In Partial Fulfillment of the Requirements for the

CS 223 - Object-Oriented Programming

**“FOUR PRINCIPLES OF**

**OBJECT ORIENTED PROGRAMMING”**

Presented to:

Dr. Unife O. Cagas

Professor

Prepared by:

Gonzales, Cecille T.

BSCS 2A2 Student

**"Vehicle Management System: Cars, Motorcycles, and Trucks in Python"**

Project Title

**PROJECT DESCRIPTION**

The Vehicle Management System in Python exemplifies object-oriented programming principles through a well-structured hierarchy of vehicle classes, offering users the ability to create, manage, and interact with cars, motorcycles, and trucks. By encapsulating distinct attributes and behaviors within each class, such as starting engines, emitting horn sounds, and loading cargo, the system provides a comprehensive demonstration of inheritance and encapsulation concepts. This hierarchical structure not only facilitates code organization but also promotes code reusability and extensibility, allowing for the easy addition of new vehicle types or functionalities in the future. Through its clear organization and practical functionality, the project serves as an effective educational resource, allowing users to deepen their understanding of OOP principles while gaining hands-on experience in Python development. Furthermore, the system's modular design lends itself well to customization, making it suitable for a wide range of applications beyond basic vehicle management, such as simulations, games, or fleet tracking systems. Overall, the Vehicle Management System embodies the versatility and power of object-oriented design, empowering users to create robust, scalable, and efficient solutions in Python.

**OBJECTIVES**

1. **Define Vehicle Class**: Create a base class Vehicle to represent vehicles with attributes such as make, model, and year. Include methods for displaying vehicle information and controlling the engine.
2. **Implement Subclasses (Inheritance)**:Define subclasses for different types of vehicles, including Car, Motorcycle, and Truck, inheriting from the base Vehicle class. Each subclass should incorporate additional attributes specific to its type and introduce methods for specialized actions such as honking, performing wheelies, loading, and unloading cargo.
3. **Instantiate Objects:** Create instances of Car, Motorcycle, and Truck with appropriate attributes, such as make, model, year, and any subclass-specific attributes.
4. **Execute Actions**: Perform actions specific to each vehicle type, such as starting the engine, performing wheelies, or loading cargo, to ensure proper functionality.
5. **Display Information:** Call the display\_info() method for each vehicle object to verify accurate information display, including make, model, year, and any additional attributes.
6. **Ensure Error Handling:** Implement error handling mechanisms, such as providing default values for optional parameters or validating inputs, to ensure the robustness of the system.
7. **Test Inheritance and Polymorphism:** Validate inheritance and polymorphism by calling methods from both parent and subclass objects, ensuring that the appropriate methods are invoked based on the object's type.
8. **Enhance Readability and Documentation:** Ensuring that the code is well-commented and adheres to best practices for readability and documentation, including clear variable names, comments where necessary, and adherence to PEP 8 style guidelines.
9. **Ensure Functionality and Usability:** Confirm that the system meets the requirements by thoroughly testing all aspects of functionality, including object instantiation, method execution, and information display, while also considering user experience and ease of use.
10. **Refinement and Optimization:** Continuously refining and optimizing the codebase to improve performance, efficiency, and maintainability, addressing any issues or areas for improvement identified during testing and review processes.

**IMPORTANCE AND CONTRIBUTION OF THE PROJECT**

The provided code presents a simplified model of a vehicle management system, showcasing several implications and contributions across various domains:

* **Efficient Employee Management:** This project lays the groundwork for efficient employee management systems, fostering streamlined administrative processes and enhancing workflow efficiency.
* **Educational Tool:** The code serves as an educational tool, facilitating the teaching of fundamental object-oriented programming concepts like encapsulation and inheritance, preparing students for software development careers.
* **Real-world Application:** Students gain practical insights into applying object-oriented programming principles in real-world scenarios, fostering problem-solving skills essential for future roles in software development or IT-related industries.
* **Collaborative Learning:** Through collaborative projects involving code modification, students engage in teamwork, communication, and knowledge sharing, enriching their learning experience and peer interaction.
* **Career Preparation:** Understanding this code helps students develop foundational skills demanded in technology fields, offering practical experience in software development practices crucial for future job roles.

**FOUR PRINCIPLES OF OOP WITH CODE**

**ENHERITANCE**

Inheritance plays a pivotal role in organizing the class hierarchy and promoting code reuse. The Car, Motorcycle, and Truck classes all inherit from the Vehicle superclass, establishing an "is-a" relationship and allowing them to inherit attributes and methods such as make, model, year, display\_info(), start\_engine(), and stop\_engine(). This inheritance not only facilitates the reuse of common functionality across different types of vehicles but also promotes abstraction by providing a unified interface for interacting with vehicles regardless of their specific type. By leveraging inheritance, the code achieves a modular and extensible design, where each subclass can focus on extending and customizing the behavior inherited from the superclass, thereby adhering to the principles of object-oriented programming.



**ENCAPSULATION**

Encapsulation is exemplified in this code through the organization of related data and behaviors within classes such as Vehicle, Car, Motorcycle, and Truck. Each class encapsulates attributes like make, model, and year, along with methods like display\_info() and start\_engine(), reflecting the principle of encapsulation in object-oriented programming.



**POLYMORPHISM**

Polymorphism is exemplified in this code through method overriding, where each subclass—Car, Motorcycle, and Truck—provides its own implementation of the display\_info() method inherited from the Vehicle class. This allows for different behaviors based on the specific type of vehicle. For instance, the display\_info() method in the Car subclass might include additional details such as the number of doors, while the Truck subclass might include cargo capacity information. By overriding the method in this manner, the code achieves flexibility and extensibility, enabling each type of vehicle to exhibit its unique characteristics while still adhering to a common interface.



**ABSTRACTION**

Abstraction is evident in this code through the use of classes and methods to represent real-world entities and their behaviors in a simplified manner. Each class, such as Vehicle, Car, Motorcycle, and Truck, abstracts the common properties and actions associated with vehicles, such as make, model, year, starting the engine, and loading cargo. This abstraction allows users to interact with these objects without needing to know the intricate details of their implementations, promoting a clearer understanding and efficient use of the code.



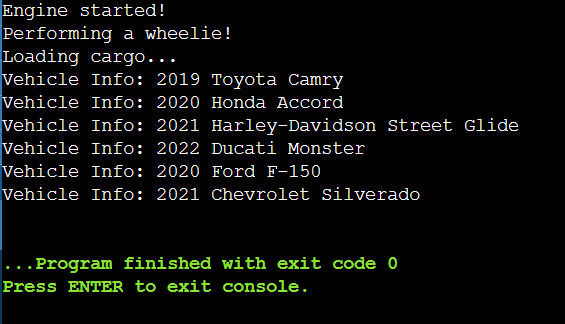
**SOFTWARE USED:**

* ONLINE GDB

**HARDWARE USED:**

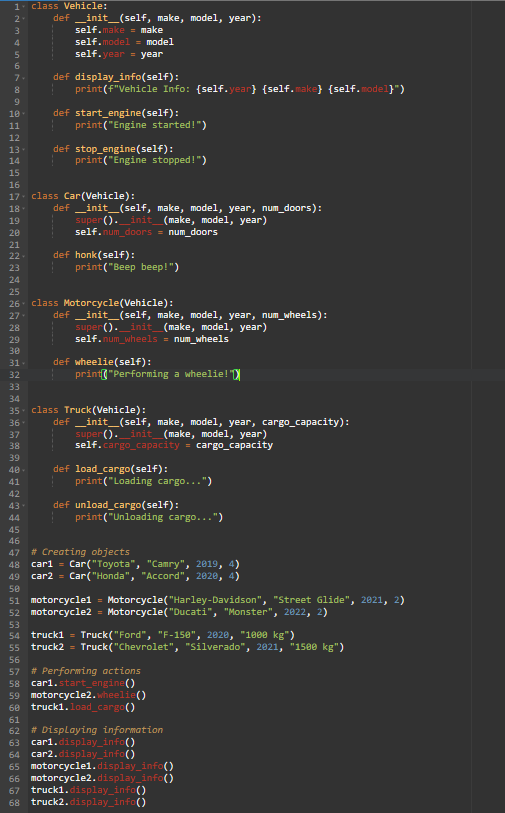
* MOBILE PHONE
* LAPTOP

**OUTPUT**



Based on the code, it indicates the instantiation of several vehicles, each belonging to different classes such as Car, Motorcycle, and Truck. Through method invocations like start\_engine(), wheelie(), and load\_cargo(), the code illustrates the diverse actions associated with these vehicles, ranging from starting the engine of a car to performing a wheelie with a motorcycle and loading cargo onto a truck. Moreover, the display\_info() method presents comprehensive details about each vehicle, including its make, model, and year of manufacture, offering a holistic view of their attributes. This portrayal underscores the versatility and functionality of the Vehicle class and its subclasses, showcasing their capability to model distinct types of vehicles and simulate their behaviors effectively.

**CODE**



**USER GUIDE: VICHICLE MANAGEMENT SYSTEM**

The Vehicle Management System is a Python-based application designed to facilitate the management of various types of vehicles, such as cars, motorcycles, and trucks. It enables users to handle basic vehicle details, perform specific actions associated with each vehicle type, and view comprehensive information about them.

1. **Creating a Vehicle Instance:**

To create a Vehicle instance, you need to provide the make, model, and year of manufacture. This instance represents a generic vehicle within the system.

1. **Creating a Car Instance:**

To create a Car instance, provide additional details such as the make, model, year, and number of doors. This instance represents a specific type of vehicle, namely a car.

1. **Creating a Motorcycle Instance:**

Similarly, to create a Motorcycle instance, include the make, model, year, and the number of wheels. This instance represents a motorcycle within the system.

1. **Creating a Truck Instance:**

To create a Truck instance, input the make, model, year, and cargo capacity of the truck. This instance represents a truck capable of transporting cargo.

1. **Performing Actions:**

Utilize methods such as start\_engine() to start the engine of a vehicle, wheelie() to perform a wheelie with a motorcycle, and load\_cargo() to load cargo onto a truck. These actions demonstrate the functionality associated with each vehicle type.

1. **Displaying Vehicle Information:**

After performing actions or making changes, use the display\_info() method to view detailed information about each vehicle, including its make, model, and year of manufacture. This step allows users to observe the effects of the actions taken within the system.

**REFERENCES**

* <https://www.onlinegdb.com/>
* <https://python.org/>
* <https://chatgpt.com/>