Day 2 – C# & OOP Basics

Date: 10-10-2025

Topic: Methods, Classes, Structs, Interfaces, Enums, and OOPs Concepts

1. WHAT ARE METHODS?

- A **method** is a block of code that performs a specific task when it is called.
- Methods help in **code reusability, readability, and modularity**.
- They can have **parameters** (inputs) and **return types** (outputs).

Parts of a Method:

1. **Access Modifier:** Defines visibility (e.g., public, private, protected).
2. **Return Type:** Type of value the method returns (int, string, void etc.).
3. **Method Name:** Identifier for the method.
4. **Parameters:** Optional input data.
5. **Body:** Code block inside { }.

Example structure:

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

Types of Methods:

- **Static Methods:** Belong to the class, not to an object.
- **Instance Methods:** Belong to an object (need to create an instance first).
- **Parameterised Methods:** Accept inputs.
- **Optional / Output Parameters:** Can pass values back using out keyword.

2. CLASSES AND OBJECTS

- **Class:** A blueprint or template for creating objects.
  It defines data members (fields) and behaviors (methods).
- **Object:** A real-world entity created from a class.

Example:

- Class → Car
- Objects → Car c1, Car c2

Key Concepts:

- **Fields:** Variables inside a class.
- **Methods:** Functions that operate on data.
- **Access Modifiers:** Control visibility (public, private, etc.).

Object creation syntax:

```
ClassName obj = new ClassName();
```

OOP Advantage: Encapsulates data and methods together in a single unit.

3. STRUCTS IN C#

- **Structs** are similar to classes but **lightweight** and **value types** (stored on the stack).
- Useful for small data models like coordinates, colors, etc.
- Cannot inherit from another struct or class (but can implement interfaces).
- Default access modifier for members = private.

When to use struct:

- When data is small and doesn't require inheritance.
- When performance matters (since structs are stack-based).

4. INTERFACES IN C#

- **Interface** is a contract that defines what methods a class must implement, without defining how.
- All methods in an interface are **abstract** and **public** by default.
- A class can **implement multiple interfaces**, unlike classes (single inheritance only).

Syntax:

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

Rules:

- Cannot contain implementation code.
- Cannot have fields, but can have properties, methods, and events.

- Supports **multiple inheritance** behavior through interfaces.

5. ENUMS (ENUMERATIONS)

- **Enum** = a special value type that defines a group of named constants.
- Increases code readability and avoids using magic numbers.
- Default underlying type = int.

Example:

```
enum Level
{
   Low,    // 0
   Medium, // 1
   High    // 2
}
```

You can also assign values:

```
enum Status
{
   Started = 1,
   InProgress = 2,
   Completed = 3
}
```

Usage:

```
Status current = Status.InProgress;
Console.WriteLine((int)current); // prints 2
```

6. OBJECT-ORIENTED PROGRAMMING (OOPS) CONCEPTS

C# is **purely object-oriented** — the core of .NET programming.

1. Encapsulation

- Wrapping data and methods together inside a class.
- Hides internal details and allows controlled access.

- Achieved using **access modifiers** (private, public, etc.).

Example: Private variables with public getter/setter.

2. Abstraction

- Hiding complex internal logic and exposing only what's necessary.
- Achieved using:
  - **Abstract classes** (using abstract keyword)
  - **Interfaces**

You just define *what* needs to be done, not *how*.

3. Inheritance

- Enables one class to acquire properties and methods of another.
- Promotes **code reusability**.
- Achieved using the : symbol.

Example:

class Dog : Animal

- **Base class (Parent):** Animal
- **Derived class (Child):** Dog

C# supports **single inheritance** but allows multiple interfaces.

4. Polymorphism

- Means "many forms" — allows a single method to behave differently.
- Achieved through:
  - **Method Overloading (Compile-time)**
  - **Method Overriding (Run-time)**

Keywords: virtual, override

7. EXCEPTION HANDLING RECAP

Used to gracefully handle runtime errors.

Keywords:

- try → risky code
- catch → handles exceptions
- finally → always executes
- throw → raise exceptions manually

Example:

try { }

catch (Exception e) { }

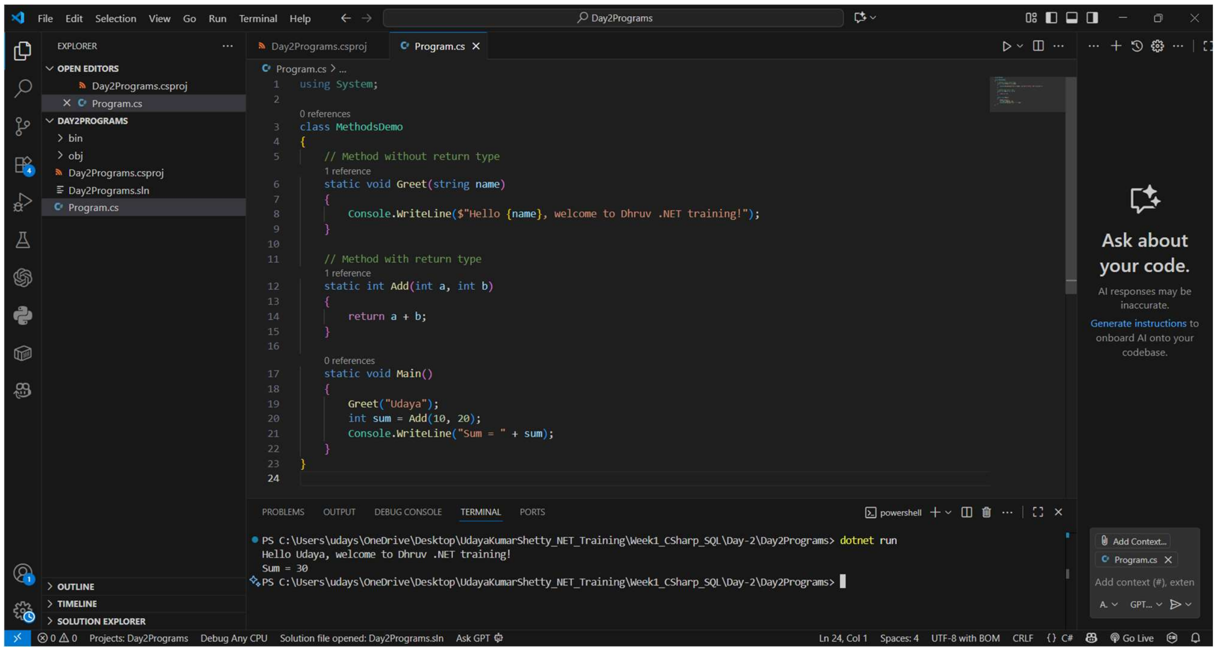
finally { }

SUMMARY – DAY 2 KEY TAKEAWAYS

Concept	Description	Example
Method	Reusable block of code	Add(int a, int b)
Class	Blueprint of object	class Student {}
Struct	Lightweight value type	struct Point {}
Interface	Contract for classes	interface IShape {}
Enum	Named constants	enum Level {Low, Medium, High}
OOPs	Core programming pillars	Encapsulation, Abstraction, Inheritance, Polymorphism

Snapshots:

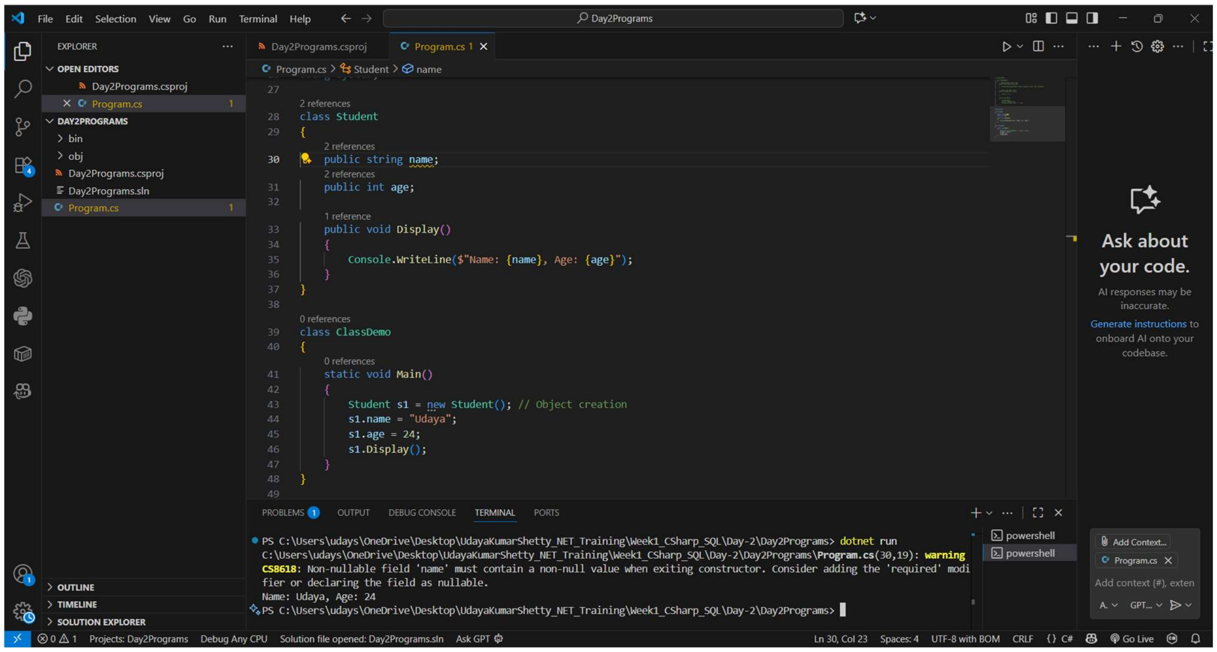
Methods in C#



```csharp
using System;

class MethodsDemo
{
    // Method without return type
    static void Greet(string name)
    {
        Console.WriteLine($"Hello {name}, welcome to Dhruv .NET training!");
    }

    // Method with return type
    static int Add(int a, int b)
    {
        return a + b;
    }

    static void Main()
    {
        Greet("Udaya");
        int sum = Add(10, 20);
        Console.WriteLine("Sum = " + sum);
    }
}
```

```
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs> dotnet run
Hello Udaya, welcome to Dhruv .NET training!
Sum = 30
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs>
```

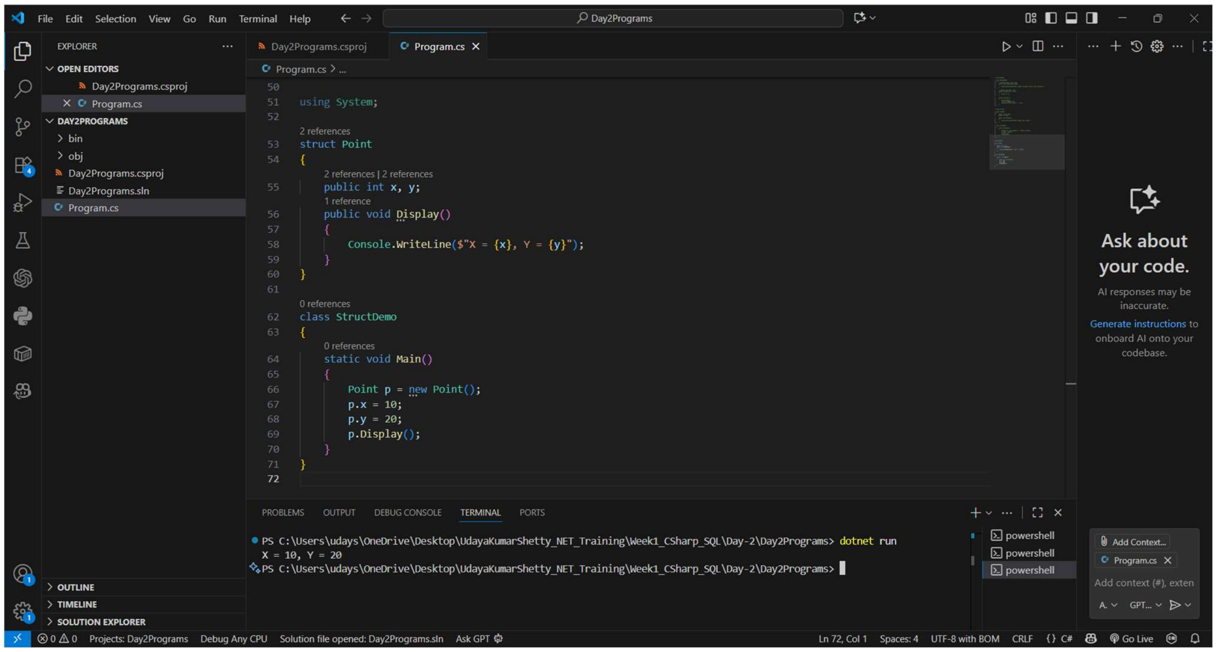
Classes and Objects



```csharp
class Student
{
    public string name;
    public int age;

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

class ClassDemo
{
    static void Main()
    {
        Student s1 = new Student(); // Object creation
        s1.name = "Udaya";
        s1.age = 24;
        s1.Display();
    }
}
```

```
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs> dotnet run
C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs\Program.cs(30,19): warning
CS8618: Non-nullable field 'name' must contain a non-null value when exiting constructor. Consider adding the 'required' modi
fier or declaring the field as nullable.
Name: Udaya, Age: 24
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs>
```

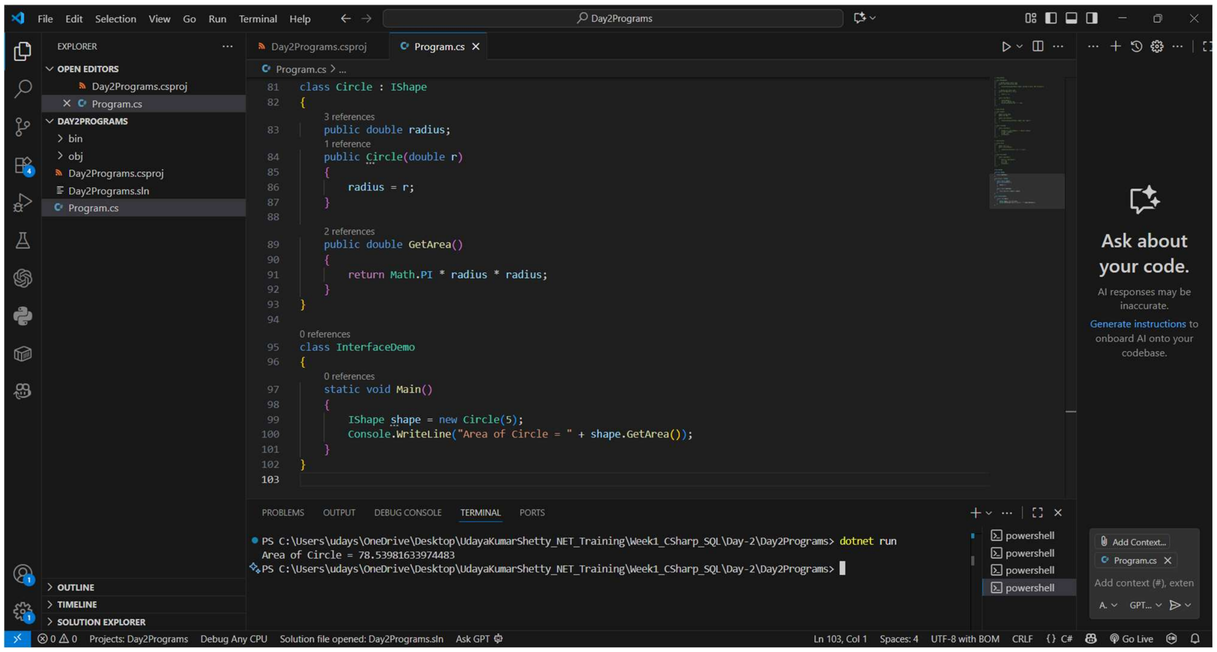
Struct Example



```csharp
using System;

struct Point
{
    public int x, y;
    public void Display()
    {
        Console.WriteLine($"X = {x}, Y = {y}");
    }
}

class StructDemo
{
    static void Main()
    {
        Point p = new Point();
        p.x = 10;
        p.y = 20;
        p.Display();
    }
}
```

```
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs> dotnet run
X = 10, Y = 20
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs>
```

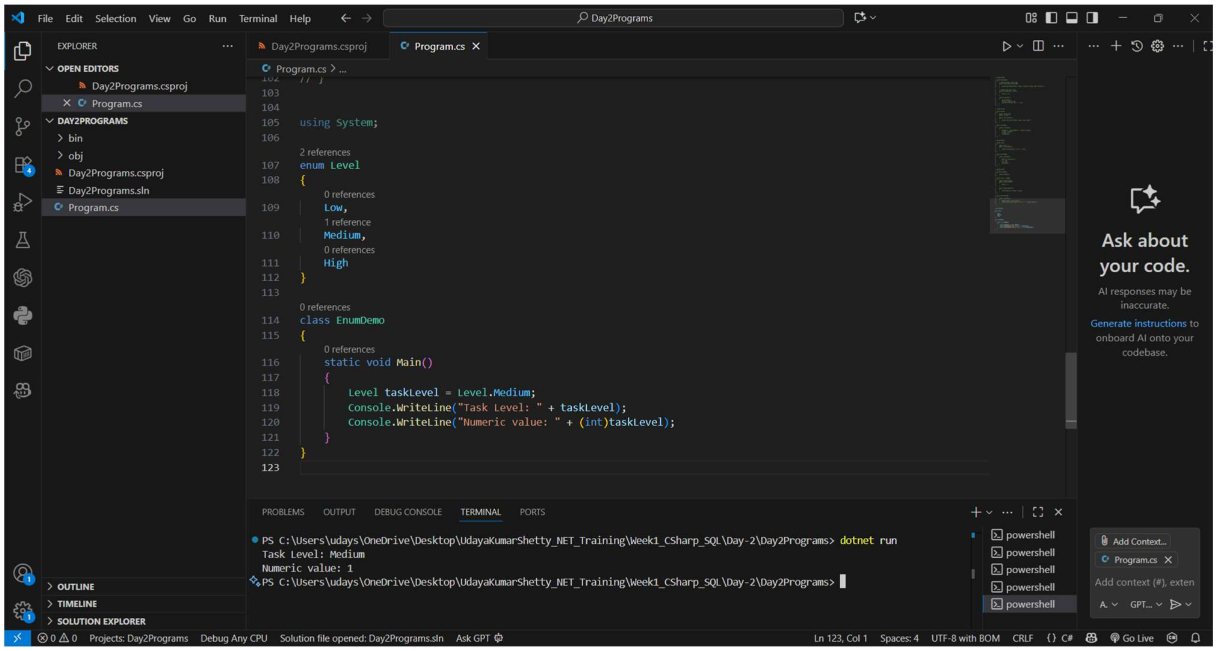
Interface Example



```csharp
class Circle : IShape
{
    public double radius;
    public Circle(double r)
    {
        radius = r;
    }

    public double GetArea()
    {
        return Math.PI * radius * radius;
    }
}

class InterfaceDemo
{
    static void Main()
    {
        IShape shape = new Circle(5);
        Console.WriteLine("Area of Circle = " + shape.GetArea());
    }
}
```

```
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs> dotnet run
Area of Circle = 78.53981633974483
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs>
```

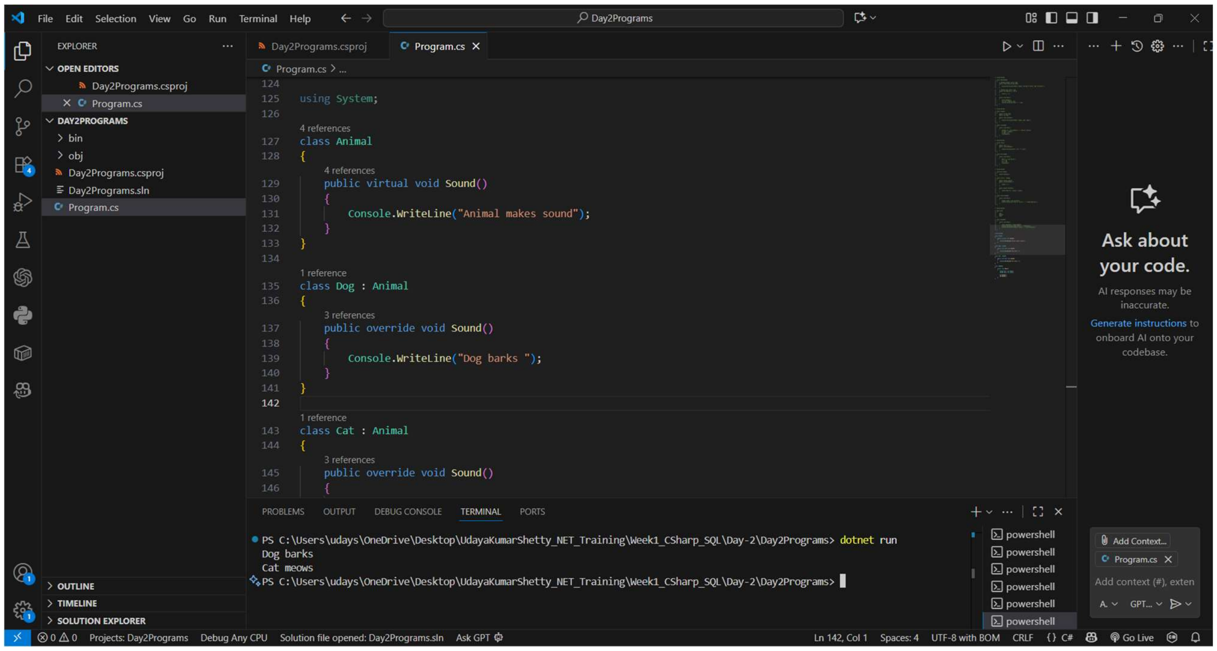
Enum Example



```csharp
using System;

enum Level
{
    Low,
    Medium,
    High
}

class EnumDemo
{
    static void Main()
    {
        Level taskLevel = Level.Medium;
        Console.WriteLine("Task Level: " + taskLevel);
        Console.WriteLine("Numeric value: " + (int)taskLevel);
    }
}
```

```
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs> dotnet run
Task Level: Medium
Numeric value: 1
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs>
```

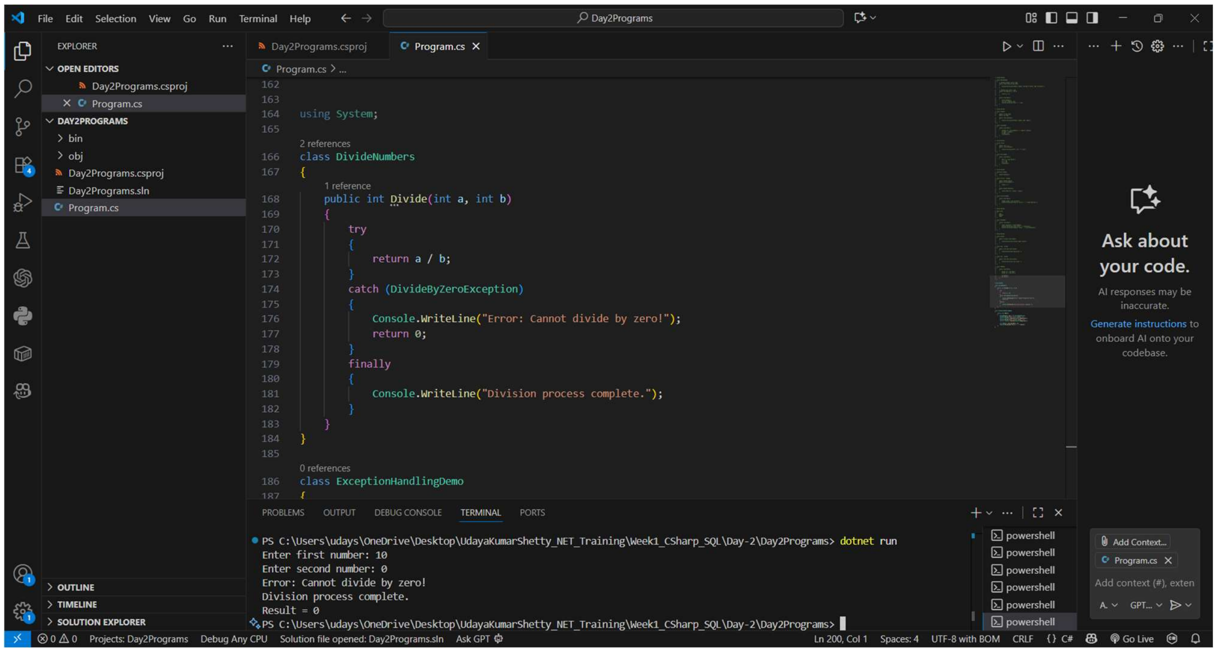
OOP Example (Encapsulation + Inheritance + Polymorphism)



```csharp
using System;

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

class Dog : Animal
{
    public override void Sound()
    {
        Console.WriteLine("Dog barks ");
    }
}

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

```
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs> dotnet run
Dog barks
Cat meows
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs>
```

Exception Handling (Recap + Inside Class)



```csharp

using System;

class DivideNumbers
{
    public int Divide(int a, int b)
    {
        try
        {
            return a / b;
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Error: Cannot divide by zero!");
            return 0;
        }
        finally
        {
            Console.WriteLine("Division process complete.");
        }
    }
}

class ExceptionHandlingDemo
{
```

```
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs> dotnet run
Enter first number: 10
Enter second number: 0
Error: Cannot divide by zero!
Division process complete.
Result = 0
PS C:\Users\udays\OneDrive\Desktop\UdayaKumarShetty_NET_Training\Week1_CSharp_SQL\Day-2\Day2Programs>
```