Interfaces

Syntax

```
public interface IList : ICollection, IEnumerable {
    int Add (object value);                 // methods
    bool Contains (object value);
    ...
    bool IsReadOnly { get; }                // property
    ...
    object this [int index] { get; set; }   // indexer
}
```

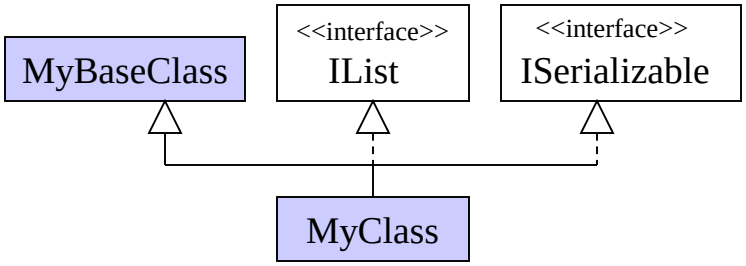
- Interface = purely abstract class; only signatures, no implementation.
- May contain methods, properties, indexers and events
  (no fields, constants, constructors, destructors, operators, nested types).
- Interface members are implicitly *public abstract* (*virtual*).
- Interface members must not be *static*.
- Classes and structs may implement multiple interfaces.
- Interfaces can extend other interfaces.

Implemented by Classes and Structs


```
class MyClass : MyBaseClass, IList, ISerializable {
    public int Add (object value) {...}
    public bool Contains (object value) {...}
    ...
    public bool IsReadOnly { get {...} }
    ...
    public object this [int index] { get {...} set {...} }
}
```

- A class can inherit from a *single base class*, but can implement *multiple interfaces*.
  A struct cannot inherit from any type, but can implement multiple interfaces.
- Every interface member (method, property, indexer) must be *implemented* or *inherited* from a base class.
- Implemented interface methods need *not* be declared as *override*.
- Implemented interface methods can be declared as *abstract* (i.e. an interface can be implemented by an abstract class).
- If a subclasses of *MyClass* should be able to override *Add()* it must be declared as *virtual* (although *Add()* is already implicitly *virtual* in *IList*).

Working with Interfaces



Assignments:

```
MyClass c = new MyClass();
IList list = c;
```

Method calls:

```
list.Add("Tom");          // dynamic binding => MyClass.Add
```

Type checks:

```
if (list is MyClass) ...          // true
if (list is ISerializable) ...    // true
```

Type casts:

```
c = list as MyClass;
c = (MyClass) list;

ISerializable ser = (ISerializable) list;
```

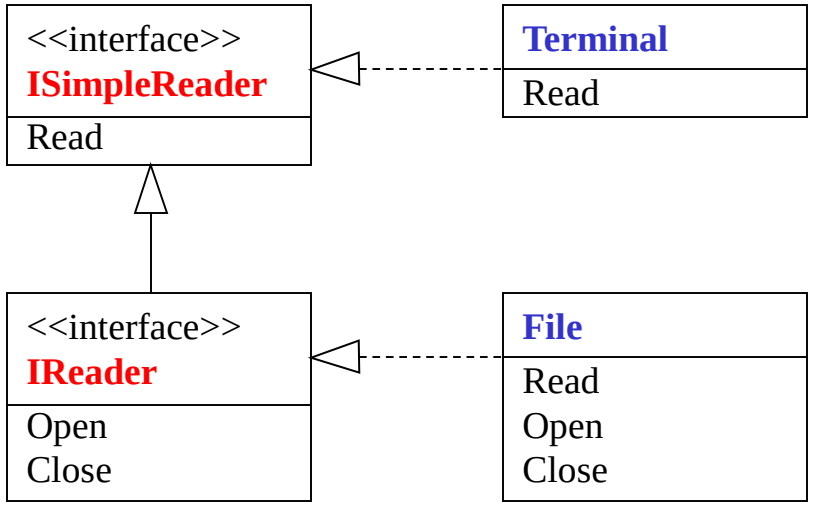
Example

```
interface ISimpleReader {
    int Read();
}

interface IReader : ISimpleReader {
    void Open(string name);
    void Close();
}

class Terminal : ISimpleReader {
    public int Read() { ... }
}

class File : IReader {
    public int Read() { ... }
    public void Open(string name) { ... }
    public void Close() { ... }
}
```




```
ISimpleReader sr = null;      // null can be assigned to any variable of an interface type
sr = new Terminal();
sr = new File();

IReader r = new File();
sr = r;
```

Name Clashes

Occurs if two base interfaces have methods with identical names

```
interface I1 {
    void F();
}

interface I2 {
    void F();
}

class B : I1, I2 {
    //----- implementation by a single F method
    public void F() { Console.WriteLine("B.F"); }
    //----- implementation by separate F methods (in addition to the above F method)
    void I1.F() { Console.WriteLine("I1.F"); }  // must not be public (don't know why)
    void I2.F() { Console.WriteLine("I2.F"); }  // -- " --
}
```

```
B b = new B();
b.F();              // B.F

I1 i1 = b;
i1.F();             // I1.F

I2 i2 = b;
i2.F();             // I2.F
```