JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY

BACHELOR OF BUSINESS INFORMATION TECHNOLOGY

YEAR: 2.1

OBJECT ORIENTED PROGRAMMING 1

GROUP 4 ASSIGNMENT

GROUP M

MEMBERS:

JAYDEEH MAINA	HDB212-0422/2023
RYAN NDUNGU	HDB212-0404/2023
KEVIN KIPKURUI	HDB212-0447/2023
MANDY MWIKALI	HDB212-0401/2023
JOSEPH MUGO	HDB212-0396/2023
SILVIA WANGECI	HDB212-0459/2023
ANN MARY GITHINJI	HDB212-2348/2023
PAXTON KIPKIRUI	HDB212-0431/2023
ROBI REHEMA	HDB212-2244/2023
RAMSEY THUKU	HDB212-0397/202

Introduction

The Transport System is a robust software application built using C++ to manage vehicle data efficiently. Designed with Object-Oriented Programming (OOP) principles, it leverages encapsulation, inheritance, polymorphism, abstraction, exception handling, and file I/O to ensure a comprehensive and user-friendly experience. The primary goal of the system is to streamline the addition, storage, and display of vehicle information, including specific details for cars and trucks.

Users can interact with the system through a simple command-line interface, providing options to add vehicles, view their details, or save data to a file for persistence. The system categorizes vehicles into two primary types: cars and trucks, each represented as classes derived from a common base class, Vehicle. This design emphasizes code reuse and modularity, allowing shared behaviors to be implemented in the base class while enabling specific attributes and methods for each derived class.

This transport system is a scalable and maintainable solution, offering a foundation for future enhancements, such as support for additional vehicle types or integration with a graphical interface. By implementing key OOP principles, it showcases a modern approach to software design and data management for transport systems.

Functional Requirements

Features:

Add Vehicles:

Users can add details of cars or trucks.

Display Vehicles:

The system displays information about all stored vehicles.

Save Vehicles to File:

Save vehicle data to a file for future retrieval.

Command-Line Interface:

Users interact with the system through a text-based menu.

OOP Principles:

Encapsulation:

Private member variables in classes ensure data security.

Public methods provide controlled access.

Inheritance:

A base class Vehicle is extended by derived classes Car and Truck.

Polymorphism:

Virtual functions enable overriding in derived classes.

Abstraction:

Abstract methods are implemented in derived classes.

File I/O:

Vehicle details can be saved to and loaded from a file.

Non-Functional Requirements

Efficiency: Optimized for handling typical operations quickly.

Usability: The menu-based interface is user-friendly and intuitive.

Modularity: The code is separated into classes and reusable functions.

Code Quality: Proper naming conventions, comments, and adherence to C++ best practices.

Class Diagram

Classes:

Vehicle (Base class):

Attributes: make, model, year

Methods: displayInfo(), saveToFile() (pure virtual).

Car (Derived class):

Methods: Override displayInfo() and saveToFile().

Truck (Derived class):

Attributes: loadCapacity

Methods: Override displayInfo() and saveToFile().

Code Structure

Files:

Header Files:

Class declarations (Vehicle.h, Car.h, Truck.h).

Source Files:

Class implementations (Vehicle.cpp, Car.cpp, Truck.cpp).

Main program logic (main.cpp).

Conclusion

The transport system functions as a comprehensive solution for managing vehicle data using C++. It allows users to add details about cars and trucks, view all stored vehicles, and save vehicle information to a file for persistence. Each vehicle type is represented as a class derived from the base class Vehicle, showcasing core OOP concepts such as inheritance and polymorphism. The command-line interface facilitates easy interaction, providing a menu for users to add vehicles, display them, or save their details to a file. Exception handling ensures that file operations and invalid inputs are managed gracefully, making the system robust. Furthermore, the modular code design ensures maintainability and extensibility, allowing for future enhancements, such as support for additional vehicle types or a graphical user interface. By employing abstraction and encapsulation, the system secures and organizes vehicle data effectively. This application is a reliable and scalable system for anyone looking to manage transport-related information efficiently.