

Лабораторная работа № 2

Создание приложения *Windows Forms*

Создадим приложение *Windows Forms*. Это можно сделать так же, как и для консольного приложения, через пункт меню *Файл – Создать ... Проект...* или на Начальной странице выбрать *Создать проект...* При этом откроется окно *Создать проект* (рис. 1)

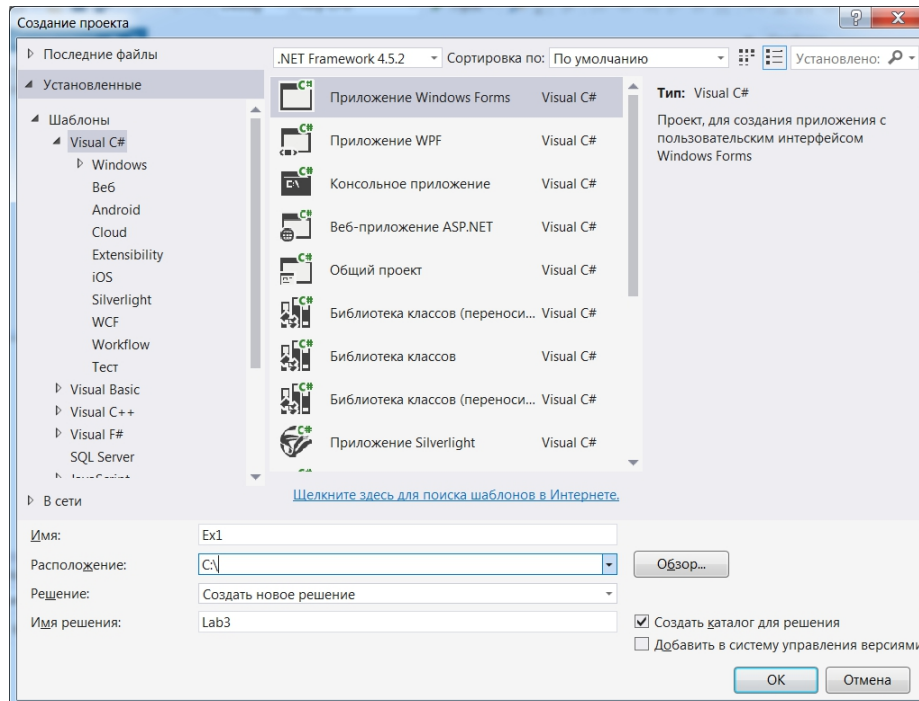


Рис. 1. Создание нового приложения *Windows Forms*

Для приложения *Windows Forms* на *C#* указываем в этом окне следующие параметры:

- Установленные шаблоны – *Visual C#, Windows*;
- Шаблон – Приложение *Windows Forms*;
- Имя – указываем имя проекта; наш проект назовем *Ex1*;
- Расположение папки приложения, его удобнее изменять, воспользовавшись кнопкой *Обзор...* Укажите путь к своей папке;
- Имя решения – оно автоматически становится равным имени проекта, изменим его на *Lab1*;
- Создать каталог для решения – для того, чтобы все файлы решения хранились в одной папке, что достаточно удобно, в этом пункте поставим галочку.

Заполнив все необходимые поля, нажимаем на кнопку *OK*.

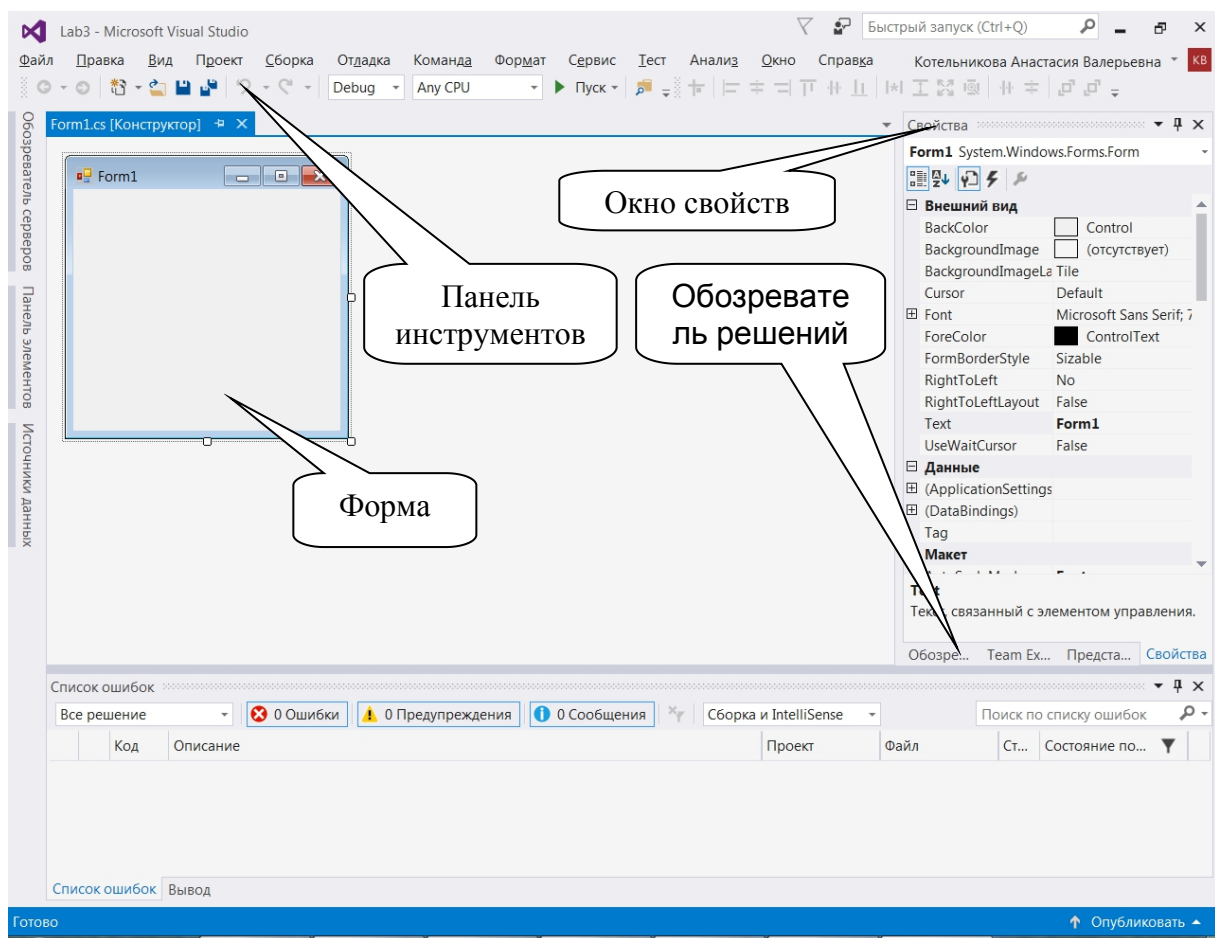


Рис. 2. Окно создаваемого проекта

Опишем некоторые основные панели, которые не были изучены ранее. Напомним, что для отображения большинства панелей можно воспользоваться пунктом меню *Вид*. Кроме того, заметим, что расположение панелей (в частности, **Обозревателя решений** и окна **Свойств**) может быть изменено пользователем на его усмотрение.

Панель элементов. *Visual Studio* 2017 предоставляет доступ ко множеству элементов управления при создании *Web*- и *Windows*- форм. Элементы управления с **Панели элементов** можно перетащить на визуальный конструктор. Более детально эта панель будет рассмотрена в последующих лабораторных работах.

Визуальный конструктор — это холст, на котором вы при помощи мыши создаете такие элементы, как формы (посредством перетаскивания, изменения размеров и т. д.). Они позволяют вам создавать элементы, которые составляют ваше приложение. Эти элементы включают: формы *Windows*, *Web*- формы, диаграммы классов, схемы *XML* и т. д. Все визуальные конструкторы работают практически одинаково. Во- первых, они занимают центральное место внутри интегрированной среды и имеют вид окон с вкладками, окруженных различными меню, панелями инструментов и другими панелями. Во- вторых, **Панель элементов** используется как палитра элементов, которые можно помещать на поверхность конструктора. Затем настраивается множество свойств каждого элемента (при помощи окна свойств).

Окно Свойств отображает свойства (или события) выбранного компонента. Оно позволяет управлять размером, внешним видом и поведением элементов управления.

На рис. 3 показано окно свойств формы *Form1*.

Комбинированная панель выбора объекта — это поле (выпадающий список), расположенное вверху окна свойств, указывающее, с каким объектом идет работа.

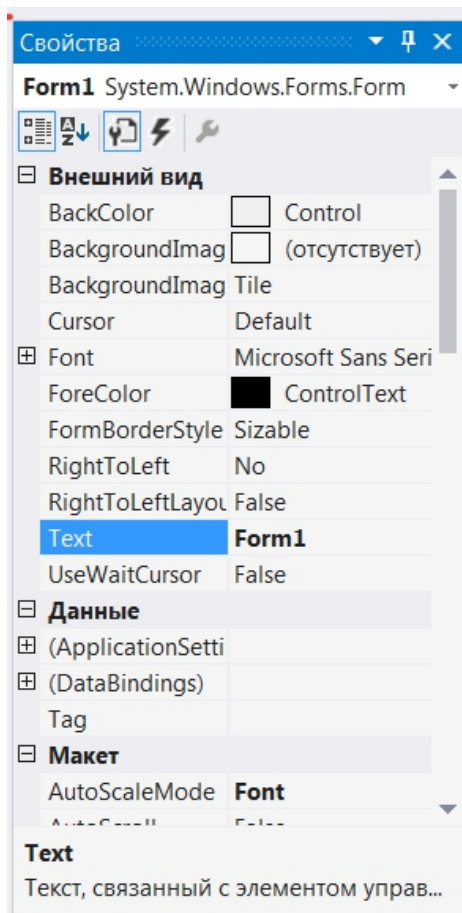






Рис. 3. Окно Свойств


По умолчанию отображаются свойства объекта, а не его события, кроме того, на свойства можно переключиться, нажав . Для каждого компонента определен список свойств, изменение которых приводит к изменению внешнего вида объекта или к изменению реакции на внешние воздействия. Левая колонка страницы свойств содержит имена свойств, а правая – их значения.



События объекта отображаются, если щелкнуть по . Каждый компонент способен реагировать на события, список которых отображен на странице событий. Левая колонка страницы событий содержит названия, а правая – имя процедуры-обработчика события, создать обработчик можно, дважды щелкнув по правой колонке напротив соответствующего события. Таким образом, окно свойств позволяет нам привязать события элемента управления к коду внутри приложения.

По умолчанию для облегчения доступа схожие свойства или события группируются в наборы при помощи категорий, например *Внешний вид* (если выбрать на панели инструментов окна **Свойств** значок ). Можно отключить эту возможность и получать список свойств/событий в алфавитном порядке (путем

нажатия значка  на панели инструментов).

Запуск приложения

Запуск выполняется так же, как и для консольного приложения: можно выбрать команду *Начать отладку* из меню *Отладка*, или щелкнуть по кнопке  **Пуск** (*Начать отладку*) панели быстрого доступа, или нажать *F5*.

Для закрытия приложения нажмите на крестик в правом верхнем углу запущенной формы , или нажмите *<Alt+F4>* на клавиатуре, или кнопку .

Изменение свойств

Свойства влияют на внешний вид и поведение объекта. Изменение свойств в ходе проектирования – это простой процесс, который включает выбор объекта, открытие страницы его свойств и изменение значений нужных свойств.

Рассмотрим некоторые свойства формы.

Свойство *Name*. Имя элемента управления. Для каждого объекта имеется свойство *Name*. Когда элемент управления помещается на форму, ему автоматически присваивается уникальное имя. Наша форма имеет имя *Form1*. Это имя может использоваться в коде для ссылки на элемент управления в ходе разработки программы, поэтому осмысленное присвоение имен компонентам упрощает создание приложений.

Назовем форму *MainForm* (основная форма). В окне **Свойств** выберите свойство *Name* и в правом столбце наберите *MainForm*. Это изменение сразу же отображается в комбинированной панели выбора объекта.

Свойство *BackColor* определяет цвет фона элемента управления. В *Visual Studio* предусмотрены предопределенные цветовые константы, которые соответствуют многим

общеупотребительным цветам. Все цвета разделены на 3 большие группы: *Другой*, *Интернет* и *Система*.

Установите цвет формы равным *Red*.

Свойство ***Text*** содержит текст, связанный с данным элементом управления. Для формы это свойство определяет заголовок формы.

Измените значение этого свойства формы на значение – «Моя форма».

Свойство ***Size*** состоит из двух – ***Height*** и ***Width***, которые определяют соответственно высоту и ширину формы (вместе с рамкой и заголовком формы). Значения этих свойств можно задать, уменьшая (или увеличивая) размеры формы, используя манипулятор мышь или вводя необходимые значения в правый столбец в окне свойств.

Свойства *X* и *Y*, входящие в свойство ***Location***, определяют расположение формы на экране, задавая расстояние от левой границы экрана до левого края формы и от верхней границы экрана до верхнего края формы соответственно (если при этом свойство ***StartPosition*** установлено в *Manual*).

Кроме свойств, изменение значений которых приводит к видоизменению внешнего вида приложения во время проектирования, существуют свойства, изменение значения которых видимо только после запуска приложения.

Свойство ***Cursor*** определяет графический вид курсора.

В окне **СВОЙСТВ** измените свойство *Cursor*, выбрав из списка любое значение. Запустите приложение, посмотрите, как изменился вид курсора. Закройте приложение.

Чтобы узнать про остальные свойства, можно воспользоваться справочной системой. Справочная система является контекстно-зависимой, при нажатии клавиши *F1* открывается подсказка, соответствующая текущей ситуации. Например, находясь на странице **СВОЙСТВ**, выберите какое-нибудь свойство и нажмите *F1*, отобразится справка о выделенном свойстве. Здесь можно увидеть назначение свойства, его тип, примеры использования и т. п.

Обработка событий

При выполнении приложений *Windows Forms* пользователь сам определяет порядок работы программы. Например, пусть программа только что начала работать. На экране показывается главная форма программы, и пользователь нажимает на какую-либо клавишу на клавиатуре. Нажатие на клавишу является событием, начинает работать определенный обработчик. Пользователь мог бы не нажимать на клавишу, и тогда работа программы пошла бы по другому сценарию. То есть здесь ситуация типа: может быть, а может и не быть.

Процедура, инициируемая событием, называется *обработчиком события*. В *C#* существует огромное число разнообразных событий. В этой лабораторной работе будут рассмотрены те, которые генерируются пользователем с помощью мыши (в следующей работе – с помощью клавиатуры, далее – некоторые другие важные события).

События мыши

Для формы событие **Click** возникает в том случае, если пользователь нажимает кнопку мыши в то время, когда курсор мыши находится на поле формы.

Упражнение 1. Напишите приложение, которое при нажатии мыши перекрашивает форму в синий цвет.

Решение

Создайте обработчик события Click: выберите в окне **СВОЙСТВ** страницу **Событий** и выполните двойной щелчок мышью в правой колонке, напротив события Click или по самому имени события (рис. 4).

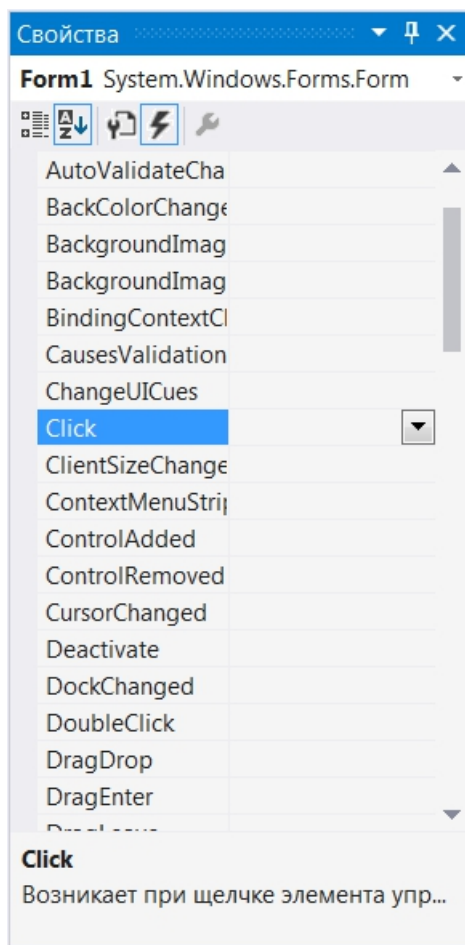


Рис. 4. Создание обработчика

На переднем плане появится окно редактора кода с помещенной сразу в нужное место заготовкой обработчика события Click (рис. 5).

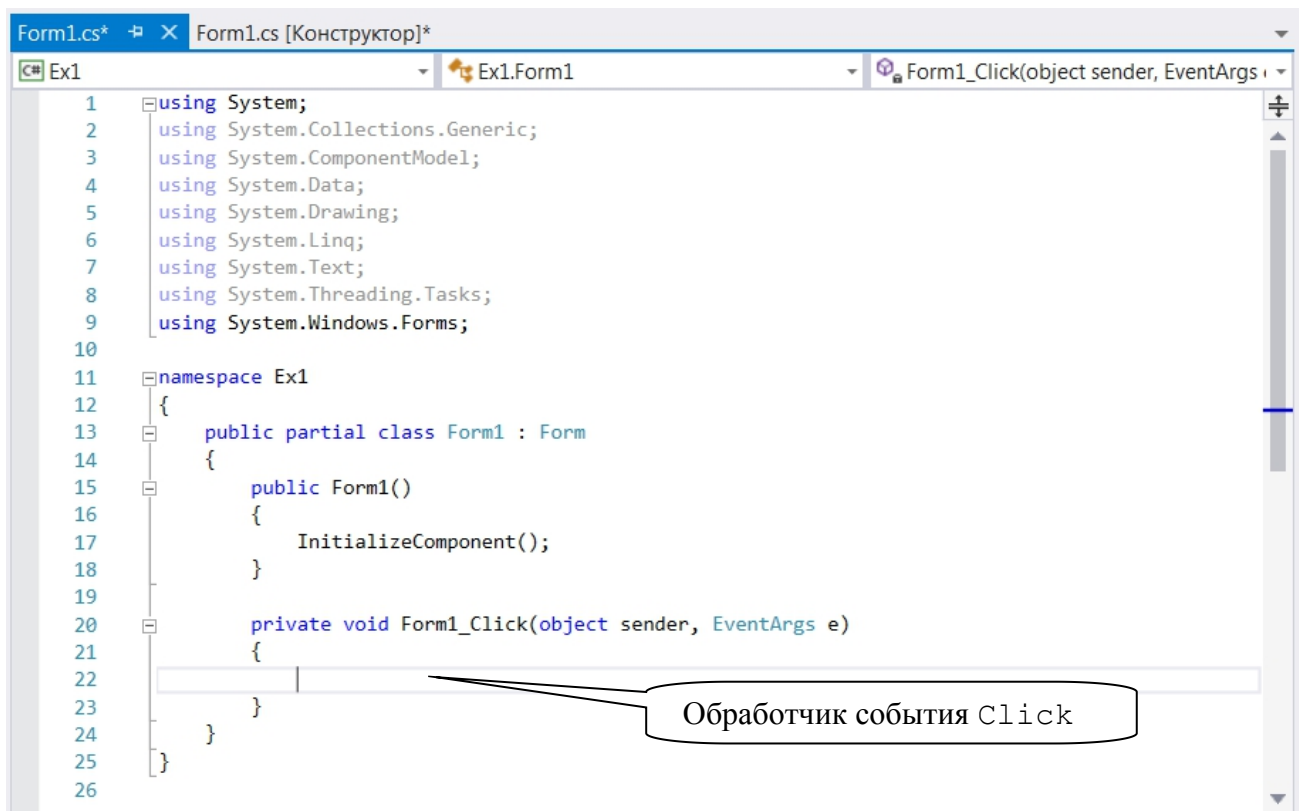


Рис. 5. Заготовка обработчика события Click

Первый параметр в обработчике события (*sender*) определяет объект, который породил событие. Второй параметр (*e*) содержит данные, которые могут быть использованы обработчиком события.

Для изменения цвета формы в обработчике события Click напомним следующий оператор:


```
private void MainForm_Click(object sender, EventArgs e)
{
    BackColor = Color.Blue;
}
```

BackColor – это свойство формы. *Color* – это класс, отвечающий за формирование цвета. Обращение к свойствам объекта происходит через точку.

Запустите приложение. Щелкните любой кнопкой мыши по форме. Убедитесь, что цвет формы стал синим. При повторном щелчке опять возникнет событие Click, форма снова становится синей, но пользователю это не заметно.

Упражнение 2. Напишите приложение, в котором при каждом щелчке левой кнопкой мыши произвольным образом изменяется цвет формы.

Решение

Сохраните текущее состояние проекта, выбрав пункт *Сохранить все* меню *Файл* или воспользовавшись клавишей на панели быстрого доступа .

Создайте в том же решении новый проект, укажите его имя *Ex2* и установите его стартовым.

Решим поставленную задачу. Во-первых, отметим, что с помощью события Click невозможно определить, какая кнопка мыши была нажата. Параметр *e* в этом событии имеет тип *EventArgs*, который не предоставляет подобной информации. Для выполнения задания мы воспользуемся другим событием – **MouseClick**. Параметр *e* в этом событии имеет тип *MouseEventArgs* и предоставляет гораздо больше информации,

например какая кнопка мыши была нажата, сколько раз, на сколько было повернуто колесико мыши, а также координаты мыши в момент генерации события.

Для получения различных цветов для формы воспользуемся методом *FromArgb()* класса *Color*. Будем передавать этому методу три параметра – уровни интенсивности красного, зеленого и синего цветов, соответственно (*Red*, *Green*, *Blue*). Для каждого уровня интенсивности цвета существует 256 возможных значений. Например, метод с параметрами (255, 0, 0) возвращает цветное значение для красного цвета.

Для получения случайного цвета все три параметра будем задавать случайными числами. В языке *C#* для генерации случайных чисел предусмотрен специальный класс *Random*. Для того чтобы воспользоваться методами этого класса, необходимо создать экземпляр класса с помощью конструктора (используя *new*). Для выбора случайного числа из диапазона $[0, n-1]$ будем использовать метод *Next(n)*. Описать переменную *Rand* нужно в классе формы:

```
public partial class Form1 : Form
{
    Random Rand;
    ...
}
```

В конструкторе формы (методе, имя которого совпадает с именем формы) добавьте оператор, инициализирующий генератор случайных чисел.

```
public Form1()
{
    InitializeComponent();
    //Инициализация генератора случайных чисел
    Rand = new Random();
}
```

В обработчике события *MouseClick* при щелчке левой кнопкой мыши будем формировать три случайных числа и устанавливать цвет формы.

```
private void Form1_MouseClick(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButton.Left)
    {
        //Определение трех целых случайных чисел [0..255]
        int a = Rand.Next(256);
        int b = Rand.Next(256);
        int c = Rand.Next(256);
        BackColor = Color.FromArgb(a, b, c);
    }
}
```

Запустите приложение. Убедитесь, что после каждого щелчка по форме левой кнопкой мыши цвет изменяется произвольным образом.

Упражнение 3. Напишите приложение, в котором при нажатии на левую кнопку мыши происходит смена цвета с красного на синий и, наоборот, с синего на красный.

Решение

Сохраните текущее состояние проекта. Создайте в том же решении новый проект, назовите его *Ex3*. Сделайте его стартовым в решении (*Назначить запускаемым проектом*).

При щелчке по любой кнопке мыши происходит событие *MouseClick*. В этом событии мы можем выяснить, по какой кнопке мыши произошло нажатие. В соответствии с условием задачи в обработчике этого события формы необходимо

проанализировать свойство *BackColor* формы: если его значение соответствует синему, то изменить его на красный, иначе – сделать синим. На языке *C#* это записывается так:

```
private void Form1_MouseClick(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButton.Left)
        if (BackColor == Color.Red)
            BackColor = Color.Blue;
        else BackColor = Color.Red;
}
```

Сохраните приложение, запустите его. Проверьте выполнение условия задачи.

Событие **DoubleClick** происходит, если пользователь выполняет двойной щелчок любой кнопкой мыши. Параметр *e* в этом обработчике имеет тип *EventArgs*, т. е. здесь невозможно определить какие-либо параметры нажатия мыши, например, какая кнопка мыши была нажата.

Событие **MouseDoubleClick** тоже происходит, если пользователь выполняет двойной щелчок любой кнопкой мыши, но параметр *e* в этом обработчике имеет тип *MouseEventArgs*, т. е. здесь можно определить какие-либо параметры нажатия мыши, например, какая кнопка мыши была нажата.

Упражнение 4. Напишите приложение, в котором при двойном щелчке левой кнопкой мыши изменяется вид курсора со значения *Cross* на значение *Hand* и обратно.

Решение

Сохраните текущее состояние проекта. Создайте в том же решении новый проект, назовите его *Ex4*. Сделайте его стартовым в решении.

Все значения свойства *Cursor* содержатся в классе *Cursors* (с помощью справки найдите все возможные значения свойства *Cursor*). При каждом двойном щелчке левой кнопкой мыши значение свойства *Cursor* будем изменять со значения *Cross* на значение *Hand* и обратно. Создайте обработчик события формы *MouseDoubleClick*:

```
private void Form1_MouseDoubleClick(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButton.Left)
        if (Cursor == Cursors.Cross) Cursor = Cursors.Hand;
        else Cursor = Cursors.Cross;
}
```

Запустите приложение. Проверьте, что происходит при каждом двойном щелчке левой кнопкой мыши.

Событие **MouseDown** возникает, если пользователь нажимает на правую, левую или среднюю кнопку мыши.

Событие **MouseUp** происходит, если пользователь освобождает кнопку мыши, которая была нажата.

Событие **MouseMove** происходит, если пользователь перемещает указатель мыши.

Все эти три события имеют два параметра: *sender* (тип *object*) и *e* (тип *MouseEventArgs*). Параметр *e* позволяет узнать ряд характеристик.

Чтобы проверить, какая кнопка мыши была нажата – левая, правая или средняя, необходимо использовать свойство *Button* и класс *MouseButton*.

```
if (e.Button == MouseButton.Left)...
if (e.Button == MouseButton.Right)...
if (e.Button == MouseButton.Middle)...
```


В значениях свойств X и Y параметр e содержит координаты указателя мыши, выраженные в пикселях относительно формы. Точка $[0, 0]$ расположена в левом верхнем углу формы, первая координата увеличивается вправо, вторая – вниз.

Свойство *Location* определяет те же координаты, что и X и Y , только имеет другой тип – тип *Point*.

Упражнение 5. Напишите приложение, позволяющее рисовать на форме отрезки.

Решение

Сохраните текущее состояние проекта. Создайте в том же решении новый проект, назовите его *Ex5*. Назначьте его запускаемым проектом.

Измените свойство *Text* формы на «Отрезки».

Графические операции выполняются с использованием класса **Graphics**. Класс *Graphics* предоставляет возможности для рисования (перо, кисть, шрифт, набор типовых геометрических фигур).

Чтобы создать экземпляр класса *Graphics*, нужно описать переменную:

```
Graphics Graph;
```

а затем инициализировать её:

```
Graph = CreateGraphics();
```

После этого можно использовать возможности класса *Graphics*.

Например, чтобы нарисовать на форме пунктирную линию красного цвета от точки с координатами $(25, 25)$ до точки $(250, 250)$, необходимы следующие операторы:

```
Pen MyPen = new Pen(Color.Red, 2);  
float[] dashValues = {5, 4};  
MyPen.DashPattern = dashValues;  
Graph.DrawLine(MyPen, 25, 25, 250, 250);
```

Поясним код. Первый оператор создает экземпляр класса *Pen* – перо. Указанные в скобках параметры определяют цвет пера и его толщину. Если второй параметр не указывать явно, по умолчанию он будет равен 1. Мы определили перо с именем *MyPen*, красного цвета, толщиной 2 пикселя.

Второй и третий операторы отвечают за пунктирный стиль кисти. Массив *dashValues* указывает, что пунктир будет таким: линия длиной 5 пикселей, пропуск между пикселями длиной 4 пикселя. Третий оператор устанавливает пунктирный стиль в соответствии с созданным массивом *dashValues*.

В четвертой строке кода используется метод *DrawLine()* – рисование отрезка. Его пять параметров определяют используемое для рисования перо и координаты начала и конца отрезка соответственно.

По условию задачи нам нужно рисовать отрезки. Отрезок – это часть прямой, расположенная между двумя точками. Пусть событие нажатие кнопки мыши послужит началом рисования отрезка (сохраним координаты этой точки), освобождение кнопки мыши – завершением рисования отрезка (нарисуем линию от сохраненной точки до точки, на которую указывает курсор).

Необходимо описать переменные *Graph* (тип *Graphics*), *MyPen* (тип *Pen*), x и y (тип *int* – целые числа). Если сделать это непосредственно в обработчике *MouseDown*, то эти переменные будут недоступны в обработчике *MouseUp*. Поэтому опишем все переменные в классе *Form1*.

```
public partial class Form1 : Form
{
    int x, y;
    Graphics Graph;
    Pen MyPen;
    ...
}
```

В конструкторе формы допишите следующие операторы:

```
Graph = CreateGraphics();
MyPen = new Pen(Color.Blue);
```

Создайте обработчик события MouseDown:

```
private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    x = e.X;
    y = e.Y;
}
```

Создайте обработчик события MouseUp:

```
private void Form1_MouseUp(object sender, MouseEventArgs e)
{
    Graph.DrawLine(MyPen, x, y, e.X, e.Y);
}
```

Запустите приложение. Попробуйте с помощью приложения нарисовать звезду.

Упражнение 6. Создайте приложение, позволяющее изображать произвольные линии.

Решение

Сохраните предыдущий проект. Создайте новый проект в том же решении, назовите его *Ex6*. Сделайте его стартовым.

Измените значение свойства *Text* формы на «Произвольные линии».

Произвольная линия в нашем приложении будет изображением следа курсора мыши. При перемещении курсора мыши происходит событие *MouseMove*. Создайте обработчик этого события и введите следующий код:

```
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    Graph.DrawLine(MyPen, x, y, e.X, e.Y);
    x = e.X;
    y = e.Y;
}
```

При этом

```
Graphics Graph;
Pen MyPen;
int x, y;
```

описаны в классе формы;

```
Graph = CreateGraphics();
MyPen = new Pen(Color.Magenta);
```

записано в конструкторе формы.

Запустите приложение. Убедитесь, что при перемещении курсора мыши на форме изображается линия, началом которой является верхний левый угол формы. Кроме того, не удается завершить рисование кривой. Устраним эти недостатки.

```
private void Form1_MouseMove(object sender, MouseEventArgs e)
```

```

{
    if (e.Button == MouseButton.Left)
        Graph.DrawLine(MyPen, x, y, e.X, e.Y);
    x = e.X;
    y = e.Y;
}

```

Запустите проект. Проверьте, что линия рисуется только тогда, когда перемещение мыши происходит с нажатой левой кнопкой.

Упражнение 7. Напишите приложение, которое изображает прямоугольники.

Решение

Создайте новый проект в том же решении, назовите его *Ex7*, установите в свойстве решения, чтобы запускался *Ex7*. Измените свойство *Text* формы на «Прямоугольники».

В классе *Graphics* определен метод *DrawRectangle(Pen, x, y, w, h)*, в котором параметры задают перо, координаты верхнего левого угла прямоугольника, а также его ширину и высоту соответственно. Если при рисовании всегда сначала нажимать кнопку мыши в верхнем левом углу прямоугольника и вести мышку к правому нижнему углу прямоугольника, то рисование прямоугольника запрограммировать достаточно просто, поскольку тогда координаты *x* и *y* будут известны (первый щелчок мыши), а *w* и *h* вычисляются как разница между соответствующими вторыми и первыми координатами. Однако при рисовании пользователь может двигать мышью по-разному, не всегда начиная с верхнего левого угла.

Опишите в классе формы целые *x* и *y*, *Graph* (тип *Graphics*) и *MyPen* (тип *Pen*). В конструкторе формы задайте значение *Graph* и *MyPen*. В событии *FormClosing* освободите ресурсы, занимаемые *Graph* и *MyPen*. Нажатие кнопки мыши (событие *MouseDown*) определяет координаты первой точки, а освобождение кнопки (событие *MouseUp*) – само рисования прямоугольника. Создайте обработчики этих событий *MouseDown* и *MouseUp*. Введите следующий код:

```

private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    x = e.X;
    y = e.Y;
}

private void Form1_MouseUp(object sender, MouseEventArgs e)
{
    int w = Math.Abs(e.X - x);
    int h = Math.Abs(y - e.Y);
    x = Math.Min(e.X, x);
    y = Math.Min(e.Y, y);
    Graph.DrawRectangle(MyPen, x, y, w, h);
}

```

Здесь используется класс *Math*, предоставляющий математические свойства и методы. В частности, мы используем методы *Abs()* – модуль, *Min()* – минимум. С помощью этого мы определяем ширину и высоту прямоугольника и координаты верхнего левого угла. При изменении значений переменных *x* и *y* определяются координаты верхнего левого угла прямоугольника.

Убедитесь в правильности работы программы.

Если метод *DrawRectangle()* поменять на *DrawEllipse()*, то вместо прямоугольников будут рисоваться эллипсы.

Упражнение 8. Напишите приложение, которое позволяет рисовать ломаные линии.

Решение

Создайте новый проект в том же решении, назовите его *Ex8*, установите в свойстве решения, чтобы запускался *Ex8*. Измените свойство *Text* формы на «Ломаные линии».

Начало (конец) рисования ломаной линии свяжем с событием `MouseDoubleClick`. Введем логическую переменную *draw* (тип *bool*): ее значение, равное *true*, будет обозначать, что ломаная сейчас рисуется, *false* – рисование ломаной завершено.

Создайте обработчик события `MouseDoubleClick`:

```
private void Form1_MouseDoubleClick
                                     (object sender, MouseEventArgs e)
{
    draw = !draw;
}
```

В обработчике события `MouseDown` будем рисовать звенья ломаной, если значение переменной *draw* равно *True*:

```
private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    if (draw) Graph.DrawLine(MyPen, x, y, e.X, e.Y);
    x = e.X;
    y = e.Y;
}
```

Не забудьте в классе формы описать все используемые переменные, в конструкторе формы – создать и задать начальное значение, в обработчике события закрытия формы – освободить ресурсы.

Используя написанное приложение, попробуйте изобразить треугольник, четырехугольник и пятиугольник.

Перечислим еще некоторые события мыши (в них параметр *e* имеет тип *EventArgs*).

Событие **MouseEnter** происходит, когда указатель мыши входит в ограничивающую область объекта.

Событие **MouseLeave** происходит, когда указатель мыши покидает ограничивающую область объекта.

Событие **MouseHover** происходит, когда указатель мыши наведен на элемент.

Событие **MouseCaptureChanged** происходит, когда элемент управления теряет или получает захват мыши.

Обычная последовательность обработки некоторых основных событий мыши:

1. `MouseEnter`.
2. `MouseMove`.
3. `MouseHover` или `MouseDown`.
4. `MouseUp`.
5. `MouseLeave`.

Задания для самостоятельного выполнения

1. Измените упражнение 8 для рисования замкнутых ломаных (при повторном двойном щелчке дорисовывается очередной отрезок ломаной, а также отрезок, соединяющий начальную и конечную точки ломаной).

События клавиатуры

Обработку любых событий клавиатуры можно выполнить, используя один или несколько из следующих трех обработчиков:

событие **KeyDown** происходит при нажатии любой клавиши, включая функциональные и специальные;

событие **KeyUp** происходит при отпускании любой клавиши;

событие **KeyPress** возникает при нажатии клавиши, имеющей символьное представление.

Каждый из обработчиков событий получает параметр *e*, который представляет информацию о нажатой клавише. В обработчике события **KeyPress** (*e* имеет тип *KeyPressEventArgs*) свойство *KeyChar* параметра *e* – это значение типа *Char*, представляющее символ *Unicode* (a...z, A...Z, a...я, А...Я, знаки препинания, цифры, клавиши *Enter*, пробел, *Esc*).

В обработчиках событий **KeyDown** и **KeyUp** параметр *e* (типа *KeyEventArgs*) имеет следующие основные свойства:

KeyCode – код клавиши (зависит от раскладки клавиатуры);

KeyValue – виртуальный код клавиши (не зависит от раскладки, для данной клавиши всегда один и тот же);

Alt, *Control* и *Shift* – были ли нажаты клавиши *Alt*, *Ctrl* и *Shift* соответственно (логические свойства);

Modifiers – какая из клавиш *Ctrl*, *Shift* и *Alt* была нажата.

Упражнение 9. Напишите приложение, позволяющее установить максимальные размеры окна приложения либо вернуть размеры окна в исходное состояние нажатием сочетания клавиш <Ctrl + пробел>.

Решение

Создайте новое решение, выполнив последовательность команд: *Файл – Создать – Проект...* Сохраните новое решение как *Lab3* в отдельной папке, имя проекта *Ex1*.

Измените значение свойства заголовка формы следующим образом: «Нажмите <Ctrl + пробел>».

Приложение должно изменять состояние окна формы в случае, когда одновременно нажаты обе клавиши. Для обработки такой ситуации можно воспользоваться событием **KeyDown**. Создайте обработчик этого события.

За состояние окна формы отвечает свойство *WindowState*, принимающее одно из трех возможных значений перечисления *FormWindowState*:

Minimized – свернутое окно;

Maximized – развернутое окно;

Normal – окно с размерами по умолчанию.

Для определения нажатия клавиши *Ctrl* воспользуемся значением соответствующего свойства параметра *e*. Добавьте в обработчик события **KeyDown** следующие операторы:

```
if (e.Control)
    if (WindowState == FormWindowState.Normal)
        WindowState = FormWindowState.Maximized;
    else WindowState = FormWindowState.Normal;
```

Запустите приложение, нажатие клавиши *Ctrl* приводит к изменению состояния окна формы.

Определим код клавиши пробел. Для вывода на экран сообщения с кодом клавиши добавьте в обработчик события `KeyDown` оператор:

```
MessageBox.Show(e.KeyValue.ToString());
```

Здесь *e.KeyValue* – это код виртуальной клавиши, метод *ToString()* преобразует код в значение строкового типа, *MessageBox.Show()* выводит строковое сообщение. Запустите приложение. При нажатии клавиши пробел в сообщение выводится число 32.

Исправьте первое условие в обработчике события `KeyDown` формы на следующее (операторы, изменяющие свойство *WindowState*, оставьте без изменений):

```
if ((e.KeyValue == 32) && (e.Control))
```

Запустите приложение. При одновременном нажатии клавиш *Ctrl* и пробел размеры окна приложения изменяются.

Другой способ решения рассмотренной задачи – с использованием перечисления *Keys*. В этом случае знание кода виртуальной клавиши не потребуется.

Измените первое условие, проверяющее нажатые клавиши, в обработчике события `KeyDown` на следующий код:

```
if ((e.KeyCode == Keys.Space) && (e.Control))
```

При этом выполнение программы не изменяется. Список *Keys* определяет все клавиши клавиатуры. Посмотрите с помощью *Help* его возможные значения.

При решении следующей задачи мы будем использовать **структуру** *DateTime*, предназначенную для хранения даты и времени в диапазоне от 01.01.0001 0:00:00 до 31.12.9999 23:59:59 в григорианском календаре. Значения времени измеряются в 100- наносекундных единицах, называемых тактами (тиками).

Некоторые свойства и методы структуры *DateTime* приведены в табл. 1.

Некоторые свойства и методы структуры *DateTime*

Название	Вид	Описание
<i>AddYears()</i> , <i>AddMonths()</i> , <i>AddDays()</i> , <i>AddHours()</i> , <i>AddMinutes()</i> , <i>AddSeconds()</i> , <i>AddMilliseconds()</i> , <i>AddTicks()</i>	Экземплярные методы	Добавляют к текущему экземпляру заданное количество лет, месяцев, дней, часов, минут, секунд, миллисекунд и тактов соответственно и возвращают новый экземпляр объекта <i>DateTime</i>
<i>Date</i>	Свойство	Возвращает дату текущего экземпляра структуры
<i>Day</i>	Свойство	Возвращает день месяца текущего экземпляра структуры
<i>DayOfWeek</i>	Свойство	Возвращает день недели текущего экземпляра структуры – константу перечисления <i>DayOfWeek</i> , (в диапазоне от нуля, что соответствует значению <i>DayOfWeek.Sunday</i> , до шести, что соответствует значению <i>DayOfWeek.Saturday</i>)
<i>DayOfYear</i>	Свойство	Целочисленное свойство, возвращающее номер дня в году текущего экземпляра структуры
<i>DaysInMonth()</i>	Статический метод	Возвращает количество дней в указанном месяце конкретного года
<i>Now</i>	Свойство	Возвращает текущие дату и время
<i>IsLeapYear()</i>	Статический метод	Определяет, является ли заданный год високосным
<i>Today</i>	Свойство	Возвращает текущую дату
<i>ToLongDateString()</i>	Экземплярный метод	Возвращает дату: день, название месяца, год через пробел
<i>ToLongTimeString()</i>	Экземплярный метод	Возвращает время в формате «часы:минуты:секунды»
<i>ToShortDateString()</i>	Экземплярный метод	Возвращает дату в формате «день.месяц.год»
<i>ToShortTimeString()</i>	Экземплярный метод	Возвращает время в формате «часы:минуты»
<i>Year</i> , <i>Month</i> , <i>Hour</i> , <i>Minute</i> , <i>Second</i> , <i>Millisecond</i> , <i>Ticks</i>	Свойства	Возвращают год, месяц, часы, минуты, секунды, миллисекунды и такты соответственно

Упражнение 10. Напишите приложение «Клавиатурный тренажер». Программа начинает работать при нажатии клавиши *Enter*. В центре формы отображается символ, который пользователю нужно напечатать на клавиатуре. В случае если пользователь правильно напечатал символ, отображается следующий символ задания, иначе звуковой сигнал указывает на ошибку. После того, как задание закончено (количество символов определяется константой), форма закрывается, выводя сообщение о том, за сколько секунд оно было выполнено.

Решение

Создайте новый проект в том же решении, назовите его *Ex2*, установите его стартовым. Измените значения свойств формы следующим образом:

Свойство	Значение
<i>Name</i>	<i>MainForm</i>
<i>Text</i>	Нажмите <i>Enter</i> , чтобы начать!
<i>Size</i>	390; 270

Опишите в классе формы строковую переменную *TargetString* для хранения символов задания и целочисленную переменную *CurrentIndex* для определения номера текущего символа в строке задания. Также опишите целочисленную константу *MaxCount* для определения количества символов, выводимых при проверке, а также целочисленную переменную *count*, изначально инициализированную в нуль, для сохранения количества уже выведенных символов:

```
string TargetString = " абвгдеёжзийклмнопрстуфхцчщъыьэюя";
int CurrentIndex;
const int MaxCount = 10;
int count = 0;
```

Кроме того, здесь же опишите следующее:

```
Graphics Graph;
Font MyFont = new Font("Arial", 32);
Random Rand = new Random();
DateTime start;
```

Для отображения графических объектов на форме будем использовать объект *Graph* класса *Graphics* (не забудьте проинициализировать *Graph* в конструкторе формы). Также мы проинициализировали *MyFont* – шрифт с нужными параметрами (здесь задан шрифт размера 32, семейство шрифтов *Arial*), *Rand* для генерации случайных чисел и переменную *start* типа *DateTime* для сохранения времени начала проверки.

В обработчике события *FormClosing* освободите ресурсы, занимаемые объектами *Graph* и *MyFont*.

Поскольку при наборе слов используются только символьные клавиши, воспользуемся обработчиком события *KeyPress*.

```
private void Form1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (((int)e.KeyChar == 13) && (count == 0))
    {
        //если нажат Enter и символы еще не выводились
        start = DateTime.Now; //запоминаем время начала
        //определяем номер символа в строке
        CurrentIndex = Rand.Next(TargetString.Length);
        //отображаем этот символ на форме
        Graph.DrawString(TargetString.Substring(CurrentIndex, 1),
            MyFont, Brushes.Black, 160, 75);
        //увеличиваем количество выведенных символов
        count = 1;
        //изменяем заголовок
        Text = "Нажми правильную клавишу!";
    }
    else if ((count > 0) && (e.KeyChar ==
TargetString[CurrentIndex]))
    {
        //если проверка началась и введен правильный символ
        if (count == MaxCount) //если проверка закончилась
        {
            //определяем количество секунд с начала проверки
            int time = DateTime.Now.Subtract(start).Seconds;
```

```

        //выводим сообщение
        MessageBox.Show("Время выполнения = " +
                        time.ToString() + " секунд");
        Close(); //закрываем форму
    }
    else //введен не последний символ
    {
        //очищаем форму цветом формы
        Graph.Clear(BackColor);
        //определяем номер символа в строке
        CurrentIndex = Rand.Next(TargetString.Length);
        //отображаем этот символ на форме
        Graph.DrawString(TargetString[CurrentIndex].ToString(),
                        MyFont, Brushes.Black, 160,
                        75);
        //увеличиваем количество выведенных символов count++;
    }
}
//если введен неверный символ,
//воспроизводим звуковой сигнал
else System.Media.SystemSounds.Hand.Play();
}

```

Поясним код. В соответствии с заданием проверка начинает выполняться после нажатия клавиши *Enter*, код символа которой равен 13 (не путайте код символа и виртуальный код клавиши, который мы использовали в упражнении 3.1, – это два разных значения). Заметьте, что для преобразования символьного значения нажатой клавиши *e.KeyChar* в код символа используем явное приведение типов. Второе условие для начала проверки – переменная *count* должна быть равна нулю. Без этого условия все последующие нажатия на клавишу *Enter* расценивались бы как запуск проверки, а поскольку переменная *count* увеличивается, то заход в эту ветку условного оператора будет выполнен только один раз.

Теперь рассмотрим, что выполняется при начале проверки. Во-первых, в переменной *start* с помощью метода *Now()* класса *DateTime* запоминается время начала проверки. Переменной *CurrentIndex* присваивается произвольное значение в диапазоне от нуля включительно до длины строки *TargetString* не включительно (нумерация символов в строке начинается с нуля). Далее символ строки с этим номером отображается на форме с помощью метода *DrawString()*. Кроме того, изменяется заголовок формы и значение переменной *count*, определяющей, сколько символов задания уже было выведено на форму.

Когда проверка уже началась и на форме выводится очередной символ, при нажатии правильной клавиши мы заходим в следующую ветку условного оператора. Внутри этой ветки вложен еще один условный оператор, проверяющий, был ли очередной нажатый символ последним в проверке. Если да (при этом значение переменной *count* совпадает с константой *MaxCount*), то в целочисленной переменной *time* определяется количество секунд, прошедших с начала выполнения задания, для этого из текущего времени вычитается время начала выполнения задания и это значение переводится в секунды. Затем это значение выводится в сообщении диалогового окна с помощью метода *Show()* класса *MessageBox*, после чего форма закрывается методом *Close()*. Если же введен правильный символ, но проверка еще не завершена, выполняется следующее. Во-первых, с помощью метода *Clear()* класса *Graphics* очищается вся поверхность рисования и выполняется заливка поверхности указанным цветом формы.

Далее, как и при начале проверки, определяется новый номер символа, он выводится на форме, а также увеличивается значение переменной *count*.

Если нажатая клавиша при выполнении задания не соответствует выведенному на форме символу, то выводится звуковой сигнал. Для этого используется свойство *Hand* класса *System.Media.SystemSounds*, получающее звуковой файл, связанный с событием программы *Hand* в текущей звуковой схеме *Windows*. Звук воспроизводится с помощью метода *Play()*.

Запустите приложение, убедитесь в правильности его работы. Доработайте приложение, чтобы при завершении проверки выводилась также информация о количестве допущенных ошибок.

Упражнение 11. Напишите приложение, отображающее на форме график функции $y = -x^2 + 3$.

Решение

Создайте новый проект в том же решении, назовите его *Ex3*, установите его стартовым.

Измените значения свойств формы следующим образом:

Свойство	Значение
<i>Name</i>	<i>MainForm</i>
<i>Text</i>	График
<i>WindowState</i>	<i>Maximized</i>
<i>BackColor</i>	<i>Window</i>

При построении на экране любого графика используются как фактические координаты графика функции в декартовой системе координат, так и экранные координаты для отображения графика на форме (рис. 6). Отметим, что начало координат экранной системы координат находится в левом верхнем углу, ось абсцисс направлена вправо, а ось ординат – вниз.

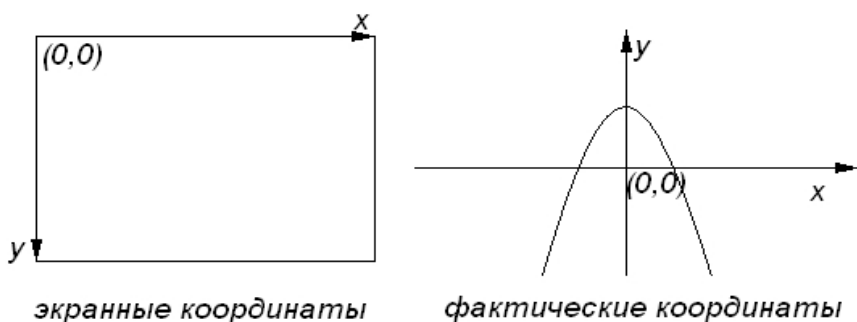


Рис. 6. Экранные и фактические координаты

Для преобразования фактических координат в экранные воспользуемся следующими формулами:

$$x_э = x_ф + x_ф \times k,$$

$$y_э = y_ф - y_ф \times k,$$

где координаты $(x_ф, y_ф)$ задают положение начала координат на форме, $(x_ф, y_ф)$ – это фактические координаты в декартовой системе координат, $(x_э, y_э)$ – экранные координаты, k – коэффициент масштабирования.

При отображении графика на форме будем использовать метод *DrawLine()* класса *Graphics*. В этом случае на форме будет отображаться отрезок прямой, соединяющий две соседние точки графика. Опишите и проинициализируйте объект *Graph* класса *Graphics*, и

объект *MyPen* класса *Pen*, а также освободите занимаемые ими ресурсы в обработчике события *FormClosing*.

Будем выводить график функции на экран при нажатии клавиши *Enter*. Создайте обработчик события *KeyDown* и поместите в него следующий код:

```
if (e.KeyCode == Keys.Enter)
{
    //отображение графика на форме
}
```

Итак, для отображения графика функции нам потребуются следующие переменные (опишите их в обработчике события *KeyDown*):

целочисленные переменные x_0 и y_0 , определяющие положение начала координат на форме. Для нашего примера зададим начало координат в середине формы (свойство *ClientSize* определяет размер клиентской области элемента управления – в нашем случае для формы размер без учета заголовка и границ):

```
int x0 = this.ClientSize.Width / 2;
int y0 = this.ClientSize.Height / 2;
```

целочисленные переменные x_1 , y_1 и x_2 , y_2 – экранные координаты концов отрезка графика:

```
int x1, y1, x2, y2;
```

фактические координаты текущей точки графика функции:

```
double x, y;
```

Будем отображать график функции в заданном диапазоне изменения аргумента $[x_{Min}, x_{Max}]$ с заданным приращением аргумента *step*. Введем соответствующие константные значения:

```
const double xMin = -5;
const double xMax = 5;
const double step = 0.01;
```

Также опишем константу для определения коэффициента масштабирования:

```
const double k = 5.5;
```

Для отображения графика функции вставьте в обработчик события *KeyDown* следующие операторы:

```
//фактические координаты в начальной точке заданного диапазона
x = xMin;
y = - x * x + 3;
//соответствующие им экранные координаты
x1 = (int) (x0 + x * k);
y1 = (int) (y0 - y * k);
while (x < xMax)
{
    //определение фактических координат графика в следующей точке
    x = x + step;
    y = - x * x + 3;
    //соответствующие им экранные координаты
    x2 = (int) (x0 + x * k);
    y2 = (int) (y0 - y * k);
}
```

```

//вывод отрезка графика на экран
Graph.DrawLine(MyPen, x1, y1, x2, y2);
//запоминаем текущие координаты
x1 = x2;
y1 = y2;
}

```

Отметим, что каждый раз отрезок графика рисуется от предыдущей сохраненной точки графика ($x1, y1$) до текущей ($x2, y2$).

Сохраните изменения, внесенные в проект. Запустите приложение, убедитесь в правильности его работы. Для большей наглядности, самостоятельно добавьте отображение осей координат (рис. 7).

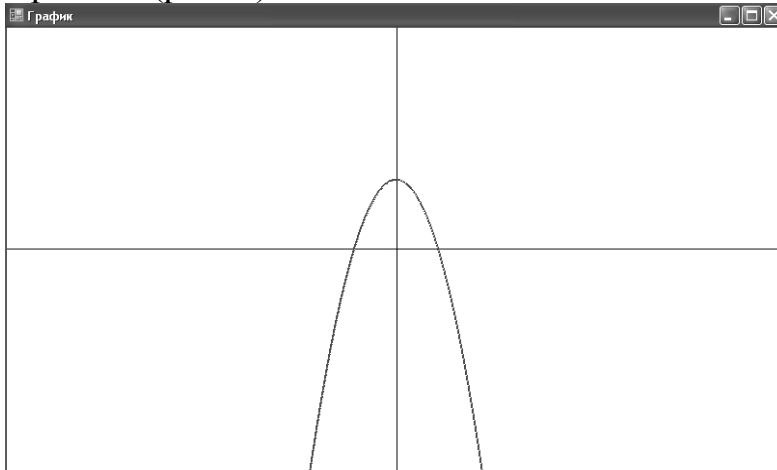


Рис. 7. Приложение «График»

Задания для самостоятельного выполнения

2. Напишите программу отображения графика функции $y=\text{tg}(x)$. Введите два коэффициента масштабирования: k_x – коэффициент масштабирования по оси x и k_y – коэффициент масштабирования по оси y . При нажатии на клавиши управления курсором Вверх/Вниз – увеличивается/уменьшается значение коэффициента k_x , при нажатии на клавиши Влево/Вправо – увеличивается/уменьшается значение коэффициента k_y , а график перерисовывается.

Системные события

События мыши и клавиатуры, рассмотренные в двух предыдущих лабораторных работах, вызываются воздействиями пользователя на программу с помощью мыши и клавиатуры. События, рассматриваемые здесь, исходят непосредственно от ОС *Windows*.

Событие **Load** происходит при запуске приложения, до первоначального отображения формы. Можно использовать это событие для инициализации объектов, используемых на протяжении всего времени работы с приложением.

Событие **Paint** генерируется при перерисовке формы.

Событие **Shown** происходит только при первом отображении формы; последующие сворачивание, разворачивание, отображение и перерисовка формы не приводят к генерации этого события.

Событие **Activated** происходит при активации формы пользователем или с помощью метода *Activate()*. Можно использовать это событие для таких задач, как обновление содержимого формы на основе изменений ее данных во время неактивного состояния.

Событие **Deactivate** происходит, если, например, пользователь переключается на другое окно в текущем приложении или на окно другого приложения. За время работы приложения форма может быть активирована и деактивирована много раз.

События, связанные с изменением размеров элемента управления: **Resize**, **SizeChanged**, **ResizeBegin**, **ResizeEnd**.

Событие **Resize** происходит при изменении размеров формы.

Событие **ResizeBegin** происходит, когда форма входит в режим изменения размеров (если пользователь начинает с помощью мыши изменять размеры формы).

Событие **ResizeEnd** происходит при выходе формы из режима изменения размеров (например, когда пользователь заканчивает перетаскивание какой-либо границы).

Как правило, во время операции изменения размера выполняется следующий набор событий:

1. Отдельное событие **ResizeBegin** происходит при переходе формы в режим изменения размеров.

2. Событие **Resize** генерируется при изменении свойства формы *Size* (в зависимости от ситуации или не происходит, или выполняется один или больше раз).

3. Отдельное событие **ResizeEnd** выполняется при выходе формы из режима изменения размеров.

Пары событий **ResizeBegin** и **ResizeEnd** также возникают при перемещении пользователем формы путем щелчка и перетаскивания за строку заголовка (событие **Resize** при этом не генерируется). Кроме того, события **ResizeBegin** и **ResizeEnd** не создаются при программном управлении формой, к примеру, путем изменения свойства *Size* или *Location*.

Событие **FormClosing** происходит при закрытии формы. В этом событии ещё можно отменить закрытие формы.

Событие **FormClosed** происходит после закрытия формы пользователем или при помощи вызова метода формы *Close()*. При этом форма закрывается, все ресурсы, созданные внутри объекта, освобождаются, а форма удаляется.

Также *C#* предоставляет программистам большое число событий, генерирующихся при изменении какого-либо свойства элемента управления, например **BackColorChanged** (при изменении свойства *BackColor*), **TextChanged** (свойства *Text*), **CursorChanged** (свойства *Cursor*) и многие другие. Название таких событий заканчивается на - **Changed**.

Упражнение 12. Для демонстрации описанных событий напомним небольшое приложение.

Решение

Создайте новый проект *Ex4* в решении *Lab3*.

Измените свойство *Text* формы, сделав заголовок пустым. Установите размер формы (*Size*): ширина (*Width*) – 500, высота (*Height*) – 500.

Опишите в классе формы следующие объекты.

```
Graphics Graph;  
Font MyFont;  
SolidBrush MyBrush;  
Random Rand;
```

Создайте обработчики событий *Load*, *Paint* и *Shown*. В обработчике *Shown* напишите оператор, добавляющий к заголовку формы название события.

```
Text = Text + "Shown";
```

В обработчике *Load* задайте значение используемых объектов и также измените значение заголовка формы:

```
Graph = CreateGraphics();  
Rand = new Random();  
MyFont = new Font("Arial", 30, FontStyle.Bold);  
MyBrush = new SolidBrush(Color.Black);  
Text = Text + "Load";
```

В конструкторе объекта *MyFont* (шрифта, имеющего тип *Font*) можно задать такие свойства, как семейство шрифтов, размер символов и стиль начертания. Здесь задан полужирный шрифт размера 30, тип *Arial*.

В обработчике *Paint* формы напишите следующие операторы.

```
Graph.DrawString("Упражнение 3.4", MyFont, MyBrush, 50, 150);  
MyBrush.Color = Color.FromArgb(Rand.Next(256), Rand.Next(256),  
Rand.Next(256));
```

Метод *DrawString()* выводит на форме строку «Упражнение 5.1» шрифтом *MyFont*, кистью *MyBrush* в позиции (50, 150). После вывода сообщения цвет кисти меняется на случайный с помощью метода *FromArgb()*.

Запустите приложение. По заголовку формы определите, в какой последовательности и сколько раз выполняются события *Load* и *Shown*. С помощью мыши, произвольно меняя размер формы, определите, когда выполняется событие *Paint* (при этом произойдет изменение цвета сообщения).

Добавим в проект еще одну форму, выполнив команду *Проект – Добавить форму Windows...* В диалоговом окне нажмем кнопку *Добавить*, установив имя формы *Form2.cs*. Это же можно сделать, щелкнув правой кнопкой мыши по имени проекта *Ex1* в *Обозревателе решений* и выбрав *Добавить – Форма Windows...*

Очистите значение свойства *Text* формы. Установите значение ширины формы, равное 500. Напишите обработчики событий *Load*, *Shown*, *Resize*, *Activated* и *DeActivate*. В каждом из них напишите оператор, добавляющий к заголовку формы название соответствующего события (например, в событии *Load* должно быть написано *Text = Text + "Load";*).

Для того чтобы вторая форма отображалась во время выполнения приложения, в обработчике события *DoubleClick* первой формы напишите операторы:

```
Form2 form2 = new Form2();  
form2.Show();
```

Запустите приложение. Дважды щелкнув по первой форме, вы загрузите вторую форму. Обратите внимание на последовательность выполнения событий (см. заголовок формы). Не закрывая второй формы, щелкните мышью по первой форме, а потом снова по второй форме. Отметьте для себя, когда выполняются события *Activated* и *Deactivate*. Измените мышью размер формы (событие *Resize*). Закройте обе формы приложения.

Отметим, что для открытия формы можно использовать метод *Show()*, как в нашем примере, и метод *ShowDialog()*. Второй метод открывает форму как модальное диалоговое окно: в этом случае, пока открыта вторая форма, пользователь не сможет сделать первую форму активной.

Измените в своей программе метод *Show()* на метод *ShowDialog()* и протестируйте работу приложения.

Установим точки прерывания. Простейший способ установить точку прерывания – щелкнуть мышью на левом краю окна редактирования. При этом выбранная для остановки строка выделяется красной полосой, на левом краю строки появляется красный круглый значок (рис. 8).

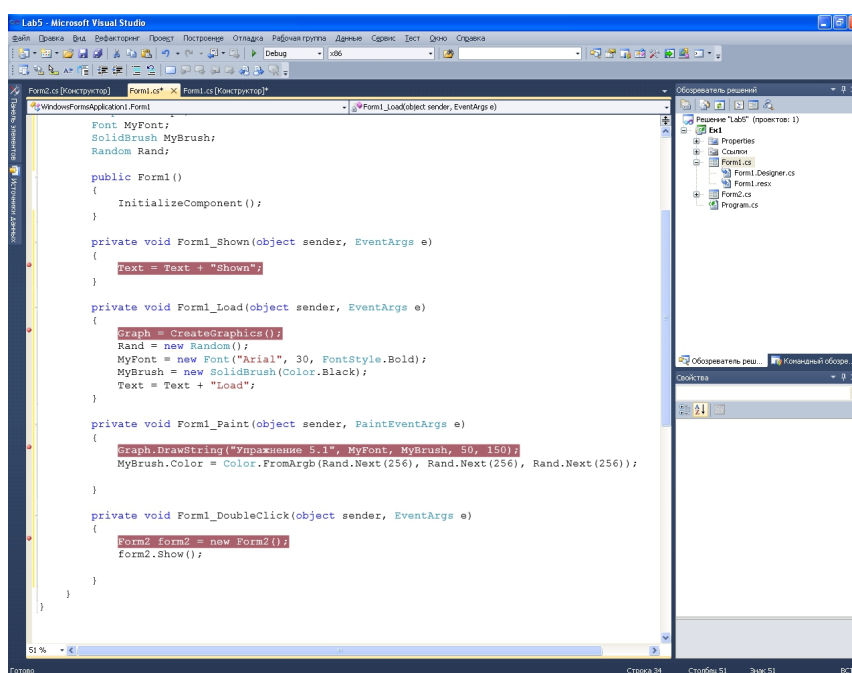


Рис. 8. Установка точек прерывания



Другой способ установить точку прерывания – вызвать команду меню *Отладка – Точка останова* или нажать на клавишу *F9*. Также можно выполнить команду *Отладка – Создать точку останова – Прервать в функции...* Появится диалоговая панель добавления точки прерывания *Создать точку останова* с несколькими полями, среди которых имя функции и номера строки (где будет задана точка прерывания) и символа в этой строке.

Командой *Отладка – Окна – Точки останова* открывается окно *Точки останова* ниже редактора кода. Здесь, например, можно задать параметр *Условие*, в котором вводится выражение, при истинности которого точка прерывания сработает, иначе выполнение приложения не будет прервано при прохождении через эту строку. Дополнительно можно задать количество проходов, после которых точка прерывания переходит в активное состояние, и многие другие параметры точек останова. Эти же параметры можно задать, щелкнув правой кнопкой мыши по красному круглому значку самой точки останова в редакторе кода.

Кроме точек останова для отладки программы могут быть полезны окна *Локальные* и *Контрольные значения*. Чтобы их открыть, нужно запустить отладку программы (*F5*), после чего выбрать пункт меню *Отладка – Окна – Локальные* или *Отладка – Окна – Контрольные значения – Контрольные значения1*.

Окно *Локальные* отображает имена, значения и типы локальных переменных в текущей области действия. В этом окне автоматически отображаются все параметры выполняющегося в текущий момент метода.

В окне *Контрольные значения1* отображаются имена, значения и типы переменных и выражений, указанных пользователем. Для добавления новых элементов для просмотра достаточно щелкнуть по ним в программном коде правой кнопкой мыши и выбрать пункт *Добавить контрольное значение*.

Установите точки останова в обработчиках событий первой формы: *Load*, *Shown*, *FormClosed* и *DoubleClick*; второй формы: *Load* и *Shown*. Запустите приложение. Приложение начнет выполняться. Первая остановка произойдет на процедуре *Form1_FormLoad* (это говорит о том, что произошло событие загрузки первой формы), продолжите выполнение приложения, выполнив команду *Отладка – Продолжить* (можно также щелкнуть по кнопке с зеленым треугольником  на панели инструментов или нажать клавишу *F5*). Чтобы проследить за выполнением всех операторов пошагово, можно выполнять команду *Отладка – Шаг с заходом* (можно также щелкнуть по кнопке  на панели инструментов или нажать клавишу *F11*). Проследите далее за выполнением приложения по шагам. Можно также нажимать не на *F11*, а на *F10* (*Отладка – Шаг с обходом*), тогда при пошаговой отладке не будет отображаться выполнение тел методов.

Выполните команду *Отладка – Удалить все точки останова*, чтобы удалить все точки останова.

Добавьте обработчик *FormClosing* для первой формы. Здесь можно, например, спросить пользователя, действительно ли он хочет немедленно закрыть форму. Запишите в обработчике следующий оператор:

```
if (MessageBox.Show("Вы действительно хотите закрыть приложение?",  
"Закрытие приложения", MessageBoxButtons.OKCancel) == DialogResult.Cancel)  
    e.Cancel = true;
```

Логическое свойство *Cancel* параметра *e* определяет, отменять ли закрытие формы.

В этом фрагменте кода используется диалоговое окно *MessageBox*. Оно имеет много вариантов комбинаций параметров. В используемом варианте первый параметр – это сообщение, которое выводится в окне. Второй параметр – заголовок диалогового окна. Третий параметр определяет, какие кнопки должны появиться в диалоговом окне, его возможные значения – *OK*, *OKCancel*, *AbortRetryIgnore*, *YesNoCancel*, *YesNo*, *RetryCancel*.

Установите точки останова в обработчиках *FormClosing* и *FormClosed*. Запустите приложение. Изучите, в какой последовательности происходят эти события.

Задания для самостоятельного выполнения

3. Напишите приложение, которое в заголовке формы выводит ее размеры и координаты на экране, а по центру формы независимо от ее размеров изображает круг радиусом 30 пикселей. Минимальный размер формы – 150×150.