Часто задаваемые вопросы при выполнении курсового проекта (Бизнес-информатика, 1 курс)

Оглавление

Как создать проект Windows-приложения?

При создании проекта для Windows-приложения нужно выбрать специальный тип проекта, который называется «Windows Application» (Visual Studio 2005), «Приложение Windows Forms» (Visual Studio 2010) или аналогичным образом в других версиях программы.

В состав проекта будут входить два основных файла с кодом Program.cs (в нем определена функция Main и происходит создание и запуск главного окна) и Form1.cs (в котором определяется класс главного окна приложения).

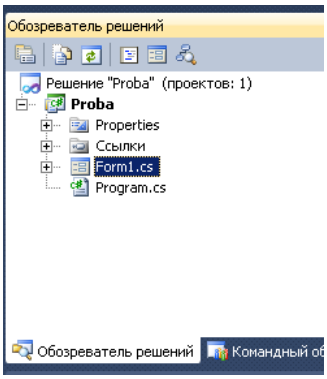


Рис.1. Окно файлов проекта.

Сгенерированное приложение можно запустить.

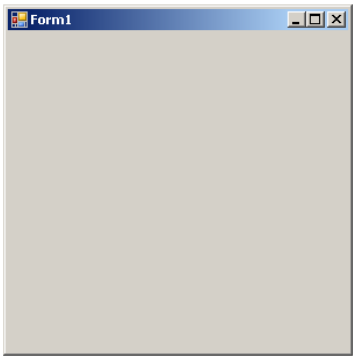


Рис.2. Окно простого Windows-приложения.

Данное окно может изменять размер, сворачиваться и разворачиваться, максимизироваться на весь экран, перемещаться и закрываться, как и любые окна в операционных системах семейства Windows. Приложение, реализующее подобное окно, называется каркасом Windows-приложения.

Как изменить заголовок окна?

Любая форма представляет собой с точки зрения программиста объект, который обладает множеством свойств. Каждое свойство можно задать, как программным способом (например, в специальной функции-конструкторе объекта-окна, которая называется как класс окна):

```
public partial class Form1 : Form
{
        public Form1()   // метод – конструктор объекта-окна
        {
                InitializeComponent();
                // задание свойства-заголовка окна
                Text = "Новый заголовок окна";
         }
 }
```

Другим способом задания свойств является использования средств оболочки проектирования Visual Studio 2010 (или др.версии). Для этого требуется:

- Открыть форму в режиме конструктора:

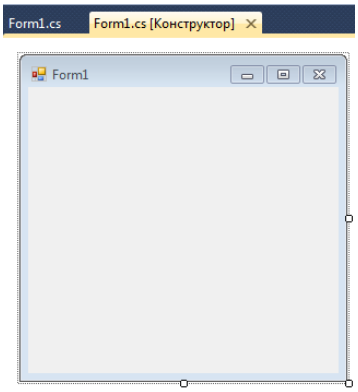


Рис.3. Окно конструктора формы.

- Выбрать форму и вызвать контекстное меню (правая кнопка мыши), выбрать пункт «Свойства»(Properties). Появится окно, в котором перечисляются все свойства окна и их значения:

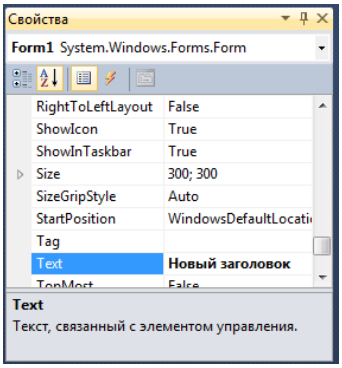


Рис 4. Окно свойств формы – задание свойства Text.

- Далее следует выбрать нужное свойство (заголовок задается с помощью свойства с именем Text) и ввести его значение, например, «Новый заголовок».

Как изменить цвет фона у окна?

Цвет фона у окна также является свойством окна с названием BackColor:

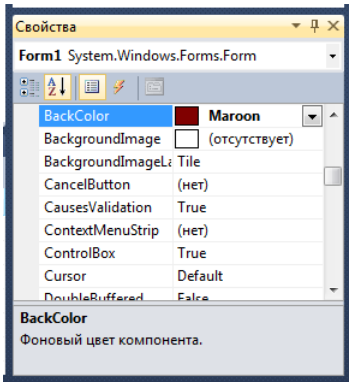


Рис 5. Окно свойств формы – задание свойства BackColor.

Цвет с помощью окна свойств можно задать из системных цветов, именованных цветов и палитры цветов. При определении цвета из палитры цвет характеризуют три целых числа в интервале от 0 до 255, которые определяют количество компонент красного, зеленого и синего цвета, которые используются при смешивании выбранного цвета (модель RGB).

Как назначить какие-то действия на щелчок мыши?

Основной принцип Windows-программирования – настройка внешнего вида используемых окон и определение, в какие момент должны выполняться те или иные действия программы. Такой стиль программирования называют обработкой событий. Классы, которые предоставляют события, позволяют связать событие с функцией, которая будет вызвана при возникновении события. Например, пользователь щелкнул по форме – произошло событие щелчка мыши (MouseClick). Если был назначен обработчик на этой событие, он будет вызван и будут выполнены все действия этой функции.
Назначить обработчик можно в программе:

```
// MouseClick – название события формы,
// MyMouseClick – название функции-обработчика события
MouseClick += MyMouseClick;
```

Удобнее задавать обработчики с помощью среды разработки Visual Studio 2010 (или др.версий). В окне свойств формы следует выбрать кнопку на панели инструментов с молнией. Тогда в окне свойств будут отображаться только события, которые могут происходить с формой или другим элементом окна:

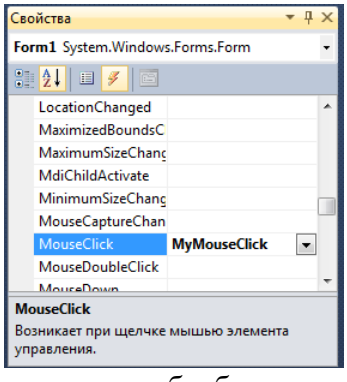


Рис 6. Окно свойств – задание обработчика события MouseClick.

Система сгенерирует функцию-обработчик с тем именем, которое будет введено программистом (например, MyMouseClick), или сама назовет этот обработчик по схеме «ИмяФормы_ИмяСобытия». Программисту остается только внести программный код, который будет выполняться при наступлении события. Например, пусть при щелчке мыши в форме должно показаться окно с некоторым сообщением.

```
// текст обработчика
private void MyMouseClick(object sender, MouseEventArgs e)
{
        MessageBox.Show("Сообщение: Вы нажали кнопку мыши");
}
```

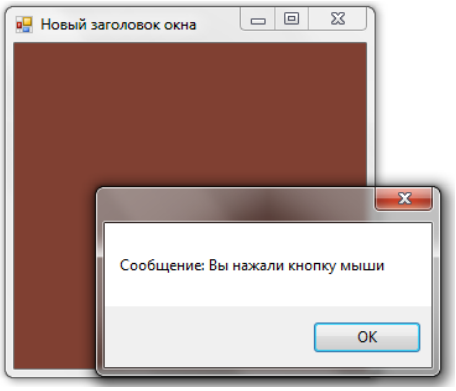


Рис 7. Окно с обработчиком щелчка мыши.

Как понять, каковы были координаты мыши в момент щелчка по ее кнопке?

Большинство обработчиков событий в классах Windows.Forms (System.Window.Forms – пространство имен, в котором определены большинство классов создания окон в стиле операционной среды Windows) имеют стандартный набор параметров. Разберем пример из предыдущего вопроса:

```
// текст обработчика
private void MyMouseClick(object sender, MouseEventArgs e)
{
        MessageBox.Show("Сообщение: Вы нажали кнопку мыши");
}
```

Все функции-обработчики событий не имеют возвращаемого значения (тип функции void) и имеют два параметра. Первый параметр – ссылка на тот объект, который вызвал событие (например, нажали кнопку, тогда sender – ссылка на объект-кнопку), второй параметр – это специальный объект, через который передается дополнительная информация о произошедшем событии. В случае событий мыши этот объект имеет тип MouseEventArgs и, в частности, содержит координаты мыши в момент ее щелчка. Изменим обработчик, чтобы в сообщении выводились координаты мыши:

```
private void MyMouseClick(object sender, MouseEventArgs e)
{
    MessageBox.Show("Сообщение: Вы нажали кнопку мыши. Координаты мыши:
                                                    ("+e.X+"; "+e.Y+")");
}
```

Координаты мыши определяются в пределах окна – за начало координат принимается левый верхний угол клиентской области окна (без заголовка). Ось X направлена вправо, ось Y – вниз.

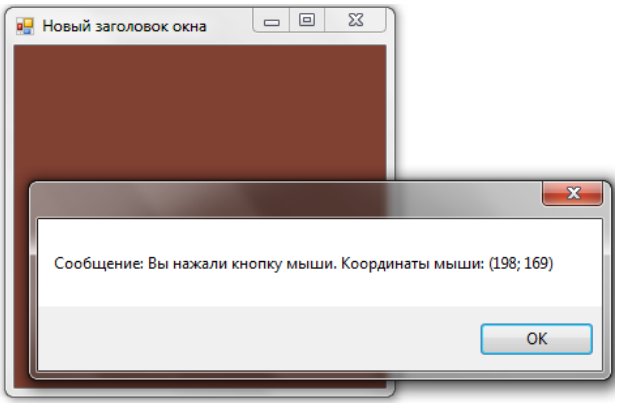


Рис 8. Окно с обработчиком щелчка мыши и показом координат щелчка.

Как понять, какая клавиша на клавиатуре была нажата?

Для получения сообщений о событиях клавиатуры предназначено событие формы OnKeyDown. Вместе с сообщением о возникновении события и вызовом функции-обработчика передается информация о коде нажатой клавиши, дополнительно нажатых клавишах Control, Shift, Alt и пр. Узнать, какая клавиша была нажата, удобно с помощью дополнительной информации KeyCode, значение которого – значение из перечисления Keys, в котором каждой из клавиш задано символьное обозначение.

Добавим обработчик события нажатия клавиши:

```
private void MyKeyDown(object sender, KeyEventArgs e)
{
    MessageBox.Show("Нажата клавиша " + e.KeyCode);
}
```

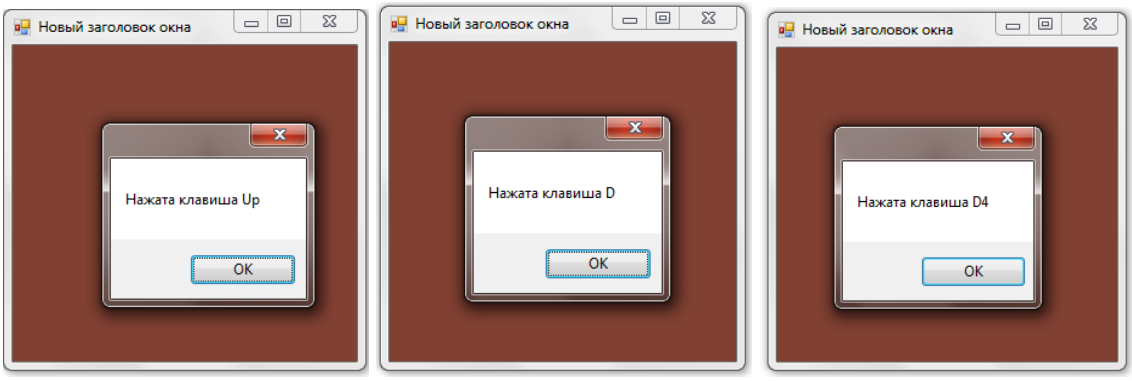


Рис 9. Окно с обработчиком нажатия клавиши клавиатуры (клавиши «Вверх», «D» и «8»).

Что такое перо (Pen)?

Перо (Pen) – это один из инструментов рисования в окне приложения. Перо применяется при рисовании линий и контуров фигур. Классы, с помощью которых

производится рисование, в частности, класс Pen, определены в пространстве имен System.Drawing. При создании пера задают его цвет и толщину (в пикселях (точках)). Например, создание пера голубого цвета и толщиной 2:

```
Pen p = new Pen(Color.Aqua, 2);
```

Подпространство имен System.Drawing.Drawing2D содержит дополнительно средства для задания особых стилей пера, например, пунктирного пера:

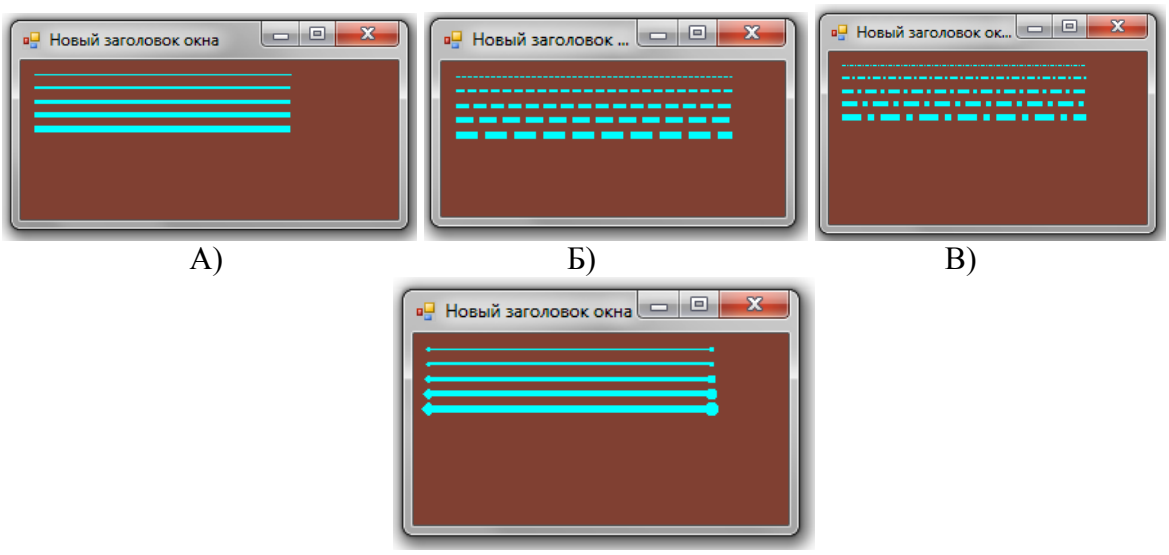
```
Pen p = new Pen(Color.Aqua, 2);
p.DashStyle = DashStyle.Dash;
```

или штрих-пунктирного пера:

```
Pen p = new Pen(Color.Aqua, 2);
p.DashStyle = DashStyle.DashDot;
```

или с заданием специального «наконечника» линий:

```
Pen p = new Pen(Color.Aqua, i);
// начало линии – наконечник в виде ромба, конец линии - круг
p.StartCap = LineCap.DiamondAnchor;
p.EndCap = LineCap.RoundAnchor;
```



А) Б) В)

Г)

Рис 10. Использование пера (А – разная толщина пера, Б – разная толщина в стиле «пунктир», В – разная толщина пера в стиле «штрих-пунктир», Г – разные «наконечники»).

Можно воспользоваться «системными перьями», которые определены в перечислении Pens. Название перьев в перечислении определяются названием цвета пера, например, Pens.Red.

Что такое кисть (Brush)?

Кисть (Brush) – это еще один инструмент рисования, с помощью которого происходит заливка (закраска) фигур. Как и для перьев, существует перечисление «системных кистей» Brushes, название кисти в котором определяется названием цвета. Так

определяются «сплошные» кисти, т.е. кисти, производящие закраску сплошным цветом. Сплошную кисть можно создать с помощью класса SolidBrush. При создании требуется задать только цвет.

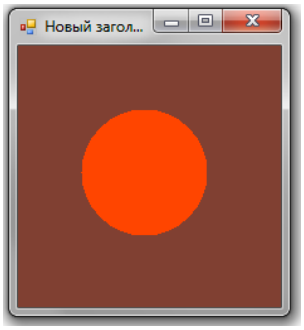


Рис 11. Использование стандартной кисти Brushes.OrangeRed.

Как сделать нестандартную закраску фигуры (картинкой, переливающимся цветом и пр.)?

В пространстве имен System.Drawing.Drawing2D есть еще несколько классов для создания кисти, которые используют узор, градиентную заливку, изображение.
Например, кисть с узором (HatchBrush) создается с указанием вида узора (перечисление HatchStyle), цвета узора и цвета фона:

```
HatchBrush br = new HatchBrush(HatchStyle.Cross,Color.Red,Color.Yellow);
```

Здесь была задана кисть, которая на желтом фоне красным цветом рисует узор «решетка».

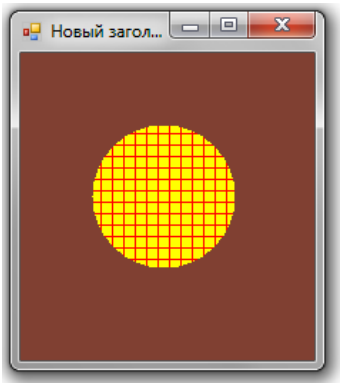


Рис 12. Использование кисти-узора.

Градиентные кисти (класс LinearGradientBrush) задают «перелив» из одного цвета в другой. При создании такой кисти задается прямоугольник, который в памяти заполняется указанными цветами. Кроме двух цветов, указывается направление «перелива» (значение из перечисленияLinearGradientMode). Например, в следующем примере создается кисть с «переливом» от желтого цвета к синему по главной диагонали.

```
Rectangle r = new Rectangle(0,0,300,300);
LinearGradientBrush br = new LinearGradientBrush(r, Color.Yellow,
                             Color.Blue,LinearGradientMode.ForwardDiagonal);
```

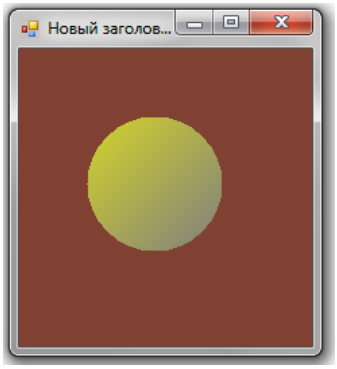


Рис 13. Использование градиентной кисти.

Текстурные кисти (класс TextureBrush) задают заливку с помощью изображения. Для создания такой кисти требуется указать изображение, например, загруженное из файла.

```
TextureBrush br = new TextureBrush(
      new Bitmap("C:\\Users\\Public\\Pictures\\Sample Pictures\\Chrysanthemum.jpg"));
```

Рис 14. Использование текстурной кисти.

Как задать цвет?

При создании перьев и кистей, а также задании свойств фона, текста и др. может потребоваться указать цвет. За хранение цвета отвечает структура Color. Создать объект этой структуры можно следующими способами:

- Выбрать из перечисления;

```
Color c = Color.Yellow;
```

- Создать цвет по его имени:

```
Color c = Color.FromName("Red");
```

- Создать цвет по трем составляющим модели RGB (количество красного, зеленого и синего цвета):

```
Color c = Color.FromArgb(200,100,0);
```

Далее можно будет обратиться к свойствам объекта для получения и имени цвета (Name), и каждого из компонентов цвета (R – красный цвет, G – зеленый цвет, B – синий цвет).

Как изменить цвет фона окна во время работы программы?

Большинство свойств окна являются доступными в любой функции, определенной для окна по имени свойства. Для цвета фона таким свойством является переменная BackColor. Например, пусть при щелчке мыши в форме цвет формы изменяется случайным образом (генерируется случайный цвет и устанавливается в качестве цвета фона окна). Также изменим заголовок окна (свойство Text), в котором укажем параметры нового цвета окна. Обработчик события щелчка мыши в этом случае будет таким:

```
  private void Form1_MouseClick(object sender, MouseEventArgs e)
{
    Random r = new Random();
    // изменение цвета фона окна
    BackColor = Color.FromArgb(r.Next(255), r.Next(255), r.Next(255));
    // показ компонентов цвета в заголовке окна
    Text = BackColor.ToString();
  }
```

Как в окне нарисовать прямоугольник?

В основном, собственно рисование происходит в обработчике события Paint формы, хотя это не обязательно (иногда рисование происходят в других обработчиках). Рисование происходит с помощью класса Graphics, который иногда называют контекстом устройства. Этот класс содержит большой набор функций для рисования графических примитивов (прямоугольников, эллипсов и др.фигур). В частности, для рисования прямоугольника нужно использовать функцию DrawRectangle(), в которой следует задать перо, координаты левого верхнего угла прямоугольника и его размеры (ширину и высоту).

```
 private void Form1_Paint(object sender, PaintEventArgs e)
{
    e.Graphics.DrawRectangle(Pens.Azure, 50, 50, 100, 70);
}
```

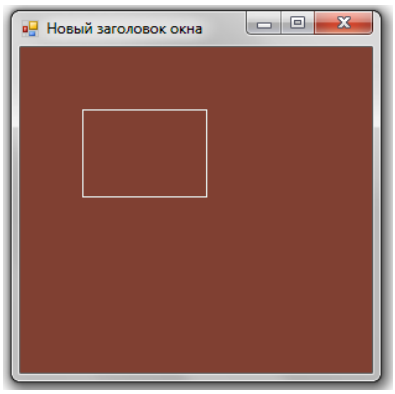


Рис 15. Рисование прямоугольника.

Заметим, что если рисование происходит в обработчике события Paint, то контекст устройства будет передан в данный обработчик вместе с дополнительной информацией в параметрах обработчика. Если рисование происходит в других функциях, то объект класса Graphics следует создать, например, следующим образом:

```
Graphics g = Graphics.FromHwnd(this.Handle);
```

Как в окне нарисовать закрашенный прямоугольник?

Рисование закрашенных фигур также происходит с помощью контекста устройства, но с помощью функций, которые начинаются со слова Fill. Эти функции должны указывать параметры местоположения фигуры, а также кисть для закраски фигуры. В частности, рисование закрашенного прямоугольника осуществляется с помощью функции FillRactangle():

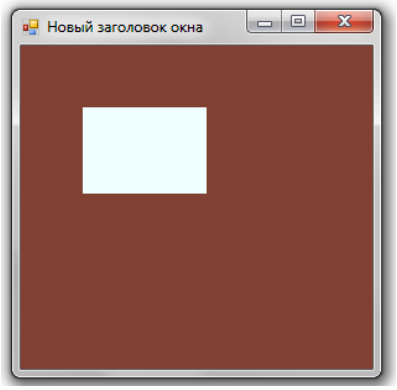


Рис 16. Рисование закрашенного прямоугольника.

Какие фигуры, кроме прямоугольника, можно рисовать?

Фигуры, которые можно нарисовать с помощью контекста устройства, называют графическими примитивами. Вот только несколько примеров:

- Эллипсы – для задания эллипса требуется указать параметры прямоугольника, в который вписан эллипс:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    e.Graphics.FillEllipse(Brushes.Azure, 50, 50, 100, 70);
}
```

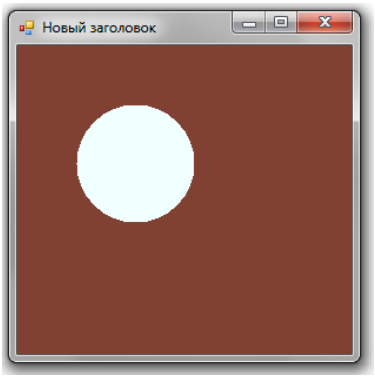


Рис 17. Рисование эллипса.

- Многоугольник задается с помощью массива точек, которые являются его вершинами:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
       Point [] pp={new Point(10,10), new Point(50,100),
                    new Point(200,50), new Point(20,150)};
        e.Graphics.FillPolygon(Brushes.MistyRose, pp);
 }
```

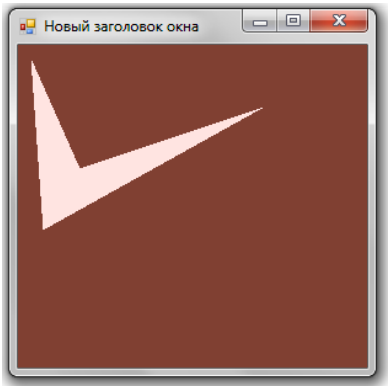


Рис 18. Рисование многоугольника.

- Когда точки соединяются не прямыми линиями, а «кривыми», получается фигура, которую называют «Замкнутая кривая»:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
   Point [] pp={new Point(10,10), new Point(50,100),
                new Point(200,50), new Point(20,150)};
   e.Graphics.FillClosedCurve(Brushes.OldLace, pp);
}
```

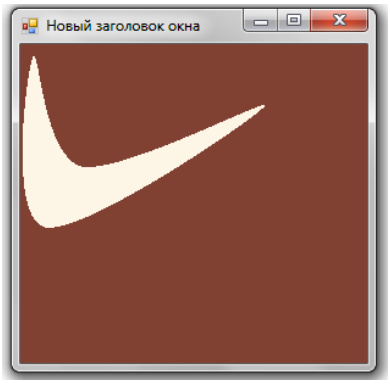


Рис 19. Рисование замкнутой кривой.

- Сектор задается параметрами эллипса, от которого он «отрезается», углом начальной радиальной линии и углом, определяющим сектор.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
     e.Graphics.FillPie(Brushes.Moccasin, 50, 50, 100, 100,30, 90);
}
```

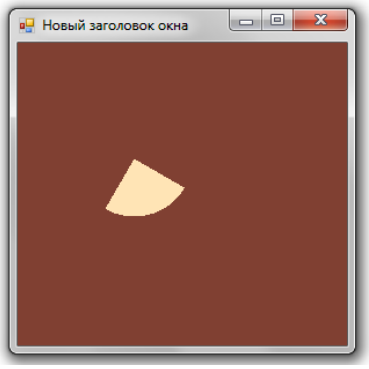


Рис 20. Рисование сектора.

- Набор прямоугольников.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Rectangle [] rr={new Rectangle(10,10,50,50),
                     new Rectangle(30,70,100,10), new Rectangle(200,50,20,20)};
    e.Graphics.FillRectangles(Brushes.MintCream, rr);
}
```

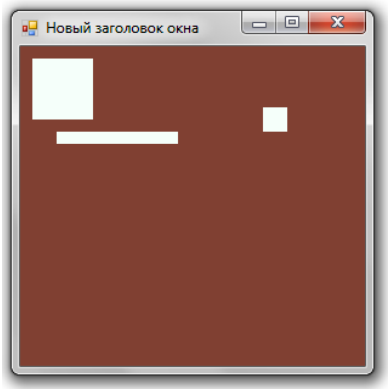


Рис 21. Рисование набора прямоугольников.

- Аналогичные функции существуют и для рисования контуров фигур. Отдельно выделим функцию рисования дуги эллипса (определяется аналогично сектору):

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    e.Graphics.DrawArc(Pens.Azure,10,10,100,100,30,90);
}
```

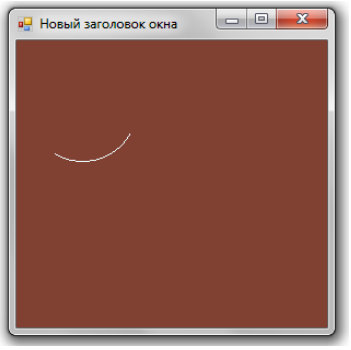


Рис 22. Рисование дуги.

Как нарисовать линию?

Линия также является графическим примитивом. Чтобы нарисовать линию, требуется определить две точки (начало и конец линии) и перо.

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
    e.Graphics.DrawLine(Pens.Azure,10,10,100,100);
 }
```

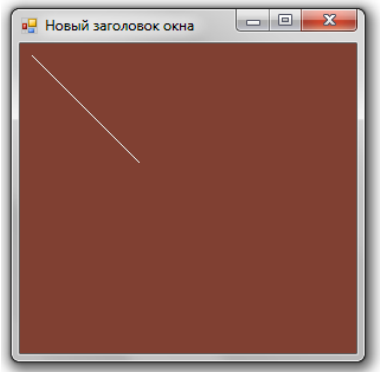


Рис 23. Рисование линии.

Рисование ломаной линии можно осуществить с помощью функции DrawLines() и определения массива точек, которые соединяются отрезками ломаной:

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
     Point[] pp = { new Point(10,10), new Point(100,50),
                    new Point(200,10), new Point(70,120)};
     e.Graphics.DrawLines(Pens.Azure,pp);
 }
```

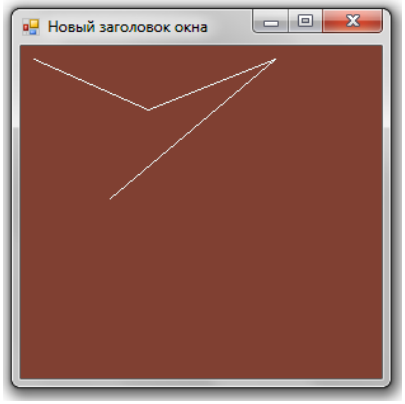


Рис 24. Рисование ломаной линии.

Для рисования замкнутой ломаной линии требуется, чтобы в массиве первая и последняя точка совпадали.

```
 private void Form1_Paint(object sender, PaintEventArgs e)
 {
     Point[] pp = { new Point(10, 10), new Point(100, 50), new Point(200, 10),
                    new Point(70, 120), new Point(10, 10) };
    e.Graphics.DrawLines(Pens.Azure,pp);
 }
```

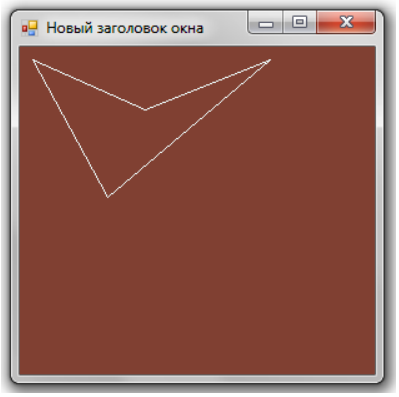


Рис 25. Рисование замкнутой ломаной линии.

Как нарисовать фигуру, полученную в виде совместного рисования нескольких фигур, например, прямоугольника и круга?

В пространстве имен System.Drawing2D существует специальный класс GraphicsPath, который описывает сложные контуры, полученные несколькими фигурами. Фактически, этот класс хранит список различных фигур, для добавления которых существует ряд функций, например, AddRectangle() или AddEllipse(). Для рисования таких фигур существует функции DrawPath() и FillPath() контекста устройства Graphics. Например, создадим GraphicsPath, состоящий из прямоугольника и круга:

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
     // создание контура из прямоугольника и круга
     GraphicsPath path = new GraphicsPath();
     path.AddRectangle(new Rectangle(50,50,100,100));
     path.AddEllipse(new Rectangle(100,100,100,100));
     // рисование закрашенного красным цветом круга
     e.Graphics.FillPath(Brushes.Red, path);
 }
```

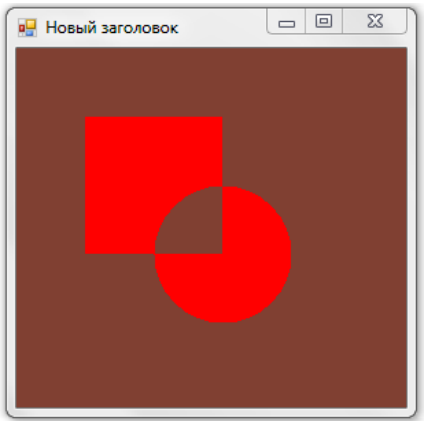


Рис 26. Рисование закрашенного GraphicsPath

Как видно, GraphicsPath определяет некоторые границы, которые закрашиваются. В случае, когда такое закрашивание нежелательно, можно комбинировать несколько GraphicsPath посредством специального класса Region, который также можно рисовать. При комбинировании доступны режимы пересечения, объединения, исключающего объединения фигур.

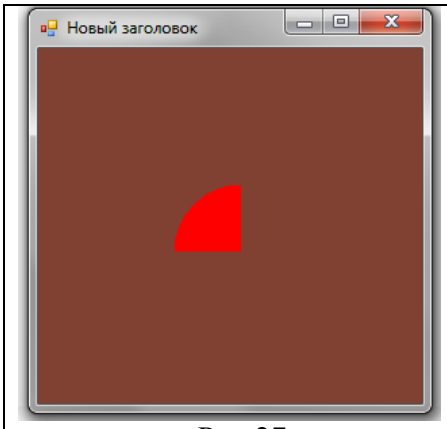


Рис.27.

а) функция пересечения (Intersect)

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    // создание GraphicsPath с прямоугольником
    GraphicsPath path1 = new GraphicsPath();
    path1.AddRectangle(new Rectangle(50,50,100,100));
    // создание GraphicsPath с кругом
    GraphicsPath path2 = new GraphicsPath();
    path2.AddEllipse(new Rectangle(100,100,100,100));

    // создание Region на базе path1
    Region r = new Region(path1);
    // пересечение с path2
    r.Intersect(path2);

    // рисование полученного объекта Region
    e.Graphics.FillRegion(Brushes.Red, r);
}
```

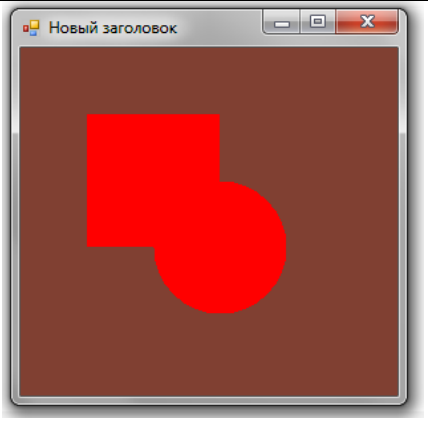


Рис.28.

б) функция объединения (Union)

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    // создание GraphicsPath с прямоугольником
    GraphicsPath path1 = new GraphicsPath();
    path1.AddRectangle(new Rectangle(50,50,100,100));
    // создание GraphicsPath с кругом
    GraphicsPath path2 = new GraphicsPath();
    path2.AddEllipse(new Rectangle(100,100,100,100));

    // создание Region на базе path1
    Region r = new Region(path1);
    // объединение с path2
    r.Union(path2);

    // рисование полученного объекта Region
    e.Graphics.FillRegion(Brushes.Red, r);
}
```

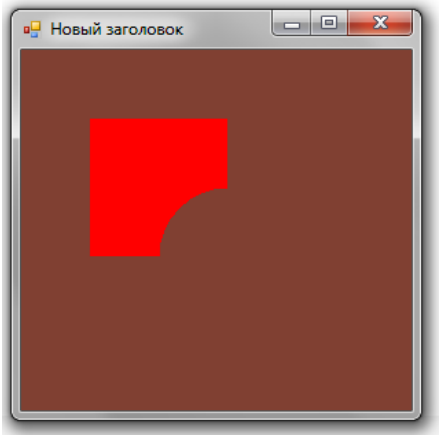


Рис.29.

в) функция разности (Exclude)

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    // создание GraphicsPath с прямоугольником
    GraphicsPath path1 = new GraphicsPath();
    path1.AddRectangle(new Rectangle(50,50,100,100));
    // создание GraphicsPath с кругом
    GraphicsPath path2 = new GraphicsPath();
    path2.AddEllipse(new Rectangle(100,100,100,100));

    // создание Region на базе path1
    Region r = new Region(path1);
    // объединение с path2
    r.Exclude(path2);

    // рисование полученного объекта Region
    e.Graphics.FillRegion(Brushes.Red, r);
}
```

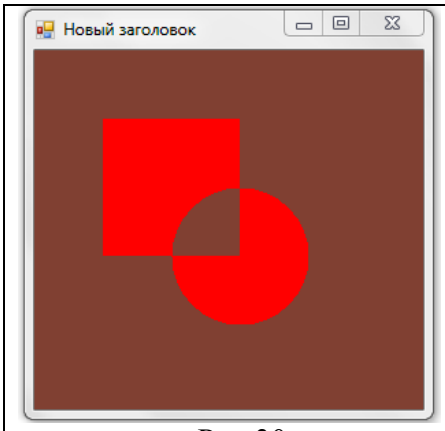


Рис.30.

г) функция исключающего объединения (Xor)

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    // создание GraphicsPath с прямоугольником
    GraphicsPath path1 = new GraphicsPath();
    path1.AddRectangle(new Rectangle(50,50,100,100));
    // создание GraphicsPath с кругом
    GraphicsPath path2 = new GraphicsPath();
    path2.AddEllipse(new Rectangle(100,100,100,100));

    // создание Region на базе path1
    Region r = new Region(path1);
    // объединение с path2
    r.Xor(path2);

    // рисование полученного объекта Region
    e.Graphics.FillRegion(Brushes.Red, r);
}
```

Как вывести в окно текст?

Символьные строки также считаются графическими примитивами. Для вывода строки используют функцию DrawString(), в параметрах которой указывают саму строку, шрифт начертания строки (можно использовать свойство Font формы), кисть заливки символов строки и точку, с которой начинается вывод строки (можно задать прямоугольник, в который должна поместиться строка).

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
     e.Graphics.DrawString("Мы учимся рисовать!", Font, Brushes.MintCream, 20, 50);
 }
```

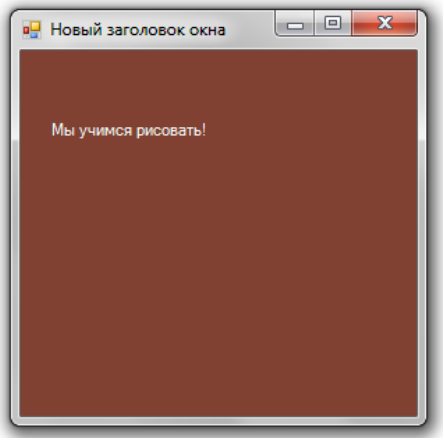


Рис 31. Вывод текста в окно.

Как задать шрифт текста?

Шрифт является инструментом рисования текста. Для задания шрифта используется класс Font, в котором задаются различные свойства. Основными свойствами являются название семейства шрифта (Arial, Times New Roman и пр.) и его размер.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Font f = new Font("Arial", 20);
    e.Graphics.DrawString("Мы учимся рисовать!", f, Brushes.MintCream, 50, 50);
}
```

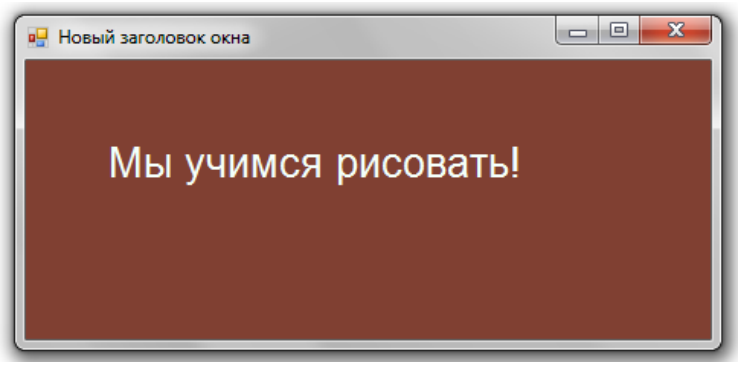


Рис 32. Вывод текста в окно с заданием шрифта.

Как вывести текст в рамке?

Часть нужно отображать текст как будто «в рамке», например, для имитации таблички. Обычно в табличках данные определяются с форматированием, например, с выравниванием по центру. Размер рамки вычисляется по характеристикам шрифта:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    // создаем шрифт
    Font f = new Font("Arial", 20);
    // измеряем размеры строки, записанной с помощью созданного шрифта
    SizeF s = e.Graphics.MeasureString("Мы учимся рисовать!", f);
    // определяем прямоугольник рамки, немного растягивая ее
    Rectangle ramka = new Rectangle(50, 50, (int)(s.Width * 1.3), (int)(s.Height * 1.3));
    // рисуем рамку
    e.Graphics.DrawRectangle(Pens.Azure, ramka);
    // определяется форматирование текста - по центру
    StringFormat sf = new StringFormat();
    sf.Alignment = sf.LineAlignment = StringAlignment.Center;
    // вывод строки в заданном прямоугольнике с заданным форматированием
    // с помощью заданного шрифта
    e.Graphics.DrawString("Мы учимся рисовать!", f, Brushes.MintCream, ramka, sf);
}
```

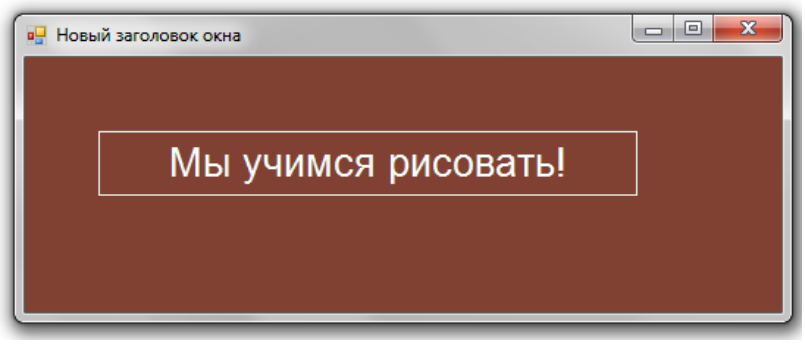


Рис 33. Вывод текста «в рамке».

Как загрузить изображение из файла?

Для работы с изображениями используется класс Bitmap. Этот класс можно использовать, когда мы сами формируем изображение в памяти. Тогда создается изображение заданного размера, формируется его контекст устройства и с его помощью производится рисование в Bitmap с помощью обычных функций рисования, например, FillRectangle() или DrawString().

```
// создание изображения размером 500 на 300
Bitmap bmp = new Bitmap(500, 300);
// создания контекста устройства рисования в изображении
Graphics g = Graphics.FromImage(bmp);
// рисование в bmp
g.DrawEllipse(Pens.Azure, 10, 10, 100, 100);
```

Класс Bitmap можно использовать и для загрузки изображения из файла. Для этого при создании изображения следует указать строку с адресом расположения файла-источника. Заметим, что класс Bitmap отвечает за все известные форматы хранения изображений (bmp, gif, jpg, png и пр.), что очень удобно.

Как вывести изображение в окно?

Для вывода изображения в окно используется функция DrawImage() класса Graphics, которая имеет много перегруженных форм. Простой вывод изображение в заданном месте окна может быть следующим, например, в левый верхний угол окна (точка (0,0):

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Bitmap image = new Bitmap("C:\\Users\\Public\\Pictures\\SamplePictures
                                \\Chrysanthemum.jpg");
    e.Graphics.DrawImage(image, 0, 0);
}
```

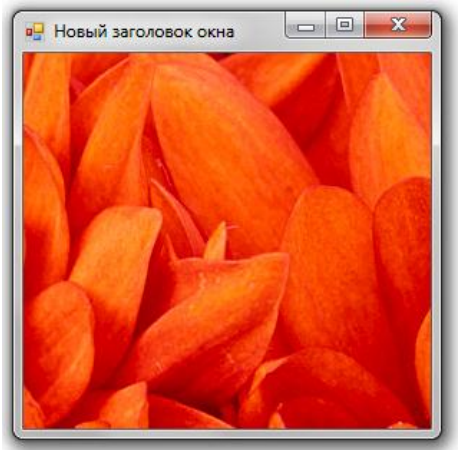


Рис 34. Вывод изображения.

Данный вариант не всегда хорош. Например, в данном случае мы видим в окне только кусочек изображения, которое намного больше окна.

Можно в качестве местоположения рисунка указывать прямоугольную область, в которую нужно рисунок поместить. Тогда изображение будет масштабировано до

размеров прямоугольника. Здесь нужно быть осторожными, поскольку не все форматы изображений позволяют без искажений масштабировать рисунок.

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
     Bitmap image = new Bitmap("C:\\Users\\Public\\Pictures\\Sample Pictures
                                \\Chrysanthemum.jpg");
     // вывод изображения в прямоугольнике, определяющем
      // клиентскую область окна (ClientRectangle)
     e.Graphics.DrawImage(image, ClientRectangle);
 }
```

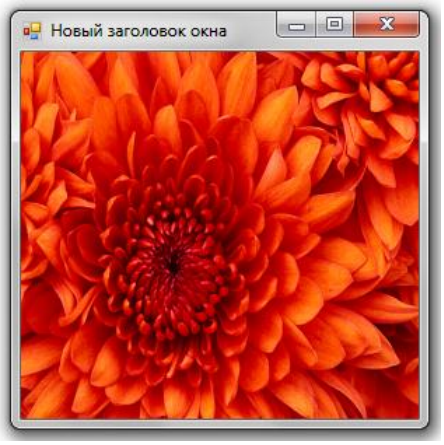


Рис 35. Вывод изображения с масштабированием.

Наконец, можно выводить не все изображение, а какой-то его кусочек, определяя его также с помощью прямоугольника. В этом случае определяют прямоугольник-источник (в координатах изображения), прямоугольник-адресат (в координатах окна) и единицу измерения (обычно пиксель):

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
     Bitmap image = new Bitmap("C:\\Users\\Public\\Pictures\\Sample Pictures
                                \\Chrysanthemum.jpg");
     // прямоугольник-источник
     Rectangle rSrc = new Rectangle(500, 10, 100, 100);
     //прямоугольник-адресат
     Rectangle rDest = new Rectangle(20, 20, 100, 100);
     // вывод изображения попиксельно
     e.Graphics.DrawImage(image, rDest, rSrc, GraphicsUnit.Pixel);
 }
```

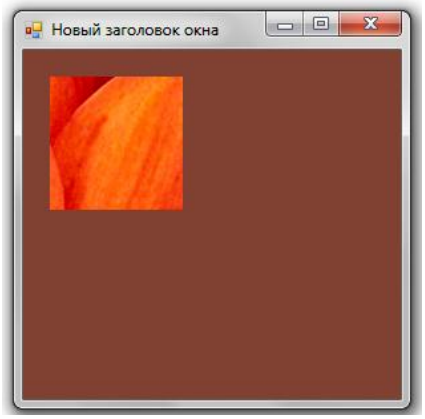


Рис 36. Вывод части изображения.

Как принудительно вызвать событие Paint, т.е. перерисовать окно?

Событие Paint вызывается операционной системой в моменты, когда окно впервые появляется, а также, когда окно перекрывается другими окнами, частично уводится за пределы экрана, сворачивается и в других случаях. Эти ситуации отслеживать не надо, поскольку за них отвечает операционная система.

Однако часто бывает необходимо перерисовывать окно в силу получения новой информации или других действий пользователя. Такая перерисовка вызывается с помощью функций окна Invalidate() или Refresh(). Обе функции обращаются к операционной системе с запросом на перерисовку окна.

Где объявлять переменные, значения которых нужны для всех функций окна?

Переменные, которые нужны в нескольких функциях окна, объявляются в пределах класса. Поскольку обработчики событий не могут иметь дополнительные параметры, многие переменные, нужные для алгоритма работы с окном хранятся на уровне класса. Эти переменные с точки зрения объектно-ориентированного программирования являются структурными характеристиками, свойствами класса. Ими может пользоваться любая функция в этом классе.

Например, пусть по щелчку мыши происходит запоминание координат, в которых был произведен щелчок. Далее требуется именно в этих координатах нарисовать заданную фигуру. В этом случае удобно на уровне класса хранить две переменные для координат щелчка мыши (x и y). При возникновении события щелчка мыши (MouseClick) запоминать текущие координаты мыши и вызывать перерисовку окна. В обработчике же события Paint окна производить собственно рисования в координатах x и y.

```
public partial class Form1 : Form
{
 // переменные для хранения координат фигуры
 int x, y;

 public Form1()
 {
     InitializeComponent();
     // начальные значения координат фигуры
     x = 0; y = 0;
 }

 // обработчик щелчка мыши в окне
 private void MyMouseClick(object sender, MouseEventArgs e)
 {
     // запоминаем координаты мыши
     x = e.X;
     y = e.Y;
     // вызываем перерисовку окна
     Refresh();
 }
```

```
    // обработчик события Paint
    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        // рисование фигуры в координатах x и y
        e.Graphics.FillRectangle(Brushes.MediumTurquoise,x,y,100,100);
    }
}
```

Как сделать анимацию?

Анимация представляет собой смену изображений окна через определенный промежуток времени. Каждый кадр представляет собой набор графических объектов в определенном состоянии и положении. Через некоторое время состояние и/или положение объекта должно измениться и новый кадр должен быть нарисован заново. Таким образом, при формировании анимации требуется:

- Создать переменные класса, которые будут хранить состояние объектов на кадре;
- В зависимости от значений этих переменных рисовать кадр в функции-обработчике события Paint формы;
- Создать таймер, который будет генерировать событие Tick через определенный интервал времени (интервал смены кадров);
- В обработчике события Tick таймера предусмотреть изменение переменных, описывающих состояние объектов кадра, и вызвать перерисовку формы, чтобы отобразить новый кадр.

Например, пусть прямоугольник передвигается по диагонали из левого верхнего угла в правый нижний. По достижению края окна, он должен остановиться. Код обработчиков событий в классе Form1 может быть следующим:

```
public partial class Form1 : Form
{

    // переменные для хранения координат прямоугольника
    int x, y;


    public Form1()
    {
        InitializeComponent();
        // начальные значения координат прямоугольника
        x = 0; y = 0;
    }

    // обработчик события Paint – рисование прямоугольника в заданных координатах
    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        e.Graphics.FillRectangle(Brushes.MediumTurquoise,x,y,50,50);
    }

    // обработчик события таймера – изменение положения прямоугольника и вызов перерисовки окна
    private void MyTimer_Tick(object sender, EventArgs e)
    {
        // запоминаем текущие координаты прямоугольника
        int prev_x = x, prev_y = y;
        // изменение координаты x, если прямоугольник остается в клиентской части окна
        if (x + 50 < ClientSize.Width)
            x += 5;
```

```
            // изменение координаты y, если прямоугольник остается в клиентской части окна
            if (y + 50 < ClientSize.Height)
                y += 5;
            // если координаты прямоугольника не изменились, он достиг правого нижнего угла окна,
            // поэтому можно остановить таймер
            if (x == prev_x && y == prev_y)
                MyTimer.Stop();                // остановка таймер
            // вызов перерисовки окна
            Refresh();
        }
    }
```

Как добавить в окно таймер?

Таймер является одним из так называемых элементов управления формы. В состоянии активности таймер генерирует через заданный интервал времени событие Tick.
Чтобы добавить таймер в окно (как, собственно, и любой другой элемент управления) нужно воспользоваться панелью ToolBox (Панель элементов) в режиме конструктора формы (если эта панель не появляется автоматически, ее можно открыть с помощью меню «Вид (View)» - «Панель элементов (ToolBox)»).

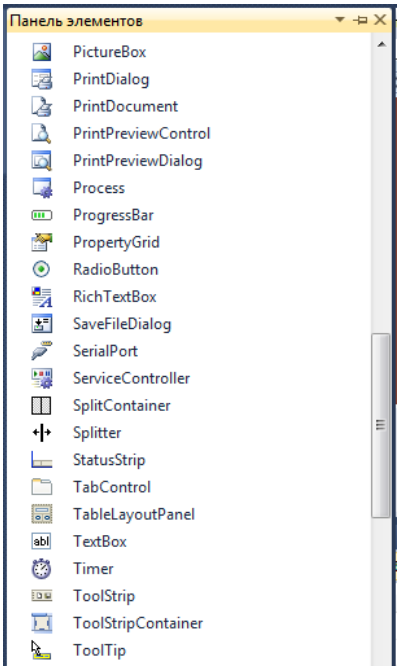


Рис 37. Панель элементов.

Выбранный элемент управления с этой панели можно «перетащить» в окно. В зависимости от возможностей его отображения элемент будет показан на форме. В частности, таймер не является видимым элементом. Поэтому он отображается в нижней панели окна конструктора.

Как и окно, элемент управления имеет свойства и события, список которых можно просмотреть в окне свойств, вызванном, например, с помощью контекстного меню:

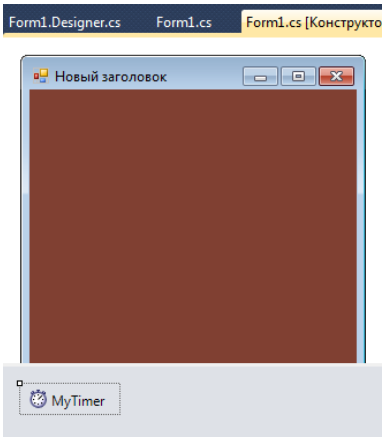


Рис 38. Окно с элементом управления Timer

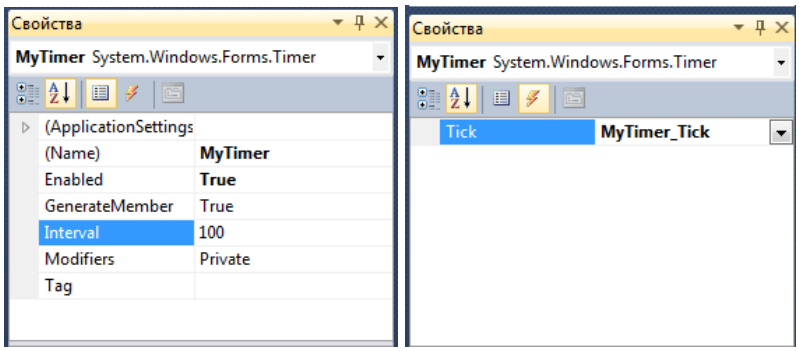


Рис 39. Окно просмотра и редактирования свойств и событий таймера MyTimer.

Среди свойств таймера выделим Name (имя объекта-таймера), через которое к нему можно обращаться из программы, в частности, для запуска (MyTimer.Start();) или остановки (MyTimer.Stop();), свойство Enabled (если оно равно True, то таймер запущен, иначе остановлен), свойство Interval, которое хранит количество миллисекунд между двумя генерациями события таймера. Таймер реализует только одно событие Tick.

Что такое элемент управления?

На Рис.37 представлено окно со всеми возможными элементами управления, которые можно добавить на форму. Первоначально элементами управления считали только визуальные элементы, которые позволяли в особом виде представлять данные, вводить их (текстовые поля, списки и пр.). Особое внимание выделяли элементам, с помощью которых и мыши можно было бы запускать некоторую программную обработку (кнопки, меню, панели инструментов). В этих случаях элементы управления являются окнами, которые обладают набором свойств, аналогичных форме, а также имеют собственные свойства.

Более важная особенность элементов управления – возможность обрабатывать некоторый набор событий. Этот набор событий для каждого элемента управления свой, например, для кнопки главное событий – нажатие, для списка – выбор элемента, для текстового поля – ввод в поле какого-то значения, для меню – выбор какого-то его пункта. Именно по признаку обработки событий были введены многие другие элементы управления, которые не всегда являются «видимыми». Примером такого

«невидимого» элемента управления является таймер (Timer), который рассматривался в предыдущем вопросе.

Как добавить в окно текстовое поле или кнопку?

Добавление визуального элемента управления заключается в «перетаскивании» выбранного элемента из «Панели элементов» (Toolbox) на форму. Место «отпускания» элемента управления будет местом расположения его на форме. Далее можно редактировать его свойства и назначать обработчики событий с помощью выбора элемента управления и вызова через контекстное меню окна свойств «Properties». Свойство Name элемента управления задает имя, по которому к нему можно будет обращаться в программе из функций формы, которой принадлежит элемент. По умолчанию система сама задает имена элементам управления. Так, если будут добавлены на форму два текстовых поля, то их имена по умолчанию будут textBox1 и textBox2.

Пусть, например, требуется спроектировать форму, с помощью которой можно будет играть в игру «Угадай число» (или «Больше-меньше»). Установим на форму кнопку с надписью «Загадать» (элементу управления Button), при нажатии на которую происходит генерация случайного числа, которое загадывает программа. Ниже будет располагаться поясняющая надпись (Label с надписью «Введите число») и текстовое поле (TextBox), в которое пользователь должен ввести то число, которое он хочет проверить. Еще ниже располагаются кнопка «Проверить» и еще две надписи для пояснения «больше-меньше» и подсчета количества попыток. Такая форма будет выглядеть так:

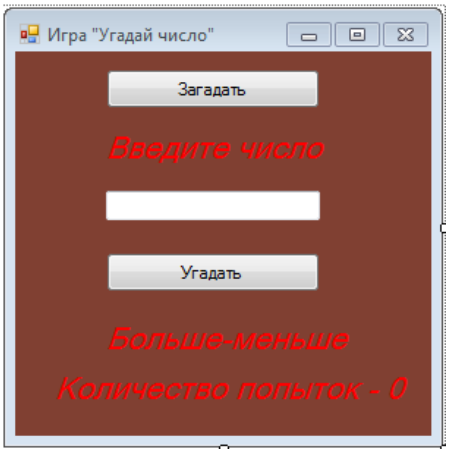


Рис 40. Вид формы для игры «Угадай число».

Как получить то, что было введено в текстовое поле?

В зависимости от типа элемента управления свойства, с помощью которых можно получить информацию из элементу, будут различны. Самая простая ситуация с текстовым полем – свойство для получения того, что пользователь ввел в текстовое поле называется Text. Так что для получения (также как и для установки значения текста в поле) требуется обратиться по имени к текстовому полю и его свойству Text

(например, textBox1.Text, если текстовое поле названо системой по умолчанию, или MyTextBox.Text, если имя текстового поля было названо MyTextBox).

Как понять, что была нажата кнопка и выполнить какие-то действия?

Нажатие кнопки является основным ее событием. Для его обработки требуется назначить функцию-обработчик для события Click в окне свойств-событий кнопки:

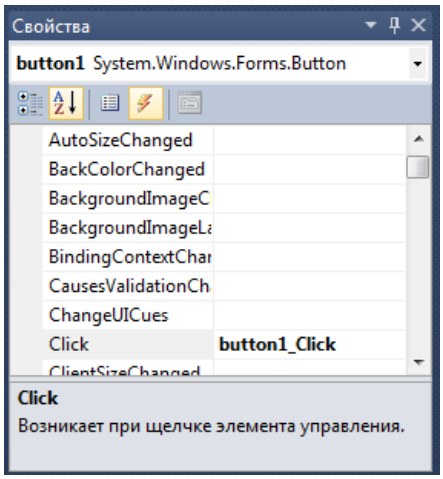


Рис 41. Окно свойств для кнопки. Задание обработчика нажатия кнопки.

В примере игры «Угадай число» две кнопки – «Загадать» и «Угадать». При нажатии на кнопку «Загадать» система должна сгенерировать число и запомнить его в переменных формы.

```
public partial class Form1 : Form
{

        // загаданное число
       int x;

       //количество попыток
       int count = 0;
       .      .      .
}
```

Обработчик нажатия кнопки «Загадать» может быть таким:

```
private void Generate_Click(object sender, EventArgs e)
 {
     Random r = new Random();
     x = r.Next(100);            //генерация числа
     count = 0;                  //попыток угадать число пока не было
 }
```

В обработчике кнопки «Угадать» требуется считать то, что пользователь ввел в текстовое поле и сравнить введенное число с загаданным, увеличить количество попыток угадывания и сообщить результат попытки (больше, меньше, угадано):

```
    private void Try_Click(object sender, EventArgs e)
    {
        count++;            // сделана еще одна попытка угадать число
        label3.Text = "Количество попыток - " + count;
        // считываем число из текстового поля
        int y = int.Parse(textBox1.Text);
        // число угадано
        if (y == x)
            label2.Text = "Угадали";
        // число меньше загаданного
        if (y < x)
            label2.Text = "Меньше";
        // число больше загаданного
        if (y > x)
            label2.Text = "Больше";
    }
```

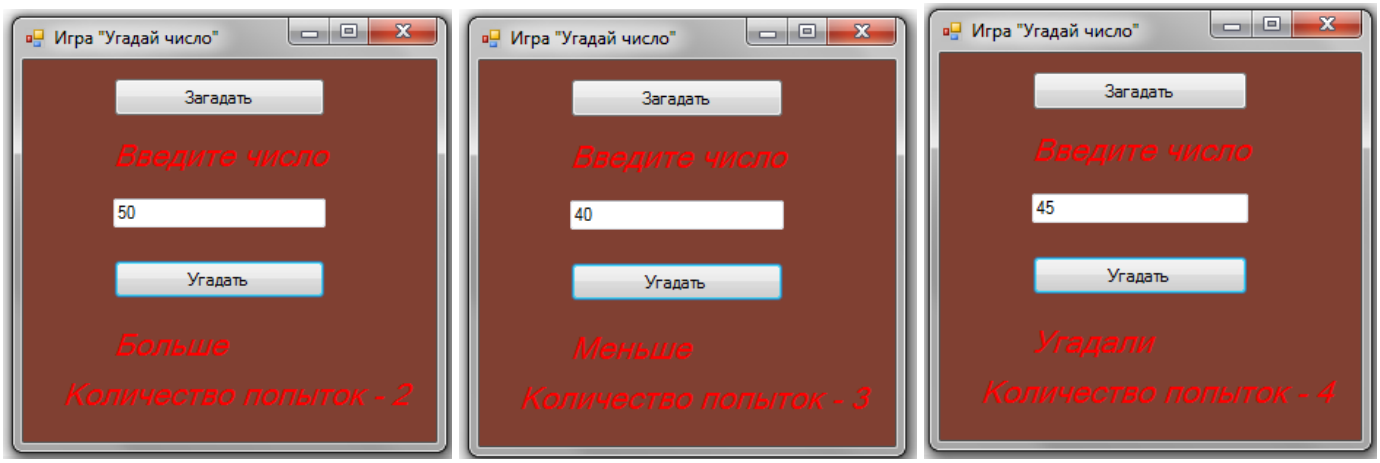


Рис 42. Окно игры «Угадай число».

Программный код для реализации данного окна таков:

```
public partial class Form1 : Form
{
    // загаданное число
    int x;

    //количество попыток
    int count = 0;

    public Form1()
    {
        InitializeComponent();
    }

    private void Generate_Click(object sender, EventArgs e)
    {
        Random r = new Random();
        x = r.Next(100);
        count = 0;
    }

    private void Try_Click(object sender, EventArgs e)
    {
        count++;            // сделана еще одна попытка угадать число
        label3.Text = "Количество попыток - " + count;
        // считываем число из текстового поля
        int y = int.Parse(textBox1.Text);
        // число угадано
        if (y == x)
```

```
                label2.Text = "Угадали";
            // число меньше загаданного
            if (y < x)
                label2.Text = "Меньше";
            // число больше загаданного
            if (y > x)
                label2.Text = "Больше";
        }
    }
```

Что такое меню?

Меню обычно называется систематизированный набор команд, которые может выполнить пользователь приложения. Главное меню окна обычно прикреплено к его верхнему краю и представляет собой «линейку» именованных групп команд меню. Выбор меню верхнего уровня обычно приводит к появлению выпадающего меню с перечислением конкретных команд этой группы. Некоторые пункты могут иметь свои выпадающие подменю и т.д. Такое меню называют каскадным.

Другой вид меню – контекстное меню. Оно вызывается при нажатии правой кнопки мыши в окне или на некотором объекте окна (например, элементе управления). Контекстным оно называется, так как содержание его команд зависит от того объекта, который был выбран в момент нажатия правой кнопки мыши.

Как добавить в окно меню?

Меню в C# является элементом управления MenuStrip (контекстное меню – элемент управления типа ContextMenuSttrip). Добавление элемента управления в окно осуществляется «перетаскиванием» элемента управления на форму из «Панели элементов» (ToolBox).

После добавления меню в окно дизайнера формы можно визуально задать названия всех пунктов меню приложения. Для новых опций меню конструктор меню содержит подсказки «Вводить здесь». В следующем примере спроектировано меню из двух пунктов верхнего уровня. Первый пункт «Подменю 1» содержит выпадающее меню из трех пунктов.

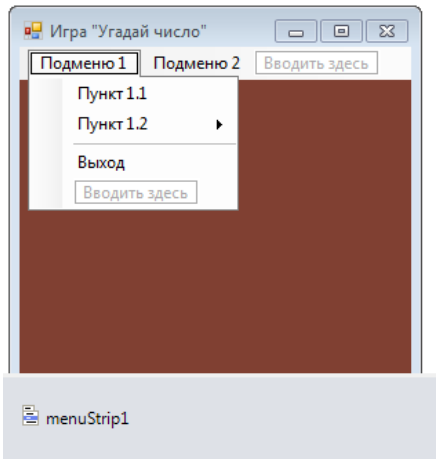


Рис 43. Конструктор меню.

Второй пункт «Пункт 1.2» также приводит к выпадающему меню следующего уровня:

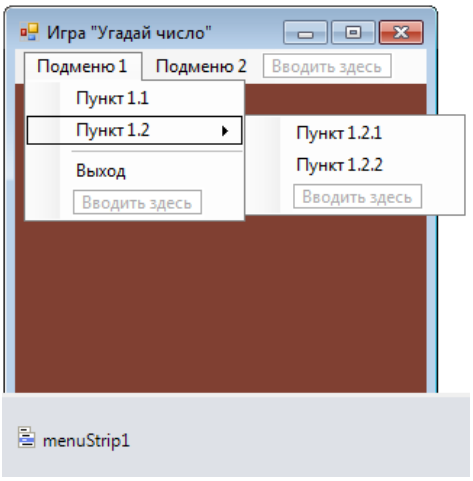


Рис 44. Конструктор меню. Выпадающее каскадное меню.

Как назначить действия, которые должны быть выполнены при выборе пункта меню?

Каждый «конечный» пункт меню должен приводить к вызову некоторых функции-обработчика аналогично командным кнопкам. Например, для пункта меню «Выход» может быть назначен следующий обработчик:

```
private void выходToolStripMenuItem_Click(object sender, EventArgs e)
{
    Dispose();      // закрыть окно приложения
}
```

Что такое диалог?

Диалогом называют специальные окна, которые не являются главным окном приложения, вызываются по необходимости (при возникновении определенных событий – выборе пункта меню, нажатии кнопок, нажатии кнопки мыши и т.д.). Эти окна используются для того, чтобы пользователь задавал какие-то данные, выбирал настройки приложения и для других служебных целей. Особенностью диалога является то, что его окно не позволяет переключаться на другие окна приложения, пока пользователь не закроет диалог с помощью кнопок (OK, Отмена, Да, Нет, Повторить, Игнорировать, Прервать и др.). После закрытия диалогового окна можно узнать, по нажатию какой кнопки диалог был завершен.

Как выбрать цвет с помощью диалога?

Диалоги можно создавать самостоятельно в приложении. Для стандартный настроек, которые часто требуется сделать в различных приложениях, в библиотеке языка C# существуют классы для стандартных диалогов. Примером такого диалога является диалог выбора цвета.

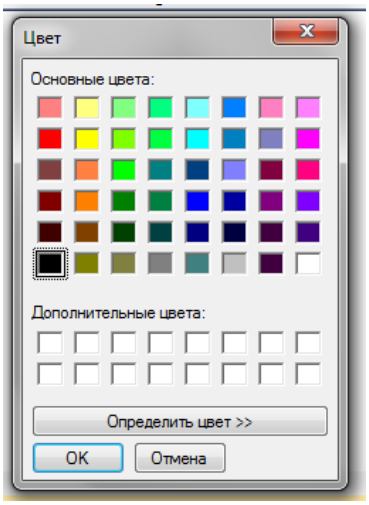


Рис 45. Стандартное окно диалога выбора цвета.

Для вызова такого диалога используется класс ColorDialog. Например, пусть по щелчку мыши на форме требуется вызвать этот диалог и поменять на цвет, выбранный в нем, цвет фона окна. Для этого потребуется создать объект диалога, вызвать его с помощью специальной функции ShowDialog() и, если диалог был закрыт по нажатию кнопки OK (функция ShowDialog() возвращает значение из перечисления DialogResult со значением соответствующей кнопки), изменить цвет фона на тот цвет, который был выбран (свойство диалога Color).

```
// создание объекта окна диалога
ColorDialog dlg = new ColorDialog();
// показ диалога на экран и проверка, с помощью какой кнопки он был закрыт
if (dlg.ShowDialog() == DialogResult.OK)
    // изменение цвета фона путем обращения к цвету, выбранному в диалоге
    BackColor = dlg.Color;
```

Как выбрать файл с помощью диалога?

Другой пример стандартного диалога – диалог открытия (OpenFileDialog) или сохранения (SaveFileDialog) файла. Эти два диалога отличаются только названием кнопки, нажатие которой эквивалентно OK.

Например, такой диалог можно применить для выбора файла-картинки, который должен отобразиться в окне.

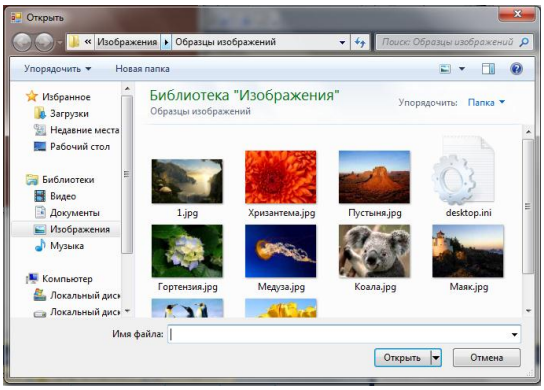


Рис 46. Стандартное окно диалога выбора файла.

После выбора файла:

Рис 47. Загруженное изображение из файла.

Сначала, как и в примере в ответе на предыдущий вопрос, создается объект-диалог, производятся его настройки и производится его вызов (обращение к функции ShowDialog()). Вывод изображения будет осуществляться в специальный элемент управления, который предназначен для просмотра изображений (PictureBox). Для создаваемого PictureBox указываются левый верхний угол (Location), размеры (Size – из ClientSize окна), Image – загруженное изображение из файла, SizeMode – режим показа изображения (центрирование, сжатие и пр.). После задания всех свойств PictureBox он добавляется на форму в коллекцию принадлежащих ей элементов управления Controls.

```
// создание объекта окна диалога
OpenFileDialog dlg = new OpenFileDialog();
// настройка типов файлов, которые должны отображаться в диалоге
dlg.Filter = "jpg-файлы|*.jpg|gif-файлы|*.gif|png-файлы|*.png|bmp-файлы|*.bmp";
// вызов диалога
if (dlg.ShowDialog() == DialogResult.OK)
{
    // создание PictureBox для показа изображения
    PictureBox pb = new PictureBox();
     // установка левого верхнего угла изображения
    pb.Location = new Point(0, 0);
    // установка размеров изображения
    pb.Size = ClientSize;
    // загрузка изображения из выбранного файла dlg.FileName
    pb.Image = new Bitmap(dlg.FileName);
    // указание, что изображение должно быть масштабировано под размер PictureBox
    pb.SizeMode = PictureBoxSizeMode.StretchImage;
    // добавление PictureBox на форму
    Controls.Add(pb);
}
```

Как загрузить изображение из файла, чтобы фон изображения считался прозрачным?

Для некоторых форматов графических файлов (например, gif или png) есть возможность задать определенный цвет или цвет фона изображения прозрачным. Эта возможность позволяет выделять из изображения только значимую часть даже очень сложной структуры. Эта возможность реализована в классе Bitmap с помощью функции MakeTransparent(). Например, в следующем примере загружается png-файл с

указанием, что его фон будет прозрачным. Далее рисуется эллипс и частично накладывается на него загруженное изображение.

```
// загрузка изображения из файла
Bitmap bmp = new Bitmap("1.png");
// делаем фон изображения прозрачным
bmp.MakeTransparent();
// рисуем эллипс и поверх него изображение
e.Graphics.FillEllipse(Brushes.Aqua, e.X - 10, e.Y - 10, 40, 40);
e.Graphics.DrawImage(bmp, e.X, e.Y);
```

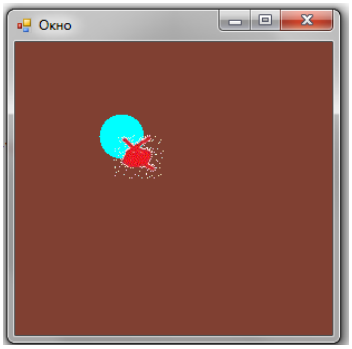


Рис 48. Изображение с прозрачным фоном.

Как загрузить звуковой файл?

В языке C# в пространстве имен System.Media определен класс-проигрыватель wav-файлов SoundPlayer. Использовать этот класс достаточно просто – требуется создать объект класса с указанием имени открываемого файла формата wav. После требуется вызвать функцию Play() для одиночного проигрывания файла или PlayLooping() для цикличного проигрывания файла.

```
SoundPlayer sw = new SoundPlayer("Example.wav");
sw.Play();
```