# Tutorial - Interfaces

In this exercise you will learn more about interfaces and when to use them, as well as how to create your own.

## Interfaces:

Interfaces allow us to add extra functionality to existing classes by defining a structure that implementing classes must define and implement. It ensures a common structure between different classes (but not necessarily common functionality as the functions could do different things).

Interfaces can only declare empty methods, events, indexers and properties that an implementing class must provide definitions for. Implementing an interface is almost like a contractual binding between the class and the interface – the class is guaranteeing to implement the interface, else a compiler error will be generated.

A class can only inherit from one other class at a time, but it can implement multiple interfaces at a time. C# also supports abstract classes, which are similar to interfaces. For more information on abstract classes, see here.

C# has many different built in interfaces which provide various different functionality. One of the more useful ones is the **IComparable** interface, which lets us sort our own classes using the built in sort functionality in .NET collections. Implementing **IEnumerable** will let you iterate over a custom class that holds a collection using the foreach keyword.

**Exercise:**

1. Create a new class or modify an existing one to implement the **IComparable** interface.
2. Create a list of your class and sort it using a .NET library.
3. Research the **IEquatable** and **ICloneable** interfaces. Create a small class or modify your existing one to demonstrate these in action.

## Creating your own Interfaces:

Now, we are going to look at creating your own interface. The syntax is quite straightforward. It is much like declaring a class but uses the keyword **interface** instead. The following code demonstrates how to create a sample interface. Note that members of an interface are always public – they can never be private. Interfaces cannot have member variables. Instead, you can provide an empty property that is implemented later.

```
public interface ITestInterface
{
    void SampleFunction();

    int SampleProperty
    {
        get;
        set;
    }
}
```

An implementation of **ITestInterface** could possibly look like follows:

```
public class TestClass : ITestInterface
{
    private int sampleVariable;

    public void SampleFunction()
    {
        Console.WriteLine("This is my implementation for Sample Function. Sample
Variable is equal to: " + sampleVariable);
    }

    public int SampleProperty
    {
        get
        {
            return sampleVariable;
        }

        set
        {
            sampleVariable = value;
        }
    }
}
```

**Exercises:**

1. Create an interface, ILoggable which contains one method: Log().
2. Create a few small test classes, or have existing classes implement ILoggable, and implement the Log() method. The log method should output detailed information about the class.