# Interface in C# 12.0 vs Abstract Class

# Interface in C12.0 vs Abstract Class

## 📌 Interface in C# 12.0 An \*\*interface\*\* in \*\*C# 12.0\*\* is a contract that defines a set of methods, properties, events, or indexers that a class or struct must implement. Interfaces allow multiple inheritance and provide a way to achieve polymorphism.

### \*\*Key Features of Interfaces in C# 12.0\*\* 1. \*\*Default Implementations\*\*: Interfaces can now include method bodies. 2. \*\*Static Abstract Members\*\*: Allows defining abstract static members that implementing classes must override. 3. \*\*Multiple Interface Inheritance\*\*: A class can implement multiple interfaces. 4. \*\*No Instance Fields\*\*: Unlike abstract classes, interfaces cannot have instance fields. 5. \*\*Explicit Implementation\*\*: Allows hiding members from direct access.

---

## \*\*Interface vs. Abstract Class in C# 12.0\*\*

| Feature | Interface | Abstract Class |  
|---------|----------|---------------|  
| \*\*Definition\*\* | Defines a contract that a class must implement. | Can define both abstract and non-abstract members. |  
| \*\*Implementation\*\* | Cannot have constructors, instance fields, or state. | Can have fields, constructors, and instance state. |  
| \*\*Multiple Inheritance\*\* | A class can implement multiple interfaces. | A class can inherit only one abstract class. |  
| \*\*Method Implementation\*\* | Supports default methods (C# 8+) but no instance variables. | Can have fully implemented methods with instance variables. |  
| \*\*Static Abstract Methods\*\* | Supported (C# 11+) | Not supported |  
| \*\*Encapsulation\*\* | No private or protected instance members | Can have private and protected members |  
| \*\*Use Case\*\* | Best for defining contracts and multiple behaviors. | Best for base class behavior with common implementations. |

---

## \*\*Example: Interface with Default Implementation (C# 12.0)\*\* ```csharp public interface ILogger { void Log(string message); // Abstract method

// Default method implementation  
 public virtual void LogInfo(string message)  
 {  
 Console.WriteLine($"INFO: {message}");  
 }  
}

// Class implementing interface  
public class ConsoleLogger : ILogger  
{  
 public void Log(string message)  
 {  
 Console.WriteLine($"LOG: {message}");  
 }  
}

// Usage  
class Program  
{  
 static void Main()  
 {  
 ILogger logger = new ConsoleLogger();  
 logger.Log("Hello");  
 logger.LogInfo("Logging an Info message");  
 }  
}  
```  
✔️ `LogInfo` has a default implementation and can be overridden.

---

## \*\*Example: Static Abstract Members in Interface (C# 12.0)\*\* ```csharp public interface ICalculator<T> { static abstract T Add(T a, T b); }

public class IntCalculator : ICalculator<int>  
{  
 public static int Add(int a, int b) => a + b;  
}

// Usage  
class Program  
{  
 static void Main()  
 {  
 Console.WriteLine(IntCalculator.Add(5, 10)); // Output: 15  
 }  
}  
```  
✔️ `ICalculator<T>` forces implementing classes to define the static `Add` method.

---

## \*\*Example: Abstract Class\*\* ```csharp public abstract class Animal { protected string Name; public Animal(string name) => Name = name;

public abstract void MakeSound(); // Abstract method

public void Display() => Console.WriteLine($"Animal: {Name}");  
}

// Derived class  
public class Dog : Animal  
{  
 public Dog(string name) : base(name) { }

public override void MakeSound()  
 {  
 Console.WriteLine("Woof!");  
 }  
}

// Usage  
class Program  
{  
 static void Main()  
 {  
 Dog dog = new Dog("Buddy");  
 dog.Display();  
 dog.MakeSound();  
 }  
}  
```  
✔️ `Animal` contains \*\*state (Name)\*\* and a \*\*constructor\*\*, which is not possible in an interface.

---

## \*\*When to Use What?\*\* | Scenario | Use Interface | Use Abstract Class | |----------|--------------|--------------------| | Need multiple inheritance | ✅ | ❌ | | Need default method implementation but no instance fields | ✅ | ❌ | | Need to define common fields/state | ❌ | ✅ | | Need a contract without implementation | ✅ | ❌ | | Need constructor logic | ❌ | ✅ |

### \*\*Conclusion\*\* - \*\*Use an interface\*\* when defining a contract for multiple classes to implement. - \*\*Use an abstract class\*\* when providing common behavior and maintaining state across subclasses.