CMP1903 Object-Oriented Programming
Workshop 7: Vehicle Management System

This workshop covers **inheritance** and **class hierarchy** in Object-Oriented Programming (OOP). By the end, you should be able to:

- Understand and implement **class hierarchies** with inheritance.
- Use **encapsulation** to restrict direct access to class members.
- Implement **virtual** and **override** methods.
- Utilize **polymorphism** and work with base class collections.

You are designing a **Vehicle Management System** for a transportation company. The system should manage different types of vehicles, including **Cars, Bikes, Trucks, and Buses**. While all vehicles share some common attributes, they also have unique features.

For this workshop, you have been provided with class templates, and you must complete the remaining tasks as specified.

Class Hierarchy

Below is the **class hierarchy** for the Vehicle Management System, including attributes and methods for each class:

Base Class:

- **Vehicle (Attributes:** brand, year, **Methods:** Vehicle→ Constructor, DisplayInfo() → Displays vehicle details).

Derived Classes:

- **Car** inherits from Vehicle **(Attributes:** doors, **Methods:** Car → Constructor, DisplayInfo() → Displays car-specific details in addition to vehicle details)
- **Bike** inherits from Vehicle **(Attributes:** hasGear, **Methods:** Bike → Constructor, DisplayInfo() → Displays bike-specific details in addition to vehicle details)
- **Truck** inherits from Vehicle **(Attributes:** loadCapacity, **Methods:** Truck → Constructor, DisplayInfo() → Displays truck-specific details in addition to vehicle details)
- **Bus** inherits from Vehicle **(Attributes:** passengerCapacity , **Methods:** Bus → Constructor, DisplayInfo()→ Displays bus-specific details in addition to vehicle details)

Some Implementation Guidelines

Encapsulation Rules:

- A vehicle's **brand** and **manufacturing year** cannot be changed.
- A car's **number of doors** cannot be changed.
- A bike's **gear system** does not change.
- A truck's **load capacity** can be modified but must be **validated**:
    - It must be **greater than zero** and **cannot exceed 2300**.
- A bus's **passenger limit** may change based on regulations but must be **greater than zero**.

Constructor Implementation:

base(params)

- Derived classes should initialise **base class attributes** through constructors (derived constructors).

Virtual and Override Methods:

- The DisplayInfo() method in the **Vehicle** class should be **virtual**, allowing each vehicle type to provide additional details by overriding it.

Testing Polymorphism:

- Store multiple vehicle objects in a **list** and use **polymorphism** to display their details.
- Create an **array of vehicles** and call DisplayInfo() using **base class pointers**.

Extra task (optional):

Each vehicle type produces a unique sound when its engine starts. Implement the **StartEngine()** function to output the specific sound for each vehicle when the engine is started. Consider how you can incorporate this functionality into your existing code and implement it accordingly. Which object-oriented concept are you applying in this implementation?

Finally, consider the collection of Vehicles you created earlier. In your implementation (program.cs), are you able to start the engines of only the Trucks in this collection?

```
foreach Vehicle
    if Vehicle is Truck truck
        Truck truck = vehicle as Truck
```