Объектно-ориентированное программирование

Лекция №6. Интерфейсы, наследование, стандартные интерфейсы .NET

# Темы:

1. Интерфейсы. Синтаксис. Реализация. Работа с объектами. Проверка и приведение к интерфейсу
2. Наследование интерфейсов. Множественное наследование.
3. Стандартные интерфейсы .NET. IComparable, IEnumerable, IClonable.

# Интерфейсы

## Синтаксис интерфейса

Интерфейс является «крайним случаем» абстрактного класса. В нем задается набор абстрактных методов, свойств и индексаторов, которые должны быть реализованы в производных классах. Иными словами, интерфейс определяет поведение, которое поддерживается реализующими этот интерфейс классами. Основная идея использования интерфейса состоит в том, чтобы к объектам таких классов можно было обращаться одинаковым образом.

Каждый класс может определять элементы интерфейса по-своему. Так достигается полимофизм: объекты разных классов по-разному реагируют на вызовы одного и того же метода.

Синтаксис интерфейса аналогичен синтаксису класса:

[ атрибуты ] [ спецификаторы ] interface иняинтерфейса [ : предки ]

тело\_интерфейса [ ; ]

Для интерфейса могут быть указаны спецификаторы new, public, protected, internal и private. Спецификатор new применяется для вложенных интерфейсов и имеет такой же смысл, как и соответствующий модификатор метода класса. Остальные спецификаторы управляют видимостью интерфейса. В разных контекстах определения интерфейса допускаются разные спецификаторы.

Интерфейс может наследовать свойства нескольких интерфейсов, в этом случае предки перечисляются через запятую. Тело интерфейса составляют абстрактные методы, шаблоны свойств и индексаторов, а также события.

Методом исключения можно догадаться, что интерфейс не может содержать константы, поля, операции, конструкторы, деструкторы, типы и любые статические элементы.

В качестве примера рассмотрим интерфейс IAction, определяющий базовое поведение персонажей компьютерной игры.

Допустим, что любой персонаж должен уметь выводить себя на экран, атаковать и красиво умирать:

interface IAction

{

void Draw();

int Attack(int a);

void Die();

int Power { get; }

}

В интерфейсе IAction заданы заголовки трех методов и шаблон свойства Power, доступного только для чтения. Как легко догадаться, если бы требовалось обеспечить еще и возможность установки свойства, в шаблоне следовало указать ключевое слово set, например:

int Power { get; set; }

В интерфейсе имеет смысл задавать заголовки тех методов и свойств, которые будут по-разному реализованы различными классами разных иерархий.

Отличия интерфейса от абстрактного класса:

* элементы интерфейса по умолчанию имеют спецификатор доступа publiс и не могут иметь спецификаторов, заданных явным образом;
* интерфейс не может содержать полей и обычных методов — все элементы интерфейса должны быть абстрактными;
* класс, в списке предков которого задается интерфейс, должен определять все его элементы, в то время как потомок абстрактного класса может не переопределять часть абстрактных методов предка (в этом случае производный класс также будет абстрактным);
* класс может иметь в списке предков несколько интерфейсов, при этом он должен определять все их методы.

## Реализация интерфейса

В списке предков класса сначала указывается его базовый класс, если он есть, а затем через запятую — интерфейсы, которые реализует этот класс. Таким образом, в C# поддерживается одиночное наследование для классов и множественное — для интерфейсов. Это позволяет придать производному классу свойства нескольких базовых интерфейсов, реализуя их по своему усмотрению.

Естественно, что сигнатуры методов в интерфейсе и реализации должны полностью совпадать. Для реализуемых элементов интерфейса в классе следует указывать спецификатор publiс. К этим элементам можно обращаться как через объект класса, так и через объект типа соответствующего интерфейса.

## Работа с объектами через интерфейсы. Операции is и as

При работе с объектом через объект типа интерфейса бывает необходимо убедиться, что объект поддерживает данный интерфейс. Проверка выполняется с помощью бинарной операции is. Эта операция определяет, совместим ли текущий тип объекта, находящегося слева от ключевого слова is, с типом, заданным справа. Результат операции равен true, если объект можно преобразовать к заданному типу, и false в противном случае. Операция обычно используется в следующем контексте:

if ( объект is тип )

{

// выполнить преобразование "объекта" к "типу"

// выполнить действия с преобразованным объектом

}

Допустим, мы оформили какие-то действия с объектами в виде метода с параметром типа object. Прежде чем использовать этот параметр внутри метода для обращения к методам, описанным в производных классах, требуется выполнить преобразование к производному классу. Для безопасного преобразования следует проверить, возможно ли оно, например так:

static void Act( object A )

{

if ( A is IAction )

{

IAction Actor = (IAction) A;

Actor.Draw();

}

}

В метод Act можно передавать любые объекты, но на экран будут выведены только те, которые поддерживают интерфейс IAction.

Недостатком использования операции i s является то, что преобразование фактически выполняется дважды: при проверке и при собственно преобразовании. Более эффективной является другая операция — as. Она выполняет преобразование к заданному типу, а если это невозможно, формирует результат null, например:

static void Act( object A )

{

IAction Actor = A as IAction:

if ( Actor != null ) Actor.Draw();

}

Обе рассмотренные операции применяются как к интерфейсам, так и к классам.

## Интерфейсы и наследование

Интерфейс может не иметь или иметь сколько угодно интерфейсов-предков, в последнем случае он наследует все элементы всех своих базовых интерфейсов, начиная с самого верхнего уровня. Базовые интерфейсы должны быть доступны в не меньшей степени, чем их потомки. Например, нельзя использовать интерфейс, описанный со спецификатором private или internal, в качестве базового для открытого (public) интерфейса.

Как и в обычной иерархии классов, базовые интерфейсы определяют общее поведение, а их потомки конкретизируют и дополняют его. В интерфейсе-потомке можно также указать элементы, переопределяющие унаследованные элементы с такой же сигнатурой. В этом случае перед элементом указывается ключевое слово new, как и в аналогичной ситуации в классах. С помощью этого слова соответствующий элемент базового интерфейса скрывается. Вот пример из документации С#:

interface IBase

{

void F( int i );

}

interface Ileft : IBase

{

new void F( int i ); //переопределение метода F

}

interface Iright : IBase

{

void G();

}

interface Iderived : Ileft, IRight {}

class A

{

void Test( IDerived d ) {

d.F( 1 ); // Вызывается ILeft.F

((IBase)d).F( 1 ); // Вызывается IBase.F

((ILeft)d).F( 1 ); //Вызывается ILeft.F

((IRight)d).F( 1 ); // Вызывается IBase.F

}

}

Метод F из интерфейса IBase скрыт интерфейсом ILeft, несмотря на то что в цепочке IDerived — IRight — IBase он не переопределялся.

Класс, реализующий интерфейс, должен определять все его элементы, в том числе унаследованные. Если при этом явно указывается имя интерфейса, оно должно ссылаться на тот интерфейс, в котором был описан соответствующий элемент, например:

class А : IRight

{

IRight.G() { ... }

IBase.F( int i ) { ...} // IRight.F( int i ) - нельзя

}

Интерфейс, на собственные или унаследованные элементы которого имеется явная ссылка, должен быть указан в списке предков класса, например:

class В : А {

//IRight.GO { ... } // нельзя!

}

class С : A, IRight {

IRight.G() { ... } // можно

IBase.F( int i ) { ...} // можно

}

Класс наследует все методы своего предка, в том числе те, которые реализовывали интерфейсы. Он может переопределить эти методы с помощью спецификатора new, но обращаться к ним можно будет только через объект класса. Если использовать для обращения ссылку на интерфейс, вызывается не переопределенная версия:

interface IBase {

void А();

}

class Base : IBase

{

public void A() {…}

}

class Derived: Base

{

new public void A(){…}

}

…

Derived d = new Derived():

d.A();// вызывается Derived.A();

IBase id = d;

id.A();// вызывается Base.A();

Однако если интерфейс реализуется с помощью виртуального метода класса, после его переопределения в потомке любой вариант обращения (через класс или через интерфейс) приведет к одному и тому же результату:

interface IBase {

void А();

}

class Base : IBase

{

public virtual void A() {…}

}

class Derived: Base

{

public override void A(){…}

}

…

Derived d = new Derived():

d.A();// вызывается Derived.A();

IBase id = d;

id.A();// вызывается Derived.A();

## Стандартные интерфейсы .NET

В библиотеке классов .NET определено множество стандартных интерфейсов, задающих желаемое поведение объектов. Например, интерфейс IComparable задает метод сравнения объектов по принципу больше или меньше, что позволяет выполнять их сортировку. Реализация интерфейсов IEnumerable и IEnumerator дает возможность просматривать содержимое объекта с помощью конструкции foreach, а реализация интерфейса ICloneable — клонировать объекты.

### Сравнение объектов (интерфейс IComparable)

Интерфейс IComparable определен в пространстве имен System. Он содержит всего один метод CompareTo, возвращающий результат сравнения двух объектов — текущего и переданного ему в качестве параметра:

interface IComparable

{

int CompareTo( object obj )

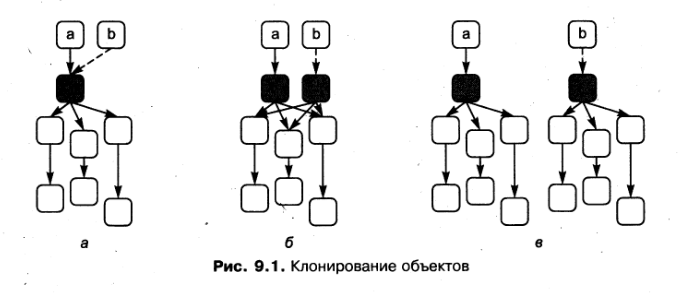
}

Метод должен возвращать:

* 0, если текущий объект и параметр равны;
* отрицательное число, если текущий объект меньше параметра;
* положительное число, если текущий объект больше параметра.

### Клонирование объектов (интерфейс ICIoneable)

Клонирование — это создание копии объекта. Копия объекта называется клоном. Как вам известно, при присваивании одного объекта ссылочного типа другому копируется ссылка, а не сам объект (рис. 9.1, а). Если необходимо скопировать в другую область памяти поля объекта, можно воспользоваться методом MemberwiseClone, который любой объект наследует от класса object. При этом объекты, на которые указывают поля объекта, в свою очередь являющиеся ссылками, не копируются (рис. 9.1, б). Это называется поверхностным клонированием.



Для создания полностью независимых объектов необходимо глубокое клонирование, когда в памяти создается дубликат всего дерева объектов, то есть объектов, на которые ссылаются поля объекта, поля полей и т. д. (рис. 9.1, в). Алгоритм глубокого клонирования весьма сложен, поскольку требует рекурсивного обхода всех ссылок объекта и отслеживания циклических зависимостей. Объект, имеющий собственные алгоритмы клонирования, должен объявляться как наследник интерфейса ICloneable и переопределять его единственный метод Clone.

### Перебор объектов (интерфейс lEnumerable) и итераторы

Оператор foreach является удобным средством перебора элементов объекта. Массивы и все стандартные коллекции библиотеки .NET позволяют выполнять такой перебор благодаря тому, что в них реализованы интерфейсы lEnumerable и IEnumerator. Для применения оператора foreach к пользовательскому типу данных требуется реализовать в нем эти интерфейсы.

Интерфейс lEnumerable (перечислимый) определяет всего один метод — GetEnumerator, возвращающий объект типа IEnumerator (перечислитель), который можно использовать для просмотра элементов объекта.

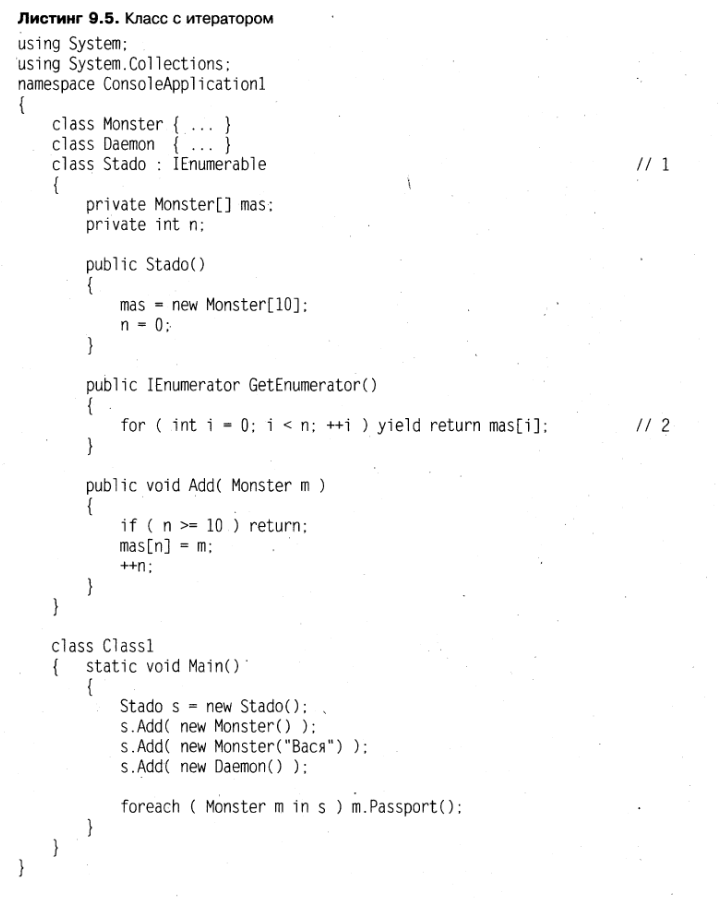
Интерфейс I Enumerator задает три элемента:

* свойство Current, возвращающее текущий элемент объекта;
* метод MoveNext, продвигающий перечислитель на следующий элемент объекта;
* метод Reset, устанавливающий перечислитель в начало просмотра.

Цикл foreach использует эти методы для перебора элементов, из которых состоит объект.

Таким образом, если требуется, чтобы для перебора элементов класса мог применяться цикл foreach, необходимо реализовать четыре метода: GetEnumerator, Current, MoveNext и Reset. Например, если внутренние элементы класса организованы в массив, потребуется описать закрытое поле класса, хранящее текущий индекс в массиве, в методе MoveNext задать изменение этого индекса на 1 с проверкой выхода за границу массива, в методе Current — возврат элемента массива по текущему индексу и т. д.

Это не интересная работа, а выполнять ее приходится часто, поэтому в версию 2.0 были введены средства, облегчающие выполнение перебора в объекте — итераторы. Итератор представляет собой блок кода, задающий последовательность перебора элементов объекта. На каждом проходе цикла foreach выполняется один шаг итератора, заканчивающийся выдачей очередного значения. Выдача значения выполняется с помощью ключевого слова yield.



Все, что требуется сделать в версии 2.0 для поддержки перебора, — указать, что класс реализует интерфейс IEnumerable (оператор 1), и описать итератор (оператор 2). Доступ к нему может быть осуществлен через методы MoveNext и Current интерфейса I Enumerator.

За кодом, приведенным в листинге 9.5, стоит большая внутренняя работа компилятора. На каждом шаге цикла foreach для итератора создается «оболочка» — служебный объект, который запоминает текущее состояние итератора и выполняет все необходимое для доступа к просматриваемым элементам объекта. Иными словами, код, составляющий итератор, не выполняется так, как он выглядит — в виде непрерывной последовательности, а разбит на отдельные итерации, между которыми состояние итератора сохраняется.

## Заключение

Интерфейсы чаще всего используются для задания общих свойств объектов различных иерархий. Основная идея интерфейса состоит в том, что к объектам классов, реализующих интерфейс, можно обращаться одинаковым образом, при этом каждый класс может определять элементы интерфейса по-своему.

Если некий набор действий имеет смысл только для какой-то конкретной иерархии классов, реализующих эти действия разными способами, уместнее задать этот набор в виде виртуальных методов абстрактного базового класса иерархии.

В C# поддерживается одиночное наследование для классов и множественное — для интерфейсов. Это позволяет придать производному классу свойства нескольких базовых интерфейсов. Класс должен определять все методы всех интерфейсов, которые имеются в списке его предков.

В библиотеке .NET определено большое количество стандартных интерфейсов. Реализация стандартных интерфейсов в собственных классах позволяет использовать для объектов этих классов стандартные средства языка и библиотеки.

Например, для обеспечения возможности сортировки объектов стандартными методами следует реализовать в соответствующем классе интерфейсы IComparable или IComparer. Реализация интерфейсов IEnumerable и IEnumerator дает возможность просматривать содержимое объекта с помощью конструкции foreach, а реализация интерфейса ICloneable — клонировать объекты.

Использование итераторов упрощает организацию перебора элементов и позволяет задать для одного и того же класса различные стратегии перебора.