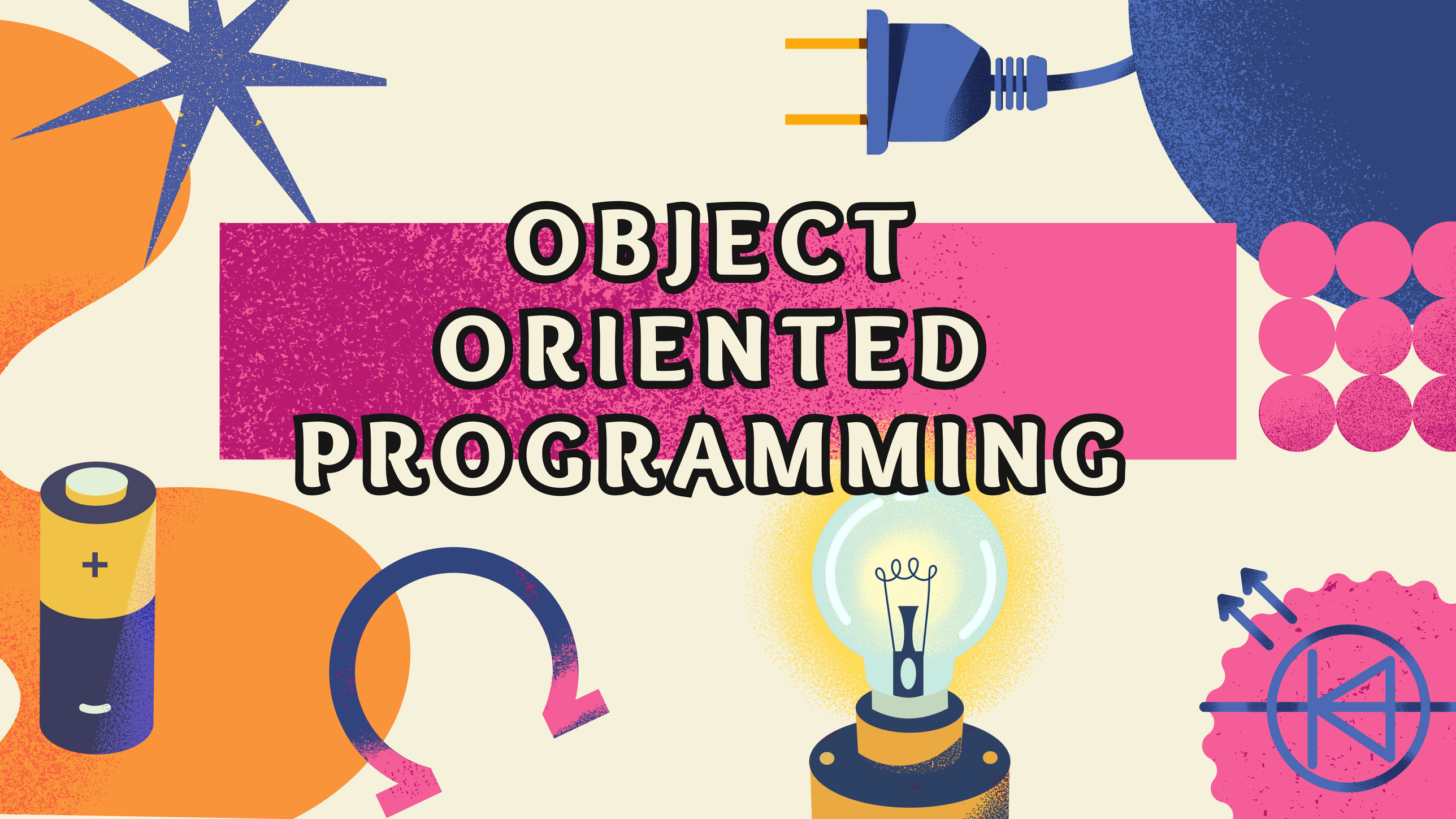
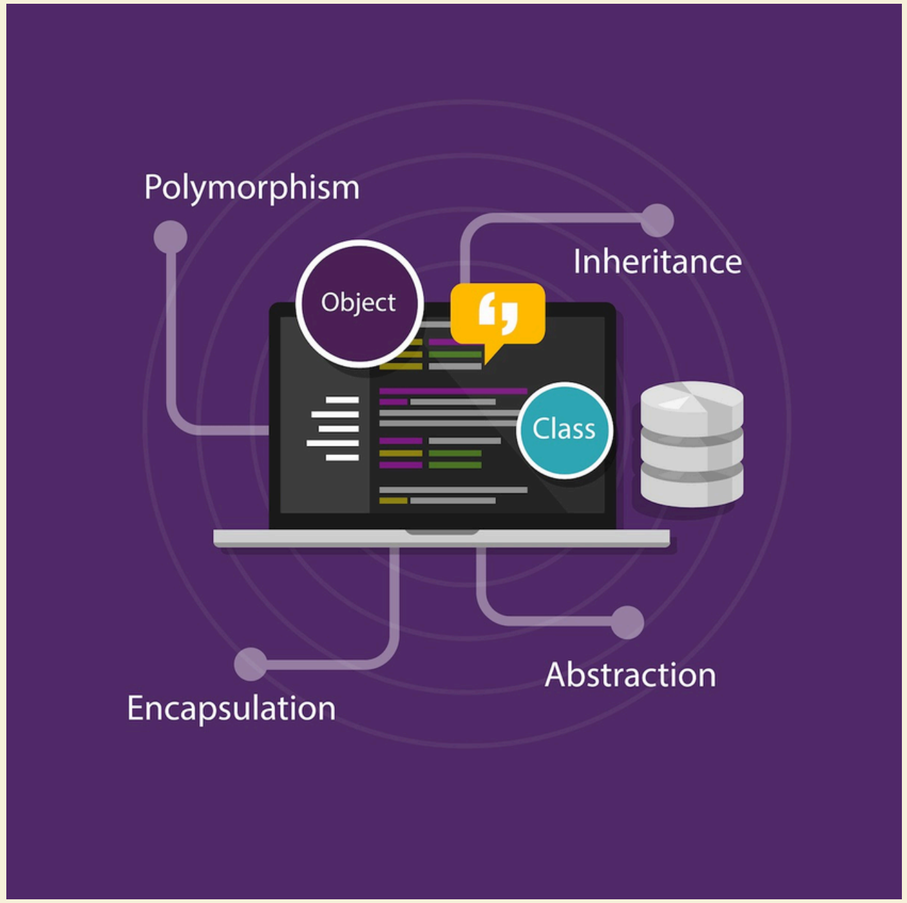

OBJECT
ORIENTED
PROGRAMMING

AGENDA

- Introduction
- Class
- Object
- Inheritance
- Abstraction
- Polymorphism
- Encapsulation

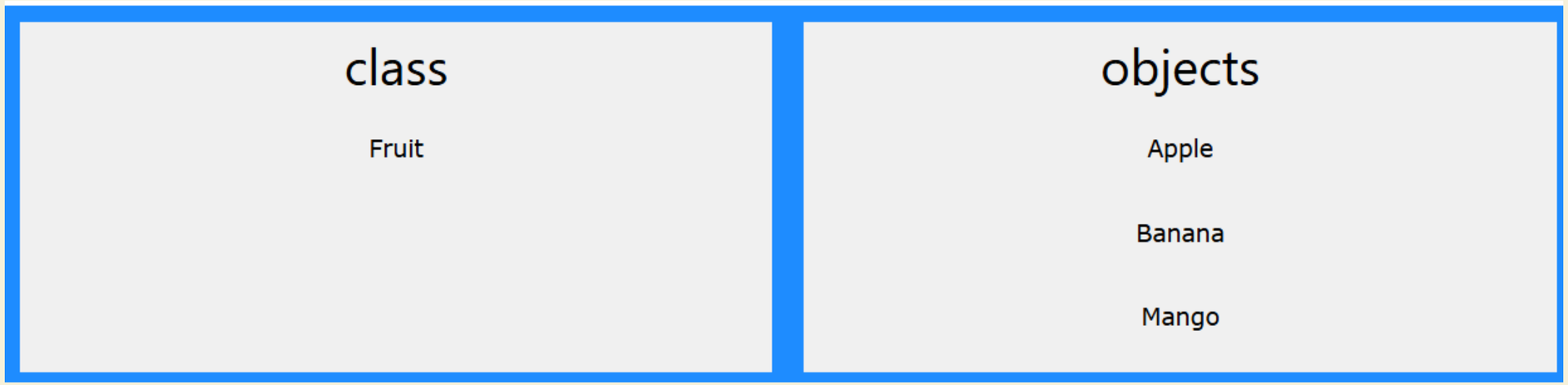
INTRODUCTION



Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the C# code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

CLASS

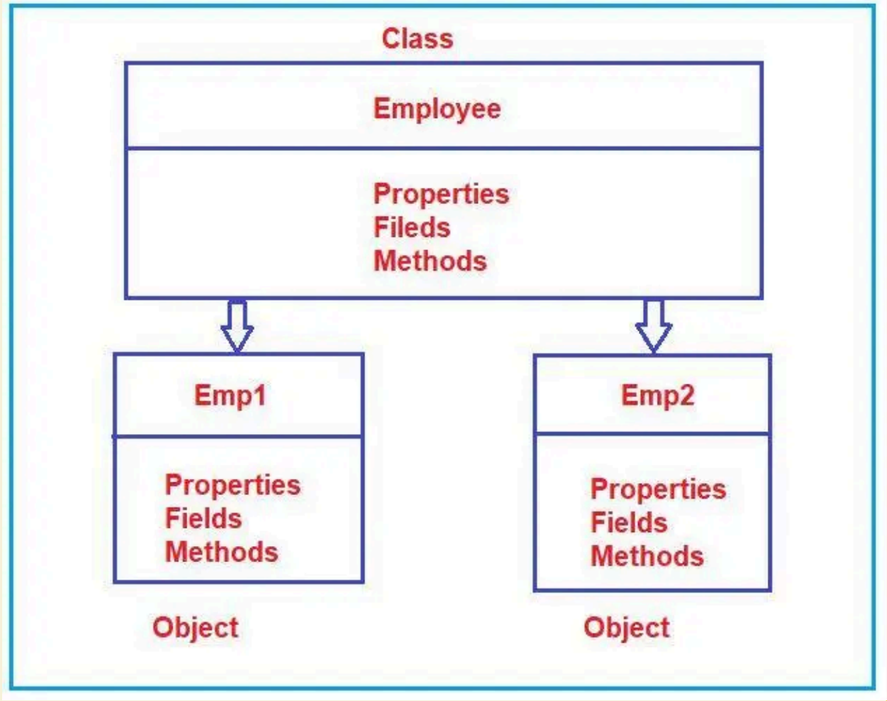


So, a class is a template for objects, and an object is an instance of a class.
When the individual objects are created, they inherit all the variables and methods from the class.

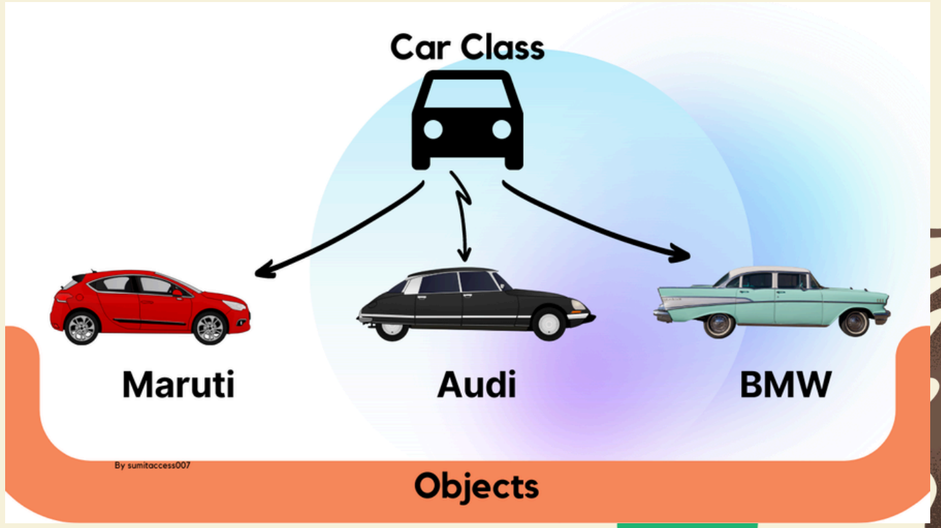
CREATING CLASS

To create a class, use the class keyword:

Create a class named "Car" with a variable color:
class Car{string color = "red";}



OBJECT



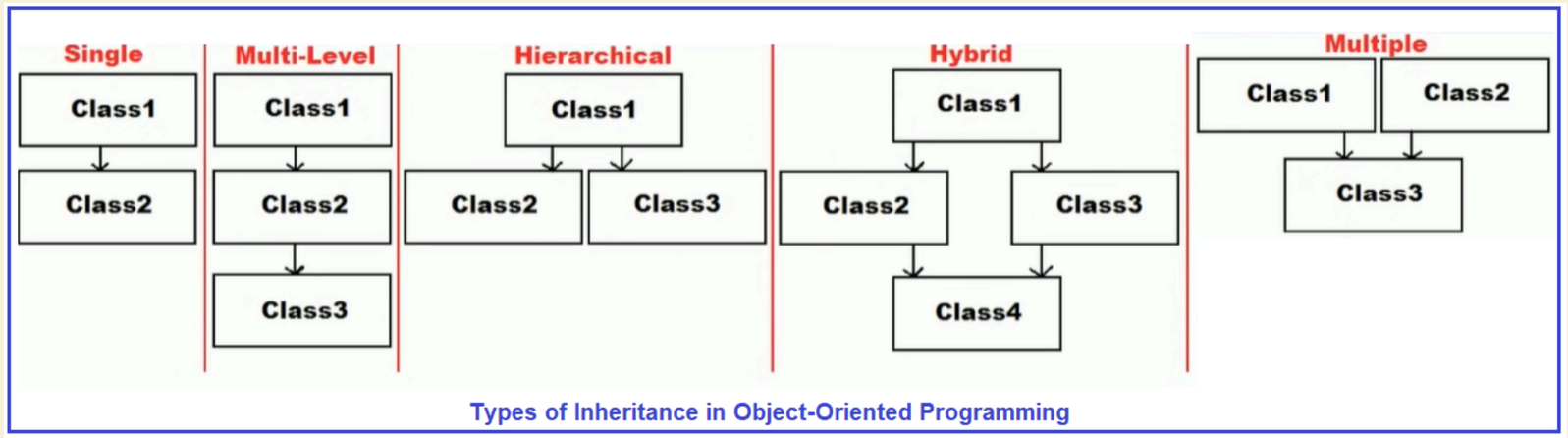

```
class Car{string color = "red";
 static void Main(string[] args){Car myObj =
                    new Car();
Console.WriteLine(myObj.color);}}
```

INHERITANCE

In C#, it is possible to inherit fields and methods from one class to another. We group the "inheritance concept" into two categories:

- Derived Class (child) – the class that inherits from another class
- Base Class (parent) – the class being inherited from

To inherit from a class, use the : symbol.

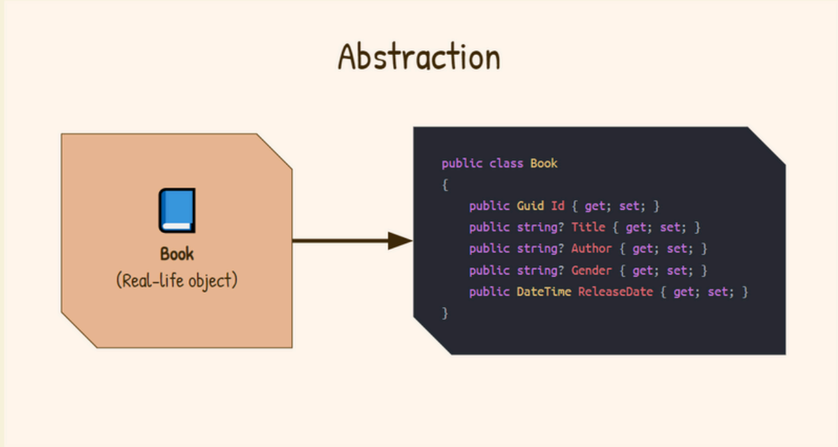


Types of Inheritance in Object-Oriented Programming

ABSTRACTION

Data abstraction is the process of hiding certain details and showing only essential information to the user.
Abstraction can be achieved with either abstract classes or interfaces
The abstract keyword is used for classes and methods.

```
abstract class Animal
{
public abstract void animalSound();
public void sleep()
{
Console.WriteLine("Zzz");
}
}
```



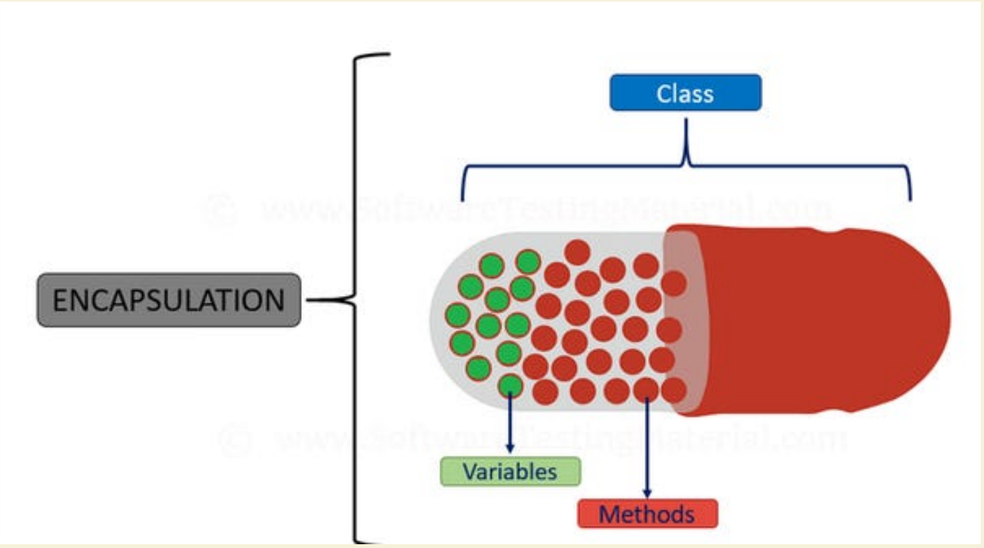
ENCAPSULATION

Encapsulation is the concept of wrapping data (fields) and methods (properties or functions) together in a single unit (class). It hides internal data and provides controlled access using public methods or properties.

```
class Person
{
    private string name;  // Private field

    public string Name    // Public property
    {
        get { return name; }
        set { name = value; }
    }
}

class Program
{
    static void Main()
    {
        Person p = new Person();
        p.Name = "Alice";  // Access via property
        Console.WriteLine(p.Name);
    }
}
```



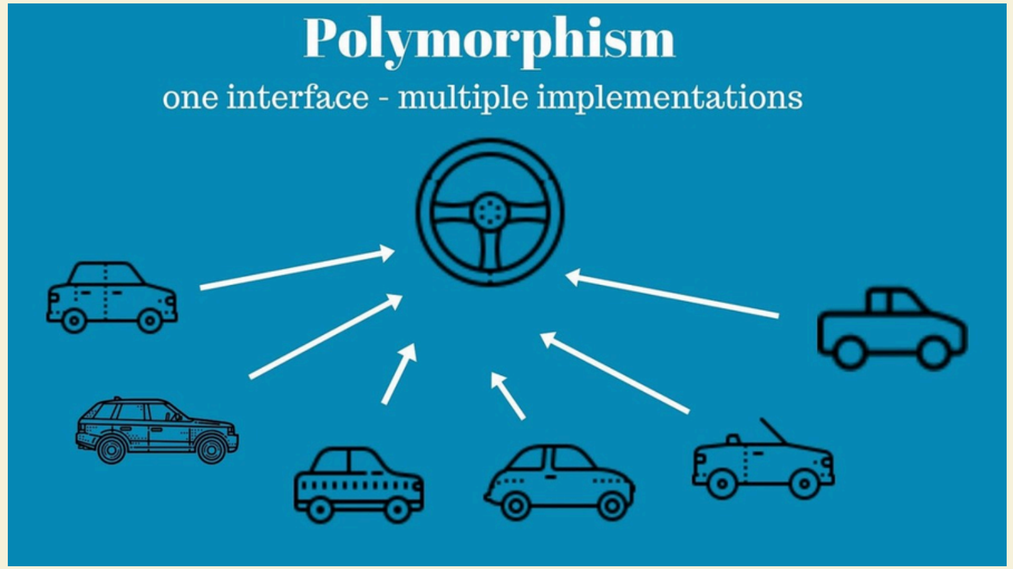
POLYMORPHISM

Polymorphism allows objects to take many forms and enables a single interface to represent different underlying types. It supports method overriding and overloading, making code more flexible and reusable.

```
class Animal
{
    public virtual void Speak() { Console.WriteLine("Animal speaks"); }
}

class Dog : Animal
{
    public override void Speak() { Console.WriteLine("Dog barks"); }
}

class Program
{
    static void Main()
    {
        Animal a = new Dog();
        a.Speak();  // Output: Dog barks
    }
}
```



THANK YOU