В этой главе рассматривается одно из самых важных в C# средств: *интерфейс,* определяющий ряд методов для реализации в классе. Но поскольку в самом интерфейсе ни один из методов не реализуется, интерфейс представляет собой чисто логическую конструкцию, описывающую функциональные возможности без конкретной их реализации. Кроме того, в этой главе представлены еще два типа данных С#: структуры и перечисления. *Структуры* подобны классам, за исключением того, что они трактуются как

типы значений, а не ссылочные типы. А *перечисления* представляют собой перечни целочисленных констант. Структуры и перечисления расширяют богатый арсенал средств программирования на С#.

# Интерфейсы

Иногда в объектно-ориентированном программировании полезно определить, что именно должен делать

класс, но не как он должен это делать. Примером тому может служить упоминавшийся ранее абстрактный метод. В абстрактном методе определяются возвращаемый тип и сигнатура метода, но не предоставляется его реализация. А в производном классе должна быть обеспечена своя собственная реализация каждого абстрактного метода, определенного в его базовом классе. Таким образом, абстрактный метод определяет *интерфейс,* но не реализацию метода. Конечно, абстрактные классы и методы приносят известную пользу, но положенный в их основу принцип может быть развит далее. В C# предусмотрено разделение интерфейса класса и его реализации с помощью ключевого слова ***interface***.

С точки зрения синтаксиса интерфейсы подобны абстрактным классам. Но в интерфейсе

ни у одного из методов не должно быть тела. Это означает, что в интерфейсе вообще

не предоставляется никакой реализации. Как только интерфейс будет определен, он может

быть реализован в любом количестве классов. Кроме того, в одном классе может быть

реализовано любое количество интерфейсов.

Каждому классу предоставляется полная свобода для определения деталей своей собственной реализации интерфейса. Следовательно, один и тот же интерфейс может быть реализован в двух

классах по-разному. Тем не менее в каждом из них должен поддерживаться один и тот

же набор методов данного интерфейса. Благодаря поддержке интерфейсов в C# может

быть в полной мере реализован главный принцип полиморфизма: один интерфейс

— множество методов.

Интерфейсы объявляются с помощью ключевого слова interface. Ниже приведена

упрощенная форма объявления интерфейса.

***interface имя{***

***возвращаемый\_тип имя\_метода1(список\_параметров);***

***возвращаемый\_тип имя\_метода2(список\_параметров);***

***// ...***

***возвращаемый\_тип имя\_методаN{список\_параметров);***

***}***

где *имя —* это конкретное имя интерфейса. В объявлении методов интерфейса используются

только их *возвращаемый\_тип* и сигнатура. ***Bсе*** методы интерфейса ***должны*** быть реализованы в каждом классе, включающем в себя этот интерфейс. В самом же интерфейсе методы неявно

считаются открытыми, поэтому доступ к ним не нужно указывать явно.

Ниже приведен пример объявления интерфейса для класса, генерирующего последовательный

ряд чисел.

public interface ISeries

{

int GetNext(); // возвратить следующее по порядку число

void Reset(); // перезапустить

void SetStart(int х); // задать начальное значение

}

Этому интерфейсу присваивается имя ISeries. Префикс I в имени интерфейса

указывать необязательно, но это принято делать в практике программирования, чтобы

как-то отличать интерфейсы от классов. Интерфейс ISeries объявляется как public

и поэтому может быть реализован в любом классе какой угодно программы.

Интерфейсы не могут содержать члены данных. В них нельзя также определить конструкторы, деструкторы или операторные методы. Кроме того, ни один из членов интерфейса не может быть

объявлен как static.

# Реализация интерфейсов

Как только интерфейс будет определен, он может быть реализован в одном или нескольких

классах. Для реализации интерфейса достаточно указать его имя после имени

класса, аналогично базовому классу. Если уж интерфейс реализуется в классе, то это должно быть сделано полностью. В частности, реализовать интерфейс выборочно и только по частям нельзя.

В классе допускается реализовывать несколько интерфейсов. В этом случае все реализуемые

в классе интерфейсы указываются списком через запятую. В классе можно

наследовать базовый класс и в тоже время реализовать один или более интерфейс.

В таком случае имя базового класса должно быть указано ***перед*** списком интерфейсов,

разделяемых запятой.

Методы, реализующие интерфейс, должны быть объявлены как public. Дело

в том, что в самом интерфейсе эти методы неявно подразумеваются как открытые,

поэтому их реализация также должна быть открытой. Кроме того, возвращаемый тип

и сигнатура реализуемого метода должны точно соответствовать возвращаемому типу

и сигнатуре, указанным в определении интерфейса.

(***glava12\_1***)

public interface ISeries

{

int GetNext(); //return next order number

void Reset(); //reload

void SetStart(int x); //first value

}

class ByTwos : ISeries

{

int start;

int val;

public ByTwos()

{

start = 0;

val = 0;

}

public int GetNext()

{

val += 2;

return val;

}

public void Reset()

{

val = start;

}

public void SetStart(int x)

{

start = x;

val = start;

}

}

class SeriesDemo

{

static void Main()

{

ByTwos ob = new ByTwos();

for (int i = 0; i < 5; i++)

Console.WriteLine("Next number = " + ob.GetNext());

Console.WriteLine("\nReset");

ob.Reset();

for (int i = 0; i < 5; i++)

Console.WriteLine("Next number = " + ob.GetNext());

Console.WriteLine("Start from 100");

ob.SetStart(100);

for (int i = 0; i < 5; i++)

Console.WriteLine("Next number = " + ob.GetNext());

}

}

Интерфейс, класс и реадизация, были разбиты на разные файлы.

В классах, реализующих интерфейсы, разрешается и часто практикуется определять

их собственные дополнительные члены. В качестве примера ниже приведен другой

вариант класса ByTwos, в который добавлен метод GetPrevious(), возвращающий

предыдущее значение.

class ByTwos : ISeries

{

int start;

int val;

int prev;

public ByTwos()

{

start = 0;

val = 0;

prev = -2;

}

public int GetNext()

{

prev = val;

val += 2;

return val;

}

public void Reset()

{

val = start;

prev = start - 2;

}

public void SetStart(int x)

{

start = x;

val = start;

prev = val - 2;

}

//this method not in interface

public int GetPrevious()

{

return prev;

}

}

Как пояснялось выше, в интерфейсе вообще отсутствует какая-либо реализация,

поэтому он может быть свободно реализован в каждом классе так, как это требуется

для самого класса.

# Применение интерфейсных ссылок

Как это ни покажется странным, но в C# допускается объявлять переменные ссылочного

интерфейсного типа, т.е. переменные ссылки на интерфейс. Такая переменная

может ссылаться на любой объект, реализующий ее интерфейс. При вызове метода

для объекта посредством интерфейсной ссылки выполняется его вариант, реализованный

в классе данного объекта. Этот процесс аналогичен применению ссылки на базовый

класс для доступа к объекту производного класса, как пояснялось в главе 11.

В приведенном ниже примере программы демонстрируется применение интерфейсной

ссылки. В этой программе переменная ссылки на интерфейс используется с

целью вызвать методы для объектов обоих классов — ByTwos и Primes. Для ясности в

данном примере показаны все части программы, собранные в единый файл.

class SeriesDemo2

{

static void Main()

{

ByTwos twoOb = new ByTwos();

Primes primeOb = new Primes();

ISeries ob;

for (int i = 0; i < 5; i++)

{

ob = twoOb;

Console.WriteLine("Next even number = " + ob.GetNext());

ob = primeOb;

Console.WriteLine("Next simple = " + ob.GetNext());

}

}

}

В методе Main() переменная ob объявляется для ссылки на интерфейс ISeries.

Это означает, что в ней могут храниться ссылки на объект любого класса, реализующего

интерфейс ISeries. В данном случае она служит для ссылки на объекты twoOb и

primeOb классов ByTwos и Primes соответственно, в которых реализован интерфейс

ISeries.

И еще одно замечание: ***переменной ссылки на интерфейс доступны только методы,***

***объявленные в ее интерфейсе***. Поэтому интерфейсную ссылку нельзя использовать для

доступа к любым другим переменным и методам, которые не поддерживаются объектом

класса, реализующего данный интерфейс.

# Интерфейсные свойства

Аналогично методам, свойства указываются в интерфейсе вообще без тела. Ниже

приведена общая форма объявления интерфейсного свойства.

***// Интерфейсное свойство***

***тип имя{***

***get;***

***set;***

***}***

Очевидно, что в определении интерфейсных свойств, доступных только для чтения

или только для записи, должен присутствовать единственный аксессор: get или set

соответственно.

Несмотря на то что объявление свойства в интерфейсе очень похоже на объявление

автоматически реализуемого свойства в классе, между ними все же имеется отличие.

При объявлении в интерфейсе свойство не становится автоматически реализуемым.

В этом случае указывается только имя и тип свойства, а его реализация предоставляется

каждому реализующему классу. ***Кроме того, при объявлении свойства в интерфейсе***

***не разрешается указывать модификаторы доступа для аксессоров. Например, аксессор***

***set не может быть указан в интерфейсе как private***.

***(glava12\_2)***

public interface ISeries

{

//property interface

int Next

{

get;

set;

}

}

//use ISeries

class ByTwos : ISeries

{

int val;

public ByTwos()

{

val = 0;

}

//get or set value

public int Next

{

get

{

val += 2;

return val;

}

set

{

val = value;

}

}

}

//show interface property

class SeriesDemo3

{

static void Main()

{

ByTwos ob = new ByTwos();

//get access through property

for (int i = 0; i < 5; i++)

Console.WriteLine("Next number = " + ob.Next);

Console.WriteLine("\nStart with 21");

ob.Next = 21;

for (int i = 0; i < 5; i++)

Console.WriteLine("Next number = " + ob.Next);

}

}

# Интерфейсные индексаторы

В интерфейсе можно также указывать индексаторы. Ниже приведена общая форма

объявления интерфейсного индексатора.

***// Интерфейсный индексатор***

***тип\_элемента this[int индекс]{***

***get;***

***set;***

***}***

Как и прежде, в объявлении интерфейсных индексаторов, доступных только для

чтения или только для записи, должен присутствовать единственный аксессор: get

или set соответственно.

(***glava12\_3***)

public interface ISeries

{

//property interface

int Next

{

get;

set;

}

int this[int index]

{

get; //return number by index

}

}

//use ISeries

class ByTwos : ISeries

{

int val;

public ByTwos()

{

val = 0;

}

//get or set value

public int Next

{

get

{

val += 2;

return val;

}

set

{

val = value;

}

}

//get number by index

public int this[int index]

{

get

{

val = 0;

for (int i = 0; i < index; i++)

val += 2;

return val;

}

}

}

//show interface property

class SeriesDemo4

{

static void Main()

{

ByTwos ob = new ByTwos();

//get access through property

for (int i = 0; i < 5; i++)

Console.WriteLine("Next number = " + ob.Next);

Console.WriteLine("\nStart with 21");

ob.Next = 21;

for (int i = 0; i < 5; i++)

Console.WriteLine("Next number = " + ob.Next);

Console.WriteLine("\nReset to 0.");

ob.Next = 0;

//get access one by one with indexator

for (int i = 0; i < 5; i++)

Console.WriteLine("Next number = " + ob[i]);

}

}

# Наследование интерфейсов

Один интерфейс может наследовать другой. Синтаксис наследования интерфейсов

такой же, как и у классов. Когда в классе реализуется один интерфейс, наследующий

другой, в нем должны быть реализованы ***все*** члены, определенные в цепочке наследования

интерфейсов, как в приведенном ниже примере.

***(glava12\_4)***

public interface IA

{

void Meth1();

void Meth2();

}

//derived interface has inherented method

//and now has one method more

public interface IB : IA

{

void Meth3();

}

//this class has to realize all method from IA IB

class MyClass : IB

{

public void Meth1()

{

Console.WriteLine("Realie method Meth1.");

}

public void Meth2()

{

Console.WriteLine("Realie method Meth2.");

}

public void Meth3()

{

Console.WriteLine("Realie method Meth3.");

}

}

class IFExtend

{

static void Main()

{

MyClass ob = new MyClass();

ob.Meth1();

ob.Meth2();

ob.Meth3();

}

Как пояснялось ранее, в любом классе, реализующем интерфейс, должны быть реализованы все методы, определенные в этом интерфейсе, в том числе и те, что наследуются из других интерфейсов.

# Сокрытие имен при наследовании интерфейсов

Когда один интерфейс наследует другой, то в производном интерфейсе может

быть объявлен член, скрывающий член с аналогичным именем в базовом интерфейсе.

Такое сокрытие имен происходит в том случае, если член в производном интерфейсе

объявляется таким же образом, как и в базовом интерфейсе. Но если не указать

в объявлении члена производного интерфейса ключевое слово ***new***, то компилятор выдаст

соответствующее предупреждающее сообщение.

# Явные реализации

При реализации члена интерфейса имеется возможность указать его имя *полностью*

вместе с именем самого интерфейса. В этом случае получается *явная реализация*

*члена интерфейса,* или просто *явная реализация.* Так, если объявлен интерфейс ImyIF

interface IMyIF

{

int MyMeth(int x);

}

то следующая его реализация считается вполне допустимой:

class MyClass : ImyIF

{

int IMyIF.MyMeth(int x)

{

return x / 3;

}

}

Как видите, при реализации члена MyMeth() интерфейса IMyIF указывается его полное имя, включающее в себя имя его интерфейса.

Для явной реализации интерфейсного метода могут быть две причины. Во-первых,

когда интерфейсный метод реализуется с указанием его полного имени, то такой метод

оказывается доступным не посредством объектов класса, реализующего данный

интерфейс, а по интерфейсной ссылке. Следовательно, явная реализация позволяет

реализовать интерфейсный метод таким образом, чтобы он *не* стал открытым членом

класса, предоставляющего его реализацию. И во-вторых, в одном классе могут быть

реализованы два интерфейса с методами, объявленными с одинаковыми именами и

сигнатурами. Но неоднозначность в данном случае устраняется благодаря указанию

в именах этих методов их соответствующих интерфейсов. Рассмотрим каждую из этих

двух возможностей явной реализации на конкретных примерах.

(***glava12\_5***)

interface IEven

{

bool IsOdd(int x);

bool IsEven(int x);

}

class MyClass : IEven

{

//explicit realize

//Have a closer look, that member

//is closed default

bool IEven.IsOdd(int x)

{

if ((x % 2) != 0) return true;

else return false;

}

//simple realize

public bool IsEven(int x)

{

IEven o = this; //interface link of calling object

return !o.IsOdd(x);

}

}

class Demo

{

static void Main()

{

MyClass ob = new MyClass();

bool result;

result = ob.IsEven(4);

if (result) Console.WriteLine("4 - odd number.");

//result = ob.IsOdd(4); ERROR!!, IsOdd of IEven inaccessible

//but next code is right, bcs of

//first create interface link IEven on MyClass object

//and then call method IsOdd() with this link

IEven iRef = (IEven)ob;

result = iRef.IsOdd(3);

if (result) Console.WriteLine("3 - even number.");

}

}

В приведенном выше примере метод IsOdd() реализуется явно, а значит, он недоступен

как открытый член класса MyClass. Напротив, он доступен только по интерфейсной

ссылке. Именно поэтому он вызывается посредством переменной о ссылочного

типа IEven в реализации метода IsEven().

Ниже приведен пример программы, в которой реализуются два интерфейса, причем

в обоих интерфейсах объявляется метод Meth(). Благодаря явной реализации исключается

неоднозначность, характерная для подобной ситуации.

(***glava12\_6***)

interface IMyIF\_A

{

int Meth(int x);

}

interface IMyIF\_B

{

int Meth(int x);

}

//both interfaces realized in MyClass

class MyClass : IMyIF\_A, IMyIF\_B

{

//both method Meth() explicit

int IMyIF\_A.Meth(int x)

{

return x + x;

}

int IMyIF\_B.Meth(int x)

{

return x \* x;

}

//call method interface link

public int MethA(int x)

{

IMyIF\_A a\_ob;

a\_ob = this;

return a\_ob.Meth(x); //call interface method IMyIF\_A

}

public int MethB(int x)

{

IMyIF\_B b\_ob;

b\_ob = this;

return b\_ob.Meth(x); //call interface method IMyIF\_A

}

}

class FQIFNames

{

static void Main()

{

MyClass ob = new MyClass();

Console.Write("Call method IMyIF\_A.Meth(): ");

Console.WriteLine(ob.MethA(3));

Console.Write("Call method IMyIF\_B.Meth(): ");

Console.WriteLine(ob.MethB(3));

}

}

Анализируя приведенный выше пример программы, обратим прежде всего внимание

на одинаковую сигнатуру метода Meth() в обоих интерфейсах, IMyIF\_A и

IMyIF\_B. Когда оба этих интерфейса реализуются в классе MyClass, для каждого из

них в отдельности это делается явно, т.е. с указанием полного имени метода Meth().

А поскольку явно реализованный метод может вызываться только по интерфейсной

ссылке, то в классе MyClass создаются две такие ссылки: одна — для интерфейса

IMyIF\_A, а другая — для интерфейса IMyIF\_B. Именно по этим ссылкам происходит

обращение к объектам данного класса с целью вызвать методы соответствующих интерфейсов,

благодаря чему и устраняется неоднозначность.

# Выбор между интерфейсом и абстрактным классом

Одна из самых больших трудностей программирования на C# состоит в правильном

выборе между интерфейсом и абстрактным классом в тех случаях, когда требуется

описать функциональные возможности, но не реализацию. В подобных случаях

рекомендуется придерживаться следующего общего правила: если какое-то понятие

можно описать с точки зрения функционального назначения, не уточняя конкретные

детали реализации, то следует использовать интерфейс. А если требуются некоторые

детали реализации, то данное понятие следует представить абстрактным классом.

# Стандартные интерфейсы для среды .NET Framework

Для среды .NET Framework определено немало стандартных интерфейсов, которыми

можно пользоваться в программах на С#. Так, в интерфейсе System.IComparable

определен метод CompareTo(), применяемый для сравнения объектов, когда требуется

соблюдать отношение порядка. Стандартные интерфейсы являются также важной

частью классов коллекций, предоставляющих различные средства, в том числе

стеки и очереди, для хранения целых групп объектов. Так, в интерфейсе System.

Collections.ICollection определяются функции для всей коллекции, а в интерфейсе

System.Collections.IEnumerator — способ последовательного обращения

к элементам коллекции. Эти и многие другие интерфейсы подробнее рассматриваются

в части II данной книги.