Object-Oriented Programming (OOP) in C# – Full Guide

A complete, beginner-friendly, easy-to-understand OOP learning document for C#.

📌Table of Contents

1. Introduction to OOP
2. Principles of OOP
3. Encapsulation
4. Abstraction
5. Inheritance
6. Polymorphism
7. Classes & Objects in C#
8. Constructors & Destructors
9. Access Modifiers
10. Properties & Getters/Setters
11. Static vs Instance Members
12. Inheritance Types in C#
13. Method Overloading & Overriding
14. Interfaces
15. Abstract Classes
16. Sealed Classes & Methods
17. OOP Real-World Example
18. Practice Questions

---

1. Introduction to OOP

OOP stands for **Object-Oriented Programming**. It organizes code into objects that represent real-world things.

Why OOP?

- Easy to maintain
- Reusable code
- Scalable structure
- Secure

---

2. OOP Principles (The Big Four)

🕰️Encapsulation

Wrapping data (variables) and behavior (methods) in a single unit (class).

Example:

```
class Student
{
    private int age; // hidden

    public void SetAge(int a)
    {
        age = a;
    }

    public int GetAge()
    {
        return age;
    }
}
```

🕰️Abstraction

Showing only **essential** information and hiding details.

Example:

```
abstract class Animal
{
    public abstract void MakeSound();
}
```

🕰️Inheritance

One class can inherit features of another.

Example:

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

class Dog : Animal
{
}
```

🕰️Polymorphism

Same function, different behavior.

Types:

- **Compile-time:** Method overloading
- **Run-time:** Method overriding

---

3. Classes & Objects

Class Example:

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

Object Example:

```
Car c1 = new Car();
c1.brand = "BMW";
c1.Drive();
```

---

4. Constructors & Destructors

Constructor

Automatically called when object is created.

```
class Person
{
    public Person()
    {
        Console.WriteLine("Object Created");
    }
}
```

Destructor

Called when object is destroyed.

```
~Person()
{
    Console.WriteLine("Object Destroyed");
}
```

---

5. Access Modifiers

Modifier	Description
public	Accessible everywhere
private	Only inside class
protected	Class + subclasses
internal	Same assembly
protected internal	Protected + internal
private protected	Private + protected

---

6. Properties (Modern Getter/Setter)

```
class Student
{
    public string Name { get; set; }
    public int Age { get; private set; }
}
```

---

7. Static vs Instance

Static: Belongs to class.

Instance: Belongs to object.

```
class Test
{
    public static int counter = 0;
}
```

---

8. Types of Inheritance in C#

- Single
- Multilevel
- Hierarchical
- Multiple (via interfaces)
- Hybrid

---

9. Method Overloading & Overriding

Overloading (same name, different parameters)

```
void Add(int a, int b)
void Add(double a, double b)
```

Overriding (parent method changed by child)

```
class A
{
    public virtual void Show() {}
}
class B : A
{
    public override void Show() {}
}
```

---

10. Interfaces

Used to achieve **multiple inheritance**.

```
interface IWalk
{
    void Walk();
}
```

---

11. Abstract Classes

Contains **abstract** + **normal** methods.

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

---

12. Sealed Classes / Methods

Sealed Class: Cannot be inherited.

```
sealed class FinalClass {}
```

Sealed Method: Cannot be overridden.

```
sealed override void Display(){}
```

---

13. Real-World Example (Full Program)

```
class Vehicle
{
    public virtual void Start() => Console.WriteLine("Vehicle Started");
}

class Car : Vehicle
{
    public override void Start() => Console.WriteLine("Car Started");
}

class Program
{
    static void Main()
    {
        Vehicle v = new Car();
        v.Start();
    }
}
```

---

14. Practice Questions

1. Create a class `Laptop` with properties and methods.
2. Write a program to demonstrate inheritance.
3. Create an abstract class `Shape` with `Area()`.
4. Show difference between **interface vs abstract class**.
5. Create a polymorphism example with overriding.

👙 15. Deeper OOP Theory (Advanced Concepts)**

🕰️Composition vs Inheritance

- **Inheritance:** “IS-A” relationship.
- **Composition:** “HAS-A” relationship.

Example:

```
class Engine {}
class Car
{
    private Engine engine = new Engine(); // Car HAS-A Engine
}
```

When to use what? - Use **inheritance** for shared behavior. - Use **composition** for flexible structures.

🕰️SOLID Principles

S – Single Responsibility Principle

Every class should do **only one** thing.

O – Open/Closed Principle

Classes should be **open for extension** but **closed for modification**.

L – Liskov Substitution Principle

Child classes should replace parent classes **without breaking code**.

I – Interface Segregation Principle

Don’t create fat interfaces.

D – Dependency Inversion Principle

Depend on **abstractions**, not concrete classes.

🕰️Virtual, Override, New Keywords

Keyword	Usage
virtual	Makes a method overridable
override	Modifies parent implementation
new	Hides parent method

Example:

```
class A { public void Show(){} }
class B : A { public new void Show(){} }
```

---

16. OOP Interview Questions with Answers

Q1: What is OOP?

A: OOP is a programming model based on objects that contain data and behavior.

Q2: What is Encapsulation?

A: Binding data + methods together and hiding internal details.

Example: Using private fields with public properties.

Q3: Difference between Abstraction and Encapsulation?

Abstraction	Encapsulation
Hides complexity	Hides data
Focuses on *what*	Focuses on *how*

Q4: What is Polymorphism?

A: Ability of a method to behave differently based on context.

- Compile-time → Overloading

- Runtime → Overriding

Q5: What is the difference between interface and abstract class?

Interface	Abstract Class
No implementation	Partial implementation
Multiple inheritance possible	Only single inheritance
All methods are abstract	Can have normal + abstract methods

Q6: What is Method Overloading?

A: Multiple methods with same name but different parameters.

Q7: What is Method Overriding?

A: Redefining parent method in child using `override`.

Q8: What is a Constructor?

A: A special method that runs automatically when an object is created.

Q9: What is Sealed Class?

A: A class that cannot be inherited.

Q10: What is Dependency Injection?

A: Providing an object its required dependencies instead of creating them inside.

Q11: What is the difference between Composition & Aggregation?

- **Composition:** Strong relationship (car → engine). Lifetime same.
- **Aggregation:** Weak (school → students). Independent lifetime.

🎉End of Document** If you want, I can also generate: - PDF version - More examples - Exercises with solutions - A full C# OOP project structure