Этап 2. Введение в визуальное конструирование

Создание нового приложения

➢ 1. Запустите Visual Studio 2015. *Скорее всего, после установки в меню Пуск на Вашем компьютере появился раздел, отмеченный подобным ярлыком*  .

В первый раз после запуска Visual Studio 2015 на экране могут появиться несколько запросов, уточняющих некоторые особенности работы и сохранения приложений на вашем компьютере. Поэтому отнеситесь к первому запуску очень внимательно.

Главное окно запуска приложений представлено на Рисунке 5.

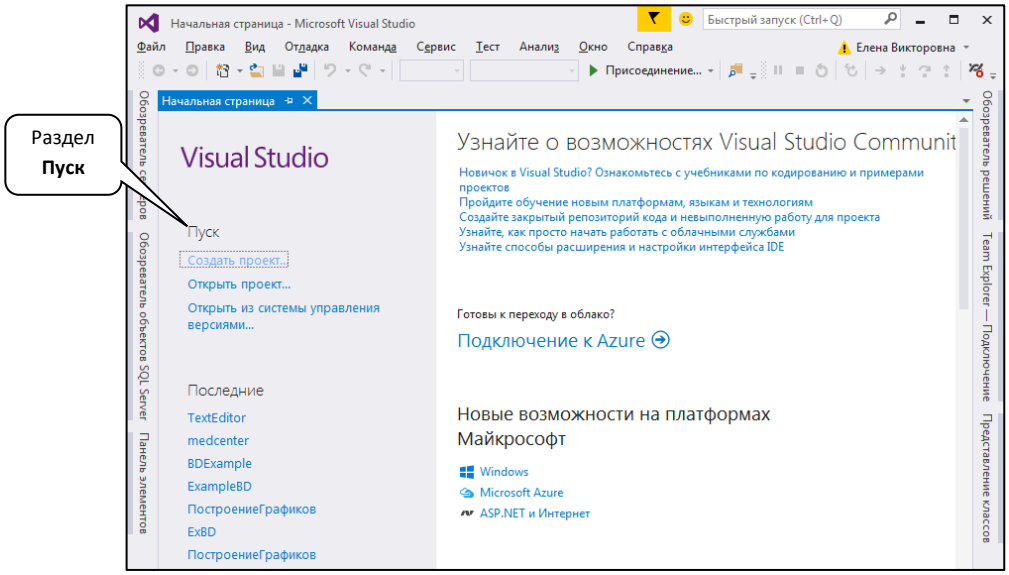

Рисунок 5. -- Главное окно Visual Studio 2015

Раздел ***Пуск*** позволяет открыть существующий проект или создать новый.

➢ 2. В разделе **Пуск** выберите пункт ***Создать проект***. Учитывая, что Visual Studio позволяет создавать разные типы приложений и выбирать из списка язык для разработки, выберите в окне **Создание проекта** тип приложения -- **Windows Forms** и отметьте в списке шаблонов (левое меню) язык C# (см. рисунок 6.)

Важно на этом этапе указать место для сохранения решения (в нижней части окна строка **Расположение**), задать имя для решения и проекта. По умолчанию эти имена совпадают, что не всегда удобно. Задайте имя решению – **Практикум1**, а имя первого проекта – PayConnect1. Проверьте, чтобы была включена опция «*Создать каталог для решения*» (см. рисунок 6).

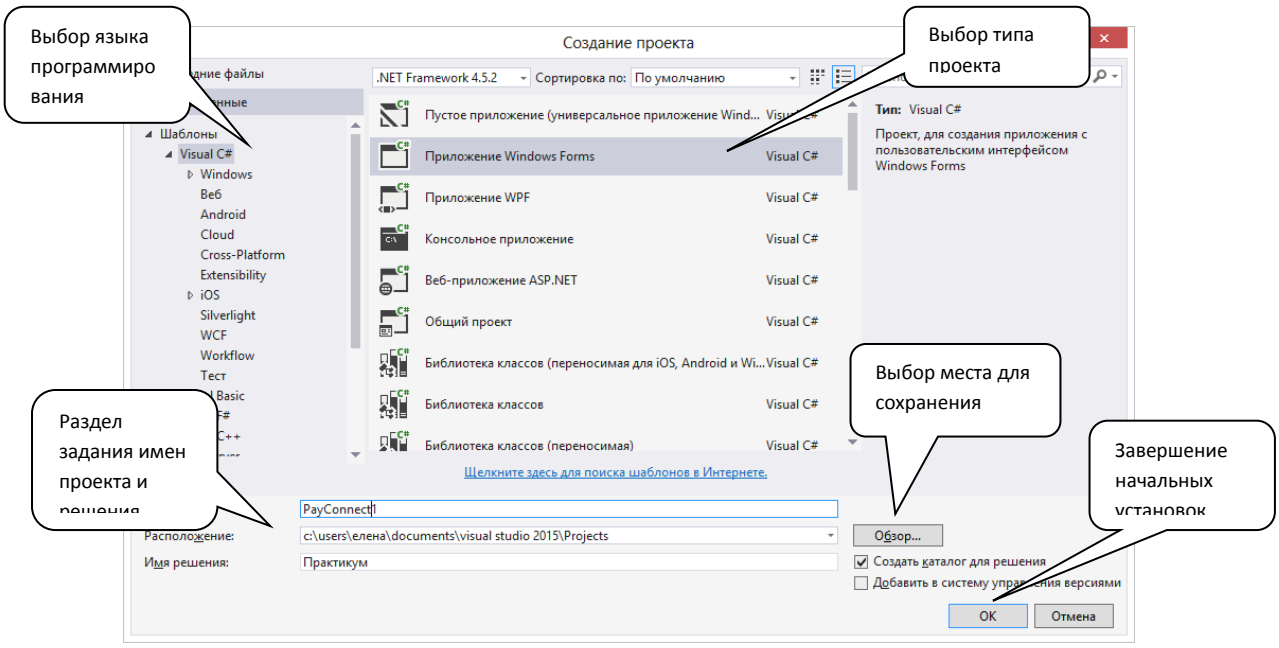

Рисунок 6. -- Первоначальные установки для вновь создаваемого решения

Нажмите **ОК**, чтобы завершить ввод первоначальных установок для решения.

➢ 3. Окно с пустой формой проекта – основное окно при визуальном проектировании. Учитывая сложную структуру проектов, создаваемых в Visual Studio, в пустом проекте автоматически будет создано несколько файлов, служебных ссылок и т.д. Доступ к этим файлам можно получить, открыв окно **Обозреватель решений**, расположенное справа.

Если окно Обозревателя решений не открылось автоматически, то следует воспользоваться меню **Вид** (Рисунок 7) или сочетанием клавиш *Ctrl+Alt+L*.

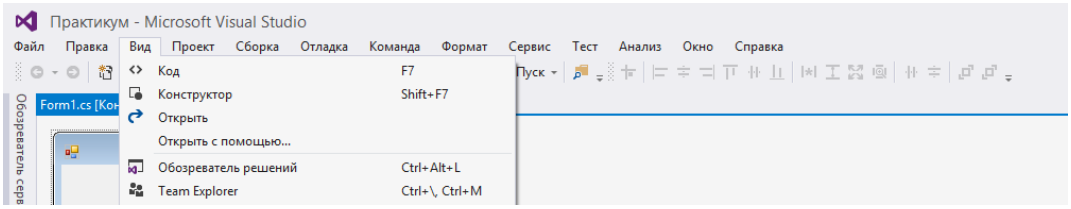

Рисунок 7. -- Открытие окна Обозреватель решений

Вам будет доступна древовидная структура нового проекта (Рисунок 8).

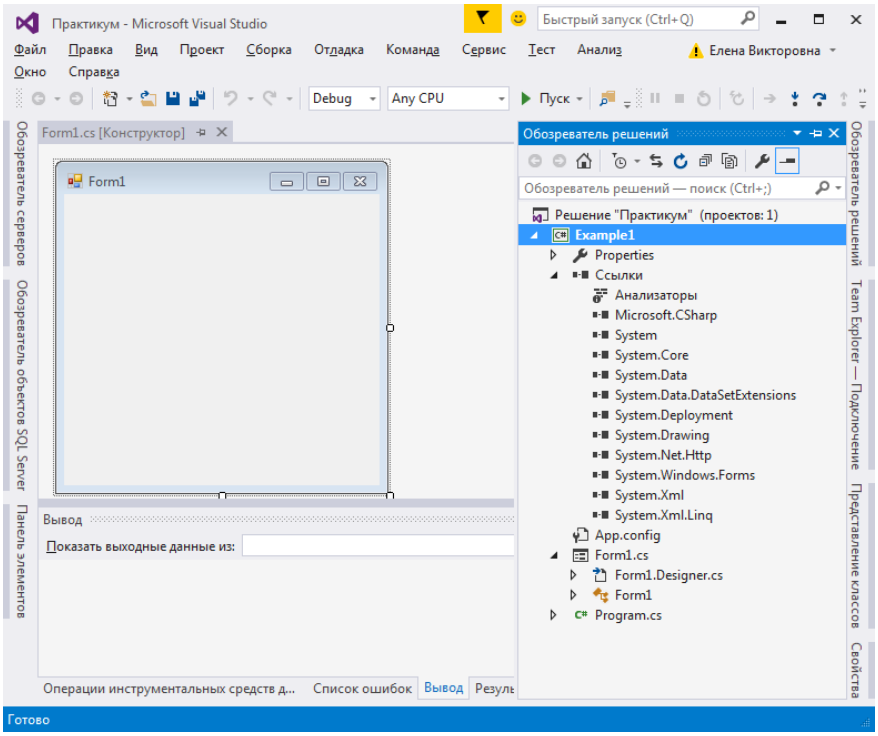

Рисунок 8. -- Вид формы созданного приложения и окно Обозреватель решения

Обозреватель решений обеспечивает упорядоченное представление проектов и их файлов, а также быстрый доступ к связанным с ними файлам.

РЕКОМЕНДАЦИЯ: Ознакомьтесь с разделом Обозреватель решений на сайте msdn.com https://msdn.microsoft.com/ru-ru/library/26k97dbc%28v=vs.90%29.aspx?f=255&MSPPError=-2147217396

Когда в Visual Studio создается проект для выбранного шаблона, в проект добавляется набор ссылок, необходимых именно для этого шаблона. Список ссылок расположен под узлом Ссылки. Некоторые ссылки обозначают пространства имен, например System.

Приступая к программированию приложения, следует продумать логику его работы, т.е. алгоритм действий пользователя, приводящий его к нужному результату. Логика работы пользователя отличается в зависимости от того с каким типом приложения он работает. Задача программиста обеспечить наиболее комфортную и понятную логику работы, сведя к минимуму возможные ошибки. Начинающие программисты считают, что создавать приложения с визуальным интерфейсом проще, но они ошибаются.

Подробнее: ☞7. ОТЛИЧИЕ ЛОГИКИ ПРОГРАММИРОВАНИЯ ПРИЛОЖЕНИЙ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ

Практикум 2: Приложение «Сотовая связь»

Постановка задачи

Рассчитать стоимость услуг сотового оператора в зависимости от количества минут исходящих звонков и количества отправленных SMS-сообщений. Пользователь может быть подключен к одному из трех предлагаемых тарифных планов. Кроме этого, стоимость обслуживания зависит от того, является ли система оплаты авансовой.

«Простой тариф» -- стоимость минуты составляет 1,75 рубля при общей продолжительности разговоров до 60 минут. Все разговоры сверх этого времени оплачиваются по 90 копеек за минуту. Стоимость одного смс сообщения составляет 80 копеек.

«Всё по рублю» -- все минуты разговоров и сообщения стоят по 1 рубль.

Акция «Лето» -- 50 рублей за все разговоры, продолжительностью до 100 минут включительно, сверх этого времени оплата по 1,5 рубля за минуту. Сообщения оплачиваются по 80 копеек.

Время разговоров округляется до целого в большую сторону.

При авансовой системе оплаты стоимость оплаты по всем тарифным планам уменьшается на 15% от общей суммы.

На рисунке 9 представлен вид окна работающей программы (Рисунок 2.5).

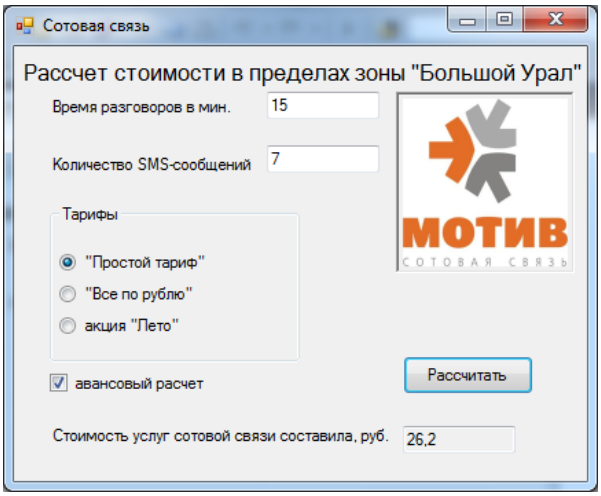

Рисунок 9. -- Вид приложения «Сотовая связь».

Упрощенный[1] алгоритм работы с программой заключается в следующем:

- пользователь вводит количество минут исходящих разговоров (может быть дробным);
- количество отправленных SMS-сообщений (целое число);
- отмечает щелчком мыши один из вариантов тарифных планов;
- отмечает щелчком мыши согласие или несогласие на авансовую систему оплаты;
- щелчком мыши по кнопке «Рассчитать» программа вычисляет стоимость услуг и выводит результат в текстовое окно.

Элементы управления

Интерфейс приложения обеспечивается элементами управления, собранными на панели элементов. Для доступа к визуальным компонентам (элементам управления) выберите ***Вид – Панель элементов*** (или сочетание клавиш ***Ctr+Alt+X***). Панель элементов может быть представлена в виде плавающей области (Рисунок 10). В среде разработки Visual Studio существует большой выбор компонентов[2], поэтому для удобства поиска они сгруппированы по областям применения.

[1] Упрощение на данный момент заключается в том, что мы не предусмотрим реакцию программы на некорректные действия пользователя.

[2] В литературе существует некоторое дублирование понятий «компонент» и «элемент управления». Чаще всего авторы под этими терминами понимают одно и то же – элемент пользовательского интерфейса, размещаемый на форме.

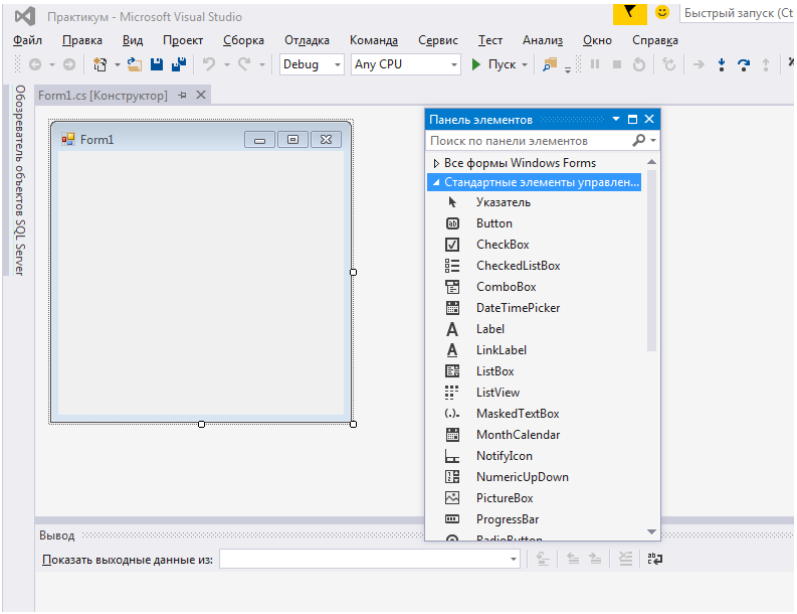

Рисунок 10. -- Фрагмент панели инструментов Стандартная

Прежде чем выбрать тот или иной компонент и поместить его на форме, необходимо определиться с тем, какие компоненты понадобятся в конкретном приложении. Исходя из описанного ранее алгоритма работы пользователя с приложением, нам понадобятся (в скобках указаны стандартные названия наиболее подходящих компонентов):

- текстовые окна для ввода числовых значений с возможностью редактирования (`textbox`, текстовое поле);
- текстовые окна для различных надписей, помогающих пользователю ориентироваться в окне приложения (без возможности редактирования пользователем) (`label`, метка);
- несколько кнопок-селекторов, позволяющих отметить один из нескольких возможных вариантов тарифных планов (`radioButton`, селектор);
- элемент для установки флажка, подтверждающий выбор авансовой системы оплаты (`checkBox`, флажок);
- кнопка для запуска программы расчета (`button`, кнопка);
- окно для вывода результата, без возможности редактирования (`textBox`, текстовое поле)
- картинка с эмблемой сотовой компании для украшения формы (`pictureBox`, изображение).

На рисунке 11 отмечено размещение некоторых компонентов на форме.

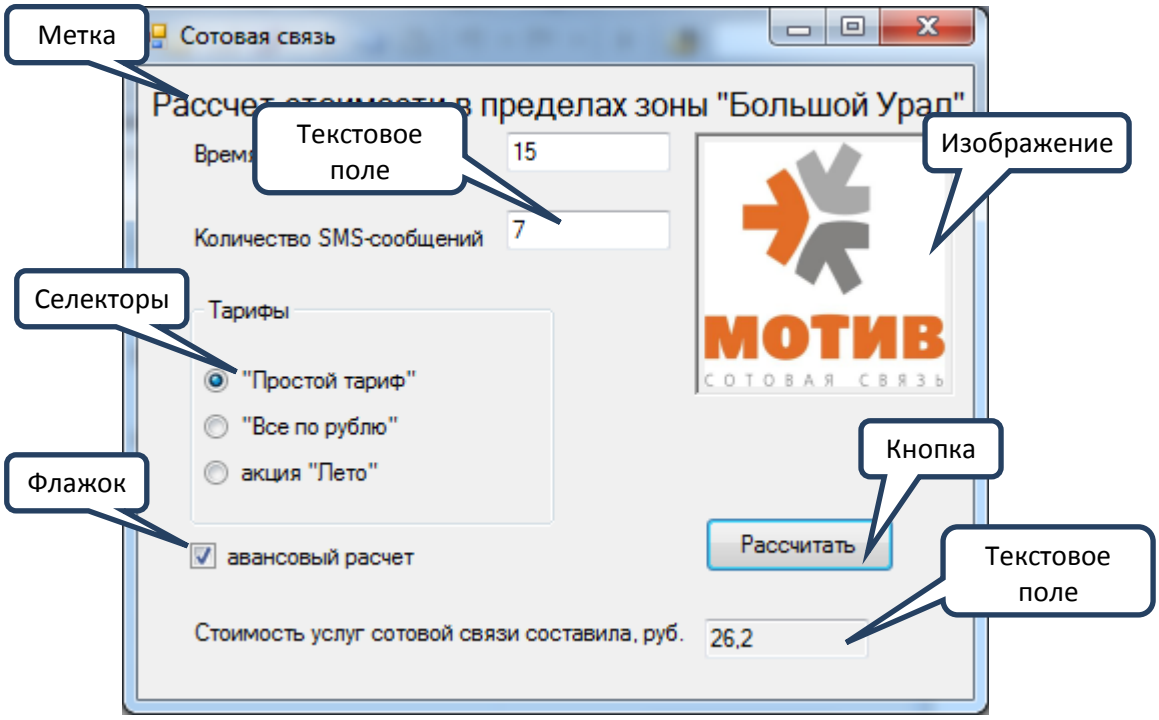

Рисунок 11. Типы компонентов, размещенные на форме приложения «Сотовая Связь»

Для размещения компонентов на форме следует открыть группу *Стандартные* элементы управления в списке компонентов окна *Элементы управления* и перетащить мышью на форму необходимые компоненты. В момент перемещения среда будет помогать в выравнивании размещаемого компонента относительно других, уже имеющихся на форме (рисунок 12).

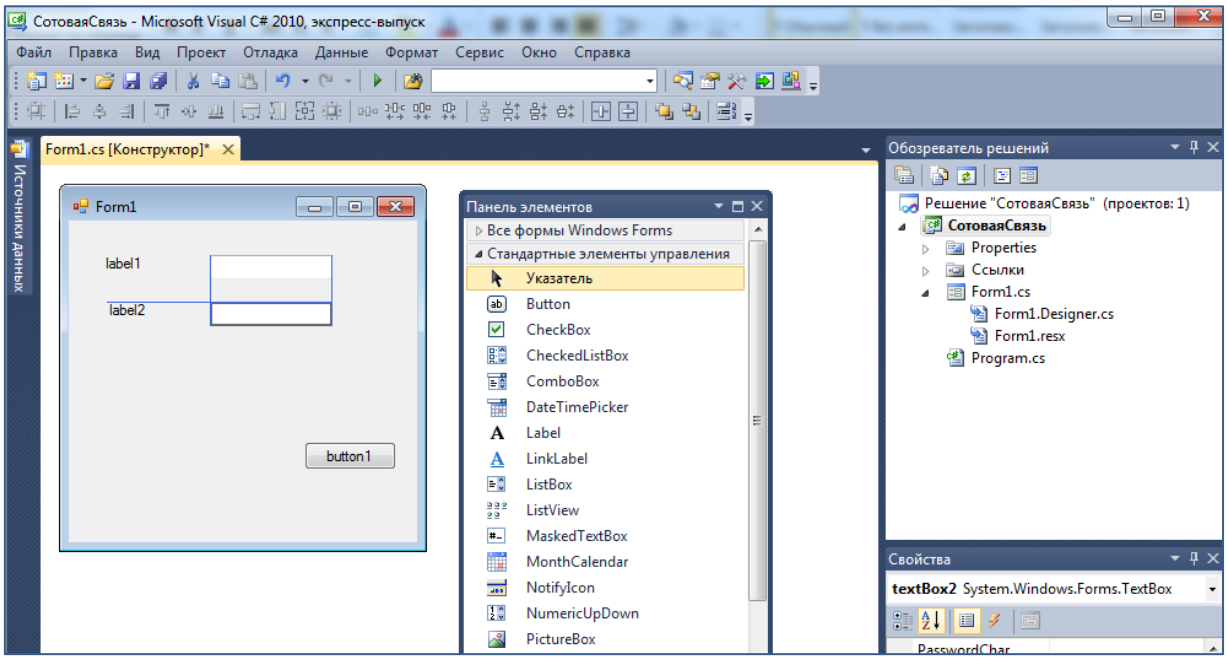

Рисунок 12. Момент размещения компонента на форме в процессе конструирования пользовательского интерфейса.

Компоненты *Селекторы* (*RadioButton*) применяют тогда, когда пользователю необходимо предоставить возможность выбора между несколькими взаимоисключающими опциями. Чтобы сгруппировать переключатели для образования единого логического блока, необходимо использовать компонент-контейнер, в данном случае, использован компонент *Рамка (GroupBox).*

Задание 1.

Руководствуясь рисунками 9 и 11, разместите на форме необходимые компоненты.

Итог задания 1 может быть примерно следующим (Рисунок 13)

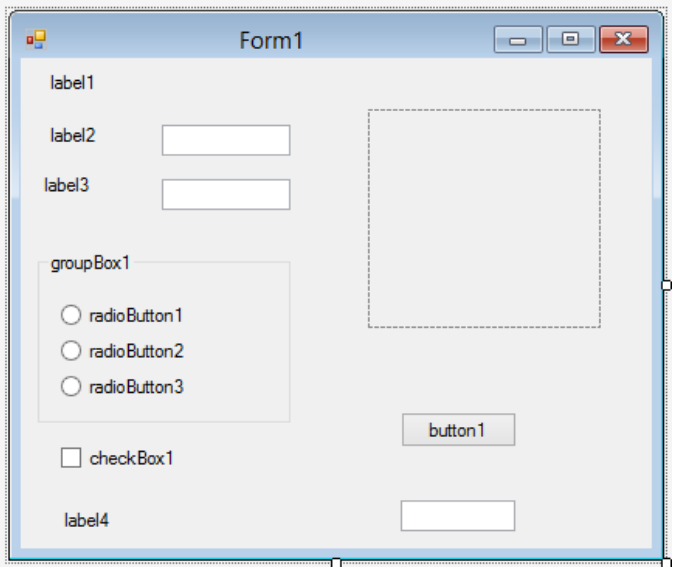

Рисунок 13. – Расположение компонентов на форме

Далее мы прейдем к настройке свойств компонентов.

Свойства компонентов

В терминах объектно-ориентированного программирования компоненты, размещенные на форме, называют ***объектами***. Судя по рисунку 13, на форме размещено несколько объектов *Метка*, несколько объектов *Текстовое поле*, *Кнопка*, группа *Селекторов* и *Флажок*. Из рисунка видно, что даже одинаковые объекты Метка отличаются по своим ***свойствам*** (например, размер шрифта для вывода заголовка). Для того чтобы видеть свойства выделенного объекта необходимо на *Стандартной панели* инструментов открыть окно *Свойства* (см. рисунок 14).

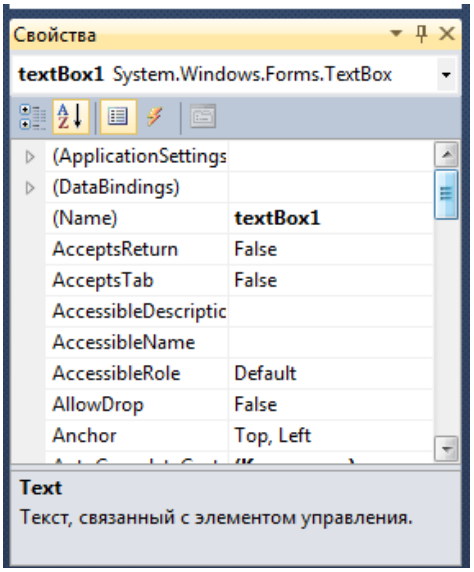

Рисунок 14. Окно свойств компонента Текстовое поле

У разных компонентов могут быть разные свойства, но некоторые из свойств обязательны для всех компонентов формы. К таким обязательным свойствам компонентов относится *Имя* (*Name*).

Как уже говорилось выше, компонент – это имеющийся в среде разработки готовый элемент управления. В процессе перетаскивания компонента, например, текстового поля на форму, программа автоматически создает новый объект выбранного типа (на языке объектно-ориентированного программирования – экземпляр класса *textbox*) и автоматически присваивает ему уникальное имя *textBox1*. Это имя (*Name*) отображается в окне Свойства данного компонента. Часто программист меняет заданное автоматически имя, так как при большом числе компонентов, размещенных на форме, трудно определить, для чего именно служит компонент с номером, например, *textBox23*. Так же как и в случае с именами переменных, стараются компонентам давать более осмысленные имена. Например, *textBoxLenghtPhone*[3]. Но изменять имена следует только у тех компонентов, которые будут в дальнейшем использоваться в программе. Вспомогательным компонентам, например, Меткам, служащим обычно для вывода надписей на форму, менять имена не принято. У каждого типа компонентов свой набор свойств. Некоторые свойства, например, *Text*, можно найти у разных типов компонентов, но есть и уникальные. Некоторые свойства являются составными, т.е. входят в состав некоторой группы свойств. К таким, например, относится свойство *Font.Size* (т.е. свойство для управления размером шрифта, входящее в группу свойств *Font*).

Все изменения свойств вводятся и фиксируются в окне Свойства объекта и, если они влияют на внешний вид объекта, то немедленно отображаются в окне дизайнера форм. На рисунке 15 представлены фрагмент окна свойств компонента метка (*Label1*) и окно формы в режиме конструирования. У метки изменены свойства размер шрифта (*Font.size*) и *Text*, определяющие внешний вид выводимой на форму надписи.

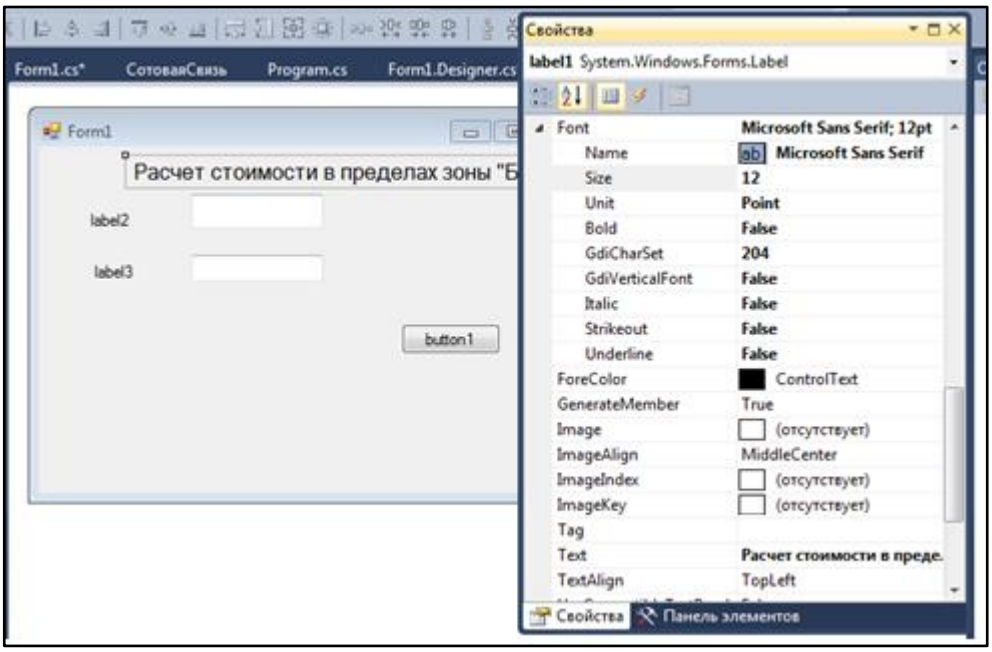

Рисунок 15. -- Окно свойств компонента Label1

Текстовое поле *TextBox3* предназначено только для вывода результата. Для того чтобы запретить пользователям ввод значений свойство *ReadOnly* данного компонента установлено в значение *True*.

Для размещения изображения на форму добавлен компонент *PictureBox*. С помощью свойства *Image* устанавливают связь с изображением, которое будет отображаться в окне. Для

[3] Язык C# позволяет использовать кириллицу в именах переменных, процедурах и т.п. Мы настоятельно не рекомендуем этого делать. Программисту придется постоянно переключать регистр раскладки, что может привести к ошибкам. Некоторые символы кириллицы и латиницы выглядят одинаково, например «с» или «а» и поиск ошибок будет требовать дополнительных усилий.

того, чтобы не зависеть от сохранности файла с изображением на локальном компьютере, файл лучше внедрить в папку *Ресурсы* проекта. Для этого вызовите контекстное меню для проекта PayConnect1 (Рисунок 16) и выберите пункт *Свойства.*

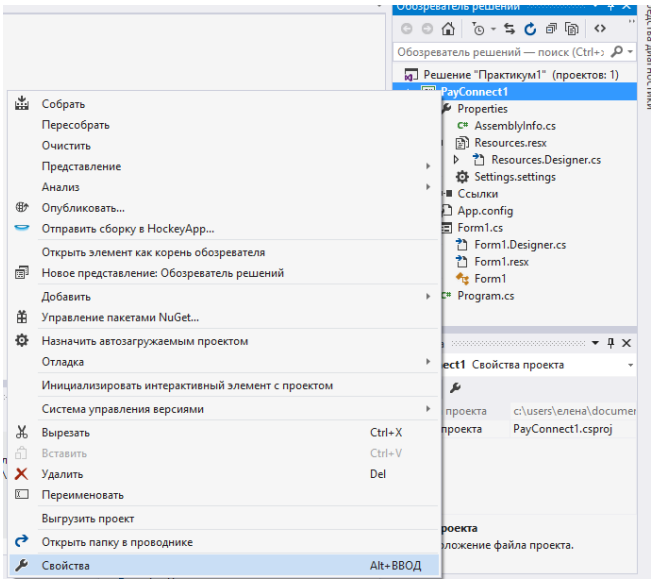

Рисунок 16. – Контекстное меню проекта PayConnect1

В окне свойств выберите Ресурсы и Добавьте существующий файл (Рисунок 17). Следует отметить, что выбрав пункт Создать изображение, Вы сможете поработать со встроенным графическим редактором.

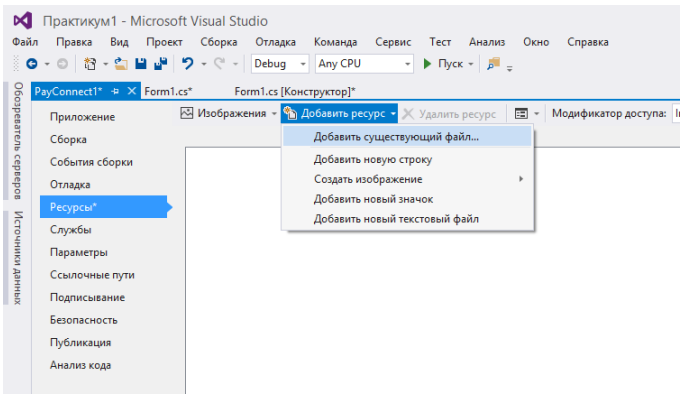

Рисунок 17. – Добавление ресурсов к проекту

На рисунке 18 представлено окно импортирования ресурсов. Там же видно, что добавить файл с изображением к ресурса проекта можно, воспользовавшись кнопкой Импорт.

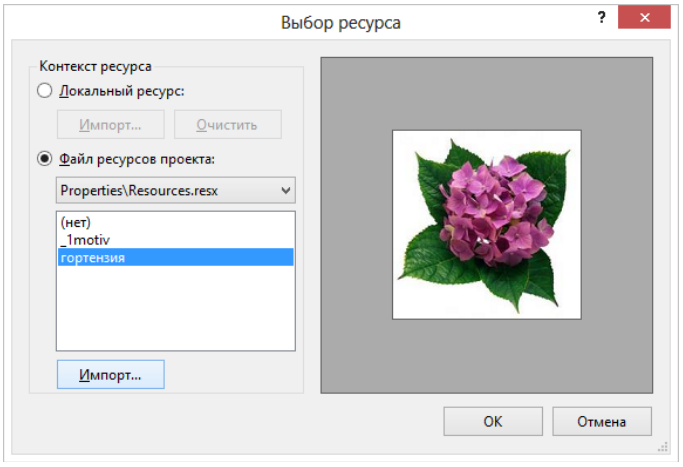

Рисунок 18. – Импортирование файла изображения

Для компонента *PictureBox* следует изменить свойство *SizeMode*, отвечающее за изменением размеров картинки в области окна. Задав ему значение *StretchImage* можно добиться того, что изображение будет точно соответствовать размеру *PictureBox*. Однако если пропорции окна не соответствуют истинному размеру изображения, то картинка будет деформирована.

Более подробную информацию обо всех свойствах компонентов следует искать в официальной справочной системе или на сайте www.msdn.ru.

Задание 2.

Задайте свойства для компонентов, исходя из собственных предпочтений и из логики приложения.

Итог задания 2 может быть, например, таким как на рисунке 19. Вы можете запустить приложение, и оно будет откомпилировано.

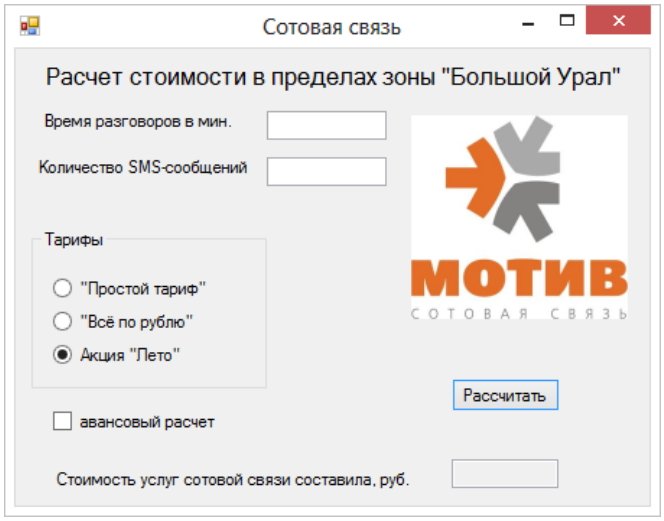

Рисунок 19. – Внешний вид запущенного приложения

На этом работа с Конструктором формы завершена. Вы всегда можете перейти в конструктор, воспользовавшись пунктом контекстного меню для Формы (Рисунок 20).

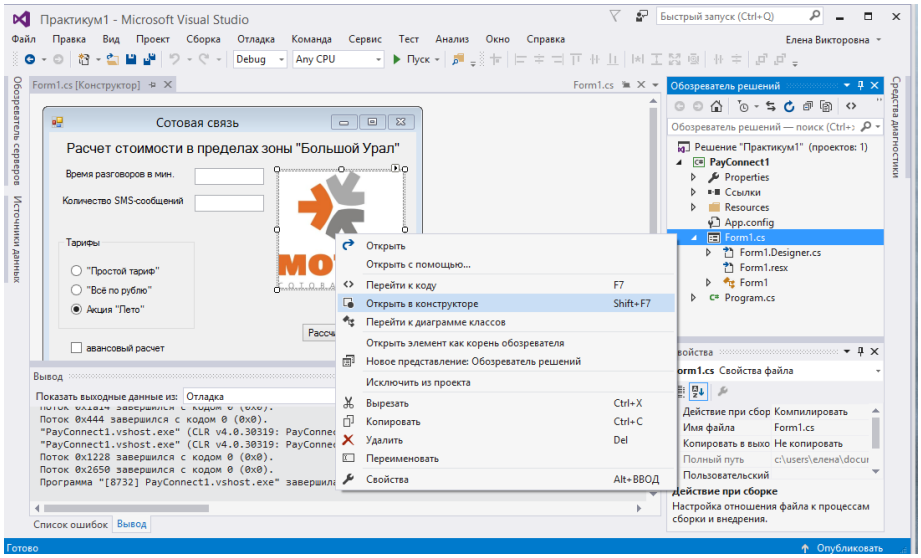

Рисунок 20. – Контекстное меню файла Form1.cs

Программирование обработчиков событий (версия 1)

Согласно разработанному алгоритму работы, по щелчку мыши по кнопке «Рассчитать» программа вычисляет стоимость услуг, и результат выводится в текстовое окно. Для компонента Кнопка щелчок мыши можно рассматривать как событие. Обычно события связаны с действиями, выполняемыми пользователем. Например, когда пользователь щелкает по кнопке, кнопка генерирует событие, указывающее, что с ней произошло. Обработка события – это средство, с помощью которого программист может снабдить данную кнопку функциональностью. Написание кода для обработчиков событий составляет основной объем работы по созданию приложения. Подробную информацию обо всех событиях основных компонентов следует искать в официальной справочной системе или на сайте www.msdn.ru.

У каждого компонента существует лишь одно событие, используемое по умолчанию. Это событие различно в зависимости от компонента. Для кнопки таким событием является щелчок (*Click*), т.е. нажатие и отпускание левой кнопки мыши в то время, когда указатель располагается над кнопкой. Событие *Click* происходит и тогда, когда кнопка находится в фокусе и пользователь нажимает клавишу <Enter>.

Перейти к коду проекта можно используя пункт Перейти к коду контекстного меню (Рисунок 20) или выполнив двойной щелчок по элементу управления.

Двойной щелчок по кнопке в режиме дизайнера формы генерирует пустой обработчик события *Click* (Рисунок 21):

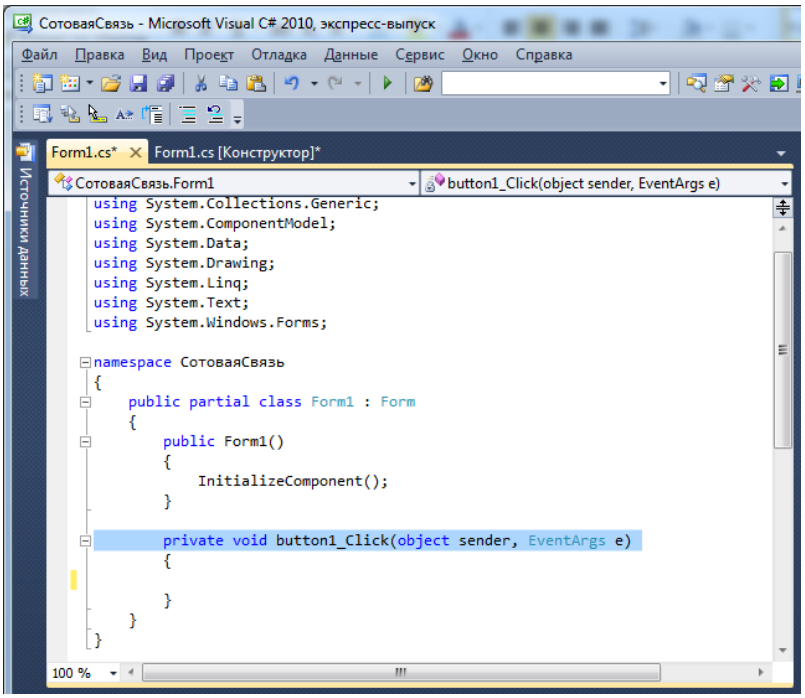

Рисунок 21. -- Окно программного кода для описания обработчика события

Файл программного кода создан средой разработки автоматически и носит имя *Form1.cs*.

Когда Visual Studio создает метод для обработки события, в качестве имени метода назначается имя компонента, за которым следует символ подчеркивания и имя обрабатываемого события.

Первый параметр события *Click* – *object sender* – содержит сведения о компоненте, на котором выполнен щелчок. В данном случае это компонент, указанный в имени. Второй параметр

– *EventArgs e* – предоставляет системе данные о произошедшем событии. В рассматриваемом случае никаких дополнительных данных не требуется.

Написание кода обработчика производится вручную. Особенно внимательно следует отнестись к именам компонентов. При обращении к свойствам компонентов настоятельно рекомендуется использовать возможности интеллектуальных подсказок среды разработки (Рисунок 22). Не следует упорствовать и вписывать свойство вручную, если оно не появляется после нажатия точки после имени компонента.

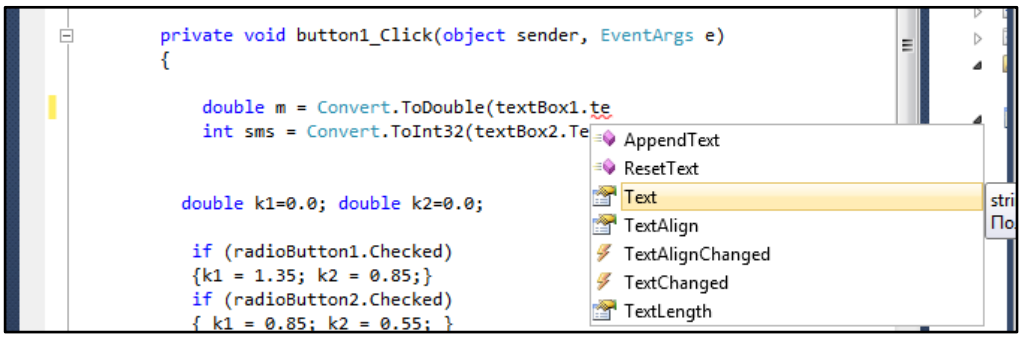

Рисунок 22. -- Режим интеллектуальной подсказки при написании обработчика события

При написании программного кода обработчиков событий для приложений Windows Forms программному изменению, в основном, подвергаются свойства расположенных на форме компонентов.

РЕКОМЕНДАЦИЯ: При отладке программного когда часто возникает необходимость закомментировать группу строк. В Visual Studio есть специальный инструмент, который позволяет выполнять эту операцию (Рисунок 23).

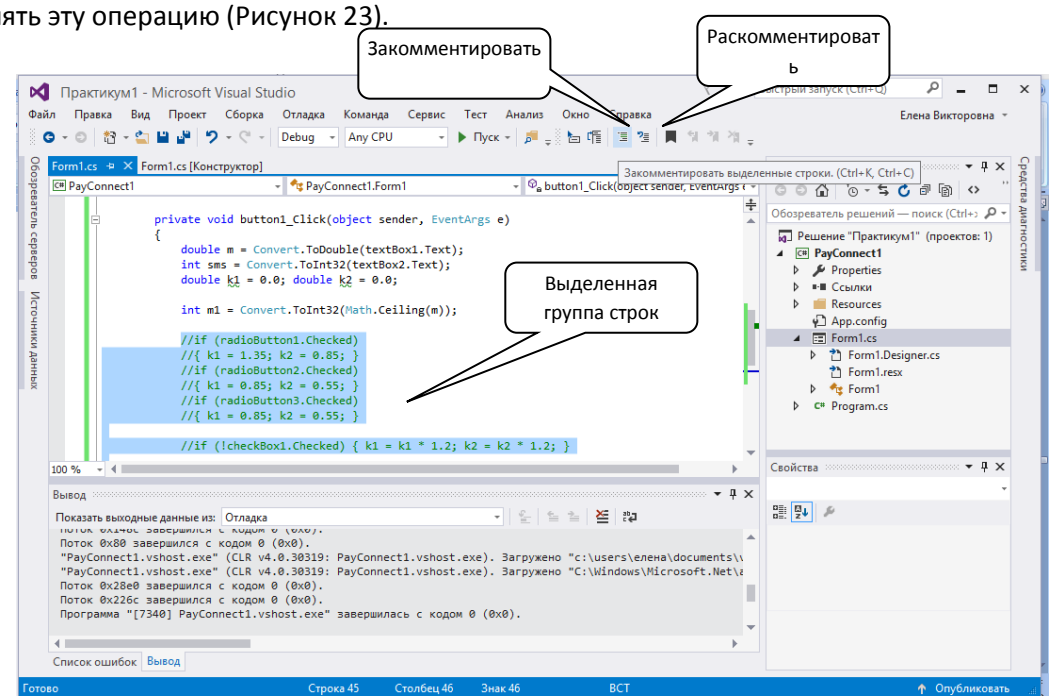

Рисунок 23. – Инструмент быстрого комментирования группы строк

Описание обработчика [4]события *Click* для рассматриваемого приложения приведено в листинге 6.

Листинг 6

```
private void button1_Click(object sender, EventArgs e)
      {
          double m = Convert.ToDouble(textBox1.Text);
```

[4] Если у Вас возникла необходимость удалить некоторые, созданные автоматически, обработчики событий, то посмотрите рекомендацию в конце данного раздела.

```
            int sms = Convert.ToInt32(textBox2.Text);
            double k1 = 0.0; double k2 = 0.0;

            int m1 = Convert.ToInt32(Math.Ceiling(m));

            if (radioButton1.Checked)
            {
                if (m1 <= 60) k1 = 1.75 * m1; else k1=60 * 1.75 + (m1 - 60) * 0.9;
                k2 = sms * 0.8;
                 }
            if (radioButton2.Checked)
            { k1 = m1; k2 = sms; }
            if (radioButton3.Checked)
            {
                if (m1 <= 100) k1 = 50; else k1 = 50 + (m1 - 100) * 1.5;
                k2 = sms * 0.8;
            }

            if (checkBox1.Checked) { k1 = k1 * 0.85; k2 = k2 *0.85 ; }

            double sum = k1 + k2;
            textBox3.Text = sum.ToString();

               }
```

Проанализируем код обработчика. Описанные переменные m, m1, k1, k2 и sms являются локальными, т.е. действующими в пределах данного обработчика. В других обработчиках они будут не доступны.

m1 – округленное до целого в бОльшую сторону значение вещественной переменной *m*, количество минут разговоров;

k1 – плата за разговоры;

k2 – плата за SMS-сообщения.

Компонент *Селектор* (*radioButton*) имеет свойство Checked, указывающее на состояние компонента. Значение этого свойства является логическим и равно true, если компонент отмечен (точка внутри окружности компонента на форме). В противном случае оно равно *false*. Аналогичное свойство есть и у компонента *Флажок* (*checkbox*). Результат вычисления выводится в текстовое окно (*textBox3*), путем изменения его свойства Text. Так как значением текстового поля может быть только значение типа string, то значение переменной sum преобразовано из числа в строку с помощью стандартного метода ToString( ).

Рассмотрим автоматически сгенерированную часть кода. В самое начало файла исходного текста приложения мастер проектов вставил строки, подключающие несколько пространств имен (name space) с помощью ключевого слова using:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

Пространство имен System содержит определение фундаментальных и базовых классов, определяющих типы данных, события, обработчики событий и другие, необходимые в каждом приложении компоненты.

Пространство имен System.Drawing необходимо для доступа к интерфейсу графических устройств.

Пространство имен System.Windows.Forms — в нем определены классы, реализующие поведение оконных форм, составляющих базу оконных приложений Microsoft windows на платформе Microsoft .NET Frameworks.

Создаваемое приложение также определяет собственное пространство имен – *PayConnect1*. Одним из первых вызывается метод InitializeComponent( ), который инициализирует все компоненты, расположенные на форме: поля, кнопки, переключатели и т. п. Реализация метода описана в файле Form1.Disigner.cs (Рисунок 24)

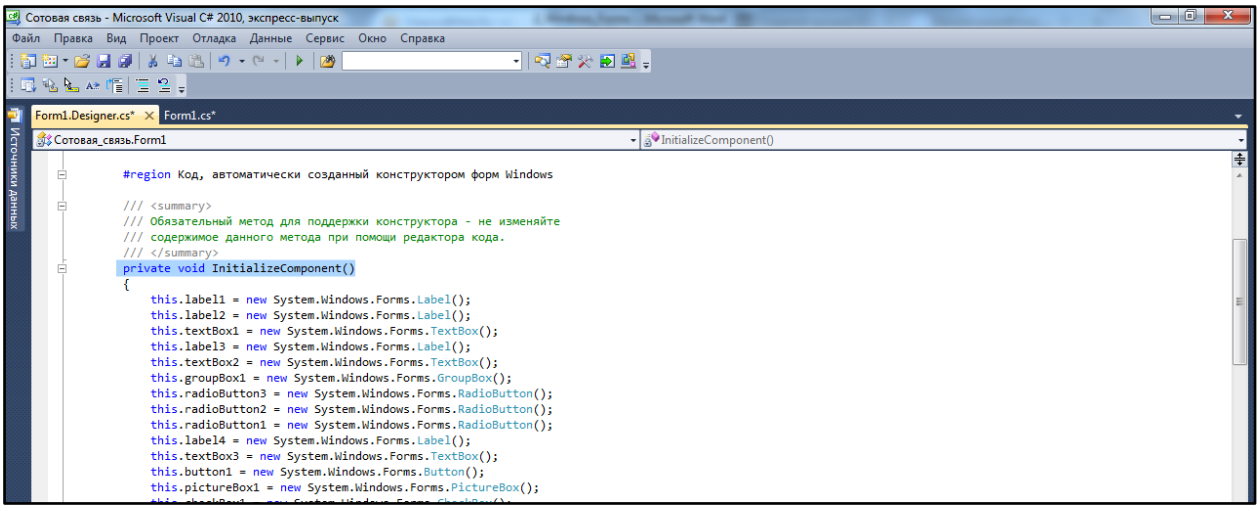

Рисунок 24 Фрагмент кода метода InitializeComponent( )

Изменять созданный автоматически метод `InitializeComponent( )` вручную в редакторе кода не следует!

В процессе конструирования формы часто возникает ситуация, когда случайный двойной щелчок по компоненту приводит к появлению пустых обработчиков. Например, на Рисунке 25 приведен пустой обработчик для компонента radioButton2.

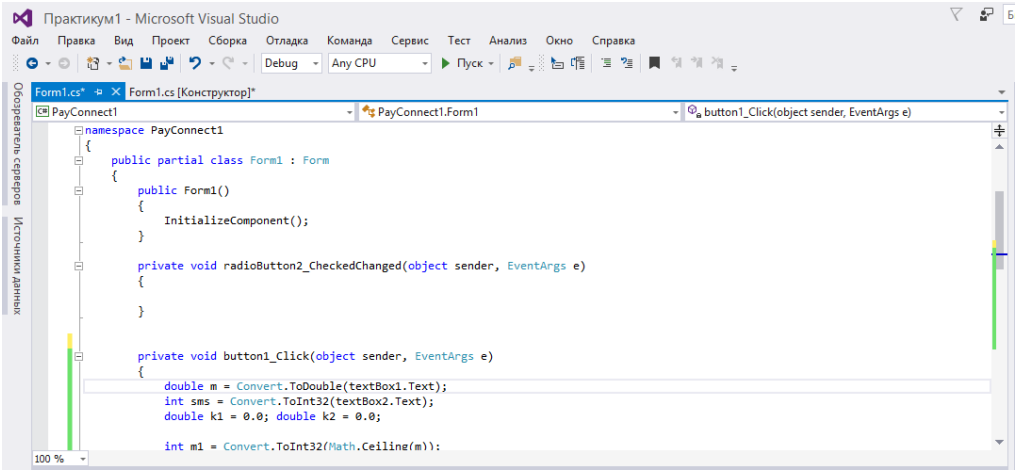

Рисунок 25. – Пример пустого обработчика события

Код лишнего обработчика необходимо удалить из основного кода программы (файл Form1.cs) и из описания метода `InitializeComponent( ) (файл Form1.Designer.cs )` `Рисунок 26.`

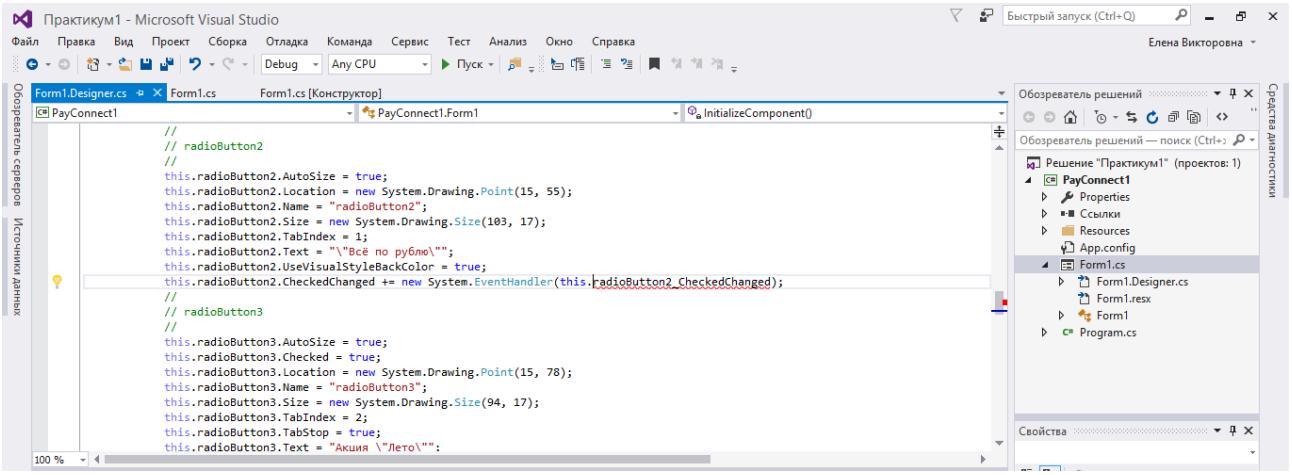

Рисунок 26. – фрагмент удаляемого кода в описании метода `InitializeComponent( )`

Задание 3.

Напишите код обработчика для кнопки «Рассчитать» и протестируйте работу приложения.

Итог задания 3 Вы можете проверить, запустив проект ***PayConnect1*** из папки ***Практикум1***.

Программирование обработчиков событий (версия 2)

В созданном коде есть много недоработок. Убедитесь в этом сами, запустив проект при следующих условиях:

1. Оставьте пустым одно из текстовых полей;
2. Введя неверный формат данных (например, запишите вещественное число, используя в качестве разделителя точку).

Кроме того, не заданы начальные установки переключателей. Например, желательно, чтобы при запуске приложения свойство `checkBox1.Checket` имело значение `true` и был выбран тариф *«Акция Лето».*

Желательно, чтобы кнопка «Рассчитать» была неактивна до тех пор, пока в оба текстовых окна не будет введено значение.

Курсор при запуске должен быть установлен в текстовое поле1.

Предварительная установка значений.

В окне свойств для Form1 переключитесь на список событий (Рисунок 27).

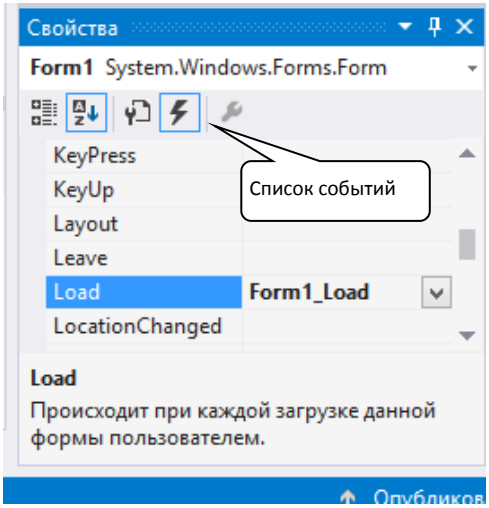

Рисунок 27.—Список событий в окне свойств компонента Form1

Двойным щелчком по пустому полю рядом с событием Load вызовите заголовок для кода обработчика события. Код приведен в листинге 7.

Листинг 7

```
private void Form1_Load(object sender, EventArgs e)
        {
            radioButton3.Checked = true;
            checkBox1.Checked = true;
            button1.Enabled = false;
            textBox1.Focus();
               }
```

Контроль вводимых значений.

Доступ к контролю вводимых символов в текстовое поле можно получить через обработчик события KeyPress компонента textBox1. **Не забывайте**, что добавление обработчиков событий возможно только **через окно свойств!**

1. Запретим ввод в текстовое поле любых символов, кроме цифр;
2. Разрешим вводить только один десятичный разделитель;
3. Исправим ошибку пользователя при вводе десятичного разделителя – заменим введенную точку на запятую;
4. Запретим вводить запятую первым символом в строке.

Код представлен в листинге 8.

Листинг 8

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            //Ввод только цифр
            if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
            {    //введена цифра
                return;
            }
            //Замена введенной точки на запятую
            if (e.KeyChar == '.') e.KeyChar = ',';

            //Запрет повторного ввода запятой и запрет ввода запятой первым символом

            if (e.KeyChar == ',')
            {
                if ((textBox1.Text.IndexOf(',') != -1)|| (textBox1.Text.Length == 0))
                { e.Handled = true; }
                return;
            }
            //по нажатию на Enter переведем фокус на другое текстовое поле
            if (Char.IsControl(e.KeyChar))
            {
                if (e.KeyChar == (char)Keys.Enter)
                    textBox2.Focus();

            }
            //Разрешаем нажимать клавишу BackSpace
            if (Char.IsControl(e.KeyChar))
            {
                if (e.KeyChar == (char)Keys.Back)
                    return;

            }

            e.Handled = true;
        }
```

Задание 4.

Дополните код проекта строками, обеспечивающими контроль вводимых значений.

Допишите аналогичный код для текстового окна textBox2, учтя, что в это окно должны вводиться только целые значения. По окончании ввода (при условии, что в свойство textBox1.Text не пустое) должна активироваться кнопка «Рассчитать».

Протестируйте работу приложения.

Итог задания 4 можно сравнить с кодом проекта PayConnect2.

Приложение

7. Отличие логики программирования приложений с графическим интерфейсом

При выполнении программы, записанной в виде консольного приложения, порядок действий определяет сама программа. Каждая строка кода выполняется в определенной последовательности. Программист определяет логику работы пользователя: в определенный момент встречается развилка, далее серия последовательных команд. Собственно пользователь достаточно пассивен. В момент, когда понадобятся новые данные, программа сама приостанавливает работу и будет ожидать ввода необходимых значений. И эта логика действий заложена программистом еще на этапе написания программного кода.

Иначе обстоит дело с программами с графическим интерфейсом. В таких приложениях активную роль играет пользователь. Он сам решает, в какой момент и какие данные он будет вводить, ничто не мешает нажать на кнопку «Рассчитать», оставив пустыми одно или несколько текстовых окон. Разумеется, в таком случае будет выдан неверный результат или программа остановит работу в результате невозможности выполнения действий («аварийная ситуация»). При работе с приложениями с графическим интерфейсом работа программиста зачастую усложняется, т.к. он должен предусмотреть возможные варианты действий пользователя и каким-то образом, путем вывода подсказок или ввода ограничений, помочь пользователю получить действительно правильный вариант решения. Таким образом, чтобы сделать приложение с графическим интерфейсом действительно удобным для пользователя, программисту приходится писать дополнительный программный код.

Правильно написанные приложения с графическим интерфейсом удобны для работы пользователей, так как они наглядны и не требуют дополнительных действий от оператора – установить флажок щелчком мыши проще, чем вводить с клавиатуры символы. Консольные же приложения удобнее и проще для программиста, т.к. они описывают суть алгоритма, не заставляя отвлекаться на несущественные детали. Поэтому многие служебные программы, утилиты или библиотечные файлы пишут в виде консольных приложений.

☞Назад к основному тексту