Faculdade de Ciências e Tecnologia em Engenharia - FCTE

Relatório do Trabalho de Inflação

Renan Curione de Castro - 242024834

Caio Bechepeche Mota - 242042340

Brasília - DF, maio de 2025

SUMÁRIO

1 INTRODUÇÃO

O objetivo deste projeto é modelar e desenvolver um protótipo de sistema para um aplicativo de compartilhamento de corridas, similar a plataformas como Uber, 99 ou Cabify. Este ecossistema digital conecta dois tipos principais de usuários: passageiros, que necessitam de transporte, e motoristas, que oferecem o serviço. Este trabalho tem como objetivo aplicar os conhecimentos adquiridos em sala de aula em um projeto integrando todo o conteúdo abordado de Orientação a Objetos na linguagem Java.

2 FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica deste projeto aborda os principais conceitos necessários para compreender o desenvolvimento de sistemas orientados a objetos, especialmente aqueles que seguem boas práticas de modelagem, organização e responsabilidade das classes. Esses elementos fornecem a base para a construção de aplicações modulares, reutilizáveis, fáceis de manter e capazes de representar corretamente entidades do mundo real, como usuários, motoristas, veículos e pagamentos.

2.1 Orientação a Objetos

A Orientação a Objetos (OO) é um paradigma de programação que organiza o software em torno de **objetos**, estruturas que combinam atributos (dados) e métodos (comportamentos). O objetivo principal desse paradigma é permitir que o sistema represente entidades reais de forma intuitiva, promovendo modularidade e reutilização de código. Os principais conceitos utilizados neste projeto incluem classes, objetos, encapsulamento, associação, herança e polimorfismo.

2.2 Padrões ou Conceitos Relevantes

Classe: é o molde ou modelo que define atributos e comportamentos comuns a um conjunto de objetos.

Objeto: é uma instância concreta de uma classe, contendo seus próprios valores para os atributos e podendo executar seus métodos.

Encapsulamento: princípio que protege o estado interno do objeto, expondo apenas o necessário por meio de métodos públicos. Isso reduz acoplamento e aumenta segurança.

Herança: permite que uma classe herde atributos e métodos de outra, promovendo reaproveitamento. No projeto, por exemplo, Passageiro e Motorista podem herdar da classe Pessoa.

Polimorfismo: possibilita que diferentes classes concretas compartilhem uma mesma interface ou método, mas com comportamentos distintos. No sistema, isso ocorre nos métodos de pagamento: Dinheiro, PIX e CartaoDeCredito implementam diferentes formas de processar a mesma ação.

Associação, agregação e composição: definem como objetos se relacionam entre si. A classe Corrida, por exemplo, associa um Passageiro a um Motorista, e compõe objetos como preço, rota e método de pagamento.

Esses conceitos permitiram que as entidades do sistema fossem representadas de forma clara e com responsabilidade bem definida, facilitando a manutenção e a evolução do código.

3 MODELAGEM DO SISTEMA

Nesta seção haverá a abordagem de modelagem do sistema, seus requisitos prévios e como foi desenvolvido e estruturado a partir disso.

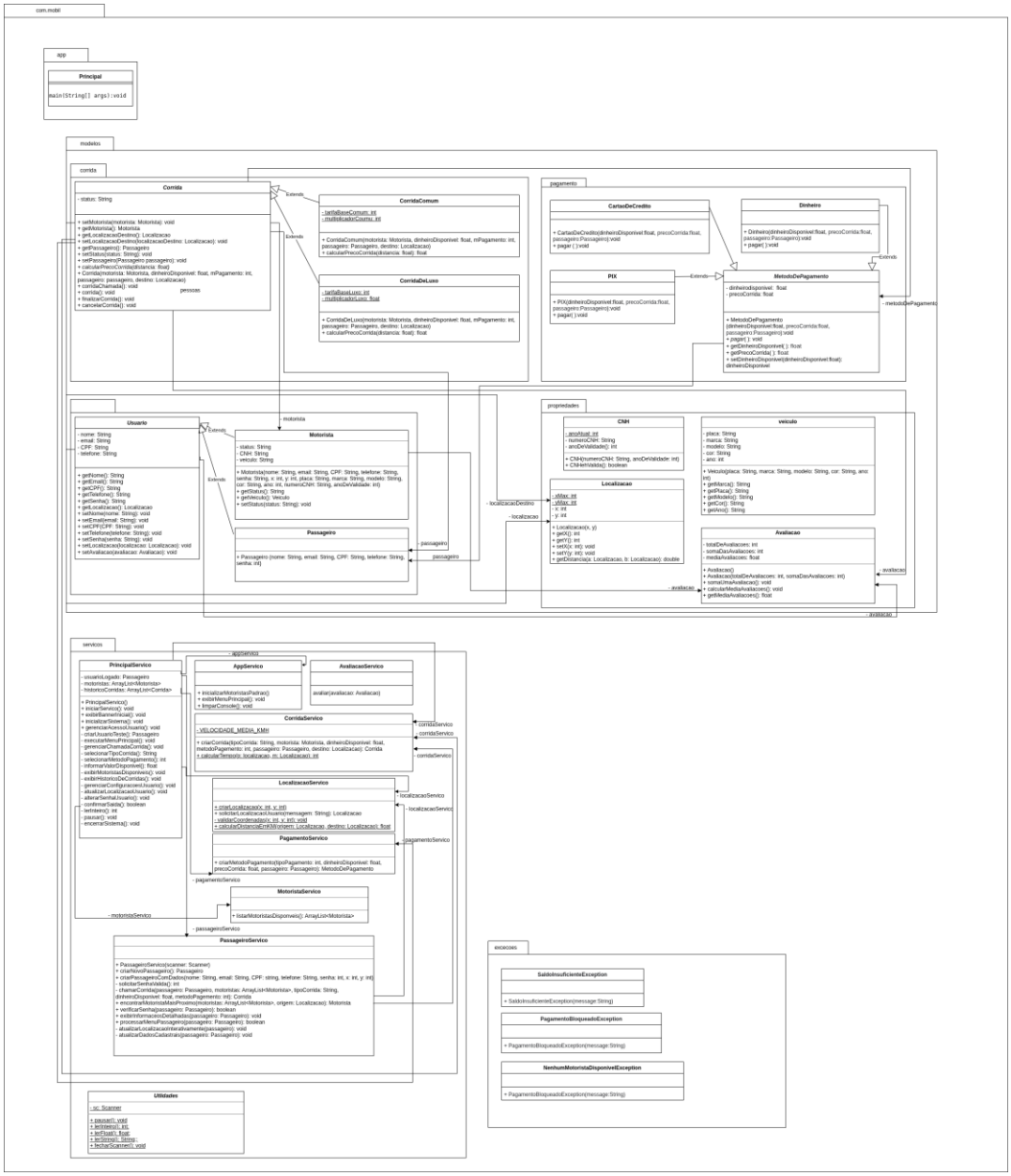
3.1 Requisitos

É necessário que o usuário se cadastre com nome, CPF, telefone, e-mail, telefone, crie uma senha de segurança e informe sua localização e a localização de destino, além de solicitar um tipo de corrida e informar o método de pagamento. Além disso é fundamental o cálculo do preço, a localização do motorista mais próximo para aceitar a corrida, informar a distância e o tempo estimado da chegada do motorista e o fim da corrida e por fim, a avaliação da corrida pelo passageiro e a consulta do histórico de corridas.

3.2 Diagrama de Classes

Link:https://app.diagrams.net/#G1QnHeagIPkerNJxlvZ0H9aUqagkX0lHmk#%7B%22pageId%22%3A%22ApVEAWr0NS10NoJyacWi%22%7D

Foto do diagrama UML:



4 ARQUITETURA E IMPLEMENTAÇÃO

Nesta seção, vamos entrar mais a fundo sobre a estrutura e fluxo do nosso projeto, Mobil. Também explicaremos todas as classes e o objetivo da implementação cada uma delas.

4.1 Estrutura do Projeto

Exemplo de estrutura:

Pacote: Definição

- Classe: Explicação

Mobil App:

com.mobil.app

- Principal: Onde o App é inicializado

com.mobil.modelos.pessoas - Entidades de usuários

- Usuario (classe abstrata): Define atributos comuns
- Passageiro: Representa o passageiro que solicita corridas
- Motorista: Representa o motorista que oferece o serviço

com.mobil.modelos.corrida - Sistema de corridas

- Corrida (classe abstrata): Define o comportamento geral
- CorridaComum: Implementa corrida econômica
- CorridaDeLuxo: Implementa corrida premium

com.mobil.modelos.pagamento - Sistema de pagamentos

- MetodoDePagamento (classe abstrata): Define interface comum
- Dinheiro, PIX, CartaoDeCredito: Implementações concretas

com.mobil.modelos.propriedades - Classes que são de algum Usuario

- Localizacao: Gerencia coordenadas e cálculos de distância
- Veiculo: Informações do veículo do motorista
- CNH: Validação da carteira de motorista
- Avaliacao: Sistema de avaliação por estrelas

com.mobil.modelos.servicos - Lógica de negócio

- PrincipalServico: Controlador principal do aplicativo
- CorridaServico: Gerenciamento de corridas

- PagamentoServico: Processamento de pagamentos
- PassageiroServico: Gerenciamento de passageiros
- MotoristaServico: Gerenciamento de motoristas
- LocalizacaoServico: Serviços de geolocalização
- AppServico: Configuração inicial e utilidades
- Utilidades: Alguns trechos de código úteis para uso durante o código

com.mobil.modelos.excecoes - Exceções específicas criadas

- NenhumMotoristaDisponivelException: 0 motoristas disponíveis.
- PagamentoBloqueadoException: errou a senha durante o pagamento.
- SaldoInsuficienteException: Usuário não tem dinheiro suficiente.

4.2 Fluxo de Funcionamento da Aplicação

1. Inicialização: O sistema carrega motoristas padrão e apresenta opções de acesso
2. Autenticação: Usuário pode criar uma conta ou usar o login de teste
3. Menu Principal: Apresenta 6 opções principais de interação
4. Solicitação de Corrida:
    a. Usuário seleciona tipo de corrida (Comum ou Luxo)
    b. Escolhe método de pagamento
    c. Informa valor disponível
    d. Sistema busca motorista mais próximo disponível
    e. Cria objeto de corrida com todas as informações
5. Execução da Corrida:
    a. Motorista se move em direção ao passageiro (simulação em grid)
    b. Passageiro entra no veículo
    c. Movimento em direção ao destino (simulação em grid)
    d. Finalização com processamento do pagamento
6. Pós-Corrida: Registro no histórico, disponibilização do motorista e volta do usuário para o Menu Principal

4.3 Trechos Relevantes de Código

Agora, vamos identificar onde encontramos os conceitos básicos de Orientação a Objeto no projeto. Muitos dos tópicos a seguir tem inúmeros exemplos no sistema, mas para efeito de melhor leitura, iremos citar apenas um exemplo para cada um.

Encapsulamento: a maioria dos atributos das classes são definidas com privadas, para evitar uso indevido delas (ninguém botar a mão sem poder).

Herança: Motorista e Passageiro, ambos são filhos de Usuário, herdando seus atributos e métodos.

Associação: Motorista se associa com Avaliacao, mesmo não tendo uma relação de herança.

Polimorfismo por Inclusão: ao finalizarCorrida(), a referência metodoPagamento é dita para pagar(), porém ela referencia uma filha de MetodoDePagamento, mas consegue pagar() porque esse método está presente de forma abstrata na mãe, e é implementada na filha, que é referenciada.

Polimorfismo por Sobrescrita: todos os filhos de MetodoDePagamento (Dinheiro, Pix e CartaoDeCredito) implementam uma forma diferente de pagar() a corrida.

Polimorfismo por Sobrecarga: existem diferentes métodos construtores de Passageiro, podendo cria-lo, um dando valor aos seus atributos com os valores dados no parâmetro, e um sendo interativo com o usuário.

Polimorfismo por Coerção: ao calcular o preço de uma corrida, as taxas base das corridas é um inteiro, enquanto o resultado pode ser float, então é usado o typecastin (float) na fórmula, para assim conseguir o resultado preciso e sem erro.

Polimorfismo Paramétrico: no historicoCorridas, o tipo do ArrayList é de Corrida, podendo armazenar tanto CorridaComum e CorridaDeLuxo.

5 CONCLUSÃO

Por fim, nesta seção iremos falar sobre o que aprendemos com este trabalho prático e todos os desafios que enfrentamos durante essa atividade. Para uma

síntese mais clara das nossas ideias, e por nosso grupo ser, na verdade, uma dupla, iremos ceder relatos individuais. Com isso, podemos nos expressar livremente sobre o que sentimos.

Relato pessoal do Caio: Esse projeto me ajudou bastante na integração e criação de classes, objetos e métodos, algo que antes eu ficava na dúvida sobre como aplicar tal método em tal classe agora ficou mais claro, além de mostrar como relacionar os elementos do projeto é algo mais complexo do que parece, exigindo uma lógica sobre Orientação a Objetos mais profunda dos quais os exemplos mais simples que (devido ao tempo) a aula proporciona. Além disso, planejar a UML e modificá-la de acordo com o desenvolvimento me mostrou que projetos grandes como esse passam por muitas alterações e revisões, sendo necessário tomar um cuidado maior para que o projeto não desande.

Relato pessoal do Renan: eu sinto que, apesar de nosso trabalho não ter ficado perfeito, aprendemos muito sobre como planejar um projeto e traduzi-lo em um código que consegue ser aprimorado ou modificado com menos esforço do que seria se não tivéssemos modularizado ele devidamente. Alguns conceitos que acabaram passando batido nas aulas, vieram à tona no desenvolvimento. Além disso, ampliamos nossa experiência com o GitHub, que eu, particularmente, nunca tive trabalhado em um projeto em conjunto, e com desenho de diagramas UML, que tanto serve para a idealização, quanto para a manutenção do projeto, ao facilitar a visualização de todo o sistema. Enfim, minha experiência foi ótima nesse trabalho, sabemos que ainda temos muito a melhorar e a aprender sobre, mas fico grato com ter sido lecionado por um ótimo professor numa disciplina tão importante para Engenharia de Software como Orientação a Objetos.