---

1. Tipos Abstratos de Dados (ADT)

Para este projeto, foram utilizados vários ADTs dos quais: Stops(guardar stops) , Routes (armazenar rotas) , Graph (armazenar o "mapa").

Exemplos de ADTs no seu projeto:

- **Stops**: Representa uma paragem no mapa de transporte.
- **Routes**: Representa uma rota entre duas paragens.
- **Graph**: Estrutura de dados para armazenar o grafo que conecta as paragens.
- **Command**: Interface para os comandos que podem ser executados, como adicionar ou remover paragens ou rotas.

Descrição:

A classe Stops define os métodos essenciais e armazena as informações sobre cada paragem para obter informações sobre a paragem, como o código da paragem, nome, latitude e longitude.

A classe Routes define os métodos para obter informações sobre a rota, como paragem de início, fim, distância, duração e custo de cada meio de transporte.

---

2. Padrões de Software

Exemplos de padrões utilizados:

1. **Padrão Command**:
    - **Descrição**: O padrão Command permite encapsular uma solicitação como um objeto, o que permite parametrizar clientes com diferentes solicitações, fazer fila de solicitações, fazer log de solicitações e suportar operações de desfazer.
    - **Razão da escolha**: Este padrão foi utilizado para que as operações como adicionar, remover paragens e rotas, bem como ativar/desativar rotas, pudessem ser tratadas de maneira genérica, com a possibilidade de desfazer as ações (undo) e visualização do Histórico.
    - **Como foi mapeado**: O padrão Command foi implementado com a interface Command e classes concretas como AddStopCommand, RemoveStopCommand, etc e o seu respetivo CommandManager.

---

3. Refactoring

Ao contrário do uso de uma tabela para a identificação de **code smells**, irei descrevê-los de forma mais clara e perceptível para melhor compreensão.

Exemplo de "Code Smells" Detetados e Refatorações Possíveis:

1. **Code Smell: Métodos Longos**
    - **Situação detectada**: Métodos muito longos, como o código para adicionar uma nova paragem, onde havia uma grande quantidade de lógica em um único método.
    - **Técnica de refactoring**: **Extração de método** – O código foi refatorado de modo a extrair partes do método longo em métodos separados, tornando o código mais legível e reutilizável.
2. **Code Smell: Classe Grande (God Class)**
    - **Situação detectada**: A classe **TransportMap** estava com demasiadas responsabilidades, como a lógica de importação de dados e todas as operações envolventes.
    - **Técnica de refactoring**: **Extração de classe** – A lógica de importação de dados foi movida para uma nova classe dedicada à manipulação de dados, melhorando a coesão e responsabilidade das classes.
3. **Code Smell: Repetição de Código (Duplicate Code)**
    - **Situação detectada**: Diversas seções do código apresentavam lógica repetitiva, como verificações de validade de dados ou manipulação de exceções.
    - **Técnica de refactoring**: **Extração de método** e **Criação de uma classe utilitária** – O código repetido foi encapsulado em métodos ou classes auxiliares, diminuindo a duplicação e facilitando a manutenção.
4. **Code Smell: Nomes Inadequados (Inconsistent Naming)**
    - **Situação detectada**: Alguns nomes de métodos e variáveis não eram suficientemente descritivos, o que dificultava a compreensão do propósito do código.

- **Técnica de refactoring**: **Renomeação** – Alteramos os nomes das variáveis e métodos para refletirem melhor o seu propósito, facilitando a leitura e o entendimento do código.

5. **Code Smell: Método com Muitos Parâmetros (Long Parameter List)**
    - **Situação detectada**: Vários métodos aceitavam um número excessivo de parâmetros, tornando-os difíceis de invocar e entender.
    - **Técnica de refactoring**: **Objeto Paramétrico** – Agrupamos os parâmetros em objetos mais significativos e reutilizáveis, o que resultou em métodos mais claros e com uma interface mais simples.
6. **Code Smell: Código Morto (Dead Code)**
    - **Situação detectada**: Algumas variáveis ou métodos não eram utilizados, mas estavam presentes no código, criando confusão e aumentando o tamanho do código desnecessariamente.
    - **Técnica de refactoring**: **Remoção de código morto** – O código que não tinha utilidade foi completamente removido, o que resultou em um código mais limpo e fácil de manter.

Conclusão:

Para concluir o projeto mostrou-se viável na aplicação de conceitos de Engenharia de Software, onde a utilização de **Tipos Abstratos de Dados (ADT)** foi essencial para organizar e estruturar as informações de forma clara e eficiente. A implementação das classes **Stops**, **Routes** e **Graph** permitiu representar as paragens, as rotas e o grafo do mapa de transporte de forma modular, facilitando o manuseio e a consulta dos dados.

O uso de padrões de design, como o **Command**, foi particularmente importante, pois permitiu a implementação de uma estrutura flexível e extensível para realizar ações como adicionar e remover paragens e rotas, além de possibilitar o **undo** e a visualização do histórico de comandos executados. Este padrão contribuiu para a generalização das operações e a separação de responsabilidades entre as classes, o que resulta em um código mais limpo e fácil de manter.

Durante o desenvolvimento, também foram identificados **code smells** que comprometeriam a qualidade e a manutenção do código a longo prazo. A aplicação das técnicas de **refactoring**, como a extração de métodos e classes, foi

fundamental para melhorar a legibilidade e coesão do código, tornando-o mais fácil de entender.

Em resumo, o projeto não só cumpriu os requisitos iniciais, mas também seguiu boas práticas de design e refatoração, resultando em uma solução eficiente, organizada e fácil de manter. A utilização de **ADTs** e padrões de design não só melhorou a estrutura do sistema, como também proporcionou uma base sólida para a implementação de novas funcionalidades e futuras modificações.