Тема 13. Свойства и индексаторы

Парадигмы программирования. C#

План

- Свойства
- Индексаторы

Свойства

Часть 1

Назначение свойств

- Свойства обеспечивают удобный способ инкапсуляции данных в классах
- Использование свойств обладает более компактным синтаксисом по сравнению с методами
  - Сравните

```
o.SetValue(o.GetValue( ) + 1);
o.Value++;
```

- Свойства добавляют большую гибкость по сравнению с использованием полей или методов
  - Возможны манипуляции с методами get и set

Реализация свойств

- Свойства обладают синтаксисом, похожим на синтаксис работы с полями
  - Метод get позволяет производить чтение значения свойства
  - Метод set позволяет записывать новое значение в свойство

```
class Button
{
    public string Caption // Свойство
    {
        get { return caption; }
        set { caption = value; }
    }
    private string caption; // Поле
}
...
Button myButton;
...
// Вызов метода "myButton.Caption.set"
myButton.Caption = "OK";
```

Сравнение свойств и полей

- Свойства являются «логическими полями»
  - Метод get может возвращать рассчитываемое значение
- Сходство
  - Синтаксис определения и использования одинаковы
    - Для определения используется тип и идентификатор
    - Допускается указывать модификатор доступа
    - Могут быть статическими
    - Могут скрывать члены базового типа
    - Синтаксис присваивания или чтения идентичны
- Различия
  - Свойства не являются переменными, поэтому не имеют адреса и на них нельзя устанавливать ссылку
  - Свойства не могут передаваться в качестве ref и out параметров методов

Пример «логического поля»

```
class User
{
    private readonly DateTime birthdate;

    public User(DateTime birthdate)
    {
        this.birthdate = birthdate;
    }
    public int Age
    {
        get
        {
            return DateTime.Now.Year -
                birthdate.Year;
        }
    }
}
```

Пример недопустимого использования свойтсв

```
class Example
{
    public string Property
    {
        get { ... }
        set { ... }
    }
    public string Field;
}
class Test
{
    static void Main()
    {
        Example eg = new Example();
        ByRef(ref eg.Property); // Ошибка компиляции
        ByOut(out eg.Property); // Ошибка компиляции
        ByRef(ref eg.Field); // Ошибки нет
        ByOut(out eg.Field); // Ошибки нет
    }
    static void ByRef(ref string name) { ... }
    static void ByOut(out string name) { ... }
}
```

Сравнение свойств с методами

- Сходства
  - И свойства, и методы содержат операторы исполняемого кода
  - И свойства, и методы используются для сокрытия деталей реализации (инкапсуляции)
  - И свойства, и методы могут размещаться в интерфейсе
  - И свойства, и методы могут быть виртуальными, абстрактными и переопределёнными
- Различия
  - Синтаксис свойств не предусматривает использование круглых скобок
  - Семантически свойства не могут возвращать void или принимать параметры

Виды свойств

- Свойства чтения/записи
  - Содержат методы get и set
- Свойства только для чтения
  - Содержат только метод get
  - Не являются константами
- Свойства только для записи
  - Содержат только метод set
  - Следует избегать использования свойств такого вида
- Статические свойства
  - Применимы только на уровне класса только для статических данных

Паттерн «ленивой инициализации»

```
public class Container
{
    public static TextReader In
    {
        get
        {
            if (reader == null)
            {
                reader = new
                    StreamReader("some_file.txt");
            }
            return reader;
        }
    }

    private static TextReader reader = null;
}
```

Использование свойств в интерфейсах

```
interface IBillable
{
    decimal Price { get; set; }
}

class Work : IBillable
{
    private decimal price = 1000;

    /// <summary>
    /// Gets or sets this work price.
    /// </summary>
    public decimal Price
    {
        get { return price; }
        set { price = value; }
    }
}
```

Сокращённое определение свойств и инициализатор

```
class Exapmle
{
    public string Text { get; set; }

    public Exapmle(string text)
    {
        Text = text;
    }

    public Exapmle() : this(String.Empty) { }
}

class Test
{
    public void Meth()
    {
        Exapmle example1 = new Exapmle("Some text 1");
        Exapmle example2 = new Exapmle() { Text = "Some text 2" };
    }
}
```

Индексаторы

Часть 2

Что такое индексатор?

- Индексатор предоставляет доступ к объекту, как к массиву (с помощью индекса, заключённого в квадратные скобки)
  - Удобен, если свойство может возвращать несколько значений
- Для определения индексатора
  - Создайте свойство с идентификатором this
  - Укажите тип индекса
- Для использования индексатора на чтение и запись
  - Используйте синтаксис, аналогичный синтаксису работы с массивом

Пример объявления и использования индексатора

```
class StringList
{
    private string[] list = new string[10];
    public string this[int index]
    {
        get { return list[index]; }
        set { list[index] = value; }
    }
}
...
StringList myList = new StringList();
// Запись индексатора
myList[3] = "Some string #3";
// Чтение индексатора
string myString = myList[3];
```

Сравнение индексаторов с массивами

- Сходство
  - И индексатор, и массив используют синтаксис работы с массивами
- Различия
  - Индексаторы могут иметь не только целочисленные индексы
  - Индексаторы могут быть перегруженными
    - Допускается определить несколько индексаторов с различными типами индексов
  - Индексаторы не являются переменными, поэтому на них нельзя создавать ссылки
    - Индексаторы нельзя использовать в качестве параметров ref или out

Пример индексатора с нецелочисленным индексом

```
class NickNames
{
    private Hashtable names = new Hashtable();
    public string this[string realName]
    {
        get { return (string)names[realName]; }
        set { names[realName] = value; }
    }
}
...
NickNames myNames = new NickNames();
myNames["Vasya Pupkin"] = "Vasek";
string myNickName = myNames["Vasya Pupkin"];
```

Пример перегруженного индексатора

```
class NickNames
{
    private Hashtable names = new Hashtable();
    public string this[string realName]
    {
        get { return (string)names[realName]; }
        set { names[realName] = value; }
    }
    public string this[int nameNumber]
    {
        get
        {
            string nameFound;
            // Проход по элементам
            // хэш-таблицы и заполнение nameFound
            return nameFound;
        }
    }
}
```

Сравнение индексаторов со свойствами

- Сходства
  - И индексаторы, и свойства используют методы get и set
  - И индексаторы, и свойства не имеют адреса
  - И индексаторы, и свойства не могут возвращать void
- Различия
  - Индексаторы могут быть перегружены
  - Индексаторы не могут быть статическими

Использование параметров для определения индексаторов

- При определении индексаторов
  - Укажите не менее одного параметра индексатора
  - Допускается использовать несколько параметров индексатора
  - Не используйте модификаторы параметров ref или out
  - Внутри индексатора проверьте значение индекса на попадание в диапазон допустимых значений

Пример класса string

- Класс string содержит неизменяемые строки
- В индексаторе класса string отсутствует метод set

```
class String
{
    public char this[int index]
    {
        get
        {
            if (index < 0 || index >= Length)
                throw new
                    IndexOutOfRangeException();
            ...
        }
    }
    ...
}
```

Пример класса BitArray

```
class BitArray
{
    public bool this[int index]
    {
        get
        {
            BoundsCheck(index);
            return (bits[index >> 5] & (1 << index)) != 0;
        }
        set
        {
            BoundsCheck(index);
            if (value)
            {
                bits[index >> 5] |= (1 << index);
            }
            else
            {
                bits[index >> 5] &= ~(1 << index);
            }
        }
    }
    private int[] bits;
}
```

Заключение

- Спасибо за внимание!