**Assignment 5: Developing a Class-Based Ride Sharing System**

**Rinku Gopali**

**University of the Cumberlands**

**Advanced Programming Languages (MSCS-632-M50)**

**Dr. Vanessa Cooper**

**03/30/2025**

**Introduction**

Ride-sharing apps have caused a stir in urban transportation by providing easy access to mobility that is both cost-effective and convenient. Structured software design provides a platform for handling ride requests, ride assignments to drivers, and fare calculation efficiently. In that regard, Object-Oriented Programming (OOP) allows modularity, reusability, and scalability in such types of applications.

The project implements a class-based Ride Sharing System in C++ and Smalltalk to emphasize the different OOP approaches within the two tools. Strict type-checking is associated with C++, the statically typed language, whereas Smalltalk, the dynamically typed language, transfers responsibilities through message-passing (Cui et al., 2023).

They include Ride, Driver, and Rider classes in the system that are examples of encapsulation, inheritance, and polymorphism. Encapsulation keeps the data secured, inheritance would allow claiming subclassing for StandardRide and PremiumRide, while polymorphism supports the uniform handling of rides.

**GitHub Link:** https://github.com/rgopali25573/Assignment-5-Developing-a-Class-Based-Ride-Sharing-System

**C++ Implementation**

**Encapsulation**:

The classes Driver and Rider encapsulate their ride lists: assignedRides and requestedRides, as they provide access through methods (addRide() and requestRide()). Encapsulation means that no one can directly modify the ride data from outside the class.

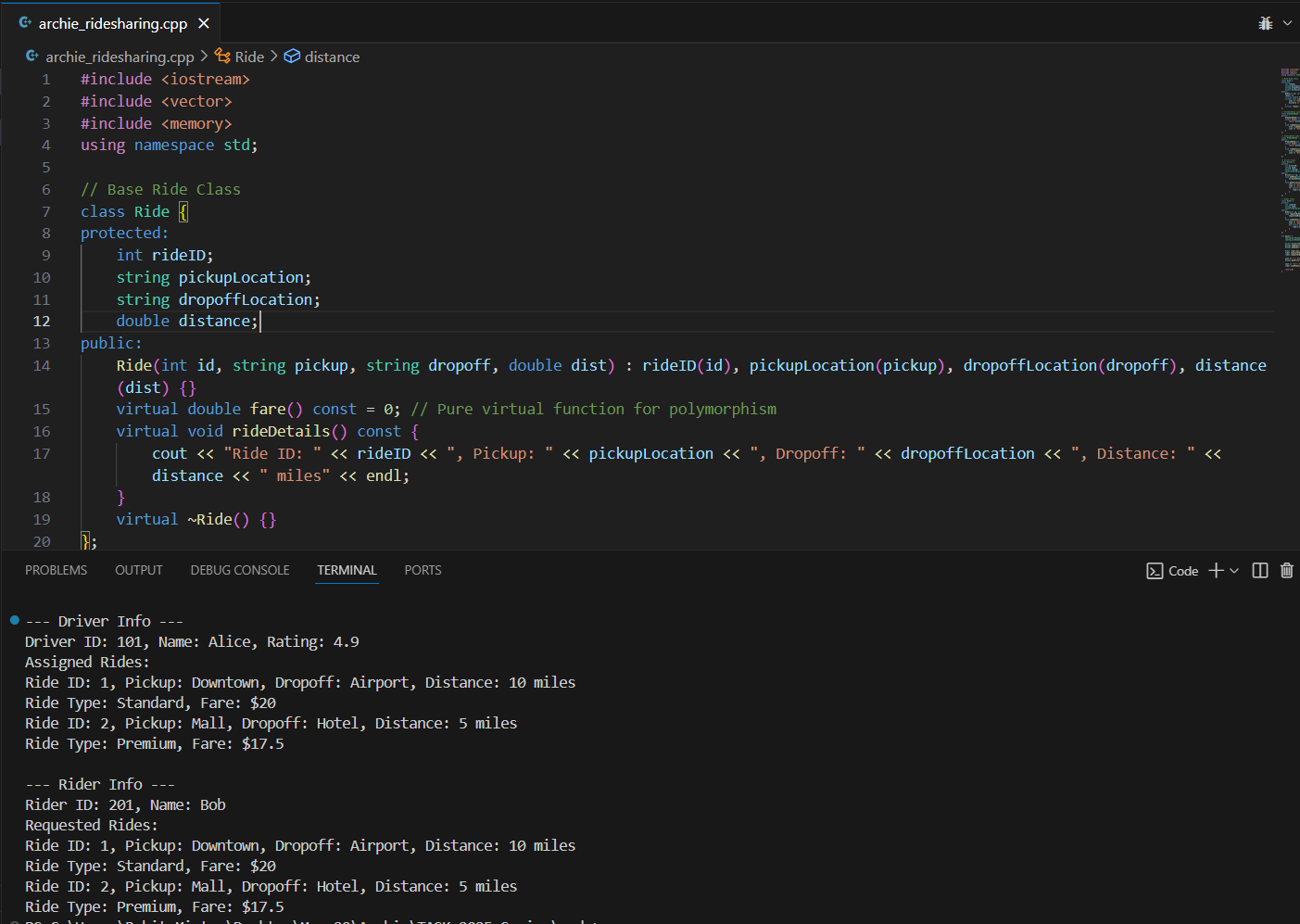
**Inheritance**:

There is a base class called Ride, which contains core ride details and declares a pure virtual fare() method.

The derived classes, StandardRide and PremiumRide, inherit from Ride and override fare() in order to define their specific fare calculations.

**Polymorphism**:

This vector of type shared\_ptr<Ride> stores various types of rides and thus grants polymorphic behavior on calling rideDetails() and fare().

Fig 1: C++ Code and Output

**Implementation in Smalltalk**

**Encapsulation**:

Instance variables, such as assignedRides and requestedRides, are kept in OrderedCollection, so access is through methods only.

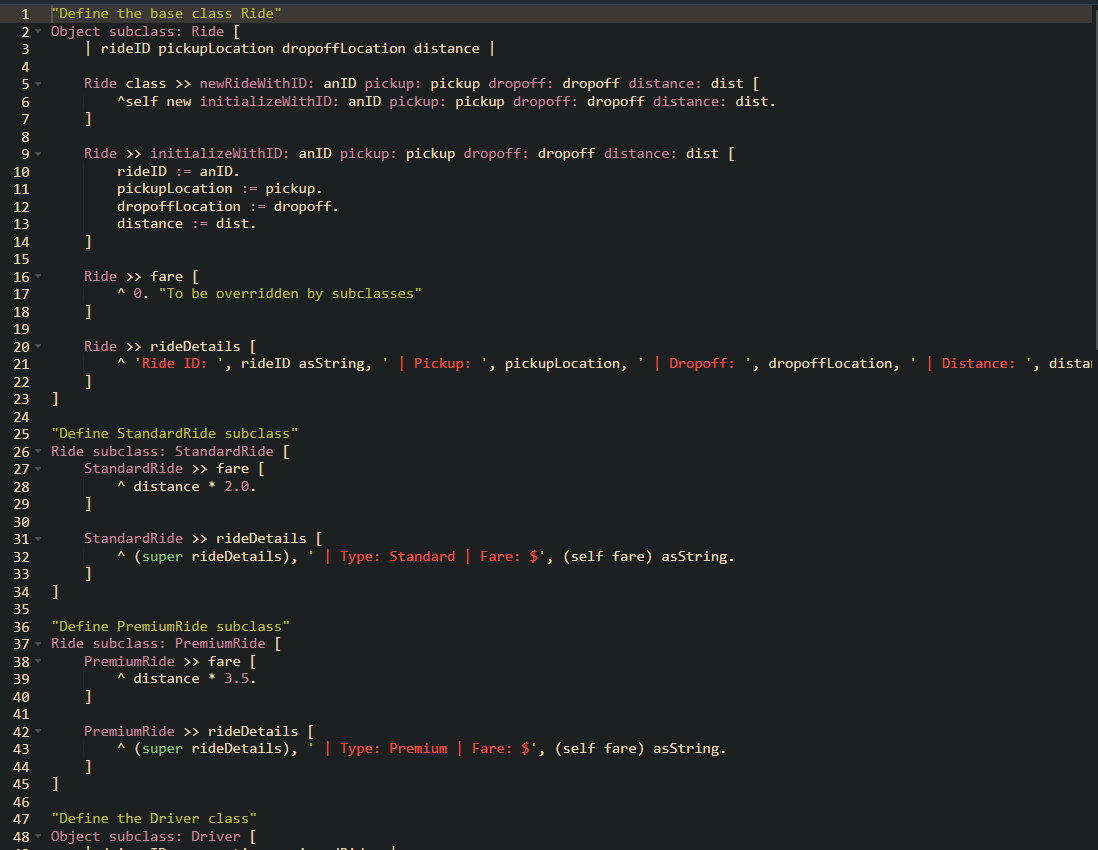
**Inheritance**:

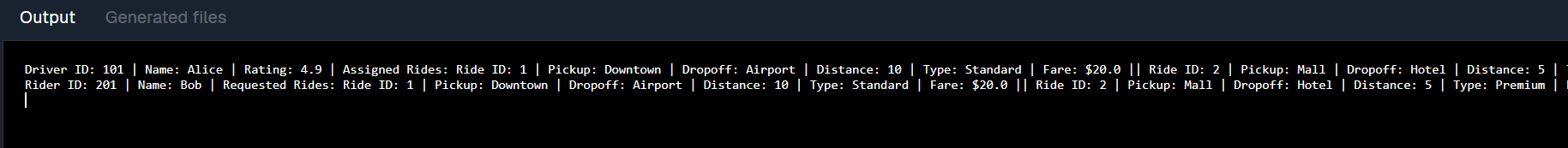
The Ride class is the parent class of StandardRide and PremiumRide, inheriting common ride attributes and defining fare().

The subclasses override fare() to determine differing rates.

**Polymorphism**:

Collections astoreRide objects, whereas calling rideDetails() dynamically dispatches to the correct implementation in the subclass.

Fig 2: Smalltalk code

Fig 3: Smalltalk co**de output**

**Comparison between C++ and Smalltalk:**

|  |  |  |
| --- | --- | --- |
| **Principles** | **C++ Implementation** | **Smalltalk Implementation** |
| Encapsulation | Private attributes in Driver and Rider | Uses message-passing to restrict direct access |
| Inheritance | StandardRide and PremiumRide inherit from Ride | Uses subclassing to achieve the same functionality |
| Polymorphism | Virtual functions and overridden methods (Zhang et al., 2021) | Smalltalk's dynamic message dispatch mechanism |

**References:**

Cui, S., Gao, Y., Unterguggenberger, R., Pichler, W., Livingstone, S., & Huang, J. (2023, May). SmallRace: Static Race Detection for Dynamic Languages-A Case on Smalltalk. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)* (pp. 1136-1147). IEEE. <https://jncsw.github.io/assets/paper/SmallRace_ICSE2023.pdf>

Zhang, M., Alawneh, A., & Rogers, T. G. (2021, March). Characterizing massively parallel polymorphism. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (pp. 205-216). IEEE. <https://par.nsf.gov/servlets/purl/10250390>