Лабораторная работа №1
Разработка Windows приложений на C#. Введение в разработку форм.

1. Цель

изучение методов построения форм Windows и получение навыков по настройке форм, созданию непрямоугольных и наследуемых форм в Microsoft Visual Studio, используя C#.

2. Методические указания

1. При изучении языка программирования C# будет использоваться интегрированная среда разработки программного обеспечения Microsoft Visual Studio Express 2013 для Windows Desktop. Будут использованы основные элементы .NET Framework и связь C# с элементами платформы .NET.
2. По окончанию работы сохраните все созданные файлы на своих носителях.
3. Защита лабораторной работы производится только при наличии электронного варианта работающего скрипта.

3. Теоретические сведения

C# (произносится *си шарп*) — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270. C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Особенности языка C# состоят в том, что он разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR. Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем (однако, эта закономерность была нарушена с выходом C# 3.0, представляющего собой расширения языка, не опирающиеся на расширения платформы .NET). CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др. Узнать подробнее о C#

Формы Windows — это основной компонент пользовательского интерфейса windows платформ. Формы предоставляют контейнер, который содержит элементы управления, меню и позволяет отображать приложение в уже привычной и единообразной модели. Формы могут реагировать на события мыши и клавиатуры, поступающие от пользователя,

и выводить на экран данные для пользователя с помощью элементов управления, которые содержатся в форме.

Формы Windows содержат множество свойств, позволяющих настраивать их внешний вид и поведение. Просматривать и изменять эти свойства (Таблица 1.1.) можно в окне **Свойства (Properties) Рис 1.1.** конструктора при разработке, а также программно во время выполнения приложения.

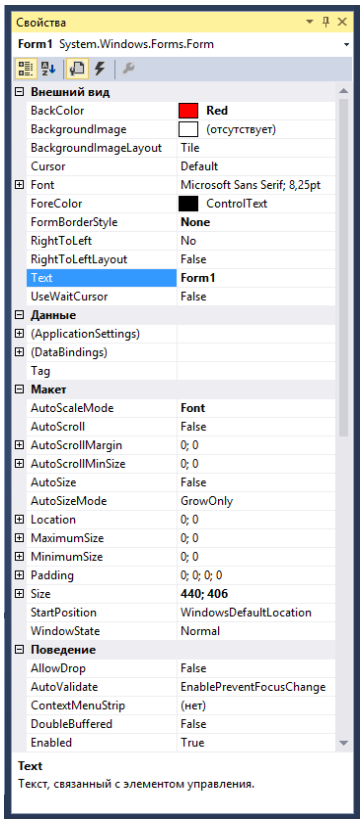

Рис. 1.1. Панель свойств формы

Таблица 1.1. Основные свойства формы

СВОЙСТВО	ОПИСАНИЕ
Name	Задает имя классу **Form,** показанному в конструкторе. Данное свойство задается исключительно во время разработки
BackColor	Указывает цвет фона формы
Enabled	Указывает, может ли форма принимать ввод от пользователя. Если свойству **Enabled** задано значение **False,** все элементы управления формы также блокируются
ForeColor	Указывает цвет переднего плана формы, то есть цвет выводимого текста. Если отдельно не указать значение свойства **ForeColor** элементов управления формы, они примут то же значение
FormBorderStyle	Указывает вид и поведение границы и строки заголовка формы Значения свойства: ▪ **None** - форма не имеет границы, не может быть минимизирована или развернута до максимальных размеров и у нее нет экранной кнопки управления окном и кнопки

	справки ▪ **FixedSingle** - форма имеет тонкую границу, и размеры формы нельзя изменить во время выполнения. Форма может быть минимизирована, развернута до максимальных размеров, и иметь кнопку справки или кнопку управления окном, что определяется остальными ▪ **Fixed3D** - форма имеет объемную границу, и размеры формы нельзя изменить во время выполнения. Форма может быть минимизирована, развернута до максимальных размеров, и иметь кнопку справки или кнопку управления окном, что определяется остальными свойствами ▪ **FixedDialog** - форма имеет тонкую границу, и размеры формы нельзя изменить во время выполнения. У формы нет экранной кнопки управления окном, но может быть кнопка справки, что определяется остальными свойствами. Форму можно минимизировать и развернуть до максимальных размеров ▪ **Sizable** - Форма имеет настройки по умолчанию, но они могут изменяться пользователем. Форма может быть минимизирована, развернута до максимальных размеров, и иметь кнопку справки, что определяется остальными свойствами ▪ **FixedToolWindow** - форма имеет тонкую границу, и размеры формы нельзя изменить во время выполнения. Форма содержит только кнопку закрытия ▪ **SizableToolWindow** - форма имеет тонкую границу, и размеры формы могут быть изменены пользователем. Форма содержит только кнопку закрытия
Location	Когда свойству **StartPosition** задано значение **Manual,** это свойство указывает исходное положение формы относительно верхнего левого угла экрана
MaximizeBox	Указывает, есть ли у формы кнопка **MaximizeBox**
MaximumSize	Устанавливает максимальный размер формы. Если задать этому свойству размер 0; 0, у формы не будет верхнего ограничения размера
MinimizeBox	Указывает, есть ли у формы кнопка **MinimizeBox**
MinimumSize	Устанавливает минимальный размер формы, который пользователь может задать
Opacity	Устанавливает уровень непрозрачности или прозрачности формы от 0 до 100%. Форма, непрозрачность которой составляет 100%, полностью непрозрачна, а форма, имеющая 0 % непрозрачности, наоборот, полностью прозрачна
Size	Принимает и устанавливает исходный размер формы
StartPosition	Указывает положение формы в момент ее первого выведения на экран
Text	Указывает заголовок формы
TopMost	Указывает, всегда ли форма отображается поверх всех остальных форм, свойству **TopMost** которых не задано значение **True**
Visible	Указывает, видима ли форма во время работы
WindowState	Указывает, является ли форма минимизированной, развернутой

	до максимальных размеров, или же при первом появлении ей задан размер, указанный в свойстве **Size**

Создание нового проекта

1. Откройте Microsoft Visual Studio и создайте новый проект Windows Forms Рис. 1.2., обратите внимание на **Имя** файла и **Расположение** проекта, в компьютерном классе он должен соответствовать Рис. 1.2.

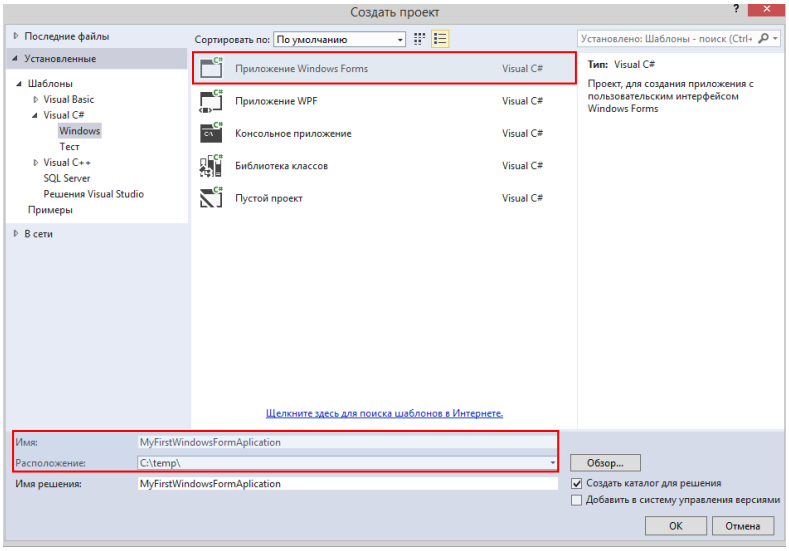

Рис. 1.2. Создание проекта

Проект откроется с формой по умолчанию с именем **Form1** в конструкторе Рис. 1.3.

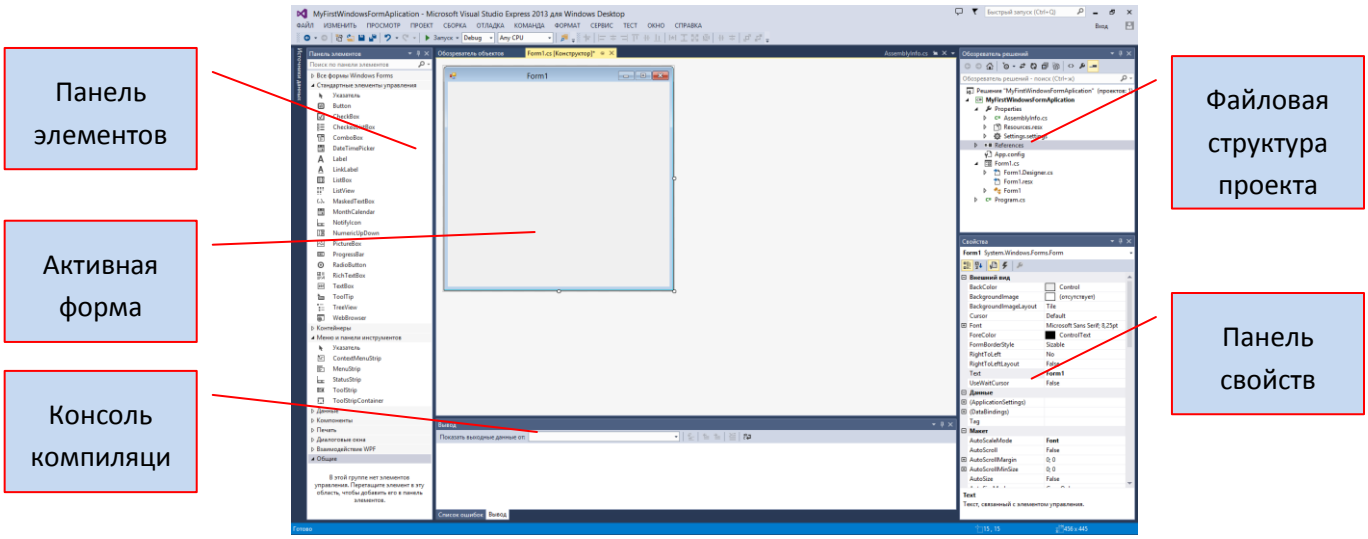

Рис. 1.3. Созданный проект

2. Выберите форму в конструкторе. Свойства формы отображаются в окне **Свойства (Properties).**
3. В окне **Свойства (Properties)** задайте свойствам значения, как указано ниже

Свойство	Значение
Text	Trey Research
FormBorderStyle	Fixed3D
StartPosition	Manual
Location	100; 200
Opacity	75%

4. Перетащите три кнопки из **Toolbox** в форму и разместите их так, как вам будет удобно.
5. Поочередно выберите каждую кнопку и в окне **Свойства (Properties)** задайте свойству кнопок **Text** значения **Border Style, Resize** и **Opacity.**
6. Для кнопки **Border Style** задайте свойство **Anchor** - **Top, Left.**

Реализация обработчиков событий

7. В конструкторе дважды щелкните кнопку **Border Style,** чтобы открыть окно с кодом обработчика события **Buttonl Click.** Добавьте в этот метод следующую строку кода:
   this.FormBorderStyle = FormBorderStyle.Sizable;
8. Возвратитесь в окно конструктора, дважды щелкните кнопку Resize и добавьте следующую строку:
   this.Size = new Size (300, 500);
9. Возвратитесь в окно конструктора, дважды щелкните кнопку Opacity и добавьте следующую строку:
   this.Opacity = 1;

Запуск готового решения

10. Для построения решения выберите меню **Build (Построение),** далее команду **Build Solution (Построить решение).** При наличии ошибок исправьте их и снова постройте решение. В дальнейшем при необходимости выбора последовательности действий очередность команд будет описываться, например, так: **Build | Build Solution.**
11. Нажмите Ctrl + F5 или выберите **Debug (Отладка) | Start Without Debugging (Запуск без отладки),** чтобы запустить приложение. Щелкайте каждую кнопку и наблюдайте, как изменяется вид формы.
12. Измените поочередно расположение левой и верхней границ формы и сравните поведение кнопок внутри формы. Обратите внимание, что расстояние до этих границ от кнопки **Border Style** остается постоянным.

Создание непрямоугольной формы Windows

1. Откройте Visual Studio и создайте новый проект Windows Forms.
2. В окне **Свойства (Properties)** задайте свойству **FormBorderStyle** значение **None,** а свойству **BackColor** значение **Red.**
3. Перетащите кнопку из Toolbox в левый верхний угол формы. Задайте свойству **Text** кнопки значение **Close Form.**
4. Дважды щелкните кнопку Close Form и добавьте в обработчик события **Button1 Click** следующий код:
   this.Close ();
5. В конструкторе дважды щелкните форму, чтобы открыть обработчик события **Form1 Load.** Добавьте в этот метод следующий код:
   System.Drawing.Drawing2D.GraphicsPath myPath = new
   System.Drawing.Drawing2D.GraphicsPath();
   myPath.AddPolygon(new Point[] { new Point(0, 0), new Point(0, this.Height),
   new Point(this.Width, 0) });
   Region myRegion = new Region(myPath); this.Region = myRegion;
6. Постройте и запустите приложение.

Создание наследуемой формы

Если у вас имеется уже готовая форма, которую вы собираетесь использовать в нескольких приложениях, удобно создать наследуемую (производную) форму. В этом

упражнении вы создадите новую форму и унаследуете ее от существующей базовой формы, а затем измените производную форму, настроив ее для конкретной работы.

1. Откройте проект из непрямоугольной формы Windows. Базовой формой для создания производной будет непрямоугольная форма.
2. Для кнопки **Close Form** задайте свойство **Modifiers** как **protected.**
3. Добавьте производную форму: меню Project (Проект) I Add Windows Form...(Добавить форму Windows), **в окне** Categories (Категории) **укажите** Windows Form, **в окне** Templates (Шаблоны) **выберите** Inherited Form (Наследуемая форма).
4. В окне **Add New Item** в поле **Name** укажите название формы: nForm.cs и нажмите **Add** для добавления формы.
5. В появившемся окне **Inheritance Picker,** в котором отображаются все формы текущего проекта, выберите базовую форму Form1 и нажмите OK.
6. Постройте проект.
7. Откройте форму nForm в режиме конструктора. Проверьте, что она имеет треугольную форму и свойства базовой формы и элемента управления наследованы.
8. Настройте свойства производной формы:
   A. для кнопки:
      свойство **Text** - Hello!!!
      свойство **BackColor** - Brown
   B. для формы:
      свойство **BackColor** – Blue
9. Постройте проект.
10. Задайте производную форму в качестве стартовой, указав в функции **Main** следующий код:

Application.Run(new nForm());

11. Постройте и запустите приложение. Должна открыться производная форма со своими свойствами. Проверьте, наследуется ли закрытие формы кнопкой.

Создание MDI-приложения

Multiple document interface (MDI) — способ организации графического интерфейса пользователя, предполагающий использование оконного интерфейса, в котором большинство окон (исключая, как правило, только модальные окна) расположены внутри одного общего окна. Этим он и отличается от SDI, в котором окна располагаются независимо друг от друга. Вопрос о том, какой тип интерфейса предпочтителен — MDI или SDI — часто становится предметом обсуждений в сообществе разработчиков и пользователей программного обеспечения. SDI, в частности, более удобен тогда, когда пользователь одновременно работает с несколькими приложениями разных типов. Разработчики широко используют оба типа интерфейса, а зачастую и интерфейс смешанного типа. Например, Microsoft меняла интерфейс Microsoft Office от SDI к MDI, а потом вернулась обратно к SDI, хотя степень реализации включает и первое, и второе. Подробнее о MDI.

1. Создайте новый проект Windows Forms, укажите имя MdiApplication.
2. Переименуйте файл Form1.cs на ParentForm.cs.
3. Для формы задайте следующие свойства:

Name	ParentForm
Size	420; 320
Text	Parent Form

4. Проверьте, что произошли изменения в функции Main так, чтобы форма ParentForm стала стартовой.
5. Откройте файл ParentForm.cs в режиме конструктора.
6. Для свойства формы **IsMdiContainer** задайте значение **True.** Таким способом эта форма будет определена как родительская форма MDI.

Создание меню для работы с формами

1. Создайте пункт меню **File:**
2. Откройте в панеле инструментов **Меню и панели инструментов (Toolbox),** добавьте на форму элемент управления **MenuStrip** и задайте для его свойства **Name** значение **MdiMenu.**
3. Выделите меню в верхней части формы и задайте имя первого пункта меню **&File.**
4. Для свойства **Name** пункта меню **File** задайте значение **FileMenuItem.**
5. Раскройте меню **File.** Выделите элемент, появившейся под элементом **File,** и задайте его как **&New.** Для свойства **Name** пункта меню **New** задайте значение **NewMenuItem.** Выделите элемент, появившийся под элементом **New,** и задайте его как **&Exit.** Для свойства **Name** пункта меню **Exit** задайте значение **ExitMenuItem.**
6. Дважды кликните левой кнопкой мыши по пункту меню **Exit** для создания обработчика события **Click.**
7. В обработчик события **Click** для пункта меню **Exit** добавьте следующий код:
   this.Close ();
8. Создайте пункт меню **Window.** Переключитесь в режим конструктора. Выделите второй пункт меню справа от **File** и задайте его значением **&Window.** Для свойства **Name** пункта меню **Window** задайте значение **WindowMenuItem.** Раскройте меню **Window.** Выделите элемент, появившейся под элементом **Window,** и задайте для его свойства **Text** значение **&Cascade.** Для свойства **Name** пункта меню **Cascade** задайте значение **WindowCascadeMenuItem.** Выделите элемент, появившийся под элементом **Cascade,** и задайте для его свойства **Text** значение **&Tile.** Для свойства **Name** пункта меню **Tile** задайте значение **WindowTileMenuItem.**
9. Дважды кликните левой кнопкой мыши по пункту меню **Cascade** для создания обработчика события **Click:**
   this.LayoutMdi (System.Windows.Forms.MdiLayout.Cascade);
10. Вернитесь в режим конструктора и дважды кликните левой кнопкой мыши по пункту меню **Tile.**
11. В обработчик события **Click** для пункта меню **Tile** добавьте следующий код:
   this.LayoutMdi(System.Windows.Forms.MdiLayout.TileHorizontal);
12. Реализуйте список открытых окон в меню **Window.** В конструкторе выберите компонент **Mdimenu.** Укажите в свойстве **MdiWindowListItem** имя пункта, созданного для этого - **WindowMenuItem.**

Создание дочерней формы

13. Создайте дочернюю форму. Выберите пункт меню Project I Add Windows Form. Задайте имя формы **ChildForm.cs.** Для свойства **Text** формы задайте значение **Child Form.** На панели элементов **(Toolbox)** дважды кликните левой кнопкой

мыши по элементу управления **RichTextBox** и задайте для его свойства **Name** значение **ChildTextBox.**

14. Для свойства **Dock** элементу управления **RichTextBox** задайте значение **Fill.**
15. Удалите существующий текст (если он есть) для свойства **Text** элемента управления **RichTextBox** и оставьте его пустым.
16. На панели элементов **(Toolbox)** дважды кликните левой кнопкой мыши по элементу управления **MenuStrip.**
17. Для свойства **Name MenuStrip** задайте значение **ChildWindowMenu.** Выделите меню в верхней части формы и наберите текст **F&ormat.** Для свойства **Name** пункта меню **Format** задайте значение **FormatMenuItem,** для свойства **MergeAction** установите значение **Insert,** а свойству **MergeIndex - 1.** В этом случае меню **Format** будет располагаться после **File** при объединении базового и дочерних меню. Выделите элемент, появившийся под элементом **Format,** и наберите текст **&Toggle Foreground.** Для свойства **Name** пункта меню **Toggle Foreground** задайте значение **ToggleMenuItem.** Дважды кликните левой кнопкой мыши по пункту меню **Toggle Foreground** и добавьте следующий код в обработчик события **Click:**

```
if (ToggleMenuItem.Checked) {
ToggleMenuItem.Checked = false;
ChildTextBox.ForeColor = System.Drawing.Color.Black;
}
else
{
ToggleMenuItem.Checked = true;
ChildTextBox.ForeColor = System.Drawing.Color.Blue;
}
```

Отображение дочерней формы

18. Отобразите дочернюю форму в родительской форме. Откройте ParentForm.cs в режиме конструктора.
19. Дважды кликните левой кнопкой мыши по кнопке **New** в меню **File** для создания обработчика события **Click.**
20. Добавьте следующий код для обработчика события **Click** для пункта меню **New:**

```
ChildForm newChild = new ChildForm();
newChild.MdiParent = this;
newChild.Show();
```

Работа с приложением

21. Проверьте работу приложения. Постройте и запустите приложение. Когда появится родительская форма, выберите пункт меню **File | New.** В родительском окне появится новая дочерняя форма. Обратите внимание на то, дочернее меню сливается с родительским и пункты меню упорядочиваются в соответствие со свойством **MergeIndex,** установленным ранее. Наберите какой-нибудь текст в дочернем окне и воспользуйтесь пунктом меню **Format** для изменения цвета шрифта текста. Откройте еще несколько дочерних окон. Выберите пункт меню **Window | Tile.** Обратите внимание на то, что дочерние окна выстраиваются в упорядоченном порядке. Закройте все дочерние окна. Обратите внимание на то, что, когда закроется последнее дочернее окно, меню родительской формы

изменится, и оттуда исчезнет пункт **Format.** Для закрытия приложения выберите пункт меню **File | Exit.**

22. Обратите внимание, что заголовок у дочерних окон одинаковый. При создании нескольких документов, например в Microsoft Word, они называются Документ^ где N — номер документа. Реализуйте эту возможность. Откройте код родительской формы и в классе ParentForm объявите переменную openDocuments:

    private int openDocuments = 0;

23. К свойству **Text** дочерней формы добавьте счетчик числа открываемых документов (в коде обработчика события **Click** для пункта меню **New):**

    newChild.Text = newChild.Text+" "+ ++openDocuments;

24. Запустите приложение. Теперь заголовки новых документов содержат порядковый номер.

4. Содержание отчёта

Отчёт должен содержать название и цель. Ход работы и выполненные примеры, результаты работы, выполненные 3 задания. Выводы.

5. Контрольные вопросы

1. Что такое C#?
2. В чем отличие C# от других языков программирования?
3. Какую парадигму программирования поддерживает C#?
4. Что такое Microsoft .NET Framework?
5. Что такое MDI?
6. Что такое SDI?
7. Какие бывают формы?
8. Основные свойства формы?

Задание 1. Создайте пользовательскую форму, которая во время выполнения будет иметь овальное очертание. Данная форма должна содержать функциональность, дающую возможность пользователю закрывать ее во время выполнения. Рекомендация: при разработке формы в виде эллипса используйте следующий код: myPath.AddEllipse(0, 0, this.Width, this.Height);

Задание 2. Создайте приложение с двумя формами и установите вторую форму как стартовую. Сделайте так, чтобы при запуске стартовая форма разворачивалась до максимальных размеров и содержала функциональность, дающую возможность пользователю открыть первую форму, отображающуюся в виде ромба зеленого цвета с кнопкой (в центре ромба) закрытия формы с надписью GREENPEACE. Первая форма должна в тайтле выводить ФИО студента и группу обучения.

Задание 3. Создайте приложение, которое будет поддерживать организацию форм MDI. При создании MDI формы, на ней должна быть кнопка для открытия формы SDI. Форма SDI при открытии должна быть черным цветом и не иметь компонентов управления, при этом ее размеры можно было менять, так же форма должна иметь в центре кнопку её закрытия.