**README**

For this lab, I opted to develop a starter code capable of aggregating taxi ride data from NYC, with the program designed to process CSV file input formatted as follows.

A screenshot of a computer

Description automatically generated

**Exercise 1**

**Single Responsibility Principle**

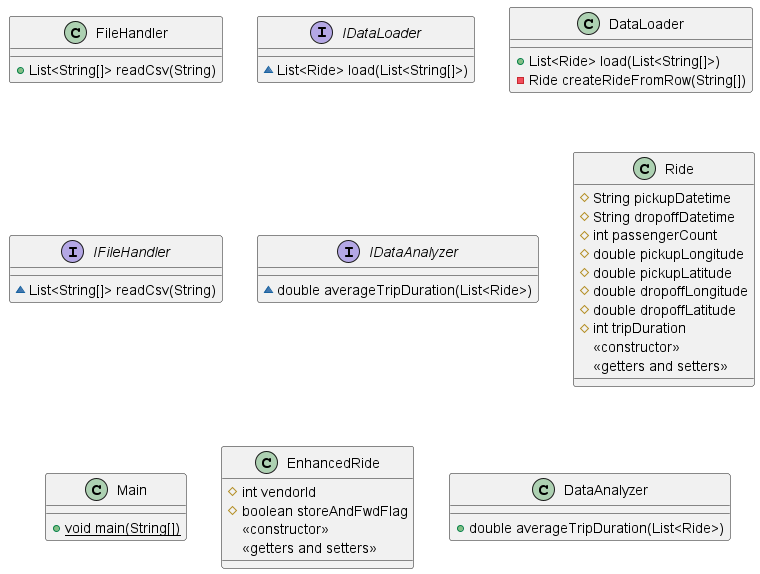
The Single Responsibility Principle (SRP) mandates that a class should have just one reason to change, equating to a singular responsibility.

Ride and PremiumRide Classes: These solely encapsulate ride data, steering clear of responsibilities like file reading or data analysis.

FileHandler Class: This class exclusively manages file-related tasks, particularly reading from a CSV file.

DataLoader Class: Its purpose is to convert raw data into Ride objects.

DataAnalyzer Class: This class takes on the role of analyzing loaded data, exemplified by its capability to calculate the average trip duration.

By assigning each class a distinct and singular responsibility, we ensure adherence to the SRP.

**Open/Close Principle**

The Open/Closed Principle (OCP) prescribes that classes should be open for extension yet closed for modification, allowing for behavior expansion without altering existing source code.

Ride and PremiumRide Classes: Demonstrating this principle, PremiumRide extends Ride, illustrating that Ride is extensible (enabling the addition of more ride types in the future) without necessitating modifications (we needn’t alter the Ride class to introduce PremiumRide).

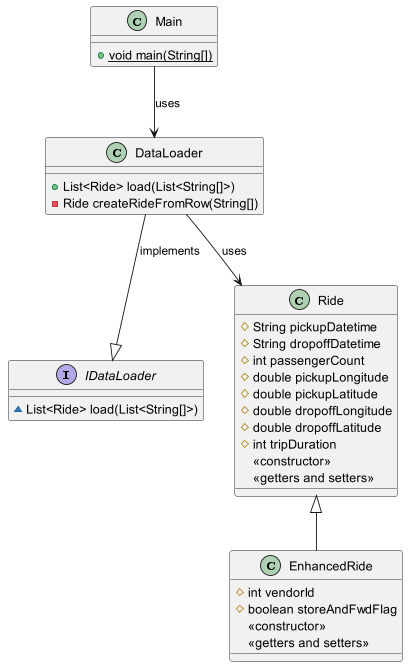
A screenshot of a computer

Description automatically generated

**Liskov Substitution Principle**

The Liskov Substitution Principle (LSP) ensures that objects of a superclass should be seamlessly replaceable with objects of a subclass, maintaining the program's correctness.

Ride and PremiumRide Classes: In adherence to LSP, a PremiumRide object, being a subclass of Ride, should function interchangeably with a Ride object without causing any disruptions. Essentially, any method or function designed to accept a Ride object should operate correctly when provided a PremiumRide object, as the latter is simply a specialized version of the former.



To demonstrate ISP I used specific interfaces rather than one general-purpose, interface. The three distinct interfaces are IFileHandler, IDataLoader, and IDataAnalyzer.

- IFileHandler is solely concerned with file handling operations, specifically reading from a CSV.

- IDataLoader is focused on loading data into our Ride objects.

- IDataAnalyzer is used for analyzing the data.

Each of these interfaces is tailored to a specific set of related operations. If a class needs to analyze data, it doesn't also have to be burdened with file reading methods or data loading methods.

A diagram of a computer

Description automatically generated

To demonstrate DIP I created a Main class, is seen as a higher-level module. The class doesn't directly instantiate or depend on the specific workings of FileHandler, DataLoader, or DataAnalyzer. Instead, it interacts with these through their respective interfaces (IFileHandler, IDataLoader, IDataAnalyzer).

For example, instead of directly creating an instance of FileHandler, we do IFileHandler fileHandler = new FileHandler();

This means our Main class is set up to work with any class that implements IFileHandler. If we later decide to have a DatabaseHandler that fetches data from a database instead of a file, our Main class can work with it with minimal to no changes, as long as DatabaseHandler implements IFileHandler.

The DIP ensures that our main logic remains decoupled from the specific implementations of its dependencies, allowing for greater flexibility and easier changes in the future.

