

## Day 3 – C# & OOP Basics

Date: 11-10-2025

**Topic:** Inheritance, Polymorphism, Abstraction, Encapsulation, Access Modifiers, Constructors, Destructors, Static & Sealed Classes, Namespaces

### 1. WHAT IS OOP (OBJECT-ORIENTED PROGRAMMING)?

Object-Oriented Programming is a paradigm that organizes software design around **objects**, rather than functions and logic.

Objects are instances of classes that contain **data (fields)** and **behaviors (methods)**.

C# is a fully OOP-based language and supports all four main principles:

Principle	Description
<b>Encapsulation</b>	Hiding internal details of objects and exposing only necessary parts.
<b>Abstraction</b>	Showing only essential features and hiding background details.
<b>Inheritance</b>	Acquiring the properties and behavior of another class.
<b>Polymorphism</b>	One name, many forms — allows method overloading and overriding.

### 2. INHERITANCE

#### Definition:

Inheritance is a mechanism that allows one class (child) to acquire the properties and behaviors of another class (parent).

#### Syntax:

```
class Parent
{
    // base class code
}

class Child : Parent
{
    // derived class code
}
```

#### Types of Inheritance in C#:

Type	Example	Description
<b>Single</b>	A → B	One parent, one child

<b>Multilevel</b>	A → B → C	Chain of inheritance
<b>Hierarchical</b>	A → B, A → C	One parent, multiple children
<b>Multiple</b>	Not supported directly (use interfaces)	

#### Keywords:

- `base` → Used to call parent class constructors or methods.
- `protected` → Members accessible in derived classes.

#### Example:

```
class Employee
{
    public void Work() => Console.WriteLine("Working...");
}

class Developer : Employee
{
    public void Code() => Console.WriteLine("Coding in C#...");
}
```

## 3. POLYMORPHISM

#### Definition:

Polymorphism means "*many forms*". It allows methods to have the same name but behave differently depending on the context.

#### Types:

Type	Description	Keyword
<b>Compile-Time (Static)</b>	Method Overloading	Same method name, different parameters
<b>Runtime (Dynamic)</b>	Method Overriding	Redefined method in child class

#### Example 1 – Method Overloading:

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

```
}
```

#### **Example 2 – Method Overriding:**

```
class Shape  
{  
    public virtual void Draw() => Console.WriteLine("Drawing a shape...");  
  
}  
  
class Circle : Shape  
{  
    public override void Draw() => Console.WriteLine("Drawing a circle...");  
  
}
```

#### **Keyword Summary:**

- **virtual** → Marks method in base class that can be overridden.
- **override** → Redefines base class method in child class.
- **new** → Hides base class method without overriding it.

## **4. ABSTRACTION**

#### **Definition:**

Abstraction is the process of hiding internal implementation details and showing only the necessary features.

In C#, abstraction can be achieved using:

- **Abstract classes**
- **Interfaces**

#### **Abstract Class Example:**

```
abstract class Vehicle  
{  
    public abstract void Start(); // abstract method  
  
    public void Stop() => Console.WriteLine("Vehicle stopped");  
  
}  
  
class Car : Vehicle
```

```
{
    public override void Start() => Console.WriteLine("Car started ");
}
```

### **Interface Example:**

```
interface IShape
{
    double GetArea();
}

class Circle : IShape
{
    public double radius = 5;
    public double GetArea() => Math.PI * radius * radius;
}
```

### **Key Points:**

Aspect	Abstract Class	Interface
Implementation	Partial	None
Multiple Inheritance	Not	yes
Access Modifiers	Allowed	Not allowed in methods

## **5. ENCAPSULATION**

### **Definition:**

Encapsulation is the process of **binding data and methods together** and protecting data from outside access.

### **Example:**

```
class BankAccount
{
    private double balance;

    public void Deposit(double amount)
```

```

    if (amount > 0) balance += amount;
}

public double GetBalance() => balance;
}

```

### **Why Encapsulation?**

- Prevents unauthorized data access.
- Makes code maintainable and secure.
- Encourages modular structure.

## **6. ACCESS MODIFIERS**

Access modifiers define the **scope** or **visibility** of members.

Modifier	Description	Scope
<b>public</b>	Accessible anywhere	Global
<b>private</b>	Accessible only within class	Class
<b>protected</b>	Accessible in derived classes	Class + Child
<b>internal</b>	Accessible within same assembly	Project-level
<b>protected internal</b>	Protected + Internal	Class + Assembly

## **7. CONSTRUCTORS AND DESTRUCTORS**

### **Constructor**

- Used to initialize objects.
- Automatically called when object is created.
- Has **same name** as class and **no return type**.

### **Types:**

1. **Default Constructor** – No parameters.
2. **Parameterized Constructor** – With parameters.
3. **Static Constructor** – Used to initialize static data.
4. **Copy Constructor** – Copies data from one object to another.

**Example:**

```
class Student

{
    public string Name;
    public int Age;

    public Student(string n, int a)
    {
        Name = n; Age = a;
    }

    public void Display() => Console.WriteLine($"{{Name}}, {{Age}}");
}
```

**Destructor**

- Used to clean up resources.
- Called automatically by Garbage Collector.

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

## 8. STATIC CLASSES AND MEMBERS

**Static Class:**

A class declared with the keyword static cannot be instantiated and contains only static members.

**Example:**

```
static class MathHelper
{
    public static int Square(int x) => x * x;
}
```

### **Notes:**

- Access directly using `ClassName.MethodName`.
- Commonly used for utility/helper functions.

## **9. SEALED CLASS**

### **Definition:**

A **sealed** class cannot be inherited.

Used when you want to prevent further derivation for security or performance reasons.

### **Example:**

```
sealed class Logger
{
    public void Log(string msg) => Console.WriteLine("Log: " + msg);
}
```

## **10. NAMESPACES**

### **Definition:**

A **namespace** is a container that organizes classes and avoids name conflicts.

### **Example:**

```
namespace DhruvTraining.Day3
{
    class Demo
    {
        public void Show() => Console.WriteLine("Inside DhruvTraining namespace");
    }
}
```

Use `using DhruvTraining.Day3;` to access it in another file.

## **11. GARBAGE COLLECTOR (CLR CONCEPT RECAP)**

- CLR (Common Language Runtime) manages memory automatically.
- **Garbage Collector (GC)** removes unreferenced objects.

- You can force GC using:
- `GC.Collect();`
- `GC.WaitForPendingFinalizers();`
- But normally, it's handled automatically by the runtime.

## SUMMARY TABLE

Concept	Keyword	Example
Inheritance	:	<code>class B : A { }</code>
Polymorphism	<code>virtual, override</code>	Method overriding
Abstraction	<code>abstract, interface</code>	Abstract classes, interfaces
Encapsulation	<code>private, public</code>	Data hiding
Static Class	<code>static</code>	Utility methods
Sealed Class	<code>sealed</code>	Prevent inheritance
Constructor	same as class name	Initialization
Destructor	<code>~ClassName()</code>	Cleanup
Namespace	<code>namespace</code>	Logical grouping

# Snapshots:

## Inheritance

A screenshot of Visual Studio Code showing a C# file named Program.cs. The code defines a base class Employee and a derived class Developer. The Employee class has a Name property and a Work() method. The Developer class inherits from Employee and adds a Language property and a Code() method. The code is annotated with several CS8618 warnings about non-nullability of fields.

```
using System;
class Employee // base class
{
    public string Name;
    public void Work()
    {
        Console.WriteLine($"{Name} is working.");
    }
}
class Developer : Employee // derived class
{
    public string Language;
    public void Code()
    {
        Console.WriteLine($"{Name} codes in {Language}");
    }
}
```

The terminal shows the output of running the application: "Udaya is working..." and "Udaya codes in C#".

## Polymorphism (Overloading + Overriding)

A screenshot of Visual Studio Code showing a C# file named Program.cs. The code demonstrates compile-time polymorphism through overloading (multiple Add methods) and run-time polymorphism through overriding (Circle and Square classes both implement the Draw() method). The application outputs "Drawing a circle" and "Drawing a square".

```
class Calculator // Compile-time polymorphism (overloading)
{
    public int Add(int a, int b) => a + b;
    public double Add(double a, double b) => a + b;
}

class Shape // Run-time polymorphism (overriding)
{
    public virtual void Draw() => Console.WriteLine("Drawing a shape...");
}

class Circle : Shape
{
    public override void Draw() => Console.WriteLine("Drawing a circle");
}

class Square : Shape
```

The terminal shows the output of running the application: "Drawing a circle" and "Drawing a square".

## Abstraction

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like `Program.cs`, `Day3Programs.csproj`, and `Day3Programs.sln`.
- Editor:** Displays the code for `Program.cs` containing an abstract class `Vehicle` and a concrete class `Car` that implements it.
- Terminal:** Shows the command `dotnet run` being executed, outputting "Car started".
- PowerShell Taskbar:** A floating window lists three PowerShell tabs.
- AI Integration:** A sidebar on the right says "Ask about your code" with a "GPT..." button.

## Encapsulation

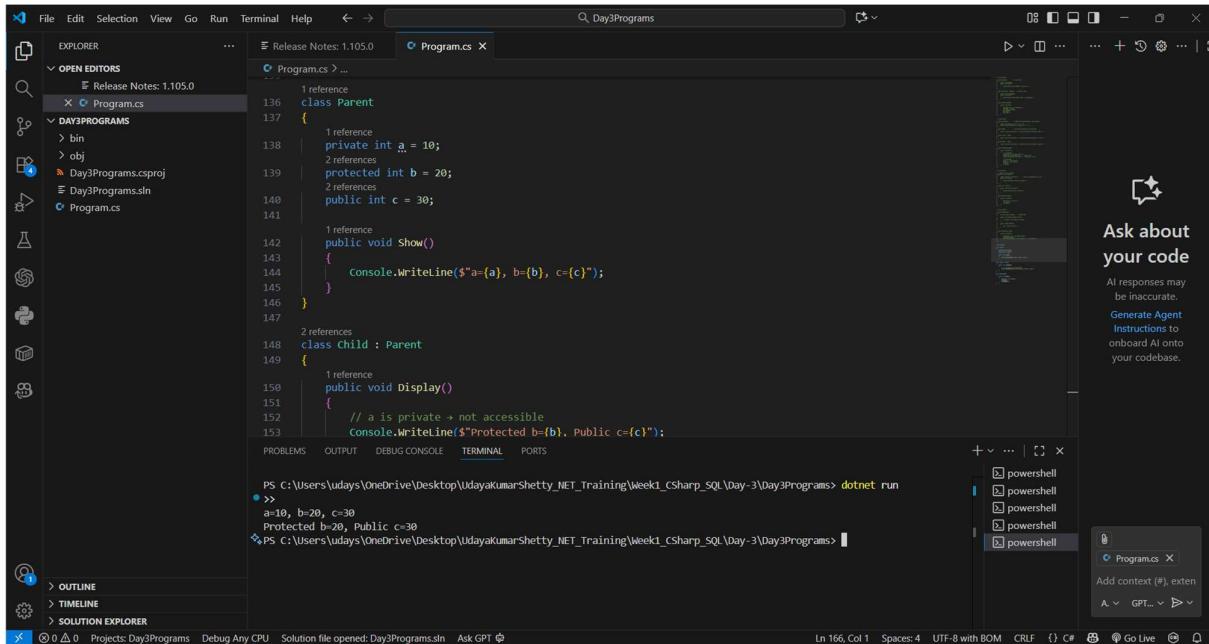
The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure with files like `Program.cs`, `Day3Programs.csproj`, and `Day3Programs.sln`.
- Code Editor:** Displays the `Program.cs` file content:

```
1 Release Notes: 1.105.0
2 Program.cs > ...
3 107
4 2 references
5 108 CLASS BankAccount
6 109 {
7     2 references
8     109 private double balance; // hidden data
9
10    1 reference
11    110 public void Deposit(double amount)
12    {
13        111 if (amount > 0) balance += amount;
14    }
15
16    1 reference
17    117 public double Balance
18    {
19        118 get { return balance; }
20    }
21
22    0 references
23    123 class EncapsulationDemo
24    {
25        0 references
26        124 static void Main()
27    }
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
```
- Terminal:** Shows a PowerShell session output:

```
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_.NET_Training\Week1_Csharp_SQL\Day-3\Day3Programs> dotnet run
>>>
Current balance: 5000
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_.NET_Training\Week1_Csharp_SQL\Day-3\Day3Programs>
```
- Right Panel:** Includes an "Ask about your code" button, AI response status ("AI responses may be inaccurate"), and a "Generate Agent Instructions to onboard AI onto your codebase" button.

## Access Modifiers Example



A screenshot of the Visual Studio Code interface. The title bar says "Day3Programs". The left sidebar shows the "EXPLORER" view with a "Program.cs" file open, and the "SOLUTION EXPLORER" view. The main editor area contains the following C# code:

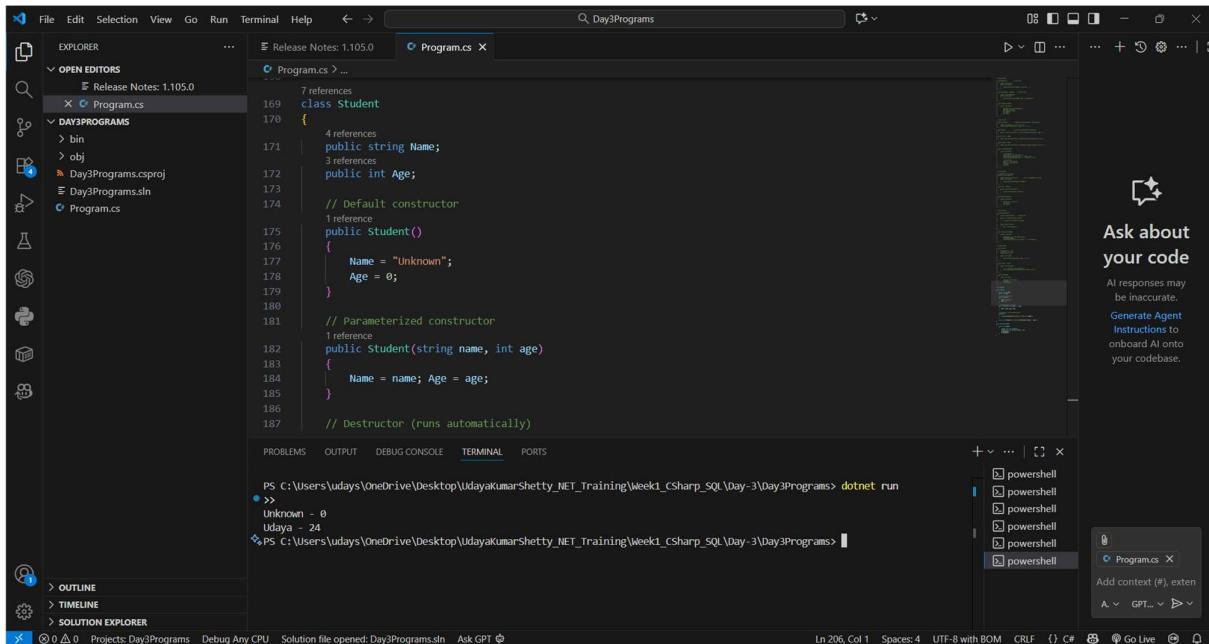
```
1 reference
class Parent
{
    1 reference
    private int a = 10;
    2 references
    protected int b = 20;
    2 references
    public int c = 30;

    1 reference
    public void Show()
    {
        Console.WriteLine($"a={a}, b={b}, c={c}");
    }
}

2 references
class Child : Parent
{
    1 reference
    public void Display()
    {
        // a is private -> not accessible
        Console.WriteLine($"Protected b={b}, Public c={c}");
    }
}
```

The terminal at the bottom shows the output of running the program: "a=10, b=20, c=30 Protected b=20, Public c=30". The status bar at the bottom right indicates "Ln 166, Col 1 Spaces:4 UTF-8 with BOM CRLF".

## Constructors & Destructors



A screenshot of the Visual Studio Code interface. The title bar says "Day3Programs". The left sidebar shows the "EXPLORER" view with a "Program.cs" file open, and the "SOLUTION EXPLORER" view. The main editor area contains the following C# code:

```
7 references
class Student
{
    4 references
    public string Name;
    3 references
    public int Age;

    // Default constructor
    1 reference
    public Student()
    {
        Name = "Unknown";
        Age = 0;
    }

    // Parameterized constructor
    1 reference
    public Student(string name, int age)
    {
        Name = name; Age = age;
    }

    // Destructor (runs automatically)
}
```

The terminal at the bottom shows the output of running the program: "Unknown - 0 Udaya - 24". The status bar at the bottom right indicates "Ln 206, Col 1 Spaces:4 UTF-8 with BOM CRLF".

## Static Class & Members

The screenshot shows the Visual Studio IDE interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The left sidebar has sections for Explorer, OPEN EDITORS, DAY3PROGRAMS, and SOLUTION EXPLORER. The main editor window displays the following C# code:

```
207     using System;
208
209     static class MathUtil
210     {
211         public static double Pi = 3.14159;
212         public static double Square(double x) => x * x;
213     }
214
215     class StaticDemo
216     {
217         static void Main()
218         {
219             Console.WriteLine("Pi: " + MathUtil.Pi);
220             Console.WriteLine("Square(4): " + MathUtil.Square(4));
221         }
222     }
223 }
```

The terminal window at the bottom shows the output of running the program with dotnet run.

```
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_NET_Training\Week1_CSharp_SQL\Day-3\Day3Programs> dotnet run
>>
Pi: 3.14159
Square(4): 16
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_NET_Training\Week1_CSharp_SQL\Day-3\Day3Programs>
```

## Sealed Class

The screenshot shows the Visual Studio IDE interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The left sidebar has sections for Explorer, OPEN EDITORS, DAY3PROGRAMS, and SOLUTION EXPLORER. The main editor window displays the following C# code:

```
225     using System;
226
227     sealed class Logger
228     {
229         public void Log(string msg) => Console.WriteLine("Log: " + msg);
230     }
231
232 // class FileLogger : Logger {} // Not allowed - sealed class
233
234     class SealedDemo
235     {
236         static void Main()
237         {
238             Logger log = new Logger();
239             log.Log("This class cannot be inherited.");
240         }
241     }
242 }
```

The terminal window at the bottom shows the output of running the program with dotnet run.

```
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_NET_Training\Week1_CSharp_SQL\Day-3\Day3Programs> dotnet run
>>
Log: This class cannot be inherited.
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_NET_Training\Week1_CSharp_SQL\Day-3\Day3Programs>
```

## Namespace Example

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like `Program.cs`, `Day3Programs.csproj`, and `Day3Programs.sln`.
- Editor:** Displays the code for `Program.cs` which includes a `namespace DhruvTraining.Day3` and a `Main` method.
- Terminal:** Shows the output of running the application with the command `dotnet run`, displaying the message "Inside DhruvTraining.Day3 namespace".
- Solution Explorer:** Shows the outline of the project.
- Status Bar:** Provides information about the file (`Program.cs`), line count (Line 267, Col 1), and encoding (UTF-8 with BOM, CRLF).
- Right Panel:** Features an "Ask about your code" AI integration and a list of PowerShell tabs.