# 实验五 结构体、接口编程练习

## 【实验目的】

1. 了解结构体的概念，理解结构体和类相比的异同，掌握结构体的使用方法；
2. 了解什么是接口，接口和抽象类的异同，掌握接口的方法实现，接口方法的重定义。

## 【实验要求】

1. 写出能够使用结构体的程序；
2. 写出使用接口实现的程序；
3. 写出程序，并调试程序，要给出测试数据和实验结果。
4. 整理上机步骤，总结经验和体会。
5. 完成实验日志和上交程序。

## 【实验内容】

### 一、接口的定义与作用

接口可以看作是没有实现的方法和常量的集合。接口与抽象类相似，接口中的方法只是做了声明，而没有定义任何具体的操作方法。使用接口是为了解决C# 语言中不支持多重继承的问题。单继承可使语言本身结构简单，层次清楚，易于管理，安全可靠，避免冲突。但同时限制了语言的功能。为了在实际应用中实现多重继承的功能，C# 使用了接口技术，一个类可以实现多个接口以达到使用公用常量和一些常用的方法。

### 二、分析实现接口的程序文件

分析以下实现接口的程序文件并回答问题：

- 本程序中的接口包含方法的构成是哪些；
  int CompareTo(IComparable comp)
- 实现接口的类包含哪些元素？
  构造器，字段，属性，方法，重写接口的方法
- 类实现接口方法的参数如何变换实现的？
  强制转换
- 给出程序的输出结果。

代码如下：

```
PS C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5> dotnet ru
n
I spent more time than the world record holder
```

```
using System;

public interface IComparable
{
    int CompareTo(IComparable comp);
}

public class TimeSpan : IComparable
{
    private uint totalSeconds;

    public TimeSpan()
    {
        totalSeconds = 0;
    }

    public TimeSpan(uint initialSeconds)
    {
        totalSeconds = initialSeconds;
    }

    public uint Seconds
    {
        get
        {
            return totalSeconds;
        }

        set
        {
            totalSeconds = value;
        }
    }

    public int CompareTo(IComparable comp)
    {
        TimeSpan compareTime = (TimeSpan) comp;

        if(totalSeconds > compareTime.Seconds)
            return 1;
        else if(compareTime.Seconds == totalSeconds)
```

```
            return 0;
        else
            return -1;
    }
}

class Tester
{
    public static void Main()
    {
        TimeSpan myTime = new TimeSpan(3450);
        TimeSpan worldRecord = new TimeSpan(1239);

        if(myTime.CompareTo(worldRecord) < 0)
            Console.WriteLine("My time is below the world record");
        else if(myTime.CompareTo(worldRecord) == 0)
            Console.WriteLine("My time is the same as the world record");
        else
            Console.WriteLine("I spent more time than the world record holder");
    }
}
```

## 三、覆盖虚接口程序

以下程序组合了多种功能，请参考如下代码解释并回答问题。

- 该程序包含的类的个数和关系？
  5 个，IComparable 接口，TimeSpan，TimeSpanAdvanced，Sorter，Tester
  其中 TimeSpan 实现 IComparable 接口，TimeSpanAdvanced 继承 TimeSpan，并重写 IComparable 接口，Sorter 是一个工具类，用于排序，Tester 类是程序的入口
- 类对接口的实现有何区别？
  抽象类可以不实现接口，类实现接口有两种方式，一种是显式实现，一种是隐式实现，但无论怎样，对于外界都是可见的。
- 第一个类中无参数构造函数是否起作用，是否可以删除不用？
  不起作用，可以删除，应为外部创建时没有使用该构造函数。
- 类中的属性在哪里被应用到？
  在实现接口的方法中使用到。
- 第一个类中哪些成员被继承，列出所有？
  int CompareTo(IComparable other)
- 第二个类中构造方法成员如何实现，有何意义？可以去掉么？
  定义一个单参数的构造函数，一种生命，不可以去掉。
- 第二个类覆盖第一个类中接口虚方法，程序体现了什么功能区别？
  没有区别。
- Sorter 类有何作用？你能否根据 Sorter 类写一个十个数比较大小的冒泡法程序？

进行排序，可以，只需要实现 IComparable 接口便可以。

```
using System;

public interface IComparable
{
    int CompareTo(IComparable comp);
}

class Number : IComparable
{
    public Number(int value) {
        Value = value;
    }
    public int Value { get; set; }
    public int CompareTo (IComparable comp){
        return Value - ((Number)comp).Value;
    }

    public static  implicit operator Number (int value) {
        return new Number(value);
    }
}

class Sorter
{
    // Sort the comparable elements of an array in ascending order

    public static void BubbleSortAscending(IComparable[] bubbles)
    {
        bool swapped = true;

        for (int i = 0; swapped; i++)
            //for (int i = 0; i< bubbles.Length; i++)
        {
            Console.WriteLine("run outter");
            swapped = false;
            for (int j = 0; j < (bubbles.Length - (i + 1)); j++)
            {
                if (bubbles[j].CompareTo(bubbles[j + 1]) > 0)
                {
                    Console.WriteLine("run inner");
                    Swap(j, j + 1, bubbles);
                    swapped = true;
                }
```

```
            }
        }
    }

    //Swap two elements of an array
    public static void Swap(int first, int second, IComparable[] arr
)
    {
        IComparable temp;

        temp = arr[first];
        arr[first] = arr[second];
        arr[second] = temp;
    }
}



class Tester
{
    public static void Main()
    {
        Number[] numbers =  {
            110, 114, 76, 54, 142, 12, 9, 65, 88, 87
        };

        foreach (var number in numbers)
        {
            Console.WriteLine(number.Value);
        }

        Sorter.BubbleSortAscending(numbers);

        Console.WriteLine("List of sorted numbers:");
        foreach (var number in numbers)
        {
            Console.WriteLine(number.Value);
        }
        Console.ReadLine();
    }
}
```

- Sorter 类中 for (int i = 0; swapped; i++)和 //for (int i = 0; i< bubbles.Length; i++)两行是否作用相同？

不同， 使用 swapped 可以提前终止冒泡排序。

- 你知道 Console.WriteLine("run outter");和 Console.WriteLine("run inner");在程序运行过程中可以起到什么作用？
  起到提示的作用。
- 将 Main 方法中的 TimeSpan 对象语句（注释掉的 5 行）和 TimeSpanAdvanced 对象语句选择轮流注释，体验排序结果的异同。
  没有区别。
- 语句 Sorter.BubbleSortAscending(raceTimes);前后的 foreach 语句功能区别。
  第一个是打印输入的数据，第二个是打印排序后的数据。

```
PS C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5> dotnet run
153
165
142
108
run outter
run inner
run inner
run outter
run inner
run inner
run outter
run inner
run outter
List of sorted time spans:
108
142
153
165
```

```
using System;

public interface IComparable
{
    int CompareTo(IComparable comp);
}

public class TimeSpan : IComparable
{
    private uint totalSeconds;

    public TimeSpan()
    {
        totalSeconds = 0;
    }

    public TimeSpan(uint initialSeconds)
    {
        totalSeconds = initialSeconds;
    }

    public uint Seconds
    {
```

```
        get
        {
            return totalSeconds;
        }

        set
        {
            totalSeconds = value;
        }
    }

    public virtual int CompareTo(IComparable comp)
    {
        TimeSpan compareTime = (TimeSpan)comp;

        if (totalSeconds > compareTime.Seconds)
            return 1;
        else if (compareTime.Seconds == totalSeconds)
            return 0;
        else
            return -1;
    }
}
public class TimeSpanAdvanced : TimeSpan
{
    public TimeSpanAdvanced(uint initialSeconds): base(initialSeconds)
    {
        //
    }
    public override int CompareTo(IComparable comp)
    {
        TimeSpan compareTime = (TimeSpan)comp;

        if (base.Seconds > compareTime.Seconds)
        {
            if (base.Seconds > (compareTime.Seconds + 50))
                return 2;
            else
                return 1;
        }
        else if (base.Seconds < compareTime.Seconds)
        {
            if (base.Seconds < (compareTime.Seconds - 50))
                return -2;
```

```
            else
                return -1;
        }
        else
            return 0;
    }
}


class Sorter
{
    // Sort the comparable elements of an array in ascending order
    public static void BubbleSortAscending(IComparable[] bubbles)
    {
        bool swapped = true;

        for (int i = 0; swapped; i++)
            //for (int i = 0; i< bubbles.Length; i++)
        {
            Console.WriteLine("run outter");
            swapped = false;
            for (int j = 0; j < (bubbles.Length - (i + 1)); j++)
            {
                if (bubbles[j].CompareTo(bubbles[j + 1]) > 0)
                {
                    Console.WriteLine("run inner");
                    Swap(j, j + 1, bubbles);
                    swapped = true;
                }
            }
        }
    }

    //Swap two elements of an array
    public static void Swap(int first, int second, IComparable[] arr)
    {
        IComparable temp;

        temp = arr[first];
        arr[first] = arr[second];
        arr[second] = temp;
    }
}
```

```
class Tester
{
    public static void Main()
    {
        //TimeSpan[] raceTimes = new TimeSpan[4];
        //raceTimes[0] = new TimeSpan(153);
        //raceTimes[1] = new TimeSpan(165);
        //raceTimes[2] = new TimeSpan(142);
        //raceTimes[3] = new TimeSpan(108);

        TimeSpanAdvanced[] raceTimes = new TimeSpanAdvanced[4];
        raceTimes[0] = new TimeSpanAdvanced(153);
        raceTimes[1] = new TimeSpanAdvanced(165);
        raceTimes[2] = new TimeSpanAdvanced(142);
        raceTimes[3] = new TimeSpanAdvanced(108);

        foreach (TimeSpan time in raceTimes)
        {
            Console.WriteLine(time.Seconds);
        }

        Sorter.BubbleSortAscending(raceTimes);

        Console.WriteLine("List of sorted time spans:");
        foreach (TimeSpan time in raceTimes)
        {
            Console.WriteLine(time.Seconds);
        }
        Console.ReadLine();
    }
}
```

## 四、结构体的使用

1.程序功能要求，创建三个结构体，MyCircle,MyCylinder,MyCone分别表示圆形、圆柱体和圆锥体，MyCircle包含一个int类型的成员r表示半径，MyCylinder和MyCone各自包含一个MyCircel类型的成员表示圆柱体和圆锥体的底面，成员h和volumn（都为整型）分别表示圆柱体和圆锥体的高和体积。写出结构体和程序的主方法求圆柱体和圆锥体的体积。

```
struct MyCircle
    {
       public int r;
    }
    struct MyCylinder
```

```
{
   public MyCircle c;
   public int h;
   public  int volumn;
}
struct MyCone
{
    public MyCircle c;
    public int h;
    public int volumn;
}
public class Tester
{
    public static void Main()
    {
        Console.Write("请输入底面半径：");
        MyCircle c = new MyCircle();
        c.r = int.Parse(Console.ReadLine());
        Console.Write("请输入圆柱体高度：");
        MyCylinder cy = new MyCylinder();
        cy.h = int.Parse(Console.ReadLine());
        cy.c = c;
        Console.Write("请输入圆锥体高度：");
        MyCone co = new MyCone();
        co.h = int.Parse(Console.ReadLine());
        co.c = c;
       //计算圆柱体体积
        double x = Math.PI * cy.c.r * cy.c.r;
        double y = x * cy.h;
        cy.volumn = (int)y;
        //计算圆锥体体积
        double x2 = Math.PI * co.c.r * co.c.r;
        double y2 = x2 * co.h/3;
        co.volumn = (int)y2;
        //输出结果
       // Console.WriteLine("圆柱体的体积为：", cy.volumn);
        Console.Write("圆柱体的体积为：");
        Console.Write(cy.volumn);
        Console.Write("圆锥体的体积为：");
        Console.Write(co.volumn);
        Console.ReadLine();
        }
}
```

```
PS C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5> dotnet run
请输入底面半径：10
请输入圆柱体高度：10
请输入圆锥体高度：10
圆柱体的体积为：3141圆锥体的体积为：1047
```

问题：

（1）若取消// Console.WriteLine("圆柱体的体积为：", cy.volumn);前的注释替代分开输出的写法，看输出结果何变化，分析原因。

```
PS C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5> dotnet run
请输入底面半径：10
请输入圆柱体高度：10
请输入圆锥体高度：10
圆柱体的体积为：
圆锥体的体积为：1047
```

没有输出对应的体积，原因是因为少了占位符。

修改为如下代码即可

```
Console.WriteLine("圆柱体的体积为：{0}", cy.volumn);
```

（2）若将结构体中的变量不加public编译能通过么？为什么？

不可以，因为结构体的变量成员默认访问属性为private，这样在外界便不能访问。

```
PS C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5> dotnet run
P3.cs(25,11): error CS0122: '“MyCircle.r”不可访问，因为它具有一定的保护级别 [C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5\exp5.csproj]
P3.cs(35,35): error CS0122: '“MyCircle.r”不可访问，因为它具有一定的保护级别 [C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5\exp5.csproj]
P3.cs(35,44): error CS0122: '“MyCircle.r”不可访问，因为它具有一定的保护级别 [C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5\exp5.csproj]
P3.cs(39,36): error CS0122: '“MyCircle.r”不可访问，因为它具有一定的保护级别 [C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5\exp5.csproj]
P3.cs(39,45): error CS0122: '“MyCircle.r”不可访问，因为它具有一定的保护级别 [C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5\exp5.csproj]
```

（3）public的变量在Main中的使用方法。

变量.成员的方式访问。

（4）不用new创建结构体将MyCircle c = new MyCircle();MyCylinder cy = new MyCylinder();MyCone co = new MyCone();改成什么样的语句起到同样的效果。

创建结构体必须使用new创建。

```
public class Tester
{
    public static void Main()
    {
        Console.Write("请输入底面半径：");
        //MyCircle c = new MyCircle();
        //c.r = int.Parse(Console.ReadLine());
        int r = int.Parse(Console.ReadLine());
        Console.Write("请输入圆柱体高度：");
        //MyCylinder cy = new MyCylinder();
        //cy.h = int.Parse(Console.ReadLine());
        //cy.c = c;
        int h1 = int.Parse(Console.ReadLine());
        Console.Write("请输入圆锥体高度：");
        //MyCone co = new MyCone();
        //co.h = int.Parse(Console.ReadLine());
```

```
        //co.c = c;
        int h2 = int.Parse(Console.ReadLine());
        //计算圆柱体体积
        double x = Math.PI * r * r;
        double y = x * h1;
        int volumn1 = (int)y;
        //计算圆锥体体积
        double x2 = Math.PI * r * r;
        double y2 = x2 * h2/3;
        int volumn2 = (int)y2;
        //输出结果
        Console.WriteLine("圆柱体的体积为：{0}", volumn1);
        //Console.Write("圆柱体的体积为：");
        //Console.Write(cy.volumn);
        Console.Write("圆锥体的体积为：");
        Console.Write(volumn2);
        Console.ReadLine();
    }
}
```

（5）结构体中内嵌结构体实例的应用总结。
在结构体中，可以内嵌类或者结构体，结构体初始化时，所有的成员都会赋值初始值，若成员为类对象， 则默认值为null，否则，为值对象的初始值。
（6）和类相比结构体适合什么样的情况。
数据量小，要求效率比较高，经常用于局部的计算的场合。

## 五、结构体实现

实现一个结构体Fraction,它包含两个int类型私有数据成员numerator和demominator。让Fraction包含以下元素：

- 一个带两个参数的构造函数来初始化numerator和demominator；
- 用于存取numerator和demominator的属性；
- 一个名叫value的属性，返回一个分数值，分数由(numerator/demominator)计算而得；
- 覆盖ToString方法，返回下面的字符串“Fraction value:xxx”，其中的xxx是以字符串表示的分数值；
- 编写适当的代码来测试此Fraction结构。

参考代码如下：

```
using System;
struct Fraction
{
    private int numerator;
    private int denominator;

    public Fraction(int initNumerator, int initDenominator)
```

```
    {
        numerator = initNumerator;
        denominator = initDenominator;
    }

    public int Numerator
    {
        get
        {
            return numerator;
        }

        set
        {
            numerator = value;
        }
    }

    public int Denominator
    {
        get
        {
            return denominator;
        }

        set
        {
            denominator = value;
        }
    }

    public double Value
    {
        get
        {
            return (double)numerator / (double)denominator;
        }
    }

    public override string ToString()
    {
        string returnString;

        returnString = "Fraction value: " + Value;
```

```
        return returnString;
    }
}

class Tester
{
    public static void Main()
    {
        Fraction myFraction = new Fraction(1, 3);
        Console.WriteLine(myFraction);
    }
}
```

```
PS C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5> dotnet run
Fraction value: 0.3333333333333333
```

## 六、分析代码结果，总结结构体和类在值类型和引用类型上的区别。

```
using System;
public struct TimeSpan
{
    private uint totalSeconds;

    public TimeSpan(uint initialTotalSeconds)
    {
        totalSeconds = initialTotalSeconds;
    }

    public uint Seconds
    {
        get
        {
            return totalSeconds;
        }

        set
        {
            totalSeconds = value;
        }
    }
}

class Tester
{
```

```
    public static void Main()
    {
        TimeSpan myTime = new TimeSpan(480);

        UpdateTime(myTime);
        Console.WriteLine("Time outside UpdateTime method: {0}", myTime.Seconds);
    }

    public static void UpdateTime(TimeSpan timeUpdate)
    {
        timeUpdate.Seconds = timeUpdate.Seconds + 50;
        Console.WriteLine("Time inside UpdateTime method: {0}",
timeUpdate.Seconds);
    }
}
```

使用结构体传参的结果

```
PS C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5> dotnet run
Time inside UpdateTime method: 530
Time outside UpdateTime method: 480
```

使用类传参的结果

```
PS C:\Users\cht\Program\Widgets\homework.gallery\computer science\csharp\exp5> dotnet run
Time inside UpdateTime method: 530
Time outside UpdateTime method: 530
```

区别：结构体传参时，总是会拷贝一份值再传入，因此内部的更改将不会影响外面的对象。而类传参时，将传递类的引用，由于内部和外面的对象都表示同一个对象，因此内部的修改会改变外面的对象。