Государственный университет – Высшая школа экономики

Факультет Бизнес-Информатики

Кафедра Основ информатики
и прикладного программного обеспечения

C#

Объектно-ориентированный язык программирования

Пособие (дополнение 2) к практическим занятиям - №2

Проф. Забудский Е.И.

Москва 2006

Тема: Индексаторы в C#

Одно практических занятия
(2 часа)

СВОЙСТВА обеспечивают удобный доступ к отдельным переменным-членам,

ИНДЕКСАТОРЫ выполняют эту функцию для массивов, размещенных внутри классов.

// **КОММЕНТАРИЙ**. На сайте http://www.firststeps.ru/ “Первые шаги” представлено много интересных обучающих материалов по различным интегрированным средам и языкам программирования, в том числе представлены C# и платформа .NET (step by step).

http://ooad.asf.ru/ – **ПОСЕТИТЕ ЭТОТ САЙТ, он посвящен объектно-ориентированному анализу, проектированию и программированию**

Содержание

1. Индексаторы: использование объектов как массивов [1, с. 553, сл.]

Индексаторы используются с классами, представляющими массивы. Так же, как свойства обеспечивают удобный доступ к отдельным переменным-членам, индексаторы выполняют эту функцию для массивов, размещенных внутри классов.

Индексатор представляет собой слегка измененное свойство C#.

В простейшем виде индексатор создается при помощи синтаксиса **this[ ]**.

За исключением применения ключевого слова **this** индексатор ничем не отличается от обычного объявления свойства C#.

Напомню: для обращения к отдельному элементу массива применяется: **1)** индекс **2)** в квадратных скобках **3)** после имени:

...accounts[**4**]**...**

Если объект представляет собой массив, было бы удобно обращаться к его элементам так же (по индексу в скобках). Для этого в C# введен специальный элемент языка – **индексатор**.

// ПРИМЕЧАНИЕ
Индексаторы позволяют использовать наглядный синтаксис доступа к элементам коллекции, инкапсулированной в объекте. Синтаксис состоит: **1)** из квадратных скобок, **2)** внутри которых находится аргумент-индекс, **3)** следующий за именем объекта:

...myArrayObject[**4**]**...**

2. Концепция индексаторов

Концепция индексаторов сходна с **концепцией свойств**. Свойство имитирует поле, но на самом деле исполняет блоки **get** и **set**, индексатор имитирует массив, но также исполняет блоки **get** и **set**. В обеих конструкциях применяется идентичный синтаксис.

В строках **7–17** листинга 1 показан пример использования индексатора в классе **BirthsList**. С его помощью метод Main (строки **22–35**) обращается к массиву **births** (строка **5**). При этом объект **bLDenmark** класса **BirthsList** рассматривается как массив (строки **27-30, 32**).

Листинг 1. Исходный код **BirthsList.cs**

```
01: using System;
02:
03: class BirthsList
04: {
05:     private uint [ ] births = new uint[ 4 ];
06:
07:     public uint this [uint index]  // индексатор создан при помощи синтаксиса this [ ]
08:     {
09:         get
10:         {
11:             return births[index];
12:         }
13:         set
14:         {
15:             births[index] = value;
16:         }
17:     }
18: }  // окончание класса  BirthsList
19:
```

```
20: class BirthListTester
21: {
22:       public static void Main()
23:     {
24:           BirthsList  bLDenmark = new BirthsList();
25:           uint sum;
26:
27:           bLDenmark[ 0 ] = 10200;
28:           bLDenmark[ 1 ] = 20398;
29:           bLDenmark[ 2 ] = 40938;
30:           bLDenmark[ 3 ] = 6894;
31:
32:           sum = bLDenmark[ 0 ] + bLDenmark[ 1 ] + bLDenmark[ 2 ] + bLDenmark[ 3 ];
33:
34:           Console.WriteLine("Сумма четырех регионов: {0}", sum);  Console.ReadLine();
35:     }
36:} // окончание класса BirthListTester
```

Результат работы программы: Сумма четырех регионов: **78430**

Индексатор класса **BirthsList** включает блоки **get** (строки **9–12**) и **set** (строки **13–16**). Синтаксис в строке **27** сходен с присваиванием значения элементу массива. Он запускает блок индексатора **set**. Перед его исполнением среда исполнения автоматически производит два присваивания: **1)** значение аргумента-индекса, определенное в строке **27** (в данном случае **0**), присваивается параметру **index** в строке **7**, а **2)** неявно определенному параметру **value** блока **set** в правой части оператора в строке **27** присваивается значение **10200**. На момент выполнения строка **15** выглядит так:

births[**0**] = **10200**;

т. е. первому элементу массива **births** присваивается **10200**.
И так далее работают строки **28–30** и соответственно пункты **1)** и **2)**.

Оператор **bLDenmark**[**0**] (строка **32**) читает значение с индексом **0**. Для этого запускается блок **get**. Среда исполнения автоматически присваивает **0** параметру **index** до исполнения строки **11**, которая затем принимает вид:

return births[**0**];

Таким образом, запись **bLDenmark**[**0**] возвращает значение первого элемента массива **births**.
Несмотря на **сходство индексаторов** и **свойств**, между ними имеется и несколько существенных различий:

1) Индексатор нельзя объявить как **static**. Он должен быть **элемент**ом **экземпляра** (**это объект**), т. е. с помощью квадратных скобок можно обращаться **к объектам**, **но не к классам.**

2) Индексатор идентифицируется объектом, в котором он находится, и комбинацией аргументов в квадратных скобках. Следовательно, **у самого индексатора имени нет**. Поскольку объект трактуется как массив, а индексатор остается анонимным, **в объявлении индексатора всегда применяется ключевое слово this.**

3) Заголовок объявления индексатора должен включать ***не*пустой** список формальных параметров.

3. Вызов индексатора внутри его объекта

Как известно, для вызова метода внутри его объекта достаточно указать имя метода и его аргументы, то есть:

```
class MyClass
{
…
private void PrintMenu()
{
…
}

…
...PrintMenu()...          // Вызов метода PrintMenu внутри объекта, где он определен
…
}
```

Подобным же образом иногда требуется обратиться к индексатору внутри объекта, в котором он определен. Поскольку индексатор не имеет имени, здесь используется другой синтаксис**:** **1)** ключевое слово **this** (указывает на объект, посредством которого идентифицируется индексатор) и **2)** аргумент-индекс **3)** в квадратных скобках (см. выше строку 07).

```
class Matrix
{
…
public int this[int index]
{
get
…                          // Вызов индексатора внутри его объекта
set
…
}
…
...this[10]...
…
}
```

4. Перегрузка индексатора: несколько индексаторов в одном классе

В один класс можно включить несколько индексаторов, но поскольку **имена у всех у них одинаковы** (**this**), каждый из них должен иметь уникальную сигнатуру. **Сигнатуре индексатора** по смыслу идентична **сигнатуре метода**.

Сигнатура индексатора определяется**:** **1)** количеством, **2)** типом и **3)** последовательностью его формальных параметров. В отличие от сигнатуры метода, **в нее не входят никакие имена**.

В качестве примера рассмотрим индексатор с сигнатурой **int, int**.

```
public int this [int idxRow, int idxColumn]
{
get
…
```

```
set
…
}
```

При использовании концепции сигнатуры индексатора необходимо помнить:

1) Тип элемента *не* входит в сигнатуру индексатора. Поэтому два следующих индексатора имеют одну сигнатуру, хотя тип первого – **double**, а второго – **int**:

```
public double this [int idxRow, int idxColum]  // сигнатура индексатора … this [int …, int …].
public int this [int idxRow, int idxColum]      // сигнатура индексатора … this [int …, int …].
```

2) Имена формальных параметров *не* входят в сигнатуру индексатора, поэтому две следующих заголовка имеют одну сигнатуру:

```
public int this [int idxRow, int idxColum]
public int this [int a, int b]
```

Когда в одном классе содержится несколько индексаторов, каждый из которых имеет уникальную сигнатуру, они называются **перегруженными**.

На рис. **1** представлен пример использования двух **перегруженных** индексаторов в классе **RainfallList**. Обе сигнатуры содержат по одному параметру разного типа: одна – **string**, другая – **int**. Вызовы **rainInUSA**[**10**] и **rainInUSA**[**"California"**] запускают разные индексаторы.

```
class RainfallList
{
…          // Выражение "California" принадлежит типу string, поэтому вызывается этот индексатор
      public int this [string index]
      {
      get
      …
      set
      …
      }
public int this [int index]          // 10 принадлежит типу int, поэтому вызывается этот индексатор
            {
            get
            …
            set
            …
            }
            …
}
…
RainfallList rainInUSA = new RainfallList()
rainInUSA[10] = 200;
rainInUSA["California"]= 400;
```

Рис. 1 Перегруженные индексаторы

5. *Не* следует злоупотреблять индексаторами

До того как добавлять в класс индексаторы, нужно обратить внимание на два важных принципа:

1) Индексаторы должны использоваться только в классах, предназначенных для представления коллекций.

2) Индексатор должен использоваться для представления только данных коллекции, а не переменных экземпляра.

6. Еще об Индексаторах [4., с. 257, 8]

Рассматриваются два специальных типа членов класса, которые тесно связаны друг с другом: индексаторы и свойства. Каждый из этих типов расширяет возможности класса, усиливая его интеграцию в систему типов языка C# и гибкость.

Индексаторы обеспечивают механизм, посредством которого к объектам можно получать доступ по индексу подобно тому, как это реализовано в массивах.

Свойства предлагают эффективный способ управления доступом к данным экземпляра класса.

Эти типы связаны друг с другом, поскольку оба опираются на еще одно средство C#: аксессор, или средство доступа к данным (**get**, **set**).

Как известно, индексация массивов реализуется с использованием оператора " [ ] ".

Индексатор позволяет обеспечить индексированный доступ к объекту. Главное назначение индексаторов — поддержать создание специализированных массивов, на которые налагается одно или несколько ограничений. При этом индексаторы можно использовать в синтаксисе, подобном реализованному в массивах.

Индексаторы могут характеризоваться одной или несколькими размерностями, но мы начнем с одномерных индексаторов.

7. Создание *одно*мерных индексаторов

Одномерный индексатор имеет следующий формат.

```
тип_элемента this[int индекс]
{   // Аксессор считывания данных.
get
{
// Возврат значения, заданного элементом индекс.
}

    // Аксессор установки данных,
      set
      {
      // Установка значения, заданного элементом индекс.
      }
```

Здесь тип_элемента – базовый тип индексатора. Таким образом, тип_элемента – это тип каждого элемента, к которому предоставляется доступ посредством индексатора. Он соответствует базовому типу массива. Параметр индекс получает индекс **опрашиваемого** – **get** (или **устанавливаемого** – **set**) элемента. Строго говоря, этот параметр не обязательно

должен иметь тип **int**, но поскольку индексаторы обычно используются для обеспечения индексации массивов, целочисленный тип – наиболее подходящий.

В теле индексатора определяются два аксессора (средства доступа) с именами **get** и **set**. **Аксессор подобен методу** за исключением того, что в нем отсутствует объявление типа возвращаемого значения и параметров. **При использовании индексатора аксессоры вызываются** автоматически, и в качестве параметра оба аксессора принимают индекс. Если индексатор стоит слева от оператора присваивания, вызывается аксессор **set** и устанавливается элемент, заданный параметром **индекс**. В противном случае вызывается аксессор **get** и возвращается значение, соответствующее параметру **индекс**. Метод **set** также получает значение (именуемое **value**), которое присваивается элементу массива, найденному по заданному индексу.

Одно из достоинств индексатора состоит в том, что он позволяет точно управлять характером доступа к массиву, "отбраковывая" попытки некорректного доступа. (См. пример [**4**, с. 258]).

8. Ссылка на текущий объект с помощью ключевого слова this

Каждый метод объекта автоматически поддерживает ссылку **this**, указывающую на объект, для которого вызван метод. Слово **this** принадлежит набору ключевых слов C# [1, с. 485-490]

Список литературы

1. Микелсен Клаус. Язык программирования C#. Лекции и упражнения. Учебник: пер. с англ./ Клаус Микелсен –СПб.: ООО «ДиаСофтЮП», 2002. – 656 с.

2. Джо Майо. C#Builder. Быстрый старт. Пер. с англ. – М.: ООО «Бином-Пресс», 2005 г. – 384 с.

3. Основы Microsoft Visual Studio .NET 2003 / Пер. с англ. - М.: Издательско-торговый дом «Русская Редакция», 2003. – 464 с. Брайан Джонсон, Крэйт Скибо, Марк Янг.

4. Герберт Шилдт. Полный справочник по C# . / Пер. с англ./ Издательство: Вильямс, 2004 г. 752 с.

5. Чарльз Петцольд. Программирование в тональности C# / Пер. с англ. Издательство: Русская Редакция, 2004 г. - 512 стр.

Анатомия класса

Класс является для объекта тем же, чем строительный чертеж для дома. Класс — это абстракция (реальная или концептуальная) объекта, принадлежащего какой-либо предметной области. Один шаблон класса можно использовать для создания нескольких объектов (экземпляров класса), которые обладают свойствами, определенными в классе.

При решении разных вычислительных задач объекты различных классов взаимодействуют друг с другом, внося свои уникальные свойства в общую программу. Конструкция класса позволяет объединять данные (называемые состоянием объекта) с функциями (представляющими его поведение) для создания объектов, составляющих структуру разрабатываемого ПО. Классы, в простейшем варианте, состоят из переменных и методов экземпляра (см. Материалы к Практ. Зан. №1, рис. 1.5).

Элементы класса являются языковыми конструкциями, составляющими тело класса, к примеру, переменные и методы экземпляра представляют собой два фундаментальных элемента класса. Однако классы настолько разнообразны, что **C#** содержит и несколько других элементов, придающих классу гибкость и расширяющих его возможности по взаимодействию с другими классами программы.

П.1. Обзор возможных элементов класса

В синтаксическом блоке **П.1** расширен синтаксис, показанный на рис. 1.5 (Практ. Зан. №1), и описаны элементы, которые можно включать в определение класса. В первых строках отображен уже знакомый синтаксис: 1) ключевое слово **class**, 2) имя (идентификатор) класса и 3) фигурные скобки, формирующие тело класса.

Элементы (члены) класса можно разделить на три широкие категории.

1. Данные-члены.
2. Функции-члены.
3. Вложенные типы.

Далее приведено их краткое описание.

П.1.1. Данные-члены состоят из **:**

1.1. Переменных-членов.

1.2. Констант.

1.3. Событий.

1.1. Переменные-члены (называемые также полями) используются для представления данных. Такая переменная может принадлежать**: а)** или конкретному экземпляру (объекту) класса — в этом случае она называется переменной экземпляра, **б)** или самому классу — тогда она называется статической (объявленной **static**) переменной (или переменной класса). Напомним, что статический метод принадлежит классу, а не объекту (и вызывается для класса). То же самое справедливо и для статической переменной.

Переменная-член может быть объявлена только для чтения (с ключевым словом **readonly**). Такие переменные тесно связаны с константами-членами (**1.2**), но существует важное различие — значения последних устанавливаются в программе в процессе компиляции и существуют на протяжении ее исполнения. А значения **readonly**

переменных-членов присваиваются им при создании объекта, и поэтому существуют, пока существует объект.

1.3. События объявляются подобно переменным-членам, но используются по-другому. При щелчке на кнопке (к примеру, ОК) в графическом пользовательском интерфейсе (**GUI**) соответствующий объект в программе генерирует (или возбуждает) событие (скажем, **OKButtonClick**), по которому определенная часть программы реагирует на действие пользователя. О приложении такого типа говорят, что оно управляется событиями, поскольку его следующее действие зависит от генерируемого события. Здесь уже неприменимо понятие программы, исполняющейся в той последовательности, в которой написаны ее операторы. Такая асинхронная способность используется в **GUI** (и других важных типах приложений), поскольку пользователь в любой момент может щелкнуть кнопкой мыши или нажать клавишу на клавиатуре.

СИНТАКСИЧЕСКИЙ БЛОК **П.1. Обзор элементов класса**

Определение_класса

```
class <Идентификатор_класса>
{
<Члены_класса>
}  ,
```

где

Члены_класса:

1. Данные-члены

2. Функции-члены

3. Вложенные_типы

Данные-члены

1.1. Переменные-члены
1.2. Константы .
1.3. События

Функции-члены

2.1. Методы
2.2. Конструкторы
2.3. Деструктор
2.4. Свойства
2.5. Индексаторы
2.6. Операции

Вложенные_типы

3.1. Вложенные_классы
3.2. Вложенные_структуры
3.3. Вложенные_перечисления

Примечания:

➢ В приведенном здесь определении класса основное место занимают его внутренние особенности, поэтому в нем опущены синтаксические элементы, связанные с модификацией доступа, наследованием и интерфейсами (см. Материалы к Практ. Зан. №**7**).

- Порядок элементов класса может быть любым внутри него, он не меняет семантики.

- <Функция-член> может быть: **а)** либо <Функция_экземпляра>, **б)** либо <Статическая_функция> (называемая также <Функция_класса>). Функция экземпляра всегда исполняется по отношению к конкретному объекту, поэтому последний необходим для ее вызова.

П.1.2. Функции-члены могут быть методами, конструкторами, деструкторами, свойствами, индексаторами и операциями:

2.1. Метод – знакомая конструкция, которая, тем не менее, обладает многими свойствами, как, например, ссылочные и выходные параметры, массивы параметров, перегрузка метода и ключевое слово **this**.

2.2. Конструкторы (рассмотрены выше).

2.3. Деструктор (называемый также завершающим методом). Объекты создаются и размещаются в определенной области памяти, где хранятся значения их переменных экземпляра и другие данные. Когда объект становится ненужным программе, занимаемую им память следует освободить для других объектов (иначе программа быстро начнет испытывать дефицит памяти). Деструктор, подобно своему собрату-конструктору, может содержать набор операторов, которые вызываются средой исполнения (их нельзя вызвать непосредственно из программы), когда происходит переполнение памяти.

2.4. Свойства (рассмотрены выше). Доступ к свойствам осуществляется так же, как к переменным-членам. Различие состоит в том, что свойство содержит операторы, которые исполняются подобно операторам метода, когда происходит обращение к нему. Свойства часто используются [вместо аксессоров и мутаторов (которые обычно имеют имена, наподобие **GetDistance** и **SetDistance**)] для доступа к закрытым переменным, поддерживая, таким образом, принцип инкапсуляции.

2.5. Индексаторы используются с классами, представляющими массивы. Так же, как свойства обеспечивают удобный доступ к отдельным переменным-членам, индексаторы выполняют эту функцию для массивов, размещенных внутри классов.

2.6. Операции. Иногда имеет смысл (для повышения читаемости кода) объединить два объекта с помощью операции. Например, можно написать оператор наподобие totalTime = myTime + yourTime, где все три переменных – объекты класса TimeInterval. Для достижения такой функциональности в классы включают операции-члены, которые задают набор команд, исполняющийся при объединении объектов в выражении с данной операцией. Этот процесс называют перегрузкой операции.

П.1.3. Вложенные типы — это классы, структуры, перечисления и интерфейсы, определенные в пределах тела класса. Они позволяют скрыть типы, которые используются только в пределах класса. Подобно тому, как вспомогательные методы объявляются закрытыми для уменьшения внешней сложности класса, вложенные типы помогают снизить количество классов, необходимых для сборки программы.