Л а б о р а т о р н а я р а б о т а № 7

**МЕТОДЫ. РЕКУРСИЯ.   
ВЫЧИСЛЕНИЕ СУММ С ВЛОЖЕННЫМИ СУММАМИ/ПРОИЗВЕДЕНИЯМИ** 

**Цель работы:**

* Освоение приемов работы с методами;
* Закрепление приемов работы с итерационными операторами;
* Освоение приемов работы с рекурсивными функциями;
* Реализация приложений Windows Forms.

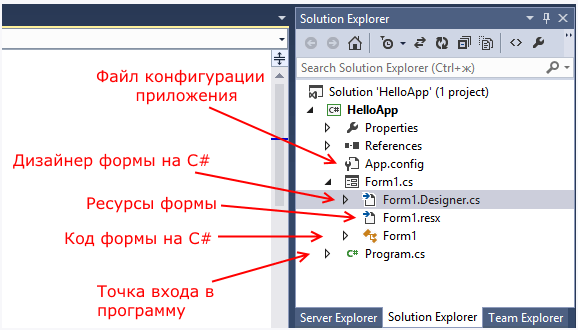
**Краткие теоретические сведения.**

***Основы форм***

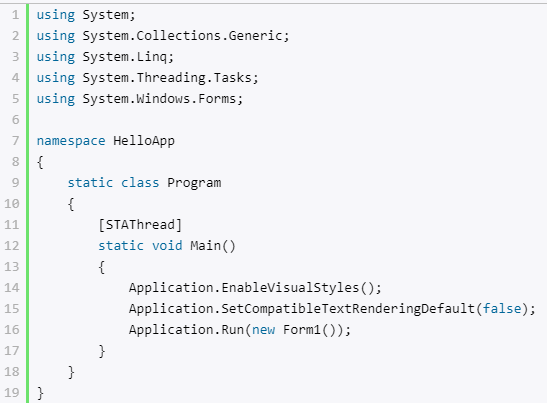
Внешний вид приложения является нам преимущественно через формы. Формы являются основными строительными блоками. Они предоставляют контейнер для различных элементов управления. А механизм событий позволяет элементам формы отзываться на ввод пользователя, и, таким образом, взаимодействовать с пользователем.

При открытии проекта в Visual Studio в графическом редакторе мы можем увидеть визуальную часть формы - ту часть, которую мы видим после запуска приложения и куда мы переносим элементы с панели управления. Но на самом деле форма скрывает мощный функционал, состоящий из методов, свойств, событий и прочее. Рассмотрим основные свойства форм.

Если мы запустим приложение, то нам отобразится одна пустая форма. Однако даже такой простой проект с пустой формой имеет несколько компонентов:



Несмотря на то, что мы видим только форму, но стартовой точкой входа в графическое приложение является класс *Program*, расположенный в файле *Program.cs*:



Сначала программой запускается данный класс, затем с помощью выражения **Application.Run(new Form1())** он запускает форму *Form1*. Если вдруг мы захотим изменить стартовую форму в приложении на какую-нибудь другую, то нам надо изменить в этом выражении *Form1* на соответствующий класс формы.

Сама форма сложна по содержанию. Она делится на ряд компонентов. Так, в структуре проекта есть файл **Form1.Designer.cs.** Здесь объявляется частичный класс формы Form1, которая имеет два метода: **Dispose(),** который выполняет роль деструктора объекта, и **InitializeComponent(),** который устанавливает начальные значения свойств формы.

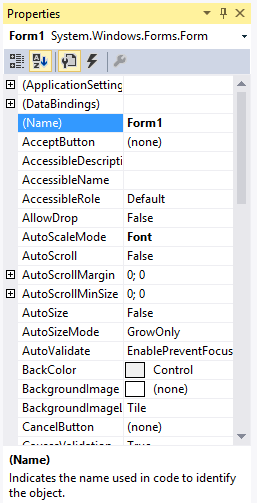
При добавлении элементов управления, например, кнопок, их описание также добавляется в этот файл. На практике мы редко будем сталкиваться с этим классом, так как они выполняет в основном дизайнерские функции - установка свойств объектов, установка переменных.

Еще один файл *- Form1.resx* - хранит ресурсы формы. Как правило, ресурсы используются для создания однообразных форм сразу для нескольких языковых культур.

И более важный файл - *Form1.cs*, который в структуре проекта называется просто *Form1*, содержит код или программную логику формы. По умолчанию здесь есть только конструктор формы, в котором просто вызывается метод **InitializeComponent(),** объявленный в файле дизайнера **Form1.Designer.cs**. Именно с этим файлом мы и будем больше работать.

***Основные свойства форм***

С помощью специального окна *Properties* (Свойства) справа Visual Studio предоставляет нам удобный интерфейс для управления свойствами элемента:



Большинство этих свойств оказывает влияние на визуальное отображение формы. Пробежимся по основным свойствам:

* **Name**: устанавливает имя формы - точнее имя класса, который наследуется от класса Form
* **BackColor**: указывает на фоновый цвет формы. Щелкнув на это свойство, мы сможем выбрать тот цвет, который нам подходит из списка предложенных цветов или цветовой палитры
* **BackgroundImage**: указывает на фоновое изображение формы
* **BackgroundImageLayout**: определяет, как изображение, заданное в свойстве *BackgroundImage*, будет располагаться на форме.
* **ControlBox**: указывает, отображается ли меню формы. В данном случае под меню понимается меню самого верхнего уровня, где находятся иконка приложения, заголовок формы, а также кнопки минимизации формы и крестик. Если данное свойство имеет значение false, то мы не увидим ни иконку, ни крестика, с помощью которого обычно закрывается форма
* **Cursor**: определяет тип курсора, который используется на форме
* **Enabled**: если данное свойство имеет значение *false*, то она не сможет получать ввод от пользователя, то есть мы не сможем нажать на кнопки, ввести текст в текстовые поля и т.д.
* **Font**: задает шрифт для всей формы и всех помещенных на нее элементов управления. Однако, задав у элементов формы свой шрифт, мы можем тем самым переопределить его
* **ForeColor**: цвет шрифта на форме
* **FormBorderStyle**: указывает, как будет отображаться граница формы и строка заголовка. Устанавливая данное свойство в *None* можно создавать внешний вид приложения произвольной формы
* **HelpButton**: указывает, отображается ли кнопка справки формы
* **Icon**: задает иконку формы
* **Location**: определяет положение по отношению к верхнему левому углу экрана, если для свойства *StartPosition* установлено значение *Manual*
* **MaximizeBox**: указывает, будет ли доступна кнопка максимизации окна в заголовке формы
* **MinimizeBox**: указывает, будет ли доступна кнопка минимизации окна
* **MaximumSize**: задает максимальный размер формы
* **MinimumSize**: задает минимальный размер формы
* **Opacity**: задает прозрачность формы
* **Size**: определяет начальный размер формы
* **StartPosition**: указывает на начальную позицию, с которой форма появляется на экране
* **Text**: определяет заголовок формы
* **TopMost**: если данное свойство имеет значение *true*, то форма всегда будет находиться поверх других окон
* **Visible**: видима ли форма, если мы хотим скрыть форму от пользователя, то можем задать данному свойству значение *false*
* **WindowState**: указывает, в каком состоянии форма будет находиться при запуске: в нормальном, максимизированном или минимизированном

***Программная настройка свойств***

С помощью значений свойств в окне Свойства мы можем изменить по своему усмотрению внешний вид формы, но все то же самое мы можем сделать динамически в коде.

***Элементы управления***

Элементы управления представляют собой визуальные классы, которые получают введенные пользователем данные и могут инициировать различные события. Все элементы управления наследуются от класса **Control** и поэтому имеют ряд общих свойств:

* **Anchor**: Определяет, как элемент будет растягиваться
* **BackColor**: Определяет фоновый цвет элемента
* **BackgroundImage**: Определяет фоновое изображение элемента
* **ContextMenu**: Контекстное меню, которое открывается при нажатии на элемент правой кнопкой мыши. Задается с помощью элемента *ContextMenu*
* **Cursor**: Представляет, как будет отображаться курсор мыши при наведении на элемент
* **Dock**: Задает расположение элемента на форме
* **Enabled**: Определяет, будет ли доступен элемент для использования. Если это свойство имеет значение False, то элемент блокируется.
* **Font**: Устанавливает шрифт текста для элемента
* **ForeColor**: Определяет цвет шрифта
* **Location**: Определяет координаты верхнего левого угла элемента управления
* **Name**: Имя элемента управления
* **Size**: Определяет размер элемента
* **Width**: ширина элемента
* **Height**: высота элемента
* **TabIndex**: Определяет порядок обхода элемента по нажатию на клавишу Tab
* **Tag**: Позволяет сохранять значение, ассоциированное с этим элементом управления

***Кнопка***

Наиболее часто используемым элементом управления является кнопка. Обрабатывая событие нажатия кнопки, мы может производить те или иные действия.

При нажатии на кнопку на форме в редакторе Visual Studio мы по умолчанию попадаем в код обработчика события *Click*, который будет выполняться при нажатии:

***Метки Label***

Для отображения простого текста на форме, доступного только для чтения, служит элемент *Label*. Чтобы задать отображаемый текст метки, надо установить свойство *Text* элемента.

***Текстовое поле TextBox***

Для ввода и редактирования текста предназначены текстовые поля - элемент *TextBox*. Так же как и у элемента *Label* текст элемента *TextBox* можно установить или получить с помощью свойства *Text*.

По умолчанию при переносе элемента с панели инструментов создается однострочное текстовое поле. Для отображения больших объемов информации в текстовом поле нужно использовать его свойства *Multiline* и *ScrollBars*. При установке для свойства *Multiline* значения *true*, все избыточные символы, которые выходят за границы поля, будут переноситься на новую строку.

***Автозаполнение текстового поля***

Элемент *TextBox* обладает достаточными возможностями для создания автозаполняемого поля. Для этого нам надо привязать свойство **AutoCompleteCustomSource** элемента *TextBox* к некоторой коллекции, из которой берутся данные для заполнения поля.

***Перенос по словам***

Чтобы текст в элементе TextBox переносился по словам, надо установить свойство WordWrap равным true. То есть если одно слово не умещается на строке, то но переносится на следующую. Данное свойство будет работать только для многострочных текстовых полей.

***Ввод пароля***

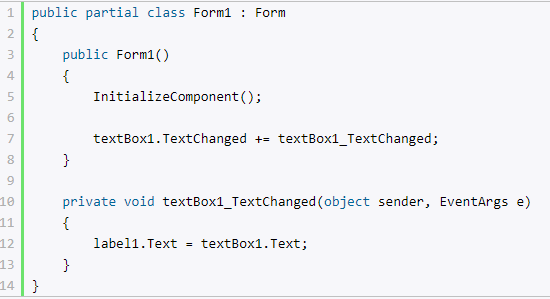
Также данный элемент имеет свойства, которые позволяют сделать из него поле для ввода пароля. Так, для этого надо использовать PasswordChar и UseSystemPasswordChar.

Свойство PasswordChar по умолчанию не имеет значение, если мы установим в качестве него какой-нибудь символ, то этот символ будут отображаться при вводе любых символов в текстовое поле.

Свойство UseSystemPasswordChar имеет похожее действие. Если мы установим его значение в true, то вместо введенных символов в текстовом поле будет отображаться знак пароля, принятый в системе, например, точка.

***Событие TextChanged***

Из всех событий элемента *TextBox* следует отметить событие *TextChanged*, которое срабатывает при изменении текста в элементе. Например, поместим на форму кроме текстового поля метку и сделаем так, чтобы при изменении текста в текстовом поле также менялся текст на метке:



**Практическая часть.**

*Задание*: Создайте оконное приложение для вычисления указанных в индивидуальных вариантах сумм. В заголовке окна укажите номер работы, номер варианта. При запуске приложения в окне должно появляться Ваше задание и кнопка «Вычислить». Предусмотрите поле ввода для переменной суммы в месте его нахождения. После начала вычислений все вхождения этой переменной должны быть заменены на её значения (при необходимости – вычислите выражения). После запуска вычислений в доступности должны остаться только кнопки «Очистить» и «Закрыть», поле ввода для переменной суммы должно быть заблокировано к использованию. При выполнении очистки окно должно вернуться к первоначальному состоянию.

**Контрольные вопросы**

1. Какой безусловный оператор цикла Вам известен?
2. Какой модуль включает в себя математические функции?
3. Приведите примеры итерационных операторов, тело которых не выполнится ни разу.
4. Перечислите несколько основных свойств форм.
5. Перечислите, какие свойства и методы кнопок использованы в вашей лабораторной работе.