MSCS-632: Assignment 5: Lab Report: Developing a Class-Based Ride Sharing System

---

Sandesh Pokharel

University of the Cumberlands

MSCS-632 Advanced Programming Languages

Dr. Vanessa Cooper

March 30, 2025

Abstract

This report presents the implementation of a Ride Sharing System developed in both C++ and Smalltalk, demonstrating core object-oriented programming (OOP) principles: encapsulation, inheritance, and polymorphism. The purpose of this project is to apply language features across two different paradigms—one statically typed (C++) and one dynamically typed (Smalltalk)—to build a robust class-based application. The report also provides code structure, discussion of OOP usage, and sample outputs from both implementations.

Introduction

Object-oriented programming (OOP) emphasizes the use of real-world entities as objects, using concepts like encapsulation, inheritance, and polymorphism. This Ride Sharing System was developed in two languages to explore how these principles are applied across platforms. The system models a basic ride service with different types of rides, drivers, and riders interacting with each other.

C++ Implementation

Overview

The C++ version uses a class hierarchy to define ride types (Ride, StandardRide, PremiumRide) and entities (Driver, Rider). The program demonstrates OOP principles using inheritance and

virtual functions, encapsulating ride data and allowing polymorphic behavior when storing different ride types in a single collection.

Encapsulation in C++

Encapsulation is demonstrated through private class members and public methods:

- assignedRides in the Driver class and requestedRides in the Rider class are **private vectors**.
- Access is restricted using public methods like addRide() and viewRides().

This protects the integrity of the data and ensures that only controlled interactions are allowed.

Inheritance in C++

The StandardRide and PremiumRide classes inherit from the base class Ride. Each subclass overrides the fare() method to calculate pricing differently:

- StandardRide: $1.5/mile
- PremiumRide: $2.5/mile

This allows code reuse and extends base behavior appropriately.

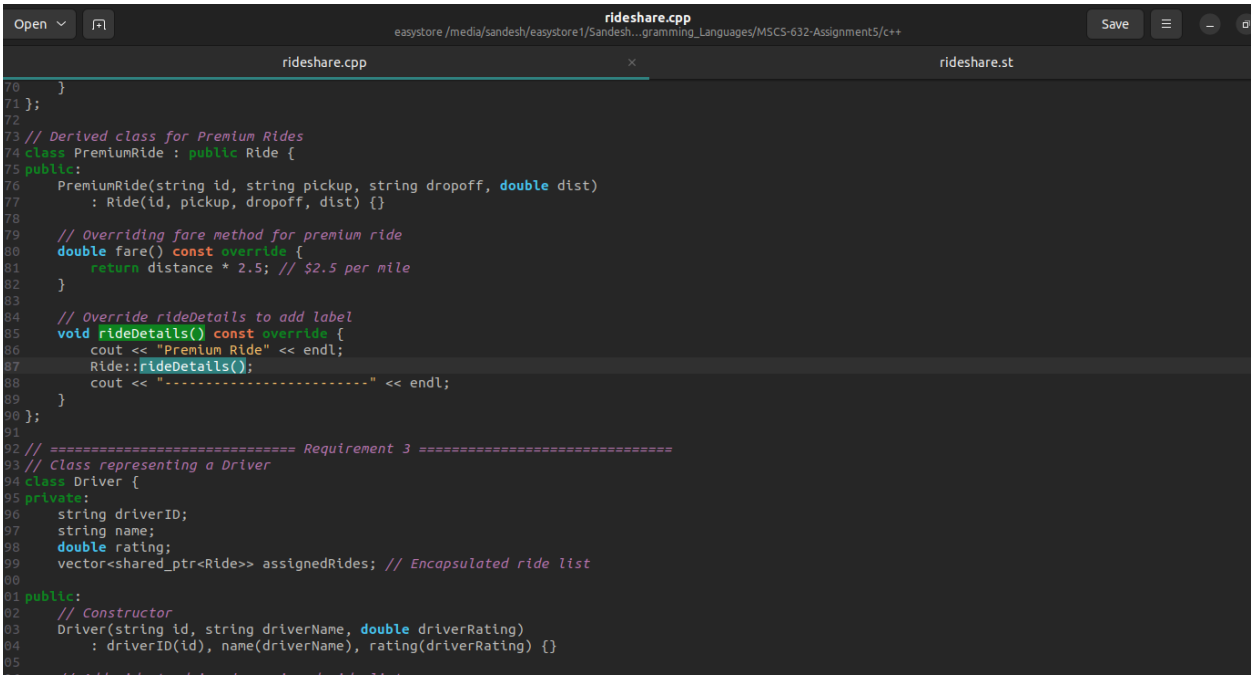
Polymorphism in C++

Polymorphism is demonstrated via a vector<shared_ptr<Ride>> that holds objects of both StandardRide and PremiumRide.

Using virtual methods like fare() and rideDetails(), the program can dynamically invoke the correct subclass method at runtime:

```
for (const auto& ride : rideList) {
    ride->rideDetails();  // Calls overridden methods based on actual type
}
```

☐ *Screenshot of C++ Code Implementation*



```
Open  rideshare.cpp
easystore /media/sandesh/easystore1/Sandesh...gramming_Languages/MSCS-632-Assignment5/c++
Save
rideshare.cpp  rideshare.st
        cout << "-------------------------" << endl;
    }
};

// ============================== Requirement 3 ==============================
// Class representing a Driver
class Driver {
private:
    string driverID;
    string name;
    double rating;
    vector<shared_ptr<Ride>> assignedRides; // Encapsulated ride list

public:
    // Constructor
    Driver(string id, string driverName, double driverRating)
        : driverID(id), name(driverName), rating(driverRating) {}

    // Add ride to driver's assigned ride list
    void addRide(shared_ptr<Ride> ride) {
        assignedRides.push_back(ride);
    }

    // Display driver info
    void getDriverInfo() const {
        cout << "\nDriver ID: " << driverID << ", Name: " << name << ", Rating: " << rating << endl;
        cout << "Completed Rides:" << endl;
        for (const auto& ride : assignedRides) {
            ride->rideDetails();
        }
    }
};

// ============================== Requirement 4 ==============================
// Class representing a Rider
class Rider {
```

☐ *Insert Screenshot of C++ Output*

```
Activities  Terminal  Mar 30 18:53
sandesh@sandesh-Inspiron-7373: /media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Assignment5/c++
sandesh@sandesh-Inspiron-7373: /media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Ad...  sandesh@sandesh-Inspiron-7373: /media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Ad...
sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Assignment5/smalltalk$ cd ../c++/
sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Assignment5/c++$ g++ -std=c++11 rideshare.cpp -o
 rideshare && ./rideshare

=== All Rides (Polymorphism Demo) ===
Standard Ride
Ride ID: RIDE001
Pickup: Downtown
Dropoff: Airport
Distance: 10 miles
Fare: $15
-------------------------
Premium Ride
Ride ID: RIDE002
Pickup: Station
Dropoff: Mall
Distance: 7.5 miles
Fare: $18.75
-------------------------

Driver ID: D001, Name: Alice, Rating: 4.8
Completed Rides:
Standard Ride
Ride ID: RIDE001
Pickup: Downtown
Dropoff: Airport
Distance: 10 miles
Fare: $15
-------------------------
Premium Ride
Ride ID: RIDE002
Pickup: Station
Dropoff: Mall
Distance: 7.5 miles
Fare: $18.75
-------------------------

Rider ID: U001, Name: Bob
Requested Rides:
```

Smalltalk Implementation

Overview

The Smalltalk version applies the same system model using a dynamic, message-passing approach. The core object structure remains the same, but Smalltalk’s syntax and flexibility offer a contrast to C++'s static typing and structure.

Encapsulation in Smalltalk

Each class uses instance variables (| name rating assignedRides |) that are accessed only via setter and getter methods:

- Driver and Rider keep ride collections in private variables (assignedRides, requestedRides).
- Interactions are done via methods like addRide: and viewRides.

This mirrors the concept of data hiding in a dynamic environment.

Inheritance in Smalltalk

StandardRide and PremiumRide are declared as subclasses of Ride using:

Ride subclass: StandardRide [...]

Each subclass overrides the fare and rideDetails methods to implement custom behavior. The superclass provides the base implementation, promoting reusability.

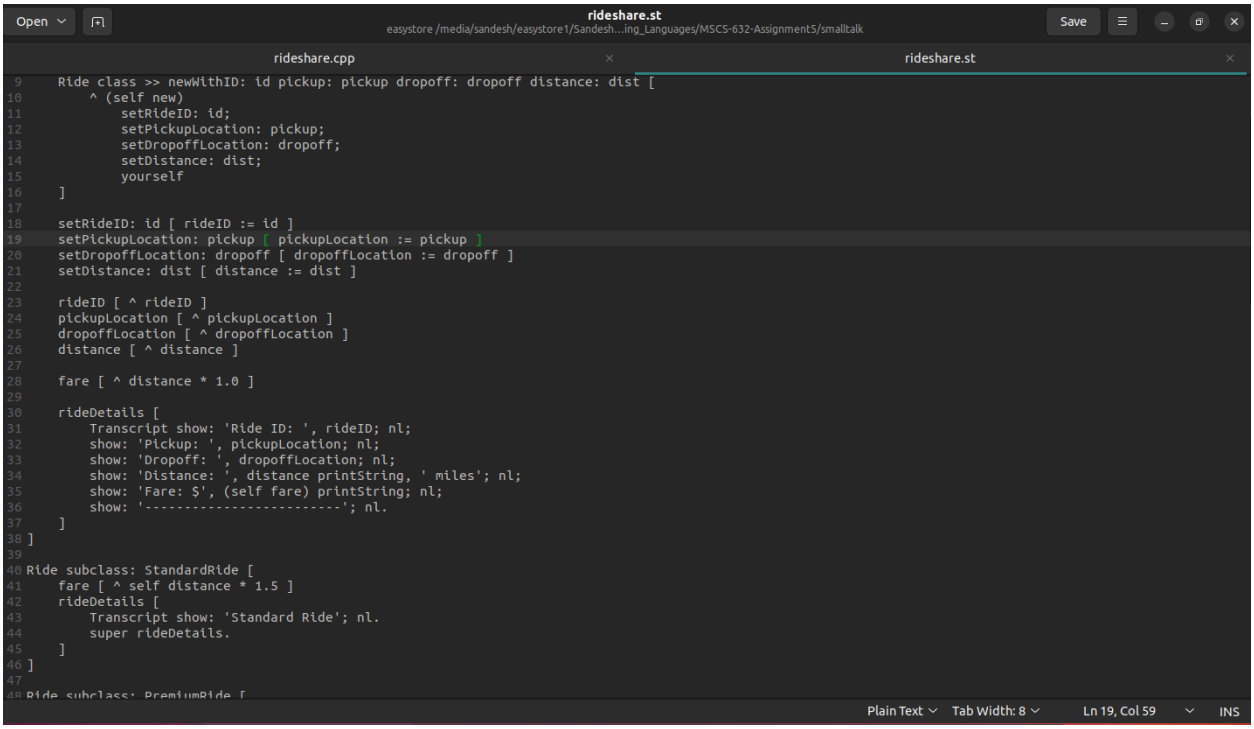
Polymorphism in Smalltalk

Polymorphism is shown by storing different ride types in a single OrderedCollection and invoking rideDetails dynamically:

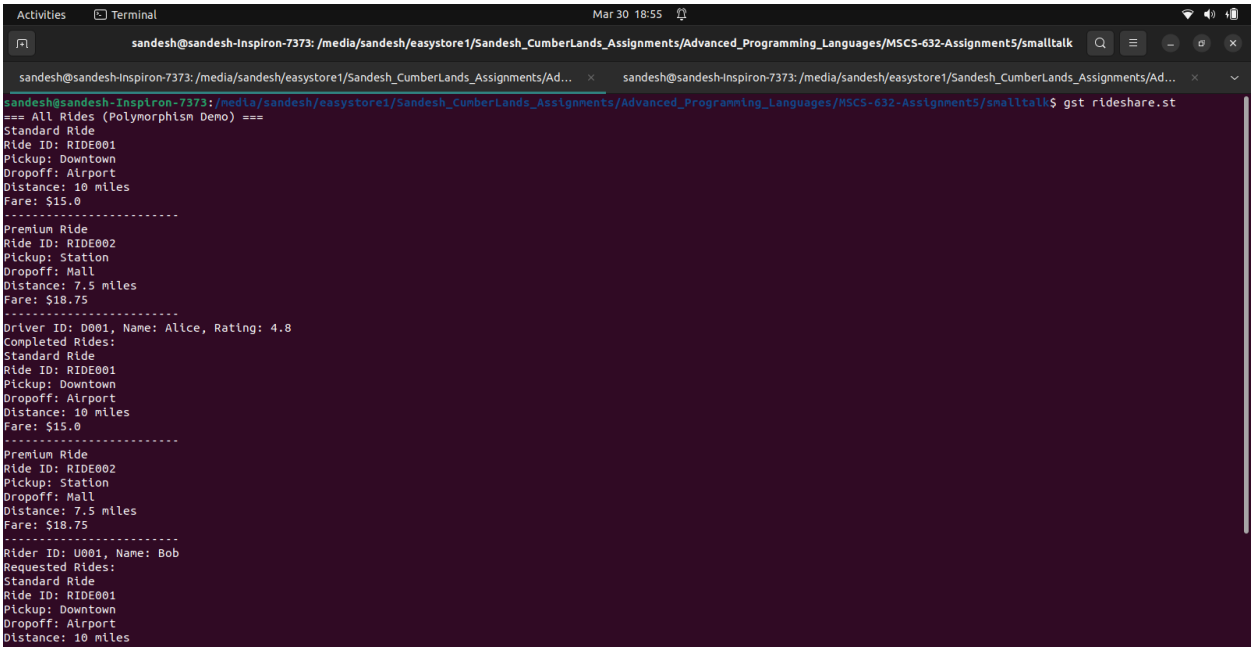
rideList do: [:ride | ride rideDetails].

No need to declare types; Smalltalk relies on message-passing and method dispatch at runtime.

☐ ***Screenshot of Smalltalk Code Implementation***



☐ ***Screenshot of Smalltalk Output]***



Conclusion

This assignment effectively demonstrates how object-oriented principles are used consistently across two different languages. While C++ provides strict type safety and compile-time checks, Smalltalk offers elegant dynamic typing and message-passing. Despite these differences, both implementations showcase robust encapsulation, clean inheritance hierarchies, and clear runtime polymorphism.

GitHub Repository

🔗 https://github.com/sanspokharel26677/MSCS-632-Assignment5

References

Sebesta, R. W. (2016). *Concepts of Programming Languages* (11th ed.). Pearson.

Oracle. (n.d.). *The Java™ Tutorials – Object-Oriented Programming Concepts.* Retrieved from https://docs.oracle.com/javase/tutorial/java/concepts/

Goldberg, A., & Robson, D. (1983). *Smalltalk-80: The Language and its Implementation*. Addison-Wesley.