## 实验八 掌握接口的实现

### 实验目的：

理解接口的意义

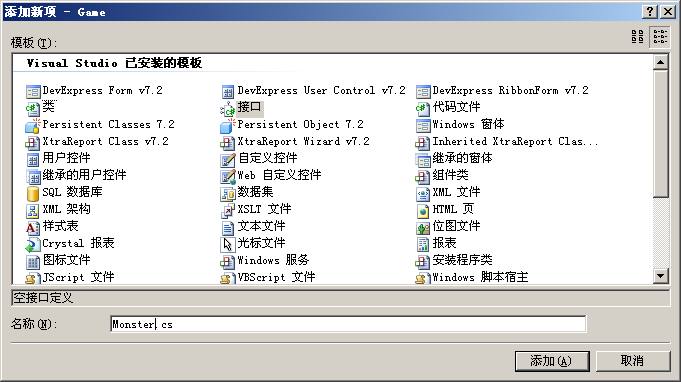
掌握接口的C#实现

### 第一部分 上机训练

1、在游戏编程中，需要对怪物的属性做统一的约定，例如所有的怪物都有生命、魔法、攻击、命中、防御、闪避等属性，所有的怪物都能攻击、站立、防御等，这些统一的约定可以做成接口，然后分别定义人类、兽类等来实现怪物接口。

1）创建名为Game的控制台应用程序

2）建立名为Monster的接口

3）定义该接口如下：

using System;

using System.Collections.Generic;

using System.Text;

namespace Game

{

public interface Monster

{

string Name

{

get;

set;

}

int HP

{

get;

set;

}

int MP

{

get;

set;

}

int Defense

{

get;

set;

}

int Attack

{

get;

set;

}

int Dodge

{

get;

set;

}

int Accuracy

{

get;

set;

}

void DoAttack();

void DoStand();

void DoDefense();

void DisplayInfo();

}

}

4）新建一个Human类，实现Monster接口

using System;

using System.Collections.Generic;

using System.Text;

namespace Game

{

class Human : Monster

{

private int healthPower; //生命

private int magicPower; //魔法

private int defense; //防御

private int attack; //攻击

private int dodge; //躲闪

private int accuracy; //命中

private string name;

public int HP

{

get { return healthPower; }

set { healthPower = value; }

}

public int MP

{

get { return magicPower; }

set { magicPower = value; }

}

public int Defense

{

get { return defense; }

set { defense = value; }

}

public int Attack

{

get { return attack; }

set { attack = value; }

}

public int Dodge

{

get { return dodge; }

set { dodge = value; }

}

public int Accuracy

{

get { return accuracy; }

set { accuracy = value; }

}

public string Name

{

get { return name; }

set { name = value; }

}

public void DoAttack()

{

Console.WriteLine(name + "的攻击动作很华丽!===!=========>");

}

public void DoStand()

{

Console.WriteLine(name + "站在那里，果然是伟岸英挺，气宇轩昂，简直是人中龙凤!");

}

public void DoDefense()

{

Console.WriteLine(name + "将手中兵器舞成一片，格开了所有人的进攻。");

}

public void DisplayInfo()

{

Console.WriteLine("姓名:" + name);

Console.WriteLine("种族: 人类");

Console.WriteLine("----------------------------------------------");

Console.WriteLine("状态");

Console.WriteLine("生命:{0}\t\t魔法:{1}", healthPower, magicPower);

Console.WriteLine("----------------------------------------------");

Console.WriteLine("战斗力");

Console.WriteLine("攻击:{0}\t\t防御:{1}", attack, defense);

Console.WriteLine("命中:{0}\t\t\t闪避:{1}", accuracy, dodge);

}

}

}

5）新建一个Beast类，实现Monster接口

using System;

using System.Collections.Generic;

using System.Text;

namespace Game

{

public class Beast : Monster

{

private int healthPower; //生命

private int magicPower; //魔法

private int defense; //防御

private int attack; //攻击

private int dodge; //躲闪

private int accuracy; //命中

private string name;

public int HP

{

get { return healthPower; }

set { healthPower = value; }

}

public int MP

{

get { return magicPower; }

set { magicPower = value; }

}

public int Defense

{

get { return defense; }

set { defense = value; }

}

public int Attack

{

get { return attack; }

set { attack = value; }

}

public int Dodge

{

get { return dodge; }

set { dodge = value; }

}

public int Accuracy

{

get { return accuracy; }

set { accuracy = value; }

}

public string Name

{

get { return name; }

set { name = value; }

}

public void DoAttack()

{

Console.WriteLine(name + "的爪子狠狠的一抓，你的头皮去了好大一块!");

}

public void DoStand()

{

Console.WriteLine(name + "四脚着地，目露凶光，你看到它时不寒而栗!");

}

public void DoDefense()

{

Console.WriteLine(name + "向你大声咆哮，吓得你抱头鼠窜。");

}

public void DisplayInfo()

{

Console.WriteLine("姓名:" + name);

Console.WriteLine("种族: 四足兽类");

Console.WriteLine("----------------------------------------------");

Console.WriteLine("状态");

Console.WriteLine("生命:{0}\t\t魔法:{1}", healthPower, magicPower);

Console.WriteLine("----------------------------------------------");

Console.WriteLine("战斗力");

Console.WriteLine("攻击:{0}\t\t防御:{1}", attack, defense);

Console.WriteLine("命中:{0}\t\t\t闪避:{1}", accuracy, dodge);

}

}

}

6）在Program.cs的Main方法中输入如下代码：

using System;

using System.Collections.Generic;

using System.Text;

namespace Game

{

class Program

{

static void Main(string[] args)

{

Monster player = new Beast();

player.Name = "天一";

player.HP = 200;

player.MP = 300;

player.Accuracy = 90;

player.Attack = 1000;

player.Defense = 1000;

player.Dodge = 80;

player.DoAttack();

player.DoStand();

player.DisplayInfo();

}

}

}

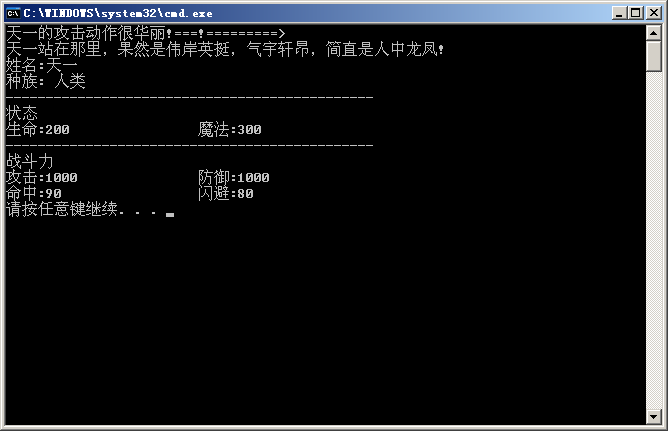
7）按Ctrl+F5运行：



再将Monster player = new Beast();

改为Monster player = new Human();

再次按Ctrl + F5运行，体会接口的作用。



### 第二部分 实战提高

1）定义Student类，用string型变量name存储学生姓名，用int型变量age存储学生年龄。Student类实现IComparable接口。要求从键盘输入学生的姓名和年龄，并注意可能出现的异常及其处理。IComparable接口定义如下（系统已定义，可直接使用）

interface IComparable

{

int CompareTo(object obj);

//如果自身与obj相等返回0，<obj返回-1，>obj返回1 ，注意可能出现的异常

}

（2）定义Student类的派生类Master。

（3）定义Sort类，定义静态方法BubbleSortDescending（IComparable[] bubbles），实现对象的降序排列。其中，在该方法中调用接口方法CompareTo（object obj）比较两个对象的“大小”。

（4）定义Test类，按给定数据生成Student实例数组，调用BubbleSortDescending（IComparable[] bubbles）使之按姓名排序，按给定数据生成Master实例数组，调用BubbleSortDescending（IComparable[] bubbles）使之按年龄排序，请遍历输出排序前后数组中每个元素实例中的name和age。如果Master实例a与Student实例b比较，总是a>b

（5）实验数据

|  |  |
| --- | --- |
| **Name** | **Age** |
| Tom | 18 |
| Nick | 20 |
| Mike | 17 |

（a）当这三个学生全为Student的实例时，输出排序后的结果；

（b）当这三个学生全为Master类的实例时，输出排序后的结果

（c）(选作)任意指定数组中元素的类型（Student或Master），输出排序后的结果

**代码和运行结果图请表示在下面：**

代码如下：

#IComparable.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MySoftwareOnDotnetFramework.Comparable

{

public interface IComparable

{

int CompareTo(object obj);

void display();

}

}

#Student.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MySoftwareOnDotnetFramework.Comparable

{

public class Student:IComparable

{

private string name;

private int age;

public Student() { }

public Student(string name,int age)

{

this.name = name;

this.age = age;

}

public string Name

{

get

{

return name;

}

set

{

name = value;

}

}

public int Age

{

get

{

return age;

}

set

{

age = value;

}

}

public virtual int CompareTo(object obj)//对接口的方法进行重写

{

if (obj.GetType() == typeof(Master))

{

return 1;

}

Student other = (Student)obj;

if (other != null)

{

if (string.Compare(name, other.Name) < 0) return 1;

else if (string.Compare(name, other.name) == 0) return 0;

else return -1;

}

else

throw new ArgumentException("Not a Student");

}

public virtual void display()

{

Console.WriteLine("{0}----{1}", name, age);

}

}

}

#Master.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MySoftwareOnDotnetFramework.Comparable

{

public class Master:Student

{

private string name;

private int age;

public Master() { }

public Master(string name, int age):base(name,age)

{

this.name = name;

this.age = age;

}

public new string Name

{

get

{

return name;

}

set

{

name = value;

}

}

public new int Age

{

get

{

return age;

}

set

{

age = value;

}

}

public override int CompareTo(object obj)//override 对student里的compareto方法进行重写

{

if (obj.GetType() == typeof(Student))

{

return -1;

}

Master other = (Master)obj;//强制类型转换

if (other != null)//判断是否空，异常

{

if (age < other.Age) return 1;

else if (age == other.Age) return 0;

else return -1;

}

else

throw new ArgumentException("Not a Master");

}

public override void display()

{

Console.WriteLine("{0}----{1}", name, age);

}

}

}

#Sort.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MySoftwareOnDotnetFramework.Comparable

{

public class Sort

{

static IComparable g;//在静态函数里面的变量也要用静态

public static void BubbleSortDescending(IComparable[] bubbles)

//因为这里student类继承icomparable,而master继承student,所以这里要用ICmparable

{

for(int i= 0; i < bubbles.Length - 1; i++)

{

for(int j = 0; j < bubbles.Length - 1; j++)

{

if (bubbles[j].CompareTo(bubbles[j + 1]) > 0)

{

g = bubbles[j];

bubbles[j] = bubbles[j + 1];

bubbles[j + 1] = g;

}

}

}

}

}

}

#Program.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace MySoftwareOnDotnetFramework.Comparable

{

class Program

{

static void Main(string[] args)

{

IComparable[] array = new Student[3];//用了多态，因为Student继承IComparable，所以可以这样写

array[0] = new Student("Tom", 18);//c初始化，也是要用new

array[1] = new Student("Nike", 20);

array[2] = new Student("Mike", 17);

Sort.BubbleSortDescending(array);

Console.WriteLine("按姓名降序排序：");

foreach (Student stu in array)

{

stu.display();

}

array = new Master[3];

array[0] = new Master("Tom", 18);

array[1] = new Master("Nike ", 20);

array[2] = new Master("Mike", 17);

Sort.BubbleSortDescending(array);

Console.WriteLine("\n按年龄降序排序：");

foreach (Master mas in array)

{

mas.display();

}

array = new IComparable[3];

array[0] = new Student("Tom", 18);

array[1] = new Student("Nike ", 20);

array[2] = new Master("Mike", 17);

Sort.BubbleSortDescending(array);

Console.WriteLine("\n混合降序排序：");

foreach (IComparable ic in array)

{

ic.display();

}

Console.ReadLine();

}

}

}

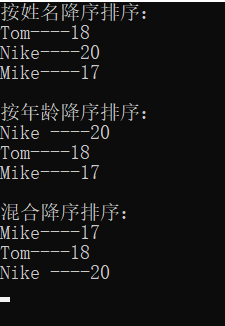


图8-1 实现了3种排序，包含Student、Master和混合类型排序

### 第三部分 知识总结

|  |  |
| --- | --- |
| **学生总结** | 请将问题答案写在下面空白处 |
| 完成上机训练模块花费的时间 | 10分钟 |
| 完成实战提高模块花费的时间 | 50分钟 |
| 本次实验对你而言的难点 | 数组类型问题，使用IComparable虽然可以接受两种类型，但是无法显示，于是我在接口中也定义了显示函数 |
| 本次实验的收获 | 面向接口编程的扩展性和灵活性 |
| 希望老师强化讲解的知识要点 |  |
| 希望老师帮助回答的问题 |  |
| 本次实验自评分数  （五分制） | 5 |