接口：定义又类实现的一组方法，而其自身不实现任何方法。

interface name {

ret-type method-name1(param-list);

ret-type method-name2(param-list);

…

}

定义一个类可以在多个接口中去实现它，一个类也可以同时实现多个接口。

接口中的所有方法默认为public，并且不允许更改，不允许使用访问控制修饰符。

接口中的成员不能是static。

调用接口的类必须实现其中的所有方法。

除了方法之外，接口也可以指定索引器、属性和事件。

class class-name : interface-name {}

类中实现接口的方法必须声明为public，因为接口方法默认为public。

注意：在C#中的虚函数和抽象方法，在派生类中不能更改其访问控制类别，在C++可以却可以更改，基类private，派生类可以为public，基类为public，在派生类可以为private。

例：interface\_test1

// Copyright 2016.刘珅珅

// author：刘珅珅

// 接口测试

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace interface\_test1

{

public interface ISeries

{

int GetNext();

void Reset();

void SetStart(int x);

}

class ByTwos : ISeries

{

int start;

int val;

public ByTwos()

{

start = 0;

val = 0;

}

public int GetNext()

{

val += 2;

return val;

}

public void Reset()

{

val = start;

}

public void SetStart(int x)

{

start = x;

val = start;

}

}

class InterfaceTest

{

static void Main(string[] args)

{

ByTwos obj = new ByTwos();

for (int i = 0; i < 5; ++i)

{

*Console*.*WriteLine*("Next value is " + obj.GetNext());

}

}

}

}

使用接口类型的引用

接口类型的引用可以指向任何实现了接口的类对象。

使用接口类型引用访问方法时，类似基类引用指向派生类对象的情况。

例：程序interface\_test2

// Copyright 2016.刘珅珅

// author：刘珅珅

// 接口类型的引用

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace interface\_test2

{

public interface ISeries

{

int GetNext();

void Reset();

void SetStart(int x);

}

class ByTwos : ISeries

{

int start;

int val;

public ByTwos()

{

start = 0;

val = 0;

}

public int GetNext()

{

val += 2;

*Console*.*WriteLine*("ByTwos Next value " + val);

return val;

}

public void Reset()

{

val = start;

}

public void SetStart(int x)

{

start = x;

val = start;

}

}

class Primes : ISeries

{

int start;

int val;

public Primes()

{

start = 2;

val = 2;

}

public int GetNext()

{

val += 1;

*Console*.*WriteLine*("Primes Next value " + val);

return val;

}

public void Reset()

{

val = start;

}

public void SetStart(int x)

{

start = x;

val = start;

}

}

class InterfaceTest

{

static void Main(string[] args)

{

ByTwos two\_obj = new ByTwos();

Primes prime\_obj = new Primes();

ISeries interface\_obj;

interface\_obj = two\_obj;

interface\_obj.GetNext();

interface\_obj = prime\_obj;

interface\_obj.GetNext();

}

}

}

输出结果：

ByTwos Next value 2

Primes Next value 3

接口属性：

type name {

get;

set;

}

只有get为只读属性，只有set为只写属性。

接口索引器：

element-type this[int index]

{

get;

set;

}

例：程序interface\_test3

// Copyright 2016.刘珅珅

// author：刘珅珅

// 接口属性与索引器

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace interface\_test3

{

public interface ISeries

{

int Next

{

get;

set;

}

int this[int index]

{

get;

}

}

class ByTwos : ISeries

{

int val;

public ByTwos()

{

val = 0;

}

public int Next

{

get

{

val += 2;

return val;

}

set

{

val = value;

}

}

public int this[int index]

{

get

{

val = 0;

for (int i = 0; i < index; ++i)

{

val += 2;

}

return val;

}

}

}

class InterfaceTest

{

static void Main(string[] args)

{

ByTwos obj = new ByTwos();

*Console*.*WriteLine*("Next value is " + obj.Next);

obj.Next = 20;

for (int i = 0; i < 5; ++i)

{

*Console*.*WriteLine*("Next value is " + obj[i]);

}

}

}

}

接口继承：一个接口可以继承另一个接口，类在继承一个接口时，必须为接口继承链中定义的所有成员提供具体的实现。

例：interface\_test4

// Copyright 2016.刘珅珅

// author：刘珅珅

// 接口继承

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace interface\_test4

{

public interface IA

{

void Method1();

void Method2();

}

public interface IB : IA

{

void Method3();

}

class MyClass : IB

{

public void Method1()

{

*Console*.*WriteLine*("Implement Method1().");

}

public void Method2()

{

*Console*.*WriteLine*("Implement Method2().");

}

public void Method3()

{

*Console*.*WriteLine*("Implement Method3().");

}

}

class InterfaceTest

{

static void Main(string[] args)

{

MyClass obj = new MyClass();

obj.Method1();

obj.Method2();

obj.Method3();

}

}

}

接口的显式实现：在实现接口的成员时，可以使用接口名完全限定成员名。

显式实现接口方法时，不能通过类对象来访问该接口方法，必须通过接口引用来访问该方法。

如果类继承多个接口，如果不同接口有完全相同的方法，显式实现可以消除多义性。

例：程序interface\_test5

// Copyright 2016.刘珅珅

// author：刘珅珅

// 接口的显式实现

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace interface\_test5

{

public interface IEven

{

bool IsOdd(int x);

bool IsEven(int x );

}

class MyClass : IEven

{

// 显式实现接口方法

bool IEven.IsOdd(int x)

{

if ((x % 2) != 0)

{

return true;

}

return false;

}

public bool IsEven(int x)

{

IEven obj = this;

return !obj.IsOdd(x);

// return !this.IsOdd(x); // error不能通过类对象来访问

}

}

class InterfaceTest

{

static void Main(string[] args)

{

MyClass obj = new MyClass();

bool result = obj.IsEven(4);

if (result)

{

*Console*.*WriteLine*("4 is even");

}

// result = obj.IsOdd(4); // error，不能访问显式实现的接口方法

IEven iRef = (IEven)obj;

result = iRef.IsOdd(3);

if (result)

{

*Console*.*WriteLine*("3 is odd");

}

}

}

}

消除多义性：

例：程序interface\_test6

// Copyright 2016.刘珅珅

// author：刘珅珅

// 接口显式实现：消除多义性

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace interface\_test6

{

interface IMyIA

{

void Method(int x);

}

interface IMyIB

{

void Method(int x);

}

class MyClass : IMyIA, IMyIB

{

void IMyIA.Method(int x)

{

*Console*.*WriteLine*("IMyIA::Method " + (x + x));

}

void IMyIB.Method(int x)

{

*Console*.*WriteLine*("IMyIB::Method " + (x \* x));

}

}

class InterfaceTest

{

static void Main(string[] args)

{

MyClass obj = new MyClass();

IMyIA iA = (IMyIA)obj;

iA.Method(3);

IMyIB iB = (IMyIB)obj;

iB.Method(3);

}

}

}