

**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.

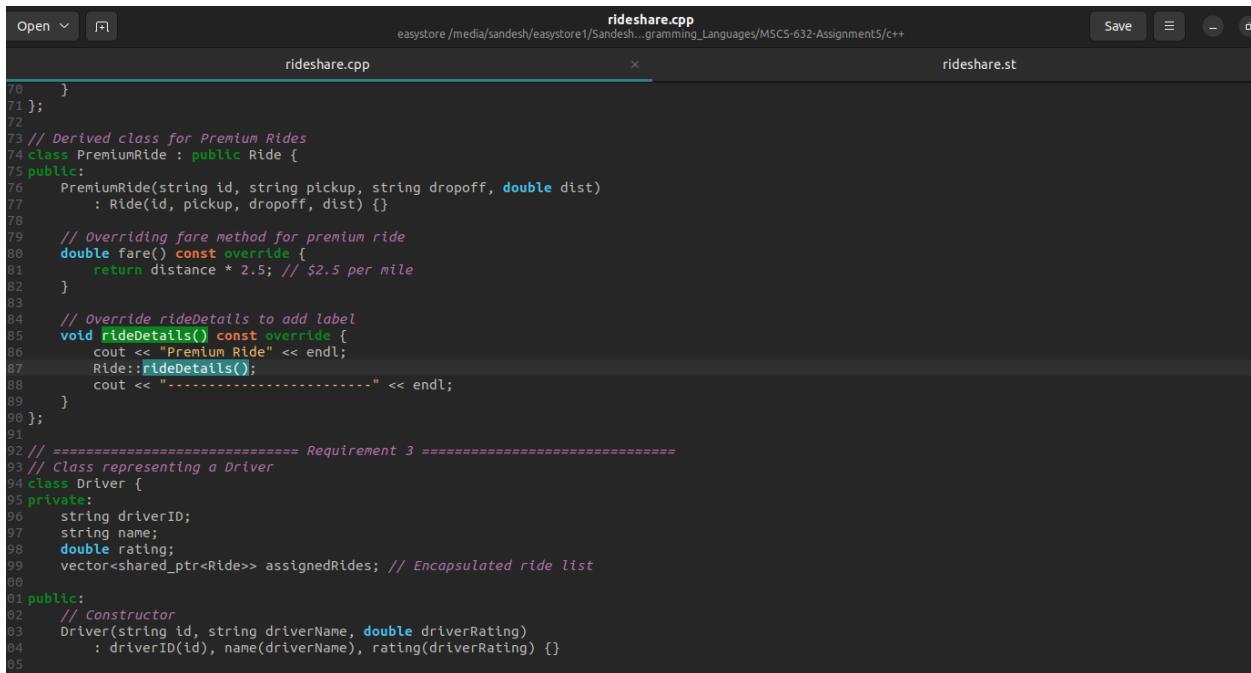
## **Polymorphism in C++**

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*



The screenshot shows a code editor window with two tabs: "rideshare.cpp" and "rideshare.st". The "rideshare.cpp" tab is active, displaying the following C++ code:

```
70 }  
71 };  
72 // Derived class for Premium Rides  
73 class PremiumRide : public Ride {  
74 public:  
75     PremiumRide(string id, string pickup, string dropoff, double dist)  
76         : Ride(id, pickup, dropoff, dist) {}  
77  
78     // Overriding fare method for premium ride  
79     double fare() const override {  
80         return distance * 2.5; // $2.5 per mile  
81     }  
82  
83     // override rideDetails to add label  
84     void rideDetails() const override {  
85         cout << "Premium Ride" << endl;  
86         Ride::rideDetails();  
87         cout << "-----" << endl;  
88     }  
89 };  
90  
91 // ===== Requirement 3 =====  
92 // Class representing a Driver  
93 class Driver {  
94 private:  
95     string driverID;  
96     string name;  
97     double rating;  
98     vector<shared_ptr<Ride>> assignedRides; // Encapsulated ride list  
99  
00 public:  
01     // Constructor  
02     Driver(string id, string driverName, double driverRating)  
03         : driverID(id), name(driverName), rating(driverRating) {}  
04  
05 }
```

The screenshot shows a code editor window with two tabs: "rideshare.cpp" and "rideshare.st". The "rideshare.cpp" tab is active, displaying C++ code for a ride sharing application. The code includes classes for Driver and Rider, and methods for adding rides and displaying driver info. Requirements 3 and 4 are indicated in the code. The "rideshare.st" tab is visible but contains no content.

```
rideshare.cpp
easystore /media/sandesh/easystore1/Sandesh...gramming_Languages/MSCS-632-Assignment5/c++
rideshare.cpp x rideshare.st Save

88     cout << "-----" << endl;
89 }
90 };
91 // ===== Requirement 3 =====
92 // Class representing a Driver
93 class Driver {
94 private:
95     string driverID;
96     string name;
97     double rating;
98     vector<shared_ptr<Ride>> assignedRides; // Encapsulated ride list
99
100 public:
101     // Constructor
102     Driver(string id, string driverName, double driverRating)
103         : driverID(id), name(driverName), rating(driverRating) {}
104
105     // Add ride to driver's assigned ride list
106     void addRide(shared_ptr<Ride> ride) {
107         assignedRides.push_back(ride);
108     }
109
110     // Display driver info
111     void getDriverInfo() const {
112         cout << "\nDriver ID: " << driverID << ", Name: " << name << ", Rating: " << rating << endl;
113         cout << "Completed Rides:" << endl;
114         for (const auto& ride : assignedRides) {
115             ride->rideDetails();
116         }
117     }
118 };
119
120 // ===== Requirement 4 =====
121 // Class representing a Rider
122 class Rider {
```

**Insert Screenshot of C++ Output**

The screenshot shows a terminal window with a dark theme. It displays the command `rideshare && ./rideshare` being run, followed by the application's output. The output shows three drivers (Alice, Bob, and Charlie) and their respective ride details, including pickup and dropoff locations, distance, and fare. The terminal also shows the completion of the program execution.

```
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/Advanced_Programming_Languages/MSCS-632-Assigment5/smalltalk$ cd .../c+/
sandesh@sandesh-Inspiron-7373: /media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Assigment5/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.

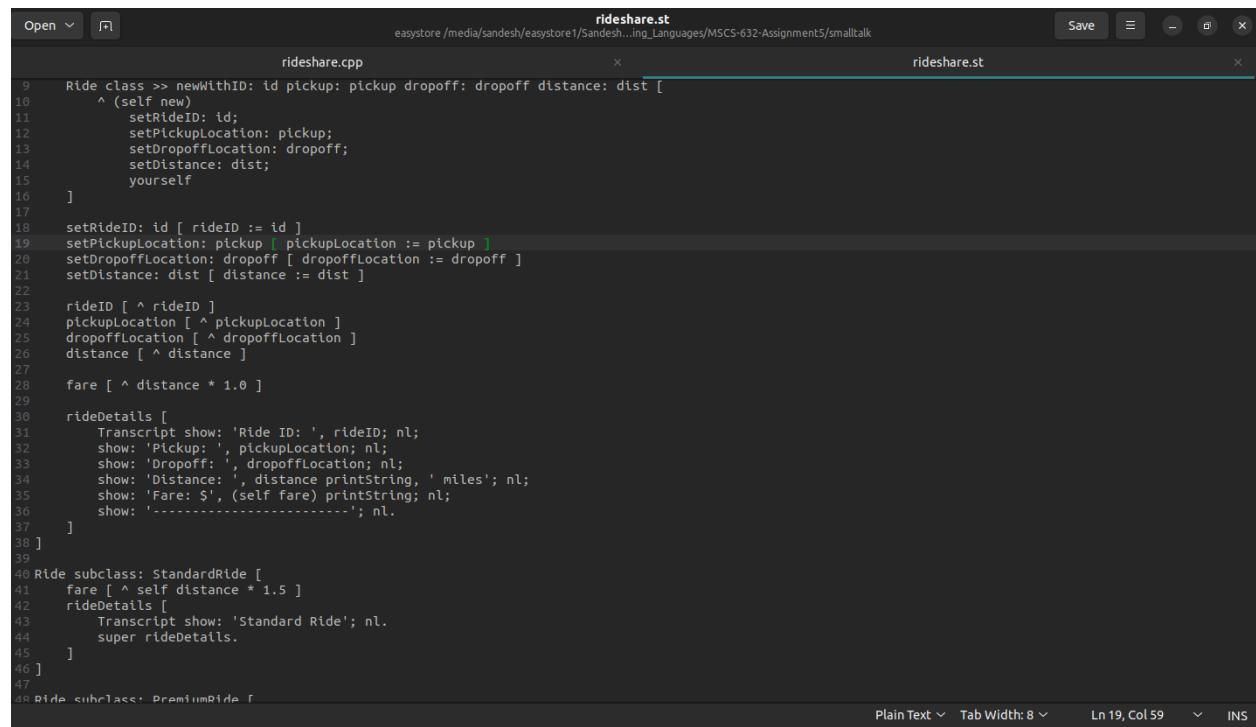
## Polymorphism in Smalltalk

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



The screenshot shows a dual-pane Smalltalk development environment. The left pane displays the source code for `rideShare.st`, which contains Smalltalk code defining a `Ride` class and its subclasses `StandardRide` and `PremiumRide`. The right pane displays the source code for `rideShare.cpp`, which contains C++ code implementing the same logic. Both panes show the same code content, illustrating the bridge between Smalltalk and C++.

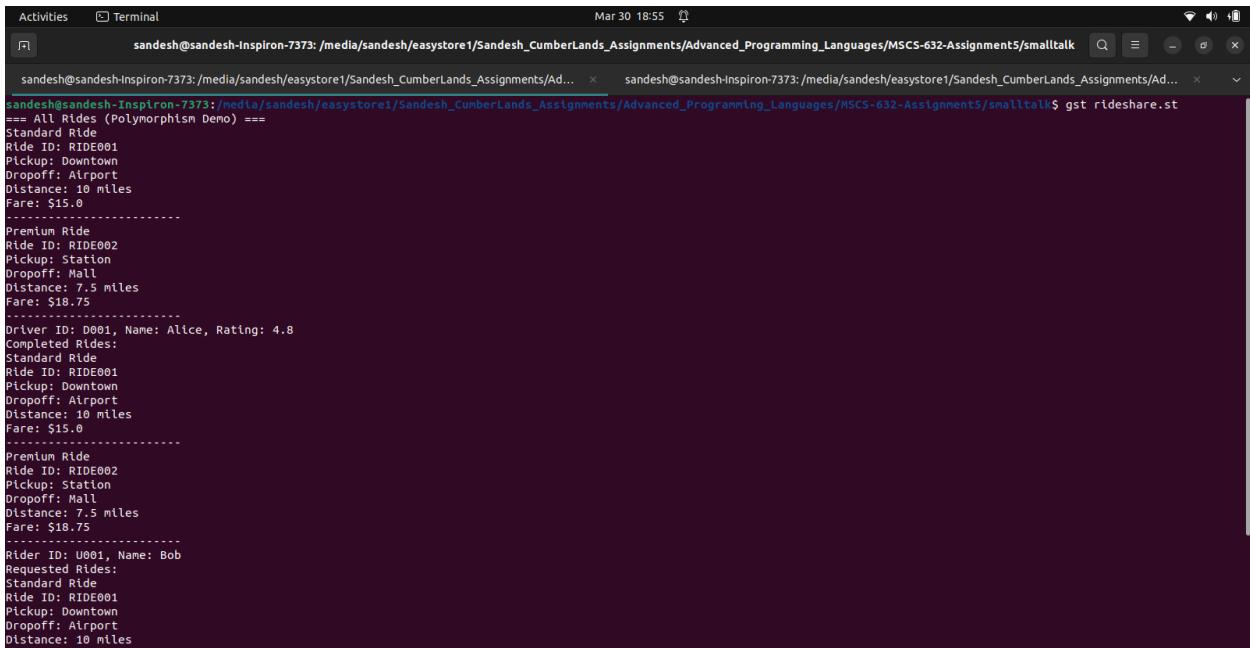
```
rideShare.st
easystore /media/sandesh/easystore1/Sandesh...ing_Languages/MSCS-632-Assignment5/smalltalk
rideShare.cpp
Ride class >> newWithID: id pickup: pickup dropoff: dropoff distance: dist [
    ^ (self new)
    setRideID: id;
    setPickupLocation: pickup;
    setDropoffLocation: dropoff;
    setDistance: dist;
    yourself
]
setRideID: id [ rideID := id ]
setPickupLocation: pickup [ pickupLocation := pickup ]
setDropoffLocation: dropoff [ dropoffLocation := dropoff ]
setDistance: dist [ distance := dist ]
rideID [ ^ rideID ]
pickupLocation [ ^ pickupLocation ]
dropoffLocation [ ^ dropoffLocation ]
distance [ ^ distance ]
fare [ ^ distance * 1.0 ]
rideDetails [
    Transcript show: 'Ride ID: ', rideID; nl;
    show: 'Pickup: ', pickupLocation; nl;
    show: 'Dropoff: ', dropoffLocation; nl;
    show: 'Distance: ', distance printString, ' miles'; nl;
    show: 'Fare: $', (self fare) printString; nl;
    show: '-----'; nl.
]
Ride subclass: StandardRide [
    fare [ ^ self distance * 1.5 ]
    rideDetails [
        Transcript show: 'Standard Ride'; nl.
        super rideDetails.
    ]
]
Ride subclass: PremiumRide [
```

rideShare.st

Save

Plain Text ▾ Tab Width: 8 ▾ Ln 19, Col 59 ▾ INS

## *Screenshot of Smalltalk Output*



```
sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Assignment5/smalltalk$ gst rideshare.st
*** All Rides (Polymorphism Demo) ***
Standard Ride
Ride ID: RIDE001
Pickup: Downtown
Dropoff: Airport
Distance: 10 miles
Fare: $15.0
-----
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: $15.0
-----
Premium Ride
Ride ID: RIDE002
Pickup: Station
Dropoff: Mall
Distance: 7.5 miles
Fare: $18.75
-----
Rider ID: U001, Name: Bob
Requested Rides:
Standard Ride
Ride ID: RIDE001
Pickup: Downtown
Dropoff: Airport
Distance: 10 miles
```

## 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/sanspokhare126677/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.