Chapter 33: Interfaces

An **Interface** is a way to define a set of behaviors that a class will perform. The definition of an interface is a list of method signatures (name, parameters, and return type). A class having all of the methods is said to "implement" that interface.

In VBA, using interfaces lets the compiler check that a module implements all of its methods. A variable or parameter can be defined in terms of an interface instead of a specific class.

Section 33.1: Multiple Interfaces in One Class - Flyable and Swimable

Using the `Flyable` example as a starting point, we can add a second interface, `Swimmable`, with the following code:

```
Sub Swim()
    ' No code
End Sub
```

The `Duck` object can `Implement` both flying and swimming:

```
Implements Flyable
Implements Swimmable

Public Sub Flyable_Fly()
    Debug.Print "Flying With Wings!"
End Sub

Public Function Flyable_GetAltitude() As Long
    Flyable_GetAltitude = 30
End Function

Public Sub Swimmable_Swim()
    Debug.Print "Floating on the water"
End Sub
```

A `Fish` class can implement `Swimmable`, too:

```
Implements Swimmable

Public Sub Swimmable_Swim()
    Debug.Print "Swimming under the water"
End Sub
```

Now, we can see that the `Duck` object can be passed to a Sub as a `Flyable` on one hand, and a `Swimmable` on the other:

```
Sub InterfaceTest()

Dim MyDuck As New Duck
Dim MyAirplane As New Airplane
Dim MyFish As New Fish

Debug.Print "Fly Check..."

FlyAndCheckAltitude MyDuck
FlyAndCheckAltitude MyAirplane
```

```
Debug.Print "Swim Check..."

TrySwimming MyDuck
TrySwimming MyFish

End Sub

Public Sub FlyAndCheckAltitude(F As Flyable)
    F.Fly
    Debug.Print F.GetAltitude
End Sub

Public Sub TrySwimming(S As Swimmable)
    S.Swim
End Sub
```

The output of this code is:

> Fly Check...
>
> Flying With Wings!
>
> 30
>
> Flying With Jet Engines!
>
> 10000
>
> Swim Check...
>
> Floating on the water
>
> Swimming under the water

Section 33.2: Simple Interface - Flyable

The interface `Flyable` is a class module with the following code:

```
Public Sub Fly()
    ' No code.
End Sub

Public Function GetAltitude() As Long
    ' No code.
End Function
```

A class module, `Airplane`, uses the **`Implements`** keyword to tell the compiler to raise an error unless it has two methods: a `Flyable_Fly()` sub and a `Flyable_GetAltitude()` function that returns a `Long`.

```
Implements Flyable

Public Sub Flyable_Fly()
    Debug.Print "Flying With Jet Engines!"
End Sub

Public Function Flyable_GetAltitude() As Long
    Flyable_GetAltitude = 10000
```

```
End Function
```

A second class module, `Duck`, also implements `Flyable`:

```
Implements Flyable

Public Sub Flyable_Fly()
    Debug.Print "Flying With Wings!"
End Sub

Public Function Flyable_GetAltitude() As Long
    Flyable_GetAltitude = 30
End Function
```

We can write a routine that accepts any `Flyable` value, knowing that it will respond to a command of `Fly` or `GetAltitude`:

```
Public Sub FlyAndCheckAltitude(F As Flyable)
    F.Fly
    Debug.Print F.GetAltitude
End Sub
```

Because the interface is defined, the IntelliSense popup window will show `Fly` and `GetAltitude` for `F`.

When we run the following code:

```
Dim MyDuck As New Duck
Dim MyAirplane As New Airplane

FlyAndCheckAltitude MyDuck
FlyAndCheckAltitude MyAirplane
```

The output is:

```
Flying With Wings!
30
Flying With Jet Engines!
10000
```

Note that even though the subroutine is named `Flyable_Fly` in both `Airplane` and `Duck`, it can be called as `Fly` when the variable or parameter is defined as `Flyable`. If the variable is defined specifically as a `Duck`, it would have to be called as `Flyable_Fly`.