OBJECT-ORIENTED PROGRAMMING

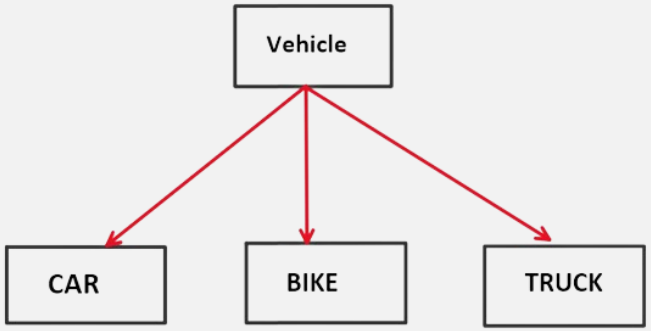
OOP PRINCIPLES

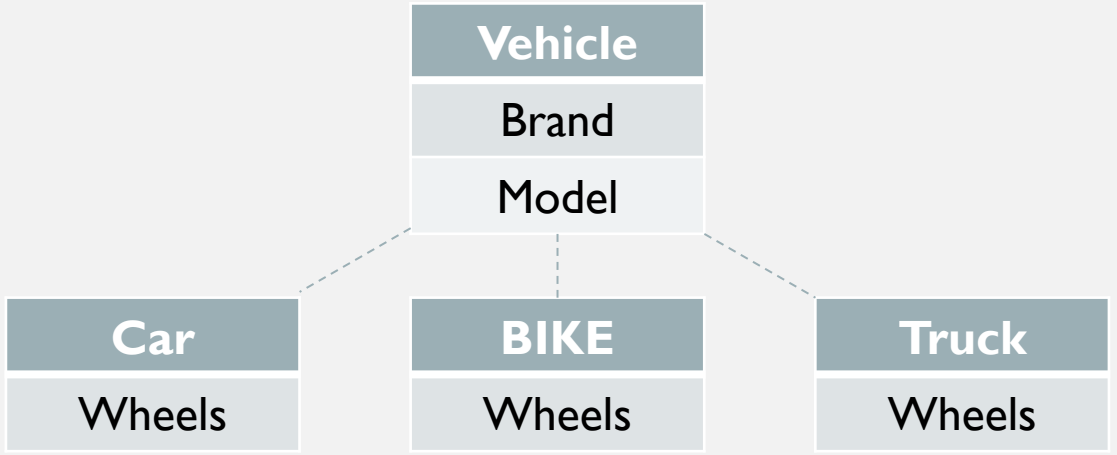
OOP PRINCIPLES

1. **Inheritance** 1 or more child classes receiving fields, methods etc. from a common parent.

Here's what it means:

- ❑ When a class inherits from another class, it shares all the attributes and methods of the referenced class.
- ❑ The class that inherits from another class is called a subclass or child class, while the class being inherited from is called either a parent class, superclass, or base class.
- ❑ Inheritance simplifies code by allowing you to create new abstractions based on existing ones. You can build a hierarchy of related classes, reusing common functionality while adding specific details to each subclass.





OOP PRINCIPLES

2. Abstraction: Modifier the indicates missing components or incomplete implementation.

Example: When designing a car class, we abstract away the intricate details of the engine, transgression and other components, focusing only on what's relevant for our software.

- **Interface:** A completely abstract class that only contain abstract method and properties (with empty body).

3. Encapsulation: bundles data and methods into a single unit, hiding the internal details from the outside world. It ensures data protection, promotes modularity, enhances flexibility, and improves security. For instance, a BankAccount class encapsulates details like account holder's name and balance, providing methods like deposit() and withdraw().

4. Polymorphism: allows objects to take on multiple forms. It's achieved through method overloading, method overriding, interfaces, abstract classes, and dynamic binding. For example, a Shape class with a draw() method can be overridden in subclasses like Circle, Rectangle, and Triangle, demonstrating polymorphic behavior when invoking draw() on different objects.

Abstraction Question:

Write a C# program to model vehicles using object-oriented programming principles.
Implement the following requirements:

1. Define an abstract class Vehicle with the following properties and methods:
    - A private integer field numWheels to store the number of wheels.
    - An abstract method Display() that should be implemented by subclasses.
    - A public getter method GetNumWheels() to retrieve the number of wheels.

2. Implement a concrete subclass Bus inheriting from Vehicle with the following additional properties and methods:
    - A private integer field seatNums to store the number of seats.
    - A private getter method GetNumSeats() to retrieve the number of seats.
    - Override the Display() method to set the number of wheels to 4 and display it.

3. Write a Main method to test the implemented functionality:
    - Create an instance of the Bus class.
    - Call the Display() method to set the number of wheels and display it.
    - Call the GetNumWheels() method to retrieve the number of wheels and print it.

Ensure that the code compiles without errors and produces the expected output.

Interface Question:

Write a C# code snippet that models an inventory management system using object-oriented principles.
Your code should include the following components:

1. An interface named IPriceable with a method signature double GetPrice().
2. An abstract class named Inventory with the following abstract members:
    - Property: Name of type string.
    - Property: Quantity of type int.
    - Method: void Display().

3. A concrete class named Products that inherits from the Inventory class and implements the IPriceable interface. The Products class should have the following members:
    - Field: Price of type double.
    - Constructor: Accepting parameters for name, quantity, and price.
    - Implementation of the GetPrice() method to return the price of the product.
    - Implementation of the Name and Quantity properties inherited from the Inventory class.
    - Implementation of the Display() method to print the name, quantity, and price of the product.

Your code should also include a Main method to demonstrate the usage of the Products class by creating an instance of it and displaying its properties.

Ensure that your code is syntactically correct and adheres to the provided requirements.