

万物有序……



对象比较

北京理工大学计算机学院
金旭亮

第一名还是第二名，影响很大的……

在现实生活中，我们经常需要按照某种标准给事物“排序”，比如按照考试成绩给学生排出名次。

软件世界中的“对象”是对现实生活中各种“事物”的一种“模拟”，因此，也有对多个对象进行“排序”的要求，这样才能更贴近于现实世界的真实场景。

要对多个对象进行排序，就必须解决两个对象间的大小比较问题。

怎样确定对象的“大小”？

- 几乎所有的编程语言，对于数值类型（比如整数或浮点数），都已经支持大小比较。
- 但那些对于我们自己设计出来的类所创建的对象，怎样定义它们的“大小”特性？

关键是制定出确定对象大小的**比较规则**！

C#中，实现了**IComparable**接口的对象可以相互比较大小：

```
public interface IComparable
{
    int CompareTo(object obj);
}
```



CompareTo()方法用于定义对象的“**比较规则**”，其返回值表明了比较结果


```
int result = One.CompareTo(Other);
```

CompareTo方法返回值	含义
小于零 (-1)	One对象小于Other对象
零	One对象等于 Other对象
大于零 (1)	One对象大于 Other对象

如果只需要知道两个对象是否相等，而不需要知道这两个对象“谁大谁小”，这时，可以重写Object类的**Equals**方法

```
public override bool Equals(object obj)
{
    if (this.CompareTo(obj) == 0)
        return true;
    return false;
}
```

重写equals()方法实现对象的**判等**



如果实现了CompareTo()方法，则通常会直接使用它来实现equals()方法的功能

重写Object.Equals方法的规则

自反性
(Reflexive)

- `a.Equals(a)` 值为True。

对称性
(Symmetric)

- `a.Equals(b)` 的值与 `b.Equals(a)` 一致。

传递性
(Transitive)

- `a.Equals(b)` 为True, `b.Equals(c)` 为True, 则 `a.Equals(c)` 也为True。

与null判等

- `a.Equals(null)` 的值为False。

重写GetHashCode方法

重写了Equals方法之后，也必须重写GetHashCode方法，否则此对象在放入需要使用对象Hash值的对象集合（比如Hashtable）中时会出现问题。

要求：

1. 如果两个对象的比较结果相等，则每个对象的 GetHashCode 方法都必须返回相同值。
2. 对象状态未改变时，多次调用它的 GetHashCode 方法，必须应该返回相同值。
3. 为了获得最佳性能，选择的哈希函数必须为所有输入生成随机分布。

C# 支持运算符重载，可以重写 “==”，为对象比较提供方便：

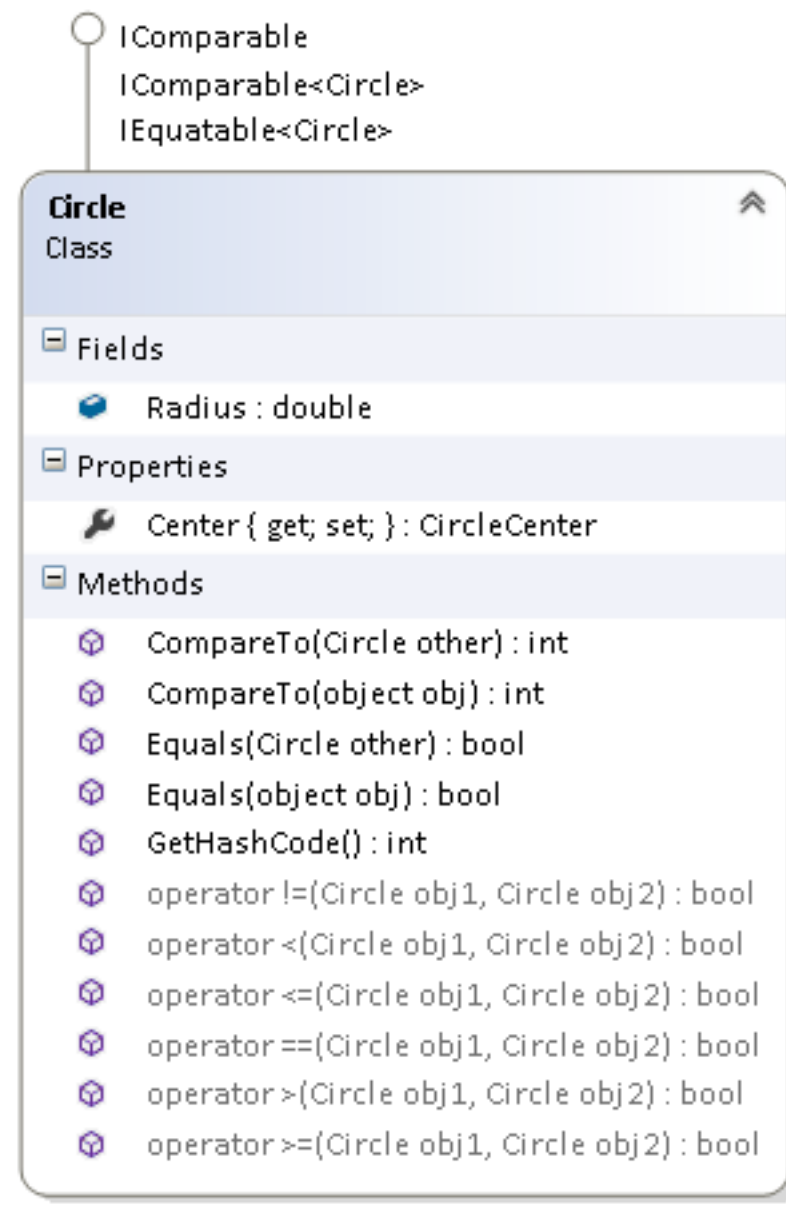
```
public static bool operator ==(MyClass obj1, MyClass obj2)
{
    return obj1.Equals(obj2);
}
```

注意：

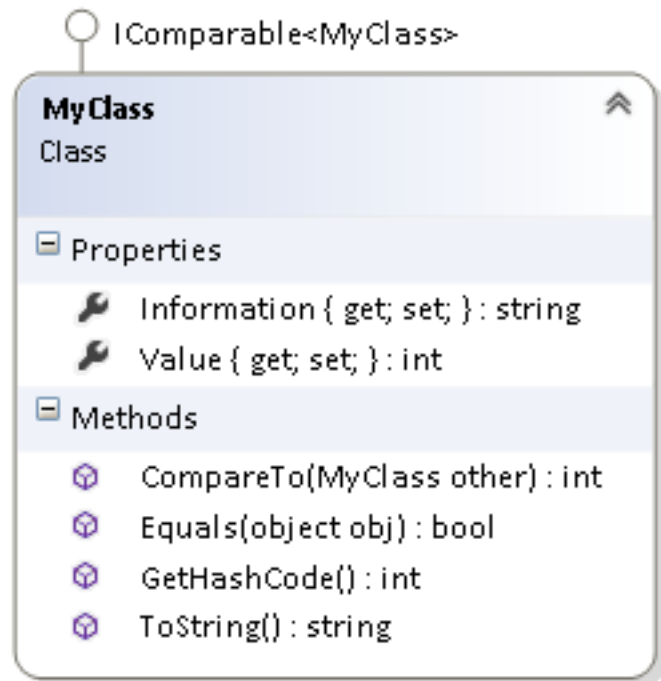
运算符重载增加了代码的理解难度，建议慎用。

- 示例ObjectCompare定义了一个Circle类，按照其“半径”大小确定大小关系，并且重写了相关的对象运算符。

- 此示例可供编程时参考。



支持“大小”比较的对象，排序和查找都比较方便。



示例ObjectSort使用.NET为对象集合所提供的扩展方法Sort()和IndexOf(), 在MyClass集合中排序和查找对象

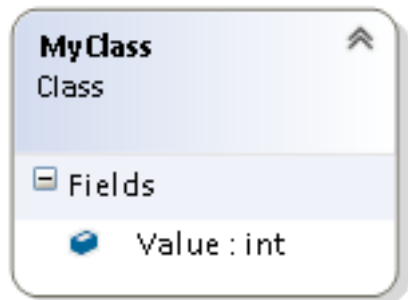
对象比较器

在实际开发中如果需要比较一些“自身不能比较大小”的对象，我们可以定义一个第三方的类，让它来负责决定某种类型的对象“谁大谁小”。这个完成对象比较任务的“中间人”，可称之为“**对象比较器（Object Comparer）**”。

对象比较器应该实现IComparer或IComparer<T>接口。

```
public interface IComparer<in T>
{
    int Compare(T x, T y);
}
```

参看示例：ObjectComparer



MyClass的
对象比较器



```
class MyClassComparer : IComparer<MyClass>
{
    0 references
    public int Compare(MyClass x, MyClass y)
    {
        return x.Value.CompareTo(y.Value);
    }
}
```

小结

- 当对象支持大小比较和判等时，将其放入到一个标准集合中，就能直接支持排序和查找，使用起来非常方便。
- 如果使用外部对象比较器，则我们可以使用多个不同的方式对集合中的对象进行“动态”的排序和查找，使用起来更为灵活。