Федеральное агентство по образованию

ПСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

И.В. Антонов, Ю.В. Бруттан

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ (C#)

Методические указания к лабораторным работам

Рекомендовано научно-методическим советом Псковского государственного политехнического института

Псков ППИ
2007

УДК 681.3.06
Д46
ББК 32.97

Рецензенты:
– Поляков А.О., д.т.н., профессор кафедры ВСиИ СПбГУВК
– Герасименко, к.т.н.

Антонов И.В., Бруттан Ю.В.
Программирование на языке высокого уровня (C#). Методические указания к лабораторным работам. — Псков, Изд. ППИ, 2007. — 70 с.

Методические указания предназначены для проведения лабораторных работ по предмету «Программирование на языке высокого уровня» (ОПД.Ф.06) студентами факультета Информатики Псковского государственного политехнического института специальностей 230101 "Вычислительные машины, комплексы, системы и сети", 230201 "Информационные системы и технологии". Они могут быть использованы студентами технических специальностей других вузов и направлений при изучении дисциплин программного профиля.

В методических указаниях изложены основы программирования под Windows в среде Microsoft Visual Studio 2005 на языке высокого уровня C#. Материал методических указаний содержит задания к 7 лабораторным работам, справочные материалы по теоретическим аспектам выполнения заданий и подробные рекомендации по выполнению лабораторных работ.

© Псковский государственный
политехнический институт, 2007.
© Антонов И.В., Бруттан Ю.В., 2007.

СОДЕРЖАНИЕ

стр.

ВВЕДЕНИЕ

C# – один из языков программирования высокого уровня, который может использоваться для создания приложений на платформе .NET. Платформа .NET — многоязыковая среда, открытая для свободного включения новых языков, создаваемых не только Microsoft, но и другими фирмами. Все языки, включаемые в платформу .NET, должны опираться на единый каркас, роль которого играет .NET Framework. Главным достоинством языка C# можно назвать его согласованность с возможностями .NET Framework и вытекающую отсюда компонентную ориентированность. Компоненты позволяют решать проблему модульного построения приложений на новом уровне.

C# – результат эволюции языков C и C++. Являясь новейшей разработкой компании Microsoft, C# конструировался очень тщательно, с учётом наилучших возможностей других языков, предназначенных для решения специфических проблем. C# является мощным языком программирования и имеет массу преимуществ: простота, объектная ориентированность, типовая защищенность, сборка мусора, поддержка совместимости версий и многое другое. Данные возможности позволяют быстро и легко разрабатывать приложения.

ЛАБОРАТОРНАЯ РАБОТА №1

“Создание Windows-приложений. Обработка событий мыши. Вывод текста в окно”

Задание

1. Ознакомиться со средой проектирования Visual Studio.NET
2. Создать простейшее приложение средствами Studio.NET.
3. Изменить размеры окна приложения, цвет фона и заголовок.
4. Реализовать обработку щелчка левой и правой кнопки мыши:

- По левой кнопке - выводить координаты курсора мыши в точке его нахождения.
- По правой – отображать диалоговое окно с сообщением «**Нажата правая кнопка мыши**» и очищать окно приложения от надписей.

Краткая справка

В состав среды проектирования Microsoft Visual Studio.NET встроены средства, облегчающие программисту разработку приложений. Данная среда позволяет быстро создавать шаблоны новых приложений. При этом программисту не приходится писать ни одной строчки кода. Достаточно ответить на ряд вопросов, касающихся того, какое приложение требуется создать, и исходные тексты шаблона приложения вместе с файлами будут готовы.

Генерируемый средой Visual Studio.NET каркас стандартного приложения Windows содержит ряд классов. Класс формы (**Form1**) предназначен для создания интерфейса и программирования функциональности приложения. Соответственно, прежде всего с ним работает разработчик приложения Windows.

С формой можно работать **в режиме дизайна** и **в режиме кодирования**. В режиме дизайна доступна панель настраиваемых свойств формы (**Properties**), которая активизируется через команду меню, вызываемого по щелчку правой кнопкой мыши в области формы.

Используя свойства, доступные в панели **Properties**, можно определить положение, размер, цвет и особенности управления для создаваемого окна.

Свойство *Text* позволяет изменить заголовок окна.

Свойства *Size* и *DesktopLocation* задают размер и положение окна при его отображении.

Свойство *ForeColor* служит для указания цвета переднего плана по умолчанию для всех элементов управления, размещенных в форме.

Свойства *BorderStyle, MinimizeBox и MaximizeBox* определяют, можно ли будет свернуть, развернуть или изменить размер формы во время исполнения программы.

Для каждого изменяемого свойства, создается соответствующий код, который помещается в файл **Form1.Designer.cs,** в секцию, отмеченную как **Windows Form Designer generated code**. Можно раскрыть ее и просмотреть сгенерированный код.

Программы Windows, основанные на графическом интерфейсе пользователя, управляются событиями. C# выполняет функции обработки сообщений с помощью специальных функций–делегатов. Помимо методов и свойств формы содержат коллекции обработчиков событий. Если требуется обрабатывать событие, то вы должны выбрать в панели свойств формы закладку «**Events**» и двойным щелчком по пустой ячейке, расположенной справа от имени интересующего события, сгенерировать шаблон функции-обработчика события.

События формы и элементов управления

Таблица 1

Событие	**Описание**
Click	Возникает при щелчке мыши
DoubleClick	Возникает при двойном щелчке мыши
KeyDown	Возникает при нажатии клавиши
KeyUp	Возникает при отпускании клавиши
MouseDown	Происходит, когда нажимается кнопка мыши, а указатель мыши находится над объектом
MouseEnter	Возникает, когда указатель мыши попадает в область объекта
MouseLeave	Возникает, когда указатель мыши покидает область объекта
MouseMove	Возникает при перемещении мыши над объектом
MouseUp	Возникает при отпускании кнопки мыши в области объекта
Resize	Возникает при изменении размера объекта

Обработчик события получает два аргумента – ссылку на объект, к которому относится событие, и дополнительную информацию о событии (тип **EventArgs** или производный от него).

Обработчики событий от мыши получают в качестве второго аргумента объект типа **MouseEventArgs**.

Свойства, определенные в классе MouseEventArgs

Таблица 2

Button	Позволяет получить информацию о том, какая кнопка мыши нажата (в виде значений перечисления MouseButtons)
Clicks	Позволяет получить информацию о том, сколько раз нажата и отпущена кнопка мыши
Delta	Позволяет получить информацию (в виде положительного или отрицательного числового значения) о повороте колесика мыши
X	Позволяет получить координату X для указателя мыши во время щелчка
Y	То же самое для координаты Y

Операции рисования и вывода текста инкапсулирует класс **System.Drawing.Graphics**. В нём присутствуют методы для отображения линий, кривых, строк и других графических элементов.

Для выполнения вывода на экран в методах класса формы требуется предварительно получить объект **Graphics**. Это обеспечивается добавлением в метод следующей строки:

Graphics g = CreateGraphics();

Текстовые строки в C# хранятся в объектах типа **string**.

Для вывода текста в окно необходимо объявить переменную этого типа, проинициализировать её и передать методу **DrawString** класса Graphics.

Например:

```
string s = "Hello, World!";
g.DrawString(s, new Font("Times New Roman", 8), new
SolidBrush(Color.Black), new Point(100, 200));
```

В операции вывода строки дополнительно указываются три объекта типов **Font, SolidBrush** и **Point**, задающие соответственно шрифт, используемый для вывода, цвет шрифта и координаты точки привязки выводимого текста.

Со строками типа string можно выполнять операцию сложения, задаваемую знаком "+".

Числовые типы языка C# можно переводить в строковое представление вызовом для них функции **toString**.

Например:

string s = e.X.ToString();

Рекомендации по выполнению задания

Пункт 2

Создайте проект типа **Windows Application**, выбрав пункт меню **File | New | Project | Visual C# | Windows**. В поле **Location** окна «**New Project**» укажите путь к папке, рекомендованной для сохранения ваших проектов. Нажмите кнопку "**OK**".

Пункт 3

В панели свойств формы задать значения свойств **BackColor**, **Size**, **Text**. Следует присвоить этим полям значения, задающие соответственно белый цвет, размер 600x450 и имя, которое вы считаете подходящим для разрабатываемого вами графического редактора, функции которого будут реализовываться в дальнейших лабораторных работах.

Пункт 4

Добавить к классу **Form1** обработчик события **MouseDown** в соответствии с информацией, изложенной в справке к работе.

Для различения нажатий на левую и правую кнопки мыши следует проверять значение поля **Button** аргумента **MouseEventArgs**. Если оно равно **MouseButtons.Left**, то была нажата левая кнопка, если **MouseButtons.Right** – правая.

Пример выражения проверки:

if (e.Button == MouseButtons.Left)

Диалоговое окно с сообщением создаётся с помощью функции **MessageBox.Show**(), которой передаётся два аргумента: строка выводимого в окне текста и строка заголовка окна.

Для очистки окна следует вызвать через объект типа Graphics функцию **Clear()**, передав ей в качестве аргумента значение цвета **Color.White**.

Лабораторная работа №2

“Создание многооконного приложения, имеющего меню. Рисование прямоугольников под управлением мыши”

Задание

1. Сделать окно приложения MDI-контейнером.
2. Создать меню приложения, содержащее на верхнем уровне пункт «**Окно**», а в распахивающемся списке команду «**Новое**» и список открытых окон. Реализовать обработку команды создания нового окна.
3. Реализовать рисование на экране прямоугольников под управлением мыши. При нажатии левой кнопки мыши и ее удержании при перемещении мыши потенциальный прямоугольник должен отображаться пунктиром, а при отпускании кнопки мыши прямоугольник должен выводиться сплошной линией. Должно рисоваться произвольное число прямоугольников.

Краткая справка.

MDI (Multiple-document interface) приложения позволяют отображать несколько документов одновременно. Такая организация приложения является типичной для редакторов документов различных форматов. При этом каждый документ будет отображаться в своем собственном окне. Обычно MDI приложения имеют в основном меню подпункты для переключения между окнами. Основным окном MDI приложения является родительская форма. Она может содержать несколько дочерних окон. Только одно из дочерних окон может быть активно в один момент времени.

Программы в среде Windows могут иметь основное меню. Меню приложения позволяет создавать иерархию вложенных друг в друга команд меню любой степени сложности. С пунктами меню связываются реализуемые в классе формы методы-обработчики команд меню. Меню могут создаваться и назначаться формам под управлением программного кода и в режиме дизайна формы с использованием интерактивного редактора меню.

Для рисования графических объектов в Windows приложениях платформа .NET использует библиотеку GDI+

Пространство имен **Drawing** содержит множество объектов, которые облегчают программисту работу графикой. GDI+ включает возможности рисования простейших объектов (линии, эллипсы...), рисование различных объектов 2D графики, отображение файлов различных графических форматов (bmp, jpeg, gif, wmf, ico, tiff...) и многое другое.

Большинство функций рисования являются методами класса **Graphics**. Для рисования нужно создавать объект типа Graphics вызовом

функции **CreateGraphics().** По завершении использования объекта Graphics программа также должна вызвать его метод **Dispose**.

Для рисования линий и контуров фигур в GDI+ используется объект перо (тип **Pen**). При создании пера можно задать его цвет и ширину:

Например:

Pen pen = new Pen(Color.Black, 1);

Прямоугольники рисует метод класса Graphics **DrawRectangle**. Ему передаются объекты прямоугольник и перо (**Pen** и **Rectangle**).

Прямоугольник можно создать, вызвав метод класса Rectangle **FromLTRB**, которому передаются четыре числа, задающие координаты левого верхнего и правого нижнего углов прямоугольника:

Например:

Rectangle r = Rectangle.FromLTRB(x1, y1, x2, y2);

Рекомендации по выполнению задания

Пункт 1

Для создания MDI-приложения откройте проект, созданный в первой работе и установите для формы свойство **IsMdiContainer** в **true**, это будет определять форму как родительское окно MDI приложения.

Задайте через свойства формы размер родительского окна **1000x700.**

Пункт 2

Для создания меню приложения Visual Studio .NET имеет в панели **ToolBox** компонент **MenuStrip**. Добавьте на форму родительского окна компонент MenuStrip. В панели компонентов ниже основной формы приложения появится объект **MenuStripl**. В верхней части формы появится проект меню с единственным полем «**Type Неге**». Поле является редактируемым, если вы измените надпись в поле, то справа и снизу от него появятся дополнительные поля. Добавьте в меню верхнего уровня пункт «**Window**» и в его подменю пункт «**New**». В панели свойств этих пунктов меню в поле «**Text**» введите русские названия этих пунктов меню – соответственно «**Окно**» и «**Новое**».

Для свойства **MainMenuStrip** в панели свойств формы выберите значение **menuStrip1.**

Для реализации окон документов MDI-приложения требуется добавить к проекту соответствующую новую форму (**Form2**). Для этого введите команду «**Project**»-«**Add Windows Form**». Для созданной формы документа задайте размеры (**800x600**) и белый цвет фона.

Укажите, что в меню Window следует отображать список созданных окон. Для этого зайдите в панель свойств меню (Properties – menuStrip1) и там для свойства **MdiWindowListItem** выберите значение **WindowToolStripMenuItem.**

Сгенерируйте обработчик команды меню «Окно»-«Новое» двойным щелчком по пункту «Новое» на странице дизайна формы. В тело обработчика поместите следующий код создания и отображения нового окна документа:

```
Form f = new Form2();
f.MdiParent = this;
f.Text = "Рисунок " + this.MdiChildren.Length.ToString();
f.Show();
```

Пункт 3

Для решения задачи по рисованию прямоугольников в классе **Form2** потребуется обрабатывать события мыши **MouseDown**, **MouseMove**, **MouseUp**. Добавьте соответствующие методы-обработчики.

Затем требуется добавить поля данных в класс Form2:

- для хранения координат рисуемого прямоугольника;
- объекта Graphics;
- флага нажатой кнопки мыши (признака процесса рисования).

При нажатии на левую кнопку следует:

- инициализировать объект Graphics;
- устанавливать флаг рисования;
- инициализировать начальные координаты прямоугольника.

При перемещении мыши следует:

- проверять флаг рисования,
- и если он установлен, то рисовать промежуточный пунктирный контур прямоугольника, предварительно стирая контур, нарисованный в предыдущем вызове этого обработчика. (Для стирания контура можно рисовать поверх него прямоугольник цветом фона).

При отпускании левой кнопки мыши следует:

- проверять флаг рисования,
- и если он установлен, то стирать последний промежуточный контур,
- рисовать окончательный контур прямоугольника обычным пером,
- сбрасывать флаг рисования.

Стиль пунктирного пера для промежуточного рисования прямоугольников задаётся следующим образом:

```
pen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
```

Белое перо для стирания промежуточных контуров может быть создано так:

```
new Pen(Color.White, 1)
```

Следует учесть, что методу **DrawRectangle** должен передаваться нормализованный прямоугольник, у которого значения координат первой точки всегда меньше значений соответствующих координат второй точки. Поскольку мышь в процессе рисования может оказаться левее и выше точки нажатия на левую кнопку, её текущие координаты нельзя непосредственно использовать в качестве координат второй точки, задающей прямоугольник. Перед передачей четырёх координат в метод Rectangle.FromLTRB их следует предварительно упорядочивать, используя вспомогательные переменные, таким образом, чтобы в первой паре аргументов передавались всегда меньшие значения координат.

ЛАБОРАТОРНАЯ РАБОТА №3

“Проектирование иерархии классов. Использование контейнерных классов. Перерисовка графической информации”

Задание

1. Спроектировать базовый класс для моделирования различных геометрических фигур.
2. Спроектировать класс, моделирующий прямоугольник, как производный от класса, спроектированного в пункте 1 задания.
3. Реализовать сохранение информации об изображении в динамическом массиве объектов, имеющих тип класса, спроектированного в пункте 1 задания.
4. Реализовать перерисовку графического изображения, создаваемого в программе.

Краткая справка.

Классы являются шаблонами, на основе которых создаются объекты. Каждый объект содержит данные и методы для работы с этими данными. Класс определяет, какие данные может содержать каждый объект этого класса, но не содержит самих данных. При создании экземпляра объекта класса его поля заполняются конкретными данными.

Поля данных класса представляют собой любые переменные, связанные с классом. Если поля и функции объявлены как **public**, они доступны за пределами класса.

После того как будет создан экземпляр объекта, к полям класса можно осуществлять доступ с помощью синтаксиса **объект.имя_поля.**

Новые классы могут наследовать содержание (данные и методы) уже существующих классов. Такой новый класс называется **производным.** Класс, от которого он наследует, называется **базовым**. На основе наследования создаются иерархии классов. Проектирование иерархий классов – основа методологии объектно-ориентированного программирования.

Абстрактные классы содержат абстрактные методы – методы без реализации. Экземпляры таких классов не могут создаваться, но абстрактные классы часто используются в качестве базовых классов для иерархий классов, так как позволяют задать типовой набор функций, обязательный для реализации в производных классах. Такой абстрактный класс содержит обобщённую модель классов определённого назначения и ссылки на его тип могут использоваться в функциях, которым

впоследствии будут передаваться экземпляры различных производных от него классов.

Контейнерные классы используются для хранения и обработки наборов объектов. Наиболее простой структурой данных такого рода является обычный массив. **Массив** в C# является экземпляром класса **System.Array**. System.Array обеспечивает эффективный доступ к отдельному элементу по его индексу. Однако Array обладает недостатком, заключающимся в необходимости указания его размера при создании экземпляра класса. После его создания не существует возможности добавлять, вставлять или удалять элементы. .NET поддерживает множество других контейнерных классов, которые полезны в различных обстоятельствах.

За исключением System.Array, все классы структур данных располагаются в пространстве имён System.Collections. В .NET 2.0 добавилось пространство имён **System.Collections.Generic**, в котором находятся параметризованные реализации контейнерных классов. Таким классам при создании в качестве параметра, передаваемого в угловых скобках, указывается тип данных, который будет храниться в контейнере. В число этих классов входит класс **List**, реализующий **динамический массив** (массив переменного размера) для данных определённого типа. Если при инициализации такого массива указать в качестве параметра определённый класс, то массив можно будет использовать для хранения объектов этого класса и классов, производных от него.

Класс **List** может быть проинициализирован следующим образом для хранения динамического массива экземпляров любых классов, производных от класса **Figure**:

```
List<Figure> array;
array = new List<Figure>();
```

После этого в контейнер можно добавлять элементы при помощи метода **Add**(), которому передаётся ссылка на экземпляр добавляемого объекта.

Число элементов, в текущий момент содержащихся в List, может быть получено с помощью свойства **Count**:

```
int  n =  array.Count;
```

Для перебора всех элементов динамического массива удобно использовать оператор языка C# **foreach** :

```
foreach (Figure f in array) { ... }
```

Блок в фигурных скобках выполняется для всех элементов **array**, ссылка на которые поочерёдно помещается в переменную (в данном

случае - **f**), доступную внутри блока. Для доступа к элементам List можно использовать и обычную индексную операцию – **[]**.

Операционная система Windows и среда .NET не обеспечивают приложениям сохранение графической информации, выводимой ими в свои окна. В ситуациях перекрытия этих окон другими окнами и их сворачивания ранее выведенная информация будет исчезать, если приложение не обеспечит восстановление содержания окна. Для этих целей предназначено событие **Paint**, которое должны обрабатывать классы форм и управляющих элементов для восстановления своего содержания. Данное событие генерируется, когда содержание окна должно быть обновлено, в первый раз – при первом отображении окна. Обработчик события Paint получает второй аргумент типа **PaintEventArgs**, поле **Graphics** которого содержит объект типа Graphics, через который должны выполняться в этом обработчике операции графического вывода.

Graphics в обработчике события Paint не должен запрашиваться через метод CreateGraphics.

Рекомендации по выполнению задания

Для создания новых классов в проектах Visual Studio.NET можно использовать команду «**Project**» –> «**Add class…**» . Visual Studio предлагает указать имя файла, в котором будет сохранён новый класс с соответствующим именем.

Пункт 1

Базовый класс для иерархии геометрических фигур рекомендуется сделать абстрактным классом.

При этом заголовок класса может выглядеть следующим образом:

abstract class Figure

Класс должен как минимум содержать:

- данные, хранящие информацию о двух точках, задающих размеры блока геометрической фигуры;
- конструктор класса, получающий эти точки в качестве аргументов;
- абстрактные методы **Draw, DrawDash, Hide**, которые должны будут обеспечивать рисование, рисование пунктиром и стирание фигур соответственно.

Абстрактные методы задаются с ключевым словом **abstract** и не содержат тела метода.

В производных классах эти методы будут получать реализацию, соответствующую типу конкретной фигуры. При этом они объявляются с использованием ключевого слова **override**.

Пример объявлений абстрактного метода в базовом классе и его реализации в производном классе:

public abstract void Draw(Graphics g);

public override void Draw(Graphics g) {…}

Функции рисования должны получать в качестве аргумента объект Graphics, который им должна передавать форма, в которой будет рисоваться фигура.

Также в классе Figure можно уже поместить реализацию метода **MouseMove**, который получая от формы Graphics и координаты мыши, будет перемещать фигуру в новое положение, последовательно стирая её в исходном положении, изменяя координаты второй точки, задающей фигуру, и отображая фигуру в новом положении пунктиром.

Также можно поместить в этот класс вспомогательный метод для нормализации координат прямоугольника, которая может требоваться для корректного вызова функций рисования. Чтобы метод в C# мог изменять значения переменных, переданных ему через аргументы, при объявлении этих аргументов и их передаче в метод должен использоваться префикс **ref**, обеспечивающий передачу аргумента по ссылке, а не по значению.

Например, заголовок функции нормализации двух пар координат может быть следующим:

public void norm(ref int x1, ref int y1, ref int x2, ref int y2)

Пункт 2

Класс, моделирующий прямоугольник, должен быть объявлен производным от класса фигуры и должен содержать соответствующую реализацию абстрактных методов, объявленных в классе фигуры.

Конструктор класса должен получать координаты углов прямоугольника и передавать их классу точки. Передача аргументов конструктору базового класса выполняется с использованием ключевого слова **base**.

Например:

public Rect(Point point1, Point point2) : base(point1, point2) {}

Пункт 3

Динамический массив следует сделать полем класса Form2 и проинициализировать это поле в конструкторе класса Form2.

При нажатии на левую кнопку мыши должен создаваться экземпляр класса прямоугольник.

При перемещении мыши он должен использоваться для прорисовки перемещения.

При отпускании левой кнопки должен рисоваться окончательный вид прямоугольника и он должен добавляться к динамическому массиву.

Код класса формы следует очистить от всех деталей, связанных с рисованием прямоугольников и обработкой их координат. Он должен лишь управлять объектами, выполняющими эти действия.

Пункт 4

Для восстановления информации в окне следует реализовать в классе Form2 обработчик события Paint. Этот метод должен перебирать все элементы динамического массива и перерисовывать их.

После этого в конец блока, обрабатывающего добавление нового прямоугольника в массив, следует добавить вызов метода **Invalidate()**, который также инициирует перерисовку окна. Этот вызов обеспечит устранение дефектов рисунка, которые могут появляться в процессе добавления новых элементов к рисунку.

ЛАБОРАТОРНАЯ РАБОТА №4

“Сохранение документов в файлах”

Задание

1. Добавить в меню программы команды сохранения и открытия файлов. Реализовать сохранение рисунков в файлах и чтение их из файлов. После сохранения файла заголовок окна соответствующего документа должен содержать имя файла. При открытии файла его имя должно использоваться в качестве заголовка окна. Пока нет ни одного открытого окна документа, команды сохранения файла должны быть в заблокированном состоянии.

2. Для измененных документов при закрытии окна следует выдавать диалоговое окно с запросом о сохранении документа и обрабатывать три варианта выбора пользователя – сохранение, отказ от сохранения и отказ от закрытия окна (**Yes-No-Cancel**).

Краткая справка

Классы .NET для операций с файловой системой находятся в основном в пространстве имен System.IO.

.NET включает также ряд классов и интерфейсов в пространстве имен System.Runtime.Serialization, которые связаны с сериализацией, то есть, процессом преобразования некоторых данных (например, содержимого документа) в поток байтов для хранения в файле или другом объекте, к которому применимы операции последовательного ввода-вывода. Чтобы обеспечить поддержку сериализации в собственных классах достаточно назначить им атрибут [**Serializable()**]. Атрибуты в C# задают дополнительные и особые свойства элементам языка и указываются в квадратных скобках непосредственно перед тем элементом, к которому они применяются. Если классу назначен этот атрибут, все его поля данных будут по умолчанию сохранятся в файле или в другом потоковом объекте при передаче объекта этого класса методу сериализации. Чтобы исключить определённые поля класса, например, временные и вспомогательные переменные, из операции сериализации, этим элементам следует назначить атрибут [**NonSerialized()**]

Например, сохранение объекта **obj** некоторого класса **X** в файле с именем **fileName** с использованием сериалиации может быть выполнено следующим образом:

```
BinaryFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream(fileName, FileMode.Create,
FileAccess.Write, FileShare.None);
formatter.Serialize(stream, obj);
```

stream.Close();

А восстановление сохранённого объекта из файла:

BinaryFormatter formatter = new BinaryFormatter();

Stream stream = new FileStream(fileName, FileMode.Open, FileAccess.Read, FileShare.Read);

X obj = (X)formatter.Deserialize(stream);

stream.Close();

Использование классов, применяемых при сериализации, требует подключения к модулю следующих пространств имён:

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

Класс **List** поддерживает сериализацию и при её выполнении в потоковом объекте сохраняются все элементы динамического массива. Но чтобы этот механизм работал, требуется, чтобы сами элементы массива также поддерживали сериализацию. Соответственно, классы, объекты которых используются в качестве элементов динамического массива, должны быть сделаны сериализуемыми.

При обработке команд меню, связанных с файловыми операциями, часто используются классы .NET, которые обеспечивают создание стандартных диалоговых панелей Windows, обеспечивающих выбор файла для открытия и сохранения. Это классы **OpenFileDialog** и **SaveFileDialog**. С примерами их использования можно ознакомиться в справочной системе Visual Studio .NET.

Рекомендации по выполнению задания

Пункт 1

Следует реорганизовать меню программы. На верхнем уровне меню слева должен находится пункт «**Файл**», справа – «**Окно**». В разделе меню «Файл» должны быть команды «**Новый**», «**Открыть**», «**Сохранить**», «**Сохранить как…**». Из раздела «Окно» следует удалить команду «Новое», а её обработчик связать с пунктом меню «Файл»-«Новый». При обработке команд сохранения и открытия файлов следует создавать соответствующие стандартные диалоговые панели и использовать результаты выбора пользователя для сериализации и десериализации массива фигур. Обработчики команд «Сохранить» и «Сохранить как…» должны различаться только обработкой ранее сохранённого файла. В первом случае он должен сохраняться без создания и вывода файлового диалога. Для начальной блокировки этих пунктов меню надо установить их свойство **Enabled** в значение **false**. При создании и закрытии окон

документов следует управлять состоянием этих пунктов меню, блокируя и разблокируя их в соответствии с ситуацией.

Объекты, соответствующие пунктам меню, являются полями класса родительской формы. Из методов дочернего окна можно обратиться к родительской форме через свойство **ParentForm**.

При инициализации файловых диалогов следует указывать в качестве стартового каталога текущий каталог программы. Это можно сделать следующим образом:

saveFileDialog.InitialDirectory = Environment.CurrentDirectory;

В поле **Filter** классов файловых диалогов следует задавать шаблон, соответствующий имени вашего редактора и расширению файлов рисунков, создаваемых в нём.

Чтобы обратиться в коде родительской MDI-формы к полям сохраняемой дочерней формы, можно использовать свойство **this.ActiveMdiChild**, содержащее ссылку на активное окно документа. Для изменения заголовка окна можно в программе изменять значение свойства формы **Text**.

Пункт 2

Для обработки попытки закрытия окна формы следует в дочерней MDI-форме реализовать обработчик события **FormClosing**. Окно запроса о сохранении документа следует выдавать только при наличии изменений в документе. То есть, оно не должно выдаваться при закрытии пустого окна или окна, отображающего документ, прочитанный из файла или записанный в файл и не содержащий изменений. Для отслеживания соответствующих состояний документа следует добавить в класс формы переменную–флаг модификации документа. Чтобы метод **MessageBox.Show** создавал окно с тремя кнопками, в качестве третьего аргумента ему следует передавать константу **MessageBoxButtons.YesNoCancel**. Метод в данном случае может возвращать константы **DialogResult.Yes**, **DialogResult.No**, **DialogResult.Cancel**.

Отказ от закрытия формы (выбор Cancel) должен приводить к установке в **true** свойства **Cancel** второго аргумента обработчика события **FormClosing**.

ЛАБОРАТОРНАЯ РАБОТА №5

“Управление атрибутами графического вывода. Использование окон диалога”

Задание

1. Дополнить модель фигуры атрибутами цвета линии, цвета фона фигуры, толщины линии. Реализовать сохранение этих атрибутов в файлах. Использовать эти атрибуты при рисовании прямоугольников.

2. Добавить в меню верхнего уровня пункт "**Параметры**". В подменю поместить пункты "**Цвет линии**", "**Цвет фона**", "**Толщина линии**".

3. Реализовать создание соответствующих диалоговых панелей для обработки новых команд меню и использование результатов ввода в программе. В диалоговом окне выбора размера пера использовать комбинированный список с полем ввода (**ComboBox**), содержащий список фиксированных размеров(**1, 3, 5, 8, 10, 12, 15**) .

Выбранные в диалоговых панелях параметры должны использоваться при рисовании фигур во всех открытых в редакторе окнах. *По умолчанию при запуске программы должны использоваться чёрный цвет линии, белый цвет фона и единичная толщина линии*.

Краткая справка

Диалоговое окно — это форма, обладающая особыми характеристиками. Для диалоговых окон, как правило, нельзя изменять их размер. Кроме того, в диалоговых окнах обычно не используются элементы управления, помещаемые в верхнюю часть обычных форм: ControlBox, MinimizeBox и Maximize Box. Для открытия модального диалогового окна (это диалоговое окно, которое должно быть обязательно закрыто для возвращения к исходной форме) используется метод **ShowD1alog().**

Код для открытия диалогового окна может выглядеть следующим образом:

```
DialogForm myDialog= new DialogForm();
myDialog.ShowDialog(this);
```

Метод **ShowDialog()** возвращает переменную **DialogResult**, которую можно использовать для проверки того, на какую кнопку нажал пользователь, если имеется несколько кнопок.

Для создания диалогового окна необходимо добавить форму в проект. Затем для формы устанавливаются свойства:

FormBorderStyle: FixedDialog

MaximizeBox: false
MinimizeBox: false
ControlBox: false
StartPosition: CenterParent

Для размещения в диалоговой панели элементов управления следует в режиме дизайна формы отобразить панель компонентов форм («**View**»–>«**Toolbox**») и перетаскивать с неё мышью нужные компоненты на поверхность формы. Для размещённого таким образом элемента формы можно, используя мышь, редактировать его размеры и положение. Другие параметры управляющих элементов настраиваются через панель свойств, которую можно вызывать через контекстное меню, активизируемое по правой кнопке мыши.

Основной элемент управления, размещаемый в диалоговых окнах — кнопка. Для активизации (генерации события) кнопку можно либо щелкнуть мышью, либо, если фокус ввода находится на кнопке, нажать пробел. **Кнопка** реализована в классе **Button.**

Комбинированные списки (объекты **ComboBox**) позволяют пользователю производить выбор из списка заранее определенных элементов. При этом пользователь может не только выбрать готовое значение из списка, но и ввести свое собственное.

Свойство **Text** позволяет получить доступ к тексту в поле редактирования. При работе с ComboBox это свойство используется чаще всех остальных. Свойство **Items** позволяет задать список заранее определенных элементов.

Среда .NET содержит реализацию ряда стандартных диалоговых панелей. Диалог выбора цвета инкапсулирует класс **ColorDialog**. Чтобы вывести диалоговую панель выбора цвета на экран, необходимо создать объект этого класса и вызвать для него метод **ShowDialog**. После отображения панели на экране пользователь может выбрать из нее цвет и нажать кнопки OK или Cancel для подтверждения выбора цвета или отказа от него. Когда диалоговая панель закрывается, метод ShowDialog возвращается значения **DialogResult.OK** и **DialogResult.Cancel** в зависимости от того, какую кнопку нажал пользователь.

Для определения цвета, выбранного пользователем, после закрытия диалоговой панели можно обратиться к свойству **Color** класса ColorDialog.

Рекомендации по выполнению задания

Пункт 1

Рекомендуется поля цвета и размера пера поместить в класс фигуры и добавить в конструкторы классов фигуры и прямоугольника соответствующие аргументы, используемые для инициализации данных полей.

Сохранение новых полей в файлах будет обеспечено автоматически за счёт работы механизма сериализации.

Класс прямоугольника должен быть также модифицирован для обеспечения возможности рисования прямоугольников с произвольной толщиной и цветом линии и имеющих фон заданного цвета. Для этого при вызове конструктора пера, используемого для рисования итогового контура прямоугольника, следует передавать ему определённый размер и цвет. Для заливки фона прямоугольника следует до рисования контура прямоугольника вызывать метод **FillRectangle**, которому передаётся кисть (тип **SolidBrush**) нужного цвета.

Пункт 2

Для реализации этого пункта задания потребуется спроектировать собственную диалоговую панель выбора толщины линии. Для этого к проекту следует добавить новую форму (**«Project» - «Add Windows Form…»**) и настроить её свойства в соответствии с рекомендациями из справки к работе. Форме следует дать англоязычное имя, отражающее её назначение. В форме следует разместить две кнопки и комбинированный список. Кнопке «OK» следует задать значение «OK» свойства **DialogResult**, кнопке «Cancel» - значение «Cancel».

Обработчики соответствующих команд меню в классе Form1 должны создавать диалоговые панели и считывать после их закрытия результат выбора пользователя. Как это сделать для стандартных панелей выбора цвета указано в справке к работе. Для получения от собственного диалога выбранной пользователем толщины линии потребуется добавить в класс этого диалога соответствующее свойство или метод (public), возвращающий значение свойства Text элемента Combobox. Узнать имя поля, которое присвоил дизайнер Visual Studio этому элементу формы, можно в файле **<имя_формы>.Designer.cs** или в панели «Class View».

Для преобразования текстового значения этого поля в целочисленное можно вызвать метод **Convert.ToInt32**.

В класс Form1 потребуется добавить поля данных, в которых будут храниться текущие настройки цвета линии, цвета фона фигуры и толщины линии. При создании новых объектов фигур эти поля должны использоваться для инициализации объектов. Значения по умолчанию им следует присвоить в конструкторе класса Form1. Новые значения должны в них записываться в результате обработки команд меню по выбору цвета линии, цвета фона фигуры и толщины линии.

ЛАБОРАТОРНАЯ РАБОТА №6

“Управление размером рисунка. Прокрутка изображения”

Задание

1. Ввести в меню команду настройки размеров для вновь создаваемых рисунков. Вызываемая диалоговая панель должна содержать выбор размера (с использованием radio-кнопок) из трех фиксированных вариантов (**320x240, 640x480, 800x600**), флажок выбора ручного ввода размера, поля ввода ширины и высоты рисунка в пикселях. Выбор размера не должен изменять размеры уже существующих рисунков, а должен определять размеры рисунков, создаваемых в дальнейшем командой «Новый».
2. Рисунок должен отображаться и записываться в файл с учетом своего размера. Фон области формы, находящийся вне рисунка, должен отображаться светло-серым цветом. Если при завершении рисования фигуры будет происходить выход контура фигуры за границы рисунка, соответствующая фигура не должна добавляться к рисунку.
3. Дополнить окна документов полосами прокрутки, обеспечить корректное отображение рисунков при скроллинге.

Краткая справка

RadioButton

Управляющие элементы RadioButton традиционно представлены в виде надписи с точкой, расположенной от них с левой стороны, которые могут быть либо выбраны, либо не выбраны. RadioButton рекомендуется использовать, когда необходимо предоставить пользователю возможность выбора из нескольких взаимоисключающих вариантов. Для того чтобы сгруппировать элементы RadioButton в единую логическую единицу, следует использовать управляющий элемент **GroupBox**. Расположив в форме сначала групповое окно, а затем необходимые элементы RadioButton внутри границ этого группового окна, можно добиться того, что все переключатели будут изменять свое состояние таким образом, что только один из них может быть выбран в рамках данной группы. Если не разместить их внутри группового окна, то это приведет к тому, что только один из них сможет быть выбран в каждый конкретный момент времени во всей форме.

Свойства управляющего элемента RadioButton

Таблица 3

Свойство	Комментарии
Appearance	Управляющий элемент RadioButton может выводиться в виде надписи с круглым флажком выбора, расположенным слева, в середине или

Свойство	Комментарии
	справа от нее, либо в виде стандартной кнопки. В последнем случае кнопка выглядит утопленной, если она выбрана, и выпуклой — если нет
AutoCheck	Когда данному свойству присвоено значение true, то в момент щелчка мышью на переключателе выводится флажок выбор
CheckAlign	Позволяет изменять местоположение переключателя. Он может располагаться справа, по центру или слева. Определяет состояние управляющего элемента
Checked	Значение равно true, если у данного управляющего элемента установлен флажок выбора, а иначе false

События управляющего элемента RadioButton

Таблица 4

Событие	Комментарии
Checkchanged	Сообщение о наступлении этого события отправляется в тот момент, когда флажок выбора RadioButton изменяет свое состояние. Если в рамках формы или группового окна существует более одного управляющего элемента RadioButton, то сообщение об этом событии будет отправляться дважды: сначала управляющему элементу, который до этого был выбран и теперь таковым не является, а затем управляющему элементу, который выбран в настоящий момент.
Click	Сообщение о наступлении этого события отправляется всякий раз, когда происходит щелчок мышью на RadioButton. Это не то же самое, что событие, связанное с изменением состояния управляющего элемента, поскольку щелчок мышью на каком-либо управляющем элементе два или более раз подряд приводит только к однократному изменению свойства "выбрано" — и то только в том случае, если оно не было выбрано до этого

CheckBox

Управляющий элемент CheckBox традиционно представляет собой надпись с маленьким окошком с флажком, который располагается слева от нее. Этот управляющий элемент следует использовать, когда необходимо

предоставить пользователю возможность выбора одной или нескольких возможностей.

Свойства и события данного управляющего элемента очень напоминают свойства и события RadioButton, однако они обладают двумя новыми свойствами:

Свойства управляющего элемента CheckBox

Таблица 5

Свойство	Комментарии
CheckState	В отличие от RadioButton, CheckBox может находиться в одном из трех состояний: • Checked (выбрано) • Indeterminate (не определено), • Unchecked (не выбрано) Если состояние окна выбора описывается как indeterminate, то окно выбора, соответствующее данной надписи, обычно изображается серым цветом, что означает, что текущее значение в окне выбора либо недопустимо, либо лишено смысла при данных обстоятельствах
ThreeState	Когда это свойство имеет значение false, то нет возможности изменять состояние CheckBox на indeterminate. Хотя при этом по-прежнему можно устанавливать данное значение программным путем

TextBox

Управляющие элементы типа TextBox (текстовое окно) предназначены для ввода пользователем текста. Класс TextBox обеспечивает ряд возможностей по работе с текстом в текстовом окне, такие как выделение текста, вырезание и вставка текста, а также набор событий.

Свойства управляющего элемента TextBox

Таблица 6

Свойство	Комментарии
Causesvalidation	Когда элемент, у которого данному свойству присвоено значение true, должен оказаться в фокусе, то генерируются два события — Validating и Validated. Их можно обработать с целью проверки допустимости данных, введенных в управляющий элемент, который выходит из фокуса
CharacterCasing	Позволяет определить, производится ли изменение регистра вводимых символов. Это

Свойство	Комментарии
	свойство может иметь одно из трех значений: • Lower — весь вводимый в окно текст переводится в нижний регистр. • Normal — текст остается без изменений. • Upper — весь вводимый в окно текст переводится в верхний регистр
MaxLength	Значение, которое определяет максимальную длину текста в TextBox, выраженную в символах. Если длина ограничена только объемом имеющейся в наличии памяти, данному свойству следует присвоить значение, равное нулю
Multiline	Указывает, допускает ли данный управляющий элемент наличие нескольких строк. Многострочный элемент позволяет одновременный вывод нескольких строк текста
PasswordChar	Указывает, что текст, находящийся в однострочном окне, должен заменяться на "секретные" символы. Если свойство Multiline имеет значение, равное true, то это свойство игнорируется
Readonly	Логическое значение, указывающее на то, что текст может использоваться в режиме "только чтение".
ScrollBars	Позволяет указать, что состоящее из нескольких строк окно должно выводиться с линейками прокрутки.
SelectedText	Определяет выбранный в окне текст
SelectionLength	Определяет число символов в выбранном тексте
SelectionStart	Определяет начало выделенного текста в текстовом окне

Управляющий элемент TextBox поддерживает обработку событий, позволяющих организовать проверку допустимости вводимого текста. Если требуется гарантированно не допускать ввода неверных символов или если нужно, чтобы вводимое в текстовое окно значение находилось в допустимом диапазоне, то необходимо указать пользователю этого управляющего элемента, является ли введенное значение допустимым или нет.

Управляющий элемент TextBox предоставляет следующие события:

События управляющего элемента TextBox

Таблица 7

Событие	Комментарии
Enter GotFocus Leave Validating Validated LostFocus	Первые шесть событий, приведенных в таблице наступают в той последовательности, в которой они перечислены. Они известны под названием "событий, связанных с фокусом" и генерируются при любом изменении фокуса управляющего элемента за исключением двух случаев. События Validating и Validated происходят только в том случае, если свойство CausesValidation имеет значение true.
KeyDown KeyPress KeyUp	Группа событий, связанных с клавишами. Они позволяют отслеживать и изменять то, что в вводится в управляющий элемент. События KeyDown и KeyUp получают код, соответствующий нажатой клавише. Это позволяет определять нажатие специальных клавиш, например Shift или Ctrl, в том числе, в комбинации с другими клавишами. KeyPress позволяет получать символ, соответствующий нажатой клавише. Это событие оказывается полезным, если требуется задать множество допустимых символов, например, разрешить ввод только численных значений.
Change	Наступает при любых изменениях текста в текстовом окне независимо от их характера

Полосы прокрутки (scroll bars).

Класс **ScrollableControl**, производным от которого является класс Form, предназначен для обеспечения поддержки вертикальной и горизонтальной полос прокрутки. Наиболее часто используемыми свойствами этого класса являются **AutoScroll** и **AutoScrollMinSize**. Эти свойства обеспечивают автоматическое появление полос прокрутки в тех ситуациях, когда ее содержимое не умещается в границах формы (например, если пользователь уменьшил ее размер). Свойство **AutoScrollMinSize** позволяет задать минимальный размер формы, при котором всегда будут появляться полосы прокрутки. Выглядеть это может так:

// Этот код помещается в конструктор класса или метод

// InitializeComponent().
tnis.AutoScroll = true;
this.AutoScrollMinSize = new System.Drawing.Size (300, 300);

После этого класс ScrollableControl обеспечит появление полос прокрутки и обработку событий прокрутки.

Рекомендации по выполнению задания

Пункт 1

В диалоговой панели выбора размера рисунка следует использовать следующие элементы панели «**Toolbox**»: **Button**, **RadioButton**, **GroupBox**, **CheckBox**, **TextBox**.

Кнопке «OK» следует задать значение «OK» свойства **DialogResult**, кнопке «Cancel» – значение «Cancel». Следует добавить в класс этой диалоговой панели public поле типа **Size**, которое будет сохранять заданный размер рисунка, и обработчик щелчка по кнопке “OK”, который должен считывать из элементов панели заданный размер (с учётом состояния флажка ручного ввода размера) и сохранять его в указанной переменной.

Реализовать обработчик события **CheckedChanged** (изменение состояния флажка ручного ввода размера), в котором через свойство **Enabled** элементов **GroupBox и TextBox** следует управлять доступностью этих элементов для ввода и выбора в соответствии с состоянием флажка.

Обработчик соответствующей команды меню в классе **Form1** должен создавать диалоговую панель выбора размера и считывать после её закрытия результат выбора пользователя.

Пункт 2

При создании новых окон рисунков следует задавать для них размер рисунка и учитывать его при выполнении операций с рисунком. Для этого надо добавить соответствующее поле к классу формы рисунка и реализовать конструктор этого класса, получающий в качестве аргумента размер рисунка. Этот конструктор должен вызываться из обработчика команды меню, создающей новое окно.

Для цвета нерабочей области окна рисунка (фон формы) в свойствах формы следует задать серый цвет (например, ControlDark).

Обработчик события Paint должен перед рисованием фигур рисунка заполнять рабочую область окна рисунка белым цветом (метод **FillRectangle**).

Обработчик отпускания кнопки мыши должен проверять условие попадания создаваемой фигуры в область рисунка и добавлять к рисунку только те фигуры, которые соответствуют этому условию.

При сохранении рисунка в файлах и чтении из файлов операции сериализации/десериализации массива фигур должны дополниться соответствующими действиями с размером рисунка.

Пункт 3

Для обеспечения обработки скроллинга следует установить в **true** свойство формы **AutoScroll** и задавать в конструкторе формы значение свойству **AutoScrollMinSize**, которое должно соответствовать размеру рабочей области окна рисунка.

При создании объектов фигур и их рисовании теперь следует учитывать, что в окнах со скроллингом координаты мыши, передаваемые обработчикам событий, и координаты, передаваемые методам класса Graphics, отсчитываются от левого верхнего угла видимой области окна, а не от левого верхнего угла рисунка, отображаемого в окне. Следовательно, координаты, отсчитываемые от верхнего левого угла полного рисунка, должны формироваться из оконных координат в результате их модификации на величину смещения, задаваемого полосами прокрутки. Эта величина доступна через свойство формы **AutoScrollPosition**, хранящее отрицательные значения ненулевых смещений видимой области рисунка по осям X и Y. Соответствующий параметр, позволяющий учесть смещение при скроллинге, потребуется добавить к методам рисования фигур для их корректного отображения в окнах со скроллингом.

ЛАБОРАТОРНАЯ РАБОТА №7

“Развитие иерархии классов. Оптимизация операций рисования”

Задание

1. Спроектировать классы, обеспечивающие рисование и сохранение в файлах помимо прямоугольников также эллипсов, прямых линий и линий произвольной формы. Рисование линии произвольной формы должно начинаться по нажатию на левую кнопку мыши и завершаться по её отпусканию.

2. Дополнить меню программы разделом «**Фигура**», пункты которого должны обеспечивать выбор режима рисования одной из фигур. Пункт меню, соответствующий выбранной фигуре, должен отмечаться галочкой. Для замкнутых фигур (прямоугольника и эллипса) должны поддерживаться режимы прозрачного фона и заливки фона фигуры выбранным цветом фона. Для выбора режима заливки в меню «Фигура» ввести пункт «**Заливка**», который должен работать как флаг, последовательно устанавливаемый и сбрасываемый при его выборе. Активность режима заливки должна отображаться галочкой рядом с пунктом меню.

3. Обеспечить сохранение целостности исходного изображения в процессе добавления новых фигур к рисунку (отсутствие в окне «мусора» и стёртых фрагментов фигур). Обеспечить отсутствие мерцания (мелькания фона) при добавлении новых фигур и изменении размеров окна.

Краткая справка

Прямые линии

Для рисования одиночных прямых служит метод **DrawLine** класса Graphics. Есть четыре перегруженных версии DrawLine, но все они требуют один и тот же набор аргументов: координаты начальной и конечной точек линии, а также перо, которым она рисуется:

void DrawLine(Pen pen, int x1, int y1, int x2, int y2)

void DrawLine(Pen pen, float x1, float y1, float x2, float y2)

void DrawLine(Pen pen, Point pointl, Point point2)

void Drawl_ine(Pen pen, PointF pointl, PointF point2)

Можно указывать координаты как четырьмя значениями типа int или float, так и парой структур Point или PointF.

Метод DrawLine чертит линию от первой точки до второй.

Эллипсы

Для рисования эллипсов служит метод **DrawEllipse**, использующий те же аргументы, что и DrawRectangle:

void DrawEllipse(Pen pen, int x, int y, int cx, int cy)

void DrawEllipse(Pen pen, float, x, float y, float cx, float cy)

void DrawEllipse(Pen pen, Rectangle rect)

void DrawEllipse(Pen pen, RectangleF rectf)

Метод DrawEllipse получает координаты, определяющие углы невидимого прямоугольника, в который вписывается рисуемый этим методом эллипс.

Заливку внутренней области эллипса выполняют методы **FillEllipse** класса Graphics

void FillEllipse(Brush brush, int x, int y, int cx, int cy)

void FillEllipse(Brush brush, float x, float y, float cx, float cy)

void FillEllipse(Brush brush, Rectangle rect)

void FillEllipse(Brush brush, RectangleF rectf)

Линии произвольной формы

В классе Graphics для рисования кривой линии может использоваться **DrawCurve**:

DrawCurve(Pen pen, Point[] apt)

DrawCurve(Pen pen, PointF[] aptf)

Для этого метода нужно задать минимум две точки. Если массив, переданный методу, содержит всего две точки, то DrawCurve соединяет их прямой. Если точек три или больше, метод рисует кривую, соединяющую все точки.

Отмеченные пункты меню

Windows-программы используют галочку для взаимоисключающих пунктов и для пунктов, меню, которые могут находиться в состоянии включено/выключено. Контролировать появление галочек позволяет свойство пункта меню **Checked** (тип bool). Чтобы слева от пункта выводилась отметка, следует задавать **true** свойству **Checked**. Для снятия отметки – false. Для автоматического переключения отметки пункта меню следует установить в true свойство **CheckOnClick.**

Оптимизация графического вывода

Для оптимизации графического вывода в .NET 2.0 используется алгоритм двойной буферизации графической информации. В этом случае отдельные операции, формирующие фон и элементы изображения, выполняются с буферной областью оперативной памяти. Затем полностью сформированное изображение одной операцией переносится в окно. Для активизации режима двойной буферизации в панели свойств формы следует установить в **true** свойство **DoubleBuffered.**

Управление созданием буферов изображений в памяти обеспечивает класс **BufferedGraphicsContext**. Среда .NET автоматически создаёт для приложений экземпляр этого класса. Получить к нему доступ можно через свойство **BufferedGraphicsManager.Current**. Для объекта **BufferedGraphicsContext** перед использованием должен задаваться максимальный размер буфера через присваивание значения его свойству

MaximumBuffer (тип Size). Максимальный размер буфера должен соответствовать максимальному размеру выводимого в окно изображения. Максимальный размер окна при текущих настройках системы можно получить, обратившись к свойству **SystemInformation.PrimaryMonitorMaximizedWindowSize.**

Для выполнения операций рисования в буфере используется объект типа **BufferedGraphics**. Этот класс содержит поле Graphics типа Graphics. Новый объект типа BufferedGraphics создаёт метод **Allocate** класса BufferedGraphicsContext. После использования объекта, занятые им ресурсы должны освобождаться методом **Dispose.**

Перенос сформированного рисунка из буфера в окно осуществляется методом **Render** класса BufferedGraphics, которому передаётся объект типа Graphics, связанный с окном.

Рекомендации по выполнению задания

Пункт 1

Проектирование классов эллипса и прямой линии может быть выполнено аналогично проектированию класса прямоугольника.

Класс линии произвольной формы для хранения множества точек, задающих линию, может использовать динамический массив. Следует учитывать, что методу DrawCurve должен передаваться обычный массив точек, а не динамический массив. Для рисования кривой можно формировать на основе информации из динамического массива буферный массив, координаты точек которого будут учитывать текущее положение полос прокрутки.

Пункт 2

Для управления фигурами разных типов рекомендуется определить перечисление, содержащее идентификаторы этих фигур. Выбор пункта меню, соответствующего определённой фигуре, и состояние флага заливки должны фиксироваться в полях класса основной формы. Затем, при обработке щелчка левой кнопки мыши следует учитывать значение этих параметров и создавать одну из фигур, используя для выбора конструкцию **switch–case.**

Пункт 3

Для формы, содержащей рисунок, установить в **true** свойство **DoubleBuffered**. Для буферизации операций рисования добавить в класс формы, содержащей рисунок, поле типа BufferedGraphics. Это поле не должно инициализироваться в конструкторе класса, так как его инициализация требует наличия объекта Graphics, который должен создаваться лишь для уже существующих и инициализированных окон. Решением проблемы является использование для инициализации этого поля обработчика события **Load** формы. Это событие генерируется уже после создания формы, но перед её первым отображением. Для

освобождения ресурсов, занятых этим объектом, может использоваться обработчик события **FormClosed**. При создании объекта BufferedGraphics можно заполнить область окна цветом фона. Обработчик события Paint после перехода к буферизованному выводу должен заполнять фон рабочей области рисунка и рисовать фигуры, но не в окне, а в буфере, а затем вызовом метода **Render** переносить сформированный рисунок в окно.

Примеры использования класса BufferedGraphics можно найти в справочной системе Visual Studio. В разделе «Index» перейдите к справке по double buffering и там по ссылке «Manually Render Buffered Graphics».

Исключение дефектов изображения, возникающих в процессе рисования, потребует переработки алгоритмов обработки событий, приходящих от мыши. При перемещении контура фигуры вместо стирания контура в предыдущем положении теперь следует вызывать метод Render, который будет восстанавливать исходное состояние рисунка, и после этого обновлять контур.

ЛИТЕРАТУРА

1. К. Ватсон и др. C#.. – СПб.:Питер, 2006.
2. Д.С. Платт. Знакомство с Microsoft .NET / Пер. с англ. –М.: Издательско-торговый дом «Русская редакция», 2001.
3. Ч. Петцольд. Программирование для Microsoft Windows на C#. В 2-х томах / Пер. с англ. –М.: Издательско-торговый дом «Русская редакция», 2002.
4. C#. Программирование на языке высокого уровня / Т.А.Павловская. – СПб.:Питер, 2007.

ПРИЛОЖЕНИЕ

Класс Form

Представляет окно или диалоговое окно, которое составляет пользовательский интерфейс приложения.

Базовый класс: ContainerControl

Пространство имен: System.Windows.Forms

Form является представлением любого окна, отображаемого в приложении. Класс Form используется для создания стандартных окон, окон инструментов, не обрамленных и перемещаемых окон. Класс Form также используется для создания модальных окон, например, диалогового окна. Особым видом формы является форма многодокументного интерфейса MDI (multiple document interface), содержащая другие формы, называемые дочерними MDI-формами. MDI-форма создается с помощью установки свойству IsMdiContainer значения true. Дочерние MDI-формы создаются с помощью установки свойства MdiParent для родительской MDI-формы, которая будет содержать дочернюю форму.

С помощью свойств, доступных в классе Form, имеется возможность определить внешний вид, размер, цвет и компоненты управления создаваемого окна или диалогового окна. Свойство Text позволяет задавать заголовок окна в строке заголовка. С помощью свойств Size и DesktopLocation определяются размер и положение окна при его отображении. Имеется возможность использовать свойство цвета ForeColor для изменения назначенного по умолчанию цвета фона всех элементов управления, помещенных в форму. Свойства FormBorderStyle, MinimizeBox и MaximizeBox позволяют управлять возможностью минимизации, максимизации или изменения размера формы во время выполнения.

Кроме свойств для управления формой используются методы класса. Например, метод ShowDialog используется для отображения формы как модального диалогового окна. Имеет возможность использовать метод SetDesktopLocation для размещения формы на рабочем столе.

События класса Form позволяют отвечать на действия, выполненные с формой. Имеется возможность использовать событие Activated для выполнения таких операций, как обновление данных, отображаемых в элементах управления формы при ее активации.

Форма используется в качестве начального класса приложения с помощью помещения в класс метода, вызванного Main В методе Main необходимо добавить код для создания и отображения формы. Необходимо также добавить атрибут [STAThread] для метода Main для запуска формы. Если начальная форма закрыта, приложение также закрыто.

Открытые конструкторы

Form - конструктор	Инициализирует новый экземпляр класса **Form**.

Открытые свойства

Свойство	Комментарии
AcceptButton	Возвращает или задает кнопку на форме, которая срабатывает при нажатии клавиши ВВОД.
AccessibilityObject (унаследовано от **Control**)	Возвращает AccessibleObject, назначенный элементу управления.
AccessibleDefaultActionDescription (унаследовано от **Control**)	Возвращает или задает описание выполняемого по умолчанию действия элемента управления для использования клиентскими приложениями со специальными возможностями.
AccessibleDescription (унаследовано от **Control**)	Возвращает или задает описание элемента управления, используемого клиентскими приложениями со специальными возможностями.
AccessibleName (унаследовано от **Control**)	Возвращает или задает имя элемента управления, используемого клиентскими приложениями со специальными возможностями.
AccessibleRole (унаследовано от **Control**)	Возвращает или задает доступную роль элемента управления.
ActiveControl (унаследовано от **ContainerControl**)	Получает или задает активный элемент управления в элементе управления контейнера.
ActiveForm	Возвращает текущую активную форму для этого приложения.
ActiveMdiChild	Возвращает дочернее окно текущего активного многодокументного интерфейса (MDI).
AllowDrop (унаследовано от **Control**)	Возвращает или задает значение, указывающее, может ли элемент управления принимать данные, перемещенные на него пользователем.
Anchor (унаследовано от **Control**)	Возвращает или задает значение, указывающее, какие края элемента управления будут привязаны к краям контейнера.

Свойство	Комментарии
AutoScale	Возвращает или задает значение, указывающее, настраивает ли форма свой размер для подгонки высоты шрифта, используемого на форме, и масштабирования элементов управления.
AutoScaleBaseSize	Возвращает или задает базовый размер, используемый для автоматического масштабирования формы.
AutoScroll	Переопределен. Возвращает или задает значение, определяющее состояние режима автопрокрутки формы.
AutoScrollMargin (унаследовано от **ScrollableControl**)	Возвращает или задает размер поля автоматической прокрутки.
AutoScrollMinSize (унаследовано от **ScrollableControl**)	Возвращает или задает минимальный размер для автоматической прокрутки.
AutoScrollPosition (унаследовано от **ScrollableControl**)	Возвращает или задает местоположение позиции автоматической прокрутки.
BackColor	Переопределен. См. Control.BackColor.
BackgroundImage (унаследовано от **Control**)	Возвращает или задает фоновое изображение, выводимое на элементе управления.
BindingContext (унаследовано от **Control**)	Возвращает или задает BindingContext для элемента управления.
Bottom (унаследовано от **Control**)	Возвращает расстояние между нижним краем элемента управления и верхним краем клиентской области контейнера.
Bounds (унаследовано от **Control**)	Возвращает или задает размер и местоположение элемента управления, включая неклиентские элементы.
CancelButton	Возвращает или задает кнопку, которая срабатывает при нажатии клавиши ESC.
CanFocus (унаследовано от **Control**)	Возвращает значение, показывающее, может ли элемент управления получать фокус.

Свойство	Комментарии
CanSelect (унаследовано от **Control**)	Возвращает значение, показывающее, доступен ли элемент управления для выделения.
Capture (унаследовано от **Control**)	Возвращает или задает значение, определяющее, была ли мышь захвачена элементом управления.
CausesValidation (унаследовано от **Control**)	Возвращает или задает значение, показывающее, вызывает ли элемент управления проверку любого элемента управления, требующего проверки при получении фокуса.
ClientRectangle (унаследовано от **Control**)	Возвращает прямоугольник, задающий клиентскую область элемента управления.
ClientSize	Возвращает или задает размер клиентской области формы.
CompanyName (унаследовано от **Control**)	Возвращает название организации или имя создателя приложения, содержащего элемент управления.
Container (унаследовано от **Component**)	Возвращает IContainer, содержащий Component.
ContainsFocus (унаследовано от **Control**)	Возвращает значение, указывающее, имеет ли элемент управления или один из его дочерних элементов фокус ввода.
ContextMenu (унаследовано от **Control**)	Возвращает или задает меню быстрого вызова, связанное с элементом управления.
ControlBox	Возвращает или задает значение, определяющее режим отображения кнопки оконного меню в строке заголовка формы.
Controls (унаследовано от **Control**)	Возвращает коллекцию элементов управления, содержащихся в элементе управления.
Created (унаследовано от **Control**)	Возвращает значение, показывающее, был ли создан элемент управления.
Cursor (унаследовано от **Control**)	Возвращает или задает курсор, отображаемый, когда указатель мыши находится на элементе управления.
DataBindings (унаследовано от **Control**)	Возвращает привязки данных для этого элемента управления.

Свойство	Комментарии
DesktopBounds	Возвращает или задает размер или расположение формы на рабочем столе Windows.
DesktopLocation	Возвращает или задает расположение формы на рабочем столе Windows.
DialogResult	Возвращает или задает результат диалога для формы.
DisplayRectangle (унаследовано от **Control**)	Возвращает прямоугольник, предоставляющий отображаемую область элемента управления.
Disposing (унаследовано от **Control**)	Возвращает значение, показывающее, находится ли элемент управления в процессе удаления.
Dock (унаследовано от **Control**)	Возвращает или задает край родительского контейнера, к которому прикрепляется элемент управления.
DockPadding (унаследовано от **ScrollableControl**)	Возвращает параметры для заполнения стыковки на всех краях элемента управления.
Enabled (унаследовано от **Control**)	Возвращает или задает значение, показывающее, имеет ли элемент управления возможность отвечать на действия пользователя.
Focused (унаследовано от **Control**)	Возвращает значение, показывающее, имеет ли элемент управления фокус ввода.
Font (унаследовано от **Control**)	Возвращает или задает шрифт текста, отображаемого элементом управления.
ForeColor (унаследовано от **Control**)	Возвращает или задает основной цвет элемента управления.
FormBorderStyle	Возвращает или задает стиль границы формы.
Handle (унаследовано от **Control**)	Возвращает дескриптор окна, к которому привязан элемент управления.
HasChildren (унаследовано от **Control**)	Возвращает значение, определяющее, содержит ли элемент управления один или несколько дочерних элементов.
Height (унаследовано от **Control**)	Возвращает или задает высоту элемента управления .

Свойство	Комментарии
HelpButton	Возвращает или задает значение, определяющее, отображается ли кнопка справки в окне заголовка формы.
Icon	Возвращает или задает значок для формы.
ImeMode (унаследовано от **Control**)	Возвращает или задает режим редактора методов ввода (IME) элемента управления.
InvokeRequired (унаследовано от **Control**)	Возвращает значение, показывающее, следует ли вызывающему оператору обращаться к методу invoke во время вызовов метода из элемента управления, так как вызывающий оператор находится не в том потоке в котором был создан элемент управления.
IsAccessible (унаследовано от **Control**)	Возвращает или задает значение, показывающее, является ли элемент управления видимым для приложений со специальными возможностями.
IsDisposed (унаследовано от **Control**)	Возвращает значение, показывающее, был ли удален элемент управления.
IsHandleCreated (унаследовано от **Control**)	Возвращает значение, показывающее, имеется ли у элемента управления связанный с ним дескриптор.
IsMdiChild	Возвращает значение, определяющее, является ли форма дочерней формой многодокументного интерфейса (MDI).
IsMdiContainer	Возвращает или задает значение, определяющее, является ли форма контейнером для дочерней формы многодокументного интерфейса (MDI).
KeyPreview	Возвращает или задает значение, определяющее, получит ли форма события клавиш перед передачей события элементу управления с фокусом.
Left (унаследовано от **Control**)	Возвращает или задает координату по оси X левого края элемента управления (в точках).
Location (унаследовано от	Возвращает или задает координаты

Свойство	Комментарии
Control)	левого верхнего угла элемента управления относительно левого верхнего угла контейнера.
MaximizeBox	Возвращает или задает значение, определяющее, отображается ли кнопка развертывания в строке заголовка формы.
MaximumSize	Возвращает максимальный размер, до которого может быть увеличена форма.
MdiChildren	Возвращает массив форм, представляющий дочерние формы многодокументного интерфейса (MDI), которые являются родительскими для этой формы.
MdiParent	Возвращает или задает текущую родительскую форму многодокументного интерфейса (MDI) этой формы.
Menu	Возвращает или задает MainMenu, который отображается в форме.
MergedMenu	Возвращает объединенное меню для формы.
MinimizeBox	Возвращает или задает значение, определяющее, отображается ли кнопка свертывания в строке заголовка формы.
MinimumSize	Возвращает или задает минимальный размер, до которого может быть уменьшена форма.
Modal	Возвращает значение, указывающее, отображается ли форма как модальная.
Name (унаследовано от **Control**)	Возвращает или задает имя элемента управления.
Opacity	Возвращает или задает уровень непрозрачности формы.
OwnedForms	Возвращает массив объектов **Form**, который представляет все формы, принадлежащие этой форме.
Owner	Возвращает или задает форму, владеющую этой формой.
Parent (унаследовано от **Control**)	Возвращает или задает родительский контейнер элемента управления.

Свойство	Комментарии
ParentForm (унаследовано от **ContainerControl**)	Получает форму, которой присвоен данный элемент управления контейнера.
ProductName (унаследовано от **Control**)	Возвращает имя продукта сборки, содержащей элемент управления.
ProductVersion (унаследовано от **Control**)	Возвращает версию сборки, содержащей элемент управления.
RecreatingHandle (унаследовано от **Control**)	Возвращает значение, показывающее, происходит ли в данный момент повторное создание дескриптора элементом управления.
Region (унаследовано от **Control**)	Возвращает или задает область окна, связанную с элементом управления.
Right (унаследовано от **Control**)	Возвращает расстояние от правого края элемента управления до левого края контейнера.
RightToLeft (унаследовано от **Control**)	Возвращает или задает значение, показывающее, выровнены ли записи элемента управления для поддержки языков, использующих шрифты с написанием справа налево.
ShowInTaskbar	Возвращает или задает значение, определяющее, отображается ли форма на панели задач Windows.
Site (унаследовано от **Control**)	Переопределен. Возвращает или задает подложку элемента управления.
Size	Возвращает или задает размер формы.
SizeGripStyle	Возвращает или задает стиль маркера изменения размера, отображаемого в правом нижнем углу формы.
StartPosition	Возвращает или задает начальное положение формы в режиме выполнения.
TabStop (унаследовано от **Control**)	Возвращает или задает значение, показывающее, можно ли передать фокус данному элементу управления при помощи клавиши TAB.
Tag (унаследовано от **Control**)	Возвращает или задает объект, содержащий данные элемента

Свойство	Комментарии
	управления.
Text (унаследовано от **Control**)	Возвращает или задает текст, связанный с данным элементом управления.
Top (унаследовано от **Control**)	Возвращает или задает координату по оси Y верхнего края элемента управления (в точках).
TopLevel	Возвращает или задает значение, определяющее режим отображения формы в качестве окна верхнего уровня.
TopLevelControl (унаследовано от **Control**)	Возвращает родительский элемент управления, не имеющий другого родительского элемента Windows Forms. Как правило, это самая внешняя **Form**, в которой содержится элемент управления.
TopMost	Возвращает или задает значение, показывающее, необходимо ли отображать форму как форму переднего плана в приложении.
TransparencyKey	Возвращает или задает цвет, представляющий прозрачные области формы.
Visible (унаследовано от **Control**)	Возвращает или задает значение, определяющее, отображается ли элемент управления.
Width (унаследовано от **Control**)	Возвращает или задает ширину элемента управления.
WindowState	Возвращает или задает состояние окна формы.

Открытые методы

Метод	Комментарии
Activate	Активирует форму и снабжает ее фокусом.
AddOwnedForm	Добавляет в эту форму собственную форму.
BeginInvoke (унаследовано от **Control**)	Перегружен. Выполняет делегат асинхронно на том потоке, на котором

Метод	**Комментарии**
	был создан основной дескриптор элемента управления.
BringToFront (унаследовано от **Control**)	Помещает элемент управления в начало z-последовательности.
Close	Закрывает форму.
Contains (унаследовано от **Control**)	Извлекает значение, показывающее, является ли указанный элемент управления дочерним элементом.
CreateControl (унаследовано от **Control**)	Вызывает принудительное создание элемента управления, включая создание дескриптора и дочерних элементов.
CreateGraphics (унаследовано от **Control**)	Создает объект Graphics для элемента управления.
CreateObjRef (унаследовано от **MarshalByRefObject**)	Создает объект, который содержит всю необходимую информацию для создания прокси-сервера, используемого для коммуникации с удаленными объектами.
Dispose (унаследовано от **Component**)	Перегружен. Освобождает ресурсы, используемые объектом Component.
DoDragDrop (унаследовано от **Control**)	Начинает операцию перетаскивания.
EndInvoke (унаследовано от **Control**)	Извлекает возвращаемое значение асинхронной операции, предоставленное переданным объектом IAsyncResult.
Equals (унаследовано от **Object**)	Перегружен. Определяет, равны ли два экземпляра Object.
FindForm (унаследовано от **Control**)	Извлекает форму, на которой находится элемент управления.
Focus (унаследовано от **Control**)	Задает фокус ввода элементу управления.
GetAutoScaleSize	Становится размерным при автоматическом масштабировании формы на основе указанного шрифта.
GetChildAtPoint (унаследовано от **Control**)	Извлекает дочерний элемент управления, имеющий указанные

Метод	**Комментарии**
	координаты.
GetContainerControl (унаследовано от **Control**)	Возвращает следующий ContainerControl в цепочке родительских элементов управления данного элемента.
GetHashCode (унаследовано от **Object**)	Служит хеш-функцией для конкретного типа, пригоден для использования в алгоритмах хеширования и структурах данных, например в хеш-таблице.
GetLifetimeService (унаследовано от **MarshalByRefObject**)	Извлекает служебный объект текущего срока действия, который управляет средствами срока действия данного экземпляра.
GetNextControl (унаследовано от **Control**)	Извлекает следующий или предыдущий элемент управления в последовательности перехода дочерних элементов.
GetType (унаследовано от **Object**)	Возвращает Type текущего экземпляра.
Hide (унаследовано от **Control**)	Скрывает элемент управления.
InitializeLifetimeService (унаследовано от **MarshalByRefObject**)	Получает служебный объект срока действия, для управления средствами срока действия данного экземпляра.
Invalidate (унаследовано от **Control**)	Перегружен. Объявляет недопустимой конкретную область элемента управления и вызывает отправку сообщения изображения элементу управления.
Invoke (унаследовано от **Control**)	Перегружен. Выполняет делегат в том потоке, которому принадлежит основной дескриптор окна элемента управления.
LayoutMdi	Располагает дочерние формы многодокументного интерфейса (MDI) внутри родительской MDI-формы.
PerformLayout (унаследовано от **Control**)	Перегружен. Заставляет элемент управления применять логику макета к дочерним элементам управления.
PointToClient (унаследовано от	Вычисляет расположение указанной

Метод	**Комментарии**
Control)	точки экрана в координатах клиента.
PointToScreen (унаследовано от **Control**)	Вычисляет расположение указанной клиентской точки в координатах экрана.
PreProcessMessage (унаследовано от **Control**)	Выполняет предварительную обработку входящих сообщений в цикле обработки сообщений переди их отправкой.
RectangleToClient (унаследовано от **Control**)	Вычисляет размер и расположение указанного прямоугольника экрана в координатах клиента.
RectangleToScreen (унаследовано от **Control**)	Вычисляет размер и расположение указанной клиентской области в координатах экрана.
Refresh (унаследовано от **Control**)	Принудительно вызывает элемент управления, который в результате делает недоступной свою клиентскую область и немедленно перерисовывает себя и все дочерние элементы.
RemoveOwnedForm	Удаляет собственную форму из этой формы.
ResetBackColor (унаследовано от **Control**)	Восстанавливает значение по умолчанию свойства BackColor.
ResetBindings (унаследовано от **Control**)	Восстанавливает значение по умолчанию свойства DataBindings.
ResetCursor (унаследовано от **Control**)	Восстанавливает значение по умолчанию свойства Cursor.
ResetFont (унаследовано от **Control**)	Восстанавливает значение по умолчанию свойства Font.
ResetForeColor (унаследовано от **Control**)	Восстанавливает значение по умолчанию свойства ForeColor.
ResetImeMode (унаследовано от **Control**)	Восстанавливает значение по умолчанию свойства ImeMode.
ResetRightToLeft (унаследовано от **Control**)	Восстанавливает значение по умолчанию свойства RightToLeft.
ResetText (унаследовано от **Control**)	Восстанавливает значение по умолчанию свойства Text.
ResumeLayout (унаследовано от **Control**)	Перегружен. Восстанавливает обычную логику макета.
Scale (унаследовано от **Control**)	Перегружен. Масштабирует элемент управления и любые его дочерние

Метод	Комментарии
	элементы.
Select (унаследовано от **Control**)	Перегружен. Активирует элемент управления.
SelectNextControl (унаследовано от **Control**)	Активирует следующий элемент управления.
SendToBack (унаследовано от **Control**)	Помещает элемент управления в конец z-последовательности.
SetAutoScrollMargin (унаследовано от **ScrollableControl**)	Возвращает или задает размеры полей для автоматической прокрутки.
SetBounds (унаследовано от **Control**)	Перегружен. Задает границы элемента управления.
SetDesktopBounds	Задает границы формы в координатах рабочего стола.
SetDesktopLocation	Задает расположение формы в координатах рабочего стола.
Show (унаследовано от **Control**)	Отображает элемент управления.
ShowDialog	Перегружен. Отображает форму как модальное диалоговое окно.
SuspendLayout (унаследовано от **Control**)	Временно приостанавливает логику макета для элемента управления.
ToString	Переопределен. См. Object.ToString.
Update (унаследовано от **Control**)	Вызывает перерисовку элементом управления недопустимых областей клиентской области.
Validate (унаследовано от **ContainerControl**)	Проверяет последний недействительный элемент управления и его родительские элементы до текущего элемента управления (но не включая его).

Открытые события

Событие	Комментарии
Activated	Происходит при активации формы в коде или с помощью пользователя.
BackColorChanged (унаследовано от **Control**)	Возникает при изменении значения свойства BackColor.

Событие	**Комментарии**
BackgroundImageChanged (унаследовано от **Control**)	Возникает при изменении значения свойства BackgroundImage.
BindingContextChanged (унаследовано от **Control**)	Возникает при изменении значения свойства BindingContext.
CausesValidationChanged (унаследовано от **Control**)	Возникает при изменении значения свойства CausesValidation.
ChangeUICues (унаследовано от **Control**)	Возникает при изменении фокуса или клавиатурных подсказок пользовательского интерфейса.
Click (унаследовано от **Control**)	Возникает при щелчке элемента управления.
Closed	Происходит при закрытой форме.
Closing	Происходит при закрытии формы.
ContextMenuChanged (унаследовано от **Control**)	Возникает при изменении значения свойства ContextMenu.
ControlAdded (унаследовано от **Control**)	Происходит при добавлении нового элемента управления к Control.ControlCollection.
ControlRemoved (унаследовано от **Control**)	Происходит при удалении элемента управления из Control.ControlCollection.
CursorChanged (унаследовано от **Control**)	Возникает при изменении значения свойства Cursor.
Deactivate	Происходит при потере фокуса неактивной формой.
Disposed (унаследовано от **Component**)	Добавляет обработчик событий для отслеживания события Disposed для компонента.
DockChanged (унаследовано от **Control**)	Возникает при изменении значения свойства Dock.
DoubleClick (унаследовано от **Control**)	Возникает при двойном щелчке элемента управления.
DragDrop (унаследовано от **Control**)	Возникает, когда операция перетаскивания завершена.
DragEnter (унаследовано от **Control**)	Происходит при перемещении объекта внутрь границ элемента управления.
DragLeave (унаследовано от **Control**)	Происходит при перемещении объекта за границы элемента управления.
DragOver (унаследовано от **Control**)	Происходит при перетаскивании объекта над границами элемента управления.
EnabledChanged (унаследовано	Возникает при изменении значения

Событие	**Комментарии**
от **Control**)	свойства Enabled.
Enter (унаследовано от **Control**)	Возникает при входе в элемент управления.
FontChanged (унаследовано от **Control**)	Возникает при изменении значения свойства Font.
ForeColorChanged (унаследовано от **Control**)	Возникает при изменении значения свойства ForeColor.
GiveFeedback (унаследовано от **Control**)	Возникает при операции перетаскивания.
GotFocus (унаследовано от **Control**)	Возникает при получении фокуса элементом управления.
HandleCreated (унаследовано от **Control**)	Происходит при создании дескриптора для элемента управления.
HandleDestroyed (унаследовано от **Control**)	Возникает в процессе уничтожения дескриптора элемента управления.
HelpRequested (унаследовано от **Control**)	Происходит при запросе справки для элемента управления.
ImeModeChanged (унаследовано от **Control**)	Возникает при изменении свойства ImeMode.
InputLanguageChanged	Происходит после изменения языка заполнения формы.
InputLanguageChanging	Происходит, когда пользователь предпринимает попытку изменить язык заполнения для формы.
Invalidated (унаследовано от **Control**)	Возникает, когда отображение элемента управления следует обновить.
KeyDown (унаследовано от **Control**)	Возникает при нажатии клавиши, если элемент управления имеет фокус.
KeyPress (унаследовано от **Control**)	Возникает при нажатии клавиши, если элемент управления имеет фокус.
KeyUp (унаследовано от **Control**)	Возникает, когда клавишу отпускают, если элемент управления имеет фокус.
Layout (унаследовано от **Control**)	Возникает, когда элемент управления должен переместить свои дочерние элементы управления.
Leave (унаследовано от **Control**)	Возникает, когда элемент управления лишается фокуса ввода.
Load	Происходит до первоначального отображения формы.
LocationChanged (унаследовано от **Control**)	Возникает при изменении значения свойства Location.

Событие	**Комментарии**
LostFocus (унаследовано от **Control**)	Возникает при потере фокуса элементом управления.
MaximizedBoundsChanged	Происходит при изменении значения свойства MaximizedBounds.
MaximumSizeChanged	Происходит при изменении значения свойства MaximumSize.
MdiChildActivate	Происходит, когда дочерняя форма многодокументного интерфейса (MDI) активируется или закрывается внутри MDI-приложения.
MenuComplete	Происходит при потере фокуса меню формы.
MenuStart	Происходит при получении фокуса меню формы.
MinimumSizeChanged	Происходит при изменении значения свойства MinimumSize.
MouseDown (унаследовано от **Control**)	Возникает, когда указатель мыши находится на элементе управления и нажата кнопка мыши.
MouseEnter (унаследовано от **Control**)	Возникает, когда указатель мыши оказывается на элементе управления.
MouseHover (унаследовано от **Control**)	Возникает, когда указатель мыши наведен на элемент управления.
MouseLeave (унаследовано от **Control**)	Возникает, когда указатель мыши покидает элемент управления.
MouseMove (унаследовано от **Control**)	Возникает, когда указатель мыши перемещается на элемент управления.
MouseUp (унаследовано от **Control**)	Возникает, когда указатель мыши находится на элементе управления и кнопка мыши не нажата.
MouseWheel (унаследовано от **Control**)	Возникает при движении колеса мыши, если элемент управления имеет фокус.
Move (унаследовано от **Control**)	Возникает при перемещении элемента управления.
Paint (унаследовано от **Control**)	Возникает при обновлении элемента управления.
ParentChanged (унаследовано от **Control**)	Возникает при изменении значения свойства Parent.
QueryAccessibilityHelp (унаследовано от **Control**)	Возникает при предоставлении справки объектом AccessibleObject для приложений со специальными

Событие	Комментарии
	возможностями.
QueryContinueDrag (унаследовано от **Control**)	Возникает во время операции перетаскивания и позволяет источнику перетаскивания определить, должна ли она быть отменена.
Resize (унаследовано от **Control**)	Возникает при изменении размеров элемента управления.
RightToLeftChanged (унаследовано от **Control**)	Возникает при изменении значения свойства RightToLeft.
SizeChanged (унаследовано от **Control**)	Возникает при изменении значения свойства Size.
StyleChanged (унаследовано от **Control**)	Возникает при изменении стиля элемента управления.
SystemColorsChanged (унаследовано от **Control**)	Происходит при изменении системных цветов.
TabStopChanged (унаследовано от **Control**)	Возникает при изменении значения свойства TabStop.
TextChanged (унаследовано от **Control**)	Возникает при изменении значения свойства Text.
Validated (унаследовано от **Control**)	Возникает при окончании проверки элемента управления.
Validating (унаследовано от **Control**)	Возникает при проверке элемента управления.
VisibleChanged (унаследовано от **Control**)	Возникает при изменении значения свойства Visible.

Защищенные свойства

Свойство	Комментарии
CreateParams	Переопределен. См. Control.CreateParams.
DefaultImeMode	Переопределен. Возвращает редактор методов ввода (IME) по умолчанию, поддерживаемый элементом управления.
DefaultSize	Переопределен. См. Control.DefaultSize.
DesignMode (унаследовано от **Component**)	Возвращает значение, указывающее, находится ли данный Component в настоящее время в режиме конструктора.
Events (унаследовано от **Component**)	Возвращает список обработчиков событий, которые прикреплены к этому объекту Component.

Свойство	Комментарии
FontHeight (унаследовано от **Control**)	Возвращает или задает высоту шрифта элемента управления .
HScroll (унаследовано от **ScrollableControl**)	Возвращает или задает значение, показывающее, отображается ли горизонтальная полоса прокрутки.
MaximizedBounds	Возвращает или задает размер формы в развернутом состоянии.
ResizeRedraw (унаследовано от **Control**)	Возвращает или задает значение, определяющее, перерисовывается ли элемент управления при изменении размеров.
ShowFocusCues (унаследовано от **Control**)	Возвращает значение, показывающее, должен ли элемент управления отображать прямоугольники фокуса.
ShowKeyboardCues (унаследовано от **Control**)	Возвращает значение, показывающее, должен ли элемент управления отображать ярлыки клавиатуры.
VScroll (унаследовано от **ScrollableControl**)	Возвращает или задает значение, показывающее, отображается ли вертикальная полоса прокрутки.

Защищенные методы

Метод	Комментарии
AccessibilityNotifyClients (унаследовано от **Control**)	Уведомляет клиентские приложения со специальными возможностями об указанных AccessibleEvents для указанного дочернего элемента управления.
CreateAccessibilityInstance (унаследовано от **Control**)	Создает новый доступный объект для элемента управления.
CreateControlsInstance	Переопределен. См. Control.CreateControlsInstance.
CreateHandle	Переопределен. См. Control.CreateHandle.
DefWndProc	Переопределен. См. Control.DefWndProc.
DestroyHandle (унаследовано от **Control**)	Уничтожает дескриптор, связанный с элементом управления.
Dispose	Перегружен.
Finalize (унаследовано от	Переопределен. Освобождает

Метод	Комментарии
Component)	неуправляемые ресурсы и выполняет другие действия по очистке, перед тем как пространство, которое использует Component, будет восстановлено сборщиком мусора. В языках C# и C++ для функций финализации используется синтаксис деструктора.
GetService (унаследовано от **Component**)	Возвращает объект, представляющий службу, которую предоставляет Component или его Container.
GetStyle (унаследовано от **Control**)	Извлекает значение указанного бита стиля элемента управления для элемента.
GetTopLevel (унаследовано от **Control**)	Определяет, является ли элемент управления элементом верхнего уровня.
InitLayout (унаследовано от **Control**)	Вызывается после добавления элемента управления в другой контейнер.
InvokeGotFocus (унаследовано от **Control**)	Вызывает событие GotFocus для указанного элемента управления.
InvokeLostFocus (унаследовано от **Control**)	Вызывает событие LostFocus для указанного элемента управления.
InvokeOnClick (унаследовано от **Control**)	Вызывает событие Click для указанного элемента управления.
InvokePaint (унаследовано от **Control**)	Вызывает событие Paint для указанного элемента управления.
InvokePaintBackground (унаследовано от **Control**)	Вызывает событие **PaintBackground** для указанного элемента управления.
IsInputChar (унаследовано от **Control**)	Определяет, является ли знак входным знаком, который распознается элементом управления.
IsInputKey (унаследовано от **Control**)	Определяет, является ли заданная клавиша обычной клавишей ввода или специальной клавишей, требующей предварительной обработки.
MemberwiseClone (унаследовано от **Object**)	Создает неполную копию текущего Object.
OnActivated	Создает событие Activated.
OnBackColorChanged	Вызывает событие BackColorChanged.

Метод	**Комментарии**
(унаследовано от **Control**)	
OnBackgroundImageChanged (унаследовано от **Control**)	Вызывает событие BackgroundImageChanged.
OnBindingContextChanged (унаследовано от **Control**)	Вызывает событие BindingContextChanged.
OnCausesValidationChanged (унаследовано от **Control**)	Вызывает событие CausesValidationChanged.
OnChangeUICues (унаследовано от **Control**)	Вызывает событие ChangeUICues.
OnClick (унаследовано от **Control**)	Вызывает событие Click.
OnClosed	Создает событие Closed.
OnClosing	Создает событие Closing.
OnContextMenuChanged (унаследовано от **Control**)	Вызывает событие ContextMenuChanged.
OnControlAdded (унаследовано от **Control**)	Вызывает событие ControlAdded.
OnControlRemoved (унаследовано от **Control**)	Вызывает событие ControlRemoved.
OnCreateControl	Переопределен. См. Control.OnCreateControl.
OnCursorChanged (унаследовано от **Control**)	Вызывает событие CursorChanged.
OnDeactivate	Создает событие Deactivate.
OnDockChanged (унаследовано от **Control**)	Вызывает событие DockChanged.
OnDoubleClick (унаследовано от **Control**)	Вызывает событие DoubleClick.
OnDragDrop (унаследовано от **Control**)	Вызывает событие DragDrop.
OnDragEnter (унаследовано от **Control**)	Вызывает событие DragEnter.
OnDragLeave (унаследовано от **Control**)	Вызывает событие DragLeave.
OnDragOver (унаследовано от **Control**)	Вызывает событие DragOver.
OnEnabledChanged (унаследовано от **Control**)	Вызывает событие EnabledChanged.
OnEnter (унаследовано от **Control**)	Вызывает событие Enter.
OnFontChanged	Переопределен. См.

Метод	**Комментарии**
	Control.OnFontChanged.
OnForeColorChanged (унаследовано от **Control**)	Вызывает событие ForeColorChanged.
OnGiveFeedback (унаследовано от **Control**)	Вызывает событие GiveFeedback.
OnGotFocus (унаследовано от **Control**)	Вызывает событие GotFocus.
OnHandleCreated	Переопределен. См. Control.OnHandleCreated.
OnHandleDestroyed	Переопределен. См. Control.OnHandleDestroyed.
OnHelpRequested (унаследовано от **Control**)	Вызывает событие HelpRequested.
OnImeModeChanged (унаследовано от **Control**)	Вызывает событие ImeModeChanged.
OnInputLanguageChanged	Создает событие InputLanguageChanged.
OnInputLanguageChanging	Создает событие InputLanguageChanging.
OnInvalidated (унаследовано от **Control**)	Вызывает событие Invalidated.
OnKeyDown (унаследовано от **Control**)	Вызывает событие KeyDown.
OnKeyPress (унаследовано от **Control**)	Вызывает событие KeyPress.
OnKeyUp (унаследовано от **Control**)	Вызывает событие KeyUp.
OnLayout (унаследовано от **Control**)	Вызывает событие Layout.
OnLeave (унаследовано от **Control**)	Вызывает событие Leave.
OnLoad	Создает событие Load.
OnLocationChanged (унаследовано от **Control**)	Вызывает событие LocationChanged.
OnLostFocus (унаследовано от **Control**)	Вызывает событие LostFocus.
OnMaximizedBoundsChanged	Создает событие MaximizedBoundsChanged.
OnMaximumSizeChanged	Создает событие MaximumSizeChanged.
OnMdiChildActivate	Создает событие MdiChildActivate.
OnMenuComplete	Создает событие MenuComplete.

Метод	**Комментарии**
OnMenuStart	Создает событие MenuStart.
OnMinimumSizeChanged	Создает событие MinimumSizeChanged.
OnMouseDown (унаследовано от **Control**)	Вызывает событие MouseDown.
OnMouseEnter (унаследовано от **Control**)	Вызывает событие MouseEnter.
OnMouseHover (унаследовано от **Control**)	Вызывает событие MouseHover.
OnMouseLeave (унаследовано от **Control**)	Вызывает событие MouseLeave.
OnMouseMove (унаследовано от **Control**)	Вызывает событие MouseMove.
OnMouseUp (унаследовано от **Control**)	Вызывает событие MouseUp.
OnMouseWheel (унаследовано от **Control**)	Вызывает событие MouseWheel.
OnMove (унаследовано от **Control**)	Вызывает событие Move.
OnNotifyMessage (унаследовано от **Control**)	Уведомляет элемент управления о сообщениях Windows.
OnPaint	Переопределен. См. Control.OnPaint.
OnPaintBackground (унаследовано от **Control**)	Рисует фон элемента управления.
OnParentBackColorChanged (унаследовано от **Control**)	Событие BackColorChanged возникает тогда, когда измененяется значение свойства BackColor контейнера элемента управления.
OnParentBackgroundImageChanged (унаследовано от **Control**)	Событие BackgroundImageChanged возникает при изменении значения свойства BackgroundImage контейнера элемента управления.
OnParentBindingContextChanged (унаследовано от **Control**)	Событие BindingContextChanged возникает при изменении значения свойства BindingContext контейнера элемента управления.
OnParentChanged (унаследовано от **Control**)	Вызывает событие ParentChanged.
OnParentEnabledChanged (унаследовано от **Control**)	Событие EnabledChanged возникает при изменении значения свойства Enabled контейнера элемента управления.
OnParentFontChanged	Событие FontChanged возникает при

Метод	**Комментарии**
(унаследовано от **Control**)	изменении значения свойства Font контейнера элемента управления.
OnParentForeColorChanged (унаследовано от **Control**)	Событие ForeColorChanged возникает при изменении значения свойства ForeColor контейнера элемента управления.
OnParentRightToLeftChanged (унаследовано от **Control**)	Событие RightToLeftChanged возникает при изменении значения свойства RightToLeft контейнера элемента управления.
OnParentVisibleChanged (унаследовано от **Control**)	Событие VisibleChanged возникает при изменении значения свойства Visible контейнера элемента управления.
OnQueryContinueDrag (унаследовано от **Control**)	Вызывает событие QueryContinueDrag.
OnResize	Переопределен. См. Control.OnResize.
OnRightToLeftChanged (унаследовано от **Control**)	Вызывает событие RightToLeftChanged.
OnSizeChanged (унаследовано от **Control**)	Вызывает событие SizeChanged.
OnStyleChanged	Переопределен. См. Control.OnStyleChanged.
OnSystemColorsChanged (унаследовано от **Control**)	Вызывает событие SystemColorsChanged.
OnTabIndexChanged (унаследовано от **Control**)	Вызывает событие TabIndexChanged.
OnTabStopChanged (унаследовано от **Control**)	Вызывает событие TabStopChanged.
OnTextChanged	Переопределен. См. Control.OnTextChanged.
OnValidated (унаследовано от **Control**)	Вызывает событие Validated.
OnValidating (унаследовано от **Control**)	Вызывает событие Validating.
OnVisibleChanged	Переопределен. См. Control.OnVisibleChanged.
ProcessCmdKey	Переопределен. См. Control.ProcessCmdKey.
ProcessDialogChar (унаследовано от **Control**)	Обрабатывает диалоговый знак.
ProcessDialogKey	Переопределен. См.

Метод	Комментарии
	Control.ProcessDialogKey.
ProcessKeyEventArgs (унаследовано от **Control**)	Обрабатывает сообщение о нажатии клавиши и создает соответствующие события элемента управления.
ProcessKeyMessage (унаследовано от **Control**)	Обрабатывает сообщение клавиатуры.
ProcessKeyPreview	Переопределен. См. Control.ProcessKeyPreview.
ProcessMnemonic (унаследовано от **Control**)	Обрабатывает назначенный знак.
ProcessTabKey	Переопределен. См. ContainerControl.ProcessTabKey.
RecreateHandle (унаследовано от **Control**)	Вызывает повторное создание дескриптора элемента управления.
RtlTranslateAlignment (унаследовано от **Control**)	Перегружен. Преобразует текущее выравнивание в выравнивание, необходимое для поддержки теста, читаемого справа налево.
RtlTranslateContent (унаследовано от **Control**)	Преобразует указанное ContentAlignment в **ContentAlignment**, необходимое для поддержки теста, читаемого справа налево.
RtlTranslateHorizontal (унаследовано от **Control**)	Преобразует указанное HorizontalAlignment в **HorizontalAlignment**, необходимое для поддержки теста, читаемого справа налево.
RtlTranslateLeftRight (унаследовано от **Control**)	Преобразует указанное LeftRightAlignment в **LeftRightAlignment**, необходимое для поддержки теста, читаемого справа налево.
ScaleCore	Переопределен. См. Control.ScaleCore.
Select	Перегружен. Переопределен. См. Control.Select.
SetBoundsCore	Переопределен. См. Control.SetBoundsCore.
SetClientSizeCore	Переопределен. См. Control.SetClientSizeCore.
SetStyle (унаследовано от **Control**)	Задает указанное значение указанному биту стиля.

Метод	Комментарии
SetTopLevel (унаследовано от **Control**)	Задает элемент управления как элемент верхнего уровня.
SetVisibleCore	Переопределен. См. Control.SetVisibleCore.
UpdateBounds (унаследовано от **Control**)	Перегружен. Обновляет границы элемента управления.
UpdateStyles (унаследовано от **Control**)	Вызывает принудительное повторное применение назначенных стилей к элементу управления.
UpdateZOrder (унаследовано от **Control**)	Обновляет элемент управления в z-последовательности родительского элемента управления.
WndProc	Переопределен. См. Control.WndProc.

Класс Graphics
Базовый класс: System.MarshalByRefObject
Пространство имен: System.Drawing.Graphics
Инкапсулирует поверхность рисования GDI+. Этот класс не наследуется.

Открытые свойства

Свойство	Комментарии
Clip	Получает или задает объект Region, ограничивающий область рисования данного объекта **Graphics**.
ClipBounds	Получает структуру RectangleF, которая заключает в себе вырезанную область данного объекта **Graphics**.
CompositingMode	Получает значение, задающее порядок рисования сложных изображений в данном объекте **Graphics**.
CompositingQuality	Получает или задает качество отображения сложных изображений, которые выводятся в данном объекте **Graphics**.
DpiX	Получает горизонтальное разрешение данного объекта **Graphics**.
DpiY	Получает вертикальное разрешение данного объекта **Graphics**.
InterpolationMode	Получает или задает режим вставки, связанный с данным объектом

Свойство	Комментарии
	Graphics.
IsClipEmpty	Получает значение, которое указывает, является ли вырезанная область данного объекта **Graphics** пустой.
IsVisibleClipEmpty	Получает значение, которое указывает, является ли видимая вырезанная область данного объекта **Graphics** пустой.
PageScale	Получает или задает масштабирование между универсальными единицами и единицами страницы для данного объекта **Graphics**.
PageUnit	Получает или задает единицу измерения для координат страницы данного объекта **Graphics**.
PixelOffsetMode	Получает или задает значение, которое задает порядок смещения точек во время отображения данного объекта **Graphics**.
RenderingOrigin	Получает или задает исходное заполнение данного объекта **Graphics** для сглаживания цветовых переходов и для штриховки.
SmoothingMode	Получает или задает качество заполнения для данного объекта **Graphics**.
TextContrast	Получает или задает значение коррекции яркости для отображения текста.
TextRenderingHint	Получает или задает режим заполнения для текста, связанного с данным объектом **Graphics**.
Transform	Получает или задает универсальное преобразование для данного объекта **Graphics**.
VisibleClipBounds	Получает или задает рабочий прямоугольник видимой вырезанной области данного объекта **Graphics**.

Открытые методы

Метод	Комментарии
AddMetafileComment	Добавляет комментарий к текущему

Метод	**Комментарии**
	объекту Metafile.
BeginContainer	Перегружен. Сохраняет графический контейнер, содержащий текущее состояние данного объекта **Graphics**, а затем открывает и использует новый графический контейнер.
Clear	Очищает всю поверхность рисования и выполняет заливку поверхности указанным цветом фона.
CreateObjRef (унаследовано от **MarshalByRefObject**)	Создает объект, который содержит всю необходимую информацию для создания прокси-сервера, используемого для коммуникации с удаленными объектами.
Dispose	Освобождает все ресурсы, используемые данным объектом **Graphics**.
DrawArc	Перегружен. Рисует дугу, которая является частью эллипса, заданного парой координат, шириной и высотой.
DrawBezier	Перегружен. Строит кривую Безье, определяемый четырьмя структурами Point.
DrawBeziers	Перегружен. Формирует набор кривых Безье из массива структур Point.
DrawClosedCurve	Перегружен. Строит замкнутую фундаментальную кривую, определяемую массивом структур Point.
DrawCurve	Перегружен. Строит замкнутую фундаментальную кривую через точки указанного массива структур Point.
DrawEllipse	Перегружен. Формирует эллипс, определенный ограничивающим прямоугольником, заданным с помощью пары координат — ширины и высоты.
DrawIcon	Перегружен. Формирует изображение, представленное указанным объектом Icon, расположенным по указанным координатам.
DrawIconUnstretched	Формирует изображение, представленное указанным объектом

Метод	**Комментарии**
	Icon, не масштабируя его.
DrawImage	Перегружен. Рисует заданный объект Image в заданном месте, используя исходный размер.
DrawImageUnscaled	Перегружен. Рисует заданное изображение, используя его исходный фактический размер, в расположении, заданном парой координат.
DrawLine	Перегружен. Проводит линию, соединяющую две точки, определенные парами координат.
DrawLines	Перегружен. Формирует набор сегментов линии, которые соединяют массив структур Point.
DrawPath	Рисует объект GraphicsPath.
DrawPie	Перегружен. Рисует сектор, определенный эллипсом, который задан парой координат, шириной, высотой и двумя радиальными линиями.
DrawPolygon	Перегружен. Рисует многоугольник, определяемый массивом структур Point.
DrawRectangle	Перегружен. Рисует прямоугольник, который определен парой координат, шириной и высотой.
DrawRectangles	Перегружен. Рисует набор прямоугольников, определяемых структурой Rectangle.
DrawString	Перегружен. Создает текстовую строку в заданном месте с указанными объектами Brush и Font.
EndContainer	Закрывает текущий графический контейнер и восстанавливает состояние данного объекта **Graphics**, которое было сохранено при вызове метода BeginContainer.
EnumerateMetafile	Перегружен. Отправляет записи указанного объекта Metafile по отдельности методу обратного вызова, который отображает их в заданной точке.
Equals (унаследовано от	Перегружен. Определяет, равны ли два

Метод	**Комментарии**
Object)	экземпляра Object.
ExcludeClip	Перегружен. Обновляет вырезанную область данного объекта **Graphics**, чтобы исключить из нее часть, определенную структурой Rectangle.
FillClosedCurve	Перегружен. Заполняет замкнутую фундаментальную кривую, определяемую массивом структур Point.
FillEllipse	Перегружен. Заполняет внутреннюю часть эллипса, определяемого ограничивающим прямоугольником, заданным с помощью пары координат — ширины и высоты.
FillPath	Заполняет внутреннюю часть объекта GraphicsPath.
FillPie	Перегружен. Заполняет внутреннюю часть сектора, определенного эллипсом, который задан парой координат, шириной, высотой и двумя радиальными линиями.
FillPolygon	Перегружен. Заполняет внутреннюю часть многоугольника, определенного массивом точек, заданных структурами Point.
FillRectangle	Перегружен. Заполняет внутреннюю часть прямоугольника, который определен парой координат, шириной и высотой.
FillRectangles	Перегружен. Заполняет внутреннюю часть набора прямоугольников, определяемого структурами Rectangle.
FillRegion	Заполняет внутреннюю часть объекта Region.
Flush	Перегружен. Вызывает принудительное выполнение всех отложенных графических операций и немедленно возвращается, не дожидаясь их окончания.
FromHdc	Перегружен. Создает новый объект **Graphics** из указанного дескриптора для контекста устройства.

Метод	**Комментарии**
FromHdcInternal	Внутренний метод. Не используется.
FromHwnd	Создает новый объект **Graphics** из указанного дескриптора для окна.
FromHwndInternal	Внутренний метод. Не используется.
FromImage	Создает новый объект **Graphics** из заданного объекта Image.
GetHalftonePalette	Получает дескриптор текущей полутоновой палитры Windows.
GetHashCode (унаследовано от **Object**)	Служит хеш-функцией для конкретного типа, пригоден для использования в алгоритмах хеширования и структурах данных, например в хеш-таблице.
GetHdc	Получает дескриптор контекста устройства, связанный с данным объектом **Graphics**.
GetLifetimeService (унаследовано от **MarshalByRefObject**)	Извлекает служебный объект текущего срока действия, который управляет средствами срока действия данного экземпляра.
GetNearestColor	Получает цвет, ближайший к указанной структуре Color.
GetType (унаследовано от **Object**)	Возвращает Type текущего экземпляра.
InitializeLifetimeService (унаследовано от **MarshalByRefObject**)	Получает служебный объект срока действия, для управления средствами срока действия данного экземпляра.
IntersectClip	Перегружен. Обновляет вырезанную область данного объекта **Graphics**, включая в нее пересечение текущей вырезанной области и указанной структуры Rectangle.
IsVisible	Перегружен. Указывает, содержится ли точка, заданная с помощью пары координат, в видимой вырезанной области данного объекта **Graphics**.
MeasureCharacterRanges	Получает массив объектов Region, каждый из которых связывает диапазон позиций символов в рамках указанной строки.
MeasureString	Перегружен. Измеряет указанную строку в процессе ее создания с помощью заданного объекта Font.

Метод	**Комментарии**
MultiplyTransform	Перегружен. Умножает универсальное преобразование данного объекта **Graphics** на преобразование указанного объекта Matrix.
ReleaseHdc	Освобождает дескриптор контекста устройства, полученный в результате предыдущего вызова метода **GetHdc** данного объекта **Graphics**.
ReleaseHdcInternal	Внутренний метод. Не используется.
ResetClip	Сбрасывает вырезанную область данного объекта **Graphics** и делает ее бесконечной.
ResetTransform	Сбрасывает матрицу универсального преобразования данного объекта **Graphics** и делает ее единичной матрицей.
Restore	Восстанавливает состояние данного объекта **Graphics**, возвращая его к состоянию объекта GraphicsState.
RotateTransform	Перегружен. Применяет заданное вращение к матрице преобразования данного объекта **Graphics**.
Save	Сохраняет текущее состояние данного объекта **Graphics** и связывает сохраненное состояние с объектом GraphicsState.
ScaleTransform	Перегружен. Применяет указанную операцию масштабирования к матрице преобразования данного объекта **Graphics** путем ее добавления к матрице преобразования объекта.
SetClip	Перегружен. Задает в качестве вырезанной области данного объекта **Graphics** свойство **Clip** указанного объекта **Graphics**.
ToString (унаследовано от **Object**)	Возвращает String, который представляет текущий Object.
TransformPoints	Перегружен. Преобразует массив точек из одного координатного пространства в другое, используя текущее универсальное преобразование и преобразование страницы данного

Метод	Комментарии
	объекта **Graphics**.
TranslateClip	Перегружен. Переводит вырезанную область данного объекта **Graphics** в указанном объеме в горизонтальном и вертикальном направлениях.
TranslateTransform	Перегружен. Добавляет заданный перевод к матрице преобразования данного объекта **Graphics**.

Защищенные методы

Метод	Комментарии
Finalize	Переопределен. См. Object.Finalize. В языках C# и C++ для функций финализации используется синтаксис деструктора.
MemberwiseClone (унаследовано от Object)	Создает неполную копию текущего Object.

Антонов Игорь Вадимович
Бруттан Юлия Викторовна

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ (C#)

Методические указания к лабораторным работам

Редактор Ю.В. Бруттан
Компьютерная верстка Ю.В. Бруттан

Формат 60×90/16. Гарнитура Times New Roman. Усл.. п.л. 2,2.
Тираж 100 экз. Заказ №.

Адрес издательства:
Россия, 180680, Псков, ул. Л.Толстого 4.
Издательско-полиграфический центр ППИ