7/13

笔记本: 暑期实习
创建时间: 2021/7/13 13:16 **更新时间:** 2021/7/14 15:24
作者: 134exetj717
URL: http://wiki.suncaper.net/pages/viewpage.action?pageId=39855832

- 类的继承：C#的继承跟C++的非常类似，但是它新增了一个功能
    - 使用sealed修饰符来禁止继承，这样的类称为：密封类

```
sealed class 类名
{
    // ...
}
```

    - 派生类的构造函数会先从最远的基类的构造函数开始调用；
    - 相反，派生类的析构函数会先从最近的基类的析构函数开始调用
    - 派生类只能继承一个基类
- 多态性：同一签名（参数相同）具有不同的表现行为，运算符重载和函数重载都属于多态性的表现形式
    - 隐藏基类方法

        1. 用新的派生成员替换基成员
        2. 重写虚拟的基成员：在子类中编写有相同名称和参数的方法
            1. 重载：编写（同一个类中）具有相同的名称，却有不同的参数的方法。具有不同的签名。
            2. virtual关键字：表名允许在派生类中重写这些对象
            3. override声明重写的方法称为重写基方法

```
1.替换
class A
{
    public void fun()
    {
        Console.WriteLine("A");
    }
}
class B : A
{
    new public void fun() //隐藏基类方法fun
    {
        Console.WriteLine("B");
    }
}

在主函数中执行以下语句：
B b=new B();
b.fun();

运行结果如下：
  B

2.重写
using System;
namespace proj6_3
```

```
{
    class Student
    {
        protected int no;                //学号
        protected string name;      //姓名
        protected string tname;     //班主任或指导教师
        public void setdata(int no1, string name1, string tname1)
        {
            no = no1; name = name1; tname = tname1;
        }
        public virtual void dispdata() //虚方法
        {
            Console.WriteLine("本科生  学号:{0} 姓名:{1} 班 主 任:{2}", no, name,
tname);
        }
    }
    class Graduate : Student
    {
        public override void dispdata() //重写方法
        {
            Console.WriteLine("研究生  学号:{0} 姓名:{1} 指导教师:{2}", no, name,
tname);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Student s = new Student();
            s.setdata(101, "王华", "李量");
            s.dispdata();
            Graduate g = new Graduate();
            g.setdata(201, "张华", "陈军");
            g.dispdata();
        }
    }
}


如果父类的构造函数需要用到子类的构造函数的参数:
public Duck(string name, string symptom, int age, string illness) : base(name,
symptom,  age, illness) { }
```

- object类：C#中所有类型（包括所有的值类型和引用类型）的基类，C#中所有类型都直接或简介从object类中继承而来。因此，对一个object的变量可以赋予任何类型的值。
- dynamic类型，动态联编：该类型的变量只有在运行时才能被确定具体类型
    - 该类型的变量在被编译时会被当成object来对待，dynamic仅在编译期间存在，运行期间会被object类型替代
    - 尚未知道其作用?
- is 运算符：判断检查对象的类型，或者可以转换为给定的类型，返回Bool值
    - operand is type
- as运算符：在兼容的引用类型之间执行转换，类似于强制转换，但是若转换失败，会返回null值

```
operand as type
等效于
operand is type ? (type)operand : (type)null
```

- 抽象类：使用abstract修饰符的类
    - 抽象类相当于仅仅是创建了一个兼容性非常强的类，然后派生出的许多类必须包含抽象类中的属性和方法

```
using System;
namespace proj6_6
{  abstract class A                    //抽象类声明
     {     protected int x = 2;
            protected int y = 3;
            public abstract void fun();        //抽象方法声明
            public abstract int px { get;set; }    //抽象属性声明
            public abstract int py { get; }    //抽象属性声明
     }
      class B : A
      {    public override void fun()   //抽象方法实现
            {  x++;  y++;  }
            public override int px //抽象属性实现
            {   set
                     {   x = value;  }
                     get
                     {  return x + 10; }
            }
            public override int py //抽象属性实现
            {    get
                 {   return y + 10; }
            }
       }
       class Program
       {    static void Main(string[] args)
             {   B b = new B();
                      b.px = 5;
                      b.fun();
                      Console.WriteLine("x={0}, y={1}", b.px, b.py);
            }
       }
}
```

- interface，接口的使用：
  - 在接口中必须定义抽象函数或者抽象属性，否则默认失效；
  - 继承了接口的类必须实现接口中所有的方法；
  - 可以作为基类使用，来取得部分派生类的接口中的所需功能

```
class progress
    {
        interface student
        {
            public string sname     //抽象方法
            {
                get;
                set;
            }
        }
        interface course
        {
            public string cname
            {
                get;
                set;
            }
            public void printName();    //抽象方法
        }
        class score:student,course
        {
            int myScore;
            string ssname;      //类中的私有数据
            string ccname;
            public string sname     //利用访问器实现对私有数据的保护，或者触发
            {
                get { return ssname; }
                set { ssname = value; }
```

```
            }
            public string cname
            {
                get { return ccname; }
                set { ccname = value; }
            }
            public score(string sname,string cname,int myScore)
            {
                this.ssname = sname;
                this.ccname = cname;
                this.myScore = myScore;
            }
            public void print()
            {
                Console.WriteLine($"{ssname}\t{ccname}\t{myScore}\n");
            }
            public void printName()
            {
                Console.WriteLine($"{ccname}\n");
            }
        }
        public delegate void myprint();
        static void Main(string[] args)
        {
            score a = new score("小明", "语文", 96);
            a.print();
            myprint p;
            course c = a;       //抽象类相当于提供了一个容器，让派生类转换为基类，拿到
部分需要的内容，
            p = new myprint(c.printName);
            p();
        }
    }
```