# C#: работаем с графикой и графическими компонентами в приложениях Windows Forms

Для работы с изображениями в библиотеке .NET определён базовый класс `System.Drawing.Image`, который предоставляет функциональные возможности для производных классов `System.Drawing.Bitmap` (растровая графика) и `System.Drawing.MetaFile` (векторная графика). Этот класс содержит методы для создания (и сохранения) объектов типа `Image` из указанного файла, потока данных и т.д.

Когда требуется перерисовка элемента управления, происходит событие `Paint`, которое, в зависимости от задачи, можно как программировать явно, так и полагаться на его автоматический вызов, происходящий, когда изменилась графическая канва объекта. Для отрисовки готового файла с изображением, имя которого задано или получено из диалога открытия файла `OpenFileDialog`, мы должны создать или получить из аргумента `PaintEventArgs` метода `Paint` графическую канву типа `System.Drawing.Graphics` а затем уже вывести на неё изображение.

Для использования готовых методов обработки изображений (поворот, масштабирование, изменение цвета и т.п.) мы программно создаём объект типа `System.Drawing.Bitmap`, копирующий имеющееся изображение, выполняем его обработку, а затем выводим изменённый объект в компоненту, предназначенную для отображения, такую как `PictureBox`.

**Проект Lab5_1**. Выведем выбранный в стандартном диалоге открытия файла рисунок на форму (пункт меню Файл - Открыть) и принудительно перерисуем по пункту меню Правка - Перерисовать. В свойствах диалога открытия файла `openFileDialog1` указано `Filter = Все файлы|*.*|Рисунки BMP|*.bmp|Рисунки JPEG|*.jpg|Рисунки PNG|*.png` а свойство `FileName` равно пустой строке. В классе формы пропишем объекты "Изображение" и "Имя файла":

```
private Image Img;
private String Name;
```

Напишем обработчик открытия файла:

```
  //Обработка меню Файл - Открыть
  openFileDialog1.ShowDialog();
  Name = openFileDialog1.FileName.Trim();
  if (String.IsNullOrEmpty(Name)) return;
  try {
   Img = new Bitmap(Name);
   //или так: Img = Image.FromFile(Name);
  }
  catch (Exception ex) {
   MessageBox.Show(ex.Message + Environment.NewLine + "(не могу открыть файл)",
    "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
   toolStripLabel1.Text = "Файл не выбран";
   Img = null;
   return;
  }
  this.ClientSize = new System.Drawing.Size(Img.Width, Img.Height);
   //Размер формы подгогнали под размер картинки
  toolStripLabel1.Text = Name;
   //Имя файла вывели в панель инструментов
  this.Refresh(); //Потребовать перерисовки!
```

По событию `Paint` формы будет выполняться отрисовка объекта `Img` на текущей канве, Y-координата для вставки рисунка учитывает пространство, занимаемое по вертикали компонентами `menuStrip1` и `toolStripLabel1`:

```
 private void Form1_Paint(object sender, PaintEventArgs e) {
  if (Img != null) {
   Point p = new Point(0, menuStrip1.Size.Height+ toolStripLabel1.Size.Height);
   e.Graphics.DrawImage(Img, p);
```

```
    }
  }
```

Объект «Графика», представляющий собой поверхность для рисования, также может быть получен для канвы формы (вот обработчик пункта меню Правка - Перерисовать):

```
  if (Img != null) {
   Graphics g = this.CreateGraphics();
   Point p = new Point(0, menuStrip1.Size.Height + toolStripLabel1.Size.Height);
   g.DrawImage(Img, p);
  }
```

или из загруженного (сгенерированного) изображения, например, см. код для пункта меню Файл - Создать:

```
  Img = new Bitmap(200, 200,
   System.Drawing.Imaging.PixelFormat.Format24bppRgb);
  Graphics Gr = Graphics.FromImage(Img);
  // Теперь становятся доступными методы класса Graphics!
  Pen pen = new Pen(Color.ForestGreen, 4.0F);
  Gr.DrawLine(pen, 0, 0, 199, 199);
  Gr.RotateTransform(180.0F); //поворот на 180 градусов
  Img.Save("example.jpg", System.Drawing.Imaging.ImageFormat.Jpeg); //сохранение
  this.Refresh();
```

Обычно рисунки не отображают на канве формы, а работают с компонентом `PictureBox` (вкладка Стандартные), представляющим собой контейнер для размещения рисунка, вот его основные свойства:

- `Image` - рисунок. Само изображение можно загрузить программно через свойство `ImageLocation`, например, в методе `Load` формы:

  ```
  openFileDialog1.ShowDialog();
  if (openFileDialog1.FileName != null)
   this.pictureBox1.ImageLocation = this.openFileDialog1.FileName;
  ```

  или же присвоить этому свойству объект `Image`, как мы делали выше.
- `SizeMode` - режим вывода: `Normal` - по левому верхнему углу контейнера с обрезанием, `StretchImage` - вписать в компонент, `AutoSize` - компонент примет размеры изображения, `CenterImage` - центрировать, не меняя размеры (может обрезать рисунок), `Zoom` - пропорционально масштабировать по размерам компонента (пространство имен `PictureBoxSizeMode`).

Для возможности прокрутки загруженного изображения достаточно разместить `PictureBox` на элементе `Panel` с установленным свойством `AutoScroll = true` (и `Dock = Fill`, если панель должна занимать всю клиентскую часть окна) и при этом для `PictureBox` указать `SizeMode = AutoSize`. После этого можно загрузить рисунок кодом вида

```
Image Img = new Bitmap(openFileDialog1.FileName);
pictureBox1.Image = Img;
```

- `ErrorImage` - позволяет задать изображение, выводимое при ошибке загрузки;
- `InitialImage` - позволяет задать изображение, выводимое в процессе загрузки.

К другим полезным компонентам можно отнести:

- `ImageList` (вкладка Компоненты) - список изображений, который можно использовать для "прокрутки" картинок или как список иконок для меню, вкладок и т.п.
- `Timer` (вкладка Компоненты), позволяет обрабатывать периодическое событие `Tick` и организовывать смену картинок в реальном времени, частота повторения события в миллисекундах задаётся свойством `Interval`.

Мы используем их в следующей теме.

Скачать пример Lab5_1 в архиве .zip с проектом C# Visual Studio 2019 (12 Кб)

**Проект Lab5_2**. Основные операции над изображениями. Кроме `pictureBox1`, размещённого на `Panel` как описано выше, форма включает в себя стандартный диалог открытия файла `openFileDialog1`, меню Файл - Открыть (обработчик аналогичен предыдущему примеру) и меню "Правка", откуда мы будем вызывать обработчики загруженного изображения.

5.2.1. Поворот и отражение изображений. В этом примере выведенный в `PictureBox` рисунок поворачивается на 180 градусов и выводится обратно в `PictureBox`:

```
  if (pictureBox1.Image != null) {
   Bitmap bitmap1 = new Bitmap(pictureBox1.Image);
```

```
 if (bitmap1 != null) {
  bitmap1.RotateFlip(RotateFlipType.Rotate180FlipY);
  pictureBox1.Image = bitmap1;
 }
}
```

Остальные повороты (отражения) – другие значения параметра `RotateFlipType`.

5.2.2. Масштабирование изображения или его части. Код ниже показывает, как можно программно уменьшить загруженное в компоненту `PictureBox` изображение в 2 раза:

```
if (pictureBox1.Image == null) return;
Bitmap  bitmap1 = new Bitmap (pictureBox1.Image); //взяли рисунок из компоненты
Graphics  Gr1 = Graphics.FromImage (bitmap1);
 //получили поверхность рисования из исходного рисунка
Bitmap  bitmap2 = new Bitmap (bitmap1.Width / 2, bitmap1.Height / 2, Gr1);
 //сделали вдвое меньший рисунок с тем же разрешением
Graphics  Gr2 = Graphics.FromImage (bitmap2);
 //получили поверхность рисования из меньшего рисунка
Rectangle compressionRectangle = new Rectangle
  (0, 0, bitmap1.Width / 2, bitmap1.Height / 2); //определили масштабирующий прямоугольник
Gr2.DrawImage (bitmap1, compressionRectangle);
 //отрисовали на поверхности второго рисунка первый со сжатием
Pen  MyPen = new Pen (Color.Red); //на измененном рисунке можно что-то подрисовать
Gr2.DrawRectangle (MyPen, 0, 0, bitmap2.Width - 1, bitmap2.Height - 1);
 //например, сделать красную рамку
pictureBox1.Image = bitmap2; //назначили второй рисунок компоненте
pictureBox1.Size = bitmap2.Size; //поставили размер компоненты по размерам нового рисунка
this.ClientSize = pictureBox1.Size; //...и такой же размер клиентской формы
```

Добавим пункт меню Файл - Сохранить и сохраним изображение:

```
if (pictureBox1.Image == null) return;
Bitmap bitmap1 = new Bitmap (pictureBox1.Image);
try {
 bitmap1.Save (openFileDialog1.FileName);
  //Сохраняем под именем из диалога открытия файла
  //Может вызвать исключение, если исходный файл ещё открыт
}
catch (Exception ex) {
 MessageBox.Show (ex.Message + "\nНе удалось сохранить файл",
   "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
 return;
}
```

Как вариант:

```
pictureBox1.Image.Save (openFileDialog1.FileName);
```

К сожалению, этот код может вызвать исключение, если исходный файл ещё открыт. Для "надёжных" операций с файлами при открытии изображений используйте `FileStream`, чтобы контролировать весь процесс (перепишем обработчик пункта меню "Открыть"):

```
openFileDialog1.ShowDialog ();
if (openFileDialog1.FileName.Trim () != "" && openFileDialog1.FileName != null) {
 System.IO.FileStream file;
 try {
  file = new System.IO.FileStream (openFileDialog1.FileName, System.IO.FileMode.Open,
    System.IO.FileAccess.Read, System.IO.FileShare.Inheritable);
 }
 catch (Exception ex) {
  MessageBox.Show (ex.Message + "\nНе удалось открыть файл", "Ошибка",
   MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
  return;
 }
 pictureBox1.Image = Image.FromStream (file);
```

```
    file.Close ();
  }
```

5.2.3. Изменение цвета на изображении. На форму добавлен стандартный `ColorDialog` и выбранный в нём цвет ставится прозрачным.

```
  if (pictureBox1.Image == null) return;
  Bitmap bitmap1 = new Bitmap (pictureBox1.Image);
  if (colorDialog1.ShowDialog () == DialogResult.OK) {
   bitmap1.MakeTransparent (colorDialog1.Color);
   pictureBox1.Image = bitmap1;
  }
```

5.2.4. Фильтрация всего изображения или его части (по пикселям). В качестве примера уменьшим вдвое интенсивность синего цвета на картинке, избегая пикселей, цвет которых близок к белому (интенсивности красной, зелёной и синей компонент больше значения 250):

```
  if (pictureBox1.Image == null) return;
  Bitmap bitmap1 = new Bitmap (pictureBox1.Image);
  for (int x = 0; x < bitmap1.Width; x++)
   for (int y = 0; y < bitmap1.Height; y++) {
    Color pixelColor = bitmap1.GetPixel (x, y);
    if (pixelColor.R > 250 && pixelColor.G > 250 && pixelColor.B > 250)
     continue; //не фильтруем пиксели, чей цвет близок к белому
    Color newColor = Color.FromArgb (pixelColor.R, pixelColor.G, pixelColor.B / 2);
    bitmap1.SetPixel (x, y, newColor);
   }
  pictureBox1.Image = bitmap1;
```

Аналогично можно реализовать любую другую фильтрацию цветов, но из-за «ручного» прохода по пикселям скорость выполнения процесса может быть заметно ниже, чем для пп. 5.2.1-5.2.3. Более быстрый способ преобразования всех цветов рисунка даёт применение фильтрации на основе класса `ColorMatrix`. В качестве примера приведём код, преобразующий цветное изображение к оттенкам серого:

```
  if (pictureBox1.Image == null) return;
  Bitmap bitmap1 = new Bitmap (pictureBox1.Image);
  Bitmap bitmap2 = new Bitmap (bitmap1.Width, bitmap1.Height);
  Graphics g = Graphics.FromImage (bitmap2);
  float [] [] Map = {
   new float[] {0.30f, 0.30f, 0.30f, 0.00f, 0.00f },
   new float[] {0.59f, 0.59f, 0.59f, 0.00f, 0.00f },
   new float[] {0.11f, 0.11f, 0.11f, 0.00f, 0.00f },
   new float[] {0.00f, 0.00f, 0.00f, 1.00f, 0.00f },
   new float[] {0.00f, 0.00f, 0.00f, 0.00f, 1.00f }
  };
  System.Drawing.Imaging.ColorMatrix GrayscaleMatrix =
   new System.Drawing.Imaging.ColorMatrix (Map);
  System.Drawing.Imaging.ImageAttributes attributes =
   new System.Drawing.Imaging.ImageAttributes ();
  attributes.SetColorMatrix (GrayscaleMatrix);
  Rectangle rect = new Rectangle (0, 0, bitmap1.Width, bitmap1.Height);
  g.DrawImage (bitmap1, rect, 0, 0, bitmap1.Width, bitmap1.Height,
   GraphicsUnit.Pixel, attributes);
  pictureBox1.Image = bitmap2;
```

О классе `ColorMatrix` можно почитать, например, по ссылке. В нашем фильтре соотношение "весов" красной, зелёной и синей цветовых компонент 0.3 - 0.59 - 0.11 отражает чувствительность человеческого глаза к оттенкам красного, зелёного и синего.

В некоторых случаях фильтровать изображения можно и сменой свойства `Image.PixelFormat`, но вариант `Format16bppGrayScale` в GDI+ не сработал.

5.2.5. Сохранить рисунок так, как он выглядит на компоненте. Следует понимать, что свойство `SizeMode` управляет отображением рисунка в компоненте, при сохранении пропорции рисунка не изменятся от того, что он был выведен, например, при `SizeMode=StretchImage` (принудительно растянут по размерам

компоненты, возможно, с нарушением пропорций). Тем не менее - а можно ли сохранить рисунок так, как он был выведен в компоненту? Да, можно, например, так:

```
 if (pictureBox1.Image == null) return;
 pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
 pictureBox1.Dock = DockStyle.Fill; //установили растягивание
 Bitmap bitmap1 = new Bitmap (pictureBox1.Image);
 Bitmap bitmap2 = new Bitmap (pictureBox1.Width, pictureBox1.Height);
  //у 2-го рисунка - размер компоненты
 Graphics g = Graphics.FromImage (bitmap2);
  //получили графический контекст из 2-го рисунка
 g.InterpolationMode = System.Drawing.Drawing2D.InterpolationMode.Bicubic;
  //настроили режим интерполяции
 g.DrawImage (bitmap1, new Rectangle(0,0, bitmap2.Width, bitmap2.Height));
  //отрисовали в контекст 2-го рисунка исходный, неискажённый рисунок
 pictureBox1.Image = bitmap2; //назначили искажённый рисунок компоненте
 сохранитьToolStripMenuItem_Click (this, e); //вызвали метод сохранения
 pictureBox1.SizeMode = PictureBoxSizeMode.AutoSize;
 pictureBox1.Dock = DockStyle.None; //восстановили свойства
 pictureBox1.Image = bitmap1; //вернули старый рисунок
```

Убедиться в том, что рисунок был пересохранён в фоновом режиме с размерами, соответствующими клиентской части формы можно, заново открыв его с диска.

5.2.6. Выделить часть рисунка и реализовать обрезку по выделенной области. В класс формы добавим следующие глобальные данные:

```
Rectangle selRect; //выделенный прямоугольник
Point orig; //точка для привязки прямоугольника
Pen pen; //перо для отрисовки
bool flag; //флажок показывает, находимся ли в режиме выделения части рисунка
```

Инициализируем их, например, в имеющемся конструкторе формы:

```
public Form1() {
 InitializeComponent();
 pen = new Pen (Brushes.Blue, 0.8f); //цвет и толщина линии выделения
 pen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash; //штрихи
 selRect = new Rectangle (0, 0, 0, 0);
 flag = false;
}
```

Реализация динамического выделения мышью потребует взаимодействия нескольких событий (нажатие кнопки мыши, отпускание кнопки мыши и перемещение мыши). Для удобства используем также динамическое переключение обработчика события `Paint` (чтобы рисовать рамку выделения только тогда, когда она нужна – происходит перемещение курсора мыши на `pictureBox` при зажатой левой кнопке).

```
private void pictureBox1_Paint (object sender, PaintEventArgs e) {
 //Этот обработчик мы создали в конструкторе
 //Для ситуации, когда выделяем рамку
 e.Graphics.DrawRectangle (Pens.Black, selRect);
}
private void Selection_Paint (object sender, PaintEventArgs  e) {
 //Добавили свой обработчик Paint для остальных ситуаций
 e.Graphics.DrawRectangle (pen, selRect);
}

private void pictureBox1_MouseDown (object sender, MouseEventArgs e) {
 //Этот обработчик мы создали в конструкторе
 //Нажали мышку - включаем наш обработчик и выключаем стандартный
 pictureBox1.Paint -= new PaintEventHandler (pictureBox1_Paint);
 pictureBox1.Paint += new PaintEventHandler (Selection_Paint);
 orig = e.Location; //запомнили, где начало выделения
 flag = true;
}
```

```
private void pictureBox1_MouseUp (object sender, MouseEventArgs e) {
 //Этот обработчик мы создали в конструкторе
 //отжали мышку - всё наоборот
 pictureBox1.Paint -= new PaintEventHandler (Selection_Paint);
 pictureBox1.Paint += new PaintEventHandler (pictureBox1_Paint);
 pictureBox1.Invalidate (); //принудительно перерисовать
 flag = false; //выйти из режима выделения
}

private void pictureBox1_MouseMove (object sender, MouseEventArgs e) {
 //Этот обработчик мы создали в конструкторе
 if (flag) { //если в режиме выделения
  selRect = GetSelectionRectangle (orig, e.Location); //запоминаем выделенное
  if (e.Button == MouseButtons.Left) {
   pictureBox1.Refresh (); //рефрешим картинку по нажатию левой кнопки
  }
 }
}

private Rectangle GetSelectionRectangle (Point orig, Point location) {
 //Этот метод пришлось написать, чтобы координаты выделения правильно запоминались
 //независимо от того, в какую сторону тащим курсор мыши
 Rectangle rect = new Rectangle ();
 int dX = location.X - orig.X, dY = location.Y - orig.Y;
 System.Drawing.Size size = new System.Drawing.Size (Math.Abs (dX), Math.Abs (dY));
  //размеры текущего выделения
 if (dX >= 0 && dY >= 0) rect = new Rectangle (orig, size);
 else if (dX < 0 && dY > 0) rect = new Rectangle (location.X, orig.Y, size.Width, size.Height)
 else if (dX > 0 && dY < 0) rect = new Rectangle (orig.X, location.Y, size.Width, size.Height)
 else rect = new Rectangle (location, size);
 return rect;
}
```

Теперь, при наличии на изображении выделенной рамки `selRect` можно, например, реализовать его обрезание по границам этой рамки:

```
 if (pictureBox1.Image == null) return;
 if (selRect.Width > 0 && selRect.Height > 0) {
  Bitmap bitmap1 = new Bitmap (pictureBox1.Image);
  Bitmap bitmap2 = new Bitmap (selRect.Width, selRect.Height);
  Graphics g = Graphics.FromImage (bitmap2);
  g.DrawImage (bitmap1, 0, 0, selRect, GraphicsUnit.Pixel);
  pictureBox1.Image = bitmap2;
  selRect = new Rectangle (0, 0, 0, 0);
 }
```

Скачать пример Lab5_2 в архиве .zip с проектом C# Visual Studio 2019 (14 Кб)

**Проект Lab5_3**. Рисование фигур. Показанный выше подход нетрудно применить для рисования геометрических примитивов на канве `PictureBox` или формы.

Создадим форму как в предыдущем примере с `PictureBox`, расположенным на `Label`, у компонент установлены те же свойства.

Рассмотрим варианты рисования линии на канве `PictureBox`. При движении мыши с зажатой левой кнопкой наша линия должна динамически обновляться, а при отпускании кнопки - добавляться на существующий рисунок.

Опишем в классе формы необходимые данные:

```
Point p1, p2; //начало и конец линии
Pen pen1; //перо
Brush brush1; //кисть
Bitmap Img1, Img2; //основная картинка, на которой рисуем и буферная
Graphics gr; //графический контекст
bool isPressed; //флажок "кнопка мыши зажата"
```

Для самой формы нам понадобится запрограммировать событие `Load`, где мы инициализируем эти объекты, то есть, создадим рисунок размером с клиентскую часть окна формы, назначим его компоненте, создадим перо и выставим в "ложь" флажок:

```
 Img1 = new Bitmap (ClientSize.Width, ClientSize.Height);
 pictureBox1.Image = Img1;
 gr = Graphics.FromImage (Img1);
 pen1 = new Pen (Color.Black);
 isPressed = false;
```

Всё остальное запрограммируем в событиях `PictureBox`. На нажатие кнопки мыши будем включать флажок и запоминать место клика `p1`:

```
private void pictureBox1_MouseDown (object sender, MouseEventArgs e) {
 isPressed = true;
 p1 = e.Location;
}
```

На отпускание кнопки получим координаты второй точки `p2` и соединим её с первой, проведя линию на образе `Img1`. Заметим, что в реальном коде можно добавлять точки в контейнер, например, в список `List` из объектов `Point`. Если при этом запоминать, какой именно объект рисовался, можно в нужные моменты просто перерисовывать объекты по списку (что может предотвратить "утечки памяти" при работе приложения), а также удалять или динамически изменять их.

```
private void pictureBox1_MouseUp (object sender, MouseEventArgs e) {
 p2 = e.Location;
 gr = Graphics.FromImage (Img1);
 gr.DrawLine (pen1, p1, p2);
 gr.Save ();
 isPressed = false;
 pictureBox1.Invalidate ();
}
```

На перемещение мыши обработка будет немного хитрей. Если кнопка не зажата, ничего делать не нужно, а в противном случае будем проводить текущую линию на копии рисунка `Img2`, созданной из `Img1`, чтобы не получилось "веера" из линий при перемещении мыши с зажатой кнопкой. `Img2` всё равно придётся временно назначить рисунком для `PictureBox`, чтобы линия была видна в процессе движения мыши.

```
private void pictureBox1_MouseMove (object sender, MouseEventArgs e) {
 if (!isPressed) return; //Кнопка не зажата - выйти
 p2 = e.Location;
 Img2 = new Bitmap (Img1);
 pictureBox1.Image = Img2;
 gr = Graphics.FromImage (Img2);
 gr.DrawLine (pen1, p1, p2);
 pictureBox1.Invalidate ();
}
```

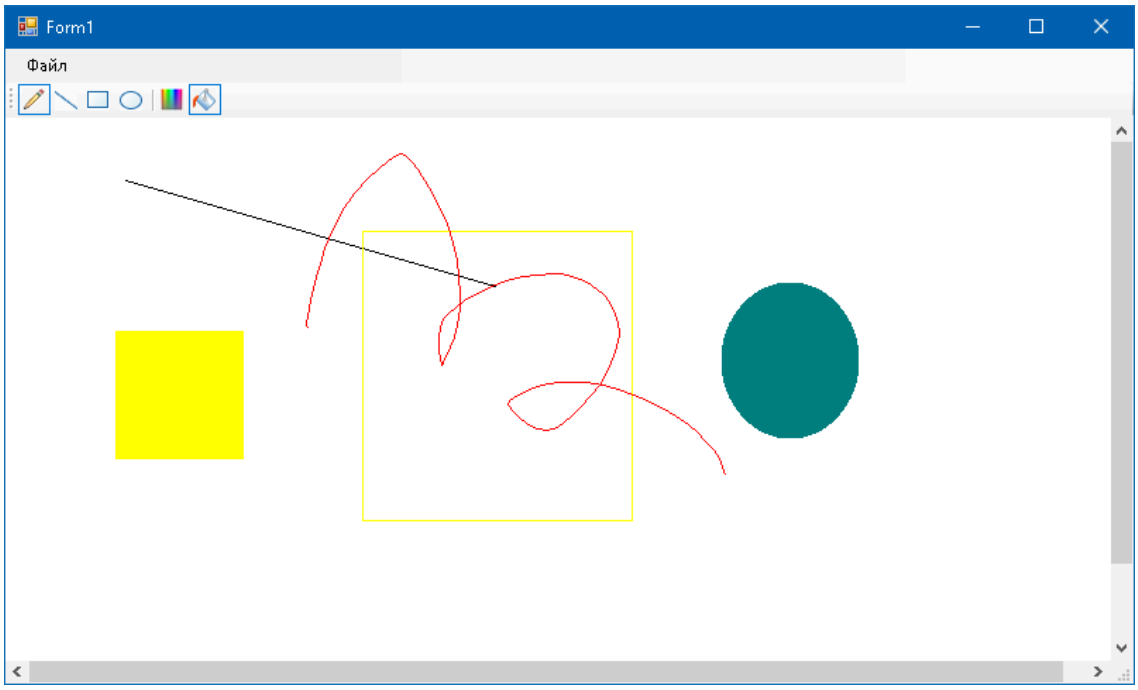
В показанном примере все координаты отсчитывались "внутри `PictureBox`" и получались непосредственно из аргумента `MouseEventArgs` обработчика события. По-другому можно сделать, используя координаты курсора относительно экрана `Cursor.Position.X`, `Cursor.Position.Y`, а затем вычитая из них координаты верхнего левого угла формы `Location.X`, `Location.Y` (и, возможно, дополнительные значения, учитывающие занятое другими компонентами пространство на форме):

```
 int x1 = Cursor.Position.X - Location.X,
  y1 = Cursor.Position.Y - Location.Y;
```

Расположив на панели инструментов приложения дополнительные кнопки для выбора геометрического примитива, цвета и т. п., мы можем получить приложение графический-редактор (см. прикреплённый проект, где ряд возможностей уже добавлен).

> Обратите внимание в коде проекта Lab5_3: при рисовании линии и эллипса координаты второй точки могут быть и "меньше" (ближе к левому верхнему углу холста), чем первой. При рисовании же прямоугольника область экрана должна быть задана, начиная с левого верхнего угла. Метод `GetRectangle`, вызываемый при обработке событий `pictureBox1_MouseUp` и `pictureBox1_MouseMove`, корректирует эту проблему, как и при выделении прямоугольником в проекте Lab5_2.

Скачать пример Lab5_3 в архиве .zip с проектом C# Visual Studio 2019 (19 Кб)




Простая рисовалка на Windows Forms C#

**Проект Lab5_4**. Другой контекст. В завершение заметим, что отрисовка, выполняемая в объекте графического контекста, позволяет без каких-либо изменений основного кода "перенести" графический объект на другую канву, например, на экран вместо окна приложения.

Пусть в классе формы имеется метод `Draw`, создающий некоторый рисунок:

```
void Draw (System.Drawing.Graphics g) { //Графический контекст передан в наш метод
 Pen [] pens =  { Pens.Red, Pens.Yellow, Pens.Green };
  //Разные перья для рисования
 int width = ( this.ClientSize.Width - 1 ) / 2,
     height = ( this.ClientSize.Height - 1 ) / 2;
  //Половинки ширины и высоты клиентской части окна
 for (int i = 0; i < 3; i++) //Демо - рисуем перьями
  g.DrawRectangle (pens [i], i * width, i * height, width, height);
 Brush [] brushes = { Brushes.Red, Brushes.Yellow, Brushes.Green };
  //Разные кисти для выполнения заливки цветом
 for (int i = 0; i < 3; i++) //Демо - рисуем кистями
  g.FillEllipse (brushes [i], i * width, i * height, width, height);
 g.DrawLine (pens [2], 0, 0, width * 2, height * 2); //Рисуем линию пером
}
```

Как и в начале статьи, мы могли бы вызвать его кодом вида

```
  Graphics g = this.CreateGraphics ();
  Draw (g);
```

для отображения картинки непосредственно на канве формы.

Теперь выведем рисунок в контексте графического экрана Windows поверх всех окон.

Убедимся, что к файлу формы подключены нужные пространства имён:

```
using System.Runtime.InteropServices;
using System.Drawing;
```

В классе формы (например, после конструктора) укажем ссылки на нужные методы библиотеки `user32.dll`, которые нам потребуются:

```
[DllImport ("user32.dll")]
public static extern IntPtr GetDC (IntPtr hwnd);
[DllImport ("user32.dll")]
public static extern void ReleaseDC (IntPtr hwnd, IntPtr dc);
```

В методе рисования (например, по событию `Paint` формы) вызовем наш метод с другим контекстом:

```
private void Form1_Paint (object sender, PaintEventArgs e) {
 IntPtr desktopPtr = GetDC (IntPtr.Zero);
```

```
  Graphics g = Graphics.FromHdc (desktopPtr);

  Draw (g);

  g.Dispose ();
  ReleaseDC (IntPtr.Zero, desktopPtr);
 }
```

Также можно потребовать где-нибудь принудительной перерисовки, например, по клику на форме:

```
private void Form1_Click (object sender, EventArgs e) {
 Invalidate ();
}
```

Подключить к проекту внешнюю библиотеку можно и непосредственно, например, для нашего случая:

- в верхнем меню выбрать команду Проект - Добавить существующий элемент...;
- в списке типов файлов выбрать "Исполняемые файлы", показать расположение нужного файла (`c:\Windows\System32\user32.dll`) и нажать "Добавить";
- выбрать добавленный файл в Обозревателе решений, в окне "Свойства" указать для него значение "Копировать в выходной каталог" равным "Копировать более новую версию";
- после этого прототипы нужных функций библиотеки можно описать в классе формы с атрибутами `public static extern` и предшествующей директивой `[DllImport ("user32.dll")]`, как мы делали выше.

Скачать пример Lab5_4 в архиве .zip с проектом C# Visual Studio 2019 (11 Кб)

**Задание** по теме может быть, например, таким: реализовать графическое приложение в соответствии с вариантом. Предусмотреть в приложении следующие возможности:

- сохранение полученных графических файлов;
- прокрутка файлов, если они «не помещаются» в окне компоненты `PictureBox`;
- возможность построить изображение более, чем в одном масштабе (или масштабировать его программно);
- не менее двух настраиваемых параметров, позволяющих изменить внешний вид объекта (например, для объекта "дом" - количество этажей и количество окон на каждом этаже).

05.04.2023, 19:26 [4231 просмотр]

теги: c# учебное программирование графика

Здесь можно оставить комментарий, обязательны только выделенные цветом поля. Не пишите лишнего, и всё будет хорошо.

Ваше имя:

Пароль (если желаете запомнить имя):

Любимый URL (если указываете, то вставьте полностью):

Текст сообщения (до 1024 символов):

Введите код сообщения (цифры): 9 5 ~~0~~ ~~7~~

Добавить Сброс

К этой статье пока нет комментариев, Ваш будет первым