## Лабораторная работа № 3. 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, а…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*](ms-help://MS.VSCC.v90/MS.MSDNQTR.v90.en/fxref_system.windows.forms/html/9337af49-b36d-0f9f-dd56-0f917bcdbeb3.htm) – код клавиши (зависит от раскладки клавиатуры);
* [*KeyValue*](ms-help://MS.VSCC.v90/MS.MSDNQTR.v90.en/fxref_system.windows.forms/html/3bd0f0c9-0be9-5adb-8a84-5c2ae59bf53d.htm) – виртуальный код клавиши (не зависит от раскладки, для данной клавиши всегда один и тот же);
* [*Alt*](ms-help://MS.VSCC.v90/MS.MSDNQTR.v90.en/fxref_system.windows.forms/html/c4ea6c44-78ee-a035-5999-38a0306a0d96.htm), [*Control*](ms-help://MS.VSCC.v90/MS.MSDNQTR.v90.en/fxref_system.windows.forms/html/692deb65-2154-e3b0-c831-a7ea0e99b04b.htm) и [*Shift*](ms-help://MS.VSCC.v90/MS.MSDNQTR.v90.en/fxref_system.windows.forms/html/e4b524d8-786c-a02c-8e7e-a12b214d0078.htm) – были ли нажаты клавиши *Alt*, *Ctrl* и *Shift* соответственно (логические свойства);
* [*Modifiers*](ms-help://MS.VSCC.v90/MS.MSDNQTR.v90.en/fxref_system.windows.forms/html/05d80223-18d2-3477-f4e2-d3c5387c7f64.htm) – какая из клавиш *Ctrl*, *Shift* и *Alt* была нажата.

**Упражнение 3.1.** 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. 3.1.

*Таблица 3.1*

**Некоторые свойства и методы структуры *DateTime***

|  |  |  |
| --- | --- | --- |
| Название | Вид | Описание |
| *AddYears*()*,*  *AddMonths*()*,*  *AddDays*()*,*  *AddHours*()*,*  *AddMinutes*()*,*  *AddSeconds*()*,*  *AddMilliseconds*(),  *AddTicks*() | Экземплярные методы | Добавляют к текущему экземпляру заданное количество лет, месяцев, дней, часов, минут, секунд, миллисекунд и тактов соответственно и возвращают новый экземпляр объекта [*DateTime*](http://msdn.microsoft.com/ru-ru/library/system.datetime.aspx) |
| *Date* | Свойство | Возвращает дату текущего экземпляра структуры |
| *Day* | Свойство | Возвращает день месяца текущего экземпляра структуры |
| *DayOfWeek* | Свойство | Возвращает день недели текущего экземпляра структуры – константу перечисления [*DayOfWeek*](http://msdn.microsoft.com/ru-ru/library/system.dayofweek(v=VS.90).aspx), (в диапазоне от нуля, что соответствует значению *DayOfWeek.Sunday*, до шести, что соответствует значению *DayOfWeek.Saturday*) |
| *DayOfYear* | Свойство | Целочисленное свойство, возвращающее номер дня в году текущего экземпляра структуры |
| *DaysInMonth*() | Статический метод | Возвращает количество дней в указанном месяце конкретного года |
| *Now* | Свойство | Возвращает текущие дату и время |
| *IsLeapYear*() | Статический метод | Определяет, является ли заданный год високосным |
| *Today* | Свойство | Возвращает текущую дату |
| *ToLongDateString*() | Экземплярный метод | Возвращает дату: день, название месяца, год через пробел |
| *ToLongTimeString*() | Экземплярный метод | Возвращает время в формате  «часы:минуты:секунды» |
| *ToShortDateString*() | Экземплярный метод | Возвращает дату в формате «день.месяц.год» |
| *ToShortTimeString*() | Экземплярный метод | Возвращает время в формате «часы:минуты» |
| *Year,*  *Month,*  *Hour*  *Minute,*  *Second,*  *Millisecond,*  *Ticks* | Свойства | Возвращают год, месяц, часы, минуты, секунды, миллисекунды и такты соответственно |

**Упражнение 3.2.** 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 |
| *Name* | *MainForm* |
| *Text* | Нажмите *Enter*, чтобы начать! |
| *Size* | 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 = " абвгдеёжзийклмнопрстуфхцчшщъыьэюя";

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 3.3**. Напишите приложение, отображающее на форме график функции *y*=‑*x*2+3.

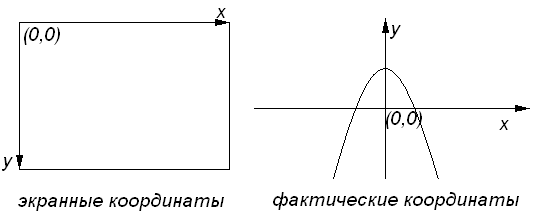
**Решение**

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

Измените значения свойств формы следующим образом:

|  |  |
| --- | --- |
| Свойство | Значение |
| *Name* | *MainForm* |
| *Text* | График |
| *Windowstate* | *Maximized* |
| *BackColor* | *Window* |

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. 3.1). Отметим, что начало координат экранной системы координат находится в левом верхнем углу, ось абсцисс направлена вправо, а ось ординат – вниз.



*Рис. 3.1.* Экранные и фактические координаты

Для преобразования фактических координат в экранные воспользуемся следующими формулами:

*хэ=х0+хф×k*,

*уэ=у0–уф×k*,

где координаты (*х0,у0*) задают положение начала координат на форме, (*хф, уф*) – это фактические координаты в декартовой системе координат, (*хэ, уэ*) – 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 *х0* и *у0*, 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 *х1*, *у1* и *х2*, *у2* – 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 [*xMin,xMax*] 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

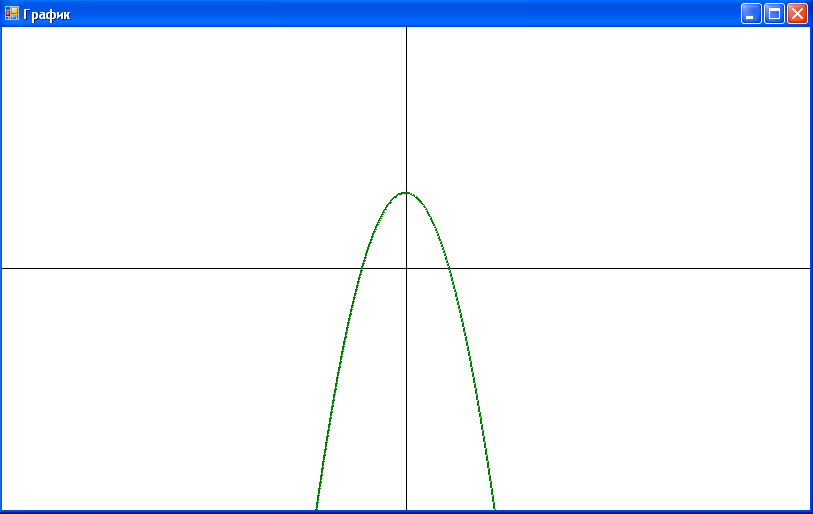
x1 = x2;

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. 3.2).



*Рис. 3.2.* Приложение «График»

### Задания для самостоятельного выполнения

* 1. Напишите приложение, выводящее всю возможную информацию о нажатой на клавиатуре клавише (например, буква это или цифра, заглавная или строчная буква, кириллица или латинский алфавит и т. д.).
  2. Напишите программу отображения графика функции *y*=*tg*(*x*). Введите два коэффициента масштабирования: *kх –*коэффициент масштабирования по оси *х* и *ky –*коэффициент масштабирования по оси *у*. При нажатии на клавиши управления курсором Вверх/Вниз – увеличивается/уменьша­ет­ся значение коэффициента *kx*, при нажатии на клавиши Влево/Вправо – увеличивается/уменьшается значение коэффициента *ky*, а график перерисовывается.
  3. Напишите такое приложение: при нажатии на клавишу *Enter* в центре формы появляется квадрат размера 10×10. Пользователь может управлять его положением с помощью клавиш управления курсором и изменять его размеры с помощью сочетания клавиш *Shift* и клавиш управления курсором. При этом размеры квадрата могут изменяться только в пределах заданного диапазона (длина стороны может принимать значение из диапазона [1..*Min*(ширина формы, высота формы) – 10]), а сам квадрат не может перемещаться за пределы формы. При нажатии на клавишу *F*1 выводится диалоговое окно с информацией о создателе программы.

***Баллы:*** первая задача – 1 балл, вторая и третья задачи – по 2 балла.

## 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*](http://msdn.microsoft.com/ru-ru/library/system.windows.forms.form.activate.aspx)(). Можно использовать это событие для таких задач, как обновление содержимого формы на основе изменений ее данных во время неактивного состояния.

Событие **Dectivate** 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**](http://msdn.microsoft.com/ru-ru/library/system.windows.forms.control.sizechanged.aspx), **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*](http://msdn.microsoft.com/ru-ru/library/25w4thew.aspx) (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](http://msdn.microsoft.com/ru-ru/library/system.windows.forms.form.resizeend.aspx) также возникают при перемещении пользователем формы путем щелчка и перетаскивания за строку заголовка (событие [Resize](http://msdn.microsoft.com/ru-ru/library/system.windows.forms.control.resize.aspx) при этом не генерируется). Кроме того, события ResizeBegin и [ResizeEnd](http://msdn.microsoft.com/ru-ru/library/system.windows.forms.form.resizeend.aspx) 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*](http://msdn.microsoft.com/ru-ru/library/system.windows.forms.form.close(v=VS.90).aspx)(). 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 3.4.** Для демонстрации описанных событий напишем 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();

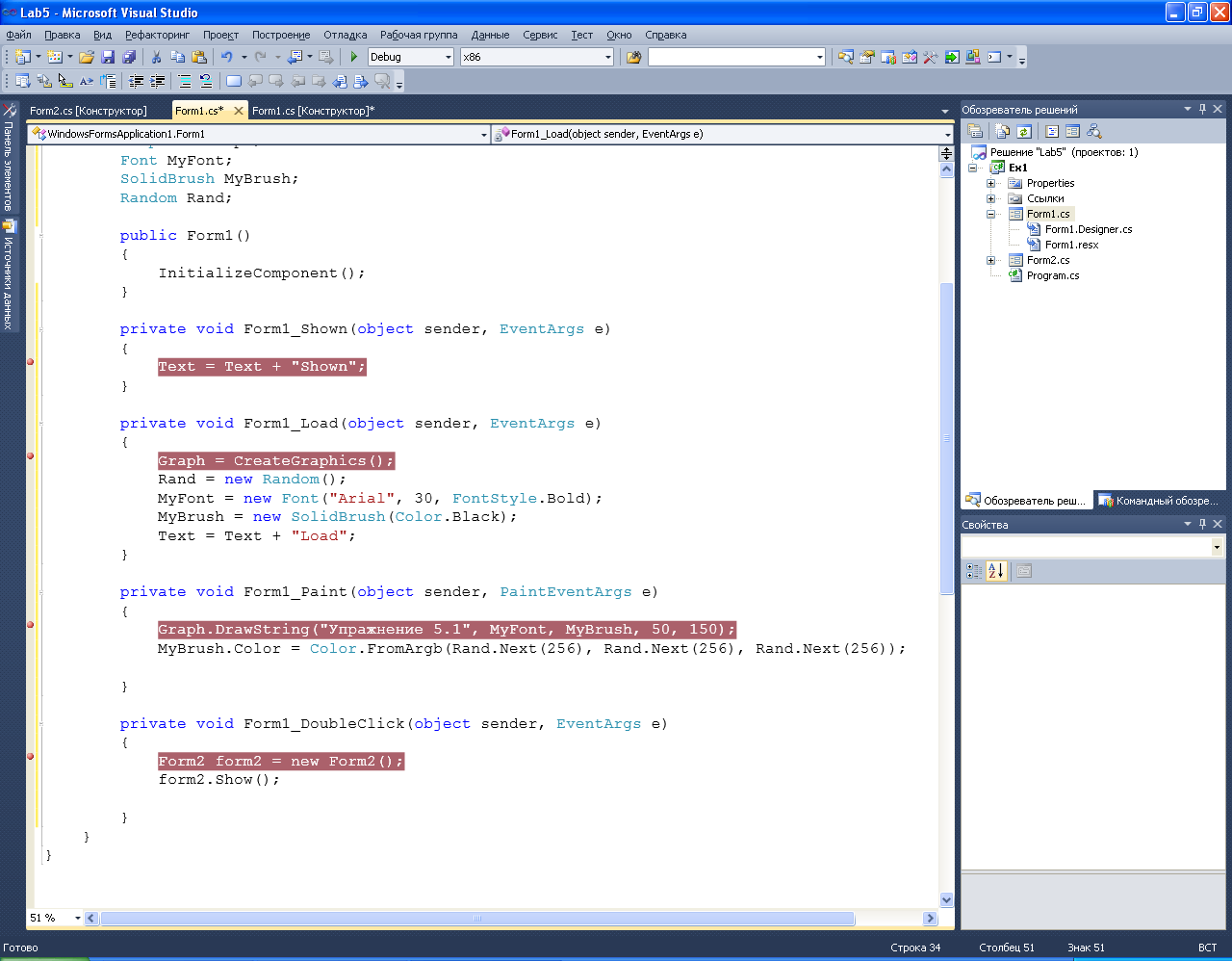
form2.Show();

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 строчки появляется красный круглый значок (рис. 3.1).



*Рис. 3.1.* Установка точек прерывания

Другой способ установить точку прерывания – вызвать команду меню *Отладка – Точка останова* или нажать на клавишу *F*9. 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 (*F*5), 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 *F*5). 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 *F*11). Проследите далее за выполнением приложения по шагам. Можно также нажимать не на *F*11, а на *F*10 (*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

* 1. 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.
  2. Напишите приложение, которое при нажатии на клавиатуре клавиш ‘1’, ‘2’, ‘3’ или ‘4’ открывает соответственно 1, 2, 3 или 4 новых окна.
  3. Создайте приложение с двумя формами, в котором на первой (главной) форме выводится сообщение «Нажми ‘э’, чтобы увидеть эллипс, ‘к’, чтобы увидеть квадрат, и ‘т’, чтобы увидеть треугольник. Для закрытия приложения нажми *Esc*». При нажатии на клавиши ‘к’, ‘п’ или ‘т’ открывается другая форма, на которой изображена указанная фигура в произвольном месте, произвольного размера, цвета заливки, а также толщины и цвета контура, а в заголовке формы указано название фигуры. При нажатии на клавишу *Esc* по главной форме, приложение закрывается, предварительно выдав сообщение, уточняющее, точно ли пользователь хочет закрыть приложение.

***Баллы:*** каждая из трех последних задача оценивается в один балл.