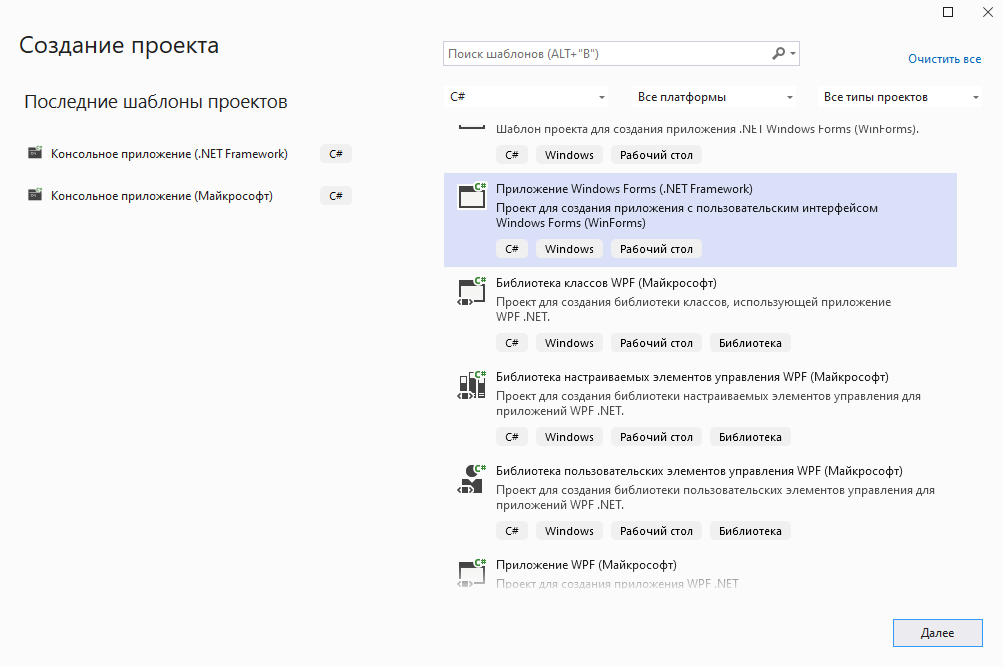
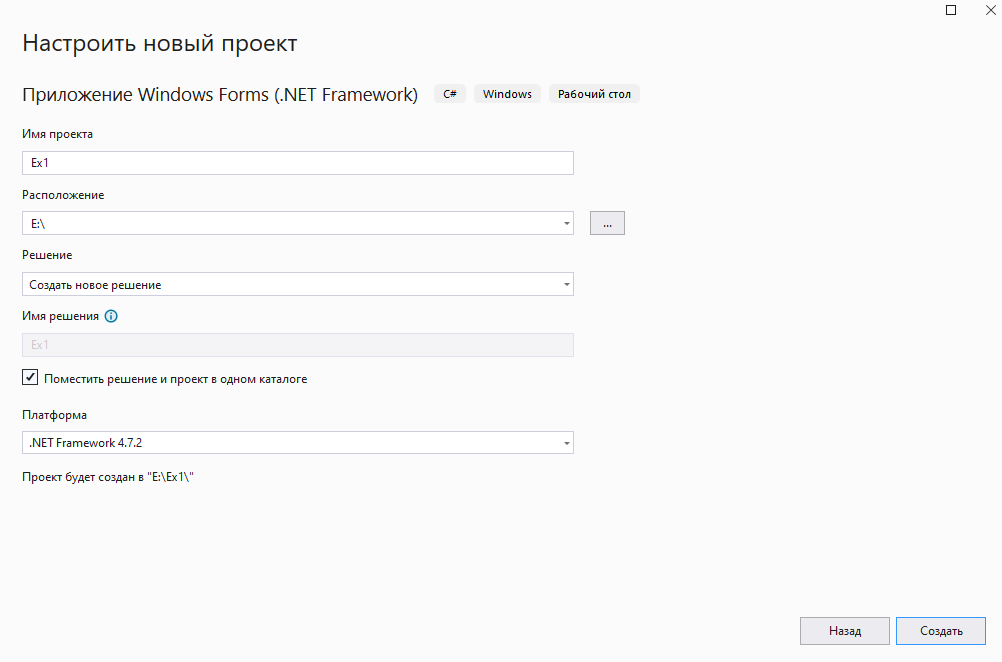
## Лабораторная работа № 2. События мыши

### 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. 2.1)



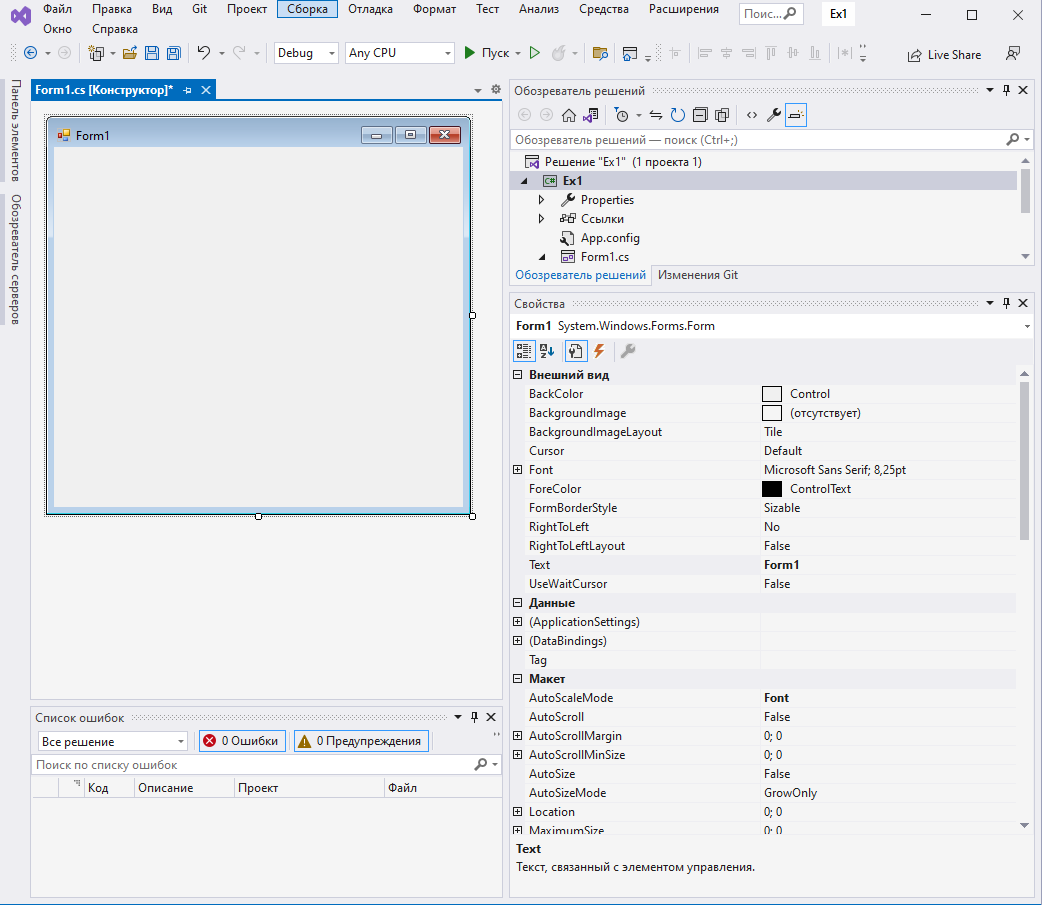


*Рис. 2.1.* Создание нового приложения *Windows Forms*

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 *Lab2*;
* 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 *Создать*.



Форма

Окно свойств

Обозреватель решений

Панель инструментов

*Рис. 2.2.* 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* 2022 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.

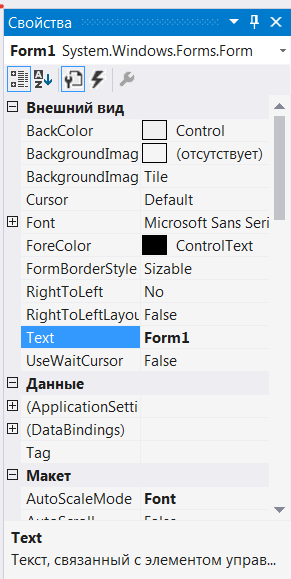
**Визуальный конструктор** – это холст, на котором вы при помощи мыши создаете такие элементы, как формы (посредством перетаскивания, изменения размеров и т. д.). Они позволяют вам создавать элементы, которые составляют ваше приложение. Эти элементы включают: формы *Windows*, *Web*‑формы, диаграммы классов, 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. 2.3 показано окно свойств формы *Form1*.

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

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.



*Рис. 2.3.* Окно Свойств

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  (*Начать отладку*) панели быстрого доступа, или нажать *F*5.

Для закрытия приложения нажмите на крестик в правом верхнем углу запущенной формы , или нажмите <*Alt+F*4> 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***, которыеопределяют 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 *F*1 открывается подсказка, соответствующая текущей ситуации. Например, находясь на странице Свойств, выберите какое‑нибудь свой­ство и нажмите *F*1, 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).

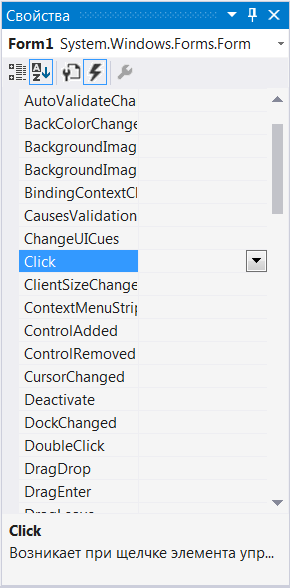
### События мыши

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

**Упражнение 2.1.** Напишите приложение, которое при нажатии мыши перекрашивает форму в синий цвет.

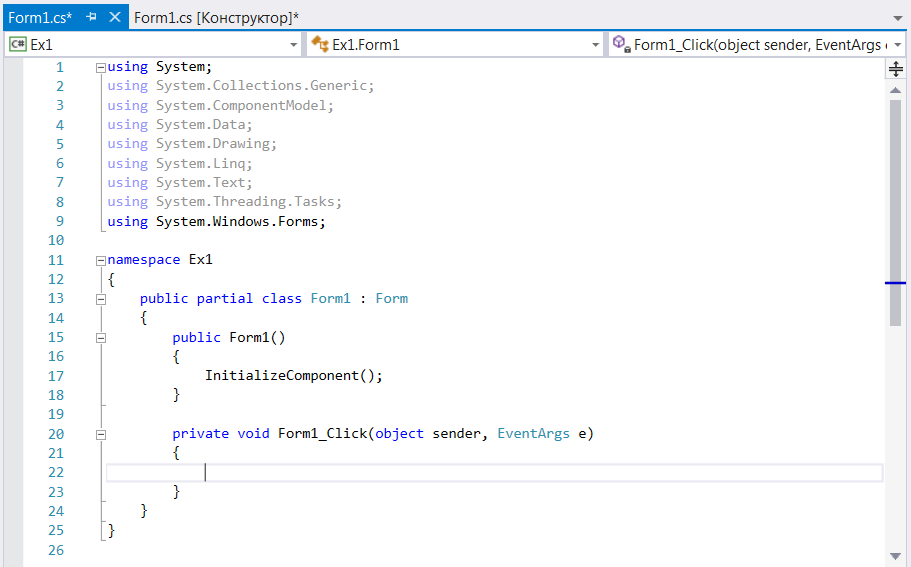
**Решение**

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. 2.4).



*Рис. 2.4.* Создание обработчика

На переднем плане появится окно редактора кода с помещенной сразу в нужное место заготовкой обработчика события Click (рис. 2.5).



Обработчик события Click

*Рис. 2.5.* 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 напишем 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 2.2.** 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 цветовое 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 == MouseButtons.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 2.3.** 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 == MouseButtons.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 2.4.** 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 == MouseButtons.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 *е* позволяет узнать ряд характеристик.

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

if (e.Button == MouseButtons.Left)…

if (e.Button == MouseButtons.Right)…

if (e.Button == MouseButtons.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 2.5.** 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 2.6.** 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 == MouseButtons.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 2.7.** 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 2.8.** 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.

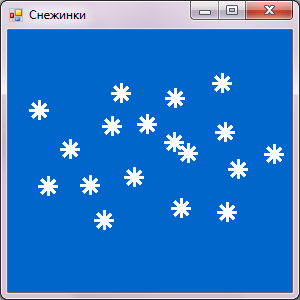
Событие **MouseCaptureChanged** происходит, когда элемент управления теряет или получает захват мыши.

Обычная последовательность обработки некоторых основных событий мыши:

1. MouseEnter.
2. MouseMove.
3. MouseHover или MouseDown.
4. [MouseUp](http://msdn.microsoft.com/ru-ru/library/system.windows.forms.toolstripitem.mouseup.aspx).
5. MouseLeave.

#### Задания для самостоятельного выполнения

* 1. Напишите приложение для рисования разноцветных пунктирных отрезков произвольной толщины.
  2. Напишите приложение для рисования снежинок: установите цвет формы в синий и при щелчке мыши в этом месте должна появляться снежинка белого цвета с восьмью лучиками (рис. 2.6).



*Рис. 2.6.* Пример выполнения

* 1. Напишите приложение для рисования эллипсов с закрашенной внутренней областью (определить кисть *SolidBrush* и использовать метод *FillEllipse*() класса *Graphics*) и разноцветным контуром произвольной толщины.
  2. Создайте приложение для рисования треугольников: когда пользователь щелкает по форме первый раз – запоминается точка, второй – рисуется отрезок, соединяющий первую точку со второй, третий – рисуется еще два отрезка, соединяющих третью точку с первой и со второй. При последующих щелчках рисуется следующий треугольник.
  3. Измените упражнение 2.8 для рисования замкнутых ломаных (при повторном двойном щелчке дорисовывается очередной отрезок ломаной, а также отрезок, соединяющий начальную и конечную точки ломаной).

***Баллы:*** каждое задание оценивается в один балл.