H-TechServices

Assignment<3> - Fall 2024

Course Title:	Java	Course Code:	Java-01	Credit Hours:	4(3,1)
Course Instructor:	Muhammad Hasnat Rasool	Program Name:	Java Mastery		
Due Date:	**010-12-2024**	**Maximum Marks:**		**10**	

Important Instructions / Guidelines:

The submission date is Dec 03, 2024. Submit your assignment in the form of a report. It should contain a problem statement, solution (code), and output. Your pdf/docx file name should be your name.

Upload your file on G**ithub .**

Ensure your program runs without errors and follows the structure .

Learning Objectives: Java programming.

Question: Implementing a Ride-Sharing System

Concepts Covered: Polymorphism, Inheritance, Interfaces

Total Marks: 10

Problem Statement:

Design a ride-sharing system where users can book different types of rides: **Standard Rides**, **Luxury Rides**, and **Bike Rides**. Each type of ride has different pricing rules and services. Your goal is to use **inheritance** and **polymorphism** to dynamically calculate fares and display ride details based on the ride type selected.

Class Structure (Attributes & Methods)

---

1. Interface: `Ride`

This interface defines the basic contract that all ride types must follow.

- **Methods (no attributes):**

- `double calculateFare(int distance)` → Calculate the fare for the given distance.
- `String getRideDetails()` → Return a string containing details of the ride.

---

2. Abstract Class: `BaseRide`

This class implements the `Ride` interface and provides a base structure for all types of rides.

- **Attributes:**
  - `String rideType` → Type of the ride (e.g., "Standard Ride", "Luxury Ride").
  - `double baseFare` → The base fare for all rides.
- **Methods:**
  - `BaseRide(String rideType, double baseFare)` → Constructor to initialize rideType and baseFare.
  - `abstract double calculateFare(int distance)` → To be implemented by subclasses.
  - `abstract String getRideDetails()` → To be implemented by subclasses.

---

3. Subclass: `StandardRide` (inherits from `BaseRide`)

Represents a standard ride with a fixed per-kilometer rate.

- **Attributes:**
  - `double perKmRate = 10` → Cost per kilometer.
- **Methods:**
  - `StandardRide()` → Constructor initializes rideType as "Standard Ride" and baseFare as 100.
  - `double calculateFare(int distance)` → Calculates fare using the formula:
    `baseFare + (distance * perKmRate)`
  - `String getRideDetails()` → Returns ride type and fare.

---

4. Subclass: `LuxuryRide` (inherits from `BaseRide`)

Represents a luxury ride with higher per-kilometer rate and additional luxury tax.

- **Attributes:**
  - `double perKmRate = 25` → Cost per kilometer.
  - `double luxuryTax` → 15% of base fare.
- **Methods:**
  - `LuxuryRide()` → Constructor initializes rideType as "Luxury Ride" and baseFare as 200.
  - `double calculateFare(int distance)` → Calculates fare using the formula:
    `baseFare + (distance * perKmRate) + (0.15 * baseFare)`
  - `String getRideDetails()` → Returns ride type, fare, and tax information.

---

5. Subclass: `BikeRide` (inherits from `BaseRide`)

Represents a bike ride with an optional helmet service.

- **Attributes:**
  - `double perKmRate = 5` → Cost per kilometer.
  - `boolean helmetIncluded` → True if helmet is included.
- **Methods:**
  - `BikeRide(boolean helmetIncluded)` → Constructor initializes rideType as "Bike Ride" and baseFare as 50.
  - `double calculateFare(int distance)` → Calculates fare using the formula:
    `baseFare + (distance * perKmRate)`
  - `String getRideDetails()` → Returns ride type, fare, and helmet inclusion status.

---

6. Class: `RideBookingSystem`

This class manages the booking process and displays ride details.

- **Methods:**
  - `void bookRide(Ride ride, int distance)` → Books a ride and prints the fare calculated using `calculateFare()`.
  - `void displayRideDetails(Ride ride)` → Prints the details of the ride using `getRideDetails()`.

Example Walkthrough:

Step 1: Create instances of different ride types.

StandardRide standardRide = new StandardRide();

LuxuryRide luxuryRide = new LuxuryRide();

BikeRide bikeRide = new BikeRide(true);

Step 2: Book a ride using `RideBookingSystem`.

RideBookingSystem system = new RideBookingSystem();

system.bookRide(standardRide, 10); // Book Standard Ride for 10 km

system.bookRide(luxuryRide, 10); // Book Luxury Ride for 10 km

system.bookRide(bikeRide, 5); // Book Bike Ride for 5 km

Step 3: Display ride details.

system.displayRideDetails(standardRide);

system.displayRideDetails(luxuryRide);

system.displayRideDetails(bikeRide);

Expected Output:

Booking Standard Ride for 10 km:

- Ride Type: Standard Ride
- Distance: 10 km

- Fare: 100 + (10 * 10) = 200

Booking Luxury Ride for 10 km:

- Ride Type: Luxury Ride
- Distance: 10 km
- Fare: 200 + (10 * 25) + 0.15 * 200 = 475

Booking Bike Ride for 5 km:

- Ride Type: Bike Ride
- Distance: 5 km
- Fare: 50 + (5 * 5) = 75
- Additional: Helmet Included

Tricky Points to Consider:

1. **Dynamic Dispatch:** The `bookRide()` method should invoke the correct overridden method (`calculateFare()`) based on the ride type, demonstrating polymorphism.
2. **Optional Attribute:** The `BikeRide` class should correctly display the helmet inclusion status based on the boolean attribute.
3. **Luxury Tax:** Ensure that the additional 15% luxury tax is applied correctly to luxury rides.

Marks Distribution (Total 10 Marks):

1. **Correct Inheritance Structure:** 2 Marks
2. **Polymorphism Implementation:** 3 Marks
3. **Correct Fare Calculation for Each Ride Type:** 3 Marks
4. **Conditional Logic for Optional Attributes:** 2 Marks