**实验三 类的设计（2）继承和多态**

**实验内容（目的）：**

1. 练习类的重用设计语法，构建类的继承体系（可参考教材1.7节的内容）
2. 练习在继承体系的基础上设计多态性（参考教材1.11和1.12节的内容）

**实验前思考并回答：**

1. 继承技术中，派生类如何表现与基类的不同？派生类能否删除定义于基类的成员？

派生类可以从数据成员和函数成员两方面表现与基类的不同。数据成员方面可以通过增加数据成员。函数成员方面通过方法的覆写即override可以使基类中的抽象方法（abstract）和虚方法（virtual）在派生类中与基类表现不同。派生类不能删除定义于基类的成员。

。

1. （动态）多态性是指什么？多态技术能否独立于继承技术？

动态多态性指的是运行时的多态性，即在系统运行时，不同对象调用一个名字相同、参数类型和个数也完全相同的方法，会完成不同的操作。多态技术不能独立于继承，因为没有继承技术加持的多态技术只能完成编译时的多态性，即系统在编译时根据调用方法的实参类型和个数决定调用哪个同名方法，而不能实现运行时的多态，对于需要调用一个名字相同、参数类型和个数也完全相同的方法而且要根据对象的继承关系给出不同的输出的情景，只靠独立于继承的多态技术是无法实现的。

1. 如果没有（动态）多态的应用需求，继承技术的重用作用是否有绝对意义（即是否不可替代，若可替代，如何替代）

个人认为关于继承技术的重用作用并非具有绝对意义，继承是实现代码重用、扩展软件功能的重要手段，子类中与父类完全相同的属性和方法不必重写，只需写出新增或改写的内容，即子类可以复用父类的内容，不必一切从零开始。所以本人认为一个可行的替代方案是使用类的接口来实现原来继承的功能，我们可以在类中写好一些接口，用来共享数据成员和方法成员，当其他类需要“继承”这些的时候，只需要调用这些接口就好了，这个方法还可以实现类的“多继承”，即一个类可以“继承”来自不同类的数据和方法。

1. 实现多态设计的关键要素有两个，分别是什么？

多态是同一操作作用于不同的类的实例，将产生不同的执行结果，即不同类的对象收到相同的消息时，得到不同的结果。

**任务：**

1. 设计complex 类，用操作符重载实现复数的四则运算。

代码：

using System;

namespace complex

{

    class Complex {

        private double real;

        private double imag;

        public Complex(double x, double y) {

            real = x;

            imag = y;

        }

        static public Complex operator -(Complex left, Complex right) {

            return (new Complex(left.real - right.real, left.imag - right.imag));

        }

        static public Complex operator + (Complex left, Complex right) {

            return (new Complex(left.real + right.real, left.imag + right.imag));

        }

        public void display() {

            string sign = "+";

            if (imag < 0) sign = "-";

            Console.WriteLine("{0} {1} {2}i", real, sign ,imag);

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            Complex x = new Complex(1, 2);

            Complex y = new Complex(3, 4);

            Complex z = new Complex(1.5, 2.5);

            z.display();

            z += x;

            z.display();

            z -= y;

            z.display();

            z = x - y;

            z.display();

            z = x + y;

            z.display();

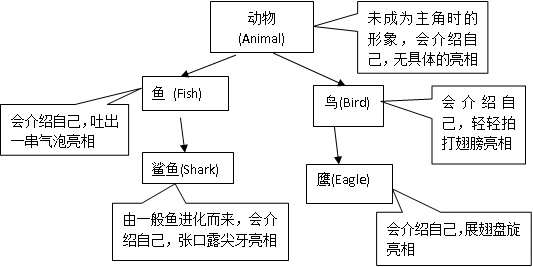
        }

    }

}

1. 考虑设计一个以动物为角色形象的游戏，动物出现时会介绍(Introduce())自己(名字(name)和年龄(age))，不同类型的动物设计有不同的标志性亮相(Show())，当前考虑的动物种类为鱼和鸟。游戏早期阶段，角色的亮相方式都较平淡，但经过一段时间进化为狠角色就会改变亮相方式 (如动物的分类图如下，同时给出各类型的行为说明) 。

请据此设计支持游戏的动物类家族，要求：（1）每个类要有合适的构造函数（2）使用多态技术设计动物的亮相行为。（3）写客户程序来测试游戏角色的行为表现（提示：自我介绍和亮相，以屏幕输出字串表示即可）



代码：

using System;

namespace Animal\_Character

{

    class Program

    {

        class Animal {

            private string name;

            private int age;

            public Animal(string nam, int ag) {

                name = nam;

                age = ag;

                introduce();

            }

            public void introduce() {

                Console.WriteLine("name: {0}\nage: {1}", name, age);

            }

            public virtual void show() { }

        }

        class Fish:Animal {

            public Fish(string nam, int ag) : base(nam, ag) {

                show();

            }

            public override void show() {

                Console.WriteLine("{0}", "吐出一串气泡亮相");

            }

        }

        class Shark:Fish {

            public Shark(string nam, int ag):base(nam, ag) { }

            public override void show() {

                Console.WriteLine("{0}", "张口露尖牙亮相");

            }

        }

        class Bird:Animal {

            public Bird(string nam, int ag):base(nam, ag) {

                show();

            }

            public override void show() {

                Console.WriteLine("{0}", "轻轻拍打翅膀亮相");

            }

        }

        class Eagle:Bird {

            public Eagle(string nam, int ag) : base(nam, ag) { }

            public override void show() {

                Console.WriteLine("{0}", "展翅盘旋亮相");

            }

        }

        static void Main(string[] args) {

            Animal newGuy = new Animal("unknow", 0);

            Fish newFish = new Fish("fish", 1);

            Shark newShark = new Shark("shark", 2);

            Bird newBird = new Bird("bird", 1);

            Eagle newEagle = new Eagle("eagle", 2);

        }

    }

}