OOP Concepts

1. Class and Object

- **Class**: A blueprint for creating objects.
- **Object**: An instance of a class.

Example:

```
public class Car
{
   public string Color;
   public void Drive()
   {
      Console.WriteLine("Car is driving");
   }
  }
class Program
{
   static void Main()
   {
      Car myCar = new Car();
      myCar.Color = "Red";
      myCar.Drive(); // Output: Car is driving
   }
}
```

---

2. Encapsulation

- Wrapping data and methods into a single unit (class).
- Use of **access modifiers** like `private`, `public`, `protected`.

Example:

```
public class BankAccount
{
    private double balance;

    public void Deposit(double amount)
    {
        if (amount > 0) balance += amount;
    }

    public double GetBalance()
    {
        return balance;
    }
}
```

---

3. Inheritance

- Allows a class to inherit members (fields, methods) from another class.
- `:` symbol is used to inherit.

Example:

```
public class Animal
{
    public void Eat()
    {
        Console.WriteLine("Eating...");
    }
}

public class Dog : Animal
{
    public void Bark()
    {
        Console.WriteLine("Barking...");
    }
}
```

---

4. Polymorphism

- **Compile-time (Method Overloading)**: Same method name with different parameters.
- **Run-time (Method Overriding)**: Derived class provides specific implementation of a method.

Method Overloading Example:

```
public class MathOperations
{
    public int Add(int a, int b) => a + b;
    public double Add(double a, double b) => a + b;
}
```

Method Overriding Example:

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

public class Cat : Animal
{
    public override void MakeSound()
    {
        Console.WriteLine("Meow");
    }
}
```

OOP Concepts

5. Abstraction

- Hides internal implementation and shows only the necessary details.
- Achieved using **abstract classes** or **interfaces**.

Abstract Class Example:

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

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

```
using System;


namespace OOPConceptsDemo
{
  // ✅ Encapsulation:
  // The class hides the data (balance) and exposes access only through methods
  public class BankAccount
  {
    private decimal balance;  // Private field, can't be accessed directly


    // Public method to modify the balance
    public void Deposit(decimal amount)
    {
      if (amount > 0)
        balance += amount;
    }
```

OOP Concepts

```
    // Public method to read the balance
    public decimal GetBalance()
    {
        return balance;
    }
}

// ✅ Inheritance:
// Base class
public class Animal
{
    // Virtual method allows derived classes to override
    public virtual void MakeSound()
    {
        Console.WriteLine("Animal makes a sound");
    }
}

// ✅ Polymorphism (Run-time): Method overriding
// Derived class inherits from Animal
public class Dog : Animal
{
    // Overrides the base class method
    public override void MakeSound()
    {
        Console.WriteLine("Dog barks");
    }
}
```

OOP Concepts

```
// ✅ Abstraction:
// Abstract class cannot be instantiated directly
public abstract class Shape
{
   // Abstract method must be implemented by derived class
   public abstract void Draw();
}

// Derived class that implements the abstract method
public class Circle : Shape
{
   public override void Draw()
   {
      Console.WriteLine("Drawing Circle");
   }
}

// ✅ Class & Object demonstration
class Program
{
   static void Main(string[] args)
   {
      Console.WriteLine("=== OOP Concepts in C# ===\n");

      // 1. Class & Object + Encapsulation
      Console.WriteLine("1. Class & Object + Encapsulation:");
      BankAccount account = new BankAccount(); // Object created
      account.Deposit(1000);                   // Calling method
      Console.WriteLine($"Balance: {account.GetBalance()}"); // Access via method
```

```
        // 2. Inheritance + Polymorphism
        Console.WriteLine("\n2. Inheritance + Polymorphism:");
        Animal genericAnimal = new Animal();
        Animal dog = new Dog(); // Polymorphism - base type holding derived object

        genericAnimal.MakeSound(); // Output: Animal makes a sound
        dog.MakeSound();           // Output: Dog barks (Overridden method)

        // 3. Abstraction
        Console.WriteLine("\n3. Abstraction:");
        Shape circle = new Circle(); // Using abstract class reference
        circle.Draw();             // Output: Drawing Circle

        Console.WriteLine("\n=== End of OOP Demonstration ===");
    }
  }
}
```