Государственный университет – Высшая школа экономики

Факультет Бизнес-Информатики

Кафедра Основ информатики
и прикладного программного обеспечения

Объектно-ориентированный анализ и программирование на языке **C# (C_Sharp)**

Материалы к 7-й лекции. Дополнение 1

Проф. Забудский Е.И.

Москва 2007

Лекция 7. Дополнение 1

Тема 9	Программная модель **Windows Forms** – основа для разработки приложений **.NET Framework** с графическим интерфейсом пользователя
	В дополнении 1 к Лекции 7 рассмотрены делегаты (delegate) и события (event)

см. также Материалы к Практ. зан. 8

NB	**Задание #2 на дом по итогам 2-го модуля (продолжение задания №1):** **Написать C#-программу – Банковский счет – применение полиморфизма.** **Программу реализовать в среде MS VS .NET 2005. Получить результат в консольном варианте.** **См. с. 38…40 в Материалах к 7-й лекции**
	Срок представления кода и результата – 21 февраля 2007 г.

Уважаемые студенты!

Основная цель, которую необходимо достигнуть в результате изучения дисциплины Объектно-ориентированный анализ и программирование – научиться разрабатывать компьютерные модели реальных и концептуальных систем соответствующих направлению Бизнес-информатика.

Необходимым условием усвоения дисциплины является ВАША самостоятельная работа

Советую Вам все материалы, подготовленные мной к лекциям и практическим занятиям, распечатать и прорабатывать их! Приведенные C#-программы реализовать в среде MS VS .NET 2005 и разобраться в них.

// **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).

Содержание дополнения 1 к Лекции 7

NB	Рекомендую перед рассмотрением материала, изложенного в разделах 10 и 11 (Лекция 7, с. 25…33), изучить настоящее Дополнение 1

1. Делегаты

Делегаты в понимании и использовании не сложнее любых других элементов языка C#. По своей структуре делегат — это объект, который может ссылаться на метод. То есть при создании делегата создается объект, который может хранить ссылку на метод. Кроме того, эта ссылка может быть использована для вызова метода. Следовательно, **делегат может вызывать метод, на который он ссылается (см. Прак. Зан. 4, Раздел 2. C# — строго типизированный язык, с. 7…14).**

Хотя метод не является объектом, для него выделяется некоторая область памяти. Адрес этой области памяти — это точка входа метода, и именно адрес инициируется при вызове метода. Значение адреса метода (области памяти) может быть присвоено делегату. Если делегат ссылается на метод, то данный метод можно вызывать с помощью этого делегата. Кроме того, один и тот же делегат может использоваться при вызове различных методов, для этого ему присваивается ссылка на требуемый метод. Важным свойством делегата является то, что он позволяет указать в коде программы вызов метода, но фактически **вызываемый метод определяется во время работы программы (то есть динамически)**, а не во время компилирования.

Для объявления делегата необходимо использовать ключевое слово **delegate**. Общая форма синтаксиса объявления делегата показана ниже.

delegate ret-type name (parameter-list);

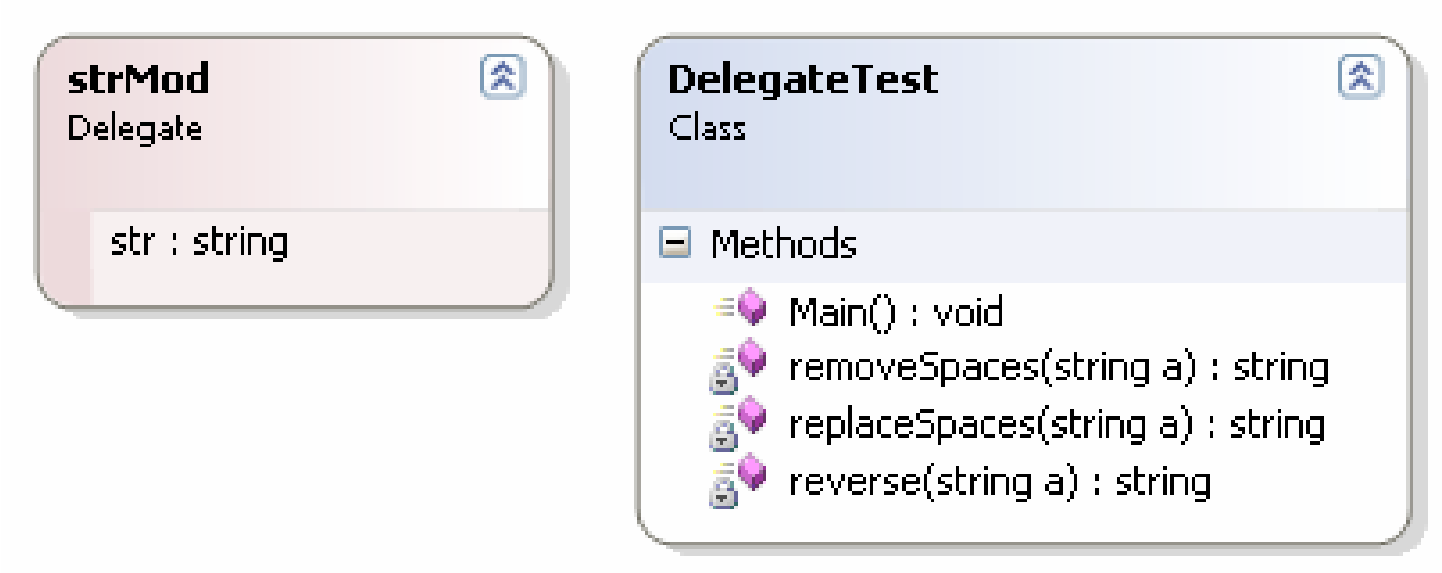
Здесь словосочетание **ret-type** — **тип значения, возвращаемого методами**, которые будут вызваны делегатом. Слово **name — имя делегата**. **Параметры, необходимые методам** при их вызове с помощью делегата, **указываются в списке параметров parameter-list**. Делегат может вызывать только те методы, типы возвращаемых значений и списки параметров которых совпадают с типами и списками, указанными при объявлении делегата.

Как уже говорилось, характерной особенностью делегата является возможность его использования для вызова любого метода, который соответствует подписи делегата. Это дает возможность определить во время выполнения программы, какой из методов должен быть вызван. Вызываемый метод может быть: **1)** методом экземпляра, ассоциированным с объектом (**см. Листинг 2**), либо **2**) статическим методом, ассоциированным с классом (**см. Листинг 1**). Метод можно вызвать только тогда, когда его подпись соответствует подписи делегата.

Рассмотрим простую программу, в которой используется делегат. (**Листинг 1**)

```
1  // Пример исп-ния делегата: делегату присв-ся ссылка на СТАТИЧЕСКИЕ м-ды.      Листинг 1
2  using System;
3
4  namespace ConsAppl_Schildt_Ch12_428deleg
5  {/* Объявление делегата strMod:
6    * "1)string" и "2)string" соответственно тип возвращаемого значения
7    * и параметр str (см. строку 50) методов - обработчиков событий */
8      internal delegate string strMod(string str);
9
10     class DelegateTest
11     {
12         static string replaceSpaces(string a)              // 1-й стат-й метод - обработчик события
13         { //Этот м-д заменяет пробелы дефисами в строке, перед-мой ему в качестве параметра
14             Console.WriteLine("1. Замена пробелов дефисами.");
15             return a.Replace(' ', '-');
16         }
17
```

```
18        static string removeSpaces(string a)                  // 2-й  стат-й метод - обработчик события
19        { //Этот м-д удаляет пробелы в строке, передаваемой ему в качестве параметра
20            string temp = "";
21            int i;
22            Console.WriteLine("2. Удаление пробелов.");
23            for (i = 0; i < a.Length; i++)
24                if (a[i] != ' ') temp += a[i];
25            return temp;
26        }
27
28        static string reverse(string a)                       // 3-й стат-й  метод - обработчик события
29        {//Этот м-д выводит строку, перед-мую ему в качестве параметра, в обратном порядке
30            string temp = "";
41            int i, j;
42            Console.WriteLine("3. Вывод строки в обратном порядке.");
43            for (j = 0, i = a.Length - 1; i >= 0; i--, j++)
44                temp += a[i];
45            return temp;
46        }
47
48        public static void Main()
49        {
50            string str;  // str – это параметр методов – обработчиков событий, строки 12,.. 18,.. 28,..
51            // Создание экз-ра strOp делегата strMod для 1-го стат-го м-да - Обр. Соб-я
              strMod strOp = new strMod(replaceSpaces);                  // строка 12
52  // перем-ной str присв-ся ссылка на м-д; эту ссылку на м-д сод-т зкз-р strOp дел-та strMod
    // Вызов стат-го м-да replaceSpaces с парам-м "2)string" (это Обр. Соб-я) с помощью дел-та
              str = strOp("Проверка работы методов.");
53            Console.WriteLine("\n(1) Преобразованная строка: " + str + "\n");
54            Console.WriteLine();
55
56            strOp = new strMod(removeSpaces);                          // строка 18
57  // Вызов  стат-го м-да removeSpaces с парам-м "2)string" (это Обр. Соб-я)с помощью дел-та
              str = strOp("Проверка работы методов.");
58            Console.WriteLine("\n(2) Преобразованная строка: " + str + "\n");
59            Console.WriteLine();
60
61            strOp = new strMod(reverse);                               // строка 28
62  // Вызов  стат-го м-да reverse с параметром "2)string"  (это Обр. Соб-я) с помощью делегата
              str = strOp("Проверка работы методов.");
63            Console.WriteLine("\n(3) Преобразованная строка: " + str + "\n");
64            Console.ReadLine();
65        }
66    }
67  }
```



1. Замена пробелов дефисами.

(1) Преобразованная строка: Проверка-работы-методов.

2. Удаление пробелов.

(2) Преобразованная строка: Проверкаработыметодов.

3. Вывод строки в обратном порядке.

(3) Преобразованная строка: .водотем ытобар акреворП

Рис. 1. Диаграмма классов и вывод программы на листинге 1

В программе объявляется делегат **strMod**, который принимает один параметр типа **string** и возвращает значение типа **string** (**строка 8**). В классе **DelegateTest** (**строки 10…**) объявляются три метода с модификатором **static** с **соответствующей** делегату подписью. Эти **методы предназначены для преобразования строки, передаваемой им в качестве параметра**. Обратите внимание, что метод **replaceSpases()** для замены пробелов дефисами использует один из методов класса **String**, который называется **Replace()** (**строка 15**).

В методе **Main()** создается объект **strOp** типа делегата **strMod**, **которому присваивается ссылка** (**то есть точка входа в метод**) **на метод replaceSpaces()** (**строка 51**). В операторе

```
51          strMod strOp = new strMod(replaceSpaces);
```

метод **replaceSpaces()** передается конструктору делегата в качестве параметра. Используется только имя метода без указания параметра (то есть при создании экземпляра делегата указывается только имя метода, на который должен ссылаться делегат). Подпись метода должна соответствовать подписи, определенной в объявлении делегата. Если это условие не выполнено, при компилировании программы произойдет ошибка компиляции.

В следующей строке (**52**) кода экземпляр делегата **strOp** используется для вызова метода **replaceSpaces().**

```
52          str = strOp("Проверка работы методов.");
```

Поскольку объект **strOp** ссылается на метод **replaceSpases()**, то вызывается именно этот метод.

Далее в программе экземпляру делегата **strOp** присваивается ссылка на метод **removeSpaces()**, после

чего экземпляр делегата **strOp** вызывается вновь. На этот раз инициируется метод **removeSpaces()** **(строки 56, 57, 58)**.

В завершение программы экземпляру делегата **strOp** присваивается ссылка на метод **revers()** **(строка 61)**, после чего вновь выполняется оператор

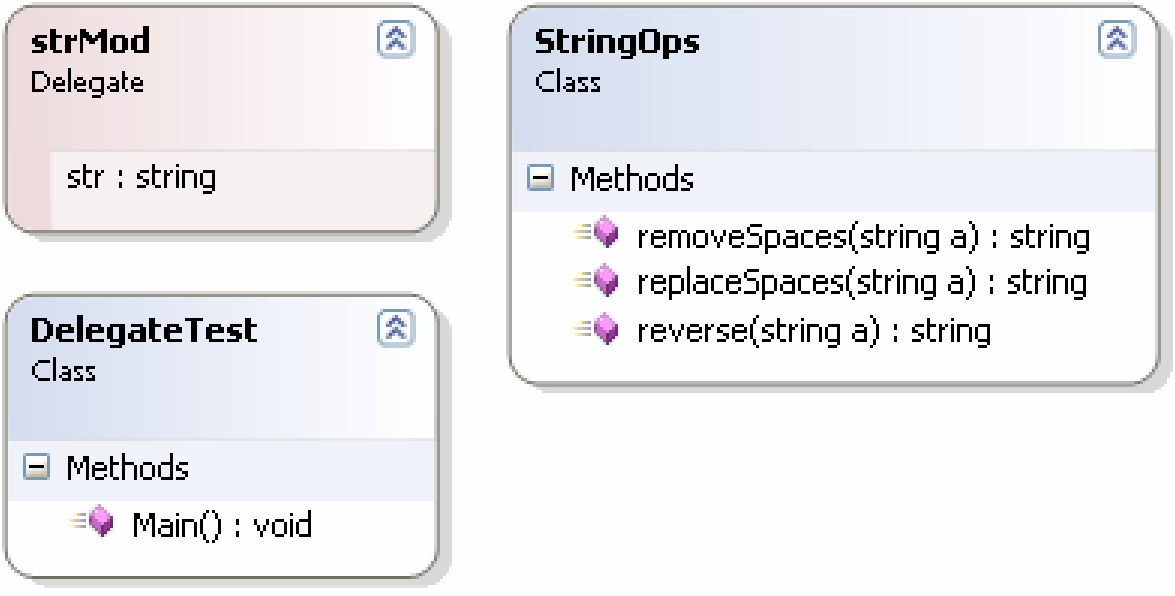
```
62          str = strOp("Проверка работы методов.");
```

На этот раз вызывается метод **revers()**. В данной программе при каждом вызове экземпляра делегата **strOp** будет вызываться тот метод, на который ссылается делегат **strOp** во время вызова. Таким образом, вызываемый метод определяется во время выполнения программы (**то есть динамически**), а не во время компилирования.

В этом примере (**листинг 1**) использовались методы с модификатором **static**, но делегатам может присваиваться также **ссылка на методы экземпляра**. Например, ниже приведена новая версия предыдущей программы (**листинг 2**), в которой **операции со строкой инкапсулированы внутри класса StringOps**.

```
1   // Пример испол-ния делегата: делегату присв-ся ссылка на м-ды ЭКЗЕМПЛЯРА.   Листинг 2
    using System;
2
3   namespace ConsAppl_Schildt_Ch12_428deleg
4   {                               /* Объявление делегата strMod:
5                                   * "1)string" и "2)string" соответственно тип возвращаемого значения
6                                   * и параметр методов - обработчиков событий */
7     internal delegate string strMod(string str);
8
9     class StringOps          ////////////////////// класс StringOps ////////////////////////////
10    {
11      public string replaceSpaces(string a)                  // 1-й метод - обработчик события
12      {         //Этот м-д заменяет пробелы дефисами в строке, перед-ой ему в кач-ве пар-ра
13        Console.WriteLine("1. Замена пробелов дефисами.");
14        return a.Replace(' ', '-');
15      }
16
17      public string removeSpaces(string a)                   // 2-й метод - обработчик события
18      {         //Этот м-д удаляет пробелы в строке, перед-ой ему в качестве параметра
19        string temp = "";
20        int i;
21        Console.WriteLine("2. Удаление пробелов.");
22        for (i = 0; i < a.Length; i++)
23          if (a[i] != ' ') temp += a[i];
24        return temp;
25      }
26
27      public string reverse(string a)                        // 3-й метод - обработчик события
28      {       //Этот м-д выводит строку, перед-мую ему в кач-ве пар-ра, в обратном порядке
29        string temp = "";
30        int i, j;
41        Console.WriteLine("3. Вывод строки в обратном порядке.");
42        for (j = 0, i = a.Length - 1; i >= 0; i--, j++)
```

```
43              temp += a[i];
44          return temp;
45      }
46  }                   ////////////////// конец класса StringOps//////////////////////////////
47
48      class DelegateTest          ////////////////////// класс DelegateTest //////////////////////////
49      {
50          public static void Main()
51          {
52              StringOps so = new StringOps(); // so  – объект (экз-р) класса StringOps
53
54              string str;
55          // Создание экз-ра strOp делегата strMod для 1-го м-да (это Обр. Соб-я) экз-ра so
                strMod strOp = new strMod(so.replaceSpaces);     // вызывается м-д для объекта so
56  // Вызов м-да replaceSpaces (экз-ра so) с парам-м "2)string" (это Обр. Соб-я) с пом-ю дел-та
                str = strOp("Проверка работы методов.");
57              Console.WriteLine("\n(1) Преобразованная строка: " + str + "\n");
58              Console.WriteLine();
59
60              strOp = new strMod(so.removeSpaces);          // вызывается м-д для объекта so
61  // Вызов м-да removeSpaces (экз-ра so) с парам-м "2)string" (это Обр. Соб-я)с пом-ю дел-та
                str = strOp("Проверка работы методов.");
62              Console.WriteLine("\n(2) Преобразованная строка: " + str + "\n");
63              Console.WriteLine();
64
65              strOp = new strMod(so.reverse);                 // вызывается м-д для объекта so
66    // Вызов м-да reverse (экз-ра so) с парам-м "2)string"  (это Обр. Соб-я) с помощью дел-та
                str = strOp("Проверка работы методов.");
67              Console.WriteLine("\n(3) Преобразованная строка: " + str + "\n");
68              Console.ReadLine();
69          }
70      }                       ////////////////////////// конец класса DelegateTest ////////////////////////
71  }
```



1. Замена пробелов дефисами.
(1) Преобразованная строка: Проверка-работы-методов.

2. Удаление пробелов.
(2) Преобразованная строка: Проверкаработыметодов.

3. Вывод строки в обратном порядке.
(3) Преобразованная строка: .водотем ытобар акреворП

Рис. 2. Диаграмма классов и вывод программы на листинге 2

В результате работы этой программы (**листинг 2**) на экран выводятся те же строки, что и в предыдущей (**листинг 1**). Но в данном случае делегат ссылается на методы, используя **экземпляр класса stringOps (то есть объект этого класса)**.

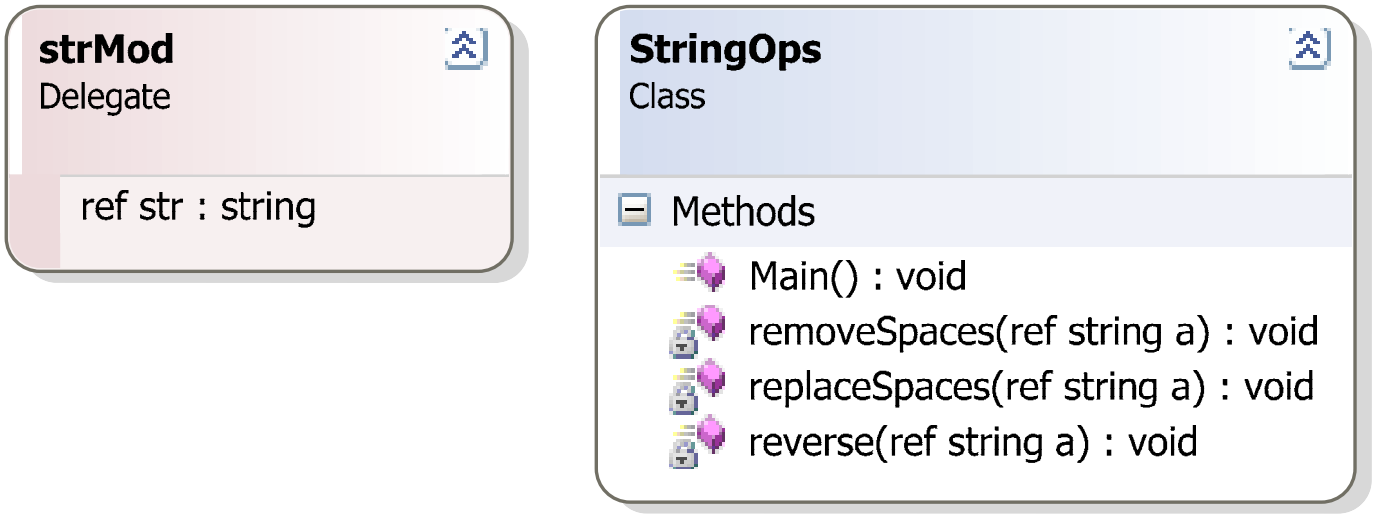
1.2. Многоадресность делегатов

Одним из замечательных свойств делегата является его способность хранить несколько адресов области памяти. Последовательно инициируя эти адреса, **делегат может один за другим вызывать соответствующие методы**. Эта характеристика делегатов называется **многоадресностью**. Такая последовательность вызываемых методов, или **цепочка методов**, конструируется достаточно просто: **1)** сначала необходимо создать экземпляр делегата, а затем **2)** использовать оператор += для добавления методов к цепочке. Хотя **фактически при этом создается не цепочка методов**, **а цепочка ссылок на методы**, в результате чего этот делегат становится **многоадресным**. Для удаления метода из цепочки используется оператор -=. **Единственное ограничение** — делегаты, хранящие несколько ссылок, должны иметь тип возвращаемого значения **void**.

Ниже приведена программа (**листинг 3**), демонстрирующая использование многоадресного делегата. Это новая версия предыдущей программы (**листинг 2**), в которой в качестве типа возвращаемого значения методов, преобразующих строку, используется тип **void**, а для возвращения вызывающей подпрограмме измененной строки применяется модификатор параметра **ref**.

```
1  using System;      // Пример исполь-ния делегата: МНОГОАДРЕСНОСТЬ делегатов.  Листинг 3
2                                                  /* Объявление делегата strMod:
                                          * "1)string" и "2)string" соответственно тип возвращаемого значения
                                                          * и параметр методов - обработчиков событий */
3  internal delegate void strMod(ref string str);
4
5     class StringOps          ////////////////////// класс StringOps ////////////////////////////
6  {
7     static void replaceSpaces(ref string a)                    // 1-й метод - обработчик события
8     {              //Этот м-д заменяет пробелы дефисами в строке, передав-ой ему в кач-ве пар-ра
9        Console.WriteLine("1. Замена пробелов дефисами.");
10       a = a.Replace(' ', '-');
11    }
12
13    static void removeSpaces(ref string a)                     // 2-й метод - обработчик события
14    {              //Этот м-д удаляет пробелы в строке, передаваемой ему в качестве параметра
15       string temp = "";
16       int i;
17       Console.WriteLine("2. Удаление пробелов.");
```

```
18          for (i = 0; i < a.Length; i++)
19              if (a[i] != ' ') temp += a[i];
20          a = temp;
21      }
22
23      static void reverse(ref string a)                    // 3-й метод - обработчик события
24      {               //Этот м-д выводит строку, перед-мую ему в кач-ве пар-ра, в обратном порядке
25          string temp = "";
26          int i, j;
27          Console.WriteLine("3. Вывод строки в обратном порядке.");
28          for (j = 0, i = a.Length - 1; i >= 0; i--, j++)
29              temp += a[i];
30          a = temp;
41      }
42
43      public static void Main()
44      {
45          strMod strOp;                                    // переменная strOp имееет тип делегата strMod
46                                      // Создание экземпляров делегата strMod
47          strMod replaceSp = new strMod(replaceSpaces);   // замена пробелов дефисами
48          strMod removeSp = new strMod(removeSpaces);   // удаление пробелов
49          strMod reverseStr = new strMod(reverse);             // реверс строки
50          string str = " Проверка работы методов.";
51
52          // присвоение двух ссылок с помощью делегата strMod
53          strOp = replaceSp;
54          strOp += reverseStr;
55
56          // 1-й вызов многоадресного делегата
57          strOp(ref str);
58          Console.WriteLine("\nПреобразованная строка: " + str + "\n\n\n");
59
60                          /* удаление из цепочки ссылок ссылки на м-д replaceSpaces
                                        и добав-ние ссылки на м-д removeSpaces */
61          strOp -= replaceSp;
62          strOp += removeSp;
63
64                                      // Присвоение переменной str строки в первоначальном виде
            str = " Проверка работы методов.";
65
66          // 2-й вызов многоадресного делегата
67          strOp(ref str);
68          Console.WriteLine("\nПреобразованная строка: " + str);
69          Console.ReadLine();
70      }
71  }               ////////////////////////// конец класса StringOps ////////////////////////////
```



1. Замена пробелов дефисами.
3. Вывод строки в обратном порядке.

Преобразованная строка: .водотем-ытобар-акреворП-

3. Вывод строки в обратном порядке.
2. Удаление пробелов.

Преобразованная строка: .водотемытобаракреворП

Рис. 3. Диаграмма классов и вывод программы на листинге 3

В методе **Main()** создаются четыре экземпляра делегата. Экземпляр **strOp** имеет значение **null**, поскольку при создании он не был инициирован (**строка 45**). Остальные три делегата ссылаются на методы, предназначенные для преобразования строки (**строки 47…49**). Далее в программе создается многоадресный делегат, который затем вызывает методы **replaceSpaces()** и **reverse()**. Это осуществляется с помощью трех операторов:

```
53        strOp = replaceSp;
54        strOp += reverseStr;
          ……
57        strOp(ref str);
```

В первом операторе (**строка 53**) экземпляру **strOp** делегата **strMod** присваивается ссылка на метод **replaceSp**. Затем с помощью оператора **+=** к цепочке присоединяется (**строка 54**) ссылка на метод **reverseStr**. При **1-м вызове** объекта **strOp** вызываются оба метода (**строка 57**). Первый метод заменяет пробелы дефисами, а второй изменяет порядок символов в строке.

В операторе (**строка 61**)

```
61        strOp - = replaceSp;
```

сначала из цепочки удаляется ссылка объекта **replaceSp**, а затем к цепочке прибавляется (**строка 62**) ссылка объекта **removeSp**:

```
62        strOp += removeSp;
```

Далее в программе вновь вызывается объект **strOp** с еще не преобразованной строкой в качестве аргумента (**строка 67**). Теперь из строки удаляются пробелы, и она выводится в обратном порядке (**строка 68**).

1.3. Преимущества использования делегатов

В предыдущем разделе описывалось использование делегатов, но не объяснялось, для чего был создан этот элемент языка. Делегаты в C# применяются по двум причинам, во-первых, они поддерживают события (см. следующий раздел), во-вторых, они предоставляют возможность выбора вызываемого метода во время выполнения программы, а не во время компилирования. Эта способность весьма полезна, когда необходимо создание базовой конструкции (программы), в которую **потом** можно **добавлять компоненты**.

2. События

Еще одним важным элементом языка C#, построенным на основе использования делегатов, является **событие**. Фактически событие — это автоматическое извещение о каком-либо произошедшем действии. Функционируют оно следующим образом: **1)** для объекта, который в соответствии с его определением (кодом) должен реагировать на какое-то событие, **РЕГИСТРИРУЕТСЯ обработчик этого события**; **2)** когда событие происходит, **ВЫЗЫВАЮТСЯ все зарегистрированные обработчики**. **Обработчики событий** создаются на основе **делегатов**.

События являются членами класса и объявляются с использованием ключевого слова **event**. Общий синтаксис объявления события представлен ниже.

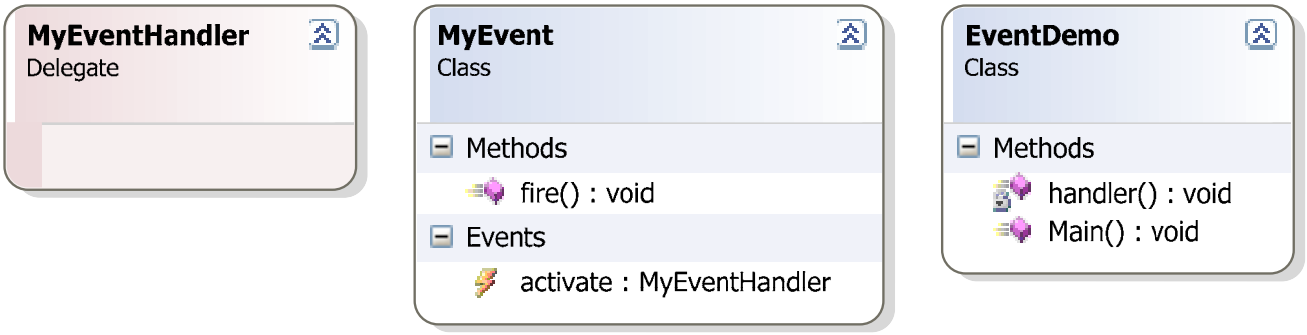
event event-delegate object-name;

Здесь словосочетание **event-delegate** — это имя делегата, используемого для поддержки события, а словосочетание **object-name** — имя создаваемого объекта события.

Рассмотрим следующую программу (**листинг 4**)

```
1  using System;                              // Пример использования СОБЫТИЯ.   Листинг 4
2
3  namespace ConsAppl_Schildt_Ch12_434event
4  {
5  // Объявление дел-та MyEventHandler на основе которого будет определено событие
       internal delegate void MyEventHandler();
6
7      class MyEvent                  // Объявление класса в котором инициируется событие.
8      {
9          public event MyEventHandler activate;                    // объявление СОБЫТИЯ
10
11         public void fire()                          // в этом методе инициируется событие
12         {
13             if (activate != null)        // определение условия для инициализации события
14                 activate();
15         }
16     }                    ////////////////////// конец класса MyEvent ////////////////////////////
17
18     class EventDemo
19     {
20         static void handler()        //  этот метод является обработчиком события
21         {
22             Console.Write("\nПроизошло событие");
23         }
24
25         public static void Main()
26         {
```

```
27          MyEvent evt = new MyEvent();                    // создание экземпляра события
28
29          // добавление м-да handler() к (возможной) цепочке обработчиков события
            evt.activate += new MyEventHandler(handler);
30
41          evt.fire();                                     // инициирование события
42          Console.ReadLine();
43      }
44  }               ////////////////////// конец класса EventDemo ////////////////////////
45  }
```



Произошло событие

Рис. 4. Диаграмма классов и вывод программы на листинге 4

Хотя данная программа достаточно проста, в ней содержатся все основные элементы, необходимые для правильной обработки события. Рассмотрим ее подробнее.

Программа начинается с объявления делегата **для хранения ссылок на обработчики события**.

```
5   internal delegate void MyEventHandler();
```

Все обработчики события активизируются с помощью делегата. Следовательно, делегат события определяет подпись события. **В данном примере событие не имеет параметров**, но их наличие допускается. Поскольку, как правило, событие является многоадресным, при его объявлении должен указываться тип возвращаемого значения **void**.

Далее в программе создается класс **MyEvent (строки 7…16)**, в котором объявляется событие **activate (строка 9)** и определяются условия его инициирования **(строка 13)**. Событие объявляется в следующей строке кода **(строка 9)**:

```
9       public event MyEventHandler activate;                   // объявление СОБЫТИЯ activate
```

Обратите внимание на используемый синтаксис. Так объявляются все типы событий.

Внутри класса **MyEvent** объявляется также метод **fire() (строки 11…15)**, который будет вызываться программой для инициализации события. Условие, при котором выполняется инициализация, определено с помощью оператора **if**.

```
13          if (activate != null)       // определение условия для инициализации события
```

Обратите внимание, что обработчик **вызывается только в случае**, если делегат, ассоциированный с событием **activate**, имеет значение, отличное от **null**.

Внутри класса **EventDemo** создается обработчик события **handler()** (**строки 20…23**) В этом примере обработчик события просто выводит сообщение **" Произошло событие "**. Но можно определить обработчик так, чтобы его операторы выполняли более содержательные действия. В методе **Main()**

создается объект **evt** типа **MyEvent (строка 27)**, а метод **handler()** **РЕГИСТРИРУЕТСЯ** как обработчик для события, объявленного в этом классе.

```
27          MyEvent evt = new MyEvent();                    // создание экземпляра события
28
29          // добавление м-да handler() к (возможной) цепочке обработчиков события
            evt.activate += new MyEventHandler(handler);
```

Обратите внимание, что обработчик добавляется с использованием оператора **+=**. **События поддерживают только применение операторов** **+=** и **-=**.

Наконец, происходит инициирование события.

```
41          evt.fire();                                     // инициирование события
```

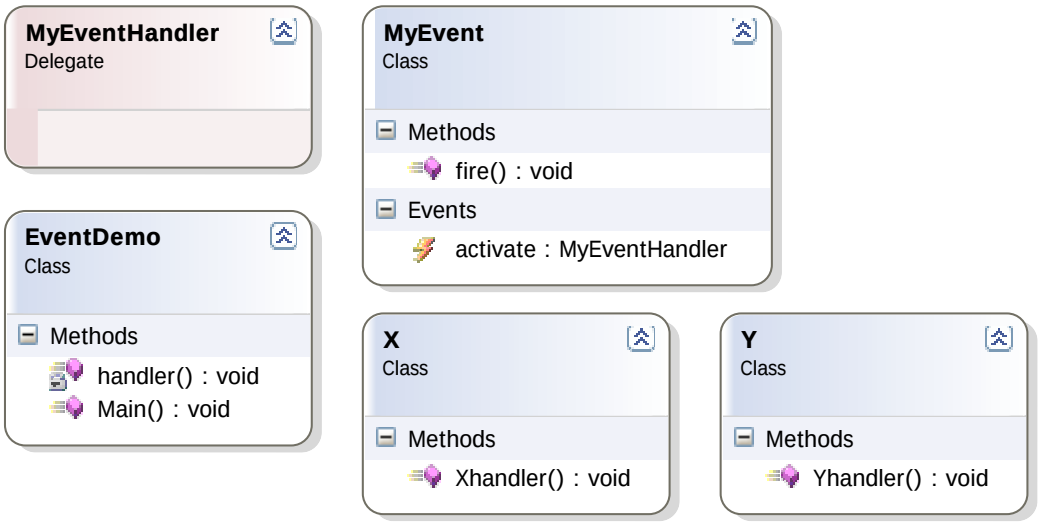
Вызов метода **fire()** приводит к вызову всех зарегистрированных обработчиков события. В данном случае вызывается только один зарегистрированный обработчик, но их может быть и несколько.

2.1. Широковещательное событие

Как уже говорилось ранее, события создаются на основе делегатов. А поскольку делегаты могут быть многоадресными, то события могут активизировать несколько обработчиков, даже те, которые были определены в других объектах. Такие события называются **широковещательными** (что является синонимом слова **многоадресные**). Их использование позволяет множеству объектов «реагировать» на извещение о событии. Ниже приведена программа (**листинг 5**), в которой используется широковещательное событие.

```
1   using System;// Демонстрация использования ШИРОКОВЕЩАТЕЛЬНОГО события.  Листинг 5
2   namespace ConsAppl_Schildt_Ch12_436event
3   {
4       // объявление ДЕЛЕГАТА MyEventHandler на основе которого будет определено событие
        delegate void MyEventHandler();
5
6       class MyEvent                       // Объявление класса в котором инициируется событие.
7       {
8           public event MyEventHandler activate;                   // объявление СОБЫТИЯ
9
10          public void fire()                                      // в этом методе инициируется СОБЫТИЕ
11          {
12              if (activate != null)
13                  activate();
14          }
15      }                   ////////////////////////// конец класса MyEvent ////////////////////////////////
16
17      class X
18      {
19          public void Xhandler()                  // 2-й обработчик события
20          {
21              Console.WriteLine("\nСобытие (event) получено объектом класса X");
22          }
23      }                   ////////////////////////// конец класса X ////////////////////////////////
24
25      class Y
```

```
26      {
27          public void Yhandler()                  // 3-й обработчик события
28          {
29              Console.WriteLine("\nСобытие (event) получено объектом класса Y" + "\n");
30          }
41      }                       ////////////////////////// конец класса Y ////////////////////////////////
42
43      class EventDemo
44      {
45          static void handler()                   // 1-й обработчик события
46          {
47              Console.WriteLine("\n\nСобытие (event) получено объектом класса EventDemo");
48          }
49
50          public static void Main()
51          {
52              MyEvent evt = new MyEvent();            // конструктор класса MyEvent
53              X xOb = new X();                    // конструктор класса X
54              Y yOb = new Y();                // конструктор класса Y
55
56  // ДОБАВ-Е м-дов handler(), Xhandler() и Yhandler() к (возможной) цепочке обраб-ов события
57              evt.activate += new MyEventHandler(handler);
58              evt.activate += new MyEventHandler(xOb.Xhandler);
59              evt.activate += new MyEventHandler(yOb.Yhandler);
60
61              evt.fire();                                     // 1-е инициирование события
62              Console.WriteLine();
63
64                                  // УДАЛЕНИЕ одного обработчика из цепочки
                evt.activate - = new MyEventHandler(xOb.Xhandler);
65              evt.fire();                                     // 2-е инициирование события
66              Console.ReadLine();
67          }
68      }               ////////////////////////// конец класса EventDemo ////////////////////////////////
69  }
```



```
Событие (event) получено объектом класса EventDemo
Событие (event) получено объектом класса X
Событие (event) получено объектом класса Y

Событие (event) получено объектом класса EventDemo
Событие (event) получено объектом класса Y
```

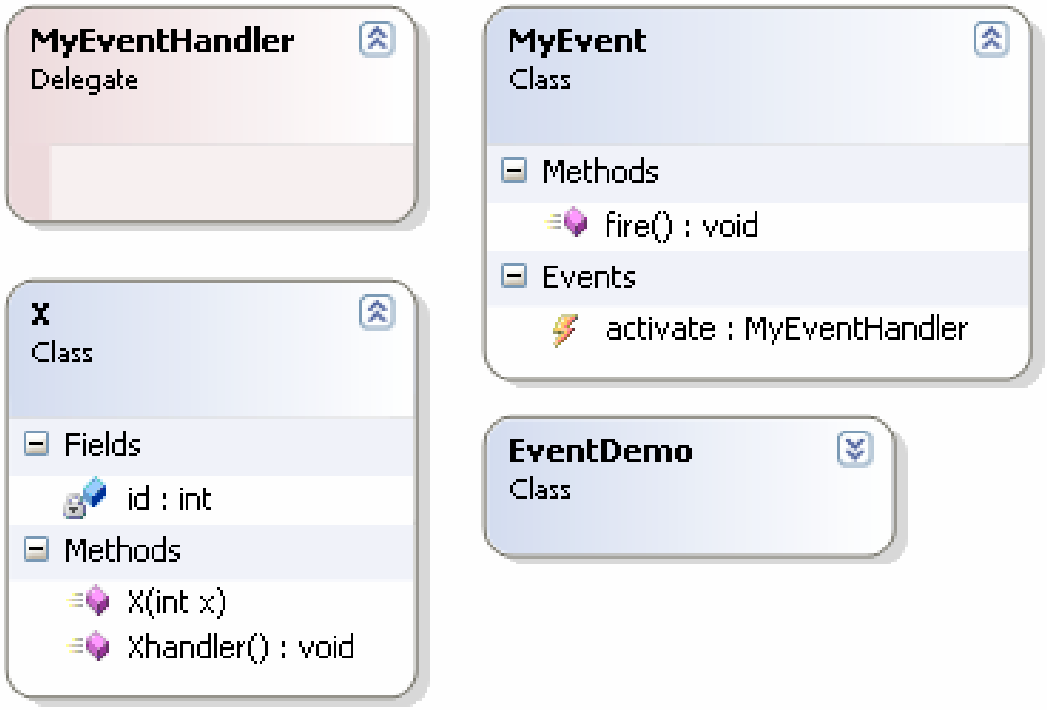
Рис. 5. Диаграмма классов и вывод программы на листинге 5

В этой программе создаются два дополнительных класса **X (строка 17…)** и **Y (строка 25…)**, в которых также определены обработчики событий, **совместимые по своей подписи** с типом **MyEventHandler (то есть они не имеют параметров, а тип возвращаемого значения void)**. Следовательно, эти обработчики могут стать частью цепочки. Обратите внимание, что обработчики **Xhandler() (строка 19…) и Yhandler() (строка 27…)**, определенные в классах **X** и **Y**, не имеют модификатора **static**. Это означает, что сначала в программе создаются объекты каждого класса (**строки 52, 53, 54**), а уже затем добавляется относящийся к объекту обработчик (**строки 57, 58, 59**).

Необходимо понимать, что сгенерированные события передаются отдельно каждому экземпляру объекта, а не классу в целом. Следовательно, для получения извещения о событии должен быть зарегистрирован (определен) обработчик каждого объекта класса. Например, в приведенной ниже программе происшедшее событие (вызов метода **fire**) инициирует обработчики трех объектов класса **X** (**листинг 6**).

```
1  // На происшедшее событие реагируют обраб-ки каждого из 3-х объектов класса X. Листинг 6
2  using System;
3
4  namespace ConsAppl_Schildt_Ch12_437event
5  {
6      // Объявление ДЕЛЕГАТА MyEventHandler на основе которого будет определено событие
       delegate void MyEventHandler();
7
8      class MyEvent                    // Объявление класса в котором инициируется событие.
9      {
10         public event MyEventHandler activate;                    // объявление СОБЫТИЯ
11
12         public void fire()                                  // в этом методе ИНИЦИИРУЕТСЯ событие
13         {
14             if (activate != null)
15                 activate();
16         }
17     }                    ////////////////////////// конец класса MyEvent ////////////////////////////
18
19     class X
20     {
21         int id;
22         public X(int x)                                        // конструктор класса X
23         {
24             id = x;
25         }
26
```

```
27      public void Xhandler()                                  // ОБРАБОТЧИК события
28      {
29          Console.WriteLine("\nСобытие получено объектом № " + id);
30      }
41  }                   ////////////////////////// конец класса X ////////////////////////////////
42
43  class EventDemo
44  {
45      public static void Main()
46      {
47          MyEvent evt = new MyEvent();          // конструктор класса MyEvent
48          X o1 = new X(1);                       // 1-й объект класса X
49          X o2 = new X(2);                      // 2-й объект класса X
50          X o3 = new X(3);                      // 3-й объект класса X
51
52          evt.activate += new MyEventHandler(o1.Xhandler);
53          evt.activate += new MyEventHandler(o2.Xhandler);
54          evt.activate += new MyEventHandler(o3.Xhandler);
55
56          evt.fire(); // инициирование события
57          Console.ReadLine();
58      }
59  }                   ////////////////////////// конец класса EventDemo ////////////////////////////////
60  }
```




```
Событие получено объектом № 1
Событие получено объектом № 2
Событие получено объектом № 3
```

Рис. 6. Диаграмма классов и вывод программы на листинге 6

Как видите, для каждого объекта отдельно **РЕГИСТРИРУЕТСЯ** обработчик события (**строки 52, 53, 54**), и можно сказать, что каждый объект получает отдельное извещение о происшедшем событии.

Примечание

В C# разрешается создание любого типа события. Однако для совместимости компонентов со средой **.NET Framework** необходимо придерживаться традиций программирования, свойственных компании **Microsoft** (**см. Раздел 11. Рекомендации по обработке событий в среде .NET Framework в лекции 7, с. 29,сл.**).

Литература

Базовый учебник

1. Мейер Б. Объектно-ориентированное конструирование программных систем. М.: Русская Редакция, 2005.

Основная

2. Буч Г., Якобсон А., Рамбо Дж. **UML**. С.-Петербург: Питер, 2006.

3. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования. С.-Петербург: Питер, 2006.

4. Забудский Е.И. Учебно-методические материалы по дисциплине «Объектно-ориентированный анализ и программирование». М.: Кафедра ОИиППО ГУ-ВШЭ, 2005,
Internet-ресурс – http://new.hse.ru/C7/C17/zabudskiy-e-i/default.aspx или http://zei.narod.ru/ .

5. Кватрани Т. Визуальное моделирование с помощью Rational Rose 2002 и UML. М.: Вильямс, 2003.

6. Лафоре Р. Объектно-ориентированное программирование в C++. С.-Петербург: Питер, 2005.

7. Троелсен Э. C# и платформа **.**NET. С.-Петербург: Питер, 2006.

8. Синтес А. Освой самостоятельно объектно-ориентированное программирование за 21 день. Москва; С.-Петербург; Киев: Вильямс, 2002.

Дополнительная – Internet-ресурсы

9. Новые книги раздела **C#**. – http://books.dore.ru/bs/f6sid16.html .

10. **C#** и ***.NET*** по шагам. – http://www.firststeps.ru .

11. ***UML*** – язык графического моделирования. – http://www.uml.org/ .

12. ***JUnit*** – каркас тестирования для испытания *Java*-классов. – http://www.junit.org .

13. Пакет объектного моделирования ***Rational Rose***. – http://www-306.ibm.com/software/rational/ .

Дополнительная – книги

14. Мэтт Вайсфельд. Объектно-ориентированный подход: Java, **.**NET, C++. М.: КУДИЦ-ОБРАЗ, 2005.

15. Дж. Кьоу, М. Джеанини. Объектно-ориентированное программирование. С.-Петербург: Питер, 2005.