In Partial Fulfillment of the Requirements for the

CS 223 – Object-Oriented Programming

Vehicle Management System

Presented to:

Dr. Unife O. Cagas
Professor

Prepared by:
Julve, John Andrei S.

BSCS-2A

May 2024

PROJECT DESCRIPTION

The implementation of this system is in relation to the car world in which helps to demonstrate the structure of Object-Oriented Programming (OOP) using Python programming language. In this document consists of illustration to gain insights into how encapsulation, abstraction, and polymorphism are applied in the designed program using the concept of automobile industry on how the vehicle works using Python.

OBJECTIVES

The primary objective of the current program is to provide a demonstration on how Object-oriented programming concepts and how it can be implemented through various principles such us encapsulation, abstraction, and polymorphism. With the goal to simplify the understanding using implementation of the functionally of Cars and its other functionality. With the clear intention to help ensure in aiding users understanding using implementation of the functionally of Cars and its other functionality.

- **Demonstrate OOP Principles:** Illustrate the functionality of OOP principles through a concise representation of automobile structures.
- **Empower User Creativity:** Enable users to construct their own programs by providing guidance and insights gleaned from the implemented functionalities within the program.

IMPORTANCE AND CONTRIBUTION

This created project solely relies on only providing a simple guidance as to how you can create your own fully functional program using OOP with a simple concept to be used as reference to beginners who needs to get into the advance level of structure building their python code. This program may be simple but can impact a significant value to new beginners in the programming world.

HARDWARE AND SOFTWARE

Hardware:

- Computer or Laptop

Software:

- Visual Studio Code
- Python 3.11

Principles of Object-Oriented Programming:

- **INHERITANCE** – The program consists of demonstrating that there is a relationship between Object Classes of with the main Class Vehicle and the sub classes are SUV, SEDAN, and CARS. In which they can inherit the attributes from the Vehicle Class.

CODE DEMONSTRATION

```python
class CarOptions:
    def __init__(self):
        self.mileage = 0

    def Travel(self, distance: int):
        print(f"Your Car has traveled {distance} km")
        self.mileage += distance

    def CarDetails(self, make, model, year):
        print(f"""
              Car Name: {make}
              Model Name: {model}
              Year: {year}
              Mileage: {self.mileage}
              """)

    def carUpdate(self, model):
        if self.mileage >= 10000:
            print(f"Your Car {model} needs maintenance")
        elif self.mileage >= 1000:
            print(f"Your Car {model} still has low mileage")
        else:
            print(f"Your Car {model} doesn't need maintenance")

```

- ENCAPSULATION – Encapsulation refers to how your class is bundled in a single unit. Here we can demonstrate on class CarsOption as it encapsulates its entire function that can only be accessed when you use the class CarsOption. Here, the mileage variable can only be accessed inside the CarsOption and cannot be called outside the class.

CODE DEMONSTRATION

```python
class Car(Vehicle):
    def __init__(self, make, model, year):
        super().__init__(make, model, year)
        self.car_options = CarOptions()

    def accelerate(self):
        distance = int(input("How many KM do you want to travel: "))
        print("Car is accelerating...")
        self.car_options.Travel(distance)


    def brake(self):
        print("Car is braking...")

    def honk(self):
        print("Car is honking...")

    def details(self):
        self.car_options.CarDetails(self.make, self.model, self.year)
```

```python
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def accelerate(self):
        pass

    def brake(self):
        pass

    def honk(self):
        pass

    def details(self):
        pass
```

- ABSTRACTION – involves encapsulating the attributes and behaviors of classes, allowing for the use of their functionalities without exposing the complexity of the entire implementation. It focuses on providing a simplified view of objects and their interactions, showcasing only the essential features necessary for users to understand and utilize without delving into the intricate details of how they are implemented.

CODE DEMONSTRATION

```python
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def accelerate(self):
        pass

    def brake(self):
        pass

    def honk(self):
        pass

    def details(self):
        pass
```

- POLYMORPHISM – The functions from Classes CAR, SUV, SEDAN represents the polymorphism principle as they all are dependent on the Class Vehicle on how each class should be implemented with functions Accelerate, honk, and brake needed to be implemented on each class inherited from abstract class Vehicle.

CODE DEMONSTRATION

```python
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def accelerate(self):
        pass

    def brake(self):
        pass

    def honk(self):
        pass

    def details(self):
        pass

```

```python
class Car(Vehicle):
    def __init__(self, make, model, year):
        super().__init__(make, model, year)
        self.car_options = CarOptions()

    def accelerate(self):
        distance = int(input("How many KM do you want to travel: "))
        print("Car is accelerating...")
        self.car_options.Travel(distance)


    def brake(self):
        print("Car is braking...")

    def honk(self):
        print("Car is honking...")

    def details(self):
        self.car_options.CarDetails(self.make, self.model, self.year)
```

FULL CONTENT OF THE PROGRAM

```python
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def accelerate(self):
        pass

    def brake(self):
        pass

    def honk(self):
        pass

    def details(self):
        pass

class CarOptions:
    def __init__(self):
        self.mileage = 0

    def Travel(self, distance: int):
        print(f"Your Car has traveled {distance} km")
        self.mileage += distance

    def CarDetails(self, make, model, year):
        print(f"""
              Car Name: {make}
              Model Name: {model}
              Year: {year}
              Mileage: {self.mileage}
              """)

    def carUpdate(self, model):
        if self.mileage >= 10000:
            print(f"Your Car {model} needs maintenance")
        elif self.mileage >= 1000:
            print(f"Your Car {model} still has low mileage")
        else:
            print(f"Your Car {model} doesn't need maintenance")

```

```
class Car(Vehicle):
    def __init__(self, make, model, year):
        super().__init__(make, model, year)
        self.car_options = CarOptions()

    def accelerate(self):
        distance = int(input("How many KM do you want to travel: "))
        print("Car is accelerating...")
        self.car_options.Travel(distance)


    def brake(self):
        print("Car is braking...")

    def honk(self):
        print("Car is honking...")

    def details(self):
        self.car_options.CarDetails(self.make, self.model, self.year)

class Sedan(Vehicle):
    def __init__(self, make, model, year):
        super().__init__(make, model, year)
        self.car_options = CarOptions()

    def accelerate(self):
        distance = int(input("How many KM do you want to travel: "))
        print("Sedan is accelerating...")
        self.car_options.Travel(distance)


    def brake(self):
        print("Sedan is braking...")

    def honk(self):
        print("Sedan is honking...")

    def details(self):
        self.car_options.CarDetails(self.make, self.model, self.year)

class SUV(Vehicle):
    def __init__(self, make, model, year):
        super().__init__(make, model, year)
        self.car_options = CarOptions()

    def accelerate(self):
        distance = int(input("How many KM do you want to travel: "))
        print("SUV is accelerating...")
        self.car_options.Travel(distance)

    def brake(self):
        print("SUV is braking...")

    def honk(self):
        print("SUV is honking...")

    def details(self):
        self.car_options.CarDetails(self.make, self.model, self.year)

```

User Guide:

STEP 1: Choose an Object you want to Create it could be SUV, SEDAN, or if you want to not be specific of type of car than just use Car class inside its parameter, we have the first the maker/brand, then the model, and finally an integer of the version released of the Vehicle.

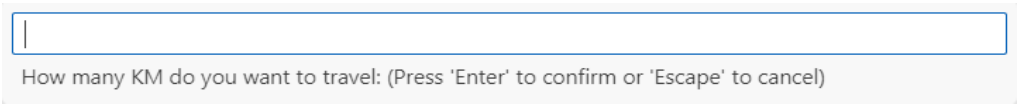
```
S1 = Car("Toyota", "Supra GR", 2019)
S2 = Sedan("Toyota", "Supra GR", 2019)
S3 = SUV("Toyota", "Supra GR", 2019)
```

STEP 2: The usage of how to use your vehicle, to use your vehicle you can call your variable with its 3 functionalities like accelerate, honk and brake.

Demonstration on acceleration function:

```
S1 = Car("Toyota", "Supra GR", 2019)
S1.accelerate()
```

Note: As you start to compile/run the program the function will now ask for mileage your car will run.



STEP 3(Car details): in the last step the program can show you the full information of your created vehicle.

```
#you only need to implement the values of your objects
S1.car_options.CarDetails(S1.make, S1.model, S1.year)
```

OUTPUT:

```
S1 = Car("Toyota", "Supra GR", 2019)
S1.accelerate()

#you only need to implement the values of your objects
S1.car_options.CarDetails(S1.make, S1.model, S1.year)
```

✓ 2.8s

```
Car is accelerating...
Your Car has traveled 15 km

        Car Name: Toyota
        Model Name: Supra GR
        Year: 2019
        Mileage: 15
```

Note: As you can see we have a mileage of 15km as I have demonstrated that Step 2, as activating accelerate program I ask for an input which I inserted 15 as demo.

DESCRIPTION

The program establishes how the functionality of a car works and how it can relate to programming as reference for its structure to easily understand the concepts of the principles in Object-Oriented Programming. As this program brings value as a simple guide to help simplify the concepts into relatable representation most people can easily work with.

CONCLUSION

In conclusion, the program serves as an effective educational resource for understanding the core principles of Object-Oriented Programming (OOP) with the context of automotive structures. By analyzing and breaking down the functionality of cars and implementing it through classes and methods, it provides a framework for grasping abstract programming concepts. Throughout the program, we've witnessed how encapsulation, abstraction, and polymorphism are not just theoretical concepts but integral components of real-world software development. With the design of classes, we can encapsulate the data which can be used to promote reusability.

Additionally, the program brings a curiosity on programming as it provides guidance and insights as to how the functionality works overall. Giving users more creativity and experimentation. As technology evolves, this the principles within the program of provide less complexities of OOP and inspires exploration, growing into the dynamic programming world.

REFERENCES

OOP Principles: https://www.geeksforgeeks.org/python-oops-concepts/

Python Tutorial: https://www.w3schools.com/python/