Object-Oriented Programming (OOP) in C#

1?? ABSTRACTION

Definition: Abstraction hides the implementation details and only exposes essential features.

How to Achieve in C#?

- Using abstract classes (`abstract` keyword)
- Using interfaces (`interface` keyword)

Example:

```
abstract class Vehicle {
    public string Brand { get; set; }
    public abstract void Start();
}

class Car : Vehicle {
    public override void Start() {
        Console.WriteLine($"{Brand} is starting with a key.");
    }
}
```

? Abstract methods must be overridden in derived classes.

? Cannot create an instance of an abstract class.

2?? POLYMORPHISM

Compile-Time Polymorphism (Method Overloading)

Definition: Methods with the same name but different parameters.

Example:

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

Object-Oriented Programming (OOP) in C#

? Resolved at compile-time.

? Improves code readability.

Run-Time Polymorphism (Method Overriding)

Definition: A derived class provides a new implementation of a method defined in a base class.

Example:

```
class Vehicle {
    public virtual void Start() {
        Console.WriteLine("Vehicle is starting.");
    }
}

class Car : Vehicle {
    public override void Start() {
        Console.WriteLine("Car is starting with a key.");
    }
}
```

? Uses virtual and override keywords.

? Resolved at run-time using base class reference.

3?? ENCAPSULATION

Definition: Restricting direct access to class data and providing controlled access.

Example:

```
class Person {
    private string name;

    public string Name {
        get { return name; }
        set { if (!string.IsNullOrEmpty(value)) name = value; }
    }
}
```

? Protects data integrity.
? Uses getters and setters for controlled access.

4?? INHERITANCE
Definition: A class (child) derives from another class (parent) to reuse its functionality.

Example:

```
class Animal {
    public void Eat() => Console.WriteLine("This animal eats food.");
}

class Dog : Animal {
    public void Bark() => Console.WriteLine("Dog is barking.");
}
```

? Promotes code reusability.
? Allows method overriding for polymorphism.

5?? COUPLING (Tight vs. Loose Coupling)

Tight Coupling (Bad Example):
Definition: One class depends directly on another class.

```
class PetrolEngine {
    public void Start() => Console.WriteLine("Petrol Engine Started");
}

class Car {
    private PetrolEngine engine = new PetrolEngine(); // Tight Coupling
    public void Start() => engine.Start();
}
```

? Hard to modify or replace dependencies.

Loose Coupling (Good Example):
Definition: Classes interact via interfaces instead of direct dependencies.

```
interface IEngine {
    void Start();
}

class PetrolEngine : IEngine {
    public void Start() => Console.WriteLine("Petrol Engine Started");
}

class Car {
    private IEngine engine;
    public Car(IEngine engine) { this.engine = engine; }
    public void Start() => engine.Start();
}
```

? Improves flexibility and testability.
? Uses dependency injection.

6?? CONCRETE METHODS
Definition: A method that has a complete implementation.

Example:

```
class Calculator {
    public int Add(int a, int b) {
        return a + b;
    }
}
```

? Concrete methods CAN have parameters.
? They provide complete functionality.

7?? MULTIPLE INHERITANCE IN C#
C# does not support multiple inheritance directly but allows it using interfaces.

Example Using Interfaces:

```
interface IAnimal {
    void MakeSound();
}

interface IWalker {
    void Walk();
}

class Dog : IAnimal, IWalker {
    public void MakeSound() => Console.WriteLine("Bark");
    public void Walk() => Console.WriteLine("Dog is walking");
}
```

? Achieves multiple inheritance behavior using interfaces.
? Avoids the diamond problem.

? SUMMARY TABLE OF OOP CONCEPTS

Concept	Definition	Key Features
Abstraction	Hides implementation details	Uses abstract classes & interfaces
Polymorphism	Same method behaves differently	Uses method overloading & overriding

Object-Oriented Programming (OOP) in C#

Encapsulation	Protects data with restricted access	Uses private fields & public properties
Inheritance	One class inherits from another	Uses base and derived classes
Coupling	Dependency between classes	Loose coupling is better than tight coupling
Concrete Method	Fully implemented methods	Can have parameters & return values
Multiple Inheritance	One class inherits multiple classes	Achieved using interfaces

----

? KEY TAKEAWAYS

? OOP improves code reusability, maintainability, and flexibility.

? Encapsulation protects data, abstraction hides details, inheritance promotes reuse, and polymorphism enables flexibility.

? Loose coupling (using interfaces) is better than tight coupling.

? C# does not support multiple inheritance, but interfaces provide a workaround.