Interfaces

Syntax


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

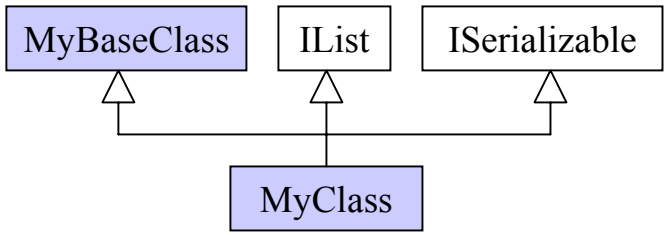
- Interface = rein abstrakte Klasse: nur Schnittstelle, kein Code.
- Darf nur Methoden, Properties, Indexers und Events enthalten
  (keine Felder, Konstanten, Konstruktoren, Destruktoren, Operatoren, innere Typen).
- Interface-Members sind implizit *public abstract* (*virtual*).
- Interface-Members dürfen nicht *static* sein.
- Klassen und Structs können mehrere Interfaces implementieren.
- Interfaces können andere Interfaces erweitern.

Implementierung eines Interface


```
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 {...} }
}
```

- Eine Klasse kann von *einer* Klasse erben, aber *beliebig viele* Interfaces implementieren.
- Jede geerbte Interface-Methode (Property, Indexer) muß *implementiert* oder von einer anderen Klasse *geerbt* werden.
- Beim Überschreiben von Interface-Methoden darf man kein *override* angeben, außer man überschreibt eine von einer Klasse geerbte Methode.
- Beim Überschreiben von Interface-Methoden darf man *abstract* angeben (d.h. ein Interface darf durch eine abstrakte Klasse implementiert werden).
- Wenn Unterklassen von *MyClass* z.B. *Add()* überschreiben sollen, muß man es als *virtual* deklarieren (obwohl *Add()* eigentlich schon in *IList* implizit *virtual* sind).

Benutzung von Interfaces



Zuweisungen:

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

Methodenaufrufe:

```
list.Add("Tom");          // dyn.Bindung => MyClass.Add
```

Typprüfungen:

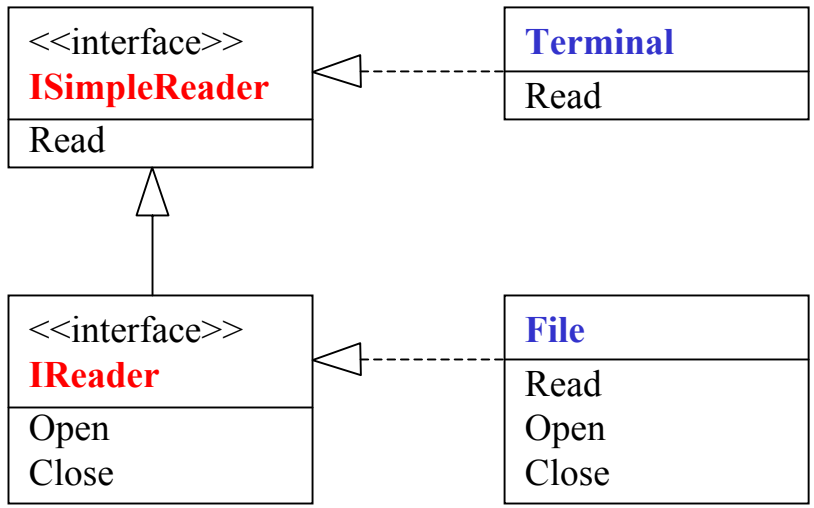
```
if (list is MyClass) ...  // true
```

Typumwandlungen:

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

ISerializable ser = (ISerializable) list;
```

Beispiel

```
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;      // Zuweisung von null erlaubt
sr = new Terminal();
sr = new File();

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



Name Clashes

Wenn zwei geerbte Interfaces eine gleichnamige Methode enthalten

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

interface I2 {
    void F();
}

class B : I1, I2 {
    //----- Implementierung durch eine einzige F-Methode
    public void F() { Console.WriteLine("B.F"); }
    //----- Implementierung durch getrennte F-Methoden
    void I1.F() { Console.WriteLine("I1.F"); }  // darf nicht public sein !?!
    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
```