

Лекция 13 ДЕЛЕГАТЫ

13.1 Понятие делегата

Существует множество задач, в которых необходимо выполнять вычисления с помощью «однотипных» методов, например, с помощью математических функций вещественного типа, имеющих вещественный аргумент (это все тригонометрические функции, логарифмы, экспонента и т.д.). В таких задачах очень хочется получить в свое распоряжение метод, в котором формальным параметром является имя вычисляемой функции или ссылка на имя.

В языке C# методы сами по себе не «гуляют», а могут определяться только внутри некоторых классов.

Поэтому в язык C# был включен специальный класс, позволяющий хранить ссылки на «однотипные» методы. Этот класс получил название Делегат.

Делегат это специальный класс, предназначенный для хранения ссылок на методы.

По определению – переменная типа класс является объектом. Класс, позволяющий описать некоторое множество объектов, каждый из которых является функцией (или ссылкой на функцию), называется функциональным типом.

Таким образом, делегаты в языке C# предназначены для описания функциональных типов. Экземплярами такого класса являются ссылки на функции (методы) – им также как переменным выделяются места в памяти компьютера, начальный адрес которых являются «точками» входа в функции и передаются ссылками.

Эта особенность делегата определила две основные области его применения – самостоятельно для решения некоторых задач (подобно приведенной выше) или для поддержки событий (смотри следующую лекцию).

В этой лекции мы будем рассматривать самостоятельное использование делегатов для решения некоторых задач.

13.2 Описание делегата

Формат записи делегата фактически задает сигнатуру (описание) методов, которые могут быть вызваны с его помощью:

[спецификаторы] **delegate** <тип> <имя> (<параметры>);

где

спецификаторы определяют условия доступа к делегату;

delegate — зарезервированное слово;

<тип> — тип возвращаемого результата;

<имя> — имя делегата (уникальный идентификатор);

<параметры> — формальные параметры вызова.

Например, описание всех функций вещественного типа, имеющих вещественный аргумент, имеет следующий вид:

```
public delegate double Funk(double argm);
```

Любая функция, соответствующая этому описанию, может использоваться в качестве параметра вызова умалчиваемого конструктора класса-делегата, который и возвращает конкретный экземпляр делегата – ссылку на функцию. Отличительной особенностью работы делегата является формирование ссылок во время работы программы (динамически), а не на этапе ее компиляции.

13.3 Пример использования делегата

Для лучшего понимания механизма работы делегата рассмотрим решение следующей задачи: разработать приложение для вычисления пяти функций вещественного типа, имеющих вещественный аргумент: Sin(x), Log(x), Cos(x), Exp(x) и Round(x). Для выбора вычисляемой функции использовать делегат.

Исходный код программы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        public delegate double Funk(double x); //Объявление делегата
        public double rab(Funk f, double x) { return f(x); }
        //Объявление функции, использующей делегат
        private void button1_Click(object sender, EventArgs e)
        {
            string st;
            double x, y;
            textBox2.Text = "";
            x = Convert.ToDouble(textBox1.Text);
            st="Значение x=" +textBox1.Text+ "\r\n";
            textBox2.AppendText(st);
            y = rab(Math.Sin, x); // Применение делегата
            st = "Sin(x)=" + y.ToString() + "\r\n";
        }
    }
}
```

```

        textBox2.AppendText(st);
        y = rab(Math.Log, x); // Применение делегата
        st = "Log(x)=" + y.ToString() + "\r\n";
        textBox2.AppendText(st);
        y = rab(Math.Cos, x); // Применение делегата
        st = "Cos(x)=" + y.ToString() + "\r\n";
        textBox2.AppendText(st);
        y = rab(Math.Exp, x); // Применение делегата
        st = "Exp(x)=" + y.ToString() + "\r\n";
        textBox2.AppendText(st);
        y = rab(Math.Round, x); // Применение делегата
        st = "Round(x)=" + y.ToString() + "\r\n";
        textBox2.AppendText(st);
    }
}
}

```

Работа программы:

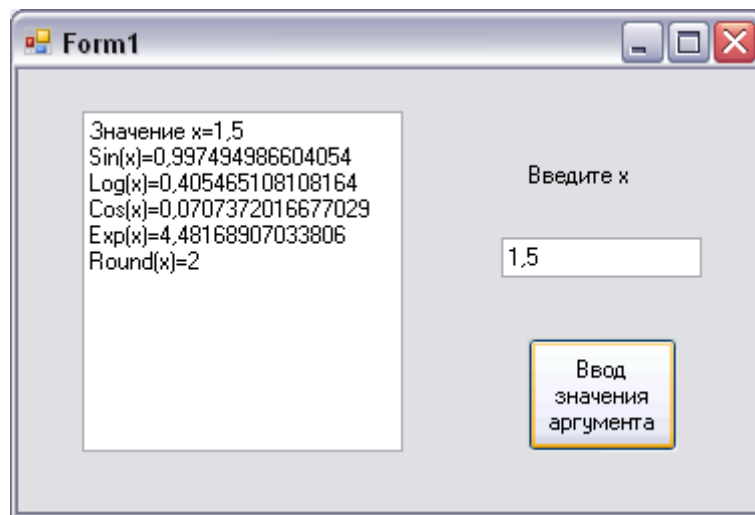


Рисунок 13.1 – Использование делегата

Естественно можно навести «красоту» в код программы, например,

```

private void button1_Click(object sender, EventArgs e)
{
    string st;
    Funk[] ff={Math.Sin, Math.Log, Math.Cos, Math.Exp, Math.Round};
    string[] sfu = { "Sin", "Log", "Cos", "Exp", "Round" };
    double x, y;
    textBox2.Text = "";
    x = Convert.ToDouble(textBox1.Text);
    st="Значение x=" +textBox1.Text+ "\r\n";
    textBox2.AppendText(st);
    for (int i = 0; i < 5; i++)
    {
        y = rab(ff[i], x);
        st = sfu[i]+"=" + y.ToString() + "\r\n";
        textBox2.AppendText(st);
    }
}

```

```
}
```

но идея объявления и использования делегата в программе полностью рассмотрена в коде программы этого учебного примера.

Еще раз отметим особенность делегатов – динамически (во время выполнения программы) обращаться к вызываемым методам. Эта особенность делегатов часто используется при создании базовых конструкций программ, в которые можно добавлять различные создаваемые фрагменты, например, фрагменты меню, включающие «однотипные» методы.

13.4 Совместимость делегатов

Рассматривая «совместимость» делегатов необходимо рассмотреть два вопроса – совместимость типов делегатов и совместимость экземпляров объектов делегатов.

Типы делегатов всегда несовместимы друг с другом, даже если их форматы записей совпадают, например,

```
public delegate void Funk1();
public delegate void Funk2();

. . .
Funk1 d1 = Metod1;
Funk2 d2 = d1;    //Ошибка о несовместимости типов
```

Если для объявления экземпляров делегатов использован один и тот же тип, то имеет смысл говорить о совместимости (равенстве) экземпляров делегатов. Экземпляры однотипных делегатов считаются равными, если они получают ссылку на один и тот же метод, например,

```
public delegate void Funk1();

. . .
Funk1 d1 = Metod1;
Funk1 d2 = Metod1;
```

В данном примере можно говорить о равенстве экземпляров делегатов `d1` и `d2`.

13.5 Методы базовые классы делегатов

Необходимо отметить, что в CTS платформы .NET есть абстрактные классы `System.Delegate` и `System.MulticastDelegate` из которых (как из базовых) создаваемые делегаты могут наследовать некоторые методы.

Перечень свойств и методов, наиболее часто наследуемых создаваемыми делегатами от `System.MulticastDelegate`, можно представить следующим списком:

`public MethodInfo Method {get;}` – это свойство возвращает имя метода, на который указывает делегат;

`public object Target {get:}` – если делегат указывает на метод – член класса, то `Target` возвращает имя класса метода; если метод статический, то `Target` возвращает `null`;

`public static Delegate Combine(Delegate[])` – добавляет новые методы к группе методов, обрабатываемых делегатом;

`public static Delegate Remove(Delegate source, Delegate value)` – удаляет метод `value` из списка методов делегата `source`;

`public sealed override Delegate[] GetInvocationList()` – возвращает список всех связанных с делегатом методов.

Для делегатов существует понятие многоадресный делегат – делегат, способный указывать на любое количество функций. Поскольку все делегаты языка C# являются производными классами от `System.MulticastDelegate`, то любой делегат C# потенциально является многоадресным.

Применим свойство многоадресности для делегата нашей программы вычисления вещественных функций. Изменения связаны только с обработчиком кнопки, поэтому приведен только фрагмент этого кода:

```
private void button1_Click(object sender, EventArgs e)
{
    string st;
    Funk[] ff={Math.Sin, Math.Log, Math.Cos, Math.Exp, Math.Round};
    string[] sfu = { "Sin", "Log", "Cos", "Exp", "Round" };
    double x, y;
    textBox2.Text = "";
    x = Convert.ToDouble(textBox1.Text);
    st="Значение x=" +textBox1.Text+ "\r\n";
    textBox2.AppendText(st);
    Funk df = Math.Sin;
    for (int i = 1; i < 5; i++)
        df += ff[i];    //df = df + ff[i];
    for (int i = 4; i >= 0; i--)
    {
        y = rab(df, x);
        df -=ff[i];    //df = df - ff[i];
        st = sfu[i]+"=" + y.ToString() + "\r\n";
        textBox2.AppendText(st);
    }
}
```

Работа программы приведена на рисунке 13.2.

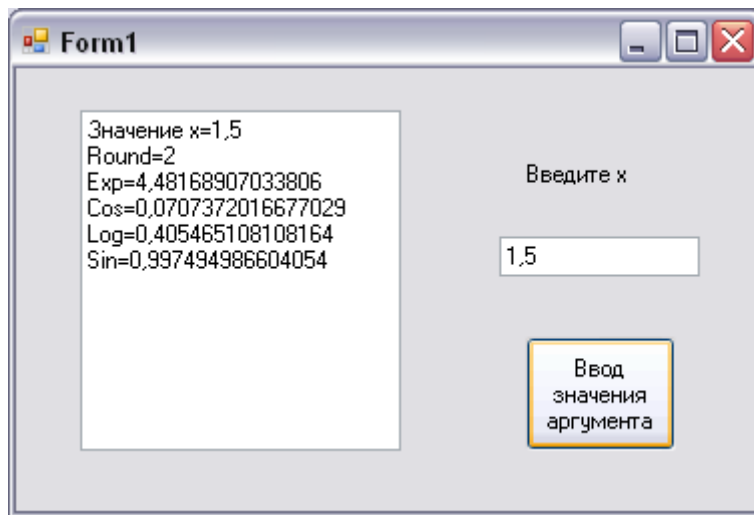


Рисунок 13.2 – Использование многоадресного делегата

Из рисунка 13.2 видно, что работа многоадресного делегата организована по принципу стека.

Заканчивая рассмотрения делегатов как специальный класс, предназначенный для хранения ссылок на методы, необходимо отметить, что любой объект представляет собой ссылку (в том числе и объект делегата) и может быть передан в некоторые методы в качестве параметра для дальнейшей работы.

Таким образом обеспечивается **функциональная параметризация**: в методы можно передавать не только данные, но и различные методы их обработки.