Developing a Class-Based Ride Sharing System

Samrat Baral

University of the Cumberlands

2025 Spring - Advanced Programming Languages (MSCS-632-B01) - Second Bi-term

April 14, 2025

This repository contains a multi-language implementation of a Ride Sharing System that demonstrates core Object-Oriented Programming (OOP) principles: encapsulation, inheritance, and polymorphism. The project includes implementations in **C++** and **Smalltalk**.

Project Overview

The Ride Sharing System includes the following components:

1. **Ride Class:**
    - A base class that holds core details such as `rideID`, `pickupLocation`, `dropoffLocation`, `distance`, and a method `fare()` to calculate ride fare.
    - A `rideDetails()` method displays ride information including the computed fare.
2. **Specific Ride Subclasses:**
    - **StandardRide:** Implements a standard fare calculation (e.g., $1 per mile).
    - **PremiumRide:** Implements a premium fare calculation (e.g., $2 per mile).
    - These subclasses override the `fare()` method, demonstrating polymorphism by providing ride-specific fare calculations.
3. **Driver Class:**
    - Contains attributes like `driverID`, `name`, `rating`, and `assignedRides` (a list of rides completed by the driver).
    - Methods include `addRide(Ride ride)` to add a ride and `getDriverInfo()` to display driver details.
    - Uses encapsulation to keep the assigned rides private.
4. **Rider Class:**
    - Contains attributes like `riderID`, `name`, and `requestedRides` (a list of rides requested by the rider).
    - Methods include `requestRide(Ride ride)` to add a ride to the rider's list and `viewRides()` to display ride history.
5. **System Functionality:**
    - Demonstrates polymorphism by storing a collection of rides of different types and invoking the appropriate overridden `fare()` and `rideDetails()` methods.

Repository Contents

- **RideSharingSystem.cpp:**
  The complete C++ implementation of the Ride Sharing System.
- **RideSharingSystem.st:**
  The Smalltalk source file containing the implementation of the Ride Sharing System for a Smalltalk environment (e.g., Pharo or Squeak).
- **run.sh:**
  A cross-platform shell script that:
  - Detects the operating system (Linux, macOS, or Windows via Git Bash/Cygwin).
  - Checks for the presence of a C++ compiler (`g++`).
  - Compiles the C++ source code.
  - Runs the compiled executable.
  - Provides a reminder on how to load the Smalltalk implementation manually.

Prerequisites

- **C++ Environment:**
  Ensure that you have a C++ compiler installed (e.g., `g++`).
  - On Linux or macOS, install using your package manager (e.g., `apt-get`, `brew`, etc.).
  - On Windows, you can use MinGW or any POSIX-compliant shell such as Git Bash.
- **Smalltalk Environment:**
  A Smalltalk image/environment (e.g., Pharo, Squeak) is required to run the Smalltalk implementation manually.
- **Shell Environment:**
  A POSIX-compliant terminal (e.g., Bash) to run `run.sh`.

How to Run

Running the C++ Implementation

1. **Clone the Repository:**

```
git clone https://github.com/baralsamrat/MSCS632_Assignment_5.git
cd MSCS632_Assignment_5
```

2. **Run the Shell Script:**

   Make sure the `run.sh` script is executable and then run it:

```
chmod +x run.sh
./run.sh
```

The script will:

- Detect your operating system.
- Compile `RideSharingSystem.cpp` into an executable.
- Run the executable if compilation is successful.
- Provide instructions for using the Smalltalk version.

Running the Smalltalk Implementation

1. Open your Smalltalk environment (e.g., Pharo, Squeak).
2. Load the file `RideSharingSystem.st` using your environment's file browser or command-line tools.
3. Execute the code within the Smalltalk image.

OOP Concepts Demonstrated

- **Encapsulation:**
  Data members (e.g., `assignedRides` in the **Driver** class and `requestedRides` in the **Rider** class) are kept private and are accessible only via well-defined public methods.

- **Inheritance:**

  - The **Ride** class acts as the base class.
  - **StandardRide** and **PremiumRide** are subclasses that inherit from **Ride** and override the `fare()` method to implement specific fare calculations.

- **Polymorphism:**

  - A collection (e.g., a vector of `shared_ptr<Ride>` in C++) is used to store instances of different ride types.
  - The overridden `fare()` and `rideDetails()` methods are invoked correctly for each object type at runtime.

Additional Functionality and Creativity

Feel free to add additional functionality or modify the system to suit your needs. Contributions, ideas, and feature suggestions are welcome!