Лабораторная работа №5
Делегаты и события

Делегаты – это новый тип данных в C#. Они предназначены для передачи в качестве параметров одних методов другим методам. Другими словами, если требуется передать метод в качестве параметра, сведения об методе необходимо поместить (обернуть) в особый тип объекта – делегат.

По своей структуре делегат - это объект, который ссылается на метод, то есть делегат указывает на адрес области памяти, являющейся точкой входа в метод. С помощью делегата можно вызвать метод, на который он указывает.

В процессе выполнения программы делегату можно присвоить ссылку на другой метод. Это дает возможность определять во время выполнения программы, какой из методов должен быть вызван. Существенным моментом, является то, что в отличие от указателей, делегаты безопасны по типу.

Делегаты реализуются как экземпляры классов, производных от библиотечного класса System.Delegate. Для создания делегата необходимо выполнит два шага.

На первом шаге необходимо объявить делегат. При этом сигнатура делегата должна полностью соответствовать сигнатуре метода, который он представляет.

Например, делегат должен ссылаться на метод класса CA:

static **int min(int x,int y)**,

тогда объявление делегата может выглядеть, следующим образом:

delegate int LpFunc(int a,int b);

На втором шаге мы должны создать экземпляр делегата для хранения сведения о представляемом им методе:

LpFunc pfnk = new LpFunk(CA.min);

Экземпляр делегата может ссылаться на любой статический метод или метод объекта любого класса, при условии, что сигнатура метода полностью соответствует сигнатуре делегата.

Использование делегата (пример 1)

```
using System;
namespace ConsoleApplication14
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    ///
    class MathOprt
    {
        public static double Mul2(double val)
        {
            return val*2;
        }
```

```
        public static double Sqr(double val)
        {
            return val*val;
        }
    }
    delegate double DblOp(double x);//объявление делегата

    class Class1
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]

static void Main(string[] args)
        {
            //
            // TODO: Add code to start application here
            DblOp [] operation = // создание экземпляров делегата
            {
                    new DblOp(MathOprt.Mul2),
                    new DblOp(MathOprt.Sqr)
            };
            for(int j=0;j<operation.Length;j++)
            {
                Console.WriteLine("Резльтаты операции[{0}]:",j);
                Prc(operation[j], 4.0);
                Prc(operation[j], 9.94);
                Prc(operation[j], 3.143);
            }
                    //
        }
        static void Prc(DblOp act, double val)
        {
        double rslt = act(val);
        Console.WriteLine("Исходное значение {0}, результат  {1}",
                val,rslt);
        }
    }
}
```

Делегаты могут хранить несколько адресов областей памяти. То есть делегат может указывать на несколько различных методов. Это позволяет, последовательно инициализируя адреса вызывать метод за методом. Эта способность делегатов называется *многоадресностью делегатов*.

Для создания цепочки вызовов методов необходимо сначала создать экземпляр делегата для одного метода, а затем с помощью операции " + = " добавить остальные методы. В процессе выполнения кода можно не только добавлять новые методы, но и удалять не нужные с помощью операции " - = ".

Многоадресные делегаты должны иметь тип возвращаемого значения **void**. Это означает, что и методы, представляемые многоадресными делегатами также должны возвращать значение **void**.

Перепишем код из примера №1 с применением многоадресного делегата.

Пример№2:

```
using System;
namespace ConsoleApplication14
{
        /// <summary>
        /// Summary description for Class1.
        /// </summary>
        ///
        class MathOprt
        {
                public static void Mul2(double val)
                {
                        double rslt= val*2;
                Console.WriteLine("Mul2 bcходное значение {0},результат  {1}",
                                val,rslt);
                }
                public static void Sqr(double val)
                {
                        double rslt = val*val;
                Console.WriteLine("Sqr исходное значение {0}, результат  {1}",
                                val,rslt);
                }
        }
        delegate void DblOp(double x);//объявление делегата

        class Class1
        {
```

```
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            //
            // TODO: Add code to start application here
            DblOp  operations = new DblOp(MathOprt.Mul2);
                operations += new DblOp(MathOprt.Sqr);

                Prc(operations, 4.0);
                Prc(operations, 9.94);
                Prc(operations, 3.143);
        }
            //
        static void Prc(DblOp act, double val)
        {
            Console.WriteLine("\n*********\n");
            act(val);
        }
    }
}
```

События

Работа с событиями осуществляется в C# согласно модели «издатель-подписчик». Класс, ответственный за инициализацию (выработку) событий публикует событие, и любые классы могут подписаться на это событие. При возникновении события исполняющая среда уведомляет всех подписчиков о произошедшем событии, при этом вызываются соответствующие методы-обработчики событий подписчиков. Какой обработчик события будет вызван – определяется делегатом.

Платформа .NET требует для всех обработчиков событий следующей сигнатуры кода:

```
void OnRecChange(object source, ChangeEventArgs e)
{
    // Код для обработки события
}
```

Обработчики событий обязательно имеют тип возвращаемого значения void. Обработчики событий принимают два параметра. Первый параметр _ это ссылка на объект, сгенерировавший событие. Эта ссылка передается обработчику самим генератором событий. Второй параметр – это

ссылка на объект класса EventArgs или класса производного от него. В производном классе может содержаться дополнительная информация о событии.

Обработчик события определяется делегатом. Согласно сигнатуре обработчика события делегат должен принимать два параметра и выглядеть следующим образом:

```
public delegate void ChangeEventHandle(object source,ChangeEventArgs e);
```

Для того, чтобы иметь возможность подписаться на событие класс генератора событий должен содержать член типа указанного делегата с ключевым словом **event** и метод, который будет вызываться при возникновении события, например:

```
public event ChangeEventHandler OnChangeHandler;
```

Этот член является специализированной формой многообъектного делегата. Использую операцию “+=”, клиенты могут подписаться на это сообщение:

```
gnEvent.OnChangeHandler +=
new GenEvent.ChangeEventHandler          (OnRecChange);
```

где gnEvent имя класса генератора событий.

Работа с событиями (Пример №3):

```
using System;
namespace sobit
{
        /// <summary>
        /// Summary description for Class1.
        /// </summary>

        class ChangeEventArgs : EventArgs
        {
                string str;
                public string Str
                {
                        get
                        {
                                return str;
                        }
                }
                int change;
                public int Change
                {
                        get
```

```
            {
                return change;
            }
        }
        public ChangeEventArgs(string str,int change)
        {
            this.str = str;
            this.change = change;
        }
}
        class GenEvent          // Генератор событий - издатель
        {
            public delegate void ChangeEventHandler
                (object source,ChangeEventArgs e);

            public event ChangeEventHandler OnChangeHandler;

            public void UpdateEvent(string str,int change)
            {
                if(change==0)
                    return;
                ChangeEventArgs e =
                new ChangeEventArgs(str,change);

                if (OnChangeHandler != null)
                    OnChangeHandler(this,e);
            }

        }
//Подписчик
class RecEvent
{
//Обработчик события
        void OnRecChange(object source,ChangeEventArgs e)
        {
            int change = e.Change;
            Console.WriteLine("Вес груза '{0}' был {1} на {2} тонны",
                e.Str,change > 0 ? "увеличен" : "уменьшен",
                Math.Abs(e.Change));
        }
        GenEvent gnEvent;
```

```
        // в конструкторе класса осуществляется подписка
            public RecEvent(GenEvent gnEvent)
            {
            this.gnEvent = gnEvent;
            gnEvent.OnChangeHandler +=        //здесь осуществляется подписка
                        new GenEvent.ChangeEventHandler(OnRecChange);
            }
        }
        class Class1
        {
            /// <summary>
            /// The main entry point for the application.
            /// </summary>
            [STAThread]
            static void Main(string[] args)
            {
                //
                // TODO: Add code to start application here

                GenEvent gnEvent = new GenEvent();
                RecEvent inventoryWatch =  new RecEvent(gnEvent);
                gnEvent.UpdateEvent("грузовика", -2);
                gnEvent.UpdateEvent("автопоезда", 4);
                //
            }
        }
}
```