



**МИНОБРНАУКИ РОССИИ**  
**федеральное государственное бюджетное образовательное учреждение**  
**высшего профессионального образования**  
**«Московский государственный технологический университет «СТАНКИН»**  
**(ФГБОУ ВПО МГТУ «СТАНКИН»)**

---

Факультет «Факультет информационных технологий и систем управления»  
Кафедра «Компьютерные системы управления»

Козак Николай Владимирович

**Графические системы и интерфейс оператора**

Лабораторный практикум  
по дисциплине «Информатика» для студентов МГТУ «СТАНКИН», обучающихся по  
направлению 22700.62 «Автоматизация технологических процессов и производств»

Москва 2014 г.

**Содержание:**

<b>Основная часть .....</b>	<b>- 4 -</b>
Создание заготовки приложения .....	- 4 -
Установка пароля на главное окно приложения .....	- 7 -
Добавление списка для запуска упражнений .....	- 12 -
<b>Вариант 1. Смайлики .....</b>	<b>- 13 -</b>
<b>Вариант 2. Простой калькулятор .....</b>	<b>- 19 -</b>
<b>Индивидуальные задания.....</b>	<b>- 28 -</b>
ВАРИАНТ 1 .....	- 28 -
ВАРИАНТ 2 .....	- 29 -
<b>Приложение 1 Некоторые компоненты панели Toolbox.....</b>	<b>30</b>
<b>Приложение 2. Рекомендуемые префиксы для некоторых элементов управления ...</b>	<b>- 35 -</b>
<b>Основная часть .....</b>	<b>- 37 -</b>
Построение элемента управления BEEPBUTTON .....	- 37 -
Создание заготовки библиотеки компонентов .....	- 37 -
Переопределение наследуемого от CONTROL виртуального метода диспетчеризации события CLICK .....	- 43 -
<b>Вариант 1. Проектирование диалогового окна сообщений MyDialogBox .....</b>	<b>- 45 -</b>
Тестирование пользовательских элементов BEEPBUTTON и MYDIALOGBOX .....	- 49 -
<b>Вариант 2. Рисование круглой кнопки .....</b>	<b>- 52 -</b>
Создание заготовки компонента CIRCLEBUTTON.....	- 52 -
Задание круглой формы для кнопки .....	- 53 -
Тестирование компонента круглой кнопки.....	- 56 -
<b>Индивидуальные задания.....</b>	<b>- 58 -</b>
ВАРИАНТ 1: .....	- 58 -
ВАРИАНТ 2: .....	- 58 -
<b>Основная часть .....</b>	<b>- 61 -</b>
Разработка стартовой формы .....	- 61 -
Добавление объектов всплывающей подсказки .....	- 67 -
Разработка кнопки CLICKMATICBUTTON .....	- 69 -
Наполнение класса CLICKMATICBUTTON функциональностью генерации щелчков мыши ....	- 74 -
Тестирование генерирующей кнопки CLICKMATICBUTTON .....	- 76 -
<b>Разработка кнопки в стиле полос прокрутки .....</b>	<b>- 78 -</b>
Тестирование скроллирующей кнопки ARROWBUTTON .....	- 80 -
<b>Комбинирование элементов в единый компонент NumericScan .....</b>	<b>- 82 -</b>
Тестирование компонента NUMERICSCAN .....	- 88 -
<b>Индивидуальные задания.....</b>	<b>- 90 -</b>
<b>Приложение 1 Таблица цветов структуры Color.....</b>	<b>91</b>

## **Лабораторная Работа №1**

### **Пользовательский интерфейс Windows Forms для C#**

Краткое описание: в данной лабораторной работе рассматриваются некоторые возможности интегрированной среды разработки программного обеспечения (IDE - Integrated Development Environment) Visual Studio.NET 2005. Установка пароля на главное окно приложения. Добавление списка для запуска упражнений. Смайлики. Краткое описание части компонентов панели Toolbox. Простой калькулятор.

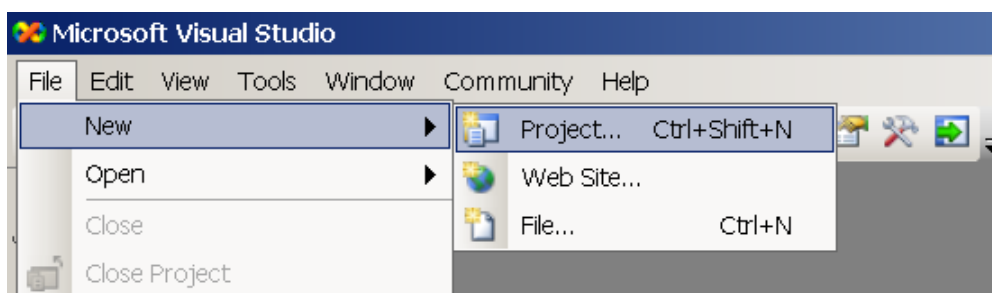
Файлы к лабораторной работе находятся в соответствующей папке с описанием.

## Основная часть

В данной лабораторной работе мы рассмотрим некоторые возможности интегрированной среды разработки программного обеспечения (IDE - Integrated Development Environment) Visual Studio.NET 2005. Мы будем использовать разработанный корпорацией Microsoft самый красивый и современный язык программирования C# (Си-шарп). Мы будем опираться на мощную библиотеку классов .NET Framework (FCL - Framework Class Library, она же BCL - Basic Class Library).

### Создание заготовки приложения

Выполните команду File/New/Project оболочки для создания нового проекта (Рис. 1)



**Рис. 1**

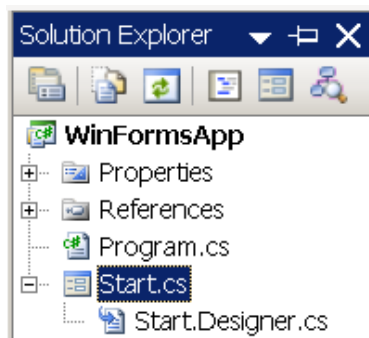
В диалоговом окне New Project выберите кнопкой Browse папку для размещения проекта, задайте тип, шаблон и имя проекта WinFormsApp, как показано на Рис. 2



**Рис. 2**

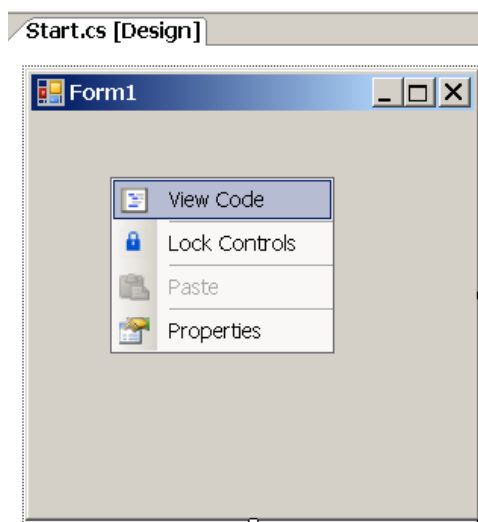
После щелчка на кнопке ОК мастер создаст заготовку приложения и выведет в режиме дизайна пустую форму, которая при запуске приложения будет представлять окно. Стандартный размер сгенерированной формы равен 300x300. Сделаем эту форму стартовой и разместим на ней элементы управления для вызова дочерних форм, реализующих отдельные упражнения данной работы.

Откройте панель Solution Explorer (View -> Solution Explorer или комбинация клавиш Ctrl+Alt+L) оболочки и переименуйте файл Form1.cs в файл Start.cs (Рис. 3).



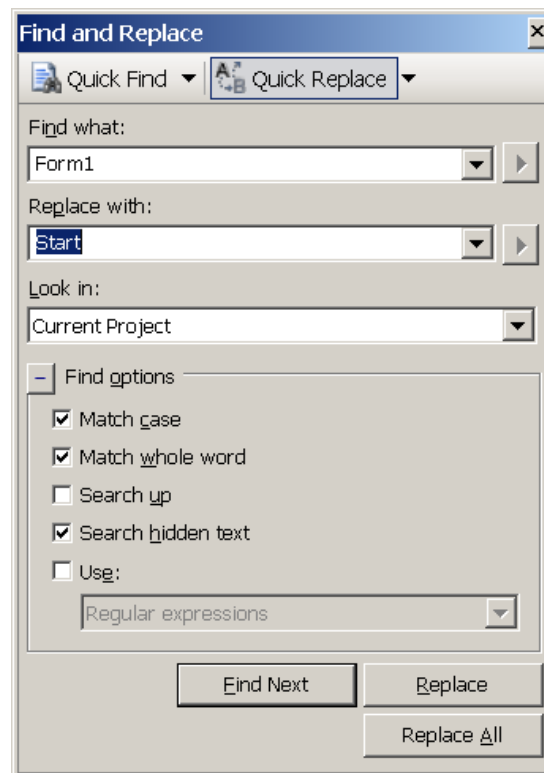
**Рис. 3**

Щелкните на форме правой кнопкой мыши и выполните команду контекстного меню View Code, чтобы вызвать представление файла Start.cs в режиме редактора кода (Рис. 4).

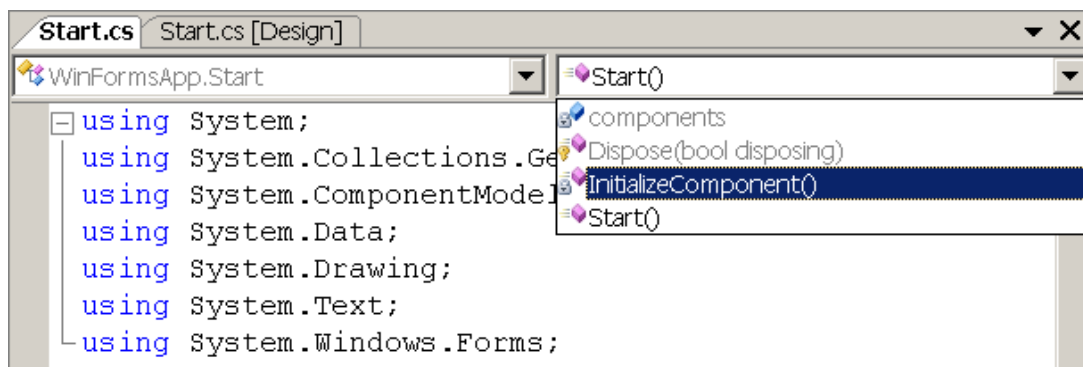


**Рис. 4**

В режиме кода вызовите комбинацией клавиш Ctrl-F окно Find and Replace для замены всех вхождений Form1 на Start. После заполнения полей поиска и замены щелкните на кнопке Replace All (Рис. 5).

**Рис. 5**

В раскрывающемся списке Members выделите функцию InitializeComponent(), чтобы быстро позиционироваться в соответствующее место кода (Рис. 6).

**Рис. 6**

Отредактируйте в функции InitializeComponent() код установки заголовка окна в режиме выполнения:

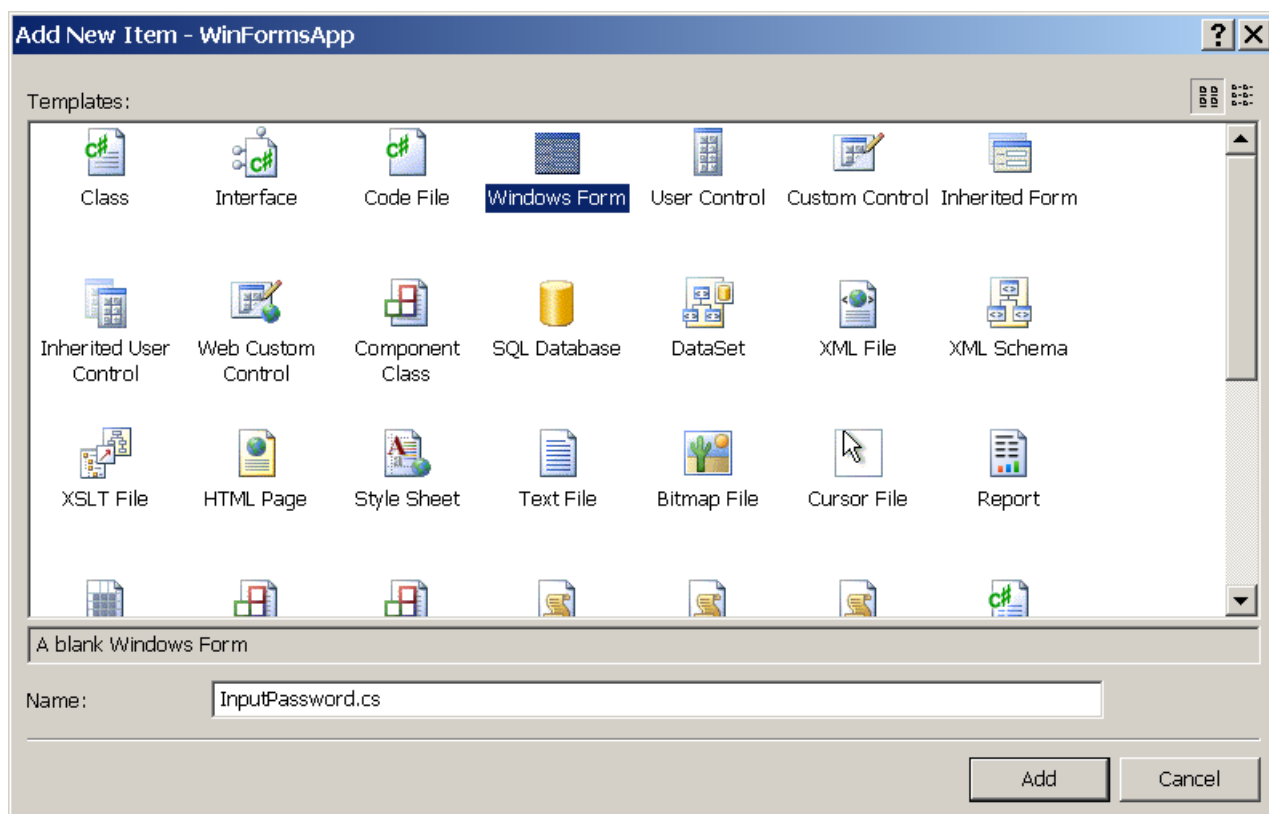
```
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.Text = "Lab1 - «Фамилия Имя, Группа»";
}
```

## Установка пароля на главное окно приложения

Добавим к главной форме Start код поддержки пароля при запуске приложения. Для этого нам потребуется диалоговое окно с текстовым полем. Сконструируем его. Создание диалогового окна будет состоять из трех шагов:

- Создание самой формы диалогового окна (класс InputPassword);
- Размещение на форме интерфейсных элементов пользователя;
- Создание кода, управляющего диалоговым окном;

В панели Solution Explorer выделите корневой узел проекта и командой Add/Windows Form добавьте в проект файл формы с именем InputPassword.cs (Рис. 7).



**Рис. 7**

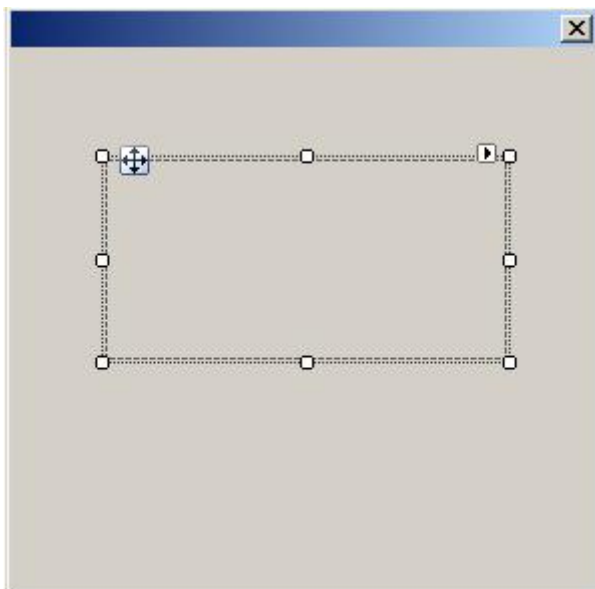
- Удалите из узла References лишние добавленные мастером ссылки на библиотечные сборки System.Data и System.Xml;
- Откройте файл InputPassword.cs в режиме View Code и удалите инструкции using подключения пространств имен System.Data и System.Xml;
- Настройте форму InputPassword в соответствии с Таблица 1;

**Таблица 1 Свойства формы InputPassword**

Свойство	Значение	Пояснения
StartPosition	CenterScreen	Открывать в центре экрана
FormBorderStyle	FixedDialog	Запретить пользователю менять размер
MaximizeBox	false	Отключить системную кнопку развертывания
MinimizeBox	false	Отключить системную кнопку свертывания

Свойство	Значение	Пояснения
ShowInTaskbar	false	Запретить появление в панели задач
AutoSize	true	Подстраиваться под содержимое
AutoSizeMode	GrowAndShrink	Подстраиваться под содержимое (расширяться и сжиматься)
Text	Пусто	Заголовок окна

Из свитка Containers панели инструментов Toolbox поместите на форму компоновочную панель FlowLayoutPanel () и настройте ее в соответствии с таблицей.



**Рис. 8**

**Таблица 2 Свойства компоновочной панели FlowLayoutPanel**

Свойство	Значение	Пояснения
Name	flow	Имя компоновочной панели
AutoSize	true	Подстраиваться под содержимое
FlowDirection	TopDown	Размещать сверху вниз
WrapContents	false	Не переносить дочерние элементы в следующий столбец
Padding.All	8	Внутренний отступ дочерних элементов по периметру окна
Dock	Fill	Распахнуть на всю клиентскую область родителя (формы)

Эта панель будет автоматически следить за размещением своих дочерних элементов в соответствии с настройками.

Выделите на форме компоновочную панель flow, чтобы именно она принимала дочерние элементы, поместите в нее из свитка Common Controls текстовую метку Label () и настройте ее свойства в соответствии с Таблица 3.



**Таблица 3 Свойства текстовой метки Label**

Свойство	Значение	Пояснения
Name	lblMessage	Имя текстовой метки
AutoSize	true	Подстраиваться под содержимое
Anchor	None	Расположить посередине
Margin.All	8	Отступ по наружному периметру текстовой метки

Аналогичным образом поместите в панель flow последовательно экземпляры компонентов TextBox и Button, которые настройте в соответствии с Таблица 4 и Таблица 5.

**Таблица 4 Свойства поля ввода TextBox**

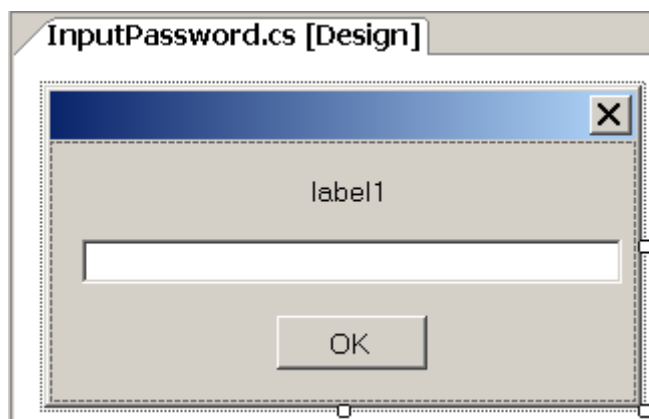
Свойство	Значение	Пояснения
Name	txtInput	Имя текстовой метки
Anchor	None	Расположить посередине
Margin.All	8	Отступ по наружному периметру текстовой метки
PasswordChar	*	Эхо ввода в текстовом поле

**Таблица 5 Свойства кнопки Button**

Свойство	Значение	Пояснения
Name	btnOk	Имя кнопки
AutoSize	true	Подстраиваться под содержимое
Anchor	None	Расположить посередине
Margin.All	8	Отступ по наружному периметру текстовой метки
Text	OK	Надпись
DialogResult	OK	Назначить статус "кнопки OK" в родительской форме

Свойству дочерних кнопок формы DialogResult можно присвоить любое значение из перечисления System.Windows.Forms.DialogResult (Abort, Cancel, Ignore, No, None, OK, Retry). Оно и будет возвращено функцией ShowDialog() формы, если пользователь выберет именно эту кнопку. Если свойству DialogResult кнопки будет присвоен статус, отличный от None, то при щелчке пользователя на такой кнопке форма будет автоматически закрываться.

Подтяните низ формы до приемлемого размера, чтобы она стала выглядеть примерно как на Рис. 9.

**Рис. 9**

Откройте файл `InputPassword.cs` в режиме View Designer, щелкните на заголовке формы и выполните команду `Format/Lock Controls` (в левом верхнем углу формы должен появиться значок замка), чтобы защитить интерфейс формы от случайных изменений. Это групповая команда, которая устанавливает свойство `Locked` всех дочерних визуальных элементов и самой формы в значение `True`. То же самое можно сделать индивидуально для каждого элемента через меню `Properties`.

Мы создали сам пользовательский интерфейс класса `InputPassword` диалогового окна. Теперь нужно настроить класс `InputPassword`, добавив в него метод создания диалогового окна и вызова его на экран.

Модифицируйте класс `InputPassword` следующим образом:

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

namespace WinFormsApp
{
    public partial class InputPassword : Form
    {
        public InputPassword()
        {
            InitializeComponent();
        }

        public InputPassword(string windowTitle, string message)
        {
            InitializeComponent();

            this.Text = windowTitle;
            lblMessage.Text = message;
        }
    }
}
```

```
public static string Show(string windowTitle, string message)
{
    using(InputPassword inputDlg =
        new InputPassword(windowTitle, message))
    {
        inputDlg.ShowDialog();
        return inputDlg.txtInput.Text;
    }
}
}
```

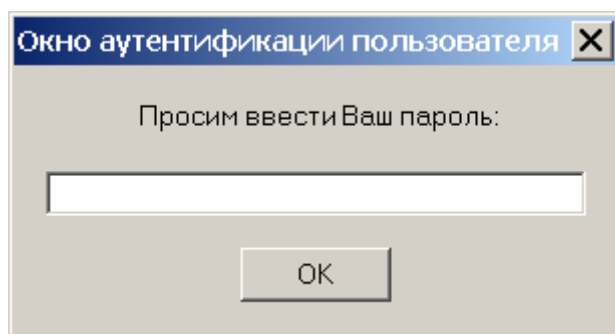
Мы создали параметризованный конструктор для ввода заголовков диалогового окна в момент его создания. Мы добавили статический метод создания диалогового окна, который возвращает значение текстового поля после щелчка пользователем на кнопке, и такой метод нужно вызывать по имени класса. Упаковка создания экземпляра формы в блок using позволяет вызвать сборщик мусора сразу после выхода из этого блока.

Теперь нужно поместить вызов окна запроса пароля в обработчик события Load запуска формы.

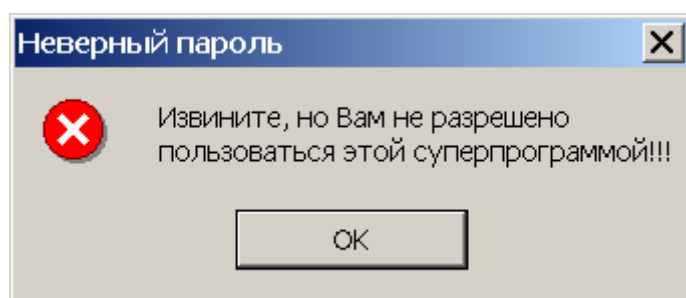
Откройте файл Start.cs в режиме View Designer, выделите форму через ее заголовок, перейдите в панель Properties, установите ее в режим Events, найдите событие Load и создайте для него обработчик Start\_Load(). Заполните обработчик события следующим образом:

```
private void Start_Load(object sender, System.EventArgs e)
{
    string password = InputPassword.Show(
        "Окно аутентификации пользователя",
        "Просим ввести Ваш пароль:");
    if (password != "root") // Плохой способ хранения пароля
    {
        MessageBox.Show(
            "Извините, но Вам не разрешено\n" // Сообщение
            + "пользоваться этой суперпрограммой!!!",
            "Неверный пароль", // Заголовок окна
            MessageBoxButtons.OK, // Кнопка ОК
            MessageBoxIcon.Stop); // Критическая иконка
        this.Close();
    }
}
```

Запустите приложение, диалоговое окно ввода пароля должно выглядеть так:

**Рис. 10**

При неправильном пароле появляется простое диалоговое окно сообщений (Рис. 11), и приложение завершит свою работу.

**Рис. 11**

### Добавление списка для запуска упражнений

На стартовой форме мы создадим список с наименованиями упражнений, который будем пополнять по мере их выполнения. В окончательном варианте работы щелчок по пункту списка должен запускать дочернюю форму, соответствующую выполненному упражнению. Это удобно при демонстрации выполненной лабораторной работы.

Перейдите в режим Design, откройте в панели Toolbox вкладку Common Controls и двойным щелчком на компоненте ListBox создайте на форме его экземпляр.

Выделите объект listBox1 и в панели Properties установите для него следующие значения свойств:

**Таблица 6 Свойства объекта ListBox формы Start**

Свойство	Значение
Name	generalList
Dock	Fill
Items	1) Простое диалоговое окно со смайликами

Щелкните на заголовке формы и увеличьте ее ширину за маркеры так, чтобы поместилось все предложение в списке ListBox.

Через раскрывающийся список вверху панели Properties выделите объект generalList и установите для нее режим Events.

Найдите событие SelectedIndexChanged и двойным щелчком на его поле значений создайте обработчик generalList\_SelectedIndexChanged().

Теперь мы будем добавлять к приложению новые формы для выполнения каждого нового упражнения, регистрировать их в списке и помещать вызовы упражнений в этот

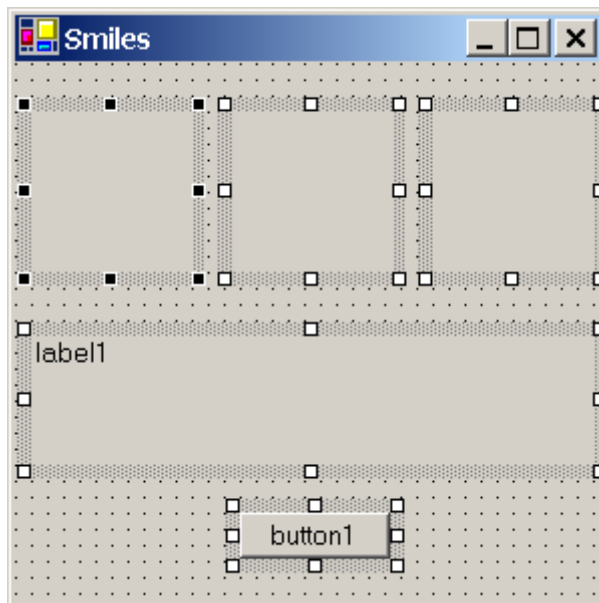
обработчик главной формы. Дальнейшая работа ведется по вариантам (Вариант 1. Смайлики/Вариант 2. Простой калькулятор).

## Вариант 1. Смайлики

Выполните команду Project/Add Windows Form, чтобы добавить к проекту новую форму.

Задайте имя новой форме Smiles.cs (Улыбки).

Поместите на форму три компонента PictureBox из вкладки Common Controls панели Toolbox, текстовую метку Label и одну кнопку Button. Все расположите так, как показано на Рис. 12 (установите свойство label1.AutoSize =false).



**Рис. 12**

Для выравнивания компонентов пользуйтесь командами меню оболочки, входящими в категорию Format, или одноименными кнопками панели инструментов Layout.

Выполните команду меню Format/Lock Controls, чтобы закрепить расположение интерфейсных элементов на форме и, тем самым, защитить их от случайного сдвига или изменения размеров. Обратите внимание, что теперь на выделенных элементах маркеры сменились гладкой рамкой с пиктограммой замка.

Теперь установим для помещенных на форму объектов нужные свойства через панель Properties. При этом важно понимать, что все визуальные манипуляции с формой и помещенными на нее компонентами, в том числе и установка свойств через панель Properties, - это ничто иное, как удобный и наглядный способ генерации кода с помощью графических средств оболочки. Работа в режиме дизайна сопровождается параллельной работой оболочки в режиме кода. И все, что можно сделать через графический режим, можно сделать и вручную через код. Более того, код, набранный вручную, будет менее избыточным, чем это сделает оболочка по нашим визуальным инструкциям.

Графические возможности IDE являются бесспорным помощником только при разработке пользовательского интерфейса окон. Но дальнейшая работа требует от программиста широких профессиональных знаний не только принципов программирования и тонкого знания языка - как основного инструмента, но и знания предметной области, методов и алгоритмов решения поставленной задачи.

Интерфейс пользователя является только посредником. Непонятный интерфейс может сослужить плохую службу действительно толковой программе. Нужно стремиться к простому и понятному интерфейсу, по возможности вписывающемуся в уже сложившиеся представления о принципах программного взаимодействия с пользователем.

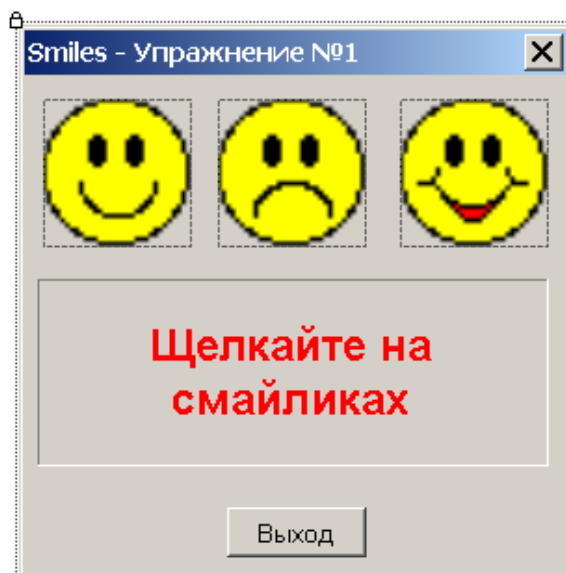
Задайте свойства размещенных на форме объектов через панель Properties в порядке их следования слева-направо, сверху-вниз, как указано в Таблица 7.

**Таблица 7 Значения свойств дочерних объектов формы Smiles**

Класс	Свойство	Значение	Пояснения
Form	Text	Smiles - Упражнение №1	Заголовок окна
	FormBorderStyle	FixedDialog	Запретили изменение размера окна формы
	MaximizeBox	False	Отключили системную кнопку разворачивания окна
	MinimizeBox	False	Запретили системную кнопку свертывания окна
PictureBox	Name	picSmile	Идентификатор объекта (улыбка)
	Image	FACE02.ICO (возьмите из данной работы в каталоге Source. Используйте кнопку Import для радиокнопки Local resource)	Файл рисунка
	SizeMode	StretchImage	Опция - растянуть рисунок по размерам объекта
PictureBox	Name	picFrown	Идентификатор объекта (хмурый взгляд)
	Image	FACE04.ICO (возьмите из данной работы в каталоге Source. Используйте кнопку Import для радиокнопки Local resource)	Файл рисунка
	SizeMode	StretchImage	Опция - растянуть рисунок по размерам объекта

Класс	Свойство	Значение	Пояснения
PictureBox	Name	picHappy	Идентификатор объекта (счастливый)
	Image	FACE03.ICO (возьмите из данной работы в каталоге Source. Используйте кнопку Import для радиокнопки Local resource)	Файл рисунка
	SizeMode	StretchImage	Опция - растянуть рисунок по размерам объекта
Label	Name	lblMessage	Идентификатор объекта (сообщение)
	Text	Щелкайте на смайликах	Начальное содержимое текстовой метки
	Font/Name	Arial	Наименование шрифта
	Font/Size	14	Размер шрифта
	Font/Bold	True	Полужирный
	ForeColor	Red	Цвет переднего плана (красный)
	TextAlign	MiddleCenter	Выравнивание текста - середина по вертикали, центр по горизонтали
	BorderStyle	Fixed3D	Объемный вид элемента
Button	Name	btnExit	Идентификатор объекта
	Text	Выход	Надпись на кнопке
	AutoSize	true	Подстраиваться под размер надписи

После всех настроек внешний вид формы должен быть как на Рис. 13.

**Рис. 13**

Разработку интерфейса первой дочерней формы на этом можно считать завершенной. Теперь необходимо обеспечить ее функциональность.

Выделите первый рисунок `picSmile` и через панель `Properties` (в режиме `Events`) создайте обработчик события `Click`.

Заполните обработчик следующим кодом.

```
private void picSmile_Click(object sender, System.EventArgs e)
{
    picSmile.BorderStyle = BorderStyle.FixedSingle; // Рамка
    picFrown.BorderStyle = BorderStyle.None; // Нет рамки
    picHappy.BorderStyle = BorderStyle.None; // Нет рамки
    lblMessage.Text = "Щелкнули на первом рисунке";
}
```

Двойным щелчком на втором рисунке (`picFrown`) создайте для него обработчик, который заполните следующим образом:

```
private void picFrown_Click(object sender, System.EventArgs e)
{
    picSmile.BorderStyle = BorderStyle.None; // Нет рамки
    picFrown.BorderStyle = BorderStyle.FixedSingle; // Рамка
    picHappy.BorderStyle = BorderStyle.None; // Нет рамки
    lblMessage.Text = "Щелкнули на втором рисунке";
}
```



Двойным щелчком на третьем рисунке (picHappy) создайте для него обработчик, который заполните так :

```
private void picHappy_Click(object sender, System.EventArgs e)
{
    picSmile.BorderStyle = BorderStyle.None; // Нет рамки
    picFrown.BorderStyle = BorderStyle.None; // Нет рамки
    picHappy.BorderStyle = BorderStyle.FixedSingle; // Рамка
    lblMessage.Text = "Щелкнули на третьем рисунке";
}
```

Двойным щелчком на кнопке btnExit создайте для нее обработчик, который заполните следующим образом:

```
private void btnExit_Click(object sender, System.EventArgs e)
{
    this.Close(); // Закрыть форму
}
```

Теперь подключите вызов этой дочерней формы к движку приложения, обеспечив возможность ее вызова по щелчку на соответствующей строке списка с упражнениями.

Сохраните проект командой меню File/Save All и закройте все окна редактирования командой Window/Close All Documents.

В панели Solution Explorer щелкните правой кнопкой мыши на файле Start.cs и выполните команду контекстного меню View Designer (или откройте файл в режиме дизайна двойным щелчком левой кнопки мыши).

Выделите элемент списка generalList и через панель Properties щелкните два раза на имени уже созданного обработчика в поле события SelectedIndexChanged, чтобы быстро позиционироваться к заголовку кода обработчика. Заполните обработчик следующим кодом:

```
private void generalList_SelectedIndexChanged(object sender,
System.EventArgs e)
{
    switch (generalList.SelectedIndex + 1)
    {
        case 1:
            Smiles frm1 = new Smiles();
            frm1.ShowDialog();
            break;
    }
}
```

Здесь мы применили метод ShowDialog() класса Form (вместо метода Show()). Это позволит нам открывать дочерние формы в модальном режиме (пока не закроется очередная форма, другие окна приложения не получают фокус ввода).

Выделите в режиме дизайна форму Start (щелкните на заголовке окна формы) и установите через панель Properties ее свойство StartPosition в значение CenterScreen, чтобы при первом запуске она появлялась в центре экрана.

Откройте в режиме дизайна (в панели Solution Explorer) форму Smiles, выделите ее через заголовок окна формы и установите свойство `StartPosition` в значение `Manual`. Проследите, чтобы свойство `Location` (размещение) имело значение `(0, 0)`. Тем самым мы задали позицию появления дочерней формы в верхнем левом углу экрана.

Два последних действия позволят держать форму движка в центре экрана, а дочерние формы переместить в начало экрана (удобно для демонстрации упражнений, полезно для тренировки).

Запустите приложение и проверьте его работоспособность.

На этапе построения формы бывает полезным сразу запускать ее для проверки. В нашем случае форма `Smiles` может быть вызвана только из главной формы. Исправим это.

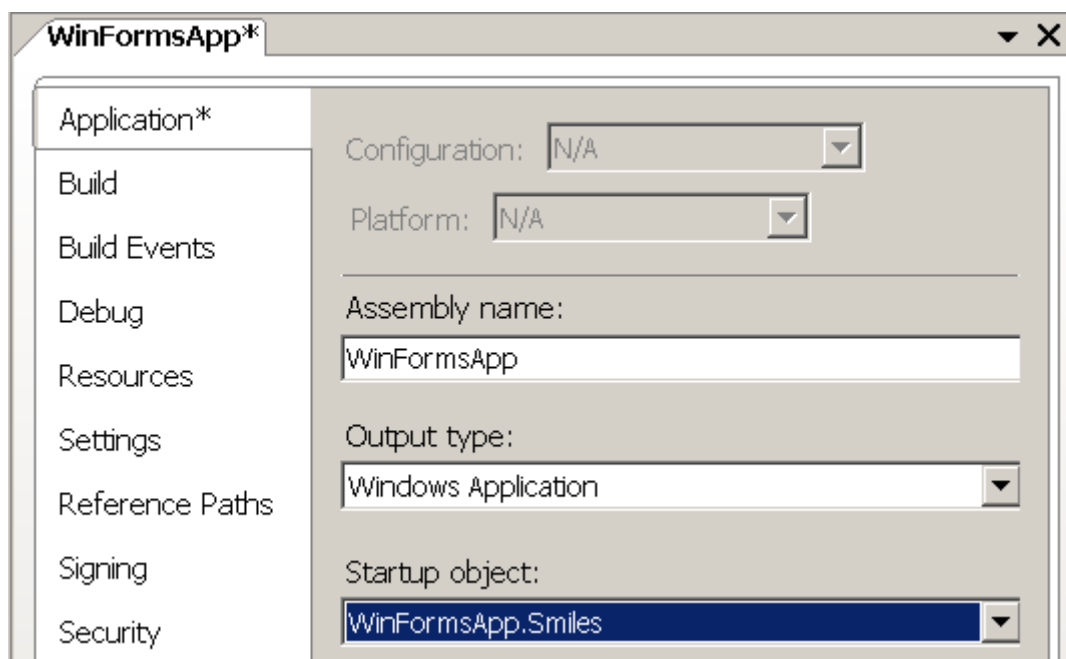
Добавьте в конец класса `Smiles` следующую функцию-член `Main()`, которая позволит отображать форму первой при запуске приложения:

```
public class Smiles : System.Windows.Forms.Form
{
    .....

    private void btnExit_Click(object sender, System.EventArgs e)
    {
        this.Close(); // Закрыть форму
    }

    static void Main()
    {
        Application.Run(new Smiles());
    }
}
```

Установите в качестве стартовой форму `Smiles`. Для этого выполните команду меню оболочки `Project/WinFormsApp Properties` и в появившемся диалоговом окне (Рис. 14) установите свойство `Startup Object` в значение `WinFormsApp.Smiles`.



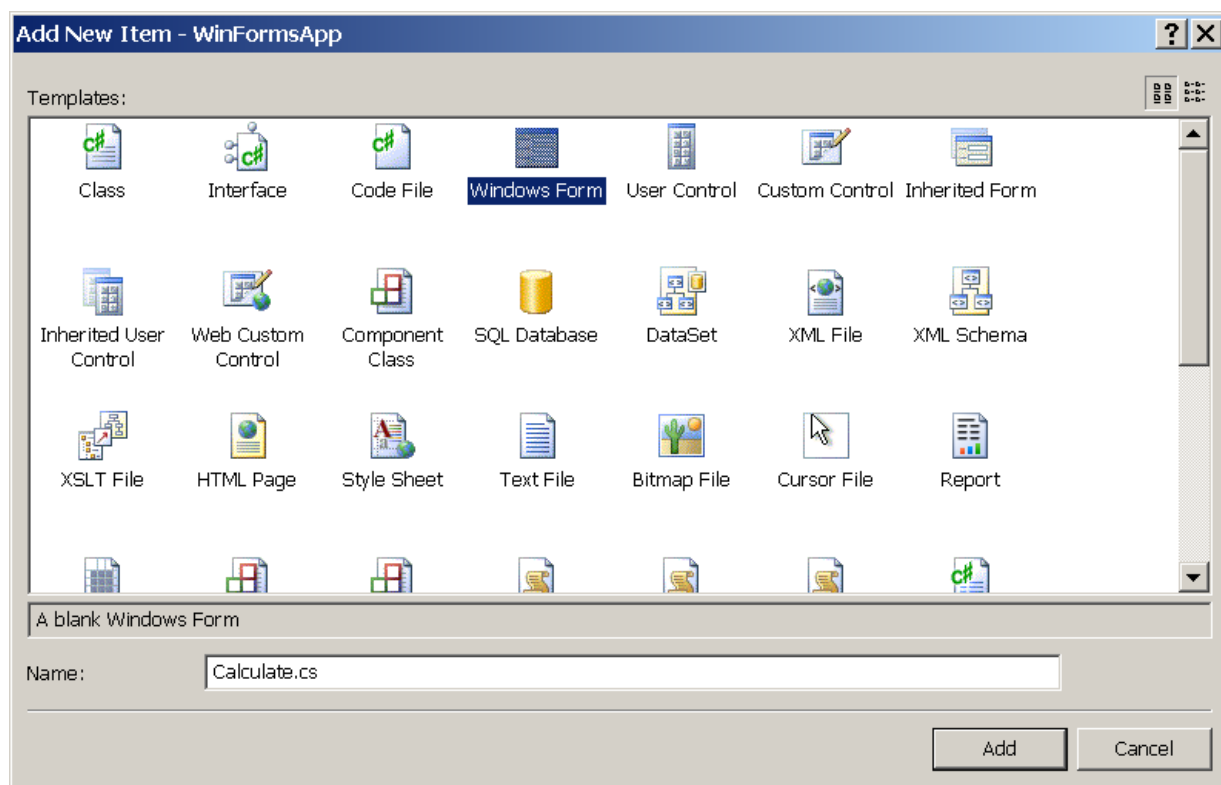
**Рис. 14**

Запустите приложение и убедитесь, что теперь дочерняя форма Smiles стартует первой и единственной.

## Вариант 2. Простой калькулятор

Разработаем калькулятор, выполняющий простые действия над целыми и вещественными числами в десятичной форме.

Добавьте к проекту новую форму командой Project/Add Windows Form и задайте ей имя Calculate.



**Рис. 15**

Поместите на форму элементы из вкладки Common Controls панели Toolbox в соответствии с таблицей, чтобы пользовательский интерфейс формы соответствовал схеме калькулятора.

**Таблица 8 Таблица свойств формы Calculate**

Элемент	Свойство	Значение
Form	Name	Calculate
	StartPosition	Manual
	Location	0; 0
	Text	Простой калькулятор
	Icon	Выбрать по усмотрению... (можно из Source)
	FormBorderStyle	FixedDialog

Элемент	Свойство	Значение
Label	Name	lblFirst
	Text	Первое число:
	TextAlign	MiddleRight
Label	Name	lblSecond
	Text	Второе число:
	TextAlign	MiddleRight
Label	Name	lblResult
	Text	Результат:
	TextAlign	MiddleRight
TextBox	Name	txtFirst
	Text	пусто
TextBox	Name	txtSecond
	Text	пусто
TextBox	Name	txtResult
	Text	пусто
	ReadOnly	True
	TabStop	False
Button	Name	btnIncrement
	Text	+
	Font/Bold	True
	Font/Size	20
	Size	40; 40
Button	Name	btnDecrement
	Text	-
	Font/Name	Courier New
	Font/Bold	True
	Font/Size	20
	Size	40; 40

Элемент	Свойство	Значение
Button	Name	btnIncrease
	Text	*
	Font/Name	Courier New
	Font/Bold	True
	Font/Size	20
	Size	40; 40
Button	Name	btnDivide
	Text	:
	Font/Name	Courier New
	Font/Bold	True
	Font/Size	20
	Size	40; 40
Button	Name	btnExit
	Text	Выход

Здесь мы использовали некоторые префиксы при назначении имен элементам управления. Для лучшей узнаваемости в коде программы элементов управления и их типа рекомендуется использовать общеупотребительные префиксы, которые приведены в Таблица 10.

Добавьте к классу Calculate функцию Main() для автономной отладки формы:

```
static void Main()
{
    Application.Run(new Calculate());
}
```

Выполните команду меню WinFormsApp Properties и для вкладки Application установите в свойстве Startup Object форму Calculate стартовой.

Защитите интерфейс формы Calculate от случайных изменений, выполнив команду Format/Lock Controls меню оболочки.

Теперь нужно реализовать функциональность интерфейса. Прежде всего нужно защитить поля ввода от всех символов, кроме цифр, одного знака минус и одной точки. Стрелки и клавиша Del в текстовых полях функционируют, но работу клавиши Backspace нужно учесть явно.

Объявите в конце класса Calculate логическое поле isNumeric для отфильтровывания нецифровых символов, которые пользователь может попытаться ввести в текстовые поля.

```
private bool isNumber = false;
```

Выделите текстовое поле txtFirst и через панель Properties в режиме Events создайте обработчик для события KeyDown. Заполните обработчик так:

```
private bool isNumber = false;

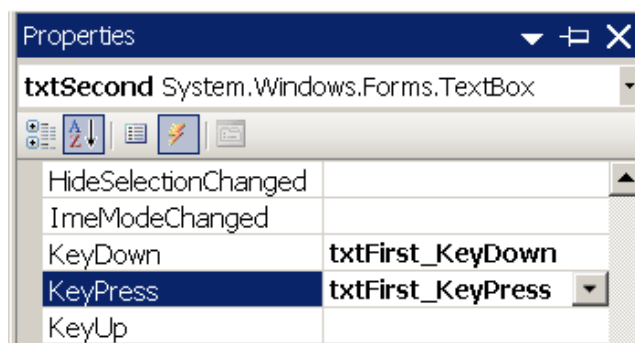
private void txtFirst_KeyDown(object sender,
System.Windows.Forms.KeyEventArgs e)
{
    isNumber =
        e.KeyCode >= Keys.D0 && e.KeyCode <= Keys.D9 // keyboard -
основная клавиатура
        || e.KeyCode >= Keys.NumPad0 && e.KeyCode <= Keys.NumPad9 //
keypad - дополнительная клавиатура
        || e.KeyCode == Keys.Back;
}
```

Выделите текстовое поле txtFirst и через панель Properties в режиме Events создайте обработчик для события KeyPress. Заполните обработчик так:

```
private void txtFirst_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
{
    TextBox box = (TextBox) sender; // Явное преобразование типов

    switch(e.KeyChar) // Переключатель
    {
        case '-': // Разрешаем минус, если он первый
            if(box.Text.Length == 0)
                isNumber = true;
            break;
        case '.':
            // Точка не должна быть первой
            if(box.Text.Length == 0)
                break;
            // Точка не должна следовать сразу за минусом
            if(box.Text[0] == '-' && box.Text.Length == 1)
                break;
            // Точка должна быть одна
            if(box.Text.IndexOf('.') == -1)
                isNumber = true; // Еще не было точек
            break;
    }
    // Запрещаем в текстовом поле лишние символы
    if(!isNumber)
        e.Handled = true;
}
```

Выделите текстовое поле txtSecond и через панель Properties в режиме Events подключите к событиям KeyDown и KeyPress соответствующие обработчики, только что созданные для поля txtFirst. Обработчики подключите через раскрывающиеся списки полей значений, как показано на Рис. 16.



**Рис. 16**

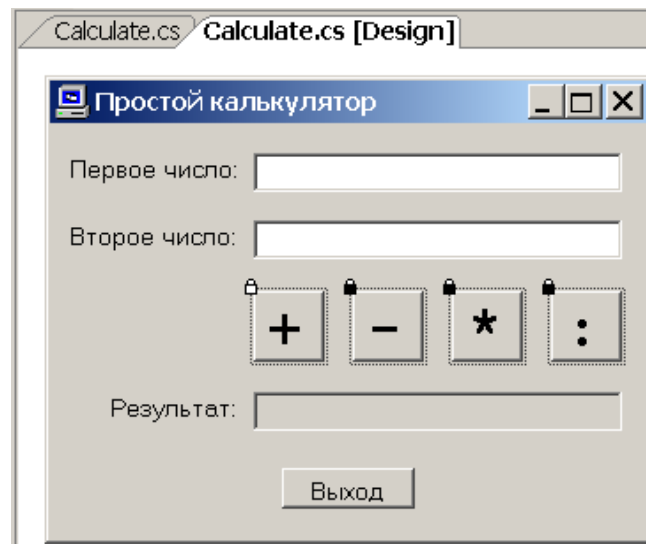
Здесь мы одни и те же обработчики используем для разных полей, поскольку необходимо выполнять одинаковые действия. Задача одна - не допускать для ввода в поля никакие другие символы, кроме цифр, одного минуса и одной точки. Возможность ввода цифр реализована для основной (keyboard) и дополнительной (keypad) частей клавиатуры. Параметр sender передает в обработчик ссылку на активное текстовое поле, вызвавшее событие редактирования.

Конечно, программа будет более понятна, если для каждого элемента создавать свой обработчик. Но очень часто, как в нашем случае, преобладающая часть кода может быть совершенно одинакова в каждом обработчике. Тогда имеет смысл объединять несколько индивидуальных обработчиков в один. Какой подход выбрать - зависит от конкретной задачи и предпочтений программиста.

Добавьте в конец класса Calculate объявления числовых переменных, с которыми мы будем производить простые арифметические операции.

```
// Объявляем числовые переменные как поля-члены класса
// Их можно объявить как локальные и внутри обработчика btn_Click()
// Но оставляем область видимости класс, вдруг где-нибудь еще
пригодятся!
private double numFirst, numSecond, numResult;
```

Перейдите в режим Design формы Calculate и выделите все четыре кнопки с операциями:

**Рис. 17**

Установите в панели Properties режим Events, найдите событие Click и введите вручную в поле значений этого события имя обработчика btn\_Click.

В созданный оболочкой общий обработчик для всех выделенных кнопок введите следующий код:

```
// Один обработчик для всех кнопок-операций
private void btn_Click(object sender, System.EventArgs e)
{
    // Копируем текстовые поля в локальные переменные
    string strFirst = string.Copy(txtFirst.Text);
    string strSecond = string.Copy(txtSecond.Text);

    // Замена в строке точки символом запятой
    // для корректного преобразования в число
    int pos = strFirst.IndexOf('.');
    if(pos != -1)
    {
        strFirst = strFirst.Substring(0, pos)
            + ','
            + strFirst.Substring(pos + 1);
    }
    pos = strSecond.IndexOf('.');
    if(pos != -1)
    {
        strSecond = strSecond.Substring(0, pos)
            + ','
            + strSecond.Substring(pos + 1);
    }
}
```



```
// Преобразуем текст в число для выполнения операций
if(txtFirst.Text.Length > 0)
    numFirst = Convert.ToDouble(strFirst);
else
    numFirst = 0.0D;
if(txtSecond.Text.Length > 0)
    numSecond = Convert.ToDouble(strSecond);
else
    numSecond = 0.0D;

// Выполняем нужную операцию
string btnText = ""; // Создали строковую переменную
bool divideFlag = false; // Флаг деления на ноль
Button btn = (Button) sender; // Явное приведение типов для
распознавания кнопок
switch(btn.Name) // Переключатель
{
    case "btnIncrement": // Операция сложения
        btnText = "\"+\""; // Экраны кавычек
        numResult = numFirst + numSecond;
        break;
    case "btnDecrement": // Операция вычитания
        btnText = "\"-\""; // Экраны кавычек
        numResult = numFirst - numSecond;
        break;
    case "btnIncrease": // Операция умножения
        btnText = "\"*\""; // Экраны кавычек
        numResult = numFirst * numSecond;
        break;
    case "btnDivide": // Операция деления
        btnText = "\":\\""; // Экраны кавычек
        // Проверяем корректность деления
        if (Math.Abs(numSecond) < 1.0E-30)
        {
            MessageBox.Show(
                "Делить на ноль нельзя!", // Сообщение
                "Ошибка", // Заголовок окна
                MessageBoxButtons.OK, // Кнопка ОК
                MessageBoxIcon.Stop); // Критическая иконка
            divideFlag = true;
        }
        else
            numResult = numFirst / numSecond;
        break;
}
```

```
// Для отображения в панели Output режима Debug
System.Diagnostics.Debug.WriteLine("Нажата кнопка " + btnText); //
Конкатенация

// Отображение результата
if(!divideFlag)
{
    txtResult.Text = Convert.ToString(numResult);
    this.Validate(); // Обновить экран (можно убрать-излишне)
}
}
```

Здесь мы прежде всего скопировали значения полей ввода в промежуточные текстовые переменные, используя статический метод `System.String.Copy()` класса `String` (псевдоним `string`). Затем мы проверяем наличие символа "точки" в этих переменных и при обнаружении производим замену на символ "запятая" с полным формированием новой строки. Мы не можем заменить символ "на месте" в существующей строке, обратившись к нему как элементу массива, поскольку индексная форма адресации в строках существует только для чтения.

Но мы можем, сформировав новую строку справа от знака присваивания, присвоить результат вновь той же самой ссылке, которая фактически будет адресоваться к новой области памяти, а старая строка, как брошенный фрагмент памяти, будет оставлена для сборщика мусора. Операция замены точки на запятую выполняется для того, чтобы корректно преобразовать текстовое значение поля ввода в числовое, иначе генерируется исключение.

Далее мы явно приводим переданную в обработчик ссылку типа `object` на нажатую кнопку к типу `Button`. В заголовке переключателя выделяем имя нажатой кнопки и выполняем соответствующую арифметическую операцию. Перед выполнением последней операции мы проверяем возможность возникновения случая "деления на ноль" и при его обнаружении предупреждаем пользователя выводом простого диалогового окна сообщений, отменяя саму операцию деления.

После переключателя мы располагаем оператор вывода строки `System.Diagnostics.Debug.WriteLine()` с выполненной операцией в панель оболочки `Output`. Этот класс предназначен для работы с программой в отладочном режиме `Debug` и автоматически изымается из кода при компиляции окончательного варианта приложения в режиме `Release`.

На последнем этапе, в коде обработчика, мы выводим результат в соответствующее текстовое поле с предварительным преобразованием типов. Затем мы даем команду системе обновить окно приложения, хотя в данном случае эта мера является излишней - система и так обновляет результат в текстовом поле.

Теперь создадим обработчик нажатия на кнопку закрытия формы.

В режиме `Design` двойным щелчком на кнопке `btnExit` создайте обработчик, который заполните так:

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

Запустите проект и проверьте функциональность простого калькулятора.

Теперь мы должны подключить разработанную форму калькулятора к основному окну приложения.

Установите в качестве стартовой форму Start. Для этого выполните команду меню оболочки Project/WinFormsApp Properties и в появившемся диалоговом окне установите для вкладки Application свойство Startup Object в значение WinFormsApp.Program.

Откройте файл Start.cs в режиме View Code. Через раскрывающийся список Members в верхней части окна редактора кода найдите обработчик события generalList\_SelectedIndexChanged(), который мы создали ранее, и дополните его вызовом окна калькулятора:

```
private void generalList_SelectedIndexChanged(object sender,
System.EventArgs e)
{
    switch(generalList.SelectedIndex + 1)
    {
        case 1:
            Smiles frm1 = new Smiles();
            frm1.ShowDialog();
            break;
        case 2:
            Calculate frm2 = new Calculate();
            frm2.ShowDialog();
            break;
    }
}
```

В панели Solution Explorer оболочки щелкните дважды на пиктограмме файла Start.cs, чтобы открыть его в режиме View Designer, выделите элемент списка в клиентской области формы и через панель Properties добавьте в свойство Items строку "1) Простой калькулятор".

Запустите приложение и проверьте его работоспособность.

## Индивидуальные задания

### Вариант 1

1. Измените программу так, чтобы при выборе любого изображения цвет остальных менялся случайным образом.
2. Измените код таким образом, чтобы в ходе исполнения приложения при выборе одного рисунка он пропадал с экрана, но при выборе другого появлялся обратно.
3. Измените код таким образом, чтобы в ходе исполнения приложения при наведении на ячейку одного рисунка цвета других ячеек менялись на любые (в произвольном порядке).
4. Измените код таким образом, чтобы в ходе исполнения приложения при выборе одного рисунка цвет ячейки слева изменялся бы на красный, а левой на синий.
5. Измените код программы так, чтобы при выборе «грустного» смайлика программа запрашивала бы пароль. Также добавить кнопку на форму с вводом пароля "Забыл пароль" с подсказкой пароля.
6. Переделайте программу таким образом, чтобы при выборе одной картинке, при первом нажатии две другие менялись между собой местами и при втором появлялись новые рисунки, при третьем нажатии возвращалась все на свои места.
7. Измените программу так, чтобы при выборе «веселого» смайлика программа запрашивала бы пароль, при неправильном вводе пароля, смайлы менялись местами.
8. Измените код таким образом, чтобы в ходе исполнения приложения при выборе одного рисунка два других исчезали с формы.
9. Напишите код, при котором кнопка «Выход» будет доступна только при вводе пароля до этого будет находиться в невидимом состоянии до третьего и после пятого кликов по картинкам.
10. Измените код программы так, чтобы каждой картинке Smile соответствовал бы свой цвет фона ячейки lblMessage.
11. Измените код таким образом, чтобы в ходе исполнения приложения при выборе одного рисунка цвет его ячейки изменялся бы на зеленый.
12. Задание повышенной сложности (по желанию): Напишите код, сделав перемещение ячеек с рисунками от мышки, чтобы появлялось в произвольном месте на поле, но не в том месте, где уже была ячейка и не попадала на другие ячейки и не выходила за края окна приложения, размер оконого приложения усановить 600x500.









**Вариант 2**










1. Добавьте кнопку «=» на форму и создайте обработчик для нее.
2. Измените код программы так, чтобы результат операции выводился в отдельном диалоговом окне.
3. Добавьте на форму кнопки основных констант и реализуйте их использование.
4. Добавьте функцию возведения в квадрат.
5. Реализуйте расчет процентов.
6. Добавьте функцию извлечения квадратного корня.
7. Добавьте на форму клавишу очистки содержимого всех контейнеров и реализуйте ее функционирование.
8. Реализуйте функцию подсчета синуса острого угла.
9. Реализуйте функцию подсчета косинуса острого угла.
10. Реализуйте функцию подсчета тангенса острого угла.
11. Реализуйте возведение в любую степень.
12. Сделайте подсчет факториала.
13. Сделайте подсчет десятичного логарифма.
14. Добавьте изменение числа в отрицательное при нажатии первый раз и обратно при нажатии второй раз.
15. Задание повышенной сложности (по желанию): Добавить буфер, который будет хранить информацию и эту информацию можно будет вызвать и сделать пересчет, если это понадобится.











## Приложение 1 Некоторые компоненты панели Toolbox

В панели Toolbox имеется возможность добавлять свои вкладки и копировать в них нужный набор инструментов из других вкладок. Добавить вкладку можно, щелкнув правой кнопкой мыши в любом месте панели Toolbox и выполнив команду контекстного меню Add Tab. Удалить вкладку можно, щелкнув правой кнопкой мыши на заголовке соответствующей вкладки и выполнив команду Delete Tab. Копировать нужные компоненты из одной вкладки в другую можно выполнением команд контекстного меню Copy и Paste.











**Таблица 9 Некоторые компоненты панели Toolbox**











Компонент	Назначение
 Pointer	Установка курсора в режим выделения объектов.
 Button	Инструмент для генерирования события щелчка, по которому вызывается функция - обработчик. В эту функцию можно разместить любой код, который требуется выполнить при щелчке на этом инструменте.
 CheckBox	Дает возможность пользователю установить нужный режим из двух возможных. При объединении этих элементов в тематическую группу пользователь может производить множественные настройки нашего приложения.
 CheckedListBox	Множество элементов CheckBox, объединенных в скролирующую группу.
 ColorDialog	Стандартное диалоговое окно для предоставления пользователю возможности выбирать цвета
 ComboBox	Раскрывающийся список. Состоит из верхней части с редактируемым текстовым полем, и нижней части со списком для выбора готовых элементов.
 ContextMenuStrip	Контекстные меню обычно вызываются щелчком правой кнопкой мыши на нужном элементе управления формы. Туда обычно включают часто используемые команды из основного меню приложения. Это обеспечивает быстрый доступ пользователя к таким командам.
 	Этот элемент отображает данные в виде таблицы, состоящей из строк и столбцов. В простейшем случае этот

DataGridView	элемент просто отображает все данные связанной с ним одной таблицы базы данных без всяких условий и позволяет оперировать с ними как в обычной электронной таблице. В более сложном случае элемент отображает выборку данных из нескольких связанных таблиц по некоторым условиям.
 DateTimePicker	Этот элемент дает возможность пользователю быстро выбирать нужные дату из предоставляемого удобного интерфейса в виде календаря.
 DomainUpDown	Представляет собой текстовое поле со стрелками для изменения значений списка вверх или вниз.
 ErrorProvider	Этот элемент управления (невизуальный компонент) применяется для проверки правильности пользовательского ввода в допустимых пределах данных. Он предоставляет разработчику удобный интерфейс для оповещения пользователя о допущенных нарушениях и является лучшей заменой простому окну сообщений.
 FolderBrowserDialog	Представляет собой диалоговое окно для навигации по папкам файловой системы компьютера.
 FontDialog	Является стандартным диалоговым окном, с помощью которого можно выбирать шрифты, установленные в настоящее время на текущем компьютере.
 GroupBox	Применяется для связывания элементов управления в единую группу как дочерних. Используется для наглядности компактного размещения логически связанных элементов и обеспечения их согласованной работы. Перемещение элемента GroupBox в режиме дизайна в пределах формы приводит к одновременному перемещению всех включенных в группу элементов.
 HelpProvider	Используется для связывания файлов справки (*.chm, *.htm) с приложением.
 HScrollBar	Самостоятельный элемент управления, который присоединяется к другим элементам интерфейса для продвижения по большим массивам информации в горизонтальном направлении.
 ImageList	Используется для хранения последовательности рисунков, подлежащих отображению приложением. Рисунки можно выбирать с помощью индексов списка и отображать на экране другими элементами управления.

 Label	Простая текстовая метка для отображения не редактируемого текста на экране компьютера.
 LinkLabel	Позволяет использовать текстовую метку в стиле гиперссылки. С помощью этого элемента часть текста можно связать с файлом, каталогом или Web-страницей.
 ListBox	Отображает список элементов, из которых пользователь может выбрать один или несколько элементов. Если количество элементов в списке превышает то, что может быть отображено, то автоматически добавляется вертикальная полоса прокрутки. При установке свойства MultiColumn в значение true добавляется горизонтальная полоса прокрутки. Если установлено в true свойство ScrollAlwaysVisible, то полоса прокрутки появляется всегда, независимо от количества элементов. Свойство SelectionMode устанавливает, сколько элементов списка может быть выделено одновременно. Свойство SelectedIndex возвращает индекс первого выделенного элемента в списке, начиная с нуля. Если нет выделенных элементов, свойство возвращает значение -1.
 ListView	Отображает список элементов с иконками, наподобие правой области Windows Explorer. Элемент имеет четыре режима отображения: LargeIcon, SmallIcon, List, и Details.
 MenuStrip	Создает главное меню формы, за пунктами которого можно закреплять различные команды управления приложением.
 MonthCalendar	Обеспечивает привычный интерфейс работы с датой в виде календаря.
 NotifyIcon	Используется для отображения в виде значков процессов, которые выполняются в фоновом режиме и долгое время не видны пользователю. Примером может служить программа антивирусной защиты, которую при необходимости пользователь может вызвать щелчком на уведомляющем значке в панели задач.
 NumericUpDown	Представляет собой комбинацию текстового поля и пары стрелок, которые пользователь может применять для наглядного управления целым числовым значением.
 OpenFileDialog	Представляет стандартное диалоговое окно открытия файла.
	Предоставляет стандартное диалоговое окно настройки страницы документа при печати.



PageSetupDialog	
 Panel	Элемент Panel подобен элементу GroupBox. Применяется для связывания элементов управления в единую группу как дочерних. Используется для наглядности компактного размещения логически связанных элементов и обеспечения их согласованной работы. Имеет собственные полосы прокрутки, но не отображает заголовок.
 PictureBox	Элемент служит для отображения рисунков в формате bitmap, GIF, JPEG, metafile, или icon. Отображаемый рисунок определяется свойством Image. Свойство SizeMode устанавливает режим согласования размеров элемента и рисунка. Рисунок отображается как в режиме разработки, так и во время выполнения приложения.
 PrintDialog	Это стандартное диалоговое окно используется для настроек принтера перед печатью.
 PrintDocument	Устанавливает, что печатать и как печатать. Может использоваться совместно с элементом PrintDialog для полного описания всех аспектов печати.
 PrintPreviewControl	Позволяет предварительно просмотреть документ перед печатью, как он будет выглядеть после печати.
 PrintPreviewDialog	Позволяет предварительно просмотреть документ перед печатью, как он будет выглядеть после печати. Отображает стандартное диалоговое окно с необходимыми элементами управления.
 ProgressBar	Индикаторная линейка - отображает продвижение процесса в горизонтальном направлении.
 RadioButton	Радиокнопка (переключатель)
 RichTextBox	Представляет собой полноценный текстовый редактор.
 SaveFileDialog	Обеспечивает стандартный настраиваемый диалог сохранения файлов.

 Splitter	Используется для изменения относительных размеров склеенных элементов управления во время выполнения приложения.
 StatusBar	Элемент создает строку состояния.
 TabControl	Элемент для создания вкладок. Создает одну или несколько вкладок, на каждой из которых могут быть размещены другие объекты.
 TextBox	Редактируемое текстовое поле.
 Timer	Таймер
 ToolStrip	Панель инструментов - контейнер для кнопок быстрого доступа.
 ToolTip	Для создания всплывающих коротких подсказок, которые появляются при наведении курсора на элемент управления.
 TrackBar	Регулятор для визуального изменения числовой информации (иногда называют Slider - бегунок).
 TreeView	Отображает информацию в древовидном формате.
 VScrollBar	Самостоятельный элемент управления, который присоединяется к другим элементам интерфейса для продвижения по большим массивам информации в вертикальном направлении.

## Приложение 2. Рекомендуемые префиксы для некоторых элементов управления

**Таблица 10** Список рекомендуемых префиксов

Объект	Префикс
Button	btn
CheckBox	chk
ComboBox	cmb
DataGrid	dat
Form	frm
Label	lbl
ListBox	lst
MenuStrip	mnu
PictureBox	pic
RadioButton	rad
TextBox	txt

## Лабораторная Работа №2

### Лабораторная работа: Пользовательские элементы управления

Краткое описание: Графический интерфейс пользователя (GUI - Graphics User Interface). Построение элемента управления BeepButton. Переопределение наследуемого от Control виртуального метода диспетчеризации события Click. Проектирование диалогового окна сообщений MyDialogBox. Тестирование пользовательских элементов BeepButton и MyDialogBox. Рисование круглой кнопки CircleButton и ее тестирование.

## Основная часть

Элементы управления являются посредниками между приложением и пользователем. Они составляют так называемый графический интерфейс пользователя (GUI - Graphics User Interface). Их задача состоит в отображении информации для пользователя, обеспечении удобного ввода и генерации событий, отрабатывающих предпочтения пользователя.

Как правило, библиотечных элементов управления достаточно для решения большинства задач, но иногда программисту может потребоваться нестандартный элемент управления. В таком случае он может разработать его самостоятельно с нуля или расширить стандартный элемент. Такой элемент управления называют пользовательским.

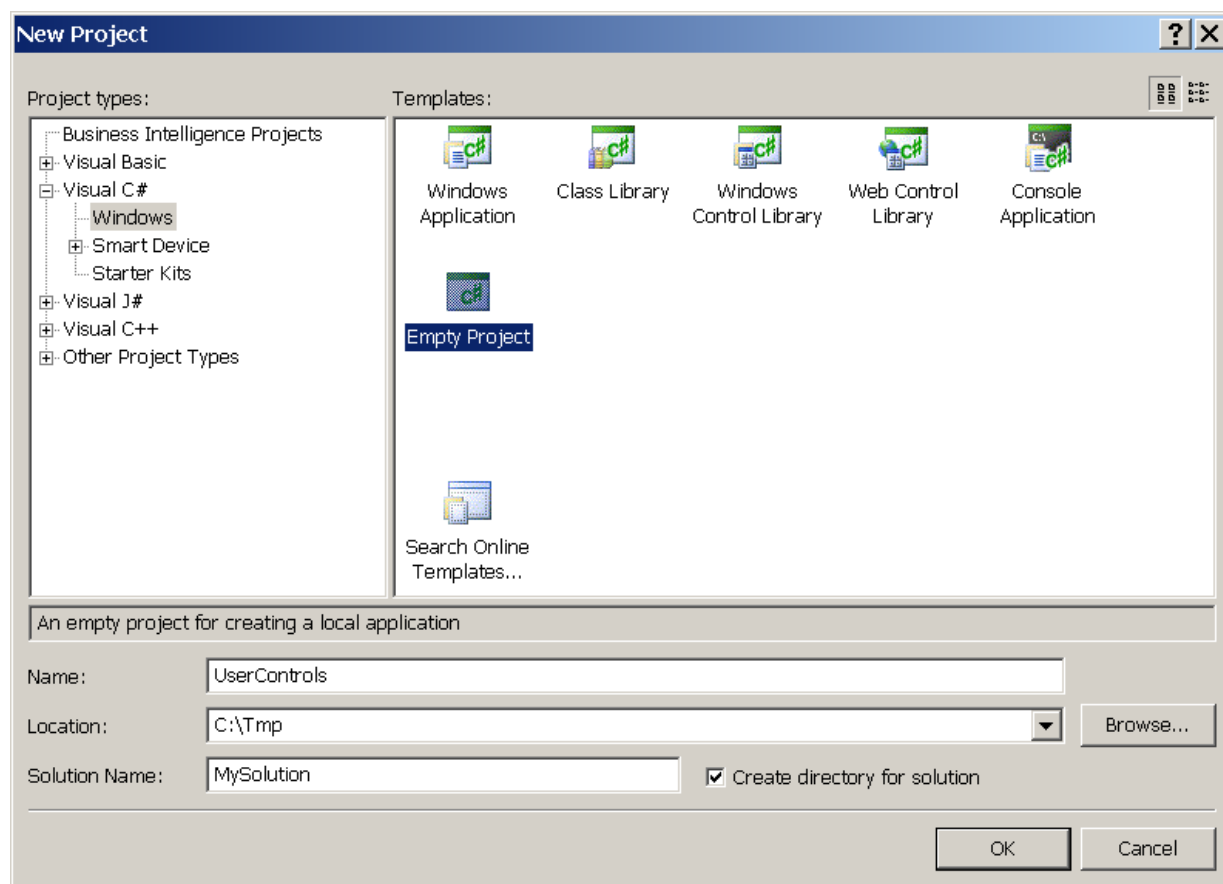
Механизм наследования C# предоставляет широкие возможности модификации существующих элементов управления на уровне расширения классов. Взяв за основу библиотечный элемент управления, мы можем использовать его в качестве базового и внутри производного класса (расширения) переопределять виртуальные методы и свойства, добавлять новые поля, свойства, события.

### Построение элемента управления BeepButton

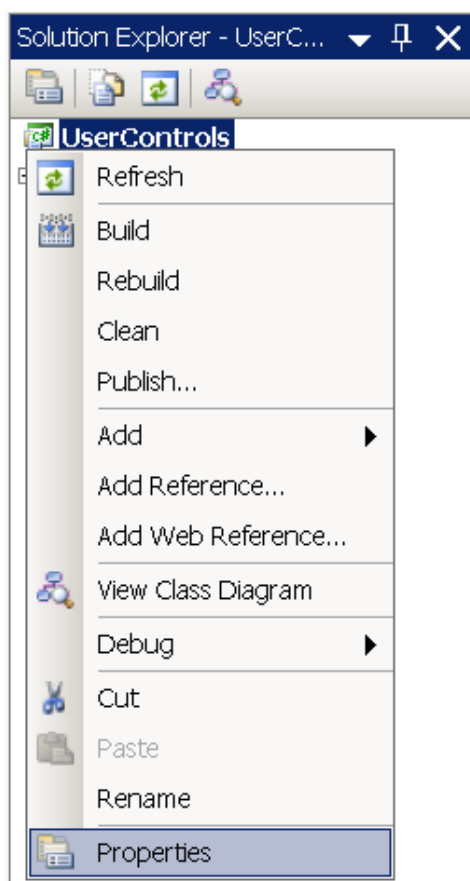
Построим кнопку, при щелчке на которой компьютер выдаст звуковой сигнал. Для этого используем библиотечный элемент управления Button в качестве базового класса и расширим его для решения своей задачи.

### Создание заготовки библиотеки компонентов

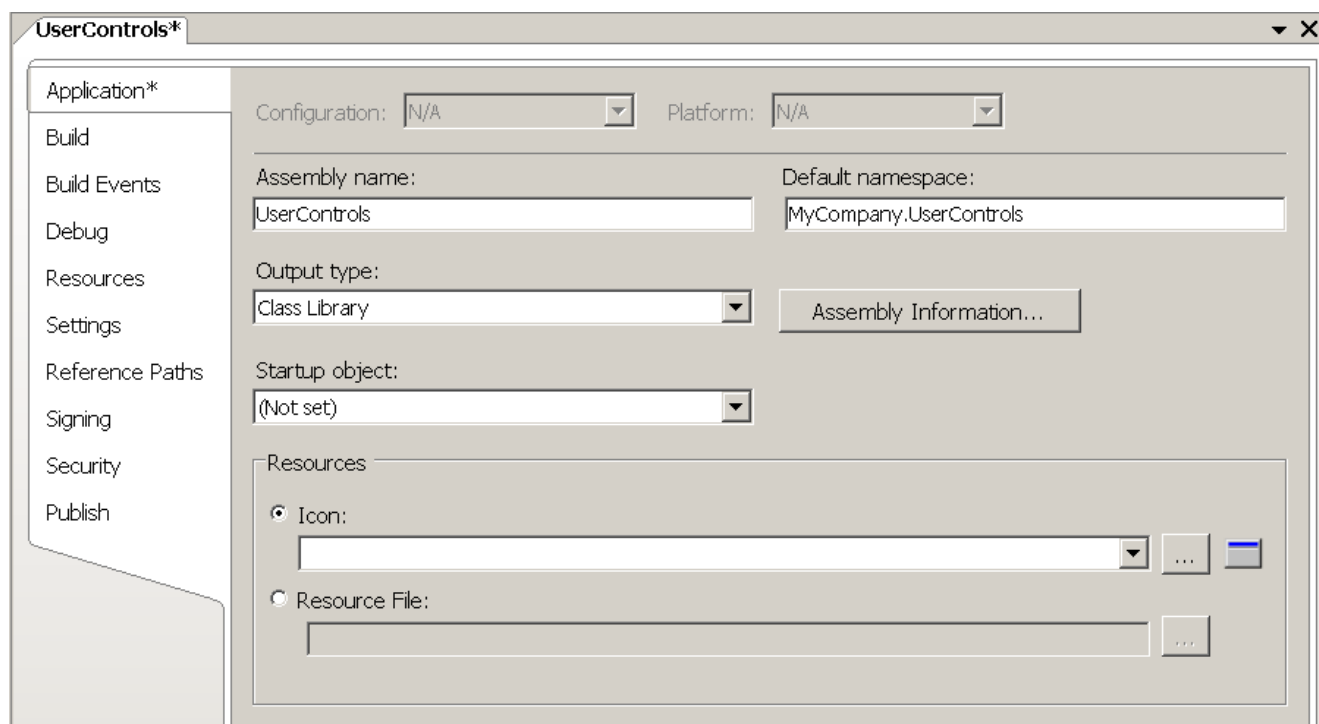
Создайте новый пустой проект командой File/New/Project и заполните окно мастера так, как показано на Рис. 18.

**Рис. 18**

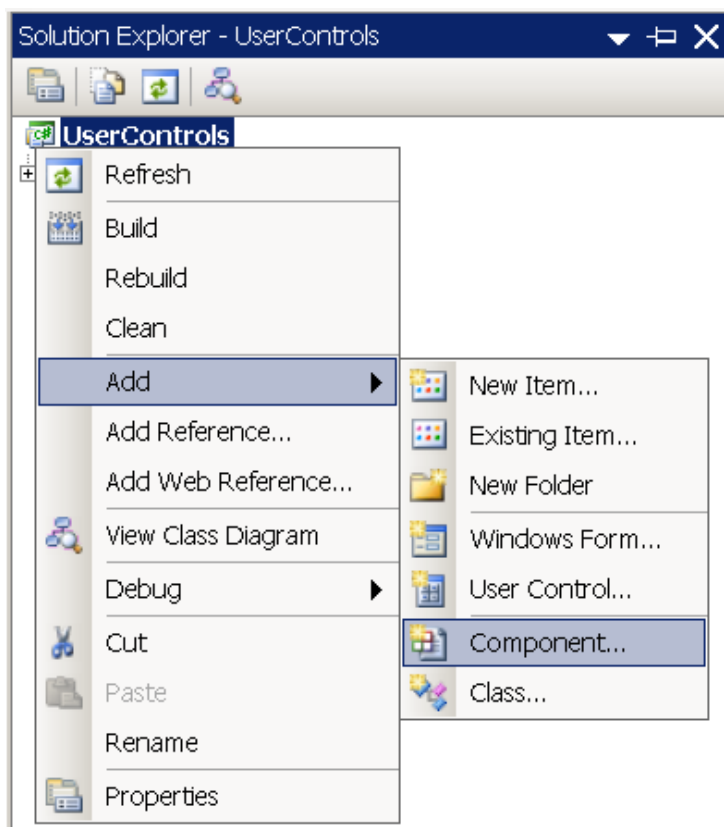
В панели Solution Explorer выделите корневой узел проекта UserControls и через контекстное меню выполните команду Properties (Рис. 19).

**Рис. 19**

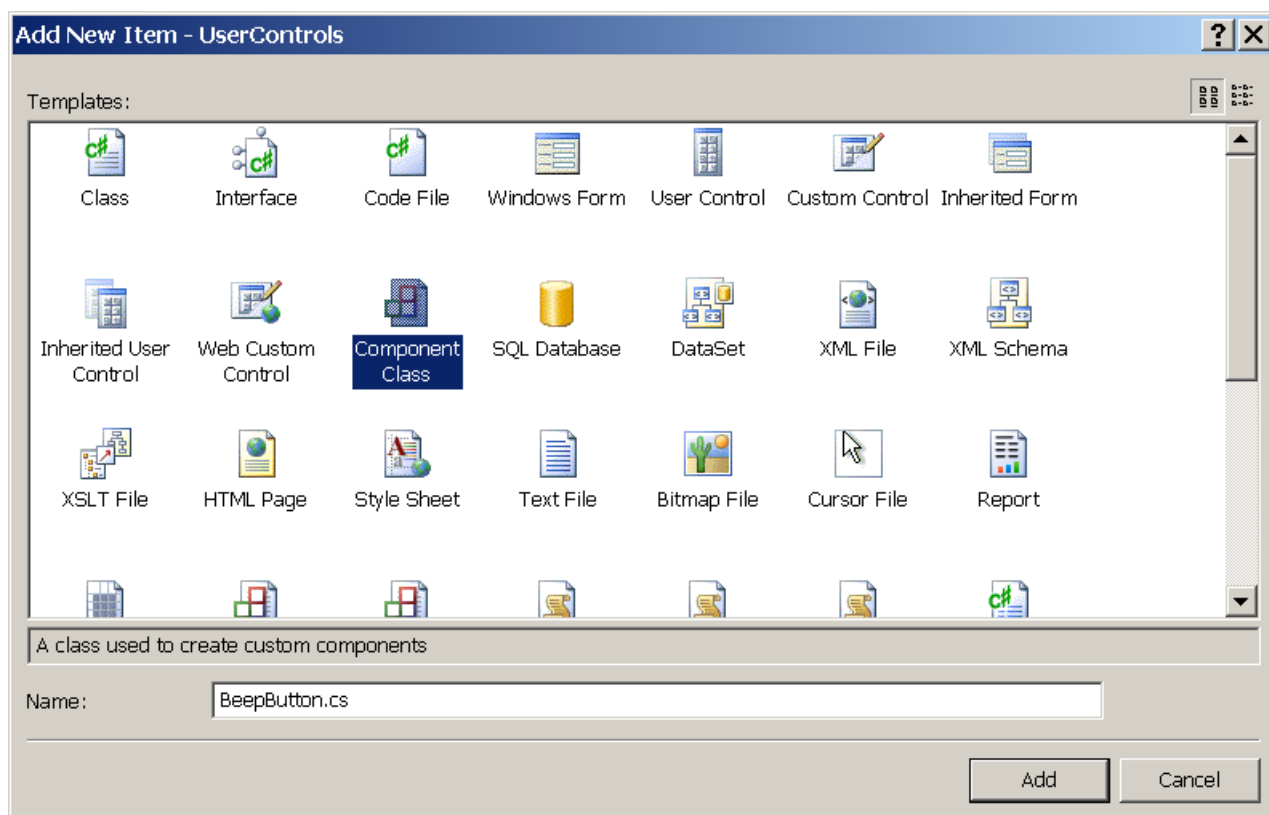
Откройте вкладку Application свойств оболочки и настройте поля *Default namespace* и *Output type* как показано на Рис. 20. После настройки закройте окно.

**Рис. 20**

В панели Solution Explorer вызовите для корневого узла проекта UserControls контекстное меню и выполните команду Add/Component (Рис. 21).

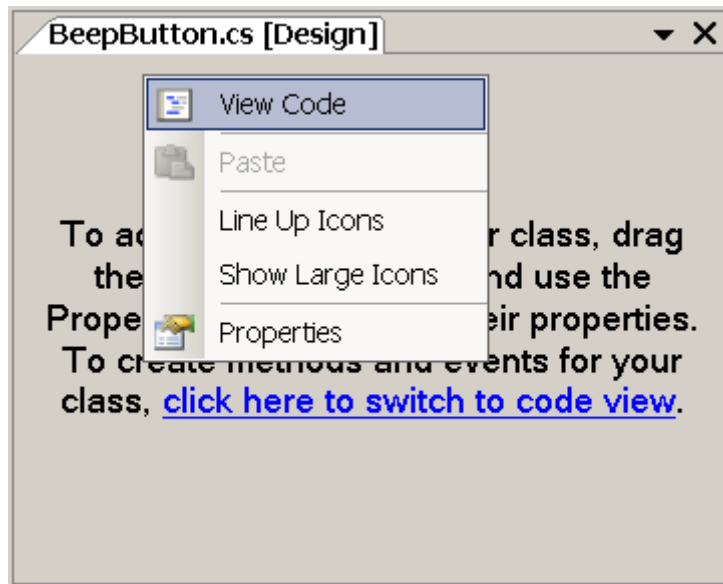
**Рис. 21**

Добавьте к проекту новый файл компонента с именем BeepButton.cs (Рис. 22).

**Рис. 22**

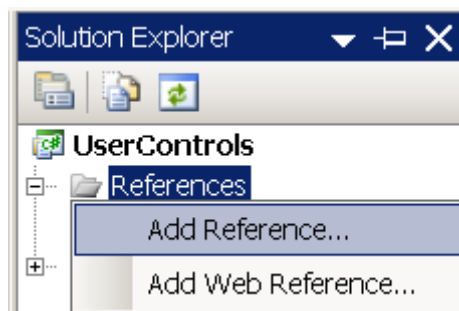


В рабочей области конструктора форм щелкните на ссылке [click here to switch to code view](#) или исполните команду контекстного меню View Code, чтобы перейти в режим редактирования кода (Рис. 23).



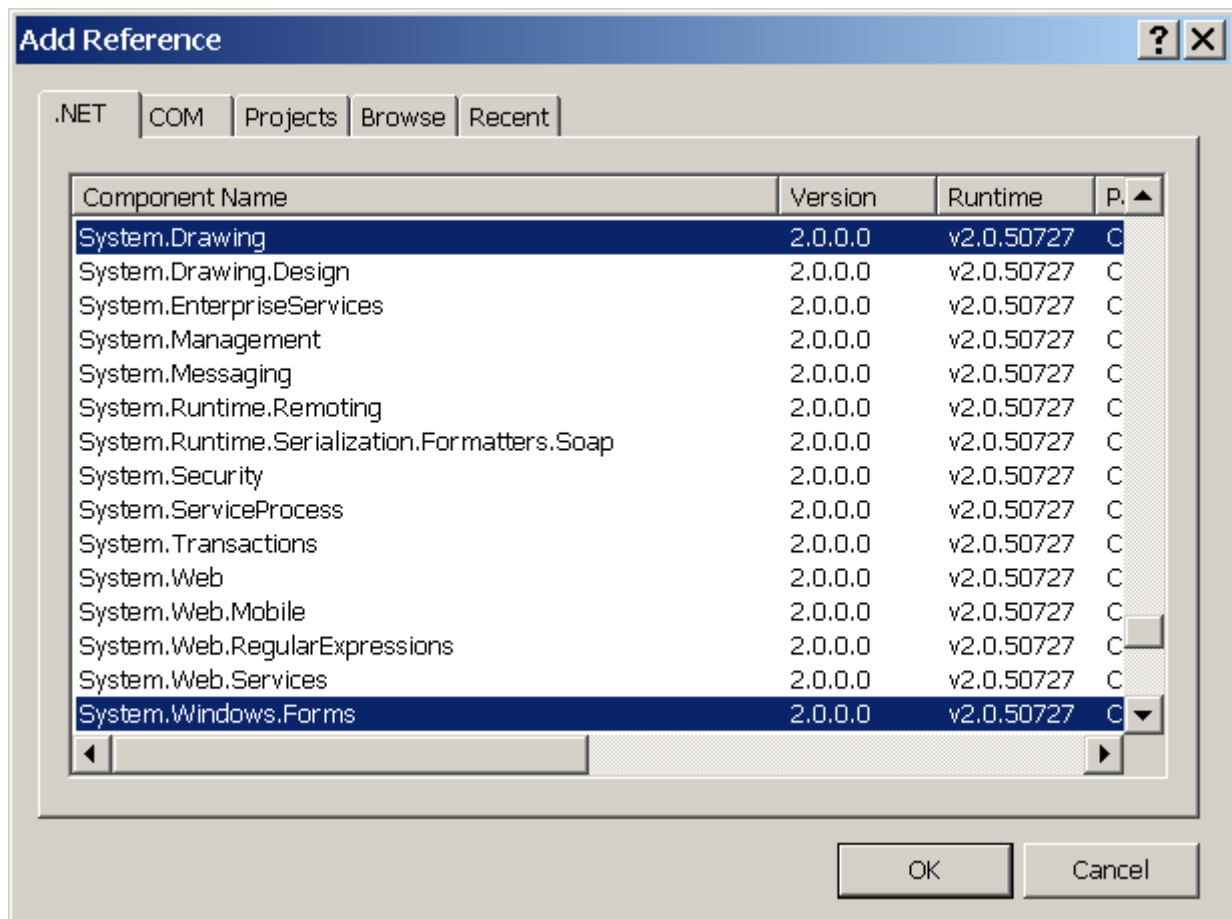
**Рис. 23**

В панели Solution Explorer для узла References, где уже должна находиться ссылка на библиотечную сборку System, выполните команду Add Reference (Рис. 24).

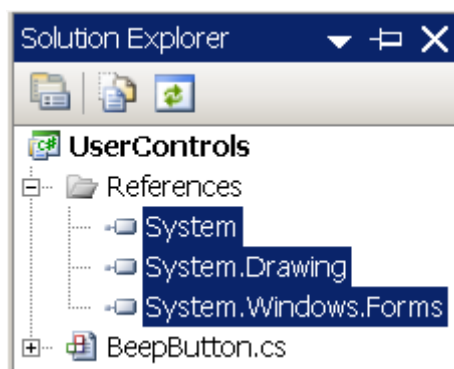


**Рис. 24**

**Удерживая клавишу Ctrl**, через окно Add Reference выделите и добавьте к проекту ссылки на библиотечные сборки System.Drawing и System.Windows.Forms (Рис. 25).

**Рис. 25**

Убедитесь, что теперь узел References проекта стал выглядеть так:

**Рис. 26**

Перейдите в режим редактирования файла BeepButton.cs и откорректируйте его следующим образом:

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

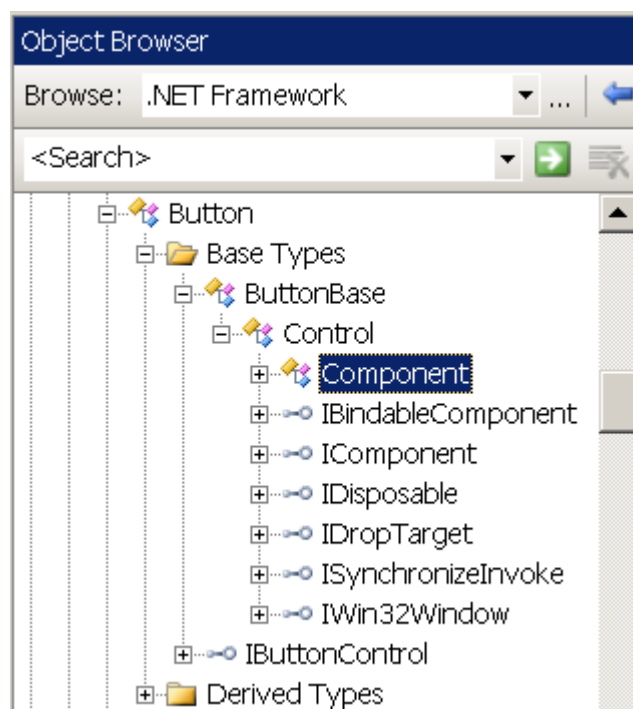
using System.Media; // Для звука
```

```
namespace MyCompany.UserControls
{
    public partial class BeepButton : Button
    {
        public BeepButton()
        {
            InitializeComponent();
        }

        public BeepButton(IContainer container)
        {
            container.Add(this);

            InitializeComponent();
        }
    }
}
```

Класс Button является наследником класса Component, поэтому мы применяем его в качестве базового, продолжая цепочку наследования.



**Рис. 27**

### **Переопределение наследуемого от Control виртуального метода диспетчеризации события Click**

Большинство библиотечных классов, в которых определены события, имеют соответствующие одноименные виртуальные методы диспетчеризации этих событий с приставкой On. Эти методы можно переопределять в классах-наследниках, дополняя нужным кодом. Когда среда исполнения возбуждает событие для какого-то элемента управления,

вызывается метод диспетчеризации, который адресуется в направлении от объекта формы к этому элементу по цепочки наследования в соответствии с таблицей виртуальных функций.

Над библиотечным элементом управления всегда можно надстроить класс-потомок, в котором можно перехватить этот виртуальный метод и обработать нужным образом. При этом, если в переопределенном методе не продолжить вызов виртуального метода диспетчеризации события в сторону базового класса, то ни один подписавшийся на это событие обработчик не сработает.

Дополните конец файла `BeepButton.cs` кодом переопределения унаследованного метода `OnClick()`, начав ввод с ключевого слова `override`:

```
namespace MyCompany.UserControls
{
    partial class BeepButton
    {
        // Переопределяем виртуальный метод диспетчеризации
        // класса Control (задействуется наследуемый метод
        // диспетчеризации класса кнопки)
        protected override void OnClick(EventArgs e)
        {
            // Проиграть системный звук
            SystemSounds.Exclamation.Play(); // Сочный через внешние
динамики
            //SystemSounds.Beep.Play();      // Простой через внешние
динамики
            //System.Console.Beep();          // Простой через внутренний
динамик

            // Подняться до базового класса Control,
            // чтобы сгенерировать его событие Click и
            // запустить наш будущий обработчик ButtonOnClick
            base.OnClick(e);
        }
    }
}
```

В переопределенном методе диспетчеризации события `Click` кнопки мы предусмотрели вызов базового метода диспетчеризации непосредственно кнопки `Button`, чтобы дать возможность клиентам класса `BeepButton` подписать свои обработчики на это событие. Событие `Click`, в свою очередь, унаследовано классом `Button` от класса `Control` и генерируется его методом `OnClick` примерно так:

```
protected virtual void OnClick(EventArgs args)
{
    //...
    // Если хоть один обработчик зарегистрирован
    if (Click != null)
        Click(this, args);
    //...
}
```

Испытание кнопки BeepButton мы проведем позднее, разработав для этого тестовую исполнимую Windows-сборку.

Для проверки правильности написанного кода выполните команду Build Solution. Далее добавьте к проекту UserControl1 Windows-форму с именем Check. Перейдите в режим Designer и проверьте наличие в панели Toolbox разработанного вами компонента BeepButton.

Далее лабораторная работа выполняется по вариантам (Вариант 1. Проектирование диалогового окна сообщений MyDialogBox/Вариант 2. Рисование круглой кнопки).

## Вариант 1. Проектирование диалогового окна сообщений MyDialogBox

Для выдачи пользователю простых сообщений существует стандартное диалоговое окно, поддерживаемое классом System.Windows.Forms.MessageBox. Но в целях тренировки в данном разделе мы спроектируем собственное простое диалоговое окно сообщений с именем MyDialogBox, которое будет вызываться из обработчика события Click нашей кнопки, если на него подпишется клиент компонента BeepButton.

Создание диалогового окна будет состоять из трех шагов:

1. Создание формы диалогового окна (класс MyDialogForm);
2. Размещение на форме интерфейсных элементов пользователя;
3. Создание класса MyDialogBox.cs, управляющего диалоговым окном;

В панели Solution Explorer выделите узел проекта UserControl1 и командой Add/Windows Form добавьте к компоненту файл формы с именем MyDialogForm.cs (Рис. 28).

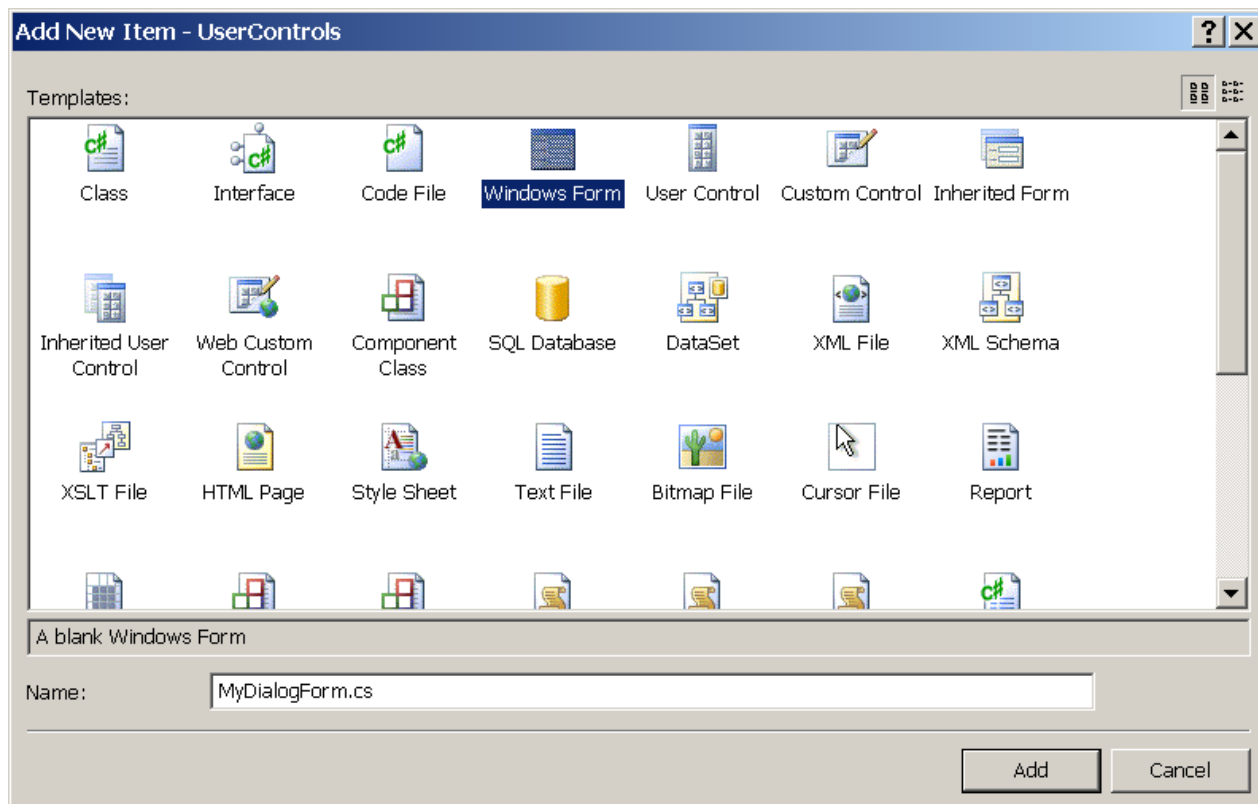
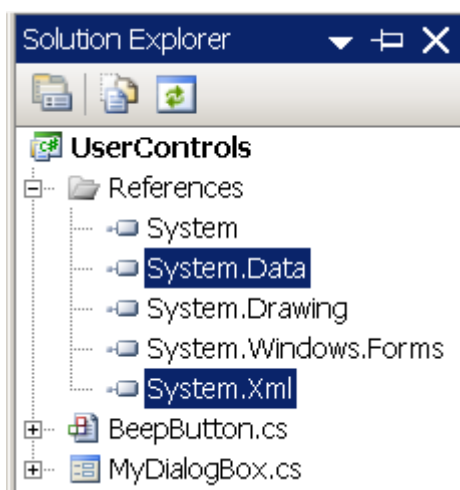


Рис. 28

Удалите из узла References лишние добавленные мастером ссылки на библиотечные сборки System.Data и System.Xml.



**Рис. 29**

Удалите соответствующие инструкции using подключения пространств имен в начале файла MyDialogForm.cs (аналогично л/р №1).

Настройте форму MyDialogForm в соответствии с Таблица 11.

**Таблица 11 Таблица свойств формы MyDialogForm**

Свойство	Значение	Пояснения
StartPosition	CenterScreen	Открывать в центре экрана
FormBorderStyle	FixedDialog	Запретить пользователю менять размер
MaximizeBox	false	Отключить системную кнопку развертывания
MinimizeBox	false	Отключить системную кнопку свертывания
ShowInTaskbar	false	Запретить появление в панели задач
AutoSize	true	Подстраиваться под содержимое
AutoSizeMode	GrowAndShrink	Подстраиваться под содержимое (расширяться и сжиматься)
Text	Пусто	Заголовок окна

Из свитка Containers панели инструментов Toolbox поместите на форму компоновочную панель FlowLayoutPanel и настройте ее в соответствии с Таблица 12.

**Таблица 12 Таблица свойств компоновочной панели FlowLayoutPanel**

Свойство	Значение	Пояснения
Name	flow	Имя компоновочной панели
AutoSize	true	Подстраиваться под содержимое
FlowDirection	TopDown	Размещать сверху вниз

Свойство	Значение	Пояснения
WrapContents	false	Не переносить дочерние элементы в следующий столбец
Padding.All	8	Внутренний отступ дочерних элементов по периметру окна
Dock	Fill	Распахнуть на всю клиентскую область родителя (формы)

Эта панель будет следить за размещением своих дочерних элементов в соответствии с настройками.

Поместите внутрь экземпляра компоновочной панели flow из свитка Common Controls текстовую метку Label и настройте ее свойства в соответствии с Таблица 13.

**Таблица 13 Таблица свойств текстовой метки Label**

Свойство	Значение	Пояснения
Name	lblMessage	Имя текстовой метки
AutoSize	true	Подстраиваться под содержимое
Anchor	None	Расположить посередине
Margin.All	8	Отступ по наружному периметру текстовой метки
Modifiers	Public	Публичный доступ для клиентов формы (для класса MyDialogBox)

Для того, чтобы объект lblMessage был доступен из метода класса MyDialogBox, при его создании мы сделали его общедоступным.

Поместите внутрь компоновочной панели flow из свитка Common Controls кнопку Button и настройте ее свойства в соответствии с Таблица 14.

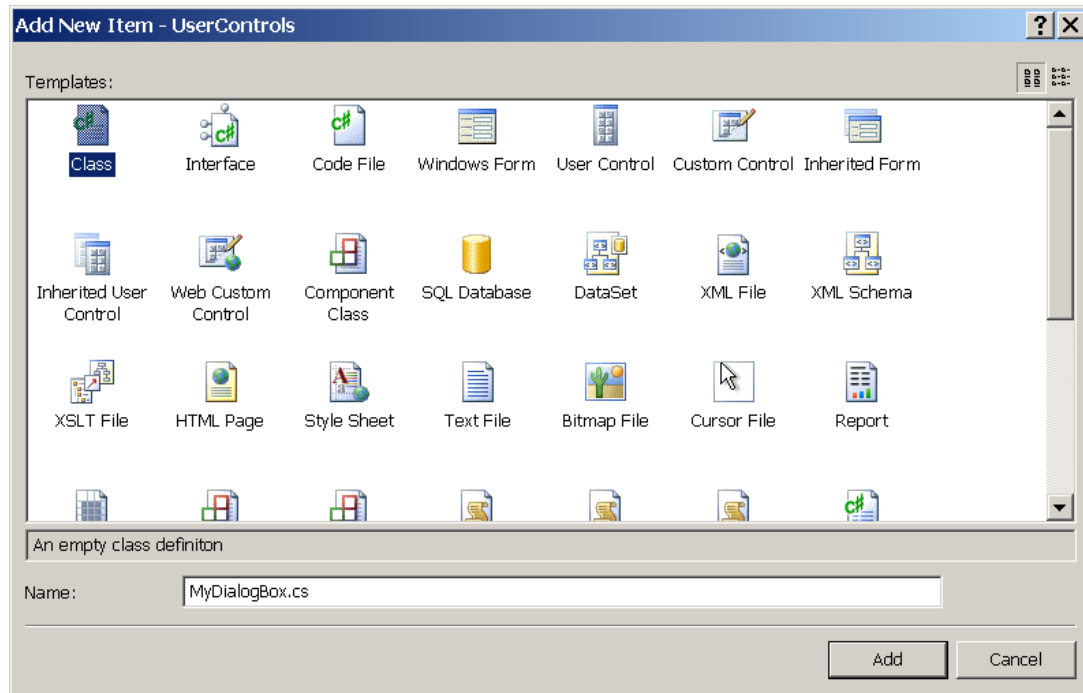
**Таблица 14 Таблица свойств кнопки Button**

Свойство	Значение	Пояснения
Name	btnShowDialog	Имя кнопки
AutoSize	true	Подстраиваться под содержимое
Anchor	None	Расположить посередине
Margin.All	8	Отступ по наружному периметру текстовой метки
Text	OK	Надпись
DialogResult	OK	Назначить статус "кнопки OK" в родительской форме

Свойству дочерних кнопок формы DialogResult можно присвоить любое значение из перечисления System.Windows.Forms.DialogResult (Abort, Cancel, Ignore, No, None, OK, Retry, Yes). Оно и будет возвращено функцией ShowDialog() формы, если пользователь выберет именно эту кнопку.

Мы создали интерфейсный класс диалогового окна. Теперь нужно создать управляющий класс, в котором мы будем создавать экземпляры интерфейсного класса диалогового окна и вызывать его на экран.

В панели Solution Explorer выделите узел проекта UserControls и, выполнив команду меню Project/Add Class, добавьте к проекту новый класс с именем MyDialogBox (Рис. 30).



**Рис. 30**

Заполните файл MyDialogBox.cs следующим кодом:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;

namespace MyCompany.UserControls
{
    // Наше диалоговое окно сообщения
    public class MyDialogBox
    {
        public static DialogResult Show(string strMessage, string
strCaption)
        {
            // Создаем диалоговое окно
            MyDialogForm frm = new MyDialogForm();
            // Заголовок окна
            frm.Text = strCaption;
            // Содержимое текстовой метки
            frm.lblMessage.Text = strMessage;
```



```

        // Отобразить диалоговое окно в модальном режиме,
        // а после его закрытия вернуть выбор пользователя
        // вызывающему коду
        return frm.ShowDialog(); // Выполняем диалоговое окно
                                // и возвращаем статус нажатой кнопки
    }
}
}

```

Класс MyDialogBox нужно объявить с модификатором public, чтобы он был виден в клиентских сборках. Объявление метода Show() в классе MyDialogBox как статического позволяет его вызывать без создания экземпляра этого класса в клиенте. Через аргументы метода Show() можно задавать заголовок диалогового окна и содержимое текстовой метки lblMessage, как это принято в библиотечном классе System.Windows.Forms.MessageBox.

Теперь осталось собрать все вместе и испытать на тестовом приложении, но прежде нужно откомпилировать проект UserControl1 и убедиться, что нет синтаксических ошибок.

Выполните команду меню оболочки Build/Build UserControl1, чтобы откомпилировать проект нашей библиотеки пользовательских элементов управления.

## Тестирование пользовательских элементов BeepButton и MyDialogBox

Добавьте к решению MySolution новый проект с именем Test типа Windows Application. Для этого выполните команду меню оболочки File/Add/New Project и заполните окно мастера следующим образом:

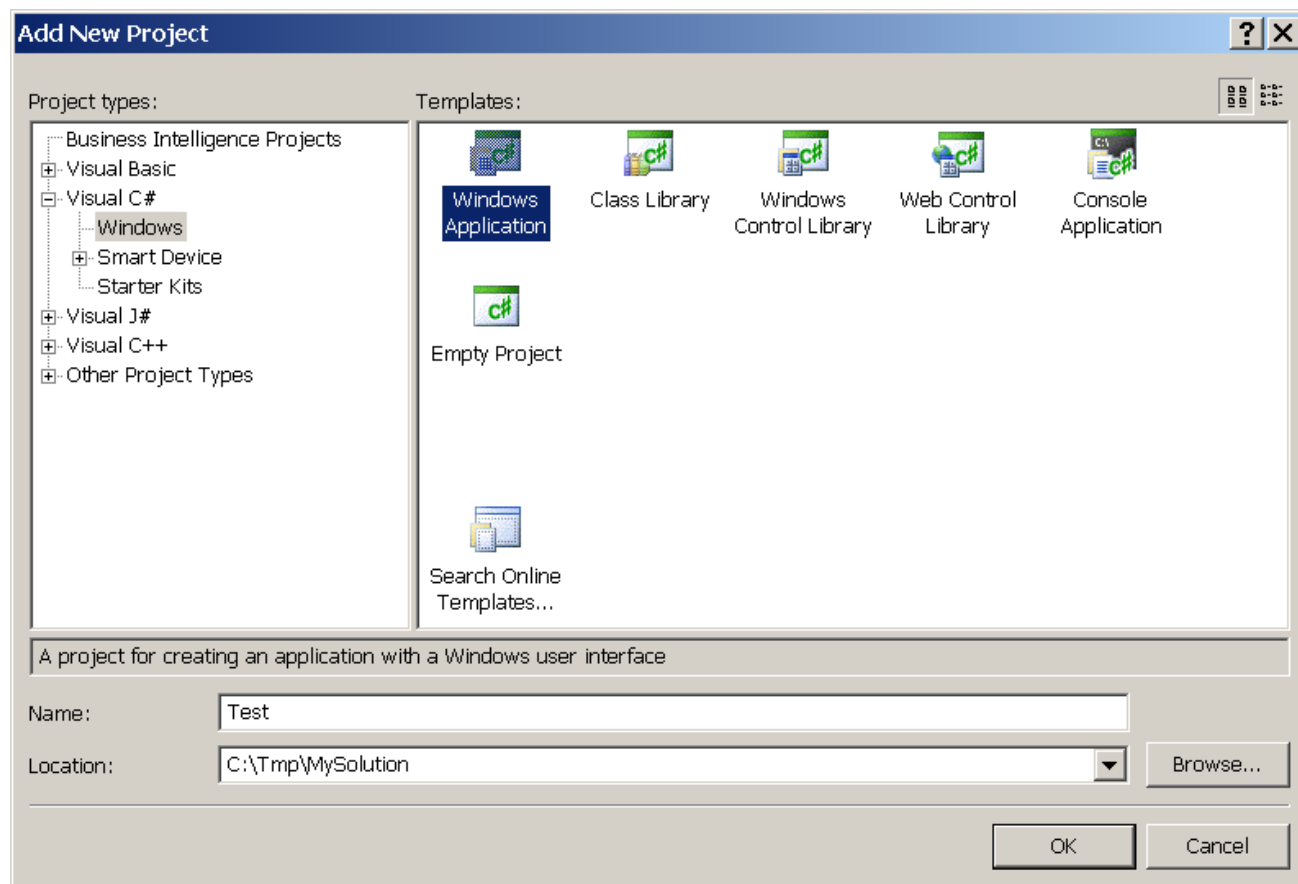


Рис. 31

В панели Solution Explorer выделите узел проекта Test и назначьте его стартовым командой Project/Set as StartUp Project меню оболочки или аналогичной командой контекстного меню.

Выделите форму Form1 и настройте ее свойства как показано в Таблица 15.

**Таблица 15 Таблица свойств формы тестового приложения**

Свойство	Значение	Пояснения
Text	Кнопка со звуком	Заголовок тестового окна
StartPosition	CenterScreen	Открывать в центре экрана

Поместите из панели Toolbox на верхнюю часть формы экземпляр textBox1 текстового поля TextBox.

Поместите из панели Toolbox на середину формы разработанную нами кнопку BeepButton и настройте ее в соответствии с Таблица 16. Если разработанный компонент не появился в панели Toolbox, то добавьте его самостоятельно с помощью команды Choose Items, вызванной с помощью контекстного меню. В открывшемся окне необходимо выбрать вкладку .NET Framework Components и с помощью кнопки Browse подключить dll-библиотеку из папки проекта.

**Таблица 16 Таблица свойств кнопки BeepButton**

Свойство	Значение	Пояснения
AutoSize	true	Подстраиваться под содержимое
Text	Звук	Надпись
Location	100; 100	Запозиционировали

Для выделенного экземпляра кнопки BeepButton переведите панель Properties оболочки в режим Events и в поле события Click задайте имя обработчика ButtonOnClick.

Заполните обработчик кодом вызова нашего диалогового окна MyDialogBox так:

```
private void ButtonOnClick(object sender, EventArgs e)
{
    string msg;

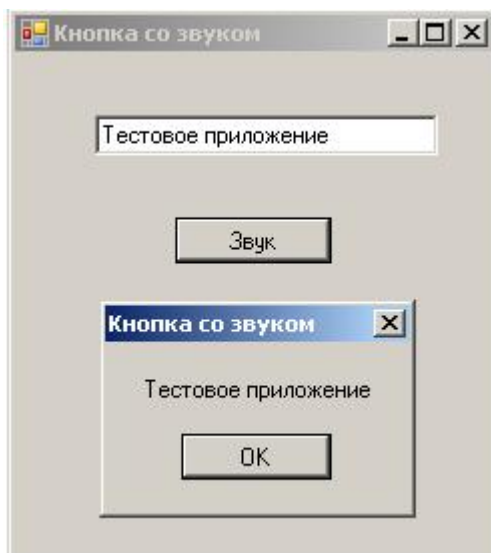
    if (textBox1.Text.Trim().Length == 0)
        msg = "Щелкнули на кнопке";
    else
        msg = textBox1.Text.Trim();

    // Запускаем наше диалоговое окно статическим методом
    MyDialogBox.Show(msg, this.Text);
}
```

Добавьте в начало файла Form1.cs строку кода подключения пространства имен нашей библиотеки элементов для того, чтобы иметь возможность написать сокращенное имя класса MyDialogBox:

```
using MyCompany.UserControls;
```

Выполните приложение и получите примерно такой результат:



**Рис. 32**

В производном от кнопки классе BeepButton мы только переопределили виртуальный метод OnClick(), в который встроили вызов звукового сигнала. В вызывающем коде тестового приложения, использующем экземпляр класса расширения кнопки, мы подписались на событие Click, чтобы щелчок на кнопке запускал самодельное диалоговое окно.

Но мы этот метод перехватили в нашем расширении, и если не предусмотреть передачу вызова base.OnClick(e) дальше по цепочке наследования в класс Control, то событие Click инициировано не будет и наш зарегистрированный обработчик ButtonOnClick() никогда не сработает. А значит и наше диалоговое окно вызываться не будет (попробуйте закомментировать вызов base.OnClick(e) в файле BeepButton.cs).

Таким образом, в переопределенном методе OnClick(), который срабатывает автоматически при щелчке на кнопке, вызывается звуковой сигнал. А при вызове этого метода в базовом классе генерируется событие Click, которое заставляет выполняться все зарегистрированные для него обработчики. В нашем случае это один обработчик, косвенно запускающий самодельное диалоговое окно. Метод OnClick() для кнопок вызывается и в тех случаях, когда кнопка имеет текущий фокус ввода и нажимается клавиша Enter или Space (пробел).

Если в переопределенном методе OnClick() класса BeepButton поменять местами строки вызова звука и события Click, то звук появится только после закрытия диалогового окна (попробуйте).

Обратите внимание, что при формировании интерфейса диалогового окна мы не расставляли по форме кнопку и текстовую метку. Но за счет применения библиотечного компонента FlowLayoutPanel эти элементы на этапе выполнения позиционируются автоматически.

## Вариант 2. Рисование круглой кнопки

В этом упражнении мы создадим кнопку непрямоугольной формы, которая будет отрисовываться по пользовательскому алгоритму. На этом примере будет показано, как создавать пользовательские элементы управления непрямоугольной формы.

Вид стандартного элемента управления можно изменить, переопределив его виртуальные методы `OnPaint()` и `OnResize()`. Предпочтительные размеры элемента управления устанавливаются в переопределении наследуемого метода `GetPreferredSize()`. Для прорисовки элементов управления можно использовать класс `System.Windows.Forms.ControlPaint`. Системная палитра, перья и кисти находятся в трех классах пространства имен `System.Drawing`:

- `SystemColors`;
- `SystemPens`;
- `SystemBrushes`;

Пространство имен `System.Windows.Forms` также содержит цвета в классах:

- `ProfessionalColors` - цвета в статических свойствах класса;
- `ProfessionalColorTable` - цвета в виртуальных экземплярных свойствах класса;

### Создание заготовки компонента `CircleButton`

В панели `Solution Explorer` выделите узел `UserControls` и добавьте к проекту библиотеки новый компонент командой `Project/Add Component` с именем `CircleButton`.

Добавьте в начало файла `CircleButton.cs` инструкции подключения пространств имен `using System.Drawing`; `using System.Windows.Forms`.

В заголовке класса `CircleButton` файла `CircleButton.cs` замените базовый класс `Component` на `Button`.

Добавьте в класс `CircleButton` закрытый метод `Init()` и поместите его вызовы в перегруженные конструкторы компонента.

```
using System;
using System.ComponentModel;
using System.Collections.Generic;
using System.Diagnostics;

using System.Drawing;
using System.Windows.Forms;

namespace MyCompany.UserControls
{
    public partial class CircleButton : Button
    {
        public CircleButton()
        {
            InitializeComponent();

            Init();
        }
    }
}
```

```
public CircleButton(IContainer container)
{
    container.Add(this);

    InitializeComponent();

    Init();
}

// Начальные настройки для выполнения в конструкторе
private void Init()
{
    // Извещаем операционную систему, что будем сами
    // заботиться об отрисовке компонента
    this.SetStyle(ControlStyles.UserPaint, true);

    // Игнорировать системное сообщение WM_ERASEBKGD
    // очистки окна, чтобы уменьшить нежелательное мерцание
    this.SetStyle(ControlStyles.AllPaintingInWmPaint, true);

    // Наружный отступ по контуру кнопки
    this.Margin = new Padding(this.Font.Height);

    // Подстраиваться под размер надписи
    this.AutoSize = true;
}
}
```

Метод `GetPreferredSize()` вызывается диспетчером размещения для определения предпочтительных размеров компонента, когда значение его свойства `AutoSize=true` (мы его установили в методе `Init()`) и это компонент готов к перерисовке в методе `OnPaint()`. Наш код при начальной загрузке компонента выдаст размеры квадрата и эллипсная кнопка будет круглой

### Задание круглой формы для кнопки

Все визуальные элементы управления наследуют от класса `System.Windows.Forms.Control` вместе с его свойством `Region` - экземпляром класса `System.Drawing.Region`. Свойство `Region` определяет форму элемента управления и по умолчанию ссылается на прямоугольник. Но эту форму можно изменить на любую сложную фигуру, передав контур этой фигуры в конструктор класса `Region`. После назначения свойства `Region` у элемента управления по-прежнему остаются размеры прямоугольника, но все, что выходит за пределы указанного контура, будет невидимо как визуально, так и для мыши.

Графический контур формируется как экземпляр класса `System.Drawing.Drawing2D.GraphicsPath`, который содержит множество методов для создания нужных графических примитивов. Если текущая фигура состоит из отрезков линий, то она считается законченной после выполнения метода `CloseFigure()`, который замыкает

начальную и конечную точки рисования. Когда рисуется последовательность замкнутых фигур, то каждая новая фигура проецируется на предыдущие, как вырезание.

Подключите в начале файла CircleButton.cs с помощью инструкции using пространство имен System.Drawing.Drawing2D для сокращенной идентификации класса GraphicsPath.

Переопределите в классе CircleButton унаследованный виртуальный метод OnResize(), начав ввод с ключевого слова override.

```
// Вычисление контура кнопки при изменении ее размеров
protected override void OnResize(EventArgs args)
{
    // Объект для размещения контура
    GraphicsPath path = new GraphicsPath();

    // Размещаем эллипс
    path.AddEllipse(this.ClientRectangle);

    // Создаем объект рисования и адресуем его кнопке
    this.Region = new Region(path);

    // Вызываем предка для инициации OnPaint()
    // или Invalidate()
    base.OnResize(args);
    // this.Invalidate();
}
```

Переопределите виртуальный метод OnPaint() для отрисовки кнопки следующим образом:

```
// Отрисовка круглой кнопки
protected override void OnPaint(PaintEventArgs args)
{
    // Адресовались к контексту графического устройства
    Graphics gr = args.Graphics;
    // Настраиваем качество рисования – сглаживать контуры
    gr.SmoothingMode = SmoothingMode.AntiAlias;

    // Флаг попадания курсора внутрь кнопки при ее нажатии
    bool bPressed =
        this.Capture & // Связь с мышью есть!
        ((Control.MouseButtons & MouseButtons.Left) != 0) & //
Нажата левая кнопка мыши
        // Нажатый курсор попал внутрь описывающего
прямоугольника
        this.ClientRectangle.
Contains(this.PointToClient(Control.MousePosition));

    Rectangle rect = this.ClientRectangle; // Псевдоним
```

```

// Отрисовка внутренней части кнопки
// При нажатии она становится темнее
GraphicsPath path = new GraphicsPath();// Заготовили контур
для заливки
path.AddEllipse(rect); // Вписали эллипс в клиентскую область
кнопки
PathGradientBrush gradient = new PathGradientBrush(path);//
Создали градиент для заливки
int k = bPressed ? 2 : 1;// Для нажатой кнопки сдвинем центр
градиента вправо и вниз
// Располагаем центр радиального градиента левее и выше
центра кнопки
gradient.CenterPoint = new PointF(
    k * (rect.Left + rect.Right) / 3,
    k * (rect.Top + rect.Bottom) / 3);
// Настраиваем цвет радиального градиента
gradient.CenterColor = bPressed ? Color.PowderBlue :
Color.White;// Центр
gradient.SurroundColors = new Color[] { Color.SkyBlue };//
Окраина
gr.FillRectangle(gradient, rect); // Залить градиентом

// Отрисовка контура кнопки, утолщенного для активных кнопок
// Создаем кисть с линейным градиентом, изменяющимся по
диагонали
Brush brush = new LinearGradientBrush(
    rect,
    Color.FromArgb(0, 0, 255),
    Color.FromArgb(0, 0, 128),
    LinearGradientMode.ForwardDiagonal);
// Создаем перо для рисования градиентной кистью
Pen pen = new Pen(
    brush,
    (this.IsDefault ? 3 : 1) * gr.DpiX / 72);// Если кнопка
имеет фокус ввода - контур толще
gr.DrawEllipse(pen, rect); // Нарисовать контур

// Отображение надписи в центре прямоугольника кнопки
// Для недоступной кнопки цвет надписи должен быть бледно-
серым
StringFormat strFormat = new StringFormat();
// Определение точки привязки в центре текстового блока
strFormat.Alignment =
    strFormat.LineAlignment =
    StringAlignment.Center;
brush = this.Enabled ? SystemBrushes.WindowText // Для
активной
    : SystemBrushes.GrayText;// Для неактивной
gr.DrawString(this.Text, this.Font, brush, rect, strFormat);

```

```
// Отображение пунктира вокруг текста, если кнопка имеет
фокус
if (this.Focused)
{
    SizeF sizeText = gr.MeasureString(
        this.Text,
        this.Font,
        PointF.Empty,
        StringFormat.GenericTypographic);
    pen = new Pen(this.ForeColor); // Текущий цвет
    pen.DashStyle = DashStyle.Dash; // Пунктир
    gr.DrawRectangle(
        pen,
        rect.Left + rect.Width / 2 - sizeText.Width / 2,
        rect.Top + rect.Height / 2 - sizeText.Height / 2,
        sizeText.Width,
        sizeText.Height);
}

// Метод диспетчеризации своего события - для тренировки
if (this.Paint != null) // Если есть хоть одна подписка
    this.Paint(this, args); // Иницилируем событие
}

new public event PaintEventHandler Paint; // Объявляем свое
событие в классе
```

Поскольку в классе расширения кнопки мы переопределили виртуальный метод OnPaint() и не предусмотрели его вызов для базового класса, то событие Paint никогда инициировано не будет. Потому что оно вызывается в методе OnPaint() именно базового класса, который мы перехватили. Чтобы не потерять событие Paint, мы его заново объявили в классе расширения, а в переопределенный метод OnPaint() вставили инициализацию этого события с предварительной проверкой, подписано ли оно в вызывающем коде.

В панели Solution Explorer вызовите контекстное меню для узла UserControls библиотечных компонентов и выполните команду Build.

Компонент круглой кнопки мы сделали, теперь нужно его протестировать.

### Тестирование компонента круглой кнопки

Добавьте к решению командой File/Add/New Project новый проект типа оконного приложения (Windows Application) с именем CircleButtonTest и назначьте его стартовым командой Project/Set as StartUp Project меню оболочки (или аналогичной командой контекстного меню).

Поместите на форму Form1 из панели Toolbox несколько экземпляров компонента CircleButton. Если компонент CircleButton не появился на панели, добавьте его в ручную.

Перейдите в файле Form1.cs в режим View Code и добавьте в начало инструкцию:

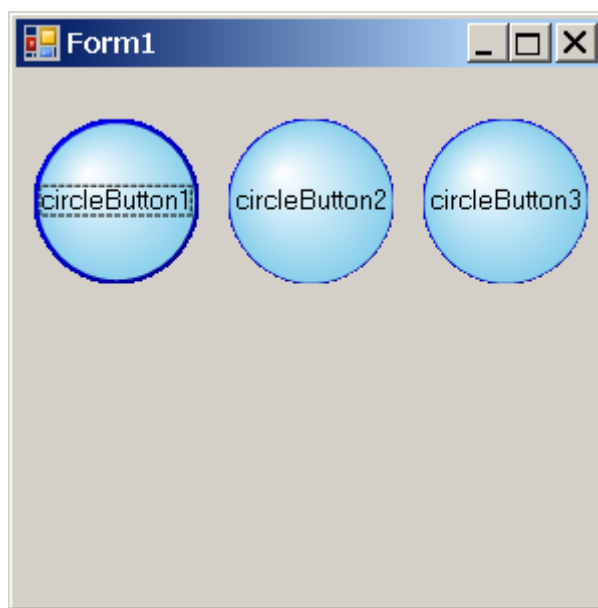
```
using MyCompany.UserControls;
```



Настройте свойства компонентов через панель Properties и создайте для всех кнопок общий обработчик `circleButton_Click`.

```
private void circleButton_Click(object sender, EventArgs e)
{
    // Создаем псевдоним переданной ссылки
    CircleButton button = sender as CircleButton;
    if (button != null)
        MessageBox.Show("Нажата кнопка: " + button.Name);
}
```

Запустите приложение и испытайте работу компонентов.  
Результат будет примерно таким, как показано на Рис. 33.



**Рис. 33**

## Индивидуальные задания

### Вариант 1:

1. Доработайте приложение таким образом, чтобы при нажатии на клавишу ОК появлялось новое диалоговое окно с текстовой панелью, в которой содержались бы данные об авторе программы.
2. Измените код программы так, чтобы при нажатии на клавишу ОК приложение завершало бы свою работу предупреждая об завершении через MyDialogBox.
3. Добавьте на форму еще 3 звуковые клавиши ассоциируйте с ними любые 3 системные мелодии.
4. Измени код программы таким образом, чтобы название приложения менялось при нажатии «Кнопка со звуком» и отображался месторасположение используемого системного звука.
5. Доработайте приложение таким образом, чтобы при нажатии на клавишу «Звук» программа воспроизводила выбранный вами музыкальный файл.
6. Добавьте на основную форму программы счетчик использования звуковой клавиши и реализуйте его так, чтобы после третьего использования приложение завершало свою работу, при этом на втором выводило сообщение что при следующем нажатии программа завершиться.
7. Добавьте на основную форму программы счетчик использования звуковой клавиши и реализуйте его так, чтобы после третьего использования приложение завершало свою работу.
8. Измените код программы так, чтобы появляющееся диалоговое окно MyDialogBox занимало весь экран и выводило название воспроизводимого звука.
9. Задание повышенной сложности (по желанию): Написать код программы так, чтобы воспроизводились mp3 файлы из плейлиста, загрузка в плейлист производилась по выбору файлов.

### Вариант 2:

1. Доработайте код программы так, чтобы нажатии на любой кнопке на форме программа выдавала бы сообщение какой клавишей мыши был произведен щелчок и по какой кнопке.
2. Добавьте на панель на панель три кнопки. При нажатии третьей добавленной кнопки загорался контур первой клавиши CircleButton, при нажатии второй добавленной кнопки загорался контур второй клавиши CircleButton, при нажатии третей добавленной кнопки загорался контур третей клавиши CircleButton.
3. Добавьте на панель еще 1 кнопку, что бы при нажатии на нее менялась заливка всех остальных кнопок в случайный цвет.
4. Измените код программы так, чтобы при нажатии на любую кнопку появлялось диалоговое окно MyDialogBox со случайным числом.
5. Добавьте на форму еще две пары по три клавиши CircleButton, а в обработчике создайте счетчик нажатия на каждую клавишу.
6. Доработайте программу так, чтобы сообщение о нажатой клавише появлялось не в отдельном окне, а на самой форме.
7. Измените код программы так, чтобы у активной кнопки контур оставался по умолчанию, а у неактивных кнопок он становился бы утолщенным.

8. Доработайте код программы так, чтобы при нажатии по каждой кнопке на их изображении оставался только один из каналов (R, G, B). Канал выбирается кнопкой.
9. Измените код программы так, чтобы при нажатии на клавишу ее текст менял бы на произвольный цвет.
10. Добавьте на форму две кнопки `CircleButton` красного цвета с градиентом, аналогичным синим кнопкам и реализуйте для них такую же функциональность.
11. Задание повышенной сложности (по желанию): Заменить в приложение повышенной сложности (задание 8 из варианта 1) кнопки на шестиугольники, добавив им любой градиентный цвет.

## Лабораторная Работа №3

### Разработка комбинированного компонента

Краткое описание: В данной лабораторной работе приведены упражнения по созданию собственных пользовательских компонентов. Добавление объектов всплывающей подсказки. Таблица цветов структуры Color. Разработка кнопки в стиле полос прокрутки. Тестирование скроллирующей кнопки ArrowButton. Комбинирование элементов в единый компонент NumericScan.

## Основная часть

Всегда пользовательский элемент управления лучше создавать на основе уже существующих, преимущественно библиотечных, элементов управления, которые хорошо отлажены и работают привычным образом. В предыдущих примерах мы усовершенствовали библиотечный класс `Button`, приспособивая его к различным задачам путем расширения в другом классе.

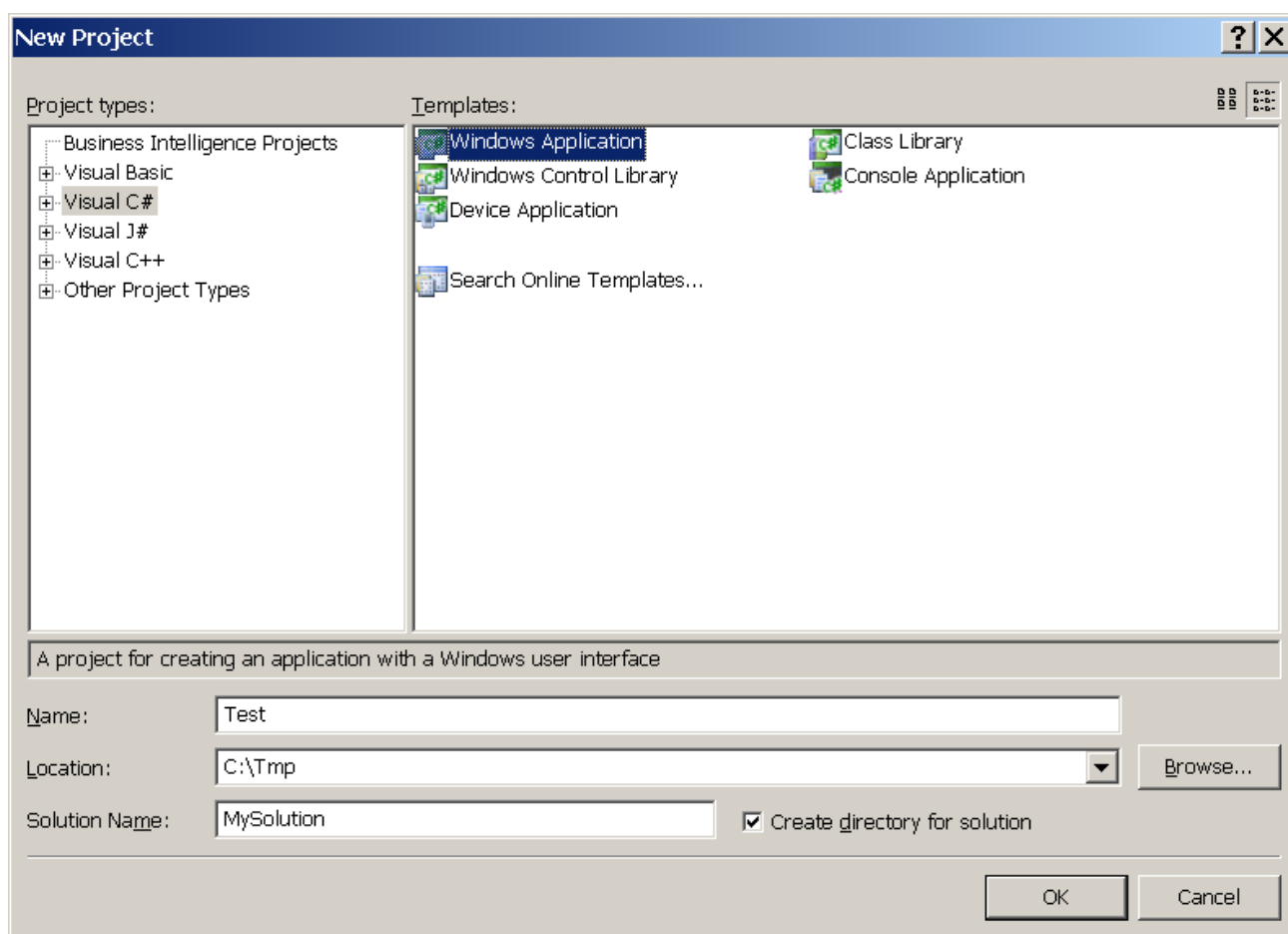
В данной работе мы продолжим выполнять упражнения по созданию собственных пользовательских компонентов, обладающих необходимыми нам свойствами. Мы создадим решение (`MySolution`), которое будет содержать проект DLL-библиотеки создаваемых компонентов (`UserControls`) и проект необходимых тестов (`Test`). Для удобства презентации выполненной работы создадим стартовую форму (`Start`) с кнопками-ссылками (`LinkLabel`), за каждой из которых закрепим вызов теста соответствующего упражнения.

В упражнениях мы последовательно разработаем свой (пользовательский) элемент управления на основе нескольких библиотечных элементов путем их комбинирования. Применим прием многоэтапного последовательного расширения классов на основе наследования. Вначале создадим из библиотечного класса `Button` класс расширения кнопки `ClickmaticButton` такой, чтобы в нажатом состоянии кнопка непрерывно генерировала событие `Click` с заданным интервалом. Затем вновь последовательно расширим класс `ClickmaticButton` классом `ArrowButton`, в котором на кнопке нарисуем стрелку. И наконец, скомбинируем два экземпляра созданной кнопки-стрелки и экземпляр библиотечного элемента `TextBox` в единый пользовательский компонент `NumericScan` по аналогии с библиотечным компонентом `NumericUpDown` - поле со счетчиком.

### Разработка стартовой формы

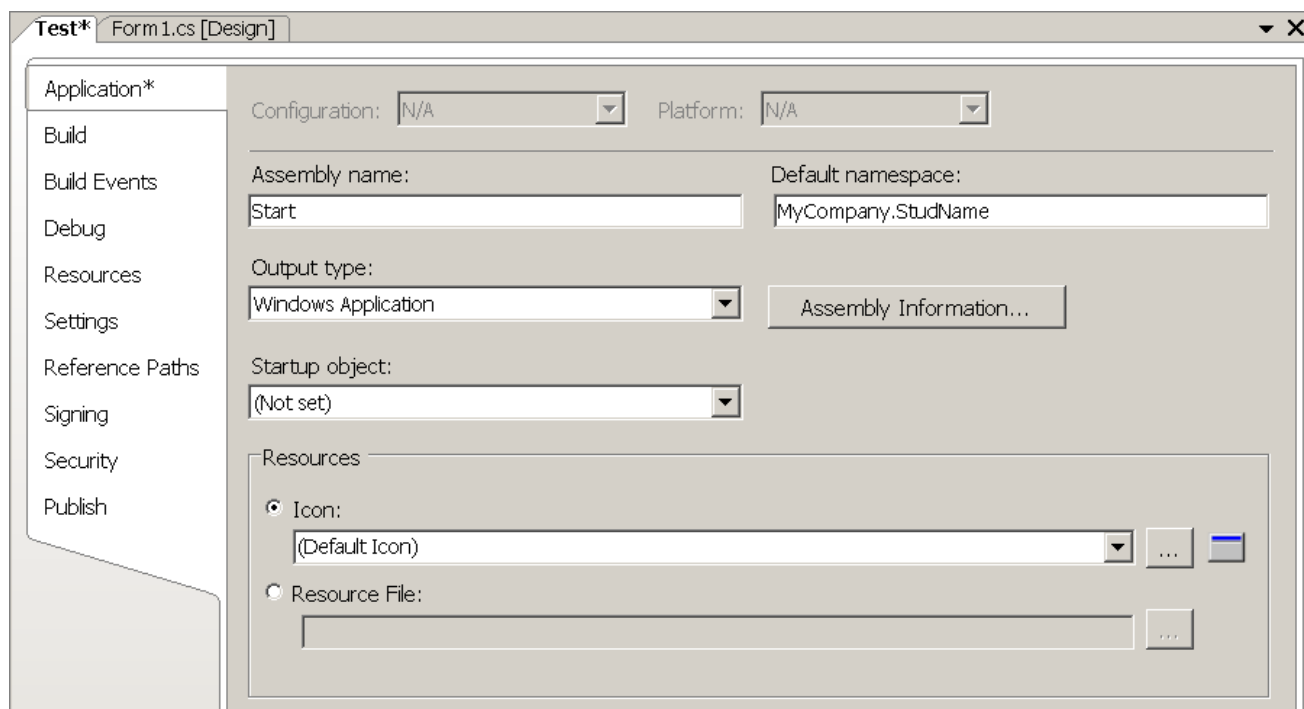
Это небольшое приложение, которое обеспечит удобный способ демонстрации результатов всех других упражнений работы. Попутно решим проблему, которая состоит в предотвращении повторного запуска уже запущенной формы. Применим для этого готовую коллекцию запущенных форм приложения `System.Windows.Forms.FormCollection`.

Выбрав удобное место размещения, создайте новое решение командой `File/New/Project` и заполните окно мастера в соответствии с (Рис. 34)

**Рис. 34**

Мастер оболочки создаст каталог MySolution для размещения многих проектов и добавит в него подкаталог Test с заготовкой проекта исполнимой сборки тестирования наших будущих компонентов.

В панели Solution Explorer вызовите контекстное меню для узла Test проекта, выполните команду Properties и настройте на вкладке Application текстовые поля Assembly name и Default namespace в соответствии с Рис. 35, где на место второй составляющей имени MyCompany. StudName подставьте свою фамилию (можно кириллицей).

**Рис. 35**

Тем самым мы указали оболочке имя исполнимой сборки и пространство имен, которое будет автоматически генерироваться при добавлении к проекту каждого нового элемента. Теперь, чтобы задействовать новые настройки оболочки, удалим старый файл Form1.cs и создадим его заново.

Откройте на редактирование файл Program.cs и замените в нем пространство имен Test на MyCompany.StudName.

В панели Solution Explorer удалите файл Form1.cs, в котором используется прежнее пространство имен Test, и на его место командой Project/Add Windows Form вновь добавьте файл с прежним именем Form1.cs.

Последнее действие выполняем для того, чтобы вручную не исправлять пространство имен в двух местах: Form1.cs и Form1.Designer.cs.

В панели Solution Explorer переименуйте файл Form1.cs в Start.cs и откройте его в режиме View Designer.

Сейчас мы займемся формированием стартовой формы для запуска всех будущих упражнений этой работы с использованием коллекции System.Windows.Forms.FormCollection. Вначале добавим к проекту три формы, чтобы испытать работоспособность подхода, а впоследствии мы их скорректируем в соответствии с выполненными упражнениями.

Добавьте к проекту три формы командой Project/Add Windows Form с именами Form1 и Form2.

В панели Solution Explorer вызовите для корневого узла проекта Test контекстное меню и командой Add/New Folder добавьте новую папку с именем ChildrenForms.

Удерживая клавишу Ctrl, выделите файлы Form1.cs, Form2.cs и переместите их мышью в папку ChildrenForms.

Мы пытаемся группировать файлы проекта по категориям решаемых задач. Но, если бы мы вначале создали папку, а потом стали генерировать в нее новые формы, то оболочка упаковала бы их в пространство имен MyCompany.StudName.ChildrenForms. Это привело бы к тому, что классы дочерних форм оказались бы в другом пространстве имен и мы вынуждены

были бы явно подключить его в файле Start.cs инструкцией using для видимости компилятору.

Откройте файл Start.cs в режиме View Designer и поместите на форму три компонента LinkLabel из панели Toolbox

Расположите компоненты LinkLabel на форме и выполните настройки в соответствии с Таблица 17.

**Таблица 17 Таблица свойств формы Start**

Свойство	Значение	Пояснения
Form	Name	По умолчанию
	Text	Lab3.СтудентName
	StartPosition	CenterScreen
	FormBorderStyle	FixedDialog
LinkLabel	Name	Form1
	Tag	Form1
	Text	Упражнение 1. ClickmaticButton
LinkLabel	Name	Form2
	Tag	Form2
	Text	Упражнение 2. ArrowButton
LinkLabel	Name	Form3
	Tag	Form3
	Text	Упражнение 3. NumericScan

Интерфейс главной формы должен стать примерно таким:



**Рис. 36**

Свойство Tag любых компонентов, в том числе LinkLabel, служит для простого сохранения объектов произвольного типа, поскольку имеет тип Object. Мы разместили в нем метки имени формы для распознавания в общем обработчике щелчка на объекте LinkLabel.

Откройте файл Start.cs в режиме View Code и добавьте в код класса поле объявления ссылки типа Form:

```
namespace MyCompany.StudName
{
    public partial class Start : Form
    {
        public Start()
        {
            InitializeComponent();
        }

        Form frm;
    }
}
```

Откройте файл Start.cs в режиме View Designer, выделите на форме одновременно все экземпляры LinkLabel и создайте через панель Properties для их события LinkClicked общий обработчик с именем OnLinkClicked.

Заполните обработчик OnLinkClicked следующим кодом:

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

```

namespace MyCompany.StudName
{
    public partial class Start : Form
    {
        public Start()
        {
            InitializeComponent();
        }

        Form frm;

        private void OnLinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
        {
            // Повышаем полномочия ссылки и извлекаем закреплённое имя
формы
            string formName = (string)((LinkLabel)sender).Tag;
            // Читаем коллекцию запущенных форм
            FormCollection fc = Application.OpenForms;
            bool IsRun = false;
            // Перебираем коллекцию
            foreach (Form form in fc)
            {
                // Имя очередной запущенной формы сравниваем с
закрепленным за кнопкой
                if (form.Name == formName)
                {
                    IsRun = true;    // Нашли, что форма запущена,
поднимаем флаг
                    frm = form;    // Сохраняем ссылку на запущенную
форму для фокуса
                    break;          // Выходим из цикла
                }
            }
            // Если форма не запущена - запускаем
            if (!IsRun)
            {
                switch (formName)
                {
                    case "Form1":
                        frm = new Form1();
                        break;
                    case "Form2":
                        frm = new Form2();
                        break;
                    case "Form3":
                        frm = new Form3();
                        break;
                }
            }
        }
    }
}

```

```

        this.AddOwnedForm(frm);           // Сделать новую frm
подчиненной Start
        // frm.Owner = this;             // Альтернативный способ
назначить владельца
        frm.ShowInTaskbar = false;       // Не отображать метку окна в
панели задач
        frm.Show();                     // Показать новую форму
    }
    else
        frm.Focus();                   // Передать фокус запущенной
форме
    }
}
}

```

Инструкция добавления подчиненной формы `this.AddOwnedForm(frm);` назначает для созданного экземпляра `frm` в качестве владельца текущую форму (в нашем случае - стартовую). Это означает, что подчиненное окно будет всегда размещаться поверх владельца, принудительно закрываться и разворачиваться тогда, когда это делает владелец. Таким поведением обладают, например, окна поиска и замены, или дочерние окна в многодокументных интерфейсах (MDI).

Если известна ссылка `frm` на подчиненную форму, то освободить ее позволяет метод владельца `RemoveOwnedForm(frm)`. Альтернативным способом назначить или сменить владельца у подчиненной формы можно через ее свойство `Owner`, а для освобождения от владельца нужно этому свойству присвоить нулевую ссылку `Owner=null`.

Запустите приложение и убедитесь, что на данном этапе наш механизм предотвращения повторного запуска форм не работает.

Дело в том, что имена форм по умолчанию в коллекции считаются пустыми до тех пор, пока явно не прописаны в теле класса формы. Оболочка сделает это явно в файле `*.Designer.cs` только после первого помещения на форму какого-либо компонента. Только тогда она посчитает, что может возникнуть необходимость их различать, и явно присвоит имя самой форме. Заставим оболочку явно прописать имена наших форм.

Поместите на каждую форму в каталоге `ChildrenForms` любой компонент пользовательского интерфейса (например, `Button`) и сразу же его удалите.

Опять запустите приложение и убедитесь, что все заработало правильно.

### Добавление объектов всплывающей подсказки

Добавим к `LinkLabel` всплывающие подсказки (оперативные справки) - объекты класса `ToolTip`, генерирующие окно сообщений при наведении курсора на прикрепленный объект. К сожалению, компонент `LinkLabel` не имеет встроенного свойства `ToolTip`, которое бы прикрепляло саму справку к компоненту. Поэтому мы вначале создадим саму подсказку - объект типа `ToolTip`, а затем уже прикрепим объект `LinkLabel` к этой подсказке. В коде стартовой формы нашей лабораторной работы имеются два объекта `LinkLabel`, поэтому создадим два объекта всплывающей подсказки `ToolTip`.

В начале класса `Start` (удобнее перед конструктором класса) создайте и инициализируйте два одномерных массива.

```

public partial class Start : Form
{
    // Массив сообщений всплывающих подсказок для LinkLabel
    String[] strTips =
    {
        "Тест для компонента кнопки, \n" +
        "генерирующей щелчки мыши",
        "Тест для компонента скроллирующей кнопки \n" +
        "со стрелками, генерирующей щелчки мыши",
        "Тест для компонента текстового \n" +
        "поля со счетчиком"
    };

    // Массив ссылок для объектов всплывающих подсказок
    ToolTip[] tips =
    {
        new ToolTip(),
        new ToolTip(),
        new ToolTip()
    };

    public Start()
    {
        InitializeComponent();
    }

    Form frm;

    .....
}

```

В конструкторе класса Start после вызова функции инициализации компонентов формы вставьте код, связывающий объекты подсказок с кнопками-ссылками и настраивающий эти подсказки:

```

public Start()
{
    InitializeComponent();

    // Привязка всплывающих подсказок к запускающим кнопкам
    tips[0].SetToolTip(linkLabel1, strTips[0]);
    tips[1].SetToolTip(linkLabel2, strTips[1]);
    tips[2].SetToolTip(linkLabel3, strTips[2]);

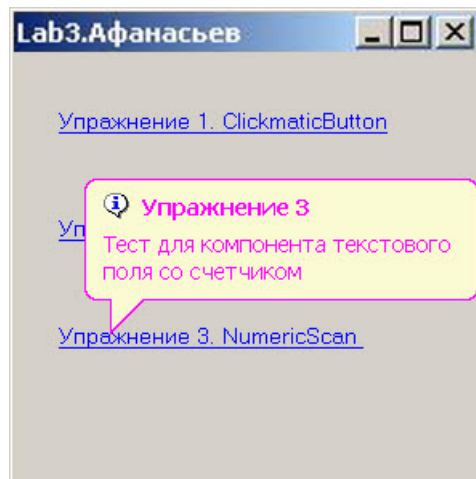
    // Настройка ToolTip
    for (int i = 0; i < tips.Length; i++)
    {
        tips[i].IsBalloon = true; // Использовать окно подсказки
        tips[i].ToolTipIcon = ToolTipIcon.Info; // Иконки
    }
}

```

```
tips[i].ToolTipTitle = "Упражнение " + (i + 1); //  
Заголовок  
tips[i].ForeColor = Color.Magenta; // Цвет текста  
tips[i].BackColor = Color.LightGoldenrodYellow; // Цвет  
фона  
}  
}
```

Возможные варианты цветового оформления всплывающих подсказок представлены в Приложение 1 Таблица цветов структуры Color.

Запустите тестовую форму (F5) и проверьте механизм работы всплывающих подсказок:

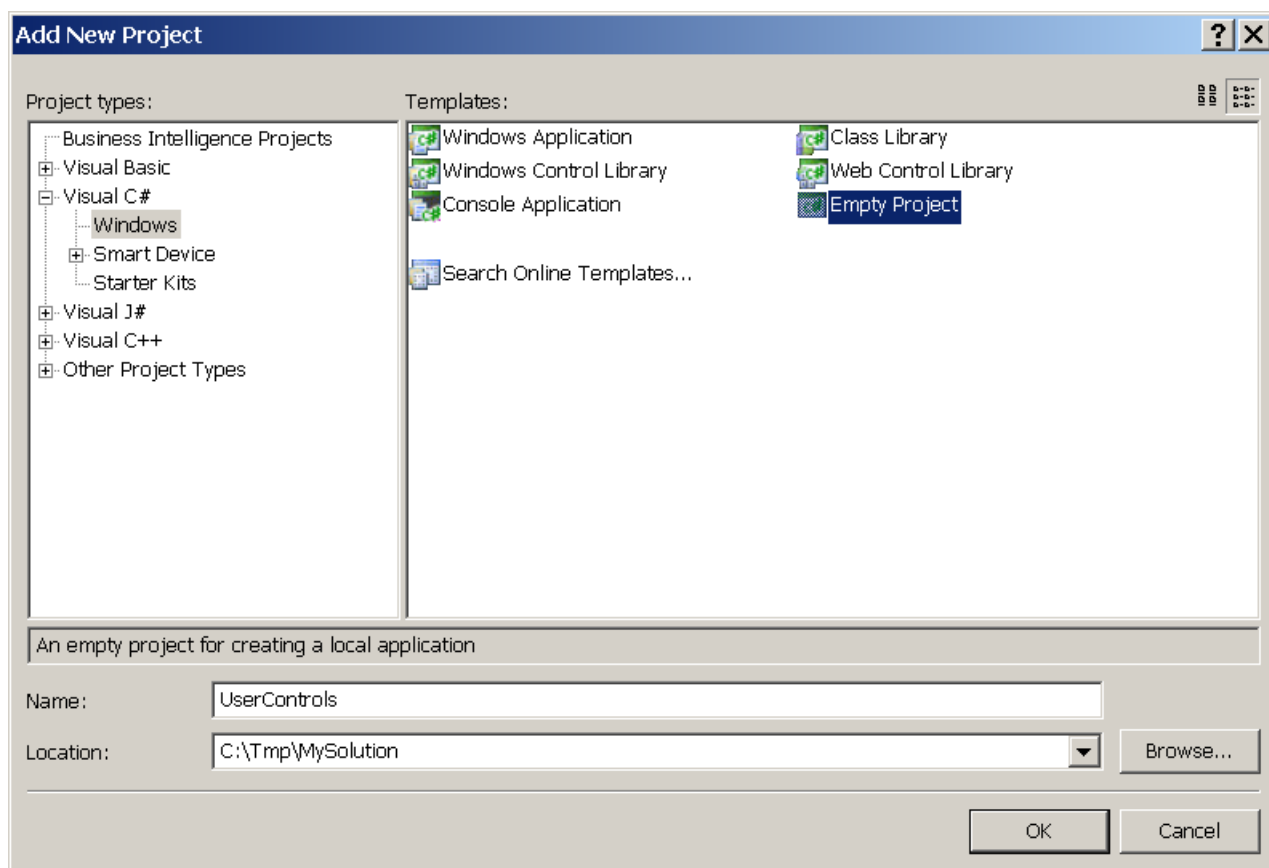


**Рис. 37**

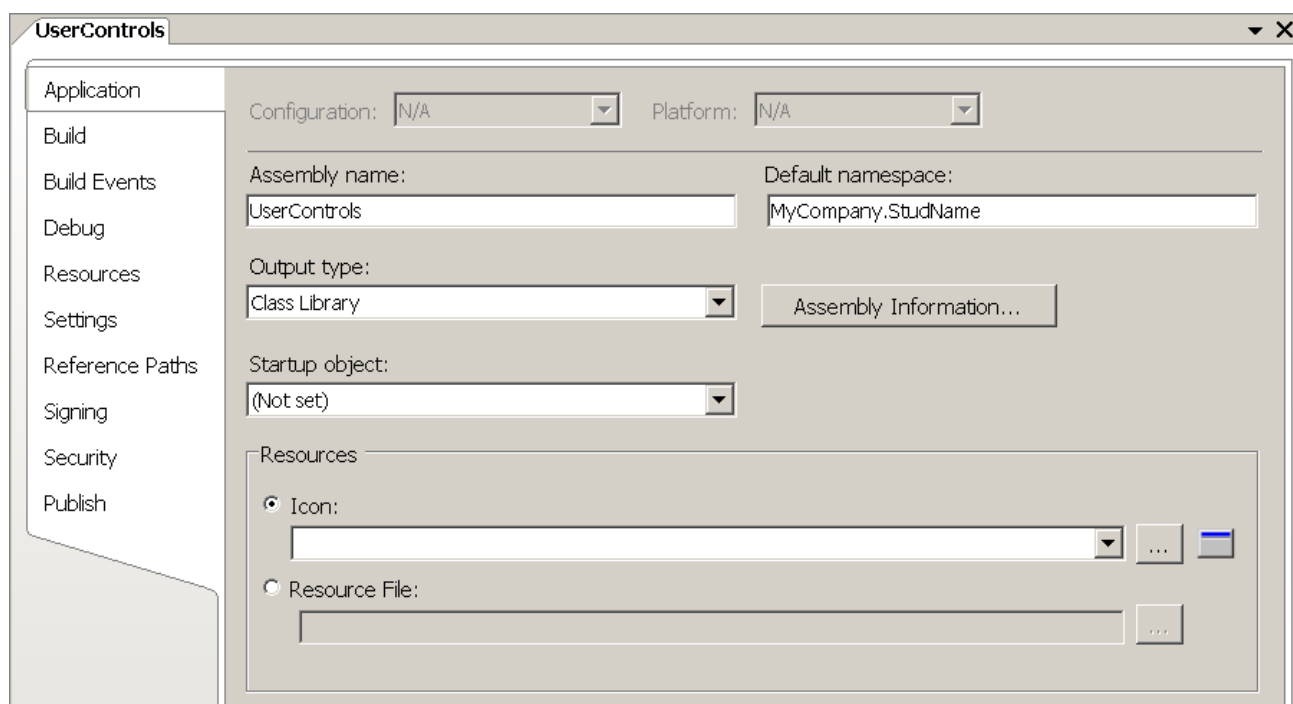
Теперь для запуска тестов нескольких упражнений подготовлена основа и можно приступать к разработке пользовательских компонентов.

### **Разработка кнопки ClickmaticButton**

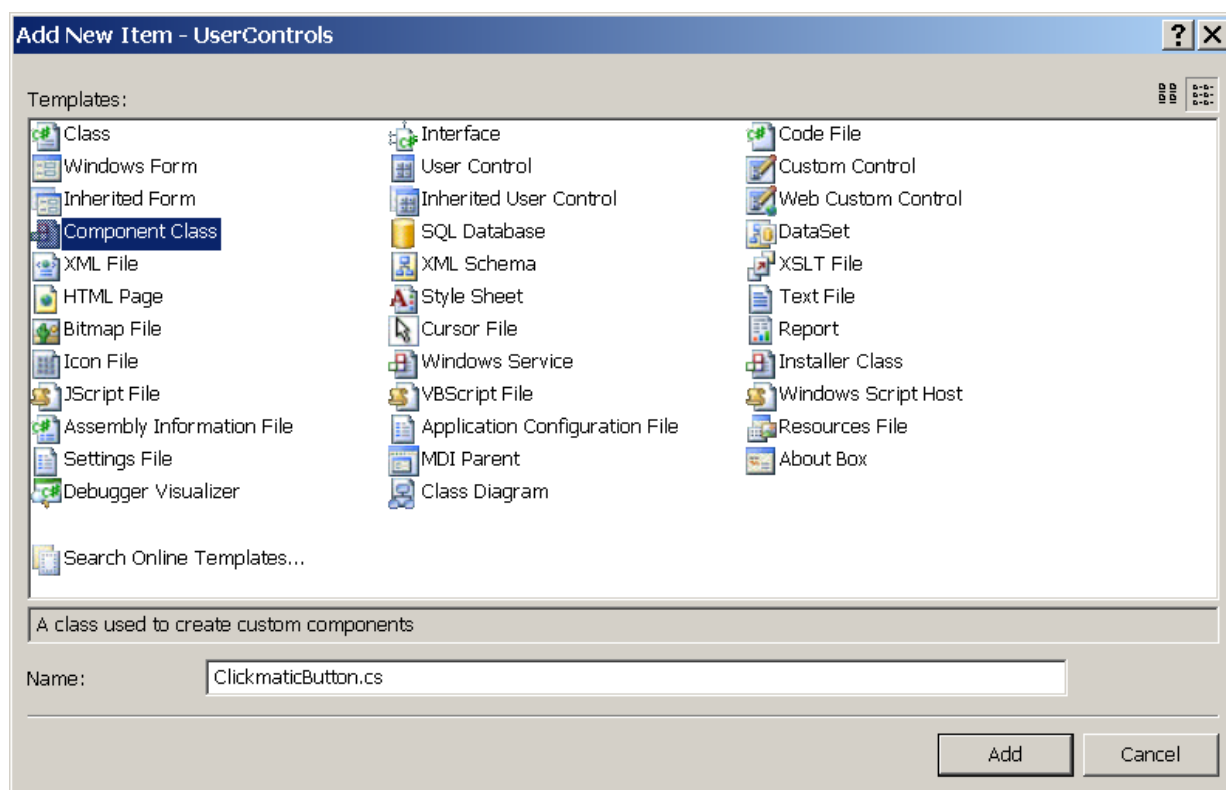
Добавьте к решению MySolution новый пустой проект командой File/Add/New Project и заполните окно следующим образом:

**Рис. 38**

В панели Solution Explorer для корневого узла проекта UserControls выполните команду Properties контекстного меню и настройте вкладку Application свойств проекта как показано на Рис. 39:

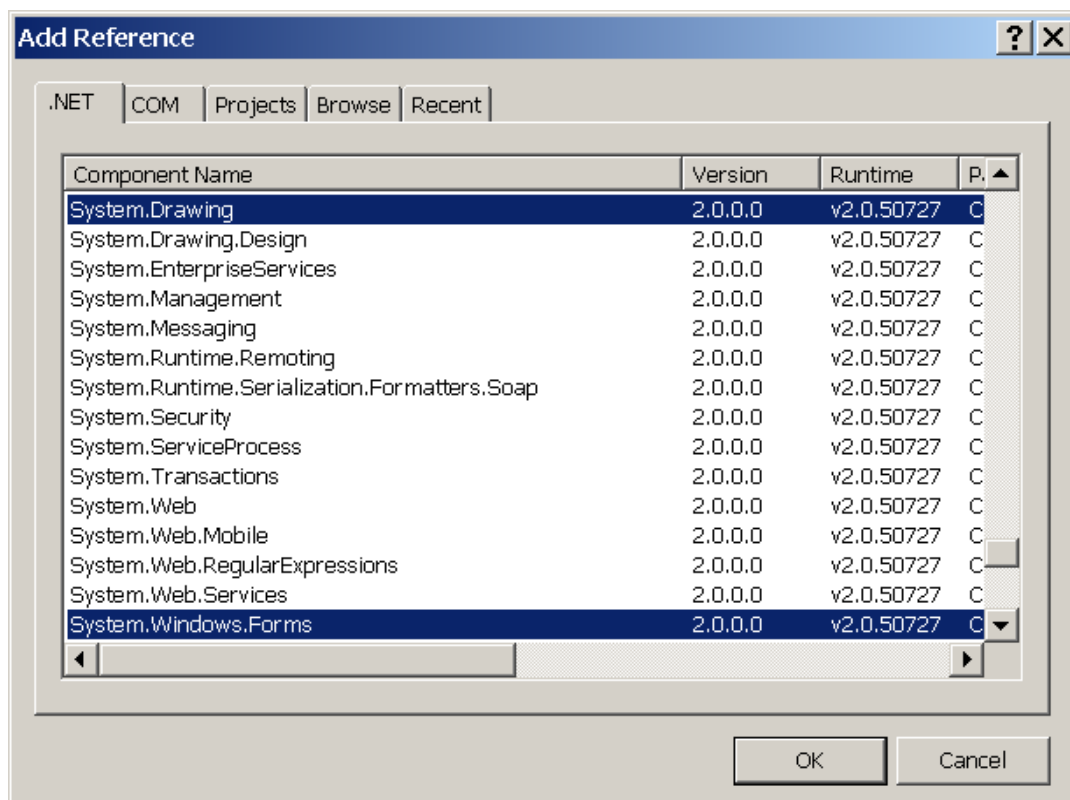
**Рис. 39**

В панели Solution Explorer вызовите для корневого узла проекта UserControls контекстное меню и командой Add/Component добавьте элемент ClickmaticButton (Рис. 40):

**Рис. 40**

В панели Solution Explorer для узла References проекта UserControls, где уже должна находиться ссылка на библиотечную сборку System, выполните команду Add Reference.

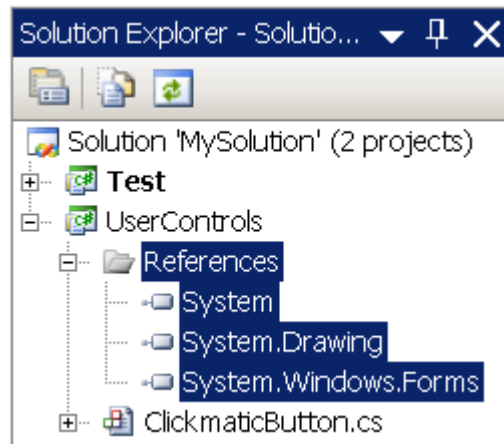
Удерживая клавишу **Ctrl** через окно Add Reference выделите и добавьте к проекту UserControl1 ссылки на библиотечные сборки System.Drawing и System.Windows.Forms (Рис. 41).



**Рис. 41**



Убедитесь, что теперь узел References проекта UserControls стал выглядеть так:



**Рис. 42**

Переведите файл ClickmaticButton.cs [Design] в режим редактирования кода командой контекстного меню View Code.

Добавьте в начало файла ClickmaticButton.cs необходимые инструкции using подключения нужных нам пространств имен и замените базовый класс Component на Button, чтобы код файла стал таким:

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

namespace MyCompany.StudName
{
    public partial class ClickmaticButton : Button // : Component
    {
        public ClickmaticButton()
        {
            InitializeComponent();
        }

        public ClickmaticButton(IContainer container)
        {
            container.Add(this);

            InitializeComponent();
        }
    }
}
```

Мы создали заготовку расширения библиотечного класса кнопки с целью изменить ее функциональность таким образом, чтобы в нажатом состоянии кнопка генерировала имитацию щелчков пользователя.

## Наполнение класса ClickmaticButton функциональностью генерации щелчков мыши

Выполните команду контекстного меню View Designer и поместите из свитка Components панели Toolbox на форму ClickmaticButton.cs [Design] компонент Timer. Присвойте созданному экземпляру имя timer.

Двойным щелчком на экземпляре timer создайте обработчик timer\_Tick().

Добавьте в класс ClickmaticButton два поля целого типа для задания задержки на момент запуска таймера и интервала повторения тиков. Для этого используйте соответствующие библиотечные свойства класса System.Windows.Forms.SystemInformation, предусмотренные для управления клавиатурой:

```
public partial class ClickmaticButton : Button // : Component
{
    .....

    // Поля характеристик работы таймера
    readonly int DELAY = 250 * (1 + SystemInformation.KeyboardDelay);
    readonly int SPEED = 405 - 12 * SystemInformation.KeyboardSpeed;

    private void timer_Tick(object sender, EventArgs e)
    {

    }
}
```

Ключевое слово readonly дает указание компилятору следить за тем, чтобы где-нибудь в коде мы случайно не попытались повторно присвоить этим полям новое значение. Оно позволяет выполнить отложенную инициализацию один раз, причем вычисляемым выражением в любом методе класса. Ключевое слово const требует немедленной инициализации переменной, причем константной величиной, что в данном случае нам не подходит.

Начав ввод с ключевого слова override, переопределите в классе ClickmaticButton унаследованный метод OnMouseMove() для управления приостановкой работы таймера, когда курсор при наличии установленной связи "разрабатываемый элемент кнопки - устройство мыши" будет выведен пользователем за пределы области чувствительности элемента кнопки:

```
// Переопределение унаследованного метода диспетчеризации
// события MouseMove увода курсора за область чувствительности элемента
protected override void OnMouseMove(MouseEventArgs mevent)
{
    base.OnMouseMove(mevent);

    // Приостанавливаем или возобновляем запущенный таймер
    логическим выражением
    timer.Enabled = this.Capture
        // Связь с мышью установлена
        && (MouseButtons & MouseButtons.Left) != 0
```

```

        // Распознавать левую кнопку необязательно
        && this.ClientRectangle.Contains(mevent.Location);
        // Курсор над кнопкой
        &&
this.ClientRectangle.Contains(this.PointToClient(MousePosition))
// То же самое!
    }

```

Начав ввод с ключевого слова `override`, переопределите в классе `ClickmaticButton` унаследованный метод `OnMouseDown()` диспетчеризации события `MouseDown` для запуска таймера при нажатии левой кнопки мыши:

```

// Переопределение унаследованного метода диспетчеризации
// события MouseDown нажатия кнопки мыши для запуска таймера
protected override void OnMouseDown(MouseEventArgs mevent)
{
    base.OnMouseDown(mevent);    // Отправляем к базовому методу

    // Если нажата левая кнопка мыши (побитовое умножение)
    if ((mevent.Button & MouseButtons.Left) != 0)
    {
        timer.Interval = DELAY;    // Задержка для момента
нажатия
        timer.Start();    // Запустить таймер
    }
}

```

Если мы не отправим вызов перехваченного виртуального метода далее к базовому классу, то хоть таймер и запустится, но интерфейс кнопки не будет меняться привычным для пользователя образом и подписанные на событие `MouseDown` обработчики не сработают. Запущенный таймер до первого срабатывания выдержит паузу `DELAY`, затем начнет вызывать обработчик `timer_Tick()`, в котором мы установим новый интервал срабатывания `SPEED` и будем принудительно вызывать метод диспетчеризации события `Click` элемента кнопки, имитируя тем самым генерацию щелчков пользователя на нем.

Заполните ранее созданный обработчик `timer_Tick()` тиков таймера следующим кодом:

```

private void timer_Tick(object sender, EventArgs e)
{
    timer.Interval = SPEED; // Устанавливаем новый интервал
генерации тиков
    this.OnClick(EventArgs.Empty); // Генерируем щелчок на
элементе кнопки
                                     // с пустым фактическим
параметром
}

```

Остановку запущенного таймера и, соответственно, прекращение генерации щелчков пользователя на элементе кнопки предусмотрим в переопределении метода `OnMouseUp()` диспетчеризации события `MouseUp` отпускания кнопки мыши.

Начав ввод с ключевого слова `override`, переопределите в классе `ClickmaticButton` унаследованный метод `OnMouseUp()` диспетчеризации события `MouseUp` для остановки таймера при отпускании кнопки мыши:

```
// Переопределение унаследованного метода диспетчеризации
// события MouseUp отпускания кнопки мыши для остановки таймера
protected override void OnMouseUp(MouseEventArgs mevent)
{
    base.OnMouseUp(mevent);    // Отправляем к базовому методу

    timer.Stop();    // Стоп таймер
}
```

На этом функциональность компонента кнопки с генерацией серии событий щелчков на нем закончена. Далее мы выполним еще одно расширение класса `ClickmaticButton`, чтобы нарисовать на кнопке стрелку. Для изменения направления стрелки предусмотрим открытое свойство `ScrollButton`. Но прежде проверим работоспособность сконструированного расширения кнопки командой `Build`.

### Тестирование генерирующей кнопки `ClickmaticButton`

Закройте все редактируемые текущие документы командой меню `Window/Close All Documents`.

В панели `Solution Explorer` вызовите на редактирование в режиме `View Designer` форму `Form1` двойным щелчком по одноименному узлу.

В панели `Solution Explorer` вызовите контекстное меню для узла `UserControls` проекта пользовательских `DLL`-компонентов и выполните команду `Rebuild`, чтобы заново откомпилировать единственный пока компонент `ClickmaticButton`.

Откройте панель `Toolbox`, найдите в верхней ее части свиток для категории `UserControls Components` и двойным щелчком на расширенном компоненте `ClickmaticButton` создайте на форме `Form1` его экземпляр. Присвойте его свойству `Name` значение `btnClickGen`, а свойству `Text` - значение `ClickGen`.

Поместите на форму библиотечный компонент `Label` для счетчика тиков и присвойте его свойству `Name` значение `lblTickCount`.

Двойным щелчком на экземпляре кнопки `ClickmaticButton` создайте обработчик события `Click`, который заполните так:

```
private void btnClickGen_Click(object sender, EventArgs e)
{
    int tickCount;
    if (!Int32.TryParse(lblTickCount.Text, out tickCount))
        tickCount = 1;    // При некорректном преобразовании
установить 1
    else
        tickCount++;    // При корректном преобразовании
увеличить на 1

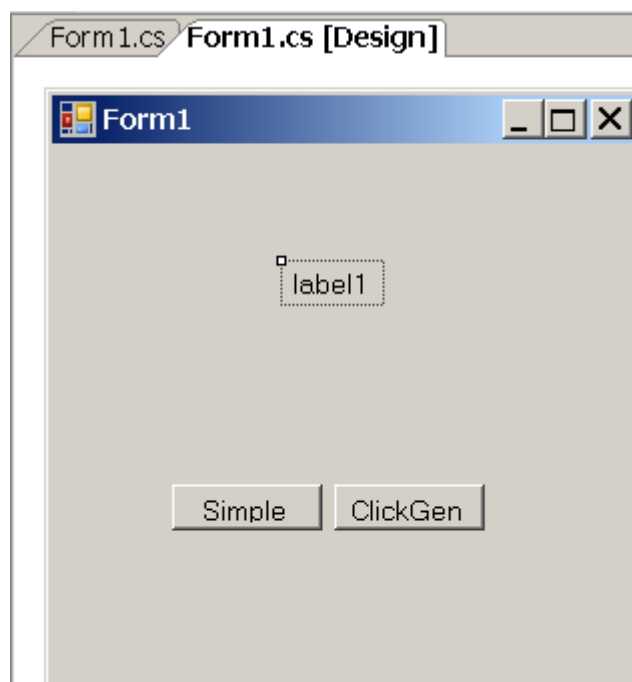
    lblTickCount.Text = tickCount.ToString();
}
```

Статический метод `TryParse()` структуры `Int32` преобразует строковый тип в целый без генерации исключения в случае невозможности выполнить эту операцию. В нашем тесте при первом вызове этого метода свойство `Text` метки не содержит числовых символов, следовательно, преобразование невозможно, и метод возвращает `false`. Мы такую ситуацию контролируем и устанавливаем `tickCount=1`, иначе увеличиваем значение счетчика.

Поместите на форму экземпляр обычного библиотечного компонента `Button` и присвойте его свойству `Text` значение `Simple`.

Выделите на форме экземпляр обычного компонента `Button`, переведите панель `Properties` оболочки в режим `Events` и через раскрывающийся список в поле события `Click` присоедините к обычной кнопке тот же обработчик `btnClickGen_Click()`.

В итоге пользовательский интерфейс тестовой формы должен быть примерно таким:



**Рис. 43**

Запустите стартовую форму, выберите тест "Упражнение 1" и убедитесь в том, что спроектированная нами кнопка имеет дополнительную функциональность генерации щелчков, а также - в том, что работает механизм приостановки таймера в случае выхода курсора за область чувствительности кнопки.

Такая кнопка нужна только для дальнейшего расширения и самостоятельной ценности не имеет, поэтому для тренировки запретим создавать из нее экземпляры как объекты, а тестовый пример уничтожим.

Откройте файл `Form1.cs` (если он еще не открыт) в режиме `View Designer` и удалите на ней все экземпляры пользовательского интерфейса, созданные ранее для тестирования кнопки `ClickmaticButton`.

С помощью контекстного меню для формы откройте `Form1.cs` в режиме `View Code` и удалите вручную обработчик `btnClickGen_Click()`, поскольку при удалении элементов заполненные обработчики автоматически не удаляются.

Убедитесь, находясь в режиме `Designer` формы, что в панели `Toolbox` компонент `ClickmaticButton` пока присутствует и готов к размещению на форме.

Откройте файл `ClickmaticButton.cs` в режиме `View Code` и добавьте перед объявлением класса `ClickmaticButton` ключевое слово `abstract`.

```
namespace MyCompany.StudName
{
    abstract public partial class ClickmaticButton : Button // :
Component
    {
        .....
    }
}
```

Таким действием мы запретили создание экземпляров этого класса. Теперь его можно только наследовать.

В панели Solution Explorer вызовите контекстное меню для узла UserControls и выполните команду Build.

Откройте файл Form1.cs в режиме View Designer и убедитесь, что пиктограмма компонента ClickmaticButton исчезла из панели Toolbox.

Правильно, если создавать экземпляры нельзя, значит, и компоненту там не должно быть.

Теперь восстановите тестовую форму Form1 для "Упражнения 1", убрав объявление abstract.

## Разработка кнопки в стиле полос прокрутки

Наследуем от ClickmaticButton еще один класс ArrowButton, в котором переопределим виртуальный метод OnPaint() для отрисовки кнопки как в полосах прокрутки, где надпись будет замещена треугольной стрелкой. Образ скроллирующей кнопки будем отрисовывать методом ControlPaint.DrawScrollButton() из пространства имен System.Windows.Forms. Направление рисования функцией стрелки регулируется ее параметром типа библиотечного перечисления ScrollButton из того же пространства имен.

В панели Solution Explorer вызовите контекстное меню для узла UserControls и добавьте к проекту командой Add/Component класс для размещения нового компонента с именем ArrowButton.

Перейдите в режим View Code, замените наследуемый класс Component на ClickmaticButton, чтобы не потерять ту функциональность, которую мы сделали ранее

Теперь нужно установить такой стиль для кнопки, чтобы в конечном пользовательском элементе (NumericScan - поле с кнопками) она никогда бы не получала фокус ввода. Фокус должен быть только у текстового поля.

Поместите в конец каждого из конструкторов ArrowButton() после вызова функции создания и инициализации дочерних компонентов инструкцию установки нужного стиля, а также добавьте подключение необходимых пространств имен.

В результате код пользовательской части класса ArrowButton на данном этапе расширения должен стать таким:

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

```

namespace MyCompany.StudName
{
    public partial class ArrowButton : ClickmaticButton
    {
        public ArrowButton()
        {
            InitializeComponent();

            // Отключение фокуса ввода
            this.SetStyle(ControlStyles.Selectable, false);
        }

        public ArrowButton(IContainer container)
        {
            container.Add(this);

            InitializeComponent();

            // Отключение фокуса ввода
            this.SetStyle(ControlStyles.Selectable, false);
        }
    }
}

```

Определим поле для хранения значения перечисления ориентации скроллирующей кнопки и обернем его в общедоступное свойство. Для этого дополним класс ArrowButton следующим кодом :

```

// Управление ориентацией стрелки на кнопке
System.Windows.Forms.ScrollButton scrollButton =
    System.Windows.Forms.ScrollButton.Right;
public System.Windows.Forms.ScrollButton ScrollButton
{
    set
    {
        scrollButton = value;
        this.Invalidate(); // Перерисовать
    }
    get { return scrollButton; }
}

```

Начав ввод с ключевого слова `override`, переопределим в классе ArrowButton унаследованный метод `OnPaint()` для отрисовки кнопки в стиле скроллирующей и заполните его следующим кодом:

```

// Отрисовка кнопки в стиле со стрелкой
protected override void OnPaint(PaintEventArgs pevent)
{
    base.OnPaint(pevent); // Отправляем к базовому методу
}

```

```

// или вообще убираем!!!

Graphics gr = pevent.Graphics; // Извлекаем контекст
устройства

// Кнопка в контакте с мышью и курсор находится в области
чувствительности
bool mouseInButton = this.Capture &&

this.ClientRectangle.Contains(this.PointToClient(MousePosition));

// Флаг состояния кнопки для управления ее стилем отрисовки
System.Windows.Forms.ButtonState buttonState =
    !this.Enabled ? ButtonState.Inactive // Если
кнопка недоступна
    : (mouseInButton ? ButtonState.Pushed // Если
доступна и нажата
    : ButtonState.Normal); // Если
доступна и отпущена

// Рисуем кнопку как кнопку со стрелкой
ControlPaint.DrawScrollButton(gr, this.ClientRectangle,
    scrollButton, // В какую сторону рисовать стрелку
    buttonState); // Каким стилем рисовать
}

```

Когда активная мышь теряет или восстанавливает контакт с кнопкой, кнопку со стрелкой нужно перерисовывать в соответствии с принятым стилем.

Начав ввод с ключевого слова `override`, переопределите в классе `ArrowButton` унаследованный метод `OnMouseCaptureChanged()` для повторения отрисовки кнопки в стиле со стрелкой и заполните его следующим кодом:

```

// Повторяем отрисовку при потере или восстановлении контакта с мышью
protected override void OnMouseCaptureChanged(EventArgs e)
{
    base.OnMouseCaptureChanged(e);

    this.Invalidate(); // Инициировать перерисовку
}

```

## Тестирование скроллирующей кнопки `ArrowButton`

Откройте файл `Form2.cs` в режиме `View Designer`.

Откомпилируйте проект пользовательской библиотеки `UserControls`, чтобы в панели `Toolbox` появился разработанный компонент `ArrowButton`.

Поместите на `Form2` четыре компонента `ArrowButton`.

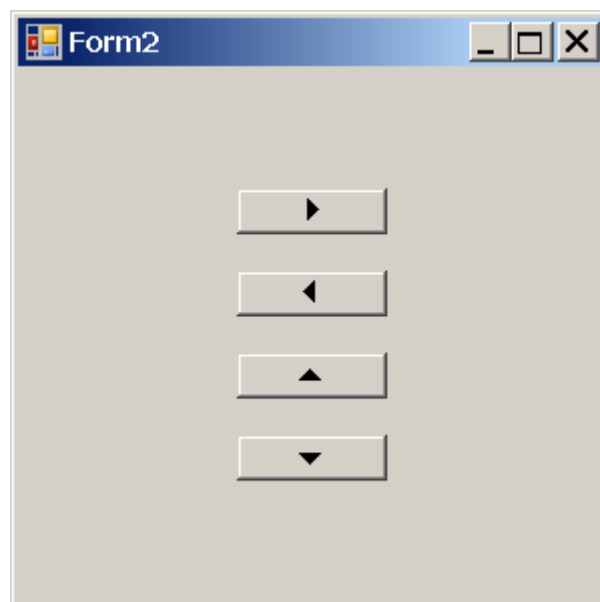


Переведите форму в режим View Code и настройте программно (в отличие от декларативного способа через панель Properties) свойство ScrollButton компонентов на разные значения в конструкторе класса Form2:

```
namespace MyCompany.StudName
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent(); // Выполнение кода, сгенерированного
                                // визуальным конструктором оболочки
                                // в декларативном режиме
                                // и расположенного в файле
                                // проектирования
                                // Form2.Designer.cs

            // Настройка ориентации стрелок на скроллирующих кнопках
            arrowButton1.ScrollButton = ScrollButton.Right;
            arrowButton2.ScrollButton = ScrollButton.Left;
            arrowButton3.ScrollButton = ScrollButton.Up;
            arrowButton4.ScrollButton = ScrollButton.Down;
        }
    }
}
```

Запустите приложение (F5) и посмотрите на результат, который будет следующим:



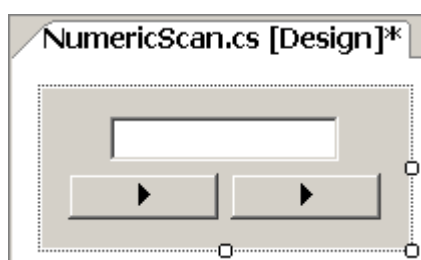
**Рис. 44**

## Комбинирование элементов в единый компонент NumericScan

Теперь мы готовы сконструировать аналог библиотечного компонента NumericUpDown - поле со списком, только с горизонтальным расположением скролирующих кнопок.

В панели Solution Explorer вызовите контекстное меню для узла UserControls и добавьте к проекту командой Add/User Control форму для размещения нового комбинированного компонента с именем NumericScan. Отметьте для себя, что в качестве базового в заготовке класса наследуется UserControl.

Перенесите на форму NumericScan.cs в режиме View Designer из панели Toolbox два наших компонента ArrowButton и один библиотечный компонент TextBox. На данном этапе особо не заботьтесь о размещении экземпляров компонентов на форме. Мы это действие выполним позже программным способом. А пока расположите объекты компонента примерно так:



**Рис. 45**

Далее мы часть настроек выполним декларативно через панель Properties, а часть - программно в конструкторе компонента.

Через панель Properties выполните декларативные настройки объектов в соответствии с Таблица 18.

**Таблица 18 Таблица свойств декларативной настройки объектов компонента NumericScan**

Тип	Свойство	Значение	Пояснения
TextBox	Name	txtBox	Имя экземпляра компонента текстового поля
	TextAlign	Right	Расположение текста
ArrowButton	Name	btnLeft	Имя объекта скролирующей кнопки
	ScrollButton	Left	Ориентация стрелки влево
ArrowButton	Name	btnRight	Имя объекта скролирующей кнопки
	ScrollButton	Right	Ориентация стрелки вправо (по умолчанию)

Через панель Properties в режиме Events зарегистрируйте обработчики для составляющих объектов в соответствии с Таблица 19.

**Таблица 19 Таблица событий декларативной настройки объектов компонента NumericScan**

Объект	Событие	Обработчик	Пояснения
--------	---------	------------	-----------

txtBox	TextChanged	TextBoxOnTextChanged	Возбуждается при завершении ввода клавишей Enter или потери фокуса
	KeyDown	TextBoxOnKeyDown	Возбуждается при нажатии любой клавиши, когда объект имеет фокус ввода
btnLeft	Click	ButtonOnClick	Возбуждается при щелчке на кнопке курсором мыши
btnRight	Click	ButtonOnClick	События двух кнопок подписаны на один обработчик

Все остальные настройки составляющих объектов и самого компонента выполним программно.

Переведите редактирование файла NumericScan.cs в режим View Code и добавьте в конструктор компонента код вычисления размеров формы:

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

namespace MyCompany.StudName
{
    public partial class NumericScan : UserControl
    {
        public NumericScan()
        {
            InitializeComponent();

            // Вычисляем и устанавливаем размеры компонента
            // Четыре высоты текущего шрифта
            this.Width = 4 * this.Font.Height;
            // Предпочтительная высота текстового поля +
            // высота горизонтальных линеек прокрутки
            this.Height = txtBox.PreferredHeight +
                SystemInformation.HorizontalScrollBarHeight;
        }

        // При завершении ввода клавишей Enter или потери фокуса
        private void TextBoxOnTextChanged(object sender, EventArgs e)
        {
        }

        // При нажатии любой клавиши, когда объект имеет фокус ввода
```

```
private void TextBoxOnKeyDown(object sender, KeyEventArgs e)
{

}

// Щелчки на скроллирующих кнопках
private void ButtonOnClick(object sender, EventArgs e)
{

}
}
}
```

Добавьте в класс `NumericScan` закрытые поля для сохранения настроек компонента, а также оберните их общедоступными свойствами:

```
public partial class NumericScan : UserControl
{
    // Объявили базовые закрытые поля для открытых свойств
    int iDecimalPlaces = 0; // Количество знаков после запятой
    decimal mValue = 0;     // Значение текстового поля
    decimal mIncrement = 1; // Шаг изменения
    decimal mMinimum = 0;   // Минимальное вещественное значение
    decimal mMaximum = 100; // Максимальное вещественное значение

    // Свойства доступа к полям
    public int DecimalPlaces
    {
        get { return iDecimalPlaces; }
        set { iDecimalPlaces = value; }
    }
    public decimal Value
    {
        get { return mValue; }
        set { txtBox.Text = (mValue = value).ToString(); }
    }
    public decimal Increment
    {
        get { return mIncrement; }
        set { mIncrement = value; }
    }
    public decimal Minimum
    {
        get { return mMinimum; }
        set
        {
            // Контроль нижней границы диапазона
            if (Value < (mMinimum = value))
                Value = mMinimum;
        }
    }
}
```

```

    }
}
public decimal Maximum
{
    get { return mMaximum; }
    set
    {
        // Контроль верхней границы диапазона
        if (Value > (mMaximum = value))
            Value = mMaximum;
    }
}

.....
}

```

Обратите внимание, что свойство Value в аксессоре set выполняет двойное присваивание: текстовому полю и полю класса.

Начните с ввода ключевого слова `override` и переопределите в классе `NumericScan` унаследованную виртуальную функцию `GetPreferredSize()` вычисления начальных размеров компонента при создании его экземпляра:

```

// Срабатывает автоматически и устанавливает начальные размеры
// экземпляра компонента this.Width, this.Height при его
// создании или помещении на форму в режиме проектирования
public override Size GetPreferredSize(Size proposedSize)
{
    return new Size(4 * this.Font.Height, // Ширина
        txtBox.PreferredHeight +
        SystemInformation.HorizontalScrollBarHeight);
}

```

Начните с ввода ключевого слова `override` и переопределите в классе `NumericScan` унаследованную виртуальную функцию `OnResize()` вычисления локализации и размеров дочерних объектов компонента:

```

// Срабатывает автоматически, позиционирует
// дочерние объекты и устанавливает их размеры
protected override void OnResize(EventArgs e)
{
    base.OnResize(e);

    txtBox.Location = new Point(0, 0); // В левом верхнем углу
    txtBox.Size = new Size(this.Width, txtBox.PreferredHeight);
// По всей ширине
    btnLeft.Location = new Point(0, txtBox.Height); // Позиция
    btnRight.Location = new Point(this.Width / 2, txtBox.Height);
// Позиция
    btnLeft.Size = btnRight.Size = new Size(this.Width / 2,

```

```
        this.Height - textBox.Height); // Одинаковый размер
    }
```

Поддержка интерфейсного вида компонента обеспечена. Теперь необходимо согласовать логику работы кнопок и текстового поля. Начнем с обработки события щелчка на кнопках. Для этого мы предусмотрели общий обработчик `ButtonOnClick()`. В нем нужно распознать нажатую кнопку и изменить значение текстового поля, а также сохранить это значение во внутреннем поле `mValue` класса. Но прежде объявим событие, информирующее клиента, который будет создавать экземпляр компонента, об изменении значения поля компонента, а также функцию, которая будет инициировать генерацию этого события.

Объявите в классе `NumericScan` с помощью библиотечного делегата `EventHandler` событие с именем `ValueChanged` и определите функцию `OnValueChanged()` диспетчеризации этого события. Функцию диспетчеризации объявите виртуальной на случай дальнейшего наследования нашего компонента:

```
// Объявили пользовательское событие
public event EventHandler ValueChanged;

// Ввели свою функцию диспетчеризации события ValueChanged
// изменения величины прямым вводом в текстовое поле
protected virtual void OnValueChanged(EventArgs args)
{
    // Последовательно проверяем выход величины за левую и правую
    // границы, и если выходит за границу, то обрезаем по границе
    Value = Math.Min(mMaximum, mValue);
    Value = Math.Max(mMinimum, mValue);
    // Округляет вещественный денежный тип до заданного
    // количества значащих цифр после запятой
    Value = Decimal.Round(mValue, iDecimalPlaces);

    // Генерируем событие
    if (ValueChanged != null)
        ValueChanged(this, args);
}
```

Задайте в теле обработчика `ButtonOnClick()` код, который будет контролировать изменения внутреннего и текстового полей компонента при щелчках на кнопках, а также извещать подписавшегося клиента об этих изменениях:

```
// Щелчки на скроллирующих кнопках
private void ButtonOnClick(object sender, EventArgs e)
{
    // Повышаем полномочия ссылки на объект скролирующей кнопки
    ArrowButton btn = (ArrowButton)sender;

    // Создаем пробную переменную
    decimal tmpValue = mValue;

    // Идентифицируем кнопку и меняем значение поля
```

```
if (btn == btnLeft)
{
    if ((tmpValue -= mIncrement) < mMinimum)
        return;
}
else // Других кнопок нет
{
    if ((tmpValue += mIncrement) > mMaximum)
        return;
}

// Обновляем внутреннее и текстовое поля
this.Value = tmpValue;

// Генерируем событие, извещающее подписавшегося
// клиента о произошедшем изменении значения счетчика
OnValueChanged(EventArgs.Empty);
}
```

Заполните обработчик `TextBoxOnTextChanged()` кодом, который будет обновлять внутреннее поле при его корректном изменении или восстанавливать из него корректное значение текстового поля при неправильном заполнении:

```
// При завершении ввода клавишей Enter или потери фокуса
private void TextBoxOnTextChanged(object sender, EventArgs e)
{
    if (txtBox.Text.Length == 0)
        return;

    // При неудачной попытке преобразования восстанавливаем
старое
    decimal tmpValue;
    if (!Decimal.TryParse(txtBox.Text, out tmpValue))
        txtBox.Text = mValue.ToString();
    else
        mValue = tmpValue;
}
```

Когда компонент теряет фокус, возбуждается событие `Leave`, которое инициируется методом диспетчеризации `OnLeave()`. В нашем случае тоже нужно обновить состояние компонента и известить подписавшегося клиента.

Переопределите в классе компонента виртуальный метод `OnLeave()` и заполните его следующим кодом:

```
// При уходе с компонента
protected override void OnLeave(EventArgs e)
{
    base.OnLeave(e);
}
```

```
// Возбуждаем событие и обновление состояния компонента  
OnValueChanged (EventArgs.Empty);  
}
```

Заполните обработчик TextBoxOnKeyDown() кодом, который будет контролировать нажатие клавиши Enter в текстовом поле и обновлять состояние компонента:

```
// При нажатии любой клавиши, когда объект имеет фокус ввода  
private void TextBoxOnKeyDown(object sender, KeyEventArgs e)  
{  
    switch (e.KeyCode)  
    {  
        case Keys.Enter:  
            OnValueChanged (EventArgs.Empty);  
            break;  
    }  
}
```

В панели Solution Explorer выберите узел UserControls и командой Rebuild контекстного меню откомпилируйте готовый компонент.

### Тестирование компонента NumericScan

Поместите из панели Toolbox на форму Form3 два экземпляра компонента NumericScan и одну текстовую метку Label.

Выделите одновременно оба объекта NumericScan и в панели Properties для их события ValueChanged, введенное нами при разработке компонента, зарегистрируйте общий обработчик с именем NumericScanOnValueChanged, который заполните так:

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Text;  
using System.Windows.Forms;  
  
namespace MyCompany.StudName  
{  
    public partial class Form3 : Form  
    {  
        public Form3()  
        {  
            InitializeComponent();  
        }  
    }  
}
```



```
private void NumericScanOnValueChanged(object sender, EventArgs e)
{
    label1.Text = "Первый: " + numericScan1.Value +
        "; Второй: " + numericScan2.Value;
}
}
```

Настройте в конструкторе класса Form3 экземпляры компонентов NumericScan:

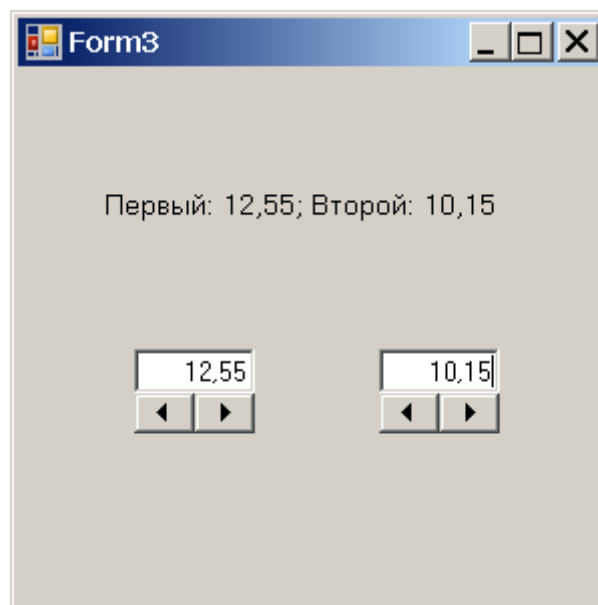
```
public Form3()
{
    InitializeComponent();

    numericScan1.DecimalPlaces =
        numericScan2.DecimalPlaces = 2;
    numericScan1.Increment =
        numericScan2.Increment = 0.01M;
}
```

Запустите "Упражнение 3" и испытайте работу кода. Изменяйте значения текстового поля клавиатурой и мышью, а также пользуйтесь клавишей Tab для смены фокуса ввода, испытав работу перегруженного метода OnLeave().

**Замечание.** Текстовое поле, ожидающее клавиатурный ввод числа типа Decimal, символом разделителя дробной части считает запятую.

Внешний результат будет приблизительно таким:



**Рис. 46**

## Индивидуальные задания

1. Доработайте программу, сделав скроллирующие кнопки не горизонтальными, а вертикальными.
2. Измените программу, разместив поля для вывода текста под скроллирующим кнопками.
3. Измените скорость генерации щелчка мыши до двух раз в секунду.
4. Разместите скроллирующую кнопку для увеличения значения над текстовым полем, а кнопку для уменьшения значения под ним.
5. Разместите скроллирующую кнопку для увеличения значения слева от текстового поля, а кнопку для уменьшения значения справа от него.
6. Разместите скроллирующие клавиши друг под другом снизу от текстового поля.
7. Разместите скроллирующие клавиши друг над другом сверху от текстового поля.
8. Реализуйте для каждого компонента NumericScan отдельно поле Label для вывода его значений.
9. Дополните программу возможностью подсчета суммы имеющихся в текстовых полях значений с выводом в отдельное окно.
10. Дополните программу возможностью подсчета разности имеющихся в текстовых полях значений с учетом знака с выводом в отдельное текстовое окно.
11. Дополните программу возможностью подсчета произведения имеющихся в текстовых полях значений с выводом в отдельное текстовое окно.
12. Сделайте скроллирующие кнопки правого компонента NumericScan синего цвета, а левого – зеленого.
13. Реализуйте закрытие программы при достижении одним из компонентов значения трех, а также диалоговое окно, обозначающее данное событие.
14. Отключите в правом компоненте NumericScan автоматическую генерацию щелчков мыши.
15. Реализуйте аналогичную программу, управление в которой осуществляется с помощью колесика мыши.
16. Измените программу так, чтобы значение в текстовой метке Label обновлялось на 0,5 секунды позднее, чем в поле TextBox.
17. Задание повышенной сложности (по желанию): Реализовать отдельные клавиши которые позволят хранить список mp3 файлов, при нажатии первый раз будет воспроизводиться музыка, при нажатии еще раз будет производиться отсановка музыки, а при наведении будет показываться подсказка с местом расположения файла.

## Приложение 1 Таблица цветов структуры Color

Цвета для объектов подсказки выберите из приведенной таблицы произвольно.

Color	Color Name	RGB Value	Color	Color Name	RGB Value
	LightPink	#FFB6C1		Lime	#00FF00
	Pink	#FFC0CB		ForestGreen	#228B22
	Crimson	#DC143C		Green	#008000
	LavenderBlush	#FFF0F5		DarkGreen	#006400
	PaleVioletRed	#DB7093		Chartreuse	#7FFF00
	HotPink	#FF69B4		LawnGreen	#7CFC00
	DeepPink	#FF1493		GreenYellow	#ADFF2F
	MediumVioletRed	#C71585		DarkOliveGreen	#556B2F
	Orchid	#DA70D6		YellowGreen	#9ACD32
	Thistle	#D8BFD8		OliveDrab	#6B8E23
	Plum	#DDA0DD		Beige	#F5F5DC
	Violet	#EE82EE		LightGoldenrodYellow	#FAFAD2
	Magenta	#FF00FF		Ivory	#FFFFFF0
	Fuchsia	#FF00FF		LightYellow	#FFFFE0
	DarkMagenta	#8B008B		Yellow	#FFFF00
	Purple	#800080		Olive	#808000
	MediumOrchid	#BA55D3		DarkKhaki	#BDB76B
	DarkViolet	#9400D3		LemonChiffon	#FFFACD
	DarkOrchid	#9932CC		PaleGoldenrod	#EEE8AA
	Indigo	#4B0082		Khaki	#F0E68C
	BlueViolet	#8A2BE2		Gold	#FFD700
	MediumPurple	#9370DB		Cornsilk	#FFF8DC
	MediumSlateBlue	#7B68EE		Goldenrod	#DAA520
	SlateBlue	#6A5ACD		DarkGoldenrod	#B8860B
	DarkSlateBlue	#483D8B		FloralWhite	#FFFAF0
	Lavender	#E6E6FA		OldLace	#FDF5E6
	GhostWhite	#F8F8FF		Wheat	#F5DEB3
	Blue	#0000FF		Moccasin	#FFE4B5
	MediumBlue	#0000CD		Orange	#FFA500
	MidnightBlue	#191970		PapayaWhip	#FFEFD5
	DarkBlue	#00008B		BlanchedAlmond	#FFEBCD
	Navy	#000080		NavajoWhite	#FFDEAD
	RoyalBlue	#4169E1		AntiqueWhite	#FAEBD7
	CornflowerBlue	#6495ED		Tan	#D2B48C

	LightSteelBlue	#B0C4DE		BurlyWood	#DEB887
	LightSlateGray	#778899		Bisque	#FFE4C4
	SlateGray	#708090		DarkOrange	#FF8C00
	DodgerBlue	#1E90FF		Linen	#FAF0E6
	AliceBlue	#F0F8FF		Peru	#CD853F
	SteelBlue	#4682B4		PeachPuff	#FFDAB9
	LightSkyBlue	#87CEFA		SandyBrown	#F4A460
	SkyBlue	#87CEEB		Chocolate	#D2691E
	DeepSkyBlue	#00BFFF		SaddleBrown	#8B4513
	LightBlue	#ADD8E6		Seashell	#FFF5EE
	PowderBlue	#B0E0E6		Sienna	#A0522D
	CadetBlue	#5F9EA0		LightSalmon	#FFA07A
	Azure	#F0FFFF		Coral	#FF7F50
	LightCyan	#E0FFFF		OrangeRed	#FF4500
	PaleTurquoise	#AFEEEE		DarkSalmon	#E9967A
	Cyan	#00FFFF		Tomato	#FF6347
	Aqua	#00FFFF		MistyRose	#FFE4E1
	DarkTurquoise	#00CED1		Salmon	#FA8072
	DarkSlateGray	#2F4F4F		Snow	#FFFAFA
	DarkCyan	#008B8B		LightCoral	#F08080
	Teal	#008080		RosyBrown	#BC8F8F
	MediumTurquoise	#48D1CC		IndianRed	#CD5C5C
	LightSeaGreen	#20B2AA		Red	#FF0000
	Turquoise	#40E0D0		Brown	#A52A2A
	Aquamarine	#7FFFD4		FireBrick	#B22222
	MediumAquamarine	#66CDAA		DarkRed	#8B0000
	MediumSpringGreen	#00FA9A		Maroon	#800000
	MintCream	#F5FFFA		White	#FFFFFF
	SpringGreen	#00FF7F		WhiteSmoke	#F5F5F5
	MediumSeaGreen	#3CB371		Gainsboro	#DCDCDC
	SeaGreen	#2E8B57		LightGrey	#D3D3D3
	Honeydew	#F0FFF0		Silver	#C0C0C0
	LightGreen	#90EE90		DarkGray	#A9A9A9
	PaleGreen	#98FB98		Gray	#808080
	DarkSeaGreen	#8FBC8F		DimGray	#696969
	LimeGreen	#32CD32		Black	#000000

