# Name: Chris Mawuko Tuffour

ID: 22039694

# Vehicle Rental Management System

## Abstract

This document outlines the implementation of a Vehicle Rental Management System demonstrating Object-Oriented Programming (OOP) principles including encapsulation, inheritance, polymorphism, abstraction, and composition.

## Introduction

The Vehicle Rental Management System provides a framework to manage vehicle rentals, including cars, motorcycles, and trucks. It incorporates core OOP principles to ensure a robust and scalable design.

## Core Classes and OOP Principles

### 1. Vehicle (Abstract Class)

The abstract base class 'Vehicle' defines the blueprint for all vehicle types. It includes private fields for encapsulation, abstract methods for rental cost calculation and availability checks, and a toString method for string representation.

### 2. Rentable Interface

The 'Rentable' interface defines common rental operations, including methods for renting and returning vehicles. This ensures a consistent API for all vehicle types.

### 3. Vehicle Specific Classes

Concrete classes 'Car', 'Motorcycle', and 'Truck' inherit from 'Vehicle' and implement the 'Rentable' interface. They override abstract methods to provide unique rental characteristics such as additional fees for GPS or load capacity.

### 4. Customer Class

The 'Customer' class maintains rental history and customer-specific details. It provides methods to manage and retrieve rental history.

### 5. RentalAgency Class

The 'RentalAgency' class manages the vehicle fleet, processes rentals, and handles returns. It demonstrates composition by maintaining a list of vehicles.

## Demonstrated OOP Principles

### 1. Encapsulation

Private fields and controlled access via getters and setters ensure proper data encapsulation. Input validation in setters protects data integrity.

### 2. Inheritance

The inheritance hierarchy allows 'Car', 'Motorcycle', and 'Truck' to share common functionality while providing specialized implementations.

### 3. Polymorphism

Method overriding allows specific behaviors for rental cost calculation and availability checks in each subclass. The 'Rentable' interface ensures polymorphic behavior for rental operations.

### 4. Abstraction

Abstract methods in the 'Vehicle' class define essential operations, while concrete classes implement the details.

### 5. Composition

The 'RentalAgency' class composes a list of vehicles to manage the fleet and demonstrate composition over inheritance.

## Testing and Validation

Unit tests are recommended to ensure functionality for each class. Test cases should validate encapsulation, inheritance, polymorphism, and abstraction implementations.

## Conclusion

The Vehicle Rental Management System demonstrates a clean and extensible design using OOP principles. Future enhancements can include loyalty programs, custom exceptions, and a rating system.