Урок №3

Содержание

1. Необходимость создания дополнительных форм.

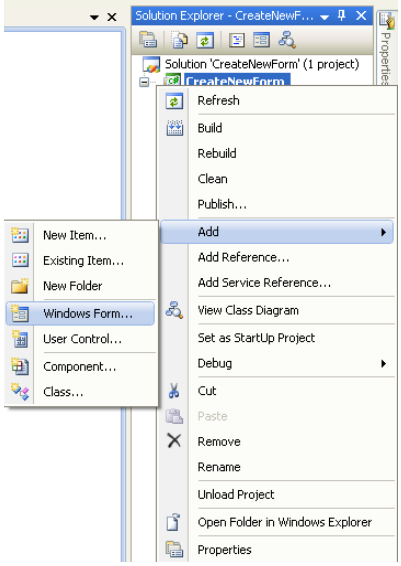
Чем объемнее программа, тем сложнее поместить всю функциональность на одной форме. В таких случаях прибегают к созданию дополнительных форм. Дополнительные формы очень удобны для отображения различных графиков и диаграмм, для редактирования или добавления информации, для заполнения анкет, для выдачи сообщений пользователю.

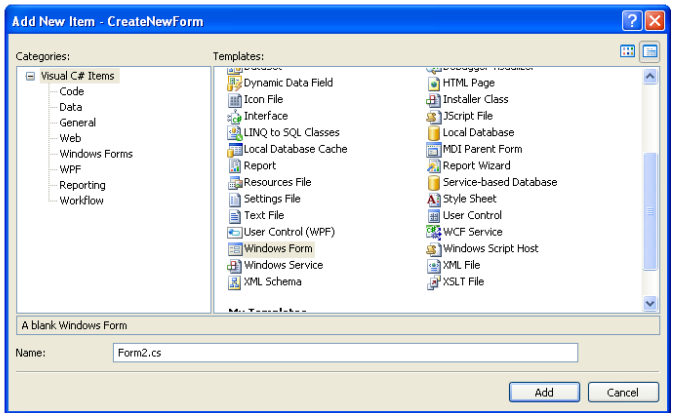
Формы бывают модальные и немодальные. Модальная форма перехватывает на себя все сообщения текущего приложения. Модальная форма полностью блокирует весь рабочий процесс до тех пор, пока не будет закрыта. Немодальная форма не блокирует рабочий процесс, позволяя пользователю переключаться между формами приложения.

2. Создание дополнительной формы.

Для того, чтобы вызвать дополнительную форму, необходимо сначала ее создать. **MS Visual Studio 2008** предоставляет нам несколько вариантов это сделать.

1) В окне **Solution Explorer** нажмите правой кнопкой мыши на названии проекта, затем выберите пункт **Add->Windows Form**.

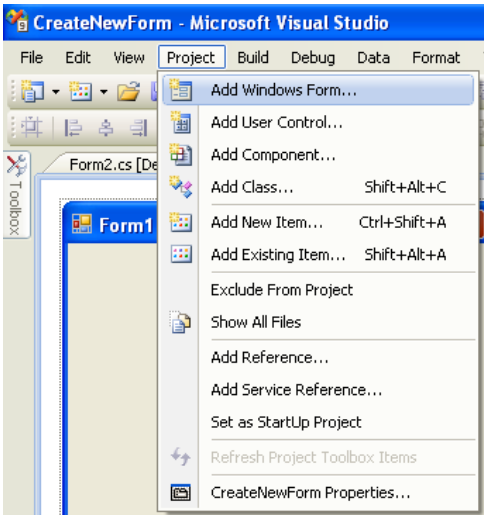


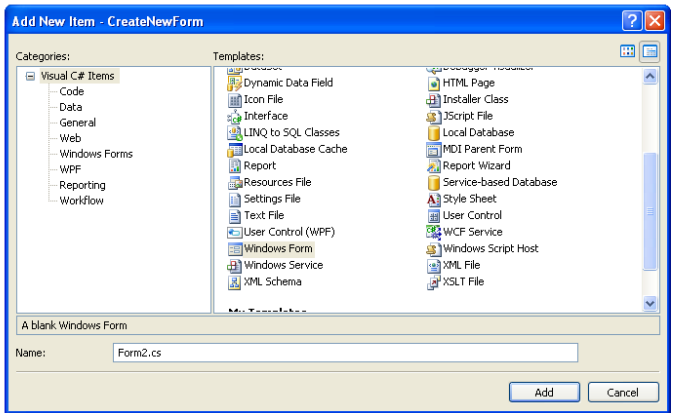


В появившемся окне задайте название новой форме

Форма готова, теперь добавьте на нее необходимые элементы управления.

2) Выберите пункт меню **Project->Add Windows Form…**





В появившемся окне задайте название новой форме

Форма готова, теперь добавьте на нее необходимые элементы управления.

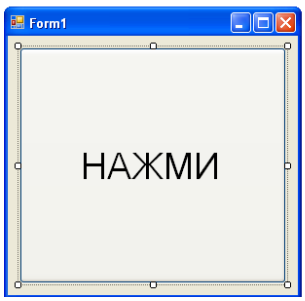
Для вызова новой формы необходимо создать экземпляр второй формы и вызвать для нее функцию **ShowDialog**.

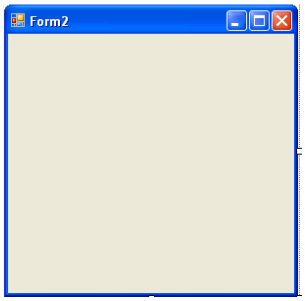
Например:

```
Form2 f = new Form2();
f.ShowDialog();
```

Функция **ShowDialog** возвращает перечислимый тип данных **DialogResult.** В следующей главе будет рассмотрен пример с анализом этого типа данных.

Необходимо понимать, что функция, внутри которой будет размещен данный текст, продолжит свою работу только после закрытия второй формы.
Давайте рассмотрим следующий пример:
Создадим две формы **Form1** и **Form2**. На форму **Form1** добавим кнопку. Форма **Form2** без элементов управления.

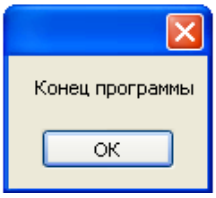




В обработчике на кнопку напишем следующий текст:

```
Form2 f = new Form2();
f.ShowDialog();
MessageBox.Show("Конец программы");
```

Запустим приложение и убедимся что сообщение



появляется только после закрытия второй формы.

3. Обмен данными между формами.

Итак, мы уже знаем, как создавать новые формы. Как правило, нет смысла создавать формы, которые никак между собой не связаны. В связи с этим возникает необходимость осуществить обмен данными между двумя формами.

Форма, которая осуществляет вызов дополнительной формы, называется родительской, форма, которую вызвали, называется дочерней.

Обмен данными между дочерней и родительской формами может происходить в двух направлениях: передача данных от родительской к дочерней, передача данных от дочерней к родительской.

Передача данных от родительской формы к дочерней форме.

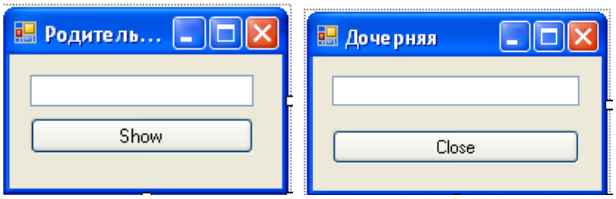
Передача данных от родительской формы к дочерней форме проще осуществима, т.к. родительская форма содержит ссылку на дочернюю форму.

Передать данные дочерней форме можно следующими способами:

- Через конструктор дочерней формы.
- Через дополнительную функцию или свойство дочерней формы.
- Через перегрузку функции **ShowDialog**

Передача данных через конструктор дочерней формы.

Создадим две формы **Parent** и **Child**. На первой форме разместим текстовое поле и кнопку **Show**. На второй форме разместим текстовое поле и кнопку **Close**.



Переопределим конструктор формы **Child**:

```
public Child(string s)
        {
            InitializeComponent();
            textBox1.Text = s;

        }
```

Добавим обработчик на копку **Close**:

```
private void Close_Click(object sender, EventArgs e)
        {
            this.Close();

        }
```

Добавим обработчик на кнопку **Show**:

```
private void Show_Click(object sender, EventArgs e)
        {
            Child form1 = new Child(textBox1.Text);
            form1.ShowDialog();
```

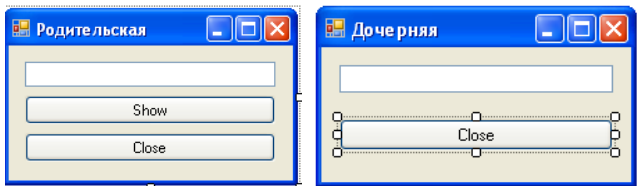
```
        }
```

Запускаем приложение:

Недостаток данного метода в том, что передать данные дочерней форме можно только один раз в момент создания формы.

<u>Передача данных через дополнительную функцию или свойство дочерней формы.</u>

Создадим две формы **Parent** и **Child**. На первой форме разместим текстовое поле и кнопки **Show** и **Close**. На второй форме разместим текстовое поле и кнопку **Close**.



На дочерней форме создадим свойство и обработчик на кнопку **Close**:

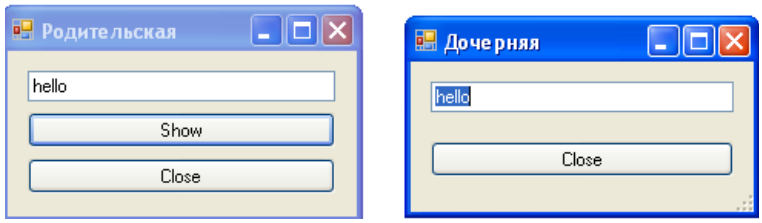
```
public string TText
        {
            set
            {
                textBox1.Text=value;
            }
        }

private void Close_Click(object sender, EventArgs e)
        {
            this.Close();
        }
```

На родительской форме создадим обработчики на кнопки:

```
private void Close_Click(object sender, EventArgs e)
        {
            this.Close();
        }

private void Show Click(object sender, EventArgs e)
        {
            Child form1 = new Child();
            form1.TText = textBox1.Text;
            form1.ShowDialog();
```

```
        }
```

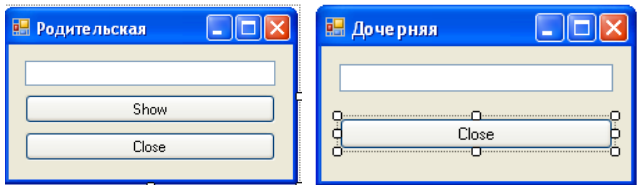
Запускаем приложение:



Недостаток данного метода в том, что можно создать и вызвать форму не присваивая свойству никакого значения, что может повлечь за собой ошибки во время выполнения.

Передача данных через перегрузку функции ShowDialog.

Создадим две формы **Parent** и **Child**. На первой форме разместим текстовое поле и кнопки **Show** и **Close**. На второй форме разместим текстовое поле и кнопку **Close**.



На дочерней форме напишем обработчик для кнопки **Close** и перегрузку функции **ShowDialog**:

```
private void Close_Click(object sender, EventArgs e)
        {
            this.Close();
        }
public DialogResult ShowDialog(string s)
        {
            textBox1.Text = s;
            return ShowDialog();
        }
```

На родительской форме создадим обработчики на кнопки:

```
private void Close_Click(object sender, EventArgs e)
        {
            this.Close();
        }

private void Show_Click(object sender, EventArgs e)
        {
            Child form1 = new Child();
            form1.ShowDialog(textBox1.Text);
        }
```

Недостаток данного метода в том, что можно вызвать перегрузку функции **ShowDialog** без параметров, что может повлечь за собой ошибки во время выполнения.

Передача данных от дочерней формы к родительской форме.

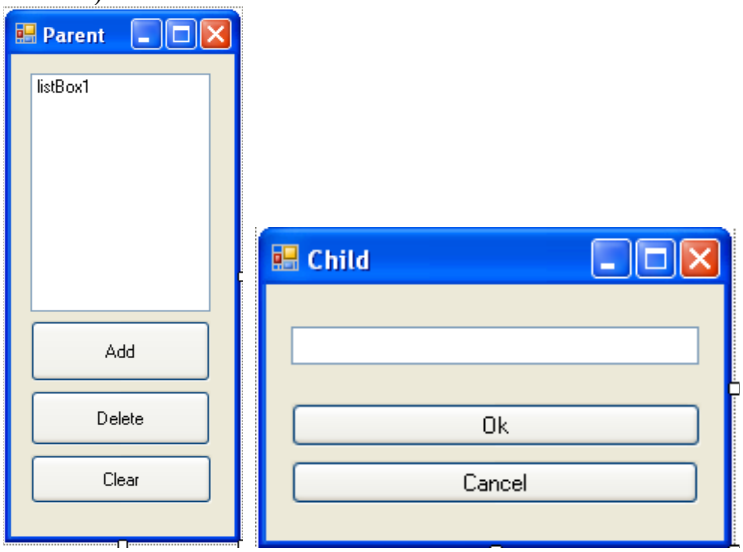
Передача данных от дочерней формы к родительской форме немного сложнее, так как дочерняя форма не содержит ссылки на родительскую форму.

Передать данные от дочерней формы к родительской форме можно следующими способами:

- Создать дополнительную функцию или свойство на дочерней форме.
- Передать ссылку на форму или другой объект в конструктор дочерней формы или в перегруженную функцию **ShowDialog**.

Также родительской форме зачастую нужно знать подтвердил пользователь свой выбор или отказался от него. Например, в дополнительной форме осуществляется добавление нового товара. Пользователь открыл форму, но передумал добавлять новый товар.
Именно для этих целей функция **ShowDialog** возвращает перечислимый тип данных **DialogResult**.

Создадим две формы **Parent** и **Child**. На первой форме разместим список и три кнопки (**Add**, **Delete**, **Clear**). На второй форме разместим текстовое поле и две кнопки (**Ok**, **Cancel**).



На дочерней форме создадим свойство и обработчики на кнопки:

```
private void Ok_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.OK;
        }

public string TText
        {
            get
            {
                return textBox1.Text;
```

```
            }
        }

private void Cancel_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }
```

На родительской форме создадим обработчики на кнопки:

```
private void Delete_Click(object sender, EventArgs e)
        {
            listBox1.Items.RemoveAt(listBox1.SelectedIndex);
        }

private void Clear_Click(object sender, EventArgs e)
        {
            listBox1.Items.Clear();
        }

private void Add_Click(object sender, EventArgs e)
        {
            Child form1 = new Child();
            if (form1.ShowDialog() == DialogResult.OK)
            {
                listBox1.Items.Add(form1.TText);
            }

        }
```

4. Создание немодальной формы.

Для вызова немодальной формы необходимо создать экземпляр второй формы и вызвать для нее функцию **Show**.

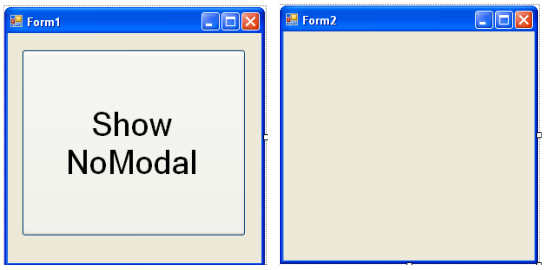
Например:

```
Form2 f = new Form2();
f.Show();
```

Нужно понимать, что функция, внутри которой будет размещен данный текст, не будет ждать закрытия второй формы и продолжит свою работу. Это может привести к тому что ссылка на вторую форму будет уничтожена. В связи с этим, при вызове немодальной формы не рекомендуется создавать локальные ссылки.

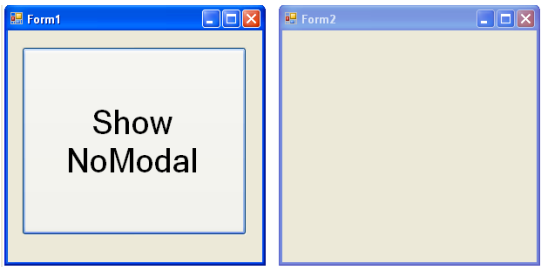
Рассмотрим следующий пример:

Создадим две формы **Form1** и **Form2**. На форму **Form1** добавим кнопку. Форма **Form2** без элементов управления.



Добавим переменную в класс **Form1** – ссылку на вторую форму и обработчик на кнопку.

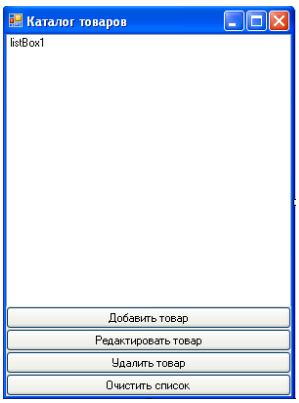
```
public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        Form2 form=null;
        private void button1_Click(object sender, EventArgs e)
        {
            /*если одна форма уже открыта, сначала закроем ее,
            прежде чем открывать следующую*/
            if (form != null) form.Close();
            form = new Form2();
            form.Show();

        }

     }
```

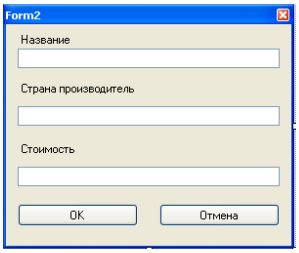


Когда вы закрываете форму, она разрушается и освобождается память, поэтому перед каждым вызовом формы необходимо выделять для нее память.

5. Пример приложения с дополнительными диалогами

Первый пример, который мы рассмотрим, представляет собой каталог товаров. Для этого создан класс товар, содержащий название, страну производителя и цену. Хранить товары будем прямо в **ListBox**, поэтому для корректного отображения перегрузим у товара функцию **ToString**. Для редактирования выбранного товара и добавления нового товара будем использовать одну и ту же дополнительную форму. Пример называется **GoodsCatalog**.





Класс товар:

```
public class Tovar
    {
         string name; //название товара
         string made_in; //страна производитель
         double price;//цена

         public string Name
         {
              get {  return name; }
              set {  name = value; }
         }

         public string Made in
         {
              get {  return made_in; }
              set {  made_in = value; }
         }

         public double Price
         {
              get {  return price; }
              set {  if(price<0)
             throw new System.Exception("Цена не может быть меньше нуля");
                   price = value; }
         }
         public Tovar()
         {
              Name = "unknown";
              Price = 0;
              Made in = "unknown";
         }
         public Tovar(string name, string made, double price)
         {
              Name = name; Made_in = made; Price = price;
         }
         public override string ToString()
         {
              return Name+" Изготовитель: "+Made in+" Цена: "+Price;

         }
```

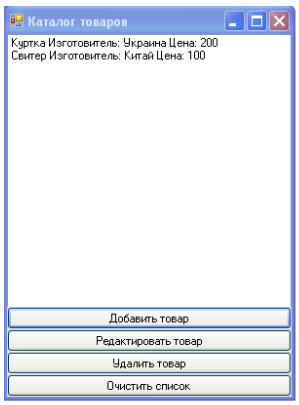
Дополнительная форма:

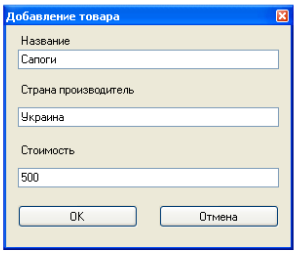
```
public partial class Form2 : Form
    {
        Tovar t; bool addnew;
        public Form2(Tovar t, bool addnew)
        {
            InitializeComponent();
            this.addnew = addnew;
            this.t = t;//Запомнили ссылку на товар
            if (addnew == false)
            { /*если форма открывается для редактирования
                то сначала занесем информацию о изменяемом
              товаре в текстовые поля*/
                textBox1.Text = t.Name;
                textBox2.Text = t.Made_in;
                textBox3.Text = t.Price.ToString();
                this.Text = "Редактирование товара";//меняем заголовок
            }
            //меняем заголовок если создание товара
            else this.Text = "Добавление товара";
         }

        private void OK Click(object sender, EventArgs e)
        {
            if (textBox1.Text == "" || textBox2.Text == "" ||
                textBox3.Text == "")
            { //Проверка заполнения полей
                MessageBox.Show("Заполните все поля"); return;
            }
            if (t == null) t = new Tovar();
            t.Name = textBox1.Text;
            t.Made_in = textBox2.Text;
            try
            { /* При преобразовании из строки в вещественное число
              произойдет ошибка, если строка неверного формата*/
                t.Price = Convert.ToDouble(textBox3.Text);
            }
            catch
            {
                MessageBox.Show("Цена указана неверно");
                return;
            }
            this.DialogResult = DialogResult.OK;

        }

        private void Cancel Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }

    }
```

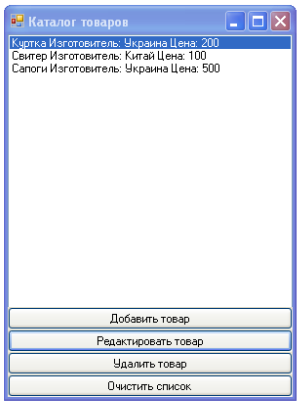
Главная форма:

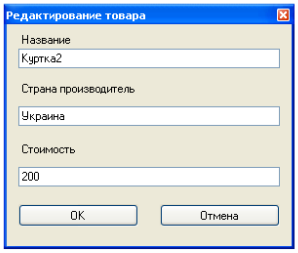
```
public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void ClearAll_Click(object sender, EventArgs e)
        {//Очистить весь список
            listBox1.Items.Clear();
        }

        private void DeleteGood Click(object sender, EventArgs e)
        {//Удаление выделенного элемента
            if (listBox1.SelectedIndex == -1)//Если товар не выбран
            {
                MessageBox.Show("Вы не выбрали товар"); return;
            }
            listBox1.Items.RemoveAt(listBox1.SelectedIndex);

        }
        Tovar t=null;
        private void AddGood Click(object sender, EventArgs e)
        {//Добавление товара
            t = new Tovar();
            Form2 addform = new Form2(t,true);
            if (addform.ShowDialog() == DialogResult.OK)
            {//если пользователь нажал ок, добавляем товар в список
                listBox1.Items.Add(t);
            }
        }

        private void EditGood_Click(object sender, EventArgs e)
        {//редактирование товара
            if (listBox1.SelectedIndex == -1)//Если товар не выбран
            {
                MessageBox.Show("Вы не выбрали товар"); return;
            }

            int n = listBox1.SelectedIndex;//запоминаем выделенный элемент
            //Забираем ссылку на выделенный элемент
            t = (Tovar)listBox1.Items[ n] ;
            Form2 editform = new Form2(t, false);
            editform.ShowDialog();
            listBox1.Items.RemoveAt(n);//Удаляем выделенный элемент
            /*и добавляем его снова в ту же позицию, чтобы он перерисовался в
                списке*/
            listBox1.Items.Insert(n, t);
            listBox1.SelectedIndex = n; //Снова выделяем этот элемент
        }

    }
```

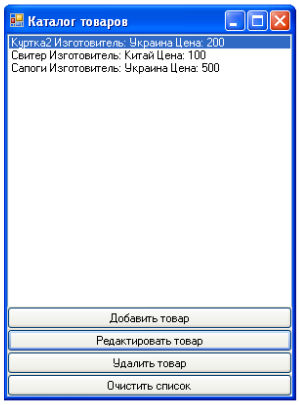
Наше приложение готово.











6. Что такое общий диалог? Типы общих диалогов.

Операционная система **Windows** предоставляет ряд основных диалоговых окон, позволяющих обеспечить единообразие пользовательского интерфейса в приложениях **Windows** при выполнении таких операций как открытие и сохранение файлов, задание цвета шрифта или текста и печать.

Общим диалогом называется дополнительное окно, которое выглядит одинаково у всех приложений.

Существуют следующие типы общих диалогов:

Тип общего диалога	**Описание**
ColorDialog	Представляет общее диалоговое окно, в котором отображаются доступные цвета и элементы управления, позволяющие

	пользователю определять собственные цвета. Для создания этого общего диалогового окна необходимо вызвать метод **ShowDialog**. Для извлечения цвета, выбранного пользователем, используется свойство **Color**.
FolderBrowserDialog	Этот класс предоставляет пользователю возможность, выбрать, просмотреть, создать папку. Используйте этот класс, когда вам необходимо, чтобы пользователь выбрал папку, а не файл. Обзор папок осуществляется с помощью дерева. Могут выбираться только папки из файловой системы; выбор виртуальных папок невозможен. Для создания этого общего диалогового окна необходимо вызвать метод **ShowDialog**. Для извлечения пути к папке используйте свойство **SelectedPath**. Установите свойство **ShowNewFolderButton** в true, чтобы разрешить пользователю создавать новые папки.
FontDialog	Предлагает пользователю выбрать шрифт среди шрифтов, установленных на локальном компьютере. Для создания этого общего диалогового окна необходимо вызвать метод **ShowDialog**. Для получения информации о выбранном шрифте, используйте свойство **Font**. Объект **Font** содержит сведения о размере, но не содержит сведений о цвете.
OpenFileDialog	Запрашивает пользователя об открытии файла. Этот класс позволяет проверить, существует ли файл, и открыть его. Свойство **ShowReadOnly** определяет, отображается ли в диалоговом окне флажок "доступно только для чтения". Свойство **ReadOnlyChecked** показывает, установлен ли флажок "доступно только для чтения". Для получения имени выбранного файла используйте свойство **FileName**. Для открытия выбранного файла можете использовать функцию

	OpenFile.
PageSetupDialog	Позволяет пользователям изменять параметры печати для страницы, включая поля и ориентацию бумаги. Поскольку для отображения **PageSetupDialog** необходимы параметры страницы, нужно задать свойство **Document**, **PrinterSettings** или **PageSettings** перед вызовом метода **ShowDialog.** В противном случае произойдет исключение.
PrintDialog	Позволяет пользователям выбирать принтер и определять, какие разделы документа должны быть напечатаны из приложения **Windows Forms.** Для получения параметров принтера, измененных пользователем с помощью **PrintDialog,** используйте свойство **PrinterSettings.**
PrintPreviewDialog	Представляет форму диалогового окна, которая содержит объект **PrintPreviewControl** для печати из приложения **Windows Forms.**
SaveFileDialog	Запрашивает у пользователя местоположение для сохранения файла. Для получения имени выбранного файла используйте свойство **FileName.** Для открытия выбранного файла можете использовать функцию **OpenFile.**

7. Классы OpenFileDialog, SaveFileDialog.

Как видим, имеется достаточно много соответствующие стандартные диалоги. А теперь давайте познакомимся с некоторыми из них поближе.
Основные методы класса **OpenFileDialog:**

Метод	**Описание**
OpenFile	Открывает выбранный пользователем файл в режиме "только чтение". Файл задается свойством **FileName.**
Reset	Восстанавливает значения всех свойств по умолчанию.
ShowDialog	Показывает диалог модально.

Основные свойства класса **OpenFileDialog:**

Метод	Описание
CheckFileExists	Получает или задает значение, указывающее, отображается ли в диалоговом окне предупреждение, если пользователь указывает несуществующее имя файла.
CheckPathExists	Возвращает или задает значение, указывающее, отображает ли диалоговое окно предупреждение, если пользователь указывает несуществующий путь.
DefaultExt	Возвращает или задает расширение имени файла по умолчанию.
FileName	Возвращает или задает строку, содержащую имя файла, выбранное в диалоговом окне файла.
Filter	Возвращает или задает текущую строку фильтра имен файлов, которая определяет варианты, доступные в поле "Сохранить как тип файла" или "Файлы типа" диалогового окна. Фильтр задается парами через символ «|». Первым в паре идет текст, который будет показан пользователю, а вторым – фильтр. В конце строки необходимо указать два символа «|». Например: txt files (*.txt)|*.txt|All files (*.*)|*.*
FilterIndex	Возвращает или задает индекс фильтра, выбранного в настоящий момент в диалоговом окне файла.
InitialDirectory	Возвращает или задает начальную папку, отображенную диалоговым окном файла.
MultiSelect	Получает или задает значение, указывающее, можно ли в диалоговом окне выбирать несколько файлов.
ReadOnlyChecked	Получает или задает значение, указывающее, установлен ли флажок доступности только для чтения.
ShowReadOnly	Получает или задает значение, указывающее, имеется ли в диалоговом окне флажок "доступно только для чтения".

Основные методы класса **SaveFileDialog:**

Метод	Описание

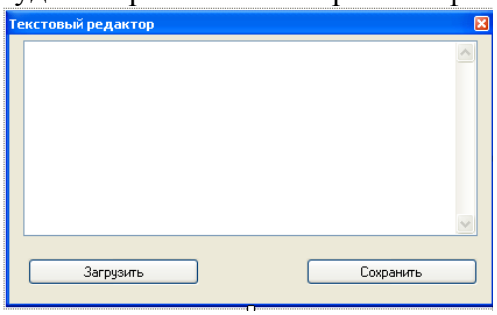
OpenFile	Открывает выбранный пользователем файл с разрешением на чтение и запись.
Reset	Восстанавливает значения всех свойств по умолчанию.
ShowDialog	Показывает диалог модально.

Основные свойства класса **SaveFileDialog:**

Метод	**Описание**
AddExtention	Возвращает или задает значение, определяющее, добавляет ли автоматически диалоговое окно расширение к имени файла, если пользователь опускает данное расширение.
CheckPathExists	Возвращает или задает значение, указывающее, отображает ли диалоговое окно предупреждение, если пользователь указывает несуществующий путь.
DefaultExt	Возвращает или задает расширение имени файла по умолчанию.
FileName	Возвращает или задает строку, содержащую имя файла, выбранное в диалоговом окне файла.
Filter	Возвращает или задает текущую строку фильтра имен файлов, которая определяет варианты, доступные в поле "Сохранить как тип файла" или "Файлы типа" диалогового окна. Фильтр задается парами через символ «|». Первым в паре идет текст, который будет показан пользователю, а вторым – фильтр. В конце строки необходимо указать два символа «|». Например: txt files (*.txt)|*.txt|All files (*.*)|*.*
FilterIndex	Возвращает или задает индекс фильтра, выбранного в настоящий момент в диалоговом окне файла.
InitialDirectory	Возвращает или задает начальную папку, отображенную диалоговым окном файла.
OverwritePrompt	Возвращает или задает значение, показывающее, будет ли диалоговое окно **Save As** выводить предупреждение,

	если файл с указанным именем уже существует.

Мы познакомились с основными методами и свойствами классов **OpenFileDialog** и **SaveFileDialog**, теперь рассмотрим пример.
Разработаем приложение, которое представляет из себя текстовый редактор. Для этого создадим форму, разместим на ней текстовое поле и две кнопки, установим у текстового поля свойства **MultiLine** и **true** и свойство **ScrollBars** в **Vertical.** При нажатии на кнопку «Загрузить» будет вызываться **OpenFileDialog**, предлагая выбрать текстовый файл, затем этот файл будет прогружаться в текстовое поле, в котором можно отредактировать содержимое файла. При нажатии на кнопку «Сохранить», будет вызываться **SaveFileDialog** и содержимое текстового поля будет сохраняться в выбранный файл.



Подключим пространство имен **System.IO** и напишем обработчики на кнопки.

```
private void Load Click(object sender, EventArgs e)
        {
            OpenFileDialog open = new OpenFileDialog(); //создали экземпляр
            //установим фильтр для файлов
            open.Filter = "All Files(*.*)|*.*|Text Files(*.txt)|*.txt||";
            open.FilterIndex = 1;//по умолчанию фильтруются текстовые файлы
            if (open.ShowDialog() == DialogResult.OK)
            {
               StreamReader reader = File.OpenText(open.FileName);
               textBox1.Text = reader.ReadToEnd(); //считываем файл до конца
               reader.Close(); //закрываем reader
            }
        }

 private void Save_Click(object sender, EventArgs e)
        {
            SaveFileDialog save = new SaveFileDialog();//создали экземпляр
            if (save.ShowDialog() == DialogResult.OK)
            {
                StreamWriter writer = new StreamWriter(save.FileName);
                //записываем в файл содержимое поля
                writer.Write(textBox1.Text);
                writer.Close();//закрываем writer
            }
        }
```

8. Класс FolderBrowserDialog

Основные методы класса FolderBrowserDialog:

Метод	Описание

Reset	Восстанавливает значения всех свойств по умолчанию.
ShowDialog	Показывает диалог модально.

Основные свойства класса **FolderBrowserDialog:**

Метод	**Описание**
RootFolder	Получает или задает корневую папку, с которой начинается просмотр.
SelectedPath	Получает или задает путь, выбранный пользователем.
ShowNewFolderButton	Получает или задает значение, указывающее, отображается ли кнопка **New Folder** в диалоговом окне просмотра папок.

9. Домашнее задание

1. Разработайте программу, которая позволяет пользователю осуществлять поиск файлов по заданному критерию. Приложение состоит из двух форм – главное окно и окно поиска. Пользователь может открыть сколько угодно окон поиска. Окно поиска запускается немодально. В окне поиска пользователь выбирает папку, в которой будет идти поиск и вводит маску поиска, например «*.doc» . Найденные файлы отображаются в списке.

2. Фирма продает составляющие компьютера. Первая форма отвечает за учет продаж, вторая за добавление и редактирование составляющих.

   Первая форма:

   Список, выпадающий список, текстовое поле, кнопка вызова второй формы. В выпадающем списке появляются наименования всех товаров, которые в наличии в магазине. Пользователь выбирает товар, в текстовом окне, которое нельзя редактировать, появляется цена. Пользователь нажимает «добавить» и товар добавляется в список продаж. Также должно быть окошко, которое выводит общую стоимость.

   Вторая форма:

   Информация о комплектующих (наименование, характеристика, описание и цена) вводится в текстовые поля; в список добавляется текстовая строка, содержащая информацию о товаре, кроме цены, цена в списке не видна, но содержится; также комплектующие можно редактировать.

3. Разработайте приложение, которое состоит из двух форм. Первая форма содержит TextBox доступный только для чтения и две кнопки «загрузить

файл» и «редактировать». Кнопка «редактировать» изначально неактивна. При нажатии на первую кнопку, открывается диалог и пользователю предлагают выбрать текстовый файл. Выбранный файл загружается в TextBox и кнопка «редактировать» становится активной. При нажатии на вторую кнопку открывается вторая форма (не модально), которая содержит TextBox доступный для редактирования и две кнопки «Сохранить» и «Отменить». При нажатии на первую кнопку изменения отображаются в TextBox первой формы.