**EXPERIMENT NO: 03**

**Title**: Implementation of Interface Inheritance

**Aim:** Study Interface concepts.

**What is an Interface in C#?**

The Interface in C# is a Fully Unimplemented Class used for declaring a set of operations/ methods of an object.

So, we can define an interface as a pure abstract class, which allows us to define only abstract methods.

The abstract method means a method without a body or implementation.

It is used to achieve multiple inheritances, which the class can’t achieve.

It is used to achieve full abstraction because it cannot have a method body.

Syntax:

**Syntax** starts with the interface keyword followed by the name for the interface and then the body.

interface <name\_for\_interface>

{

//abstract methods

//abstract properties.

}

**When to use Interface?**

* **Security:** When we have to simply hide some features and have to use those later. It is essential to hide a few details while only showing the details important to the user.
* **Multiple Inheritance:** In c#, one class can inherit from a simple parent class, inheriting all its features. Multiple Inheritance is not supported in C# for the simple reason to not make C# complex. But with the use of an interface, multiple interfaces can be implemented into a single class.

### C# Interface typically includes the following elements:

1. **Declaration** – In C#, an interface is a contract that defines a set of method signatures, properties, events, or indexers. It contains no implementation and serves as a blueprint for classes to adhere to. Classes that implement an interface must provide concrete implementations for all the members declared in the interface.
2. **Members** – interface members are methods, properties, events, and indexers declared within an interface. They define a contract that implementing classes must adhere to, ensuring consistent behavior across various classes. Implementing classes must provide concrete implementations for these members, promoting code consistency, and enabling polymorphism and code reuse.
3. **Implementation** – A class that implements an interface must provide an implementation for all the members declared in the interface. The class can explicitly specify that it implements an interface using the : interfaceName syntax.For example:

public class MyClass : IMyInterface

{

public void Method1()

{

// Method implementation

}public string Property1 { get; set; }

public event EventHandler Event1;

}

1. Multiple Inheritance : C# supports multiple inheritance through interfaces. A class can implement multiple interfaces, allowing it to inherit multiple sets of method signatures without the complexities associated with multiple inheritance of implementation. This enables greater flexibility in designing classes while avoiding the diamond problem inherent in traditional multiple inheritance.
2. Interface Inheritance: In C#, interface inheritance allows a derived interface to inherit the method signatures defined in one or more base interfaces. A class implementing the derived interface must provide implementations for all the inherited methods. This enables the creation of hierarchies of interfaces, promoting code reuse and flexibility in object design.

| **Abstract Class** | **Interface** |
| --- | --- |
| It contains both declaration and implementation parts. | It contains only the declaration of methods, properties, events, or indexers. Since C# 8,[default implementations](https://devblogs.microsoft.com/dotnet/default-implementations-in-interfaces/) can also be included in interfaces. |
| Multiple inheritance is not achieved by abstract class. | Multiple inheritance is achieved by interface. |
| It contain [constructor](https://www.geeksforgeeks.org/c-sharp-constructors/). | It does not contain [constructor](https://www.geeksforgeeks.org/c-sharp-constructors/). |
| It can contain static members. | It does not contain static members. |
| It can contain different types of access modifiers like public, private, protected etc. | It only contains public access modifier because everything in the interface is public. |
| The performance of an abstract class is fast. | The performance of interface is slow because it requires time to search actual method in the corresponding class. |
| It is used to implement the core identity of class. | It is used to implement peripheral abilities of class. |
| A class can only use one abstract class. | A class can use multiple interface. |
| If many implementations are of the same kind and use common behavior, then it is superior to use abstract class. | If many implementations only share methods, then it is superior to use Interface. |
| Abstract class can contain methods, fields, constants, etc. | Interface can only contains methods, properties, indexers, events. |
| The keyword “:” can be used for implementing the Abstract class. | The keyword “:” and “,” can be used for implementing the Interface. |
| It can be fully, partially or not implemented. | It should be fully implemented. |
| To declare abstract class , we use abstract keyword. | To declare interface, we use interface keyword. |
| Example of Abstract class:-  public abstract class Fruits{ public abstract void Mango();  } | Example of Interface:-  public interface Readable{ void read(); } |

using System;

namespace CsharpInterface {

interface IPolygon {

// method without body

void calculateArea(int l, int b);

}

class Rectangle : IPolygon {

// implementation of methods inside interface

public void calculateArea(int l, int b) {

int area = l \* b;

Console.WriteLine("Area of Rectangle: " + area);

}

}

class Program {

static void Main (string [] args) {

Rectangle r1 = new Rectangle();

r1.calculateArea(100, 200);

}

}

}

**Problem Statement:**

Design a C# program that models a multimedia device, such as a television, with multiple functionalities. Create interfaces to represent different aspects of the device, such as power control, volume control, and channel control. Implement classes for the multimedia device, ensuring that each class can support one or more of these interfaces. Allow users to perform actions like turning the device on/off, adjusting the volume, and changing channels.

**Conclusion:**

Here we studied oops concepts in c# Interface.

**Sample Questions:**

1) What is an interface in C#?

2) What is the purpose of using interfaces in object-oriented programming?

3) How do you implement an interface in a C# class?

4) Can an interface inherit from another interface?

5) Can a class implement multiple interfaces in C#? If yes, how is it done?