IComparable、IComparer接口、Comparer类

IComparable和ICompare 接口是.net framework 中比较对象的标准方式，这两个接口之间的区别如下：

1. IComparable 在要比较的对象的类中实现，可以比较该对象和另一个对象。
2.IComparer 在一个单独的类中实现，可以比较任意两个对象。
一般情况下，我们使用 IComparable 给出类的默认比较代码，使用其他类给出非默认的比较代码。

一、IComparable提供了一个方法int CompareTo(object obj)。这个方法接受一个对象，所以可以实现这个接口
比如：以便把 Person 对象传送给它，
说明这个人是否比当前的人更年老或年轻。实际上，这个方法返回一个int，所以可和下面的代码说明第二个人更年老还是更年轻。

```
if(person1.CompareTo(person2) == 0)
{
   Console.WriteLine("Same age");
}
else if(person1.CompareTo(person2) > 0 )
{
   Console.WriteLine("person 1 is older");
}
else
{
   Console.WriteLine("person1 is younger");
}
```

二、IComparer 也提供了一个方法 Compare().这个方法接受两个对象，返回一个整型结果，这与 Compar
eTo()相同。
对于支持 IComparer的对象，可以使用下面的代码：

```
if(personComparer.Compare(person1,person2) == 0)
{
   Console.WriteLine("same age");
}
else if(personComparer.Compare(person1,person2) > 0 )
{
```

```
            Console.WriteLine("person 1 is older");
        }
        else
        {
            Console.WriteLine("person1 is younger");
        }
```

在这两种情况下，提供给方法的参数是system.object类型。也就是说，可以比较任意类型的两个对象。

所以，在返回结果之前，通常需要进行某种类型比较，

如果使用了错误的类型，还会抛出异常。实际上，我们是使用泛型接口IComparable<T>，可以省略强转换。

三、.net framework 在类 Comparer 上提供了 IComparer 接口的默认实现方式，类 Comparer 位于 system.collections 名称空间中，可以对简单类型以及支持 IComparable 接口的任意类型进行特定文化的比较。例如，可以通过下面的代码使用它：

```
        string firststring = "First String";
        string secondstring = "Second string";
        Comparer.Default.Compare(firststring , secondstring);

        int firstNumber = 35;
        int secondNumber = 23;
        Comparer.Default.Compare(firstNumber , secondNumber);
```

这里使用Comparer.Default静态成员获取Comparer类的一个实例，接着使用 Compare()方法来比较。

在使用 Comparer时，必须使用可以比较的类型。例如，试图比较 firstString 和 firstNumber 就会生成一个异常。

- - - - - - - - - - - - - - - - - - - -

List<T>.sort()可以实现对T的排序，比如List<int>.sort()执行后集合会按照int从小到大排序。

如果T是一个自定义的Object，可是我们想按照自己的方式来排序，那该怎么办呢，

其实可以用过IComparable接口重写CompareTo方法来实现。流程如下：

一.第一步我们申明一个类Person但是要继承IComparable接口：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
namespace TestIComparable
{
    public class Person : IComparable<Person>
    {
        public string Name { get; set; }
        public int Age { get; set; }
        public int CompareTo(Person obj)
        {
            int result;
            if (this.Name == obj.Name && this.Age == obj.Age)
            {
                result = 0;
            }
            else
            {
                if (this.Name.CompareTo(obj.Name) > 0)
                {
                    result = 1;
                }
                else if (this.Name == obj.Name && this.Age > obj.Age)
                {
                    result = 1;
                }
                else
                {
                    result = -1;
                }
            }
            return result;
        }
        public override string ToString()
        {
            return this.Name + "-" + this.Age;
        }
    }
}
```

二.然后在主函数里面调用sort方法即可.类就会按照姓名从小到大，如果姓名相同则按照年龄
小到大排序了。

```
public class Program
{
    public static void Main(string[] args)
    {
        List<Person> lstPerson = new List<Person>();
        lstPerson.Add(new Person(){ Name="Bob",Age=19});
```

```
            lstPerson.Add(new Person(){ Name="Mary",Age=18});
            lstPerson.Add(new Person() { Name = "Mary", Age = 17 });
            lstPerson.Add(new Person(){ Name="Lily",Age=20});
            lstPerson.Sort();
            Console.ReadKey();
        }
    }
```

三，如果不继承IComparable接口，我们该如何实现排序呢。可以使用Linq来实现。
其实效果是一样的，只是如果类的集合要经常排序的话，建议使用继承接口的方法，这样可以简化sort的代码，而且更容易让人看懂。

```
    public static void Main(string[] args)
            {
                List<Person> lstPerson = new List<Person>();
                lstPerson.Add(new Person(){ Name="Bob",Age=19});
                lstPerson.Add(new Person(){ Name="Mary",Age=18});
                lstPerson.Add(new Person() { Name = "Mary", Age = 17 });
                lstPerson.Add(new Person(){ Name="Lily",Age=20});
                lstPerson.Sort((x,y) =>
                {
                    int result;
                    if (x.Name == y.Name && x.Age == y.Age)
                    {
                        result = 0;
                    }
                    else
                    {
                        if (x.Name.CompareTo(y.Name) > 0)
                        {
                            result = 1;
                        }
                        else if (x.Name == y.Name && x.Age > y.Age)
                        {
                            result = 1;
                        }
                        else
                        {
                            result = -1;
                        }
                    }
                    return result;
                });
                Console.ReadKey();
            }
```