Universidade de Brasília

DOCUMENTAÇÃO - 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

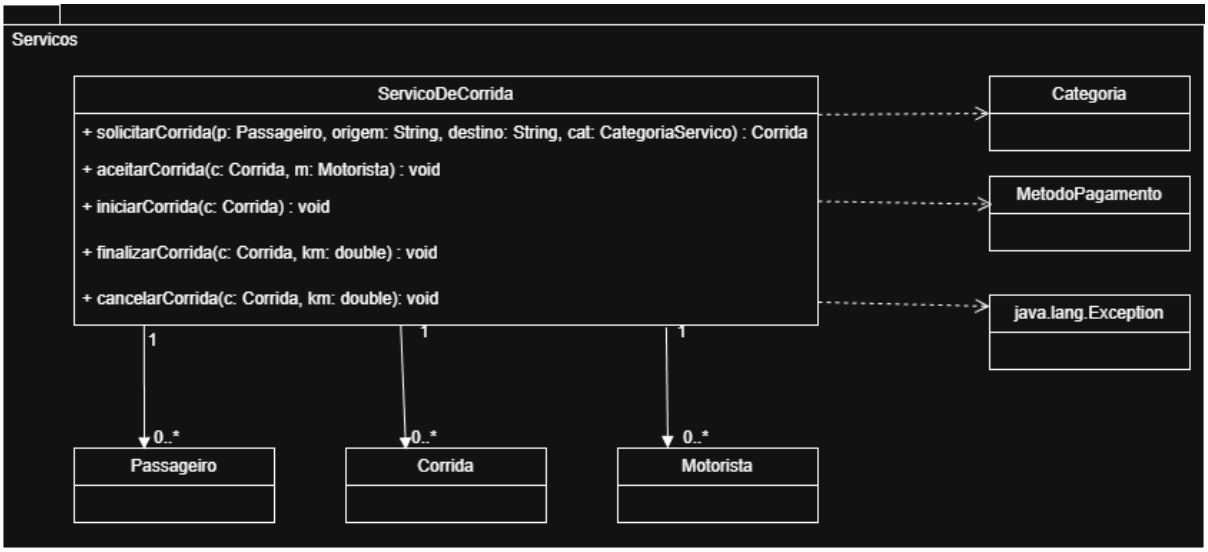
Objetivos

Desenvolver um sistema em Java que aplique todos os conceitos trabalhados nas aulas da disciplina de Orientação a Objetos, garantido que modularidade, encapsulamento, herança, polimorfismo e exceções personalizadas sejam explicitamente considerados e aplicados durante a realização do trabalho.

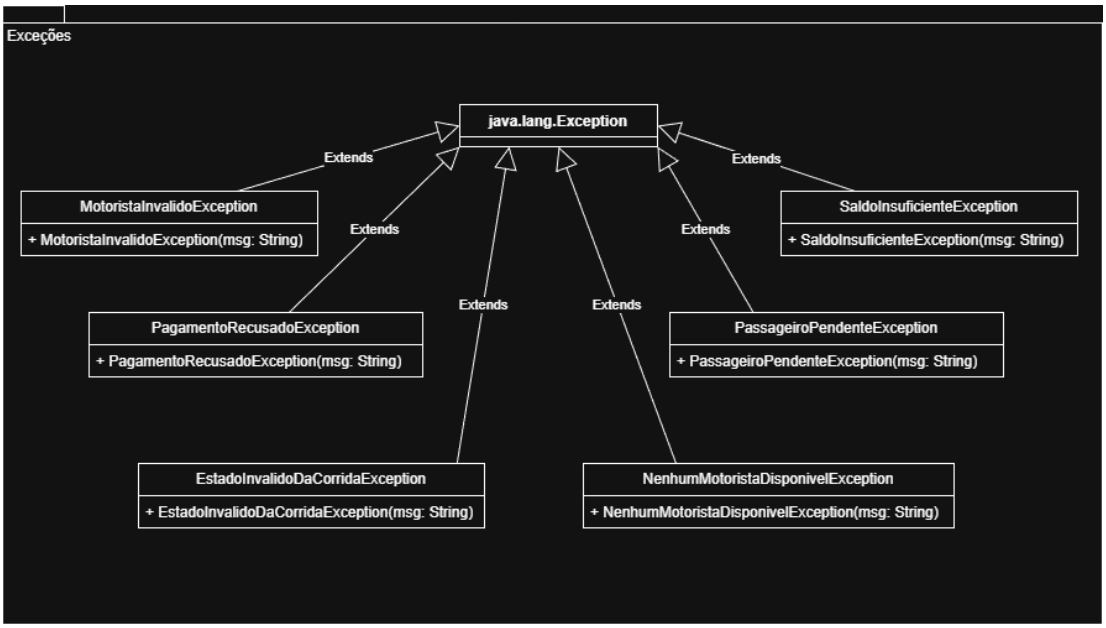
Diagrama UML

O diagrama está disponibilizado na pasta “docs”, no repositório do GitHub nos formatos de PDF e png.

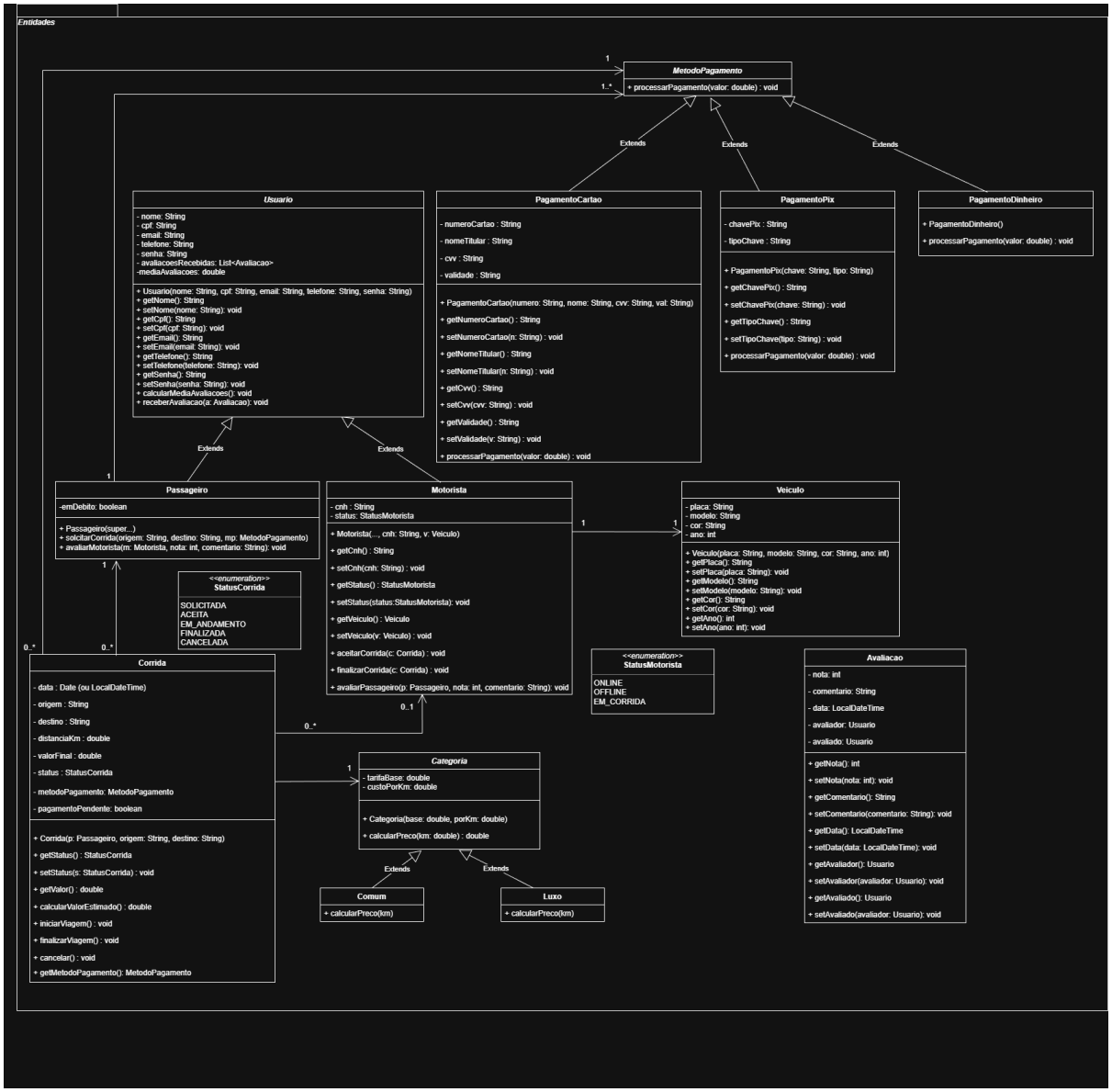
Serviços



Exceções



Entidades



Explicação das Associações, Heranças e Polimorfismos

Associações

As associações são os relacionamentos entre os objetos que define a forma com que eles trabalham entre si, delimitando as responsabilidades e dependências de cada um, em relações codependentes que só são possíveis caso ambas as classes estejam funcionando entre si. Por exemplo, a associação entre a classes “Corrida” com as classes “Passageiro” e “Motorista”, onde uma corrida só é possível se houver um passageiro e um motorista definidos para ela, mostram perfeitamente a necessidade de uma classe para o funcionamento de outra.

Heranças

A herança é um mecanismo que permite com que uma subclasse herde atributos de uma superclasse, o que promove a criação de hierarquias e reaproveitamento de artifícios, evitando redundâncias no código. No projeto de sistema de mobilidade urbana a herança foi usada especialmente com a superclasse “Usuario”, a qual são reutilizados os atributos de nome, CPF, e-mail, telefone e senha, nas subclasses “Passageiro” e “Motorista”, aprimorando a coesão do código, além da herança da superclasse “MetodosPagamento” com as subclasses, representando cada estratégia de pagamento, “PagamentoCartao”, “PagamentoPix” e “PagamentoDinheiro”, mantendo a interface unificada, e tratando cada método diferente como uma derivação do mesmo princípio.

Polimorfismo

O polimorfismo permite que diferentes classes respondam ao mesmo comando de maneiras distintas, mantendo uma interface comum, o que além de aumentar a flexibilidade do código, torna ele mais reutilizável. No trabalho prático, o polimorfismo foi utilizado diversas vezes, como, por exemplo, na classe “Categoria”, em que o cálculo do valor do pagamento é tratado de maneira uniforme pela aplicação, mesmo com cada categoria contendo seu método para o cálculo. Além disso, nos métodos de pagamento, a classe “Corrida” trata os métodos de pagamento da mesma forma, sem se preocupar com as diferenças e peculiaridades de cada um, permitindo, além de solidez, substituições e ampliações.

Exceções customizadas

As exceções personalizadas permitem precisão no tratamento de erros, podem esclarecer especificamente a origem do problema e das falhas para o usuário da aplicação. Por exemplo, exceções como “PagamentoRecusadoException” e “SaldoInsuficienteException” representam falhas relacionadas ao pagamento, e ao tratar cada uma de maneira específica, é possível, além de oferecer maior clareza com relação ao motivo do erro, maneiras de contorná-lo. Além disso, “NenhumMotoristaDisponivelException” e “PassageiroPendenteException” podem sinalizar que o passageiro ou motorista não estão em condições válidas para a corrida, portanto ela não poderá ser iniciada.