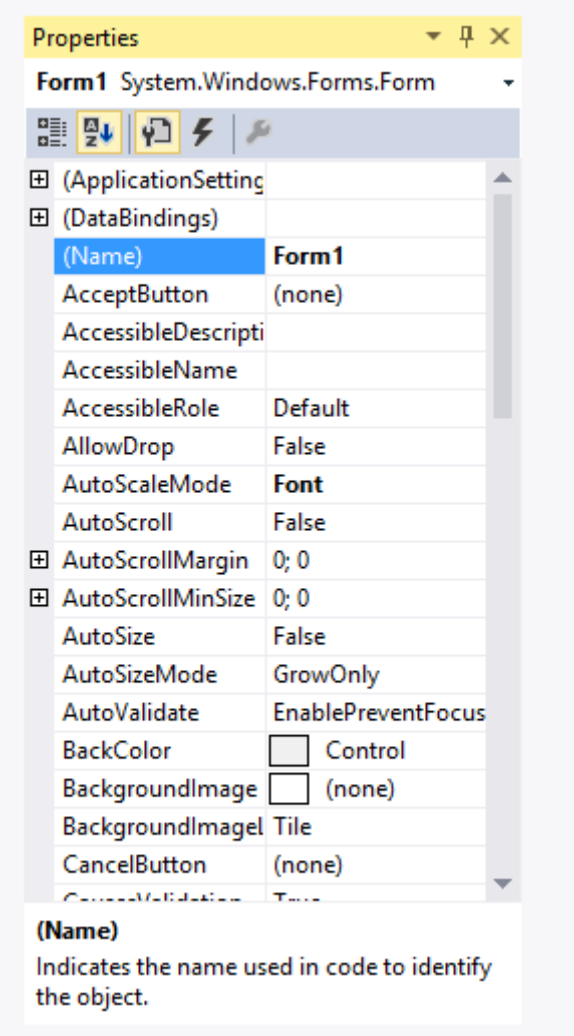
С помощью специального окна Properties (Свойства) справа Visual Studio предоставляет нам удобный интерфейс для управления свойствами элемента:



Большинство этих свойств оказывает влияние на визуальное отображение формы. Пробежимся по основным свойствам:

- **Name**: устанавливает имя формы - точнее имя класса, который наследуется от класса `Form`
- **BackColor**: указывает на фоновый цвет формы. Щелкнув на это свойство, мы сможем выбрать тот цвет, который нам подходит из списка предложенных цветов или цветовой палитры
- **BackgroundImage**: указывает на фоновое изображение формы
- **BackgroundImageLayout**: определяет, как изображение, заданное в свойстве BackgroundImage, будет располагаться на форме.
- **ControlBox**: указывает, отображается ли меню формы. В данном случае под меню понимается меню самого верхнего уровня, где находятся иконка приложения, заголовок формы, а также кнопки минимизации формы и крестик. Если данное свойство имеет значение false, то мы не увидим ни иконку, ни крестика, с помощью которого обычно закрывается форма
- **Cursor**: определяет тип курсора, который используется на форме
- **Enabled**: если данное свойство имеет значение false, то она не сможет получать ввод от пользователя, то есть мы не сможем нажать на кнопки, ввести текст в текстовые поля и т.д.

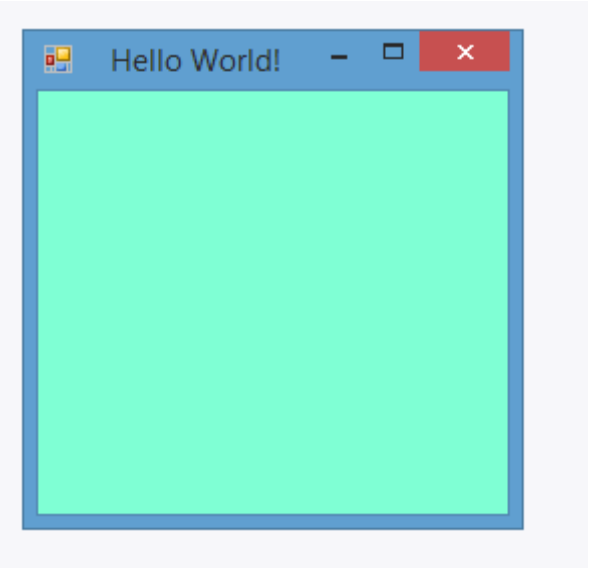
- **Font**: задает шрифт для всей формы и всех помещенных на нее элементов управления. Однако, задав у элементов формы свой шрифт, мы можем тем самым переопределить его
- **ForeColor**: цвет шрифта на форме
- **FormBorderStyle**: указывает, как будет отображаться граница формы и строка заголовка. Устанавливая данное свойство в None можно создавать внешний вид приложения произвольной формы
- **HelpButton**: указывает, отображается ли кнопка справки формы
- **Icon**: задает иконку формы
- **Location**: определяет положение по отношению к верхнему левому углу экрана, если для свойства `StartPosition` установлено значение `Manual`
- **MaximizeBox**: указывает, будет ли доступна кнопка максимизации окна в заголовке формы
- **MinimizeBox**: указывает, будет ли доступна кнопка минимизации окна
- **MaximumSize**: задает максимальный размер формы
- **MinimumSize**: задает минимальный размер формы
- **Opacity**: задает прозрачность формы
- **Size**: определяет начальный размер формы
- **StartPosition**: указывает на начальную позицию, с которой форма появляется на экране
- **Text**: определяет заголовок формы
- **TopMost**: если данное свойство имеет значение `true`, то форма всегда будет находиться поверх других окон
- **Visible**: видима ли форма, если мы хотим скрыть форму от пользователя, то можем задать данному свойству значение `false`
- **WindowState**: указывает, в каком состоянии форма будет находиться при запуске: в нормальном, максимизированном или минимизированном

Программная настройка свойств

С помощью значений свойств в окне Свойства мы можем изменить по своему усмотрению внешний вид формы, но все то же самое мы можем сделать динамически в коде. Перейдем к коду, для этого нажмем правой кнопкой мыши на форме и выберем в появившемся контекстном меню View Code (Просмотр кода). Перед нами открывается файл кода *Form1.cs*. Изменим его следующим образом:

```
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.Linq;
7   using System.Text;
8   using System.Threading.Tasks;
9   using System.Windows.Forms;
10
11  namespace HelloApp
12  {
13      public partial class Form1 : Form
14      {
```

```
15          public Form1()
16          {
17              InitializeComponent();
18              Text = "Hello World!";
19              this.BackColor = Color.Aquamarine;
20              this.Width = 250;
21              this.Height = 250;
22          }
23      }
24  }
```



В данном случае мы настроили несколько свойств отображения формы: заголовок, фоновый цвет, ширину и высоту. При использовании конструктора формы надо учитывать, что весь остальной код должен идти после вызова метода `InitializeComponent()`, поэтому все установки свойств здесь расположены после этого метода.

Установка размеров формы

Для установки размеров формы можно использовать такие свойства как Width/Height или Size. Width/Height принимают числовые значения, как в вышеприведенном примере. При установке размеров через свойство Size, нам надо присвоить свойству объект типа Size:

```
1   this.Size = new Size(200,150);
```

Объект Size в свою очередь принимает в конструкторе числовые значения для установки ширины и высоты.

Начальное расположение формы

Начальное расположение формы устанавливается с помощью свойства `StartPosition`, которое может принимать одно из следующих значений:

- **Manual**: Положение формы определяется свойством Location

- **CenterScreen**: Положение формы в центре экрана
- **WindowsDefaultLocation**: Позиция формы на экране задается системой Windows, а размер определяется свойством Size
- **WindowsDefaultBounds**: Начальная позиция и размер формы на экране задается системой Windows
- **CenterParent**: Положение формы устанавливается в центре родительского окна

Все эти значения содержатся в перечислении `FormStartPosition`, поэтому, чтобы, например, установить форму в центре экрана, нам надо прописать так:

```
this.StartPosition = FormStartPosition.CenterScreen;
```

Фон и цвета формы

Чтобы установить цвет как фона формы, так и шрифта, нам надо использовать цветовое значение, хранящееся в структуре Color:

```
this.BackColor = Color.Aquamarine;
this.ForeColor = Color.Red;
```

Кроме того, мы можем в качестве фона задать изображение в свойстве `BackgroundImage`, выбрав его в окне свойств или в коде, указав путь к изображению:

```
this.BackgroundImage = Image.FromFile("C:\\Users\\Eugene\\Pictures\\3332.jpg");
```

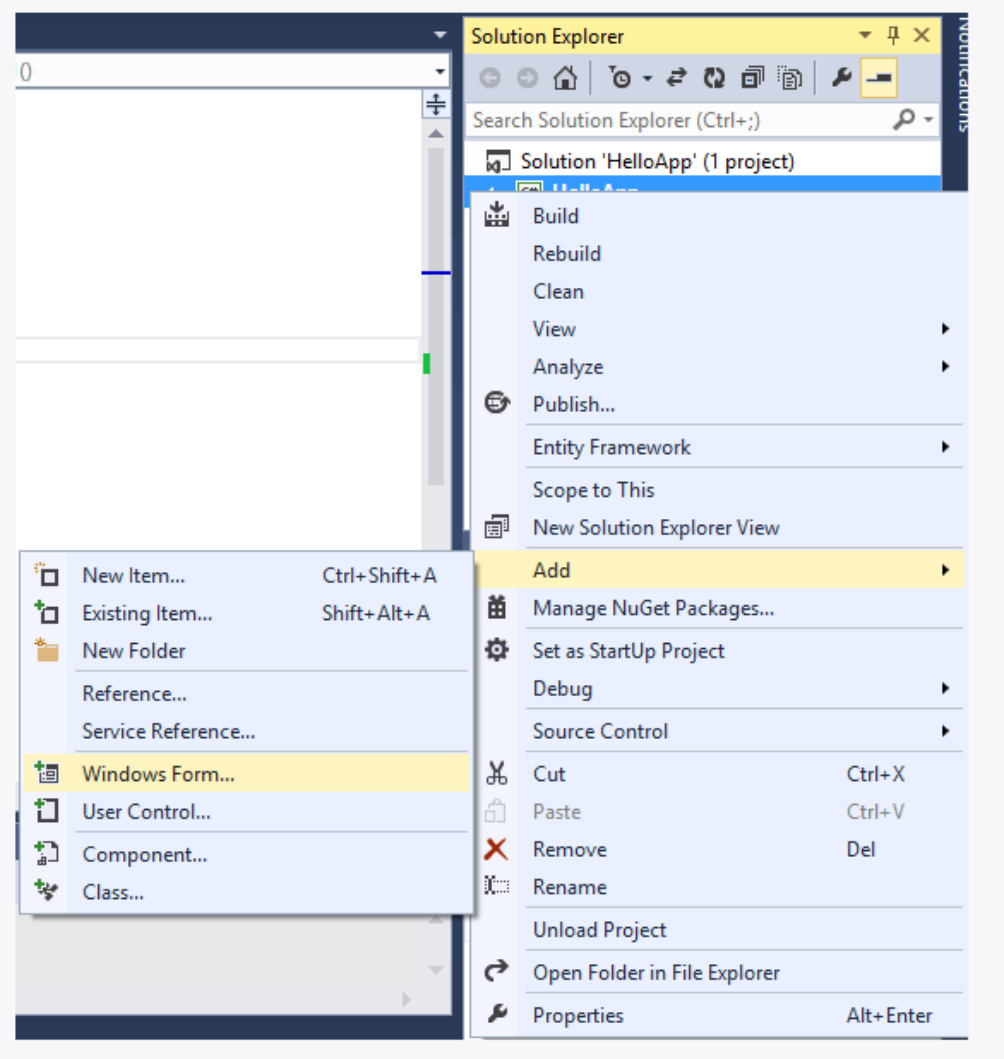
Чтобы должным образом настроить нужное нам отображение фоновой картинки, надо использовать свойство `BackgroundImageLayout`, которое может принимать одно из следующих значений:

- **None**: Изображение помещается в верхнем левом углу формы и сохраняет свои первоначальные значения
- **Tile**: Изображение располагается на форме в виде мозаики
- **Center**: Изображение располагается по центру формы
- **Stretch**: Изображение растягивается до размеров формы без сохранения пропорций
- **Zoom**: Изображение растягивается до размеров формы с сохранением пропорций

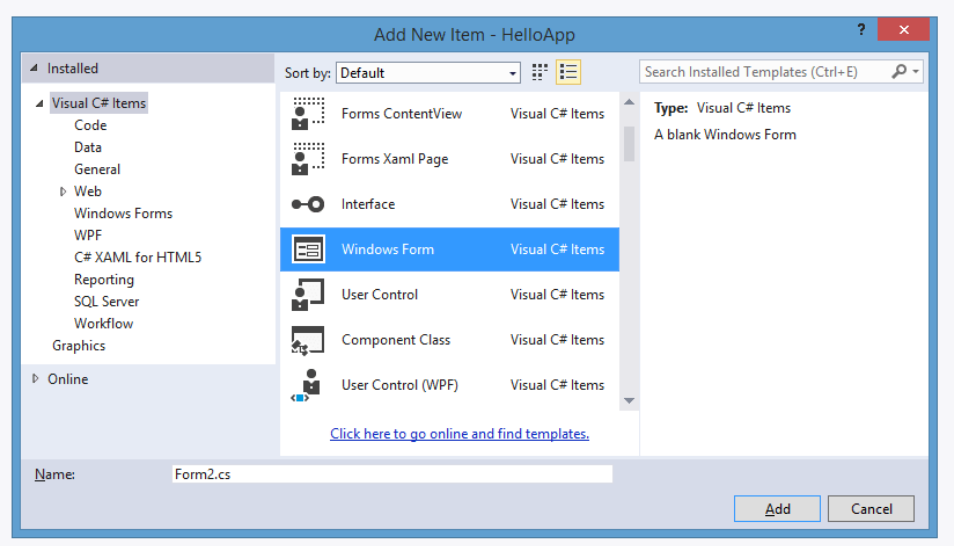
Например, расположим форму по центру экрана:

```
this.StartPosition = FormStartPosition.CenterScreen;
```

Чтобы добавить еще одну форму в проект, нажмем на имя проекта в окне Solution Explorer (Обозреватель решений) правой кнопкой мыши и выберем Add(Добавить)->Windows Form...



Дадим новой форме какое-нибудь имя, например, *Form2.cs*:



Итак, у нас в проект была добавлена вторая форма. Теперь попробуем осуществить взаимодействие между двумя формами. Допустим, первая форма по нажатию на кнопку будет вызывать вторую форму. Во-первых, добавим на первую форму Form1 кнопку и двойным щелчком по кнопке перейдем в файл кода. Итак, мы попадем в обработчик события нажатия кнопки, который создается по умолчанию после двойного щелчка по кнопке:

```
1   private void button1_Click(object sender, EventArgs e)
2   {
3
4   }
```

Теперь добавим в него код вызова второй формы. У нас вторая форма называется Form2, поэтому сначала мы создаем объект данного класса, а потом для его отображения на экране вызываем метод Show:

```
1   private void button1_Click(object sender, EventArgs e)
2   {
3       Form2 newForm = new Form2();
4       newForm.Show();
5   }
```

Теперь сделаем наоборот - чтобы вторая форма воздействовала на первую. Пока вторая форма не знает о существовании первой. Чтобы это исправить, надо второй форме как-то передать сведения о первой форме. Для этого воспользуемся передачей ссылки на форму в конструкторе.

Итак перейдем ко второй форме и перейдем к ее коду - нажмем правой кнопкой мыши на форму и выберем View Code (Просмотр кода). Пока он пустой и содержит только конструктор. Поскольку C# поддерживает перегрузку методов, то мы можем создать несколько методов и конструкторов с разными параметрами и в зависимости от ситуации вызывать один из них. Итак, изменим файл кода второй формы на следующий:

```
1    using System;
2    using System.Collections.Generic;
3    using System.ComponentModel;
4    using System.Data;
5    using System.Drawing;
6    using System.Linq;
7    using System.Text;
8    using System.Threading.Tasks;
9    using System.Windows.Forms;
10
11   namespace HelloApp
12   {
13       public partial class Form2 : Form
14       {
15           public Form2()
16           {
```

```
            InitializeComponent();
        }

        public Form2(Form1 f)
        {
            InitializeComponent();
            f.BackColor = Color.Yellow;
        }
    }
}
```

Фактически мы только добавили здесь новый конструктор `public Form2(Form1 f)`, в котором мы получаем первую форму и устанавливаем ее фон в желтый цвет. Теперь перейдем к коду первой формы, где мы вызывали вторую форму и изменим его на следующий:

```
private void button1_Click(object sender, EventArgs e)
{
    Form2 newForm = new Form2(this);
    newForm.Show();
}
```

Поскольку в данном случае ключевое слово this представляет ссылку на текущий объект - объект Form1, то при создании второй формы она будет получать ее (ссылку) и через нее управлять первой формой.

Теперь после нажатия на кнопку у нас будет создана вторая форма, которая сразу изменит цвет первой формы.

Мы можем также создавать объекты и текущей формы:

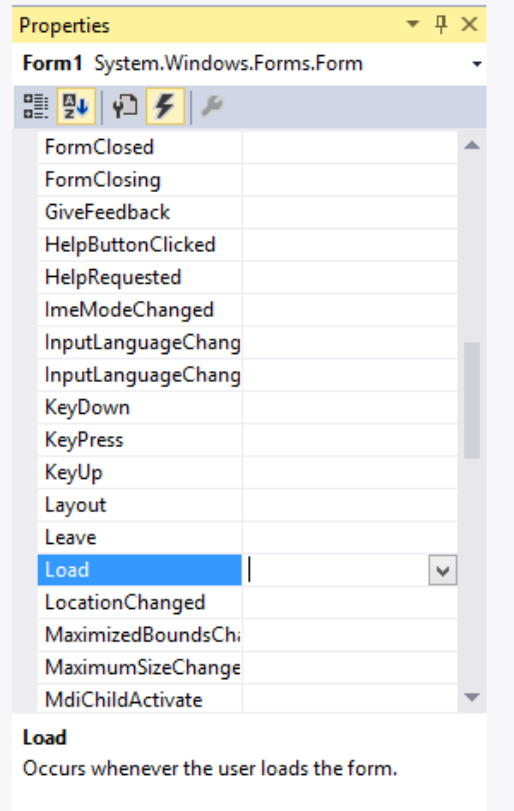
```
private void button1_Click(object sender, EventArgs e)
{
    Form1 newForm1 = new Form1();
    newForm1.Show();

    Form2 newForm2 = new Form2(newForm1);
    newForm2.Show();
}
```

При работе с несколькими формами надо учитывать, что одна из них является главной - которая запускается первой в файле Program.cs. Если у нас одновременно открыта куча форм, то при закрытии главной закрывается все приложение и вместе с ним все остальные формы.

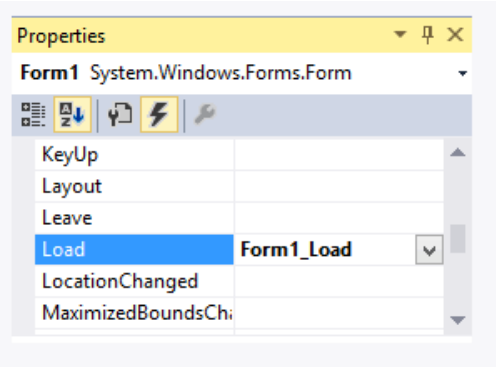
События формы

Для взаимодействия с пользователем в Windows Forms используется механизм событий. События в Windows Forms представляют стандартные события на C#, только применяемые к визуальным компонентам и подчиняются тем же правилам, что события в C#. Но создание обработчиков событий в Windows Forms все же имеет некоторые особенности.

Прежде всего в WinForms есть некоторый стандартный набор событий, который по большей части имеется у всех визуальных компонентов. Отдельные элементы добавляют свои события, но принципы работы с ними будут похожие. Чтобы посмотреть все события элемента, нам надо выбрать этот элемент в поле графического дизайнера и перейти к вкладке событий на панели форм. Например, события формы:



Чтобы добавить обработчик, можно просто два раза нажать по пустому полю рядом с названием события, и после этого Visual Studio автоматически сгенерирует обработчик события. Например, нажмем для создания обработчика для события `Load`:



И в этом поле отобразится название метода обработчика события Load. По умолчанию он называется `Form1_Load`.

Если мы перейдем в файл кода формы Form1.cs, то увидим автосгенерированный метод Form1_Load:

```
1   public partial class Form1 : Form
2   {
3       public Form1()
4       {
5           InitializeComponent();
6       }
7
8       private void Form1_Load(object sender, EventArgs e)
9       {
10
11      }
12  }
```

И при каждой загрузке формы будет срабатывать код в обработчике Form1_Load.

Как правило, большинство обработчиков различных визуальных компонентов имеют два параметра: `sender` - объект, инициировавший событие, и аргумент, хранящий информацию о событии (в данном случае `EventArgs e`).

Но это только обработчик. Добавление же обработчика, созданного таким образом, производится в файле Form1.Designer.cs:

```
1   namespace HelloApp
2   {
3       partial class Form1
4       {
5           private System.ComponentModel.IContainer components = null;
6
7           protected override void Dispose(bool disposing)
8           {
9               if (disposing && (components != null))
10              {
```

```
                components.Dispose();
            }
            base.Dispose(disposing);
        }
        private void InitializeComponent()
        {
            this.SuspendLayout();

            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(284, 261);
            this.Name = "Form1";
            // добавление обработчика
            this.Load += new System.EventHandler(this.Form1_Load);
            this.ResumeLayout(false);
        }
    }
}
```

Для добавления обработчика используется стандартный синтаксис C#: `this.Load += new System.EventHandler(this.Form1_Load)`

Поэтому если мы захотим удалить созданный подобным образом обработчик, то нам надо не только удалить метод из кода формы в Form1.cs, но и удалить добавление обработчика в этом файле.

Однако мы можем добавлять обработчики событий и програмно, например, в конструкторе формы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace HelloApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.Load += LoadEvent;
```

```
19          }
20
21          private void Form1_Load(object sender, EventArgs e)
22          {
23          }
24
25          private void LoadEvent(object sender, EventArgs e)
26          {
27              this.BackColor = Color.Yellow;
28          }
29      }
30  }
```

Кроме ранее созданного обработчика Form1_Load здесь также добавлен другой обработчик загрузки формы: `this.Load += LoadEvent;`, который устанавливает в качестве фона желтый цвет.

Создание непрямоугольной формы

По умолчанию все формы в Windows Forms являются прямоугольными. Однако мы можем создавать и непрямоугольные произвольные формы. Для этого используется свойство **Region**. В качестве значения оно принимает объект одноименного класса Region.

При создании непрямоугольных форм, как правило, не используются границы формы, так как границы задаются этим объектом Region. Чтобы убрать границы формы, надо присвоить у формы свойству `FormBorderStyle` значение `None`.

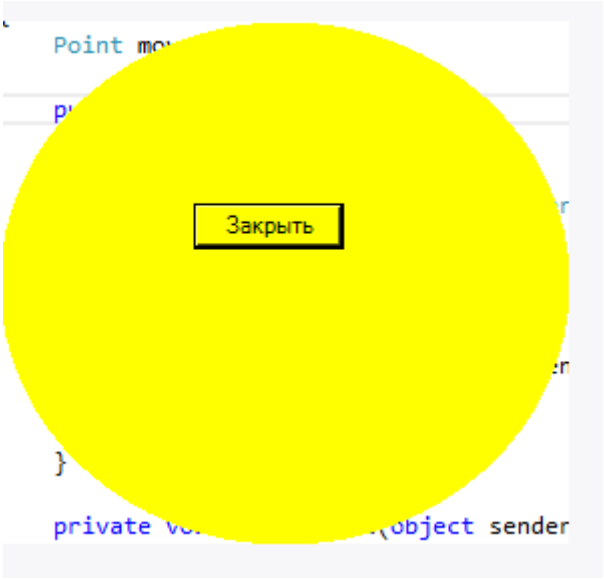
И еще один аспект, который надо учитывать, заключается в перемещении, закрытии, максимизации и минимизации форм. То есть в данном случае , как в обычной форме, мы не сможем нажать на крестик, чтобы закрыть форму, не сможем ее переместить на новое место. Поэтому нам надо дополнительно определять для этого программную логику.

Итак, перейдем к коду формы и изменим его следующим образом:

```
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.Linq;
7   using System.Text;
8   using System.Threading.Tasks;
9   using System.Windows.Forms;
10
11  namespace HelloApp
12  {
13      public partial class Form1 : Form
```

```
14      {
15          Point moveStart; // точка для перемещения
16
17          public Form1()
18          {
19              InitializeComponent();
20              this.FormBorderStyle = FormBorderStyle.None;
21              this.BackColor = Color.Yellow;
22              Button button1 = new Button
23              {
24                  Location = new Point
25                  {
26                      X = this.Width / 3,
27                      Y = this.Height / 3
28                  }
29              };
30              button1.Text = "Закрыть";
31              button1.Click += button1_Click;
32              this.Controls.Add(button1); // добавляем кнопку на форму
33              this.Load += Form1_Load;
34              this.MouseDown += Form1_MouseDown;
35              this.MouseMove += Form1_MouseMove;
36          }
37
38          private void button1_Click(object sender, EventArgs e)
39          {
40              this.Close();
41          }
42
43          private void Form1_Load(object sender, EventArgs e)
44          {
45              System.Drawing.Drawing2D.GraphicsPath myPath = new System.Drawing.Dra
46              // создаем эллипс с высотой и шириной формы
47              myPath.AddEllipse(0, 0, this.Width, this.Height);
48              // создаем с помощью элипса ту область формы, которую мы хотим видет
49              Region myRegion = new Region(myPath);
50              // устанавливаем видимую область
51              this.Region = myRegion;
52          }
53
54          private void Form1_MouseDown(object sender, MouseEventArgs e)
55          {
56              // если нажата левая кнопка мыши
57              if (e.Button == MouseButtons.Left)
58              {
59                  moveStart = new Point(e.X, e.Y);
```

```
60              }
61          }
62
63          private void Form1_MouseMove(object sender, MouseEventArgs e)
64          {
65              // если нажата левая кнопка мыши
66              if ((e.Button & MouseButtons.Left) != 0)
67              {
68                  // получаем новую точку положения формы
69                  Point deltaPos = new Point(e.X - moveStart.X, e.Y - moveStart.Y);
70                  // устанавливаем положение формы
71                  this.Location = new Point(this.Location.X + deltaPos.X,
72                    this.Location.Y + deltaPos.Y);
73              }
74          }
75      }
76  }
```



Создание области формы происходит в обработчике события Form1_Load. Для создания области используется графический путь - объект класса `System.Drawing.Drawing2D.GraphicsPath`, в который добавляется эллипс. Графический путь позволяет создать фигуру любой формы, поэтому, если мы захотим форму в виде морской звезды, то нам просто надо должным образом настроить используемый графический путь.

Для закрытия формы в обработчике события нажатия кнопки `button1_Click` форма закрывается программным образом: `this.Close()`

Для перемещения формы обрабатываются два события формы - событие нажатия кнопки мыши и событие перемещения указателя мыши.