|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| **Definition:** An abstract class is a class that cannot be instantiated on its own and may contain abstract methods (methods without a body) that must be implemented by derived classes. Abstract classes can also contain concrete methods (methods with a body) that can be inherited by derived classes. | **Definition:** An interface is a reference type similar to a class that contains only the declaration of methods, properties, events, or indexers. It defines a contract that implementing classes must adhere to, specifying the members they must implement. |
| **Usage:** Abstract classes are used when you want to provide a common base implementation for derived classes, along with some methods that must be implemented by subclasses. They can also contain state (fields) and code (methods). | **Usage:** Interfaces are used when you want to define a contract for classes to implement without specifying any implementation details. They are useful for defining multiple inheritance-like behavior, as a class can implement multiple interfaces. |
| **Inheritance:** A class can inherit from only one abstract class. Abstract classes can have constructors, fields, and other members like a regular class. | **Inheritance:** A class can implement multiple interfaces. Interfaces cannot contain fields, constructors, or any implementation details. |
| **Access Modifiers:** Abstract classes can have access modifiers (public, protected, internal, private) for their members. | **Access Modifiers:** Interface members are implicitly public and cannot have access modifiers. |
| **Versioning:** Changes to an abstract class may require changes to all subclasses, potentially causing versioning issues. | **Versioning:** Adding new members to an interface does not affect existing implementations, making it easier to evolve without breaking existing code. |
| **Example:**  **public abstract class Shape**  **{**  **public abstract double Area(); // Abstract method**  **}**  **public class Circle : Shape**  **{**  **private double radius;**  **public Circle(double radius)**  **{**  **this.radius = radius;**  **}**  **public override double Area() // Implementation of abstract method**  **{**  **return Math.PI \* radius \* radius;**  **}**  **}** | **Example:**  **public interface IShape**  **{**  **double Area(); // Method declaration**  **}**  **public class Rectangle : IShape**  **{**  **private double width;**  **private double height;**  **public Rectangle(double width, double height)**  **{**  **this.width = width;**  **this.height = height;**  **}**  **public double Area() // Implementation of interface method**  **{**  **return width \* height;**  **}**  **}** |