CS 683 Emerging Technologies
Spring Semester, 2003
Doc 24 C# Interfaces
Contents

References

C# Language Specification, http://download.microsoft.com/download/0/a/c/0acb3585-3f3f-4169-ad61-efc9f0176788/CSharp.zip

Programming C#, Jesse Liberty, O'Reilly, Chapters 8

Operator List

Unary

+, -, !, ~, ++, --, true, false

Binary

+, -, *, /, %, &, |, ^, <<, >>, ==, !=, >, <, >=, <=

The following operators require pair-wise declaration:

- operator == and operator !=
- operator > and operator <
- operator >= and operator <=

Interfaces

An interface can be
- new
- public
- protected
- internal
- private

All methods in interface are public

It is a compile-time error for interface member declarations to include any modifiers

```
interface IExample
        {
        void Read(string fileName);
        int getFoo();
        int Status { get; set; }
        }

public class Bar : IExample
        {
        public void Read(String name) { }
        public int getFoo() { return 5;}

        public int Status
                {
                get { return 10;}
                set {  }
                }
        }
```

Interface Members

- method
- property
- event
- indexer

```
public delegate void StringListEvent(IStringList sender);

public interface IStringList
{
   void Add(string s);
   int Count { get; }
   event StringListEvent Changed;
   string this[int index] { get; set; }
}
```

Extending & Combining Interfaces

```
interface IControl
{
   void Paint();
}
interface ITextBox: IControl
{
   void SetText(string text);
}
interface IListBox: IControl
{
   void SetItems(string[] items);
}
interface IComboBox: ITextBox, IListBox {}
```

It is a compile-time error for an interface to directly or indirectly inherit from itself.

is

expression is type

true if the result “expression” can converted to “type” by
- a reference conversion,
- a boxing conversion, or
- an unboxing conversion.

Other conversions, such as user defined conversions, are not considered by the is operator.

is Example

```
interface IRead
        {
        void Read(string fileName);
        }

interface IWrite
        {
        void Write();
        }

public class Bar : IRead
        {
        public void Read(String name) { }
        }


public class Tester
        {
        public static void Main()
                {
                Bar test = new Bar();
                bool result =test is IRead;
                result = test is IWrite;
                }
        }
```

as

Explicitly convert a value to a given reference type using a reference conversion or a boxing conversion

If the conversion is not possible, the resulting value is null.

```
interface IRead    { void Read(string fileName);  }

interface IWrite { void Write(); }

public class Bar : IRead  { public void Read(String name) { } }


public class Tester
      {
      public static void Main()
            {
            Bar test = new Bar();
            IRead reader =test as IRead;
            reader = (IRead) test;
            reader = test;

            IWrite writer = test as IWrite;//assigns null
            writer = (IWrite) test;                  // runtime exception
            writer = test;                           // Compile error
            }
      }
```

struct and Interfaces

```
using System;
interface IExample  { int Foo { get; set;}  }

struct StructExample : IExample  {
      int foo;
      public int Foo {
            get { return foo; }
            set { foo = value;}
      }
}
class ClassExample : IExample{
      int foo;
      public int Foo  {
            get { return foo; }
            set { foo = value; }
      }
}
public class Tester {
      public static void Main() {
            ClassExample classTest = new ClassExample();
            classTest.Foo = 5;
            IExample x = classTest;
            x.Foo = 10;
            Console.WriteLine(classTest.Foo);              //Prints 10

            StructExample structTest = new StructExample();
            structTest.Foo = 5;
            x = structTest;            //Copies the struct
            x.Foo = 10;
            Console.WriteLine(structTest.Foo);             // Prints 5
            }
      }
```

Fully qualified Names

```
interface IControl
{
   void Paint();
}
interface ITextBox: IControl
{
   void SetText(string text);
}
```

Name	Fully Qualified Name
Paint	IControl.Paint
SetText	ITextBox.SetText.

When an interface is part of a namespace, the fully qualified name of an interface member includes the namespace name

```
namespace System
{
   public interface ICloneable
   {
      object Clone();
   }
}
```

Name	Fully Qualified Name
Clone	System.ICloneable.Clone

Interface Member Access

```
interface IList
{
   int Count { get; set; }
}
interface ICounter : IList
{
   void Count(int i);
}
class C
{
   void Test(ICounter x) {
      x.Count(1);                    // OK
      x.Count = 1;                   // Error
      ((IList)x).Count = 1;       // Ok, invokes IList.Count.set
      ((ICounter)x).Count(1);        // Ok, invokes ICounter.Count
   }
}
```

Name Clashes – Ambiguous References

Structs, classes, interfaces can implement multiple interfaces

Implementing two interfaces that contain members with the same name causes problems

Following examples will illustrate some of these problems

Just say no to interfaces with members with the same name

Simple Example

```
interface IList
{
   int Count { get; set; }
}
interface ICounter
{
   void Count(int i);
}
interface IListCounter: IList, ICounter {}

class C
{
   void Test(IListCounter x) {
      x.Count(1);                    // Error
      x.Count = 1;                   // Error
      ((IList)x).Count = 1;        // Ok, invokes IList.Count.set
      ((ICounter)x).Count(1);        // Ok, invokes ICounter.Count
   }
}
```

Implicit casting and Name clash

```
interface IInteger
{
   void Add(int i);
}
interface IDouble
{
   void Add(double d);
}
interface INumber: IInteger, IDouble {}
class C
{
   void Test(INumber n) {
      n.Add(1);              // Error, both Add methods are applicable
      n.Add(1.0);                  // Ok, only IDouble.Add is applicable
      ((IInteger)n).Add(1);        // Ok, only IInteger.Add is a candidate
      ((IDouble)n).Add(1);               // Ok, only IDouble.Add is a
candidate
   }
}
```

Which is first?

```
interface IBase
{
   void F(int i);
}

interface ILeft: IBase
{
   new void F(int i);
}

interface IRight: IBase
{
   void G();
}

interface IDerived: ILeft, IRight {}

class A
{
   void Test(IDerived d) {
      d.F(1);              // Invokes ILeft.F
      ((IBase)d).F(1);     // Invokes IBase.F
      ((ILeft)d).F(1);     // Invokes ILeft.F
      ((IRight)d).F(1);          // Invokes IBase.F
   }
}
```

Explicit Interface Member Implementations

```
interface IDisposable {
        void Dispose();
}

class MyFile: IDisposable
{
        // Explicit Interface Implementation
        void IDisposable.Dispose() {
                Close();
        }
        public void Close() {
                // Do what's necessary to close the file
                        System.GC.SuppressFinalize(this);
        }
}
```

It is not possible to access an explicit interface member implementation through its fully qualified name in a method invocation, property access, or indexer access.

An explicit interface member implementation can only be accessed through an interface instance.

Why Explicit interface member implementations

- Allow interface implementations to be excluded from the public interface of a class or struct.

- Explicit interface member implementations allow disambiguation of interface members with the same signature.

Example

```
using System;
interface IHide
{
      void DoIt();
}


public class ListEntry: IHide
{
      void  IHide.DoIt() {Console.WriteLine( "Hide ");}
}

public class Tester
      {
      public static void Main()
            {
            ListEntry x = new ListEntry();
            x.DoIt();                        // Compile Error
            IHide ok = x;
            ok.DoIt();
            }
```

Two Implementations

```
using System;
interface IHide
{
      void DoIt();
}


public class ListEntry: IHide
{
      void  IHide.DoIt() {Console.WriteLine( "Hide ");}
      public void DoIt() {Console.WriteLine( "Visible");}
}

public class Tester
{
      public static void Main()
      {
            ListEntry x = new ListEntry();
            x.DoIt();                              // Prints Visible
            IHide ok = x;
            ok.DoIt();                             // Prints Hide
      }
}
```

What Happens Here?

```
using System;
interface IHide
{
	void DoIt();
}
interface ISee
{
	void DoIt();
}

public class ListEntry:  ISee,  IHide
{
	void  IHide.DoIt() {Console.WriteLine( "Hide ");}
	public void DoIt() {Console.WriteLine( "Visible");}
}

public class Tester
{
	public static void Main()
	{
		ListEntry x = new ListEntry();
		x.DoIt();
		IHide ok = x;
		ok.DoIt();
		ISee what = x;
		what.DoIt();
	}
}
```

Three Versions

```
using System;

interface IHide {   void DoIt(); }

interface ISee : IHide {   new void DoIt(); }


public class ListEntry: ISee
{
      void  IHide.DoIt() {Console.WriteLine( "Hide ");}
      void  ISee.DoIt() {Console.WriteLine( "See ");}
      public void DoIt() {Console.WriteLine( "Visible");}

}

public class Tester
{
      public static void Main()
      {
            ListEntry x = new ListEntry();
            x.DoIt();                  //Prints Visible
            IHide ok = x;
            ok.DoIt();                 //Prints Hide
            ISee what = x;
            what.DoIt();               //Prints See
      }
}
```