Теория и практика программирования

Шпилёв Пётр Валерьевич

Санкт-Петербургский государственный университет
Математико-механический факультет
Кафедра статистического моделирования

Лекция 8

Санкт-Петербург
2015 г.

e-mail: pitshp@hotmail.com

Общая форма объявления делегата

```
delegate возвращаемый_тип имя(список_параметров);
```

Общая форма объявления делегата

```
delegate возвращаемый_тип имя(список_параметров);
```

Пример 1:

```
using System; // Объявить тип делегата.
delegate string StrMod(string str);
class DelegateTest
{
    static string Reverse(string s){ ... }
    static string RemoveSpaces(string s){ ... }
    static void Main()
    {
        // Сконструировать делегат.
        StrMod strOp = new StrMod(Reverse);
        Console.WriteLine("Результат:" + strOp("Это простой тест."));
        StrMod strOp = new StrMod(RemoveSpaces);
        Console.WriteLine("Результат:" + strOp("Это простой тест."));
    }
}
```

Общая форма объявления делегата

```
delegate возвращаемый_тип имя(список_параметров);
```

Пример 2:

```
using System; // Объявить тип делегата.
delegate string StrMod(string str);
class DelegateTest
{
    static string Reverse(string s){ ... }
    static string RemoveSpaces(string s){ ... }
    static void Main()
    {
        // Сконструировать делегат.
        StrMod strOp = Reverse;
        Console.WriteLine("Результат:" + strOp("Это простой тест."));
        StrMod strOp = RemoveSpaces;
        Console.WriteLine("Результат:" + strOp("Это простой тест."));
    }
}
```

Общая форма объявления делегата

```
delegate возвращаемый_тип имя(список_параметров);
```

Пример 3:

```
using System; // Объявить тип делегата.
delegate string StrMod(string str);
class StringOps
{
    string Reverse(string s){ ... }
    string RemoveSpaces(string s){ ... }
}
class DelegateTest
    static void Main()
    {
        StringOps so = new StringOps();
        // Инициализировать делегат.
        StrMod strOp = so.Reverse;
        Console.WriteLine("Результат:" + strOp("Это простой тест."));
        StrMod strOp = so.RemoveSpaces;
        Console.WriteLine("Результат:" + strOp("Это простой тест."));
    }
}
```

Групповая адресация

Групповая адресация — это возможность создать список, или цепочку вызовов, для методов, которые вызываются автоматически при обращении к делегату. Для добавления метода в цепочку служит оператор $+$ или $+=$. Для удаления метода из цепочки служит оператор $-$ или $-=$.

Пример 4:

```
using System; // Объявить тип делегата.
delegate string StrMod(ref string str);
class MultiCastDemo
{
    static string Reverse(ref string s){ ... }
    static string RemoveSpaces(ref string s){ ... }
    static void Main()
    {
        string str = "Это простой тест.";
        // Сконструировать делегаты.
        StrMod strOp;
        StrMod reverseStr = Reverse;
        StrMod removeSp = RemoveSpaces;
        // Организовать групповую адресацию.
        strOp = reverseStr;
        strOp += removeSp;
        Console.WriteLine("Результат:" + strOp(str));
    }
}
```

Пример 4:

```
using System; // Объявить тип делегата.
delegate string StrMod(ref string str);
class MultiCastDemo
{
    static string Reverse(ref string s){ ... }
    static string RemoveSpaces(ref string s){ ... }
    static void Main()
```

Упражнение 8.1

Реализовать методы Reverse() и RemoveSpaces() в примерах 1-4. Добавить 1-2 своих метода. Продемонстрировать применение делегатов для этих примеров.

```
        StrMod removeSp = RemoveSpaces;
        // Организовать групповую адресацию.
        strOp = reverseStr;
        strOp += removeSp;
        Console.WriteLine("Результат:" + strOp(str));
    }
}
```

Ковариантность

Ковариантность позволяет присвоить делегату метод, возвращаемым типом которого служит класс, производный от класса, указываемого в возвращаемом типе делегата.

Ковариантность

Ковариантность позволяет присвоить делегату метод, возвращаемым типом которого служит класс, производный от класса, указываемого в возвращаемом типе делегата.

Контравариантность

Контравариантность позволяет присвоить делегату метод, типом параметра которого служит класс, являющийся базовым для класса, указываемого в объявлении делегата

Пример:

```
using System;
class A {...}
//Класс B, производный от класса A.
class B : A {...}
//Этот делегат возвращает объект класса A и
//принимает объект класса B в качестве аргумента.
delegate A ChangeIt(B obj);
class CoContraVariance
{
    static A IncrA(A obj) {...}
    static B IncrB(B obj) {...}
    static void Main()
    {
        B Bob = new B();
        ChangeIt change = IncrA;
        A Aob = change(Bob);
        change = IncrB;
        Bob = (B) change (Bob);
    }
}
```

Пример:

```
using System;
class A {...}
//Класс B, производный от класса A.
class B : A {...}
//Этот делегат возвращает объект класса A и
//принимает объект класса B в качестве аргумента.
delegate A ChangeIt(B obj);
```

Упражнение 8.2

Продемонстрировать свойства ковариантности и контравариантности делегатов на примере своих классов и методов.

```
    static void Main()
    {
        B Bob = new B();
        ChangeIt change = IncrA;
        A Aob = change(Bob);
        change = IncrB;
        Bob = (B) change (Bob);
    }
}
```

Общий вид

```
delegate возвращаемый_тип имя_делегата(список_параметров);
class имя_класса {
static void Main()
{
имя_делегата имя_экземпляра = delegate(список_параметров)
{ ... }; // обратите внимание на точку с запятой после "}"
// вызов анонимного метода
имя_экземпляра(список_параметров);
}
```

Общий вид

```
delegate возвращаемый_тип имя_делегата(список_параметров);
class имя_класса {
static void Main()
{
имя_делегата имя_экземпляра = delegate(список_параметров)
{ ... }; // обратите внимание на точку с запятой после "}"
// вызов анонимного метода
имя_экземпляра(список_параметров);
}
```

Замечание

Локальная переменная, в область действия которой входит анонимный метод, называется внешней переменной. Такие переменные доступны для использования в анонимном методе. И в этом случае внешняя переменная считается захваченной. Захваченная переменная существует до тех пор, пока захвативший ее делегат не будет собран в "мусор".

Общий вид

```
delegate возвращаемый_тип имя_делегата(список_параметров);
class имя_класса {
static void Main()
{
имя_делегата имя_экземпляра = delegate(список_параметров)
{ ... }; // обратите внимание на точку с запятой после "}"
// вызов анонимного метода
имя_экземпляра(список_параметров);
}
```

Замечание

Локальная переменная, в область действия которой входит анонимный метод, называется внешней переменной. Такие переменные доступны для использования в анонимном методе. И в этом случае внешняя переменная считается захваченной. Захваченная переменная существует до тех пор, пока захвативший ее делегат не будет собран в "мусор".

Упражнение 8.3

Продемонстрировать применение захваченной переменной.

Общая форма одиночного лямбда-выражения

(список_параметров) => выражение

Общая форма одиночного лямбда-выражения

(список_параметров) => выражение

Пример I:

```
count => count + 2 //значение параметра count увеличивается на 2.
```

Общая форма одиночного лямбда-выражения

(список_параметров) => выражение

Пример I:

```
count => count + 2 //значение параметра count увеличивается на 2.
```

Пример II:

```
n => n % 2 == 0 //выражение возвращает логическое значение true, если n четное, а иначе — false.
```

Общая форма одиночного лямбда-выражения

(список_параметров) => выражение

Пример I:

```
count => count + 2 //значение параметра count увеличивается на 2.
```

Пример II:

```
n => n % 2 == 0 //выражение возвращает логическое значение true, если n четное, а иначе — false.
```

Пример III:

```
(low, high, val) => val >= low && val <= high;
//выражение возвращает логическое значение true, если val внутри границ диапазона.
```

Общая форма одиночного лямбда-выражения

(список_параметров) => выражение

Пример I:

```
count => count + 2 //значение параметра count увеличивается на 2.
```

Пример II:

```
n => n % 2 == 0 //выражение возвращает логическое значение true, если n четное, а иначе — false.
```

Пример III:

```
(low, high, val) => val >= low && val <= high;
//выражение возвращает логическое значение true, если val внутри границ диапазона.
```

Упражнение 8.4

Использовать делегаты для применения описанных выше лямбда-выражений.

Общая форма одиночного лямбда-выражения

(список_параметров) => выражение

Пример I:

```
count => count + 2 //значение параметра count увеличивается на 2.
```

Пример II:

```
n => n % 2 == 0 //выражение возвращает логическое значение true, если n четное, а иначе — false.
```

Пример III:

```
(low, high, val) => val >= low && val <= high;
//выражение возвращает логическое значение true, если val внутри границ диапазона.
```

Упражнение 8.4

Использовать делегаты для применения описанных выше лямбда-выражений.

Замечание

Внешние переменные могут использоваться и захватываться в лямбда-выражениях таким же образом, как и в анонимных методах.

Общая форма одиночного лямбда-выражения

(список_параметров) => выражение

Пример I:
count => count + 2 //значение параметра count увеличивается на 2.

Пример II:
n => n % 2 == 0 //выражение возвращает логическое значение true, если n четное, а иначе — false.

Пример III: (low, high, val) => val >= low && val <= high;
//выражение возвращает логическое значение true, если val внутри границ диапазона.

Упражнение 8.4

Использовать делегаты для применения описанных выше лямбда-выражений.

Упражнение 8.5

Продемонстрировать применение захваченной переменной в лямбда-выражении.

Общая форма блочного лямбда-выражения

(список_параметров) => {тело_выражения}

Общая форма блочного лямбда-выражения

(список_параметров) => {тело_выражения}

Упражнение 8.6

Продемонстрировать применение блочного лямбда-выражения для вычисления факториала.

Общая форма блочного лямбда-выражения

(список_параметров) => {тело_выражения}

Упражнение 8.6

Продемонстрировать применение блочного лямбда-выражения для вычисления факториала.

Упражнение 8.7

Переписать упражнение 8.1 используя блочные лямбда-выражения.

Пример:

```
using System;
delegate void MyEventHandler(); //Объявить тип делегата для события.
class MyEvent //Объявить класс, содержащий событие.
{
    public event MyEventHandler SomeEvent;
    public void OnSomeEvent() //Метод для запуска события.
    {
        if (SomeEvent != null) SomeEvent();
    }
}
class EventDemo
{
    static void Handler() //Обработчик события.
    {Console.WriteLine("Произошло событие");}
    static void Main()
    {
        MyEvent evt = new MyEvent();
        //Добавить метод Handler() в список событий.
        evt.SomeEvent += Handler;
        evt.OnSomeEvent(); //Запустить событие.
    }
}
```

Пример:

```
using System;
delegate void MyEventHandler(); //Объявить тип делегата для события.
class MyEvent //Объявить класс, содержащий событие.
{
    public event MyEventHandler SomeEvent;
    public void OnSomeEvent() //Метод для запуска события.
    {
```

Упражнение 8.8

Продемонстрировать групповую адресацию события. Использовать как методы экземпляра так и статические методы для обработки события.

```
{
    static void Handler() //Обработчик события.
    {Console.WriteLine("Произошло событие");}
    static void Main()
    {
        MyEvent evt = new MyEvent();
        //Добавить метод Handler() в список событий.
        evt.SomeEvent += Handler;
        evt.OnSomeEvent(); //Запустить событие.
    }
}
```

Расширенная форма оператора event

```
event делегат_события имя_события
{
    add
    {
        // Код добавления события в цепочку событий.
    }
    remove
    {
        // Код удаления события из цепочки событий.
    }
}
```

Расширенная форма оператора event

```
event делегат_события имя_события
{
    add
    {
        // Код добавления события в цепочку событий.
    }
    remove
    {
        // Код удаления события из цепочки событий.
    }
}
```

Замечание

Когда вызывается аксессор add или remove, он принимает в качестве параметра добавляемый или удаляемый обработчик. Как и в других разновидностях аксессоров, этот неявный параметр называется value.

Расширенная форма оператора event

```
event делегат_события имя_события
{
    add
    {
        // Код добавления события в цепочку событий.
    }
    remove
    {
        // Код удаления события из цепочки событий.
    }
}
```

Упражнение 8.9

Используя расширенную форму оператора event, реализовать обработку не более чем 3-х событий*.

**Подсказка: использовать массив событий.*

Упражнение 8.10

Применить анонимные методы вместе с событиями.

Упражнение 8.10

Применить анонимные методы вместе с событиями.

Упражнение 8.11

Применить лямбда-выражения вместе с событиями.

Общая форма обработчика события

```
void обработчик(object отправитель, EventArgs e)
{
    //...
}
```

Общая форма обработчика события

```
void обработчик(object отправитель, EventArgs e)
{
    //...
}
```

Замечание

Как правило, отправитель — это параметр, передаваемый вызывающим кодом с помощью ключевого слова this. А параметр e типа EventArgs содержит дополнительную информацию о событии и может быть проигнорирован, если он не нужен.

Пример:

```
using System;
using System.Windows.Forms;
using System.ComponentModel;
class MyForm : Form
{
    private Button button1;
    public MyForm()
    {
        button1 = new Button();
        button1.Text = "MyButton";
        button1.Click += button1_Click;
        Text = "Form1 ";
        StartPosition = FormStartPosition.CenterScreen;
        Controls.Add(button1);
    }
    private void button1_Click(object sender, EventArgs e)
    {
        Console.WriteLine("Объект: " + sender.ToString());
        Console.WriteLine("Тип события: " + e.GetType());
        Form.ActiveForm.Close();
    }
}
```

```
class MyFormDemo
{
    static void Main(string[] args)
    {
        Application.Run(new MyForm());
        Console.WriteLine("Форма закрыта ");
    }
}
```

```
class MyFormDemo
{
    static void Main(string[] args)
    {
        Application.Run(new MyForm());
        Console.WriteLine("Форма закрыта ");
    }
}
```

Замечание

Для запуска примера нужно добавить в проекте ссылки на сборки "System" и "System.Windows.Forms"(Project → References → Добавить ссылку).

```
class MyFormDemo
{
    static void Main(string[] args)
    {
        Application.Run(new MyForm());
        Console.WriteLine("Форма закрыта ");
    }
}
```

Упражнение 8.12

Написать свой пример формирования .NET-совместимого события. Использовать собственные классы с обработчиками события вида:

```
public void Handler(object source, EventArgs arg)
{
    //...
}
```

Встроенный обобщенный делегат EventHandler<TEventArgs>

В среде .NET Framework предоставляется встроенный обобщенный делегат EventHandler<TEventArgs>. Тип TEventArgs обозначает тип аргумента, передаваемого параметру EventArgs события.

Встроенный обобщенный делегат EventHandler<TEventArgs>

В среде .NET Framework предоставляется встроенный обобщенный делегат EventHandler<TEventArgs>. Тип TEventArgs обозначает тип аргумента, передаваемого параметру EventArgs события.

Пример:

```
//Объявить класс, производный от класса EventArgs.
class MyEventArgs : EventArgs{...}
class MyEvent//Объявить класс, содержащий событие.
{
     public event EventHandler<MyEventArgs> SomeEvent;
    //Метод запускающий событие SomeEvent.
     public void OnSomeEvent(){...}
}
```

Встроенный обобщенный делегат EventHandler<TEventArgs>

В среде .NET Framework предоставляется встроенный обобщенный делегат EventHandler<TEventArgs>. Тип TEventArgs обозначает тип аргумента, передаваемого параметру EventArgs события.

Встроенный необобщенный делегат EventHandler

В среде .NET Framework предоставляется встроенный необобщенный делегат EventHandler. Он может быть использован для объявления обработчиков событий, которым не требуется дополнительная информация о событиях. В этом случае в качестве параметра EventArgs события используется статический параметр EventArgs.Empty.

Встроенный обобщенный делегат EventHandler<TEventArgs>

В среде .NET Framework предоставляется встроенный обобщенный делегат EventHandler<TEventArgs>. Тип TEventArgs обозначает тип аргумента, передаваемого параметру EventArgs события.

Встроенный необобщенный делегат EventHandler

В среде .NET Framework предоставляется встроенный необобщенный делегат EventHandler. Он может быть использован для объявления обработчиков событий, которым не требуется дополнительная информация о событиях. В этом случае в качестве параметра EventArgs события используется статический параметр EventArgs.Empty.

Упражнение 8.13

Переделать упражнение 8.12 удалив свой делегат и добавив встроенный обобщенный или необобщенный делегат EventHandler.

Встроенный обобщенный делегат EventHandler<TEventArgs>

В среде .NET Framework предоставляется встроенный обобщенный делегат EventHandler<TEventArgs>. Тип TEventArgs обозначает тип аргумента, передаваемого параметру EventArgs события.

Встроенный необобщенный делегат EventHandler

В среде .NET Framework предоставляется встроенный необобщенный делегат EventHandler. Он может быть использован для объявления обработчиков событий, которым не требуется дополнительная информация о событиях. В этом случае в качестве параметра EventArgs события используется статический параметр EventArgs.Empty.

Упражнение 8.13

Переделать упражнение 8.12 удалив свой делегат и добавив встроенный обобщенный или необобщенный делегат EventHandler.

Упражнение 8.14

Написать программу для обработки событий, связанных с нажатием клавиш на клавиатуре.