The Learning House

427 S 4th Street #300

Louisville KY 40202

Interfaces

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

Lesson Goals

- Learn to declare and implement interfaces
- Recognize the difference between interfaces and abstract classes

What is an Interface?

- An *interface* is a *reference type* that specifies a set of *function members* but does **not** implement them.
- Classes and structs (more on structs later) *implement the interface*.
- Think of an interface as a *contract*. Any class or struct that implements an interface **must** provide implementation for the interface's function members.
- Interfaces are a key concept for polymorphism and loose-coupling.

A simple interface

DEMO

Using Built-In Interfaces

- There are some commonly-used interfaces in the Base Class Library for doing common tasks like comparing objects, sorting them, etc.
- One such example is IComparable, which implements a method called CompareTo
  - The developer is required to implement CompareTo and return a -1, 0, or 1 depending on whether the object comparison is less, equal, or greater.

IComparable Example

DEMO

Declaring Interfaces

- An interface cannot contain:
  - Data members
  - Static members
- An interface can contain:
  - Methods
  - Properties
  - Events
  - Indexers
- The declarations of function members cannot contain code and a semicolon is put in place of the body.
- Most developers prefer to prefix interfaces with an uppercase I (ex: ICloneable).
- Interfaces can be partial.
- Interface declarations can be public, protected, internal, or private.
- Members of the interface are implicitly public; as such, *no access modifiers* are permitted.

Declaring Interfaces Illustrated

```
// interface methods have no body
interface ITaxCalculator
{
    decimal GetIncome(string ssn);
    decimal CalculateTax(decimal rate, decimal netIncome);
}

public interface ITemperatureConverter
{
    // we can not put access modifiers on the methods
    public decimal CelciusToKelvin(decimal celcius);
}
```

Implementing Interfaces

Only classes or structs can implement interfaces.

They must include the interface name in the base class list.

They must include an implementation for all of the interface's members.

We often use interfaces via polymorphism in order to inject code that behaves differently into our application. For example: we could have a list of states from the database, or one that is hard-coded with sample data.

```
interface IStateRepository
{
    string[] GetAbbreviations();
}

class HardCodedStateRepository : IStateRepository
{
    public string[] GetAbbreviations()
    {
        return new[] {"AL", "AK", "AZ", "AR"}; // etc
    }
}

class SQLStateRepository : IStateRepository
{
    public string[] GetAbbreviations()
    {
        // code to get list from database
    }
}
```

Using the *as* Operator

- If we want to check if a class implements an interface or inherits from a base class, we can use the *as* operator to check.
- If the conversion is successful, the variable will be assigned. If not, it will be null.

```
var otherTemperature = obj as Temperature;

if (otherTemperature != null)
{
    if (this.Fahrenheit < otherTemperature.Fahrenheit)
        return -1;

    if (this.Fahrenheit == otherTemperature.Fahrenheit)
        return 0;

    return 1;
}
else
    throw new ArgumentException("Object is not a Temperature");
```

You can implement multiple interfaces

Just separate them with a comma

Common Job Interview Question: What is the difference between inheritance (abstract class) and interfaces?

Two main points:

1. Interfaces can not provide implementation, but inherited classes can.
2. You may only inherit one class, but you can implement many interfaces.

Developers tend to prefer interfaces because of the single inheritance rule.

```
interface IDataRetrieve
{
    int GetData();
}

interface IDataStore
{
    void SetData( int x );
}

internal class MyData : IDataRetrieve, IDataStore
{
    private int _data;
    public int GetData()
    {
        return _data;
    }

    public void SetData(int x)
    {
        _data = x;
    }
}
```

What if you have two interfaces with the same named members?

In this case we can provide a single implementation to satisfy both interfaces

— or —

we use the *qualified interface name*, which consists of the interface name and member name, separated by a dot.

We generally try to avoid this situation because it can be confusing, but it is sometimes unavoidable when working with other libraries.

```
interface IPrintable
{
    void Output(string s);
}

interface IOutputable
{
    void Output(string s);
}

internal class MyShared : IPrintable, IOutputable
{
    public void Output(string s)
    {
        Console.WriteLine(s);
    }
}

internal class MyQualified : IPrintable, IOutputable
{

    void IPrintable.Output(string s)
    {
        Console.WriteLine(s);
    }

    void IOutputable.Output(string s)
    {
        Console.WriteLine(s);
    }
}
```