Universidade de Brasília

SISTEMA DE MOBILIDADE URBANA (RIDE-SHARING)

LUCAS ABDALLA NERY

Matrícula: 251012304

THOMAZ MARRA MARTINS

Matrícula: 251039845

SAMUEL CARVALHO DE SOUSA

Matrícula: 241026010

DISCIPLINA: ORIENTAÇÃO A OBJETOS

DOCENTE: ANDRE LUIZ PERON MARTINS LANNA

BRASÍLIA - DF

28 de novembro de 2025

1. Introdução

Objetivo

O objetivo do trabalho é desenvolver um sistema em Java que aplique os conteúdos, relacionados a programação orientada a objetos, aprendidos em aula, garantindo que modularidade, encapsulamento, herança, polimorfismo e tratamento de exceções personalizadas sejam levados em consideração.

Este relatório tem como fim documentar o desenvolvimento de tal projeto, esclarecendo as tecnologias utilizadas e a forma com que foram aplicados os conteúdos durante o desenvolvimento do trabalho prático.

Tecnologias Utilizadas

- Linguagem: Java
- Bibliotecas: “java.time”, “java.util.List”, “java.util.ArrayList”, tratamento de exceções customizadas

Diagrama de Classes UML

A representação gráfica do projeto está disponível na pasta “docs” no repositório do GitHub.

Link para acesso do PDF:

https://drive.google.com/file/d/1qM-kZi8AKI6eSJalx6O_N7FQKz7QkGTL/view?usp=sharing

Pilares de OO Aplicados

1. Encapsulamento

Com o fim de proteger o sistema e garantir acesso controlado a determinados dados, foi utilizado o encapsulamento diversas vezes durante do desenvolvimento do código.

Aplicações no Sistema:

I)

```
public class Usuario {
private String senha; // Proteção de atributo privado
public String getSenha() { return Senha } // Getter controlado
}
```

// Na aplicação I, quando são utilizados dados privados, o encapsulamento tem como fim proteger o acesso externo a esses dados, além do método getSenha() para permitir o acesso a este dado somente por meio de um método controlado.

II)

```
public class Veiculo {
private String placa; // Proteção de um atributo privado
public void setPlaca(String placa) { this.placa = placa } // Setter controlado
}
```

// Na aplicação II, o encapsulamento permite modificar o atributo placa apenas por meio do devido método, com fim de evitar mudanças indevidas.

2. Herança

Para melhorar a organização do código, evitar redundâncias e duplicações no código, além de acrescentar para a reutilização do código e organização de hierarquias, foi aplicado o conceito da herança.

Aplicações no Sistema:

I)

```
public class Passageiro extends Usuario { ... }

public class Motorista extends Usuario { ... }
```

// Na aplicação, as classes Passageiro e Motorista herdam de Usuario, o que significa que atributos em comum, como nome, cpf e senha, são definidos nas superclasse e reutilizados pelas demais.

II)

```
public class Comum extends Categoria {

public Comum(double tarifaBase, double custoPorKm) {

super(5.00, 1.00);

}
```

}

// Na aplicação, a herança permite que Comum e Luxo herdem de Categoria, além de que, para otimização do código, é possível acessar atributos exclusivos de Categoria, como a tarifa base e o custo por quilometro.

3. Polimorfismo

Para tornar o código mais flexível, foi utilizado o polimorfismo, que possibilita que um mesmo método consiga agir de diferentes maneiras dependendo do objeto que o invoca.

Aplicações no Sistema:

I)

```
public class Categoria {
public double calcularPreco(double km) {
return tarifaBase + (custoPorKm * km);
}
}
```

// A superclasse Categoria define o método calcularPreco, com a assinatura double, que define a fórmula para o cálculo do preço final, porém deixa a cargo da subclasses, Comum e Luxo, os valores da tarifa base e do custo por quilometro.

II)

```
public interface MetodoPagamento {
void processarPagamento(double valor);

}
```

// Na aplicação, a interface MetodoPagamento define o método processarPagamento, e as subclasses PagamentoPix, PagamentoCartao, PagamentoDinheiro, que implementam esse método tornam-se todos subtipos válidos. Além de que cada uma pode fornecer o seu próprio corpo para o método.

4. Associações

As associações definem responsabilidades, dependências e multiplicidades entre as classes, descrevendo relações estruturais e sendo essenciais para o funcionamento da aplicação.

Aplicações no Sistema:

I) Associação Passageiro ↔ Corrida

O passageiro está associado a diversas corridas realizadas ao longo do tempo, enquanto a corrida para existir depende de um passageiro, pois não há a corrida sem alguém que a solicite.

II) Associação Motorista ↔ Veículo

Essa associação é sólida, devido ao seu caráter de exclusividade, já que cada motorista opera exclusivamente um veículo, e o veículo só existe se registrado pelo usuário.

III) Associação Corrida → Categoria

Esta associação, diferente das anteriores, é unidirecional, já que a corrida está vincula a uma categoria para o cálculo do seu preço.

5. Exceções customizadas

As exceções customizadas são essenciais para a expressividade e comunicação do sistema com seu usuário, permitindo uma tratativa mais direta e precisa de erros específicos, melhorando sua rastreabilidade.

Aplicações no Sistema:

I) EstadoInvalidoDaCorridaException

Assegura a integridade das transições dos ciclos de vida das corrida, impedindo operações que violem a lógica interna do sistema evitando comportamentos imprevisíveis.

II) MotoristaValidoException

Indica que o motorista não cumpre requisitos necessário para operar, como habilitação válida. Essa distinção entre os erros ajuda no cumprimento das normas do sistema.

III) NenhumMotoristaDisponivelException

Expressa que nenhum motorista disponível atende aos requisitos para iniciar a corrida, podendo oferecer ajustes aos sistemas de roteamento ou contribuir com alternativas para o passageiro.

IV) PagamentoRecusadoException

Representa que não foi possível concluir a transação financeira referente à corrida, por motivos técnicos ou de segurança. A distinção dessa exceção permite ao sistema solicitar outro método de pagamento ao usuário ou indicar o erro ocorrido.

V) PassageiroPendenteException

Indica que o passageiro se encontra em situação irregular, e que portanto não foi possível iniciar a corrida, o que permite ao sistema assegurar os devidos processos, impedindo que o programa avance em operações irregulares.

VI) SaldoInsuficienteException

Representa a insuficiência de recursos financeiros para a conclusão da transação, o que permite clareza no tratamento da situação com o passageiro.