
    }

    @Test
    void testClienteComVeiculoEletricoPodeReservarVagaEletrica() {
        Cliente clienteEletrico = new Cliente(true);
        Vaga vagaEletrica = new VagaEletrica();

        boolean resultadoReserva = estacionamento.reservarVaga(vagaEletrica, clienteEletrico);
        assertTrue(resultadoReserva);
    }

    @Test
    void testClienteNaoVIPodeReservarVagaEletrica() {
        Cliente clienteNaoVIP = new Cliente(false);
        Vaga vagaEletrica = new VagaEletrica();

        boolean resultadoReserva = estacionamento.reservarVaga(vagaEletrica, clienteNaoVIP);
        assertTrue(resultadoReserva);
    }
}
```

```
@Test
void testNaoPodeReservarVagaRegular() {
    Cliente clienteVIP = new Cliente(true);
    Vaga vagaRegular = new VagaRegular();

    boolean resultadoReserva = estacionamento.reservarVaga(vagaRegular, clienteVIP);
    assertFalse(resultadoReserva);
}
}
```

ESPAÇO 01 - PARA INCLUIR AS CLASSES *

```
package org.example;
```

```
public class Main {  
    public static void main(String[] args){  
        Vaga vagaRegular = new VagaRegular();  
        Vaga vagaEletrica = new VagaEletrica();  
        Vaga vagaVIP = new VagaVIP();  
        Vaga vagaDeficiente = new VagaDeficiente();  
  
        Estacionamento estacionamento = new Estacionamento();  
  
        estacionamento.ocuparVaga(10);  
        estacionamento.sairVaga(12);  
  
        estacionamento.cobrarVaga(Vaga vagaVIP);  
  
    }  
}
```

```
package org.example;
```

```
public class Estacionamento {  
  
    private double tempoDePermanencia;  
    private double horaEntrada;  
    private double horaSaida;  
  
    public Estacionamento() { }  
  
    public boolean reservarVagacomAntecedencia(Vaga tipoVaga, Cliente ClienteEhVip) {  
        if (ClienteEhVip.isEhVip()){  
            return true;  
        } else {  
            return false;  
        }  
    }  
  
    public double cobrarVaga(Vaga tipoVaga){  
        return tipoVaga.getTarifa() * tempoDePermanencia;  
    }  
  
    public void ocuparVaga(double horaAtual){  
        horaEntrada = horaAtual;  
    }  
}
```



```
public void sairVaga(double horaAtual){  
    horaSaida = horaAtual;  
    tempoDePermanencia = horaSaida - horaEntrada;  
}
```

```
}
```

```
package org.example;
```

```
public class Cliente {  
    private boolean ehVip;  
  
    public Cliente(boolean ehVip) {  
        this.ehVip = ehVip;  
    }
```

```
    public boolean isEhVip() {  
        return ehVip;  
    }  
}
```

ESPAÇO 02 - PARA INCLUIR AS CLASSES

```
package org.example;
```

```
public abstract class Vaga {  
    private int tarifa;  
  
    public Vaga(int tarifa) {  
        this.tarifa = tarifa;  
    }  
  
    public int getTarifa() {  
        return tarifa;  
    }  
}
```

```
package org.example;
```

```
public class VagaDeficiente extends Vaga {  
    private double tempoDePermanencia;  
  
    public VagaDeficiente() {  
        super(0);  
    }  
}
```

```
package org.example;
```

```
public class VagaEletrica extends Vaga {  
    private boolean precisaCarregar;  
    private int tarifaExtra = 10;  
  
    public VagaEletrica() {  
        super(12);  
    }  
  
    @Override  
    public double cobrarVaga(int tarifa, double tempoOcupado) {  
  
        return (tarifa * tempoOcupado) + tarifaExtra;  
    }  
}
```

ESPAÇO 03 - PARA INCLUIR AS CLASSES

```
package org.example;
```

```
public class VagaRegular extends Vaga {  
    public VagaRegular() {  
        super(10);  
    }  
}
```

```
package org.example;
```

```
public class VagaVIP extends Vaga {  
    private double tempoDePermanencia;  
  
    public VagaVIP() {  
        super(12);  
    }  
}
```

ESPAÇO 04 - PARA INCLUIR AS CLASSES

ESPAÇO 05 - PARA INCLUIR AS CLASSES

Este conteúdo não foi criado nem aprovado pelo Google.

Google Formulários