Projeto: Sistema para gestão de estacionamento

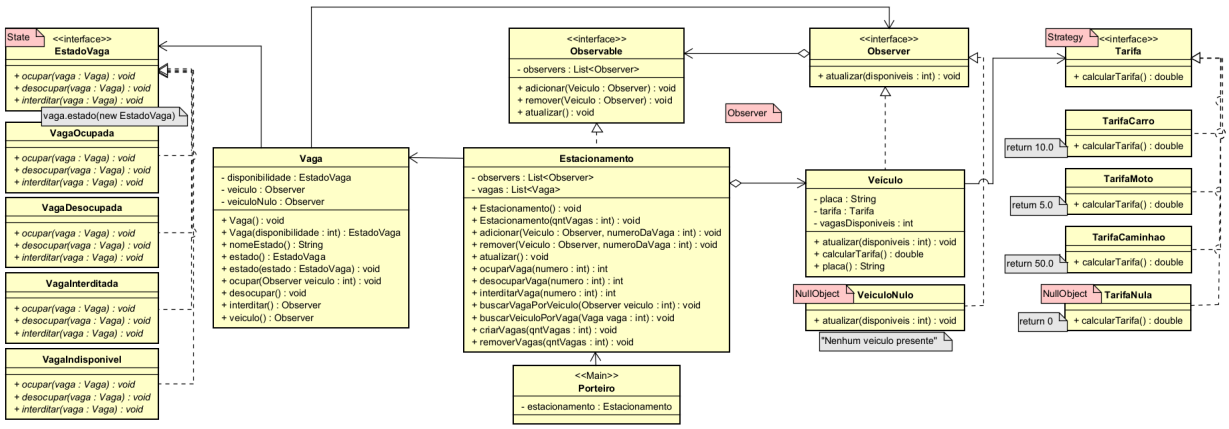
Introdução

Este documento apresenta e descreve um sistema para gestão de estacionamento que utiliza as estratégias Strategy, State, NullObject e Observer.

Descrição

Este sistema define um estacionamento, com vagas e veículos, que utiliza os padrões de projeto Strategy, para diferenciar entre tipos de veículos para o pagamento da tarifa;
State, para gerenciar o estado da vaga (ocupada, desocupada ou em-manutenção);
NullObject, para possibilitar a entrada de um veículo não previsto;
Observer, para manter o cada veículo atualizado quanto as vagas disponíveis.

Diagrama de classes



Código

Interface Observable

Define uma lista de observadores e os métodos para adicionar/inscrever e remover/desinscrever observadores e atualizá-los;

```
public interface Observable {
    ArrayList<Observer> observers = new ArrayList<Observer>();
    public void adicionar(Observer Veiculo);
    public void remover(Observer Veiculo);
    public void atualizar();
}
```

Classe Estacionamento

É um Observable, portanto pode inscrever, desinscrever e atualizar seus observadores quanto ao número de vagas disponíveis.

Além disso, age como controlador, pode ocupar, desocupar e interditar as vagas.

```java
public class Estacionamento implements Observable {
    private ArrayList<Observer> observers = new ArrayList<Observer>();
    private ArrayList<Vaga> vagas = new ArrayList<Vaga>();

    public Estacionamento(){
        vagas.add(new Vaga(new VagaIndisponivel()));//adiciona uma vaga nula para o index 0
    }
    public Estacionamento(int qntVagas){
        vagas.add(new Vaga(new VagaIndisponivel()));//adiciona uma vaga nula para o index 0
        criarVagas(qntVagas);
    }

    @Override
    public void adicionar(Observer Veiculo) {
        observers.add(Veiculo);
    }
    @Override
    public void remover(Observer Veiculo) {
        observers.remove(Veiculo);
    }

    public void ocuparVaga(int numero, Observer veiculo){
        try{
            vagas.get(numero).ocupar(veiculo);
            atualizar();
        } catch (IndexOutOfBoundsException e) {
            System.out.println("A vaga " +numero+ " não existe!");
        }
    }
    public void desocuparVaga(int numero){
        try{
            vagas.get(numero).desocupar();
            atualizar();
        } catch (IndexOutOfBoundsException e) {
            System.out.println("A vaga " +numero+ " não existe!");
        }
    }
    public void interditarVaga(int numero){
        try{
            vagas.get(numero).interditar();
            atualizar();
        } catch (IndexOutOfBoundsException e) {
            System.out.println("A vaga " +numero+ " não existe!");
        }
    }

    @Override
    public void atualizar() {
        int vagasDisponiveis = (int) vagas.stream().filter(vaga -> vaga.estado() instanceof VagaDesocupada).count();
        System.out.println(x:"Notificação para veículos{");
        for (Observer o : observers) {
            o.atualizar(vagasDisponiveis);
        }
        System.out.println(x:"}");
    }
```

Pode também adicionar e remover vagas a qualquer momento, além de buscar uma vaga ou veículo específico.

```
    public void criarVagas(int qntVagas){
        for (int i = 1; i <= qntVagas; i++) {
            vagas.add(new Vaga());//adiciona as vagas
        }
    }
    public void removerVagas(int qntVagas){
        try {
            for (int i = 0; i < qntVagas; i++) {
                vagas.remove(vagas.size() - 1); // Tenta remover a última vaga
            }
            System.out.println(qntVagas + " vagas removidas. Novo total de vagas: " + vagas.size());
        } catch (IndexOutOfBoundsException e) {
            System.out.println("Não é possível remover " + qntVagas + " vagas. Apenas " + vagas.size() + " vagas disponíveis.");
        }
    }

    public int buscarVagaPorVeiculo(Observer veiculo) {
        for (Vaga vaga : vagas) {
            if (vaga.veiculo() == veiculo) {
                return vagas.indexOf(vaga); // Retorna o numero da vaga onde o veículo está estacionado
            }
        }
        return 0;
    }
    public Observer buscarVeiculoPorVaga(Vaga vaga){
        return vaga.veiculo();
    }
}
```

Interface Observer

Define método para atualizar os observadores.

```
public interface Observer {
    public void atualizar(int disponiveis);
}
```

Classe Veículo

Implementa o método de atualização das vagas disponíveis, também registra sua placa e tarifa.

```
public class Veiculo implements Observer {
    private String placa;
    private Tarifa tarifa;
    private int vagasDisponiveis;

    public Veiculo(String placa, Tarifa tarifa){
        this.placa = placa;
        this.tarifa = tarifa;
    }

    public String placa(){
        return placa;
    }

    @Override
    public void atualizar(int disponiveis) {
        System.out.println("Notificação para veículo de placa: "+placa+"\n  Total de vagas disponiveis: "+disponiveis);
        vagasDisponiveis = disponiveis;
    }
    public double calcularTarifa(){
        return tarifa.calcularTarifa();
    }
}
```

Classe VeiculoNulo

Não implementa métodos adicionais, apenas informa impossibilidade de atualização.

```java
public class VeiculoNulo implements Observer{

    @Override
    public void atualizar(int disponiveis) {
        System.out.println(x:"Veículo não notificável.");
    }

}
```

Interface Tarifa

Define método para calculo/retorno do valor da tarifa.

Define uma “Strategy”, o método calcularTarifa será diferente para cada tipo de veículo

```java
public interface Tarifa {
    public double calcularTarifa();
}
```

Classes TarifaMoto/TarifaCarro/TarifaCaminhao - Estratégias diferentes para cada tipo de veículo.

implementam o método calcularTarifa retornando valor da tarifa respectivamente.

```java
public class TarifaMoto implements Tarifa {
    @Override
    public double calcularTarifa() {
        return 5;
    }
}
```

```java
public class TarifaCarro implements Tarifa {
    @Override
    public double calcularTarifa() {
        return 10;
    }
}
```

```java
public class TarifaCaminhao implements Tarifa {

    @Override
    public double calcularTarifa() {
        return 50;
    }

}
```

Classe TarifaNula

Estabelece a possibilidade de Tarifa Nula para Veículos não previstos.

```java
public class TarifaNula implements Tarifa {
    @Override
    public double calcularTarifa() {
        return 0;
    }
}
```

Classe Vaga

Guarda seu estado/disponibilidade e veículo que a ocupa, nulo caso não ocupada.

```java
public class Vaga {
    private EstadoVaga disponibilidade;
    private Observer veiculo;
    private Observer veiculoNulo = new VeiculoNulo();
    public Vaga(){
        this.disponibilidade = new VagaDesocupada();
        this.veiculo = veiculoNulo;
    }
    public Vaga(EstadoVaga disponibilidade){
        this.disponibilidade = disponibilidade;
    }

    public String nomeEstado(){
        return disponibilidade.getClass().getSimpleName();
    }
    public EstadoVaga estado(){
        return disponibilidade;
    }
    public void estado(EstadoVaga estado){
        this.disponibilidade = estado;
    }

    public void ocupar(Observer veiculo){
        if (disponibilidade instanceof VagaDesocupada) {
            this.veiculo = veiculo;
            disponibilidade.ocupar(this);
        }else{
            System.out.println(x:"A vaga não pode ser ocupada no momento.");
        }
    }
    public void desocupar(){
        this.veiculo = veiculoNulo;
        disponibilidade.desocupar(this);
    }
    public void interditar(){
        disponibilidade.interditar(this);
    }

    public Observer veiculo(){
        return veiculo;
    }
}
```

Interface EstadoVaga

Define uma interface para State/Estados da vaga, estes, podem alterar o estado da vaga dentro de si, alterando a si mesmos dentro da classe Vaga.

Define métodos para ocupar, desocupar e interditar uma vaga.

```java
public interface EstadoVaga {
    public void ocupar(Vaga vaga);
    public void desocupar(Vaga vaga);
    public void interditar(Vaga vaga);
}
```

Classe VagaDesocupada

Vaga desocupada, não pode ser desocupada novamente, mas pode ser ocupada ou interditada, atualizando estado da vaga.

```java
public class VagaDesocupada implements EstadoVaga {

    @Override
    public void ocupar(Vaga vaga) {
        vaga.estado(new VagaOcupada());
        System.out.println(x:"A vaga foi ocupada");
    }

    @Override
    public void desocupar(Vaga vaga) {
        System.out.println(x:"A vaga ja esta desocupada.");
    }

    @Override
    public void interditar(Vaga vaga) {
        System.out.println(x:"A vaga foi interditada.");
        vaga.estado(new VagaInterditada());
    }

}
```

Classe VagaOcupada

Vaga ocupada, não pode ser ocupada novamento, mas pode ser desocupada e interditada, atualizando estado da vaga.

```java
public class VagaOcupada implements EstadoVaga{

    @Override
    public void ocupar(Vaga vaga) {
        System.out.println(x:"A vaga ja esta ocupada");
    }

    @Override
    public void desocupar(Vaga vaga) {
        vaga.estado(new VagaDesocupada());
        System.out.println(x:"A vaga foi desocupada.");
    }

    @Override
    public void interditar(Vaga vaga) {
        vaga.estado(new VagaInterditada());
        System.out.println(x:"A vaga foi interditada.");
    }

}
```

Classe VagaInterditada

Vaga Interditada, para manutenção ou outra aplicabilidade.

```java
import java.util.Scanner;

public class VagaInterditada implements EstadoVaga {
    Scanner input = new Scanner(System.in);

    @Override
    public void ocupar(Vaga vaga) {
        System.out.println(x:"A vaga esta em manutenção e não pode ser ocupada.");
    }

    @Override
    public void desocupar(Vaga vaga) {
        System.out.println(x:"A vaga esta em manutenção, tem certeza que deseja disponibilizá-la?(true/false)");
        if (input.nextBoolean()) {
            vaga.estado(new VagaDesocupada());
            System.out.println(x:"A Vaga foi desocupada e não está mais interditada.");
        }else{
            System.out.println(x:"Desocupação negada ou valor inserido inválido, a vaga segue interditada.");
        }
    }

    @Override
    public void interditar(Vaga vaga) {
        System.out.println(x:"A Vaga ja esta interditada.");
    }

}
```

Classe VagaIndisponivel

Vaga nula/inexistente, incapaz de qualquer ação.

```java
public class VagaIndisponivel implements EstadoVaga{

    @Override
    public void ocupar(Vaga vaga) {
        System.out.println(x:"Vaga não encontrada ou indisponível.");
    }

    @Override
    public void desocupar(Vaga vaga) {
        System.out.println(x:"Vaga não encontrada ou indisponível.");
    }

    @Override
    public void interditar(Vaga vaga) {
        System.out.println(x:"Vaga não encontrada ou indisponível.");
    }

}
```

Teste na classe Main

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {

        Estacionamento estacionamento = new Estacionamento(qntVagas:25);

        Observer carro = new Veiculo(placa:"ABC-1234", new TarifaCarro());
        Observer moto = new Veiculo(placa:"DEF-5678", new TarifaMoto());
        Observer quadriciculo = new Veiculo(placa:"GHI-9101", new TarifaMoto());
        Observer bicicleta = new Veiculo(placa:"---", new TarifaNula());
        Observer skate = new VeiculoNulo();

        estacionamento.adicionar(carro);
        estacionamento.ocuparVaga(numero:1, carro);
        estacionamento.adicionar(moto);
        estacionamento.ocuparVaga(numero:2, moto);
        estacionamento.adicionar(quadriciculo);
        estacionamento.ocuparVaga(numero:3, bicicleta);
        estacionamento.adicionar(skate);

        estacionamento.desocuparVaga(numero:1);

        estacionamento.interditarVaga(numero:4);
        estacionamento.ocuparVaga(numero:4, skate);
        estacionamento.remover(skate);

        estacionamento.ocuparVaga(numero:27, quadriciculo);
        estacionamento.removerVagas(qntVagas:20);
    }
}
```

Output

Como estacionamento é o Observable, ele notifica os veículos da quantidade de vagas disponíveis sempre que uma alteração neste número é feita.

```
A vaga foi ocupada
Notificação para veículos{
Notificação para veículo de placa: ABC-1234
  Total de vagas disponiveis: 24
}
A vaga foi ocupada
Notificação para veículos{
Notificação para veículo de placa: ABC-1234
  Total de vagas disponiveis: 23
Notificação para veículo de placa: DEF-5678
  Total de vagas disponiveis: 23
}
A vaga foi ocupada
Notificação para veículos{
Notificação para veículo de placa: ABC-1234
  Total de vagas disponiveis: 22
Notificação para veículo de placa: DEF-5678
  Total de vagas disponiveis: 22
Notificação para veículo de placa: GHI-9101
  Total de vagas disponiveis: 22
}
A vaga foi desocupada.
Notificação para veículos{
Notificação para veículo de placa: ABC-1234
  Total de vagas disponiveis: 23
Notificação para veículo de placa: DEF-5678
  Total de vagas disponiveis: 23
Notificação para veículo de placa: GHI-9101
  Total de vagas disponiveis: 23
Veículo não notificável.
}
A vaga foi interditada.
Notificação para veículos{
Notificação para veículo de placa: ABC-1234
  Total de vagas disponiveis: 22
Notificação para veículo de placa: DEF-5678
  Total de vagas disponiveis: 22
Notificação para veículo de placa: GHI-9101
  Total de vagas disponiveis: 22
Veículo não notificável.
}
A vaga não pode ser ocupada no momento.
Notificação para veículos{
Notificação para veículo de placa: ABC-1234
  Total de vagas disponiveis: 22
Notificação para veículo de placa: DEF-5678
  Total de vagas disponiveis: 22
Notificação para veículo de placa: GHI-9101
  Total de vagas disponiveis: 22
Veículo não notificável.
}
A vaga 27 não existe!
```

```
20 vagas removidas. Novo total de vagas: 6
Notificação para veículos{
Notificação para veículo de placa: ABC-1234
  Total de vagas disponiveis: 2
Notificação para veículo de placa: DEF-5678
  Total de vagas disponiveis: 2
Notificação para veículo de placa: GHI-9101
  Total de vagas disponiveis: 2
}
```