Abstract Class	Interface
A class that cannot be instantiated and may contain abstract and concrete methods.	A contract that defines a set of methods and properties but contains no implementation.
Can have both abstract methods (without implementation) and non-abstract (implemented) methods.	Cannot have any method implementations (default methods are an exception in some languages, but not in C# prior to C# 8.0).
Does not support multiple inheritance (a class can inherit only one abstract class).	Supports multiple inheritance (a class can implement multiple interfaces).
Can have access modifiers like `public`, `protected`, `internal`, and `private` for methods, properties, etc.	All methods and properties in an interface are `public` by default, and no other access modifiers are allowed.
Can have fields (variables).	Cannot have fields; only properties and methods are allowed.
Can have constructors, which can be called when the class is inherited.	Cannot have constructors. Objects cannot be created from interfaces.
Slightly faster since it can have implementations.	typically slower compared to abstract classes due to the lack of implementation, which must be provided by the implementing class
Used when classes share a common base and some common functionality, but also require specific implementations.	Used to define a contract that multiple classes can implement, ensuring they follow a certain structure.
Allows default method implementation, meaning it can have methods with bodies.	Does not allow method implementations in C# before version 8.0. In C# 8.0 and later, interfaces can have default implementations with the `default` keyword.

DIFFERENCE BETWEEN OLD VERSION OF C# AND NEW VERSION OF C#

Old Versions of C# (Before C# 8.0)	**New Versions of C#** (C# 8.0 and Later)
Interfaces could not contain any method implementations. All methods were abstract and needed to be implemented by the inheriting class.	Interfaces can now contain **default method implementations** using the `default` keyword. This allows some methods to have bodies directly within the interface.
Interfaces could not contain static members.	Interfaces can now contain **static methods** and **static fields**.
Interfaces could not contain private methods.	Interfaces can now have **private methods**.
All members of an interface were implicitly `public`. No other access modifiers were allowed.	Interfaces still have public members by default, but they can now have private members as well for use in default method implementations.
Interfaces could not have static constructors.	Interfaces can now have **static constructors** to initialize static members or perform setup work.
Interfaces only allowed abstract methods (methods with no body).	Interfaces now allow a combination of **abstract methods** (without implementation) and **default methods** (with implementation).
The interface implementation was straightforward, requiring all methods to be defined by the implementing class.	The introduction of default methods maintains backward compatibility by allowing new methods to be added to interfaces without breaking existing implementations.

```
using System;

public abstract class Animal
{

    public abstract void MakeSound();
    public void Sleep()
    {
        Console.WriteLine("The animal is sleeping.");
    }
}

public class Dog : Animal
{
    public override void MakeSound()
    {
        Console.WriteLine("The dog says: Woof Woof");
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        Dog myDog = new Dog();
        myDog.MakeSound();
        myDog.Sleep();
    }
}
```

Using interfaces

```
using System;

public interface IAnimal
{
    void MakeSound();

    void Sleep();
}

public class Dog : IAnimal
{
    public void MakeSound()
    {
        Console.WriteLine("The dog says: Woof Woof");
    }

    public void Sleep()
    {
        Console.WriteLine("The dog is sleeping.");
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        IAnimal myDog = new Dog();

        myDog.MakeSound();
        myDog.Sleep();
```

```
    }
}
```