Object-Oriented Programming (OOP) in C#

1. What is Object-Oriented Programming (OOP)?

- OOP is a programming paradigm based on the concept of **"objects"** which contain **data (fields/properties)** and **behavior (methods)**.
- C# is a **fully object-oriented language**.

2. Core OOP Principles (Pillars) in C#:

OOP Concept	Description
Encapsulation	Wrapping data and methods into a single unit (class)
Inheritance	Creating new classes from existing classes
Polymorphism	Same method name behaves differently based on the object
Abstraction	Hiding internal details and showing only essential features

3. Class and Object in C#

Class: Blueprint or template for creating objects

Object: Instance of a class

```
class Car
{
    public string brand = "Toyota";
    public void Drive()
    {
        Console.WriteLine("Car is driving...");
    }
}

class Program
{
    static void Main()
    {
        Car myCar = new Car();    // Creating Object
        Console.WriteLine(myCar.brand);
        myCar.Drive();
    }
}
```

4. Constructors

- Special method used to initialize objects.

```
class Person
{
    public string Name;
    public Person(string name)  // Constructor
    {
        Name = name;
    }
}
```

---

5. Encapsulation

- Using **access modifiers** (`public`, `private`, `protected`) to control access to class members.
- Example with **properties**:

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

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

---

6. Inheritance

- Child class inherits methods and fields from the parent class.

```
class Employee
{
    public string Name;
    public double Salary;

    public void Display()
    {
        Console.WriteLine($"Name: {Name}, Salary: {Salary}");
    }
}

class Manager : Employee
{
    public string Department;
```

```
    public void DisplayManager()
    {
        Display();
        Console.WriteLine($"Department: {Department}");
    }
}
```

---

7. Polymorphism

- **Compile-time Polymorphism:** Method Overloading
- **Run-time Polymorphism:** Method Overriding

Method Overloading (Same method, different parameters)

```
// Using conventional style
class Calculator
{
    public int Add(int a, int b)
    {
        return a + b;
    }
    public double Add(double a, double b)
    {
        return a + b;
    }
}


// Using Expression-bodied members in C# were introduced in C# 6.0
class Calculator
{
    public int Add(int a, int b) => a + b;
    public double Add(double a, double b) => a + b;
}
```

Method Overriding (Using virtual and override keywords)

```
class Animal
{
    public virtual void Sound()
    {
        Console.WriteLine("Animal makes a sound");
    }
}

class Dog : Animal
{
    public override void Sound()
```

```
    {
        Console.WriteLine("Dog barks");
    }
}
```

---

8. Abstraction

- **Abstract classes** and **interfaces** provide abstraction.

Abstract Class Example:

```
abstract class Shape
{
    public abstract void Draw();
}

class Circle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing Circle");
    }
}
```

Interface Example:

```
interface IVehicle
{
    void Drive();
}

class Car : IVehicle
{
    public void Drive()
    {
        Console.WriteLine("Driving a car...");
    }
}
```

---

9. Destructor

- Cleans up when an object is destroyed (rarely used in C#).

```
~Person()
{
    Console.WriteLine("Destructor called");
}
```

---

10. Access Modifiers Summary

Modifier	Description
`public`	Accessible from anywhere
`private`	Accessible only within the class
`protected`	Accessible within the class and derived classes
`internal`	Accessible within the same assembly
`protected internal`	Accessible within assembly or derived class

---

11. Benefits of OOP in C#:

- **Code Reusability** through inheritance
- **Data Security** via encapsulation
- **Easy Maintenance** and modification
- **Scalability** for large applications
- **Real-world Modeling**

---

12. Practice Questions:

1. Create a `Vehicle` class and inherit `Car` and `Bike` from it. Add methods and demonstrate polymorphism.
2. Create an abstract class `Shape` and two derived classes `Rectangle` and `Circle` that implement `Draw()` method.
3. Design a class `Student` with properties (Name, Age), a constructor, and a method to display details.
