Модуль 3, практическое занятие 5

События

Стандартный шаблон событий

Задача 1. Последовательность событий при работе с формой

- **Создайте пустую форму.**
- **Добавьте в нее обработчики событий: Activated, Deactivate, FormClosed, FormClosing, Load, Paint, Resize.**
- **В каждый обработчик включите оператор, изменяющий текст заголовка формы, и оператор, добавляющий в общую строку название события.**
- **В обработчике события Form1_FormClosed() поместите вызов диалогового окна, где выведите список событий, произошедших при выполнении программы.**
- **В каждый обработчик событий, кроме Activated, Deactivate, поместите вывод в диалоговое окно названия события.**
- **Запустите программу на выполнение.**
- **Запишите названия произошедших событий, изменяемые заголовки формы.**
- **Сравните с результатом, выведенным в обработчике события FormClosed.**

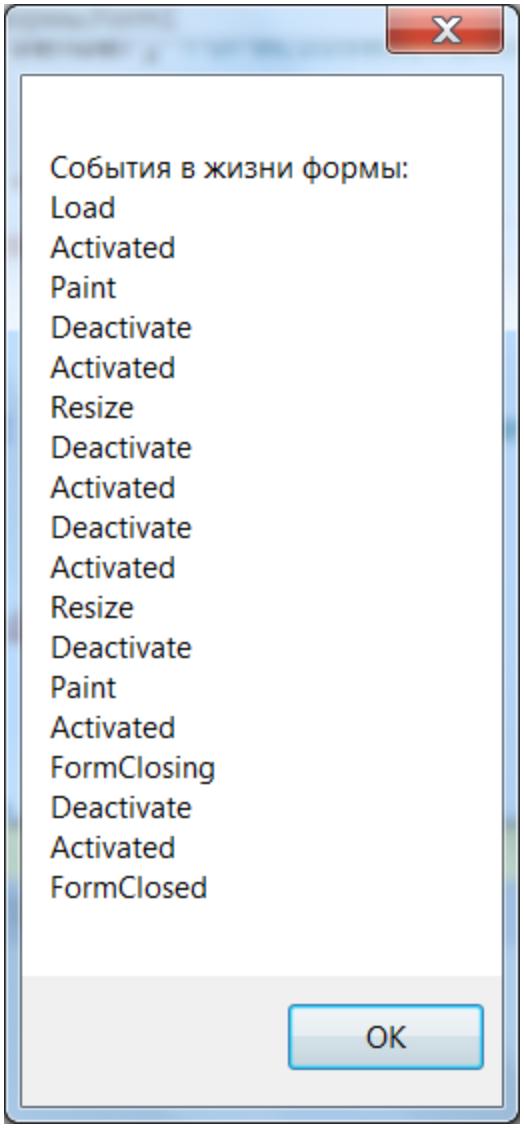
Задача 1

```
namespace WinProgram_2_2 {
    public partial class Form1 : Form {
        string result;
     public Form1() {
            InitializeComponent( );
     } // end of Form1()
     private void Form1_Load(object sender, EventArgs e) {
            this.Text = "Form1_Load";
            result += "\nLoad";
            MessageBox.Show("Событие Load");
     } // end of Form1_Load()
     private void Form1_Activated(object sender, EventArgs e) {
            this.Text = "Form1_Activated";
            result += "\nActivated";
     } // end of Form1_Activated()
     private void Form1_Deactivate(object sender, EventArgs e) {
            this.Text = "Form1_Deactivate";
            result += "\nDeactivate";
     } // end of Form1_Deactivate()
```

Задача 1

```
private void Form1_Resize(object sender, EventArgs e) {
            this.Text = "Form1_Resize";
            result += "\nResize";
            MessageBox.Show("Событие Resize");
        } // end of Form1_Resize()
private void Form1_Paint(object sender, PaintEventArgs e) {
            this.Text = "Form1_Paint";
            result += "\nPaint";
            MessageBox.Show("Событие Paint");
        } // end of Form1_Paint()
private void Form1_FormClosing(object sender, FormClosingEventArgs e) {
            this.Text = "Form1_FormClosing";
            result += "\nFormClosing";
            MessageBox.Show("Событие FormClosing");
        } // end of Form1_FormClosing()
private void Form1_FormClosed(object sender, FormClosedEventArgs e) {
           this.Text = "Form1_FormClosed";
            result = "События в жизни формы: " + result;
            MessageBox.Show(result+"\nFormClosed");
        } // end of Form1_FormClosed()
    } // end of class Form1
} // end of namespace
```

Задача 1. Пример результата выполнения



Задача 2

Создать класс **Generator** с событием **myEvent**, объявленным с делегатом **Action<int>**. Нестатический метод **Generate** класса **Generator** формирует список из заданного параметром метода количества элементов целого типа, случайно выбираемых из диапазона [0, 100). Если в процессе формирования списка очередной элемент кратен 2 или 3 или 5, то возникает событие myEvent.

Создать три класса Counter2, Counter3, Counter5, объекты которых представляют собой "счетчики", соответственно, элементов, кратных 2, 3, 5.

В основной программе создать объекты всех четырех классов. К событию myEvent объекта класса Generator подключить методы из классов Counter2, Counter3, Counter5, которые вычисляют количества элементов, кратных, соответственно, 2, 3, 5. С помощью метода Generate класса Generator создать список из N элементов. Вывести на экран сформированный список и значения счетчиков из объектов классов Counter2, Counter3, Counter5.

Делегат Action<T> (https://msdn.microsoft.com/ru-ru/library/018hxwa8(v=vs.110).aspx)
How to work with Action, Func, and Predicate delegates in C# (http://www.infoworld.com/article/3057152/application-development/how-to-work-with-action-func-and-predicate-delegates-in-c.html)
Delegates: Predicate Action Func (http://stackoverflow.com/questions/566860/delegates-predicate-action-func)

Задача 2. Генератор списка и событий

```
class Generator    {
 public event Action<int> myEvent = delegate { }; // пустой делегат
    static Random rnd = new Random();
    public List<int> Generate(int N) {
        List<int> list = new List<int>();
        for(int k=0; k< N; k++) {
            int next = rnd.Next(0,100);
            list.Add(next);
            if(next%2 == 0|| next % 3 == 0 || next % 5 == 0)
               myEvent(next);
        }
        return list;
    }
}   // class Generator
```

Задача 2

```
class Counter2 {
    // TODO: Разработать самостоятельно
    }
class Counter3 {
 // TODO: Разработать самостоятельно
    }
class Counter5 {
 // TODO: Разработать самостоятельно
    }
```

Задача 2. Пример результатов выполнения программы

50 14 45 57 11 59 85 39 84 43 77 30 15 78 51
count2.Numb2 = 5
count3.Numb3 = 8
count5.Numb5 = 5

Задача 3

- Класс Expression - представляет математическое выражение. **Поле** ex - ссылка на метод-выражение, exEvent – **событие**, происходящее при смене выражения, ExVal – **метод** вычисления значения выражения для заданного значения аргумента, **конструктор**.
- Класс ValueStore - хранит значение выражения. **Поле** exp – ссылка на выражение, x0 - значение аргумента, expCurrValue – значение выражения, CurrVal – ссылка на expCurrValue, **Конструктор**: ValueStore(Expression e, double x)
- Классы находятся Expression и ValueStore в отношении агрегации.

- В основной программе создать объект me класса Expression, использовать ссылку me в конструкторе объекта vs класса ValueStore. Задавая разные выражения поля me.ex, выводить значения vs. expCurrValue

Задача 3

```
public delegate double ExpDel(double x);
// TODO1: Определить событийный делегат ExpChanged
    public class Expression {
// TODO2: Объявить событие OnExpChanged типа ExpChanged
        ExpDel ex;  // Поле для ссылки на метод-выражение
        public Expression(ExpDel e) { // Конструктор
            ex = e;
        }
        public double ExVal(double x) {
            return ex(x);
        }
// TODO3: При обновлении выражения в аксессоре
// инициировать событие:
        public ExpDel Ex { set { ex = value; } }
    }
```

Задача 3

```
public class ValueStore {
        Expression exp;
        double x0; // точка - абсцисса
        double expCurrValue; // хранимое значение в x0
        public ValueStore(Expression e1, double x0 ) {
            exp = e1;
            this.x0 = x0;
            expCurrValue = exp.ExVal(x0);
        }
        public double CurrVal { get { return expCurrValue; } }
// TODO4: Определить метод OnExpChangedHandler(),
// изменяющий значение поля expCurrValue
// на значение выражения exp в точке x0
    }
```

Задача 3

```
class Program {
static void Main( ) {
Expression me = new Expression(x => { return x * x+2*x-3; });
    ValueStore vs = new ValueStore(me, 0);
// TODO5: Подписать объект vs на события объекта me
    Console.WriteLine(vs.CurrVal);
// изменяем выражение:
    me.Ex = x => { return Math.Sqrt(Math.Abs(x)); };
    Console.WriteLine(vs.CurrVal);
    me.Ex = x => { return Math.Sin(x); };
    Console.WriteLine(vs.CurrVal);
    me.Ex = x => { return x*x*x-1; };
    Console.WriteLine(vs.CurrVal);
       }
    }
}
```

Задача 3

Объясните результаты выполнения программы:

-3
-3
-3
-3

Задача 3

Согласно пунктам TODO1 - TODO5 добавить в код событие, происходящее при изменении выражения в объекте Expression.

Подписать объект ValueStore на событие смены его выражения и пересчитывать хранимое в объекте класса Expression значение при каждом изменении выражения.

Результат выполнения программы будет таким:

-3

0

0

-1

Задача 4

- Написать программу, моделирующую поведение цепочки из **N** бусин, нанизанных на нить длины **`len`**. Радиус бусин одинаков и равен целому числу $\left\lfloor \frac{len}{2*N} \right\rfloor$.
- Бусины цепочки должны реагировать на событие «изменение длины нити» и настраивать свой размер (количество бусин не изменяется).
- Бусины цепочки должны реагировать на событие «изменение количества бусин» и также настраивать свой размер (длина нити не изменяется).

Задача 4. Описание типов

1. Тип-делегат **public delegate void ChainLenChanged(double r);**
2. Класс **Bead** - бусина
    1. Поле **r** – вещественное число, радиус бусины
    2. Конструктор с вещественным параметром – радиус бусины. Если радиус – меньше или равен нулю, конструктор создаёт исключение **ArgumentOutOfRangeException**
3. Класс **Chain** – цепочка бусин
    1. Поле **len** – вещественное число - длина нити, на которую нанизаны бусины
    2. Поле **beads** – список **List<Bead>**, составленный из бусин, нанизанных на нить
    3. Событие **ChainLenChangedEvent**, определённое типом-делегатом **ChainLenChanged**
    4. Свойство **Len**. Обеспечивает доступ к полю – длина нити. При изменении длины нити активируется событие **ChainLenChangedEvent**
    5. Конструктор с двумя параметрами – вещественной длиной нити **len** и целым числом **N** бусин в цепочке. Создание бусин выполняет вспомогательный метод **CreateBeads()**.
    6. Метод **CreateBeads()** – создаёт объекты-бусины и добавляет их методы-обработчики в список обработчиков события **ChainLenChangedEvent**

Задача 4. События

1. Добавить в код событие, возникающее при изменении **N** - количества бусин на нити, предполагается, что длина нити не изменяется, а размеры бусин «подстраиваются» под длину нити так, чтобы занять её.
    1. В обработчике этого события добавить код (в классе **Bead**) пересчёта и изменения радиуса бусин
2. Добавить в код событие, возникающее при изменении радиуса бусин
3. Подписать объект **Chain** на события 2
4. В обработчике пересчитывать количество бусин, которые могут поместиться на нити заданной длины, удалять/добавлять бусины

Задача 4. Отладка

Тестирование кода выполните в консольном приложении. Получить от пользователя количество бусин и длину нити. Создать объект **Chain**. Вывести информацию о цепочке бусин: количество бусин, длина нити (с точностью до двух знаков после запятой), радиус бусины. Предложить пользователю экранное меню: 1) изменить длину нити; 2) изменить количество бусин на нити. После выбора пункта выводить информацию об обновлённой цепочке бусин.

(*) *Создайте оконное приложение, в поле pictureBox визуализируйте цепочку бусин. Добавьте возможность изменения параметров цепочки и бусин. Свяжите изменения в интерфейсе с изменениями бусин и цепочки.*

Задача 5

Знакомство со Standard event pattern

1. Объявить класс-наследник **`System.EventArgs`**, для представления параметров события.
2. Выбрать делегат для события **`System.EventHandler`** или **`System.EventHandler<>`**
3. Определить событие с типом выбранного делегата
4. Написать защищённый виртуальный метод, запускающий событие

Задание

В условиях задачи 3 события **изменения длины нити** цепочки бусин и **изменения количества бусин** описать при помощи шаблона стандартных событий.

Класс EventArgs (https://msdn.microsoft.com/ru-ru/library/system.eventargs(v=vs.110).aspx)
Делегат EventHandler (https://msdn.microsoft.com/ru-ru/library/system.eventhandler(v=vs.110).aspx)
Практическое руководство. Публикация событий, соответствующих рекомендациям .NET Framework (Руководство по программированию в C#)
(https://msdn.microsoft.com/ru-ru/library/w369ty8x.aspx)

Задача 5

```
// 1 шаг. Определяем класс-наследник EventArgs
// с аргументами для своего события
// у нас это событие изменения длины нити и аргумент - радиус
    public class ChainLenChangedEventArgs : EventArgs {
        public readonly double rad;
        public ChainLenChangedEventArgs(double r) {
            rad = r;
        }
    }
```

```
public class Chain {
// 2 шаг  описываем событие.
// Используем предусмотренный обобщённый делегат
    public event EventHandler<ChainLenChangedEventArgs>
                                                    ChainLenChanged;
//...
}
```

Задача 5

```
// 3 шаг. В класс Chain добавляем метод «запуска» события
protected virtual void OnChainLenChanged(ChainLenChangedEventArgs e)
{
     if (ChainLenChanged != null) ChainLenChanged(this, e);
}
```

```
// 4 шаг. Изменяем код «запуска» события, там где он производился
OnChainLenChanged(new ChainLenChangedEventArgs (newR));
```

```
// 5 шаг. Изменяем код обработчика.
// Добавляем параметры "источник" события и "параметры"
public void OnChainLenChangedHandler(object sender,
                                     ChainLenChangedEventArgs e) {
            R = e.rad;
}
```