Лабораторная работа №5

Делегаты и события.

1) Делегаты

Делегат – это тип, который определяет сигнатуру метода. При создании экземпляра делегата можно связать этот экземпляр с любым методом с совместимой сигнатурой. Метод можно запустить (или вызвать) с помощью экземпляра делегата.

Пример объявления делегата:

```
public delegate double MathFunc(double arg);
```

В этом примере создается тип MathFunc, с помощью которого можно связать экземпляр класса MathFunc с методом, принимающим в качестве аргумента переменную типа double, и возвращающую значение типа double.

Делегату можно назначить любой метод из любого доступного класса или структуры, соответствующий сигнатуре делегата, которая состоит из типа возвращаемого значения и параметров. Этот метод должен быть статическим методом или методом экземпляра. Это позволяет программно изменять вызовы метода, а также включать новый код в существующие классы. Если сигнатура делегата известна, то можно назначить собственный метод.

Пусть имеется класс, в котором объявлены 2 статических метода:

```
class Functions
{
    public static double f1(double x)
    {
        return 1 - Math.Exp(-x);
    }
    public static double f2(double x)
    {
        return x;
    }
}
```

Тогда, при создании экземпляра типа MathFunc его можно связать с любым из методов класса Functions, так как они имеют подходящую сигнатуру:

```
MathFunc func = new MathFunc(Functions.f1);
Console.WriteLine(func(1000));
```

В этом примере происходит связывание экземпляра func типа MathFunc с методом f1 класса Functions. Во второй строчке программы производится вызов метода f1 с параметром 1000.

Полный текст консольной программы:

```
namespace Delegates
{
    class Program
    {
        static void Main(string[] args)
        {
            MathFunc func = new MathFunc(Functions.f1);
            Console.WriteLine(func(1000));
            Console.ReadKey();
        }
    }

    public delegate double MathFunc(double arg);

    class Functions
    {
        public static double f1(double x)
        {
            return 1 - Math.Exp(-x);
        }
        public static double f2(double x)
        {
            return x;
        }
    }
}
```

Делегаты являются удобным средством для передачи различных методов в качестве аргументов для других методов. Например, для приведенной выше программы можно создать класс, выполняющий интегрирование функций с сигнатурой делегата MathFunc:

```
class Integrator
{
    public double integrate(MathFunc f, double min, double max, double N)
    {
        double result = 0.0;
        double eps = (max - min) / N;
        double x = min;

        while (x <= max)
        {
            result += f(x); x += eps;
        }

        result *= eps;

        return result;
    }
}
```

Класс содержит единственный метод integrate, который производит

интегрирование функции типа MathFunc в пределах (a, b), методом прямоугольников. Пример использования:

```
class Program
{
    static void Main(string[] args)
    {
        Integrator i = new Integrator();
        double result = i.integrate(Functions.f1, 0, 1, 1000);
        Console.WriteLine(result.ToString());
        Console.ReadKey();
    }
}
```

Благодаря возможности ссылаться на метод как на параметр делегаты оптимально подходят для задания функций обратного вызова. Например, ссылка на метод, сравнивающий два значения, которые можно передать в качестве аргумента алгоритму сортировки. Поскольку код сравнения находится в отдельной процедуре, алгоритм сортировки может быть написан более общим образом.

2) Анонимные методы

В приведенном выше примере для создания списка подынтегральных функций (f1, f2) использовался класс Functions. Однако язык C# версий 3.0 и выше поддерживает использование анонимных методов, которые не имеют имени. Создание анонимных методов является, по существу, способом передачи блока кода в качестве параметра делегата. Пример создания анонимного метода и использования его в качестве подынтегральной функции:

```
class Program
{
    static void Main(string[] args)
    {
        Integrator i = new Integrator();
        Console.WriteLine(i.integrate(delegate(double x)
        {
            return x;
        },
        0, 1, 1000));
        Console.ReadKey();
    }
}
```

Анонимный метод, представляющий собой функцию вида y = x, в этом

примере выделен серым цветом. Использование анонимных методов позволяет сократить издержки на кодирование при создании делегатов, поскольку не требуется создавать отдельный метод.

3) События

Делегаты являются удобным средством для реализации событий, которые используются для индикации состояния объекта. Например, класс Button (кнопка) имеет событие Click, которое вызывается при нажатии на кнопку, а класс Form (форма) имеет событие Load, вызываемое при загрузке формы. Рассмотрим использование событий на примере класса Counter (счетчик).

```
namespace Delegates
{
    public delegate void CounterEvent();

    class Counter
    {
        private int m_Counter;
        private int m_CountTo;

        public Counter(int countTo)
        {
            m_Counter = 0; m_CountTo = countTo;
        }

        public event CounterEvent onCount;

        public void inc()
        {
            if (++m_Counter == m_CountTo && onCount != null)
            {
                onCount();
                m_Counter = 0;
            }
        }
    }
}
```

Класс «Счетчик» содержит конструктор, принимающий в качестве аргумента число, до которого ведется счет. Метод inc, используется для увеличения значения счетчика на единицу. Свойство onCount типа CounterEvent (делегат) хранит ссылку на метод, который необходимо вызвать, когда значение счетчика будет равно указанному порогу срабатывания. Пример ис пользования:

```
class Program
{
    static void Main(string[] args)
    {
        Counter c = new Counter(6);
        c.onCount += delegate()
        {
            Console.WriteLine("On count!");
        };
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine(i); c.inc();
        }
        Console.ReadKey();
    }
}
```

Здесь сначала создается экземпляр класса Counter с параметром 6 (порог срабатывания). Затем, с помощью анонимного метода, производится подписка на событие onCount. Далее в цикле производится вызов метода inc. Выходные данные программы приведены на рисунке 1.

Рисунок 1 – Пример работы программы Счетчик

Из рисунка 1 видно, что на шестом вызове метода inc произошел вызов события onCount, которое привело к выводу на экран выводу сообщения «On count!». Отметим, что подписка на событие производилась следующим образом:

```
c.onCount += delegate()
{
    Console.WriteLine("On count!");
};
```

То есть в качестве обработчика события мы не назначили анонимный метод, а «добавили» его (оператор +=). Такая конструкция позволяет подписывать на одно и то же событие несколько методов, либо динамически отписываться от

события с помощью оператора (-=). Пример:

```
class Program
{
    private static Counter m_Counter;

    static void Main(string[] args)
    {
        m_Counter = new Counter(3);
        m_Counter.onCount += EventHandler1;
        m_Counter.onCount += EventHandler2;

        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine(i);
            m_Counter.inc();
        }
        Console.ReadKey();
    }

    private static void EventHandler1()
    {
        Console.WriteLine("Event handler 1!");
        m_Counter.onCount -= EventHandler1;
    }

    private static void EventHandler2()
    {
        Console.WriteLine("Event handler 2!");
    }
}
```

Здесь сначала создается экземпляр класса Counter с параметром 3 (порог срабатывания). Затем производится подписка методов EventHandler1 и EventHandler2 на событие onCount. Эти методы различаются выводимым сообщением, а также метод EventHandler1 отписывается от события onCount объекта m_Counter. Пример работы программы приведен на рисунке 2.

Рисунок 2 – Динамический отказ от подписки на событие

4) Взаимодействие с визуальными компонентами

Взаимодействие с компонентами в языке C# осуществляется через механизм событий. Установим на форме две кнопки (см. рисунок 3). Выделим первую кнопку и перейдем на вкладку Свойства (Properties), где выбираем список событий и, дважды щелкнув левой кнопкой мыши по Click, подпишемся на событие по нажатию на кнопку. В класс MainForm добавится обработчик события:

```
public partial class MainForm : Form
{
    public MainForm()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {

    }
}
```

При этом в функцию InitializeComponent(), находящуюся в файле MainForm.Designer.cs, автоматически добавится код:

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

Добавим в обработчик следующий код:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show(string.Format("Нажата кнопка " + (sender as Button).Text));
}
```

Теперь при нажатии на кнопку button1 появляется сообщение: «Нажата кнопка button1». Далее назначим в качестве обработчика события Click для кнопки button2 функцию button1_Click. Это приведет к тому, что при нажатии второй кнопки будет появляться общение «Нажата кнопка button2». Такое поведение объясняется тем, что события, как и делегаты, могут содержать аргументы, и одним из аргументов является ссылка на объект (sender), инициирующий это событие.

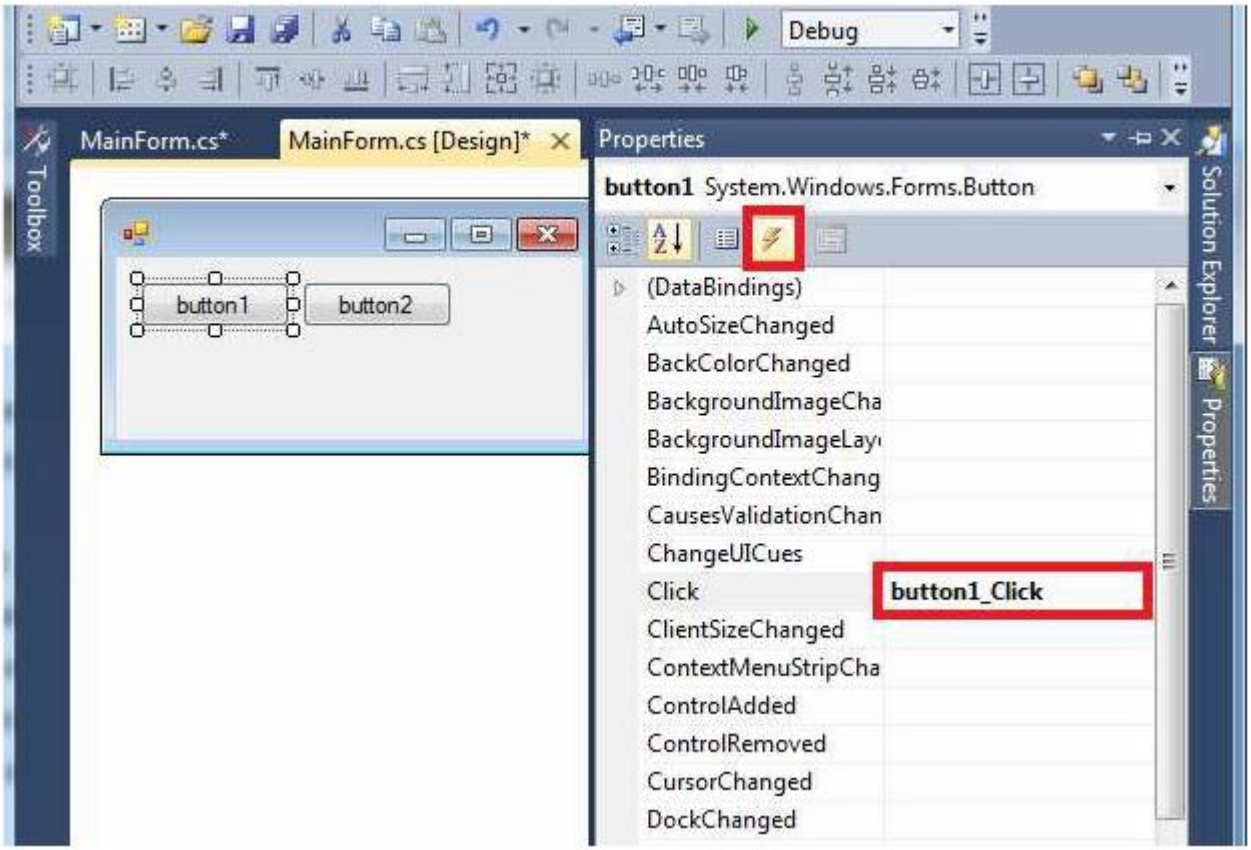

Рисунок 3 – Установка события при нажатии на кнопку

Многие визуальные компоненты являются наследниками класса Control, поэтому они имеют общие события, перечислим основные из них.

Событие	**Описание**
Click	Возникает при щелчке левой кнопки мыши по элементу управления.
DoubleClic	Возникает при двойном щелчке.
Enter	Возникает при помещении курсора на элемент управления.
KeyDown	Возникает при нажатии кнопки на клавиатуре.
KeyPress	Возникает после того, как клавиша на клавиатуре была нажата, затем отпущена.
KeyUp	Возникает при отпускании клавиши на клавиатуре.
Leave	Возникает при перемещении курсора с элемента управления.
MouseDow	Возникает при нажатии любой клавиши мыши.
MouseUp	Возникает при отпускании любой клавиши мыши.

Paint	Возникает при перерисовки элемента управления.
Resize	Возникает при изменении размера элемента.

Из таблицы видно, что многие события взаимосвязаны и возникают в различные моменты времени. Отметим также, что некоторые визуальные компоненты имеют значительно больше событий, чем определено у класса Control.

5) Задания к лабораторной работе №5

1) Написать класс, унаследованный от класса Button, который генерирует событие по тройному щелчку мышки. Учесть, что тройным щелчком будут считаться три последовательных нажатий с малым интервалом между ними. Предусмотреть возможность указания максимального интервала между нажатиями.

2) Написать класс, унаследованный от класса Panel, который распознает и генерирует событие при перемещении курсора мышки по поверхности компонента по Z-траектории.

3) Написать класс, унаследованный от любого визуального компонента, который движется по форме и отскакивает от границ формы. При щелчке по компоненту он изменяет направление своего движения произвольным образом.

4) Создать компонент, состоящий из панели, на которой располагается кнопка. Причем высота кнопки равна высоте панели, а длина панели в несколько раз больше длины кнопки. В начальный момент кнопка располагается в левой части панели. С помощью мыши пользователь может перемещать кнопку в пределах панели только по горизонтали и кнопка не должна выходить за границы панели. У кнопки есть инерция, то есть если пользователь резко передвинет кнопку и отпустит, она некоторое время будет двигаться вперед, но затем должна возвращаться в начальное положение.

Компонент должен также генерировать событие при достижении правого края кнопки правого края панели.

5) Создать компонент, который распознает и генерирует событие при движении курсора мышки по круговой траектории.

6) Написать программу для тренировки пилотов. Суть ее заключается в том, что на форме имеется несколько кнопок разного размера. Перемещением одной кнопки может управлять пользователь, остальные же двигаются по форме с постоянной скоростью и могут отталкиваться от границ формы. В начальный момент скорости и направление движение кнопок выбирается случайным образом. Пользователь должен перемещать свою кнопку так, чтобы не столкнуться с остальными кнопками. При столкновении с любой из движущихся кнопок с кнопкой пользователя возникает событие, и игра прекращается. Предусмотреть подсчет времени, которое пользователь смог уклоняться от кнопок.

7) Написать игру пятнашки. На форме имеется 15 кнопок с порядковыми номерами и одна пустая ячейка. При щелчке по кнопке она перемещается на пустую ячейку. Когда кнопки выстраиваются в правильном порядке должно генерироваться событие.

8) Написать класс, унаследованный от класса Panel, который распознает правильное введение заданного графического ключа (сама комбинация должна быть изменяемая) и генерирует при этом событие. При необходимости на панели можно отображать графическую информацию для удобства использования (аналогично разблокировке смартфона с помощью графического ключа).

9*) Реализовать класс, генерирующий событие, при нажатии определённой горячей клавиши (комбинации клавиш, например, Ctrl+Shift+F11) даже в том случае, когда форма свёрнута (или не активна, не в фокусе). Для определения нажатых клавиш использовать функцию WinAPI GetAsyncKeyState(int vkey) из библиотеки user32.dll. Внутри класса будет иметься метод Refresh(), внутри которого и будет происходить определение

нажатых клавиш. Для нормальной работы класса, метод Refresh() нужно запускать через каждые 10 миллисекунд (например, в таймере, размещенном на форме). Событие через аргумент *e* должно передавать информацию о том, какая комбинация клавиш нажата (событие с передачей дополнительных параметров). Класс реагирует не на все комбинации клавиш, а только лишь на заданные.

6) Контрольные вопросы

1) Что такое делегат? Опишите синтаксис объявления делегата.

2) Что такое событие?

3) Чем отличается анонимный метод от обычного? Каковы особенности их применения?

4) Опишите основные события визуальных элементов управления.

5) Как создать событие с передачей дополнительных параметров и без них?

6) Как сгенерировать событие?

7) Как подписаться на событие?

8) Возможно ли одному событию назначить нескольких подписчиков?

9) Возможно ли одного подписчика назначить для обработки нескольких событий?

10) Как при генерации события определить, назначен ли ему подписчик?