在C#中，派生类只能继承一个基类（这点与C++不同）。

继承的定义：class A : B

// Copyright 2016.刘珅珅

// author：刘珅珅

// 类的继承

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace inherit\_class\_test1

{

class TwoDShape

{

double width\_;

double height\_;

public TwoDShape()

{

Width = Height = 0.0;

}

public TwoDShape(double x)

{

Width = Height = x;

}

public TwoDShape(double w, double h)

{

Width = w;

Height = h;

}

public double Width

{

get { return width\_; }

set { width\_ = value < 0 ? -value : value; }

}

public double Height

{

get { return height\_; }

set {height\_ = value < 0 ? -value : value;}

}

public void ShowDim()

{

Console.WriteLine("Width and height are " + Width + " and " + Height);

}

}

class Triangle : TwoDShape

{

string style\_;

public Triangle() // 会默认调用基类的默认构造函数

{

style\_ = "null";

}

public Triangle(double x) : base(x)

{

style\_ = "isosceles";

}

// 通过base来调用基类的构造函数

// 如果不通过base来调用，会自动调用默认构造函数

// 这样会产生不可预料的后果

public Triangle(string s, double w, double h) : base(w, h)

{

style\_ = s;

}

public double Area()

{

return Width \* Height / 2;

}

public void ShowStyle()

{

Console.WriteLine("Triangle is " + style\_);

}

}

class InheritClassTest

{

static void Main(string[] args)

{

Triangle t1 = new Triangle();

Triangle t2 = new Triangle("right", 8.0, 12.0);

Triangle t3 = new Triangle(4.0);

Console.WriteLine("Info for t1: ");

t1.ShowStyle();

t1.ShowDim();

Console.WriteLine("Area is " + t1.Area());

Console.WriteLine();

Console.WriteLine("Info for t2: ");

t2.ShowStyle();

t2.ShowDim();

Console.WriteLine("Area is " + t2.Area());

Console.WriteLine();

Console.WriteLine("Info for t3: ");

t3.ShowStyle();

t3.ShowDim();

Console.WriteLine("Area is " + t3.Area());

}

}

}

继承和名称隐藏：

派生类可以定义与基类中的成员同名的成员，此时，派生类的成员会隐藏基类的同名成员。可以在派生类成员前面加一个new，用来避免可能的警告。

同理，派生类也可以定义与基类中的方法同名的方法，同样基类的方法也会被隐藏。

例：程序inherit\_class\_test2

// Copyright 2016.刘珅珅

// author：刘珅珅

// 继承和名称隐藏

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace inherit\_class\_test2

{

class A

{

public int i = 0;

}

class B : A

{

// 派生类定义基类的同名成员

new int i;

public B(int b)

{

i = b; // i in B

}

public void Show()

{

Console.WriteLine("i in derived class: " + i);

// 可以通过base来访问基类被隐藏的同名成员

// i在基类中为public成员

Console.WriteLine("i in base class: " + base.i);

}

}

class InheritClassTest

{

static void Main(string[] args)

{

B obj = new B(2);

obj.Show();

}

}

}

在C#中基类的引用可以指向派生类对象，其性质与C++基类的引用或指针指向派生类的对象很相似。

虚方法和重写

使用virtual关键字，可以在基类中定义虚方法，在派生类中重新定义虚方法时，需要使用override修饰符。

也可以使用virtual在基类中定义属性和索引器，在派生类中使用override重写属性和索引器。

虚方法不能指定为static或abstract，构造函数也不能是virtual，这点也与C++类似。

在C#中，虚方法不能是private的，但在C++中，可以是private的，但私有的虚方法无法触发动态绑定，在编译时就会报错。

在C#中，基类声明的virtual方法的访问级别，在派生类中不能更改。即基类的virtual方法为public，派生类重写也必须为public。派生类的virtual方法为protected，派生类重写也必须为protected。在C++中，可以更改，可以出现基类的virtual方法为public，而派生类为private，而且这样也可以触发动态绑定，即基类的指针或引用绑定派生类对象后，调用virtual方法时，访问的是派生类的private虚方法，这其实不怎么安全。

例：程序inherit\_class\_test3

// Copyright 2016.刘珅珅

// author：刘珅珅

// 虚函数

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace inherit\_class\_test3

{

class Base

{

public virtual void Who()

{

Console.WriteLine("Who() in Base");

}

public virtual void In()

{

Console.WriteLine("In() in Base");

}

}

class Derived : Base

{

public override void Who()

{

Console.WriteLine("Who() in Derived");

}

}

class Derived2 : Derived

{

public override void In()

{

Console.WriteLine("In() in Derived2");

}

}

class InheritClassTest

{

static void Main(string[] args)

{

Base base\_obj = new Base();

Derived de\_obj = new Derived();

Derived2 de\_obj2 = new Derived2();

base\_obj.Who();

base\_obj.In();

Console.WriteLine();

Base base\_ref1 = de\_obj;

base\_ref1.Who();

base\_ref1.In();

Console.WriteLine();

Base base\_ref2 = de\_obj2;

base\_ref2.Who();

base\_ref2.In();

}

}

}

输出结果为：

Who() in Base

In() in Base

Who() in Derived

In() in Base

Who() in Derived

In() in Derived2

从输出结果可以看出：虚方法的调用与C++比较类似。

派生类Derived中重写了Who()，基类引用指向Derived，调用了是派生类的Who()；Derived中没有重写In()方法，其调用基类的In()方法。

Derived2继承了Derived，其没有重写Who()方法，所以调用Derived中的Who()方法，其重写了In()方法，所以顶层的基类引用调用的是Derived2的In()方法。