Interfaces

This lesson discusses interfaces, their implementation as well as multiple interfaces in detail

WE'LL COVER THE FOLLOWING

Implementing an interface

An **interface** is used to enforce the presence of a *method* in any *class* that **'implements'** it.

The **interface** is defined with the *keyword* `interface` and a class can *'implement'* it by adding `: InterfaceName` after the *class* **name**.

A class can implement *multiple* **interfaces** by separating each **interface** with a *comma* like : `InterfaceName`, `ISecondInterface`.

Implementing an `interface` is simply done by *inheriting* off it and *defining* all the *methods* and *properties* declared by the `interface` after that.

Example

Let's take a look at an *example* exhibiting an *interface*.

```
using System;

public interface INoiseMaker { //defining an interface using the `interface` keyword
  string MakeNoise(); //declaring a method
}
```

```
public class Cat: INoiseMaker { //defining a class Cat that implements the interface INoiseMa
  public string MakeNoise() { //defining the method MakeNoise that was declared in the interf
    return "Nyan";
  }
}
public class Dog: INoiseMaker { //defining a class Dog that implements the interface INoiseMa
  public string MakeNoise() { //defining the method MakeNoise that was declared in the interf
    return "Woof";
  }
}

class InterfaceExample {
  static void Main() {
    Cat caty = new Cat();
    Console.WriteLine("Cat makes the noise {0}",caty.MakeNoise());

    Dog doggo = new Dog();
    Console.WriteLine("Dog makes the noise {0}",doggo.MakeNoise());
  }
}
```

In the *example* above because both the classes, `Cat` and `Dog` implement the *interface* `INoiseMaker`, both `Cat` and `Dog` are required to include the `string MakeNoise()` *method* and will fail to *compile* without it.

Implementing multiple Interfaces

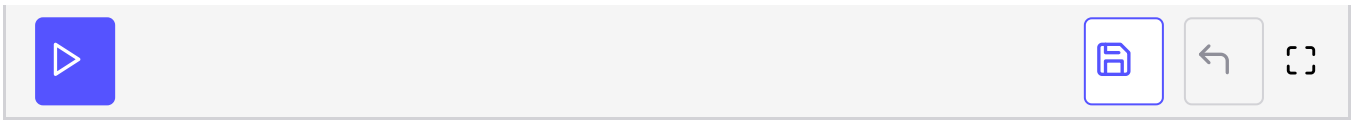
You can implement **multiple** *interfaces* in a class as well.

Example

Take a look at an example below implementing **multiple** *interfaces*.

```
using System;

public interface IAnimal { //defining interface IAnimal
  string Name {get;set;}
}

public interface INoiseMaker { //defining interface INoiseMaker
  string MakeNoise();
}

public class Cat: IAnimal, INoiseMaker { //class implementing the two interfaces
  public Cat() { //default constructor for the class Cat
    Name = "Cat";
  }
  public string Name {get;set;} //defining the name variable from interface IAnimal
  public string MakeNoise() { //defining the MakeNoise() variable from interface INoiseMaker
    return "Nyan";
  }
```

```
}
class HelloWorld {
  static void Main() {

    Cat obj = new Cat(); //making the object of Cat that inherits from the two interfaces
    Console.WriteLine("Animal name is {0}",obj.Name);
    Console.WriteLine("Animal makes the noise {0}",obj.MakeNoise());
  }
}
```



Explicit interface implementation

Explicit *interface* implementation is necessary when you implement *multiple interfaces* which define a **common** *method*, but different *implementations* are required depending on which *interface* is being used to call the *method*.

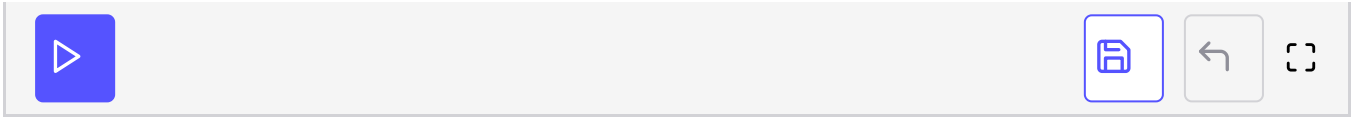
> **Note:** You don't need **explicit** *implementations* if **multiple** *interfaces* share the same *method* and a *common* implementation is possible.

Example

Let's take a look at an *example* in which the class *inherits* from **multiple** *interfaces*.

```
using System;

interface IChauffeur { //defining interface
  string Drive();
}

interface IGolfPlayer { //defining interface
  string Drive();
}

class GolfingChauffeur: IChauffeur, IGolfPlayer { //class implementing the two interfaces def
  public string Drive() { //defining Drive as a method of the class implementing the two inte
    return "Vroom!";
  }
  string IChauffeur.Drive(){ //defining Drive method for IChauffeur interface
    return "Boom Boom!";
  }
  string IGolfPlayer.Drive() { //defining Drive method for IGolfPlayer interface
    return "Took a swing...";
  }
}

class MultipleInterfacesExample {
  static void Main() {
```

```
    GolfingChauffeur obj = new GolfingChauffeur(); //making class object called obj
    IChauffeur chauffeur = obj; //creating and setting IChauffeur object `chauffeur` equal to
    IGolfPlayer golfer = obj; //creating and setting IGolfPlayer object `golfer` equal to `ob

    Console.WriteLine(obj.Drive()); // calling Drive for object obj
    Console.WriteLine(chauffeur.Drive()); // calling Drive for object chauffeur
    Console.WriteLine(golfer.Drive()); //calling Drive for object golfer

  }
}
```



An **explicit** *interface* implementation can of course only be used for *methods* that actually exist for that *interface*.

Similarly, using an **explicit** *interface* implementation without declaring that *interface* on the *class* causes an **error**, too.

> **Note: Interfaces** can *inherit* off of any number of other *interfaces*, but **cannot** *inherit* from *classes*.

This marks the end of our discussion on *interfaces*. In the next chapter, we will discuss *delegates* in C#.