**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

The following Java files are used:

**Single Responsibility Principle (SRP)**

SRP mandates that a class should have just one reason to change, equating to a singular responsibility.

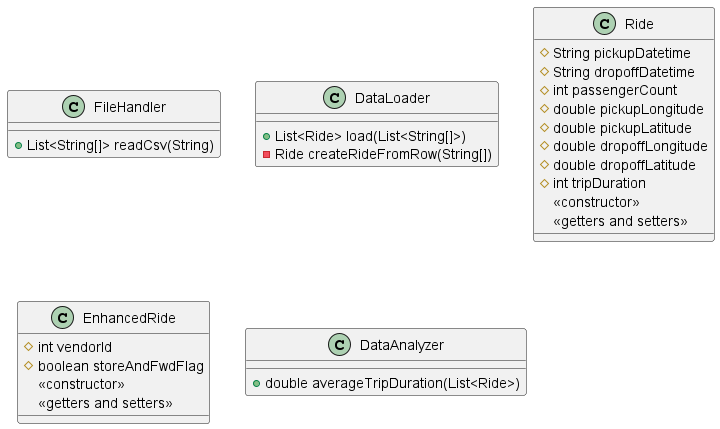
**Ride** and **EnhancedRide** Classes: These solely encapsulate ride data, steering clear of responsibilities like file reading or data analysis.

**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.



/\*\*  
 \* The Ride class represents a ride in a vehicle sharing service.  
 \* It includes details such as pickup and dropoff times and locations,  
 \* the number of passengers, and the trip duration.  
 \*/  
  
public class Ride {  
 protected String pickupDatetime;  
 protected String dropoffDatetime;  
 protected int passengerCount;  
 protected double pickupLongitude;  
 protected double pickupLatitude;  
 protected double dropoffLongitude;  
 protected double dropoffLatitude;  
 protected int tripDuration;  
  
 /\*\*  
 \* Constructs a new {@code Ride} instance.  
 \*  
 \* @param pickupDatetime the datetime when the ride was picked up  
 \* @param dropoffDatetime the datetime when the ride was dropped off  
 \* @param passengerCount the number of passengers in the ride  
 \* @param pickupLongitude the longitude of the pickup location  
 \* @param pickupLatitude the latitude of the pickup location  
 \* @param dropoffLongitude the longitude of the dropoff location  
 \* @param dropoffLatitude the latitude of the dropoff location  
 \* @param tripDuration the duration of the trip in minutes  
 \*/  
 public Ride(String pickupDatetime, String dropoffDatetime, int passengerCount,  
 double pickupLongitude, double pickupLatitude, double dropoffLongitude,  
 double dropoffLatitude, int tripDuration) {  
 this.pickupDatetime = pickupDatetime;  
 this.dropoffDatetime = dropoffDatetime;  
 this.passengerCount = passengerCount;  
 this.pickupLongitude = pickupLongitude;  
 this.pickupLatitude = pickupLatitude;  
 this.dropoffLongitude = dropoffLongitude;  
 this.dropoffLatitude = dropoffLatitude;  
 this.tripDuration = tripDuration;  
 }  
}

public class EnhancedRide extends Ride {  
 protected int vendorId;  
 protected int storeAndFwdFlag; // how data is sent, 0 for No, 1 for Yes  
  
 /\*\*  
 \* Constructs a new {@code EnhancedRide} instance.  
 \*  
 \* @param vendorId the ID of the vendor  
 \* @param storeAndFwdFlag the store and forward flag  
 \*/  
 public EnhancedRide(String pickupDatetime, String dropoffDatetime, int passengerCount, double pickupLongitude, double pickupLatitude, double dropoffLongitude, double dropoffLatitude, int tripDuration, int vendorId, int storeAndFwdFlag) {  
 super(pickupDatetime, dropoffDatetime, passengerCount, pickupLongitude, pickupLatitude, dropoffLongitude, dropoffLatitude, tripDuration);  
 this.vendorId = vendorId;  
 this.storeAndFwdFlag = storeAndFwdFlag;  
 }  
  
 /\*\*  
 \* Returns the vendor ID.  
 \*  
 \* @return the vendor ID  
 \*/  
 public int getVendorId() {  
 return vendorId;  
 }  
  
 /\*\*  
 \* Sets the vendor ID.  
 \*  
 \* @param vendorId the vendor ID to set  
 \*/  
 public void setVendorId(int vendorId) {  
 this.vendorId = vendorId;  
 }  
  
 /\*\*  
 \* Returns the store and forward flag.  
 \*  
 \* @return the store and forward flag  
 \*/  
 public int isStoreAndFwdFlag() {  
 return storeAndFwdFlag;  
 }  
  
 /\*\*  
 \* Sets the store and forward flag.  
 \*  
 \* @param storeAndFwdFlag the store and forward flag to set  
 \*/  
 public void setStoreAndFwdFlag(int storeAndFwdFlag) {  
 this.storeAndFwdFlag = storeAndFwdFlag;  
 }  
}

import java.util.List;  
  
/\*\*  
 \* DataAnalyzer implements IDataAnalyzer to provide analysis of ride data.  
 \*/  
public class DataAnalyzer implements IDataAnalyzer {  
 /\*\*  
 \* Calculates the average trip duration from a list of rides.  
 \*  
 \* @param rides a list of Ride objects  
 \* @return the average trip duration as a double. If there are no rides, returns 0.0  
 \*/  
 public double averageTripDuration(List<Ride> rides) {  
 if (rides.isEmpty()) return 0.0;  
  
 double totalDuration = 0.0;  
 for (Ride ride : rides) {  
 totalDuration += ride.getTripDuration();  
 }  
 return totalDuration / rides.size();  
 }  
}

import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
import java.util.Arrays;  
  
/\*\*  
 \* DataLoader is an implementation of IDataLoader for loading ride data.  
 \*/  
public class DataLoader implements IDataLoader {  
 /\*\*  
 \* Converts a list of raw string arrays representing ride data into a list of Ride objects.  
 \*  
 \* @param rawData List of string arrays, each representing a row of ride data.  
 \* @return List of Ride objects created from the raw data.  
 \*/  
 public List<Ride> load(List<String[]> rawData) {  
 // Skip header row and process the data  
 return rawData.stream()  
 .skip(1)  
 .map(this::createRideFromRow)  
 .collect(Collectors.toList());  
 }  
  
 /\*\*  
 \* Creates a Ride object from a string array of ride data.  
 \*  
 \* @param row A string array representing a single row of ride data.  
 \* @return A Ride object, or null if the data could not be parsed.  
 \*/  
 private Ride createRideFromRow(String[] row) {  
 try {  
 String pickupDateTime = row[0];  
 String dropoffDateTime = row[1];  
 int passengerCount = Integer.parseInt(row[2]);  
 double pickupLongitude = Double.parseDouble(row[3]);  
 double pickupLatitude = Double.parseDouble(row[4]);  
 double dropoffLongitude = Double.parseDouble(row[5]);  
 double dropoffLatitude = Double.parseDouble(row[6]);  
 int tripDuration = Integer.parseInt(row[7]);  
  
 return new Ride(pickupDateTime, dropoffDateTime, passengerCount, pickupLongitude, pickupLatitude, dropoffLongitude, dropoffLatitude, tripDuration);  
 } catch (NumberFormatException e) {  
 System.err.println("Error parsing ride data: " + Arrays.toString(row));  
 return null;  
 }  
 }  
}

import java.util.List;  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.ArrayList;  
  
/\*\*  
 \* FileHandler is an implementation of the IFileHandler interface, providing methods to handle file operations.  
 \*/  
public class FileHandler implements IFileHandler {  
 /\*\*  
 \* Reads a CSV file and returns its content as a list of string arrays.  
 \* Each line in the CSV file is split by commas and represented as an array of strings.  
 \*  
 \* @param filePath the path to the CSV file  
 \* @return a list of string arrays, each representing a line in the CSV file  
 \* @throws IOException if an I/O error occurs while reading the file  
 \*/  
 public List<String[]> readCsv(String filePath) throws IOException {  
 List<String[]> data = new ArrayList<>();  
 try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {  
 String line;  
 while ((line = br.readLine()) != null) {  
 data.add(line.split(","));  
 }  
 }  
 return data;  
 }  
}

**Open/Close Principle (OCP)**

OCP prescribes that classes should be open for extension yet closed for modification, allowing for behavior expansion without altering existing source code.

**Ride** and **EnhancedRide** Classes: Demonstrating this principle, **EnhancedRide** extends **Ride**, illustrating that Ride is extensible (enabling the addition of more ride types in the future) without necessitating modifications (we do not have to alter the **Ride** class to introduce **EnhancedRide**).

A screenshot of a computer

Description automatically generated

… same code as above (SRP)  
  
 /\*\*  
 \* Returns the vendor ID.  
 \*  
 \* @return the vendor ID  
 \*/  
 public int getVendorId() {  
 return vendorId;  
 }  
  
 /\*\*  
 \* Sets the vendor ID.  
 \*  
 \* @param vendorId the vendor ID to set  
 \*/  
 public void setVendorId(int vendorId) {  
 this.vendorId = vendorId;  
 }  
  
 /\*\*  
 \* Returns the store and forward flag.  
 \*  
 \* @return the store and forward flag  
 \*/  
 public int isStoreAndFwdFlag() {  
 return storeAndFwdFlag;  
 }  
  
 /\*\*  
 \* Sets the store and forward flag.  
 \*  
 \* @param storeAndFwdFlag the store and forward flag to set  
 \*/  
 public void setStoreAndFwdFlag(int storeAndFwdFlag) {  
 this.storeAndFwdFlag = storeAndFwdFlag;  
 }

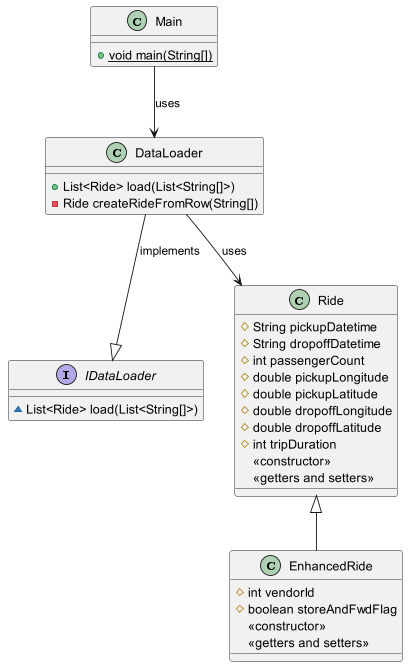
Create an EnhancedRide object as follows.

EnhancedRide enhancedRide = new EnhancedRide("2016-03-30 4:14:00 PM", "2016-03-30 5:01:00 PM", 1, -73.7898407, 40.6435585, -73.97066498, 40.68727875, 2824, 1, 1);

**Liskov Substitution Principle (LSP)**

LSP ensures that objects of a superclass should be seamlessly replaceable with objects of a subclass, maintaining the program's correctness.

Ride and **EnhancedRide** Classes: In adherence to LSP, a **EnhancedRide** 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 **EnhancedRide** object, as the latter is simply a specialized version of the former.



import java.io.IOException;  
import java.util.List;  
  
/\*\*  
 \* Main entry point for the ride data analysis application.  
 \* It utilizes various components to load, analyze, and display ride data.  
 \*/  
public class Main {  
 public static void main(String[] args) throws IOException {  
 // Dependencies are now inverted and depend on abstractions  
 IFileHandler fileHandler = new FileHandler();  
 IDataLoader dataLoader = new DataLoader();  
  
 // Read raw ride data from a CSV file  
 List<String[]> rawData = fileHandler.readCsv("train\_99.csv");  
  
 // Load the raw data into Ride objects  
 List<Ride> rides = dataLoader.load(rawData);  
  
 // adding EnhancedRide  
 Ride enhancedRide = new EnhancedRide("2016-03-30 4:14:00 PM", "2016-03-30 5:01:00 PM", 1, -73.7898407, 40.6435585, -73.97066498, 40.68727875, 2824, 1, 1);  
 rides.add(enhancedRide); // notice how we can use EnhancedRide to create a Ride object.  
 }  
}

**Interface Segregation Principle (ISP)**

To illustrate ISP, I implemented specific interfaces rather than a single, general-purpose interface, resulting in **IFileHandler**, **IDataLoader**, and **IDataAnalyzer**.

* **IFileHandler** is dedicated to file handling operations, notably reading from a CSV file.
* **IDataLoader** focuses on populating Ride objects with data.
* **IDataAnalyzer** is tasked with data analysis.

Each interface caters to a particular set of related operations, ensuring that a class responsible for data analysis, for example, is not unnecessarily encumbered with file reading or data loading methods.

A diagram of a computer

Description automatically generated

import java.util.List;  
  
/\*\*  
 \* IDataAnalyzer provides a set of methods for analyzing ride data.  
 \*/  
public interface IDataAnalyzer {  
 /\*\*  
 \* Calculates the average trip duration from a list of rides.  
 \*  
 \* @param rides a list of Ride objects  
 \* @return the average trip duration as a double  
 \*/  
 double averageTripDuration(List<Ride> rides);  
}

import java.util.List;  
  
/\*\*  
 \* The IDataLoader interface defines a contract for loading ride data.  
 \*/  
public interface IDataLoader {  
 /\*\*  
 \* Loads ride data from a list of string arrays.  
 \*  
 \* @param rawData a list of string arrays, each representing raw ride data  
 \* @return a list of Ride objects constructed from the raw data  
 \*/  
 List<Ride> load(List<String[]> rawData);  
}

import java.util.List;  
import java.io.IOException;  
  
/\*\*  
 \* Interface provides a method to read data from a CSV file.  
 \*/  
public interface IFileHandler {  
 /\*\*  
 \* Reads a CSV file and returns its content as a list of string arrays.  
 \* Each line in the CSV file is represented as an array of strings, where each string is a field.  
 \*  
 \* @param filePath the path to the CSV file  
 \* @return a list of string arrays, each representing a line in the CSV file  
 \* @throws IOException if an I/O error occurs while reading the file  
 \*/  
 List<String[]> readCsv(String filePath) throws IOException;  
}

**Dependency Inversion Principle**

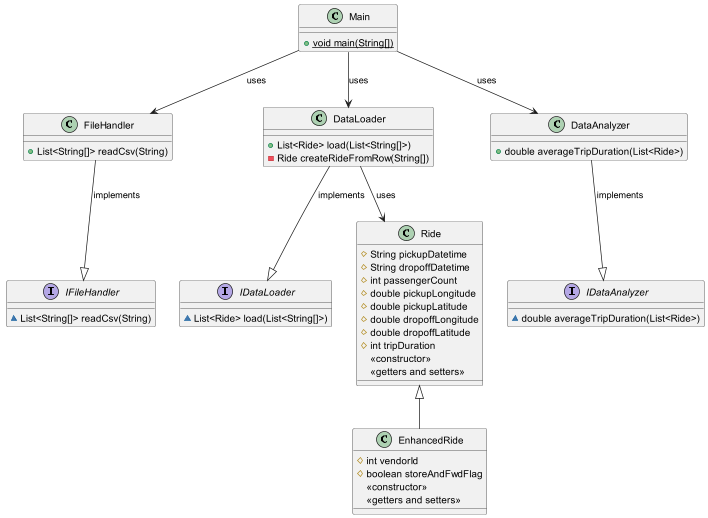
To showcase the DIP, I designed a **Main** class as a high-level module, ensuring it does not directly instantiate or depend on the specific operations of **FileHandler**, **DataLoader**, or **DataAnalyzer**. Instead, it interacts with these components through their respective interfaces: **IFileHandler**, **IDataLoader**, and **IDataAnalyzer**.

For instance, rather than directly creating a FileHandler instance, we utilize the interface, as in:

*IFileHandler fileHandler = new FileHandler();*

This approach enables our Main class to work with any class implementing **IFileHandler**, providing flexibility. If we decide to introduce a **DatabaseHandler** that retrieves data from a database, the Main class can seamlessly integrate with it, requiring minimal to no modifications, provided that **DatabaseHandler** implements **IFileHandler**.

Through DIP, our main logic remains detached from the specific implementations of its dependencies, fostering a robust foundation for future modifications and enhancements.



import java.io.IOException;  
import java.util.List;  
  
/\*\*  
 \* Main entry point for the ride data analysis application.  
 \* It utilizes various components to load, analyze, and display ride data.  
 \*/  
public class Main {  
 /\*\*  
 \* The main method that drives the application.  
 \* It creates instances of FileHandler, DataLoader, and DataAnalyzer to process ride data.  
 \* Ride data is read from a CSV file, loaded into Ride objects, and then analyzed.  
 \* The results of the analysis are displayed to the console.  
 \*  
 \* @param args command line arguments (not used)  
 \* @throws IOException if there is an issue reading the CSV file  
 \*/  
 public static void main(String[] args) throws IOException {  
 // Dependencies are now inverted and depend on abstractions  
 IFileHandler fileHandler = new FileHandler();  
 IDataLoader dataLoader = new DataLoader();  
 IDataAnalyzer dataAnalyzer = new DataAnalyzer();  
  
 // Read raw ride data from a CSV file  
 List<String[]> rawData = fileHandler.readCsv("train\_99.csv");  
  
 // Load the raw data into Ride objects  
 List<Ride> rides = dataLoader.load(rawData);  
  
 // adding EnhancedRide  
 Ride enhancedRide = new EnhancedRide("2016-03-30 4:14:00 PM", "2016-03-30 5:01:00 PM", 1, -73.7898407, 40.6435585, -73.97066498, 40.68727875, 2824, 1, 1);  
 rides.add(enhancedRide);  
  
 // Analyze the ride data and display the average trip duration  
 double averageDuration = dataAnalyzer.averageTripDuration(rides);  
 System.out.println("Average Trip Duration: " + averageDuration);  
 }  
}