Classes and Structures in C#

In C#, classes and structures (structs) are fundamental constructs that are used to define types and encapsulate data and behavior. Here's an overview of both:

Classes

Definition

A class is a blueprint for creating objects. It encapsulates data (fields) and behavior (methods, properties, events, etc.).

Key Features

• Reference Type: Classes are reference types, meaning instances of classes are allocated on the heap, and variables hold references to the actual data.
• Inheritance: Classes support inheritance, allowing for the creation of a new class based on an existing class.
• Polymorphism: Through inheritance, classes can implement polymorphism, allowing methods to be overridden in derived classes.
• Encapsulation: Classes encapsulate data and behavior, promoting modularity and code reuse.

Example

```csharp
public class Person
{
    // Fields
    private string name;
    private int age;

    // Constructor
    public Person(string name, int age)
    {
        this.name = name;
        this.age = age;
    }

    // Properties
    public string Name
    {
        get { return name; }
        set { name = value; }
```

```
    }

    public int Age
    {
        get { return age; }
        set { age = value; }
    }

    // Methods
    public void DisplayInfo()
    {
        Console.WriteLine($"Name: {name}, Age: {age}");
    }
}
```

Structures

Definition

A structure (struct) is a value type that can encapsulate data and related functionality. It is similar to a class but with some key differences.

Key Features

• Value Type: Structs are value types, meaning instances are allocated on the stack or inline in containing types, and variables hold the actual data.
• No Inheritance: Structs do not support inheritance but can implement interfaces.
• Efficient Memory Usage: Due to being value types, structs can be more memory-efficient and faster to allocate/deallocate, especially for small data structures.
• Immutability Encouraged: Although not enforced, structs are often designed to be immutable to avoid unintended side effects.

Example

```csharp
public struct Point
{
    // Fields
    public int X { get; }
    public int Y { get; }

    // Constructor
    public Point(int x, int y)
    {
        X = x;
```

```
        Y = y;
    }

    // Methods
    public void DisplayCoordinates()
    {
        Console.WriteLine($"X: {X}, Y: {Y}");
    }
}
```

Comparison

Feature	Class	Struct
Type	Reference Type	Value Type
Memory Allocation	Heap	Stack (or inline)
Inheritance	Supported	Not Supported
Polymorphism	Supported	Not Supported
Default Constructor	Allowed	Not Allowed (parameterless)
Interface Implementation	Supported	Supported
Encapsulation	Yes	Yes
Typical Use Cases	Complex data structures, requiring inheritance and polymorphism	Small data structures, performance-sensitive scenarios