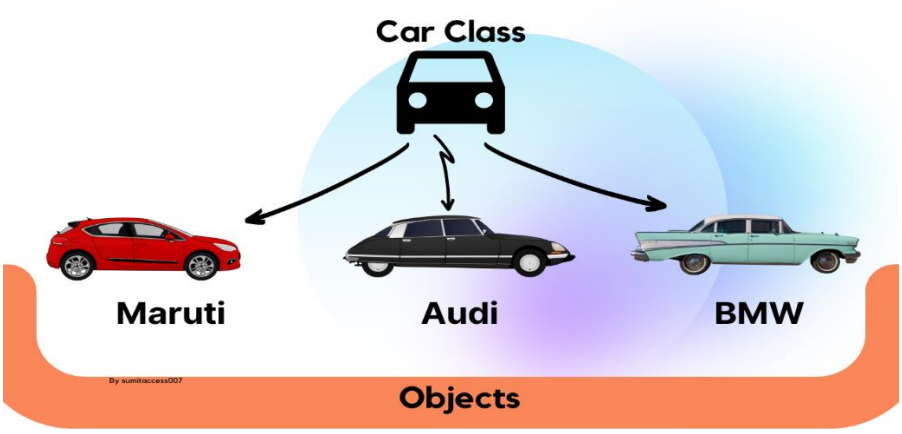
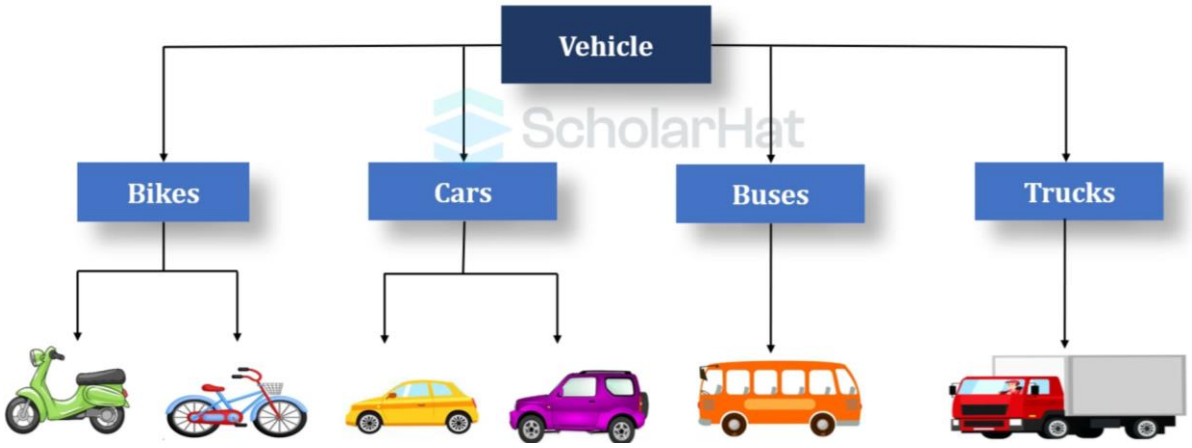
Object Oriented Programming

Object-Oriented Programming (OOP) is a programming paradigm that structures software design around **objects**, which represent real-world entities or concepts. These objects contain **attributes** (data, known as fields or properties) and **methods** (behaviors) that define what they can do.

Key Concepts of Object-Oriented Programming

1. **Class and Object**:
   - ✓ A **class** is like a blueprint or template that defines the properties and behaviors of an object.
   - ✓ An **object** is an instance of a class. Each object has its own set of data (attributes) and can perform tasks using its methods.





Benefits of Object-Oriented Programming:

1. **Modularity**: Objects are self-contained. Each class can be developed and tested independently.
2. **Reusability**: Once a class is written, it can be reused in other parts of the application or in other applications.
3. **Maintainability**: Encapsulation allows modification of one part of the system without affecting others.
4. **Scalability**: OOP makes it easier to build complex, large applications since the code can be organized into classes and objects.
5. **Flexibility**: Inheritance and polymorphism make it easier to introduce new functionality or change behaviors.

Create class and object :

```
class Car
{
  Public string color = "red";
  public string Model { get; set; }
  Public int maxSpeed = 200;
  Public int year { get; set; }
}
  static void Main(string[] args)
  {
    Car Ford = new Car();
    Ford.model = "Mustang";
    Ford.color = "red";
    Ford.year = 1969;
    Car Opel = new Car();
    Opel.model = "Astra";
    Opel.color = "white";
    Opel.year = 2005;
    Console.WriteLine(Ford.model);
    Console.WriteLine(Opel.model);
  }
```

Class Constructors : 1- Default Constructor

```
class Car
{
            public string model;  // Create a field
             public Car()
           {  model = "Mustang"; // Set the initial value for model }
}
  static void Main(string[] args)
  {
           Car Ford = new Car();  // Create an object of the Car Class (this will call the constructor)
            Console.WriteLine(Ford.model);  // Print the value of model : “Mustang”
  }
```

2- Parameterized Constructor

```
public class Car
{
   public string Model { get; set; }
   public int Speed { get; set; }

   // Parameterized constructor
   public Car(string model, int speed)
   {
      Model = model;
      Speed = speed;
      Console.WriteLine("Parameterized constructor called.");
   }
}
public class Program
{
   public static void Main()
   {
      Car myCar = new Car("Toyota", 100);  // Calls the parameterized constructor
      Console.WriteLine($"Model: {myCar.Model}, Speed: {myCar.Speed}"); // Output: Model: Toyota, Speed: 100
   }
}
```

3- Copy Constructor

```
public class Car
{
   public string Model { get; set; }
   public int Speed { get; set; }

   // Parameterized constructor
   public Car(string model, int speed)
   {
      Model = model;
      Speed = speed;
   }
```

```
    // Copy constructor
    public Car(Car existingCar)
    {
        Model = existingCar.Model;
        Speed = existingCar.Speed;
        Console.WriteLine("Copy constructor called.");
    }
}
public class Program
{
    public static void Main()
    {
        Car car1 = new Car("Honda", 120);
        Car car2 = new Car(car1);  // Calls the copy constructor

        Console.WriteLine($"car2 Model: {car2.Model}, Speed: {car2.Speed}");  // Output: Model: Honda, Speed: 120
    }
}
```

4- Static Constructor

```
public class Car
{
    // Static fields shared across all instances
    public static int MaxAllowedSpeed;
    public static string DefaultFuelType;

    // Instance fields for individual vehicle objects
    public string Model { get; set; }
    public int Speed { get; set; }
    public string FuelType { get; set; }
```

```
  // Static constructor to initialize configuration settings
   static Car()
   {
      MaxAllowedSpeed = 180;  // Maximum speed limit for all vehicles
      DefaultFuelType = "Petrol";  // Default fuel type for all vehicles
      Console.WriteLine("Static constructor called: MaxAllowedSpeed and DefaultFuelType initialized.");
   }

// Parameterized constructor to create vehicle objects
   public Car(string model, int speed, string fuelType = null)
   {
      Model = model;
      Speed = speed > MaxAllowedSpeed ? MaxAllowedSpeed : speed;  // Ensure the speed doesn't exceed the max allowed
      FuelType = fuelType ?? DefaultFuelType;  // Use default fuel type if none is provided
      Console.WriteLine($"{Model} vehicle created with {Speed} km/h speed and {FuelType} fuel type.");
   }
}
public class Program
{
   public static void Main()
   {
      // Display the configuration settings (invokes the static constructor if not already called)

      // Creating vehicle objects with default and custom settings
      Car Car1 = new Car("Toyota", 150);
      Car Car2 = new Car("Honda", 200, "Diesel");  // Speed will be adjusted to the max allowed

   }
}
```