Лабораторная работа 7. Работа с формами и визуальными компонентами в программах на C#

1. ЦЕЛЬ РАБОТЫ

Изучить основные способы работы с формами в Visual Studio; изучить возможности использования простейших визуальных компонентов в программах на языке Visual C#.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Интегрированная среда разработки (IDE) программ Visual Studio – это линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы для различных платформ.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense . Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода, добавление новых наборов инструментов и др.

В интегрированной среде разработки Visual Studio проекты логически организуются в решения (solutions). После компиляции и сборки (линкования) проекта мы получаем исполнимый модуль, называемый сборкой (например, модуль библиотеки DLL). Каждое решение состоит из одного или нескольких проектов. В свою очередь, каждый проект может состоять из любого количества исходных файлов, ссылок на внешние сборки (которые используются этим проектом) и прочих ресурсов, которые и образуют приложение.

Строка меню в верхней части экрана позволяет управлять всей работой по созданию программного продукта. **Панель инструментов** предоставляет быстрый доступ ко многим возможностям меню с помощью своих кнопок. Панели инструментов – это наборы маленьких кнопок с пиктограммами, расположенные под строкой меню. С их помощью можно получить доступ к наиболее часто используемым возможностям IDE, доступным обычно через меню. Панели инструментов можно расположить в одной строке, кроме того, можно перетащить панель инструментов и превратить ее в плавающее окно, которое можно разместить в любом удобном месте. Панель **ToolBox** используется для добавления в формы проекта различных элементов управления. В окне **Solution Explorer** отображаются те проекты, над которыми производится работа в данный момент. В окне **Properties** (свойства) можно просматривать и изменять свойства элементов управления, форм и модулей.

Последовательность создания приложения с графическим интерфейсом (GUI) можно разбить на несколько этапов:

1. Создание графического интерфейса.
2. Определение свойств у всех элементов приложения.
3. Создание функций обработки необходимых событий.
4. Сохранение, запуск, отладка.

Windows Forms — интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft.NET Framework. *Приложение Windows Forms* представляет собой событийно-ориентированное приложение, поддерживаемое Microsoft.NET Framework. В отличие от пакетных программ, большая часть времени тратится на ожидание от пользователя каких-либо действий, как, например, ввод текста в текстовое поле или клика мышкой по кнопке.

Работа с формами. Основным компонентом среды визуального программирования является форма (`Form`). Форма – это *контейнер*, в который помещаются остальные элементы приложения, определяющие впоследствии всю функциональность приложения. В форме можно разместить различные визуальные *компоненты* (элементы управления), такие как кнопка, текстовое поле для ввода данных, список, поле со списком и т.д. Для работы формами используется *дизайнер форм*.

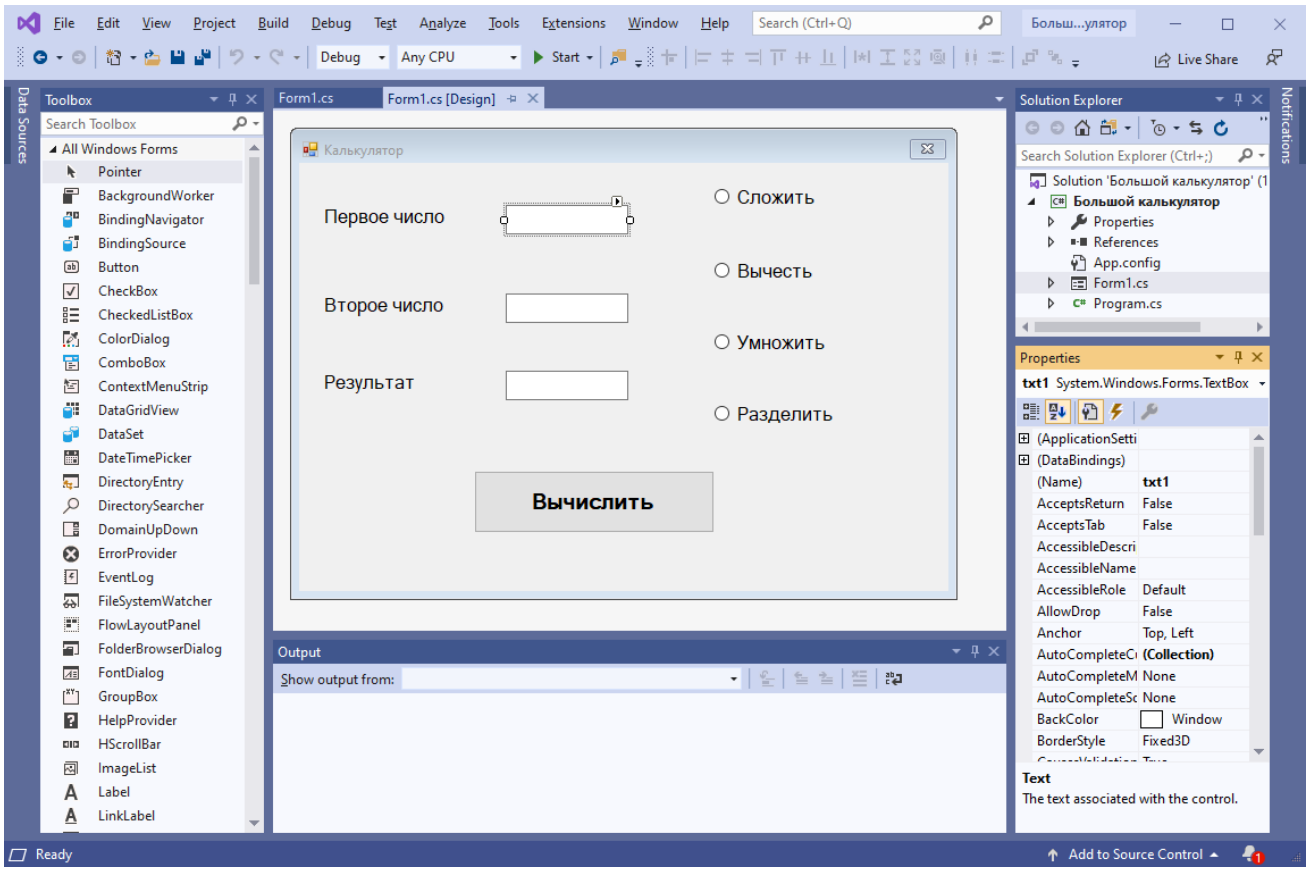


Рисунок 1. Примерный результат создания проекта с формой

Основные свойства формы. Для формы и других компонентов характерно наличие *свойств* и *событий.* Свойства служат двум главным целям. Во-первых, они определяют внешний вид формы или компонента. А во-вторых, свойства определяют поведение формы или компонента. Visual Studio позволяет изменять свойства компонентов как в режиме проектирования (design time), так и в режиме выполнения программы (run time).

Рассмотрим некоторые свойства форм (многие из этих свойств встречаются также и у компонентов, с которыми мы будем работать). Рекомендуется попробовать изменять эти свойства у формы в Вашем проекте.

`AllowDrop` – свойство определяет возможность вывода у себя данных, когда пользователь перемещает их над этим компонентом.

`AutoScaleMode` – это свойство задает возможность автоматического масштабирования. Значение свойства выбирается из выпадающего списка. Если выбрать значение `Font`, то автоматическое изменение шрифта будет полезным тогда, когда форма или компонент должны растягиваться или сокращаться в соответствии с размерами с размерами шрифта в операционной системе. Если выбрано значение `DPI`, то размеры формы или компонента будут изменяться относительно экрана. Если выбрать значение `Inherit`, то другой компьютер станет наследовать шрифт и разрешение базового компьютера.

`AutoScroll` – это свойство задает возможность автоматического появления полос прокрутки.

`AutoScrollMargin` – ширина полос прокрутки (при необходимости) в пикселах.

`AutoSize` – возможность автоматического изменения размеров.

`AutoSizeMode` – режим автоматического изменения размеров. Возможные значение: `GrowOnly` (только растягиваться), `GrowAndShrink` (растягивать и уменьшаться).

`BackColor` – цвет фона.

`BackgroundImage` – фоновое изображение.

`BackgroundImageLayout` – тип размещения фонового изображения: подгонять под размер компонента, растягивать и др.

`CancelButton` – это свойство позволяет создавать имитацию нажатию некоторой кнопки с помощью клавиши ESC в момент работы приложения. При нажатии ESC приложение будет выполнять такое же действие, как будто пользователь щелкнул по указанной кнопке.

`CausesValidation` – включает/выключает необходимость проверки на достоверность компонента во время получения им фокуса. Это свойство подавляет или не подавляет возникновение события `Validating`.

`ContextMenuStrip` – через это свойство к компоненту подключается его контекстное меню.

`ControlBox` – предоставляет возможность вывода в различном виде заголовочной полосы формы (с кнопочками (`True`) или без (`False`)).

`Cursor` – задает форму курсора мыши для формы или компонента.

`Enabled` – задает возможность доступа к компоненту.

`Font` – определяет характеристики шрифта формы или компонента.

`ForeColor` – цвет переднего плана компонента.

`FormBorderStyle` – задает стиль окантовки формы, который выбирается из выпадающего списка. По умолчанию имеет значение `Sizable` – форма может изменять свои размеры в режиме выполнения.

`HelpButton` – задает возможность вывода кнопки помощи (с вопросительным знаком) в заголовке компонента. Обработка кнопки помощи определяется в обработчике события `HelpRequested` формы или компонента.

`Icon` – определяет пиктограмму приложения.

`ImeMode` – Input Method Editor – подключает к компоненту редактор с различными режимами обработки входных данных.

`IsMdiContainer` – показывает, является ли форма контейнером для многодокументного интерфейса.

`Locked` – блокировка компонента. В результате блокировки компонент теряет возможность перемещаться или изменять размеры. При этом в левом верхнем углу компонента появляется пиктограмма замка.

`MainMenuStrip` – через это свойство к компоненту подключается главное меню.

`Opacity` – задает уровень затемнения (прозрачности) формы. Значение указывается в процентах. Чем ниже процент, тем более прозрачна форма.

`Padding` – задает отступы внутри компонента и определяет пространство внутри компонента, которое «держит» на заданной дистанции от границ компонента содержимое компонента.

`Size` – размеры компонента.

`SizeGripStyle` – это свойство позволяет задать вывод/невывод калибровочной полоски в правом нижнем углу формы.

`StartPosition` – задает стартовую позицию формы в режиме исполнения приложения, например, по центру экрана (`CenterScreen`).

`Tag` – нечто вроде буферной области, связанной с компонентом. Это свойство позволяет сохранять какие-нибудь данные, чтобы потом ими воспользоваться.

`Text` – заголовок компонента.

`TopMost` – определяет, будет ли данная форма всегда помещена над другой.

`TransparencyKey` – цвет, которым будут высвечиваться прозрачные области формы.

`UseWaitCursor` – определяет, будет ли использоваться курсор в виде песочных часов для данного компонента и всех его потомков или нет.

`WindowState` – состояние окна формы.

Некоторые события формы. Как известно, *событие* – действие, вызывающее реакцию объекта, например, нажатие клавиши на клавиатуре или кнопки мыши. С помощью свойств, связанных с событиями, можно указать, что при возникновении события следует выполнить определенную функцию обработки событий. Рассмотрим некоторые события формы.

`Activated` – возникает, когда форма активизируется.

`Click` – возникает при щелчке мышью по форме.

`ControlAdded` – возникает, когда в форму добавлен новый элемент управления (во время выполнения приложения).

`DoubleClick` – возникает, когда произведен двойной щелчок мышью по форме.

`FormClosed` – возникает после закрытия формы.

`FormClosing` – возникает перед закрытием формы.

`HelpButtonClicked` – возникает после щелчка на кнопке HelpButton.

`HelpRequested` – возникает при нажатии на кнопке F1.

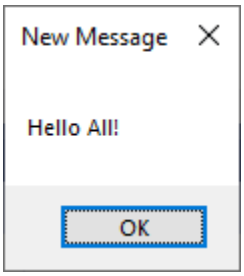
`Load` – возникает перед первым выводом формы.

`Paint` – возникает, когда форма перерисовывается.

Заготовка обработчика события формируется следующим образом: надо открыть вкладку **Events** (события) в окне **Properties**, выбрать нужное событие (например, `Click`) и дважды щелкнуть мышкой в поле этого события. Среда программирования сразу создает в файле кода формы заготовку: заголовок функции, которая должна будет выполняться, как только наступит соответствующее событие, и пустое тело функции (открывающую и закрывающую фигурные скобки), в которое мы должны вписать свои инструкции на C#, отражающие алгоритм реакции на событие.

Например, создадим обработчик события `Click` для формы со следующим кодом:

```
private void Form1_Click(object sender, EventArgs e) {
     MessageBox.Show("Hello All!", "New Message");
   }
```



При щелчке мышкой по форме должно появляться стандартное диалоговое окно с заданными текстом сообщения и заголовком.

Некоторые методы формы. *Метод* – функция, внедренная непосредственно в объект. Форма имеет большое количество методов. В разделе **Form Members** справочной системы перечислены не только методы, но и все свойства и события формы. Рассмотрим некоторые методы формы.

`Close()` – закрывает форму. Если закрывается главная форма, приложение закрывается. Ресурсы, занятые формой, освобождаются.

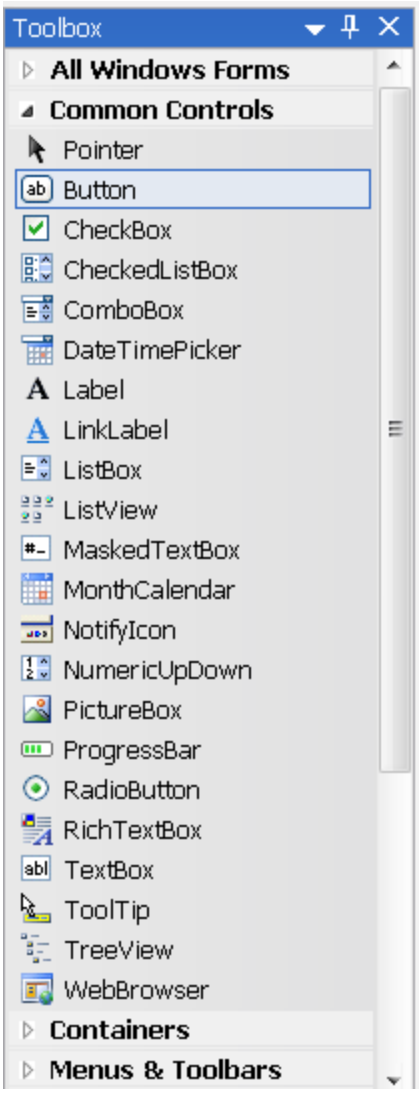
`Hide()` – форма становится невидимой.

`Show()` – выводит форму на экран.

`ShowDialog()` – показывает форму в модальном режиме.

`Focus()` – передает фокус форме или компоненту (делает форму активной).

Компоненты. Компоненты, размещаемые в форме, позволяют пользователю программы выполнять различные действия, например, щелкать на кнопках или вводить данные. Рассмотрим наиболее часто используемые элементы управления, размещенные на панели **Toolbox**.



`Button` – прямоугольная кнопка с надписью. Поместив `Button` на форму, Вы по двойному щелчку можете создать заготовку обработчика события нажатия кнопки. Далее нужно заполнить заготовку кодом. Вот основные свойства элемента управления `Button`: *`Left`* (позиция элемента управления относительно левого края его контейнера), *`Top`* (позиция элемента относительно верхнего края его контейнера), *`Height`* (высота), *`Width`* (ширина), *`Enabled`* (определяет, можно ли пользователю работать с этим элементом управления), *`Visible`* (видимость во время выполнения программы), *`Text`* (подпись).

Все указанные свойства можно менять как во время разработки программы, так и во время ее работы. Чтобы изменить свойства кнопки во время работы программы, необходимо в процедуре использовать команду вида:

```
НазваниеЭлемента.НазваниеСвойства =
                НовоеЗначение
```

Например, если в форме имеется кнопка `cmd1`, то можно изменить ее свойства следующим образом:

```
cmd1.Text = “Нажми на меня”;
cmd1.Left = 200;
```

При этом новое значение для свойства `Text` указывается в кавычках, так как оно имеет тип `String`, а для свойства `Left` – без кавычек, так как оно имеет числовой тип.

`Label` служит для отображения текста на экране. Вы можете изменить шрифт и цвет надписи, если щелкнете на кнопке с троеточием в строке `Font` в окне Properties. Текст надписи является значением свойства `Text`. Свойство `TextAlign` определяет способ выравнивания текста. Вывести текст в надпись во время работы программы можно, например, так:

```
Label1.Text = "Введите число:";
```

`TextBox` - стандартный управляющий элемент Windows для ввода. Он может быть использован для отображения короткого фрагмента текста и позволяет пользователю вводить текст во время выполнения программы. Начальное содержимое области редактирования определяет строка, являющаяся значением свойства `Text`. Свойство `Font` определяет параметры шрифта текстового поля. Установив свойство `ReadOnly` в значение True, мы запрещаем пользователю программы вводить данные в текстовое поле. Свойство `MaxLength` определяет число символов, которые можно ввести в текстовое поле. Свойство `SelectedText` содержит текущий выделенный фрагмент строки в текстовом поле. Свойства `SelectionStart`, `SelectionLength` возвращают начальную позицию и длину выделенного фрагмента строки в текстовом поле. Свойство `MultiLine` позволяет установить возможность вывода текста в несколько строк. Свойство `TextAlign` определяет способ выравнивания текста.

Событие `TextChanged` возникает при вводе и изменении данных в тектовом поле. Можно использовать процедуру обработки этого события, чтобы обрабатывать данные, вводимые пользователем. Пример функции обработки события `TextChanged`:

```
private void textBox1_TextChanged(object sender, EventArgs e){
      lblResult.Text = textBox1.Text;
}
```

`RichTextBox` – отображает прямоугольную область редактируемого текста на форме. Похож на `TextBox`, но может содержать несколько абзацев, а также форматирование разным цветом, шрифтами и др. Имеет свойство `Text`. Начальное содержимое области редактирования также определяет массив строк, являющийся значением свойства `Lines`. Окно редактора элементов списка открывается кнопкой в графе значении этого свойства. Позволяет загружать данные из файла и сохранять в файл, используя методы `LoadFile` и `SaveFile`.

Элементы управления **`CheckBox`** и **`RadioButton`** часто используются при программировании. *`CheckBox`* – это флажок для включения или отключения опций,

а *RadioButton* – переключатель, который позволяет выбирать один элемент из группы.

Основные свойства элементов управления CheckBox и RadioButton: *Name* – имя элемента; *Text* – подпись; *Checked* – признак того, что элемент выбран; *Left*, *Top* – определяет положение элемента; *Height*, *Width* – задают размеры элемента. Свойство *Checked* для компонентов CheckBox и RadioButton равно True, если переключатель выбран и False, если элемент не выбран.

PictureBox – окно рисунка. Мы будем изучать особенности использования элемента PictureBox чуть позже.

Списки (***ListBox***) и поля со списками (***ComboBox***) используются для предоставления пользователю возможности выбора нужной информации. Оба элемента управления имеют свойство **Items** – коллекцию, состоящую из всех элементов списка. Элементы списков нумеруются с нуля. Основные действия при работе со списками и полями со списком заключаются в добавлении и удалении строк в свойстве **Items** этих элементов управления.

Timer – таймер (позволяет программе выполнять действия в реальном времени без вмешательства пользователя). Вызываемое таймером событие называется **Tick.** Элемент управления Timer является невидимым. Время между наступлением событий Timer должно указываться в свойстве *Interval* элемента управления таймером. Единицей измерения интервала времени является миллисекунда.

Для добавления в формы проекта различных компонентов используется панель **ToolBox**. Доступ к ней можно получить с помощью меню **View | Toolbox**.

Организация работы с множеством форм. В общем случае приложение может иметь много форм. Формы могут быть добавлены с помощью команд меню **Project | Add New Item** (в диалоговом окне в этом случае необходимо выбрать Form (Windows Form).

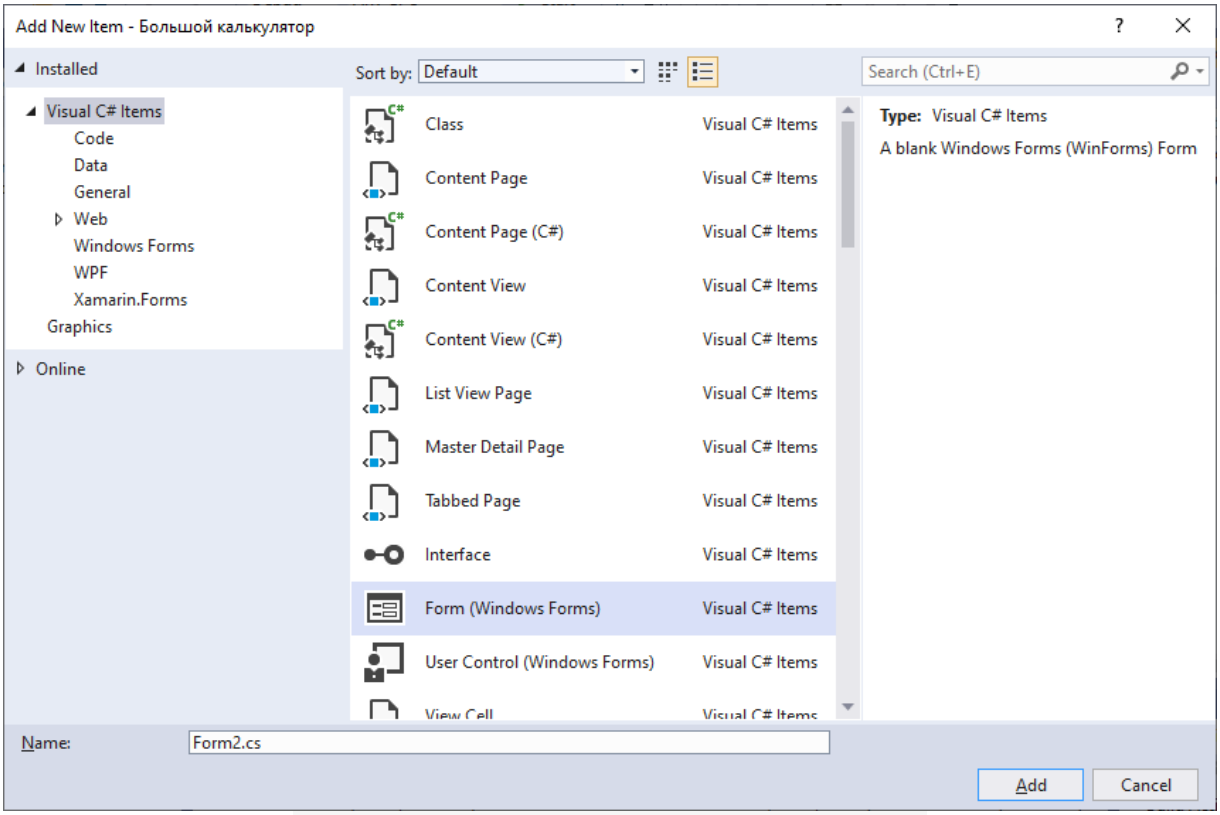


Рисунок 2. Добавление формы в проект

Результат добавления формы отображается в окне **Solution Explorer**. Для редактирования новых форм также используется дизайнер форм. Для переключения между редактируемыми формами можно щелкнуть по соответствующей вкладке (например, **Form2.cs[Design]**) или выбрать нужную форму в окне **Solution Explorer**.

Формы, которые только добавлены в проект, не выводятся на экран при запуске проекта на выполнение. При выполнении приложения формы надо принудительно вызывать.

Форма может быть отображена в двух режимах: модальном или немодальном (обычном). Разница между ними в следующем. *Модальное окно диалога* (modal dialog box) или форма должны быть закрыты, чтобы можно было продолжать работу с приложением. Например, диалоговые окна, отображающие важные сообщения, всегда должны быть модальными, то есть пользователь обязан закрывать их или отвечать на сообщения в них, прежде чем продолжить работу. *Немодальные окна диалога* (modeless dialog boxes) или формы позволяют перемещать фокус между окном диалога и другой формой, не закрывая окно диалога. Немодальные окна диалога достаточно редки.

Чтобы отобразить форму как модальное окно диалога, следует использовать метод `ShowDialog()` формы. Чтобы отобразить форму в немодальном режиме, следует использовать метод `Show()`, например:

```
private void button2_Click(object sender, EventArgs e){
        //Создание экземпляра класса
        Form MyForm = new Form2();
```

```
        //Вывод формы
        MyForm.Show();
    }
```

В этой функции обработки нажатия на кнопку создается и выводится на экран форма `Form2`.

3. ЗАДАНИЕ НА РАБОТУ

1. Разработать приложение «Калькулятор» по инструкциям, приведенным в пункте «Порядок выполнения работы».

2. Разработать 2 приложения по своему варианту.

4. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Задание 1. Ознакомиться с теоретическим материалом, приведенным в пункте «Краткие теоретические положения» данных методических указаний, а также с рекомендуемой литературой [1, 2] по данной теме.

Задание 2. Создание программы «Калькулятор»

Запустить Visual Studio. Создать новый проект -> Visual C# -> Windows Forms App (.NET Framework).

Основное окно «Калькулятора» содержит текстовые поля (`TextBox`) *`txt1`* и *`txt2`* для ввода данных, текстовое поле *`txtResult`* для вывода результата, 4 переключателя (`RadioButton`) с названиями (свойство Name) *`rbAdd`*, *`rbSub`*, *`rbMul`* и *`rbDiv`* для выбора операции и кнопку *`button1`* с надписью «Вычислить». Надо добавить эти компоненты в форму и настроить необходимые свойства в окне Properties (Свойства).

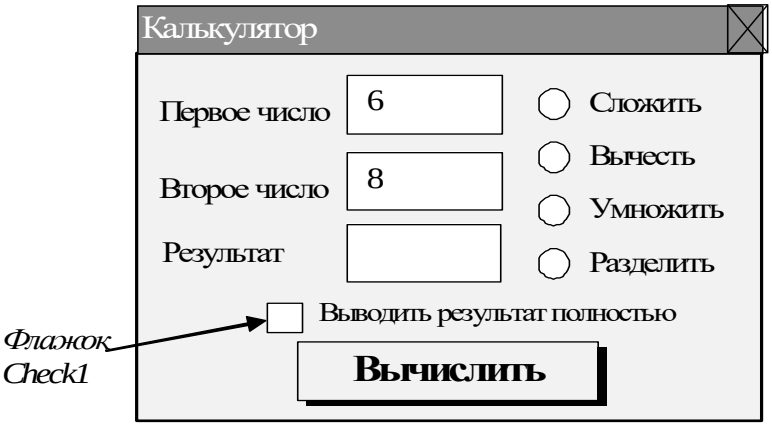
Далее необходимо написать текст функции обработки нажатия кнопки `button1` (для создания функции обработки события нажатия кнопки `button1_Click` дважды щелкните по кнопке в окне редактора формы):

```
private void button1_Click(object sender, EventArgs e) {
      //Объявляем переменные для вводимых чисел и результата
       double x, y, z = 0;

       //Преобразуем текст из числовых полей в переменные Double
       x = Convert.ToDouble(txt1.Text);
       y = Convert.ToDouble(txt2.Text);

       //Выполняем операцию, тип которой определяется
```

```
        //на основе проверки значений св-ва Checked переключателей
        if (rbAdd.Checked) z = x + y;
        if (rbSub.Checked) z = x - y;
        if (rbMul.Checked) z = x * y;
        //В случае деления надо убедиться, что 2-е число
        //не равно нулю.
        if (rbDiv.Checked)
            if (y != 0)
                z = x / y;
            else
               MessageBox.Show("На нуль делить нельзя!", "Ошибка");

        //Преобразуем значение переменной z из типа Double
        //в строку и выводим ее в текстовое поле результата
        txtResult.Text = Convert.ToString(z);
    }
```

Сохраните приложение. Проверьте его работу.

Добавим в приложение флажок (элемент `CheckBox`), который позволяет выводить в поле результата само арифметическое действие и результат его выполнения. Назовем флажок *`Check1`*.



Функцию `button1_Click` надо изменить следующим образом.

Вместо строк

```
'Преобразуем значение переменной z из типа Double в строку и
'выводим ее в текстовое поле результата
    txtResult.Text = Convert.ToString(z);
```

необходимо написать:

```
 //Проверяем флажок и, если он выбран, формируем строку
 //в текстовом поле результата.
 //Иначе выводим только одно число z
 if (Check1.Checked) {
      string str_x = Convert.ToString(x);
      string str_y = Convert.ToString(y);
      string str_z = Convert.ToString(z);
      if (rbAdd.Checked) txtResult.Text = str_x + "+" + str_y + "=" + str_z;
      if (rbSub.Checked) txtResult.Text = str_x + "-" + str_y + "=" + str_z;
      if (rbMul.Checked) txtResult.Text = str_x + "*" + str_y + "=" + str_z;
     if (rbDiv.Checked && y != 0) txtResult.Text = str_x + "/" + str_y + "="
                                        + str_z;
      }
 else
```

```
txtResult.Text = Convert.ToString(z);
```

Сохраните измененное приложение. Проверьте работу всей программы. Покажите результат преподавателю.

Задание 3. Разработайте 2 программы по своему варианту.

Покажите результаты работы преподавателю. Оформить отчет по работе.

5. ОФОРМЛЕНИЕ ОТЧЕТА

Отчет по работе должен содержать:

- название и цель работы;
- номер варианта;
- для программ по своему варианту – текст задания, макет формы, текст кода обработчиков всех событий, схемы алгоритмов обработчиков всех событий, результаты работы программы.

6. БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Шилдт Г. - C# 4.0: полное руководство. Издательство: Вильямс, 2011 г.

2. Основы программирования на C#: http://www.intuit.ru/studies/courses/2247/18/info