By Vikram
https://www.linkedin.com/in/4kvikram

Abstract Class

1. Definition:

- ❖ An abstract class is a class that cannot be instantiated on its own and is meant to be subclassed.
- ❖ **Cannot be instantiated on its own:** This means you cannot create an object (instance) of an abstract class using the new keyword. Attempting to do so will result in a compilation error.
- ❖ **Meant to be subclassed:** The primary purpose of an abstract class is to serve as a base class for other classes (known as subclasses or derived classes). Abstract classes define common behaviours and characteristics that subclasses can inherit and build upon.

2. Usage:

- ❖ It allows you to define common functionality for related classes.
- ❖ It can have both abstract and non-abstract (concrete) methods and properties.

3. Syntax:

```
public abstract class Animal
{
    public abstract void Sound();

    public void Eat()
    {
        Console.WriteLine("Eating...");
    }
}
```

4. Inheritance:

- ❖ Abstract classes support single inheritance in C#.
- ❖ A class can inherit from only one abstract class.

5. Example:

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

Interface

1. Definition: An interface is a contract that defines a set of methods and properties without any implementation.

2. Usage:

- ❖ It defines a contract for classes to implement.
- ❖ It supports multiple interface inheritance, allowing a class to implement multiple interfaces.

3. Syntax:

```
public interface IAnimal
{
    void Sound();
}
```

4. Implementation:

- ❖ Classes that implement an interface must provide implementations for all of its members.

5. Example:

```
public class Cat : IAnimal
{
    public void Sound()
    {
        Console.WriteLine("Meow");
    }
}
```

Differences

1. Instantiation:

- ❖ Abstract classes cannot be instantiated directly, while interfaces cannot be instantiated at all.

2. Multiple Inheritance:

- ❖ Interfaces support multiple inheritance, allowing a class to implement multiple interfaces, whereas abstract classes support single inheritance.

3. Default Implementation:

- ❖ Abstract classes can provide both abstract and concrete methods, while interfaces cannot contain any implementation.

4. Usage:

- Use abstract classes when you have a base class with some common functionality and you expect subclasses to extend or specialize that functionality.
- Use interfaces when you want to define a contract for a set of related classes to adhere to, without specifying any implementation details.

5. Flexibility:

- Interfaces provide more flexibility as they allow classes to implement multiple contracts, while abstract classes restrict a class to inherit from only one abstract class.

6. Versioning:

- ❖ Abstract classes can change over time without breaking existing derived classes, as long as the changes are additive (e.g., adding new methods).
- ❖ Interfaces, on the other hand, cannot be changed without potentially breaking existing implementations, as any change to an interface requires all implementing classes to be updated.

7. Extension:

- ❖ Abstract classes allow for method implementations to be added in subclasses, providing a way to extend the functionality of the base class.
- ❖ Interfaces do not allow for method implementations in any implementing class; they only define the method signatures that implementing classes must adhere to.

8. Fields and Properties:

- ❖ Abstract classes can have fields, properties, constructors, and destructors, while interfaces cannot contain any fields or implementations.
- ❖ Interfaces can include properties, but they must be implemented explicitly in implementing classes.

9. Accessibility:

- ❖ Members of an interface are implicitly public and cannot have any access modifiers.
- ❖ Abstract classes can have various access modifiers (public, private, protected, internal) for their members.

9. Partial Classes:

- ❖ Abstract classes can be declared as partial classes, allowing their members to be split across multiple files.

- ❖ Interfaces cannot be declared as partial; they must be implemented entirely in a single declaration.

10. Design Considerations:

- ❖ Use abstract classes when you need to provide a common base implementation for a group of related classes or when you want to define a template for subclasses to follow.
- ❖ Use interfaces when you want to define a contract that multiple unrelated classes can adhere to, promoting loose coupling and flexibility in implementations.

Abstract classes and interfaces are both used for abstraction in C#, but they have different purposes and usage scenarios. Abstract classes are used when you want to provide a common base with some default implementation, while interfaces are used when you want to define a contract without any implementation details.