**Интерфейсы IComparable и IComparable<T>**

Во многих классах приходится реализовывать интерфейс IComparable или

IComparable<T>, поскольку он позволяет сравнивать один объект с другим, используя

различные методы, определенные в среде .NET Framework. Интерфейсы IComparable

и IComparable<T> были представлены в главе 18, где они использовались в примерах

программ для сравнения двух объектов, определяемых параметрами обобщенного

типа. Кроме того, они упоминались при рассмотрении класса Array ранее в этой главе.

Но поскольку эти интерфейсы имеют особое значение и применяются во многих

случаях, то ниже приводится их краткое описание.

Интерфейс IComparable реализуется чрезвычайно просто, потому что он состоит

всего лишь из одного метода.

int CompareTo(object obj)

В этом методе значение вызывающего объекта сравнивается со значением объекта,

определяемого параметром *obj.* Если значение вызывающего объекта больше, чем у

объекта *obj,* то возвращается положительное значение; если оба значения равны —

нулевое значение, а если значение вызывающего объекта меньше, чем у объекта *obj,* —

отрицательное значение.

Обобщенный вариант интерфейса IComparable объявляется следующим образом.

public interface IComparable<T>

В данном варианте тип сравниваемых данных передается параметру Т в качестве аргумента

типа. В силу этого объявление метода CompareTo() претерпевает изменения

и выглядит так, как показано ниже.

int CompareTo(Т other)

В этом объявлении тип данных, которыми оперирует метод CompareTo(), может

быть указан явным образом. Следовательно, интерфейс IComparable<T> обеспечивает

типовую безопасность. Именно по этой причине он теперь считается более предпочтительным

в программировании на С#, чем интерфейс IComparable.

**Интерфейс IEquatable<T>**

Интерфейс IEquatable<T> реализуется в тех классах, где требуется определить

порядок сравнения двух объектов на равенство их значений. В этом интерфейсе определен

только один метод, Equals(), объявление которого приведено ниже.

bool Equals(T other)

Этот метод возвращает логическое значение true, если значение вызывающего

объекта оказывается равным значению другого объекта *other*, в противном случае —

логическое значение false.

Интерфейс IEquatable<T> реализуется в нескольких классах и структурах среды

.NET Framework, включая структуры числовых типов и класс String. Для реализации

интерфейса IEquatable<T> обычно требуется также переопределять методы

Equals(Object) и GetHashCode(), определенные в классе Object.

**Интерфейс Iconvertible**

Интерфейс IConvertible реализуется в структурах всех типов значений, String и

DateTime. В нем определяются различные преобразования типов. Реализовывать этот

интерфейс в создаваемых пользователем классах, как правило, не требуется.

**Интерфейс Icloneable**

Реализовав интерфейс ICloneable, можно создать все условия для копирования

объекта. В интерфейсе ICloneable определен только один метод, Clone(), объявление

которого приведено ниже.

object Clone()

В этом методе создается копия вызывающего объекта, а конкретная его реализация

зависит от способа создания копии объекта. Вообще говоря, существуют две разновидности

копий объектов: полная и неполная. Если создается полная копия, то копия

совершенно не зависит от оригинала. Так, если в исходном объекте содержится

ссылка на другой объект *О*, то при его копировании создается также копия объекта *О.*

А при создании неполной копии осуществляется копирование одних только членов,

но не объектов, на которые эти члены ссылаются. Так, после создания неполной копии

объекта, ссылающегося на другой объект *О*, копия и оригинал будут ссылаться на

один и тот же объект *О*, причем любые изменения в объекте *О* будут оказывать влияние

как на копию, так и на оригинал. Как правило, метод Clone() реализуется для

получения полной копии. А неполные копии могут быть созданы с помощью метода

MemberwiseClone(), определенного в классе Object.

Ниже приведен пример программы, в которой демонстрируется применение интерфейса

ICloneable. В ней создается класс Test, содержащий ссылку на объект класса

X. В самом классе Test используется метод Clone() для создания полной копии.

(***glava21\_10***)

class X

{

public int a;

public X(int x) { a = x; }

}

class Test : ICloneable

{

public X o;

public int b;

public Test(int x, int y)

{

o = new X(x);

b = y;

}

public void Show(string name)

{

Console.Write("Value of " + name + ": ");

Console.WriteLine("o.a: {0}, b: {1}", o.a, b);

}

//copy of calling object

public object Clone()

{

Test temp = new Test(o.a, b);

return temp;

}

}

class CloneDemo

{

static void Main()

{

Test ob1 = new Test(10, 20);

ob1.Show("ob1");

Console.WriteLine("Make object ob2 copy of ob1.");

Test ob2 = (Test)ob1.Clone();

ob2.Show("ob2");

Console.WriteLine("Change value ob1.o.a to 99, and" +

" value of ob1.b - 88.");

ob1.o.a = 99;

ob1.b = 88;

ob1.Show("ob1");

ob2.Show("ob2");

}

}

Как следует из результата выполнения приведенной выше программы, объект оb2

является копией объекта оb1, но это совершенно разные объекты. Изменения в одном

из них не оказывают никакого влияния на другой. Это достигается конструированием

нового объекта типа Test, который выделяет новый объект типа X для копирования.

При этом новому экземпляру объекта типа X присваивается такое же значение, как

и у объекта типа X в оригинале.

Для получения неполной копии достаточно вызвать метод MemberwiseClone(),

определяемый в классе Object из метода Clone(). В качестве упражнения попробуйте

заменить метод Clone() в предыдущем примере программы на следующий его вариант.

public object Clone()

{

Test temp = (Test)MemberwiseClone();

return temp;

}

}

Как видите, обе переменные экземпляра о в объектах оb1 и оb2 ссылаются на один

и тот же объект типа X. Поэтому изменения в одном объекте оказывают влияние на

другой. Но в то же время поля b типа int в каждом из них разделены, поскольку типы

значений недоступны по ссылке.

**Интерфейсы IFormatProvider и Iformattable**

В интерфейсе IFormatProvider определен единственный метод GetFormat(), который

возвращает объект, определяющий форматирование данных в удобочитаемой

форме текстовой строки. Ниже приведена общая форма метода GetFormat():

object GetFormat(Type formatType)

где *formatType* — это объект, получаемый для форматирования.

Интерфейс IFormattable поддерживает форматирование выводимых результатов

в удобочитаемой форме. В нем определен следующий метод:

string ToString(string format, IFormatProvider formatProvider)

где *format* обозначает инструкции для форматирования, a *formatProvider* —

поставщик формата.

**Интерфейсы IObservable<T> и IObserver<T>**

В версию .NET Framework 4.0 добавлены еще два интерфейса, поддерживающие

шаблон наблюдателя: IObservable<T> и IObserver<T>. В шаблоне наблюдателя

один класс (в роли наблюдаемого) предоставляет уведомления другому классу (в роли

наблюдателя). С этой целью объект наблюдаемого класса регистрирует объект наблюдающего

класса. Для регистрации наблюдателя вызывается метод Subscribe(), который

определен в интерфейсе IObservable<T> и которому передается объект типа

IObserver<T>, принимающий уведомление. Для получения уведомлений можно

зарегистрировать несколько наблюдателей. А для отправки уведомлений всем зарегистрированным

наблюдателям применяются три метода, определенные в интерфейсе

IObserver<T>. Так, метод OnNext() отправляет данные наблюдателю, метод

OnError() сообщает об ошибке, а метод OnCompleted() указывает на то, что наблюдаемый

объект прекратил отправку уведомлений.