10. Элементы управления.

Предназначены для организации расширенного диалога:

- Организация выбора
- Объединение компонентов в группу
- Наглядная визуализация

1. CheckBox

Компонент **CheckBox** (выключатель) может находиться в одном из двух состояний: выбранном или не выбранном (вкл. – выкл.). Переключается щелчком мыши или пробелом.

Предназначен ввода пользователем булевых данных («да» или «нет»).

Два способа работы с компонентом:

1. Отслеживать значение свойства `Checked` – состояние отметки. **True** – элемент выбран, **False** – не выбран.
2. Реагировать на переключение – события **Click** или **CheckedChanged**.

```
private void checkBox1_Click(object sender, EventArgs e)
        {
            if (checkBox1.Checked) BackColor = Color.Red;
               else BackColor = Color.Yellow;
        }
```

Свойства компонента **CheckBox** предназначены для его оформления.

Свойство	*Тип*	*Описание*
Text	string	Надпись около выключателя
TextAlign		Положение текста внутри компонента. По умолчанию – **MiddleRight**.
CheckAlign		Положение выключателя внутри компонента (аналогично **TextAlign**). По умолчанию – **MiddleLeft**.
Image		Рисунок около кнопки (вместе с надписью)
ImageAlign		Положение картинки внутри компонента. По умолчанию – **MiddleRight**.

2. RadioButton

Компонент **RadioButton** (переключатель) также может находиться в одном из двух состояний: выбранном или невыбранном (вкл. – выкл.). Переключается выбором другого переключателя.

Предназначен выбора пользователем одного значения из нескольких.

Два способа работы с компонентом:

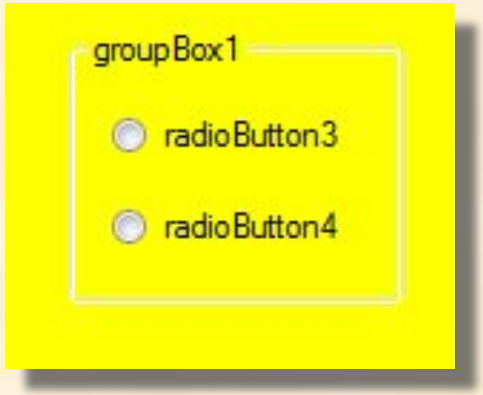
1. Отслеживать значение свойства `Checked` – состояние отметки. **True** – элемент выбран, **False** – не выбран.
2. Реагировать на переключение – событие **Click** **Click** или **CheckedChanged**.

```
private void radioButton1_Click(object sender, EventArgs e)
    {
        if (radioButton1.Checked) BackColor = Color.Red;
    }
```

Набор свойств аналогичен **CheckBox**.

3. GroupBox

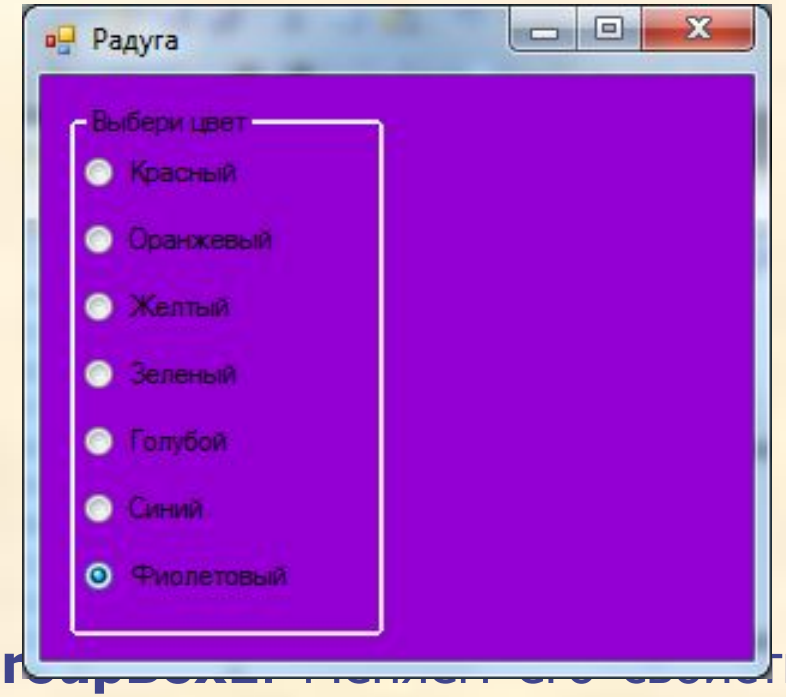
Компонент **GroupBox** (группирующая рамка) представляет собой контейнер, в котором можно размещать другие элементы управления.



Обычно работает с компонентами типа **RadioButton**. При этом компоненты **RadioButton**, расположенные в отдельном **GroupBox**, работают независимо от других компонентов **RadioButton**.

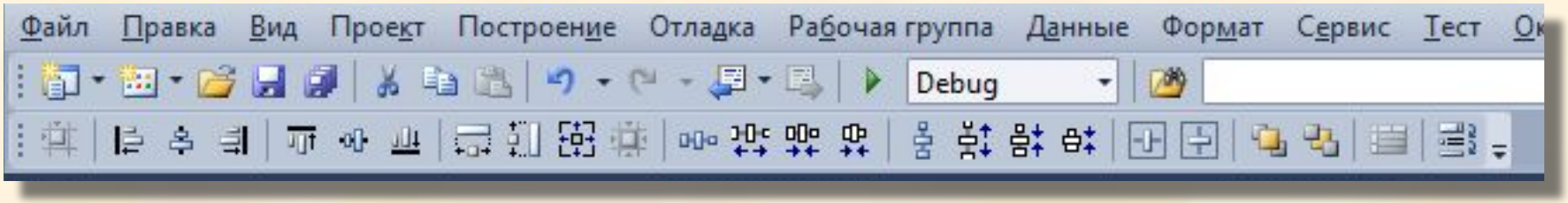
Основное свойство – **Text** – заголовок сверху.

Пример. Приложение позволяет выбирать цвет формы из заданного набора.



Решение.

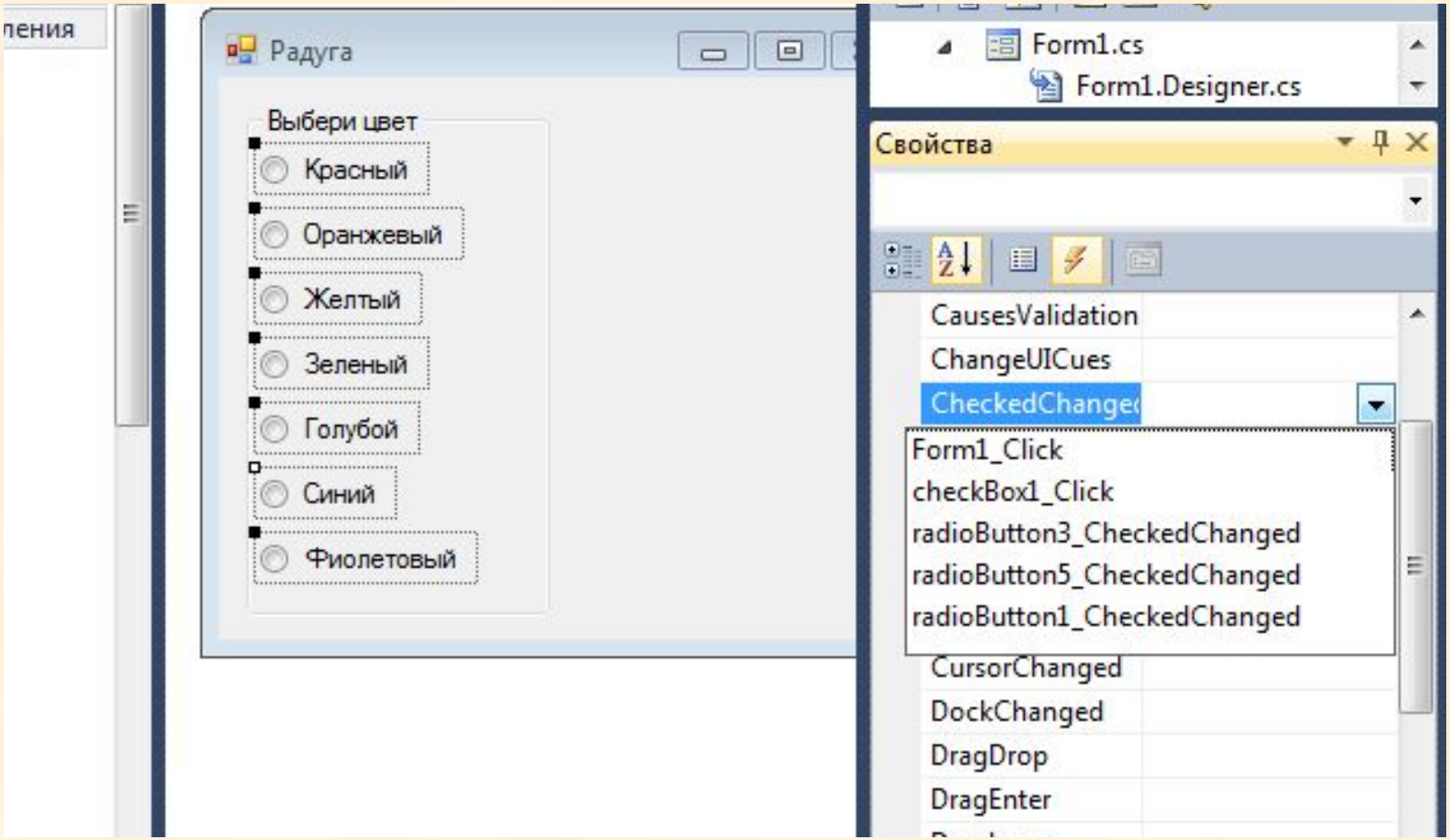
1. Располагаем на форме контейнер **gr**[illegible]во **Text** (на «**Выбираем цвет**»).
2. Располагаем на компоненте **groupBox1** 7 компонентов **RadioButton**. Меняем их свойство **Text**.
3. Настраиваем их расположение кнопками выравнивания (панель инструментов «Макет»):



4. Для компонента **radioButton1** пишем обработчик события **CheckedChanged** (изменение свойства **Checked**):

```
private void radioButton1_CheckedChanged(object sender, EventArgs e)
        {
            RadioButton rb = (RadioButton)sender;
            if (rb.Checked)
              switch (rb.Text)
              {
                case "Красный":
                    { BackColor = Color.LightCoral; break; }
                case "Оранжевый":
                    { BackColor = Color.Orange; break; }
                case "Желтый":
                    { BackColor = Color.Yellow; break; }
                case "Зеленый":
                    { BackColor = Color.LightGreen; break; }
                case "Голубой":
                    { BackColor = Color.LightBlue; break; }
                case "Синий":
                    { BackColor = Color.SteelBlue; break; }
                case "Фиолетовый":
                    { BackColor = Color.DarkViolet; break; }
              }
        }
```

5. Выделив остальные радио-кнопки, присваиваем их событию CheckedChanged написанный выше обработчик.



6. Приложение готово.

В данном примере можно для каждой радио-кнопки прописать отдельный обработчик события **CheckedChanged** (изменение свойства **Checked**), например:

```
private void radioButton1_CheckedChanged(object sender, EventArgs e)
        {
            if (radioButton1.Checked)
                  BackColor = Color.LightCoral;
        }
private void radioButton2_CheckedChanged(object sender, EventArgs e)
        {
            if (radioButton2.Checked)
                  BackColor = Color.Orange;
        }
```

И т.д. для остальных компонент.

4. Panel

Компонент **Panel** (прямоугольная площадка) представляет собой контейнер, в котором можно размещать другие элементы управления.

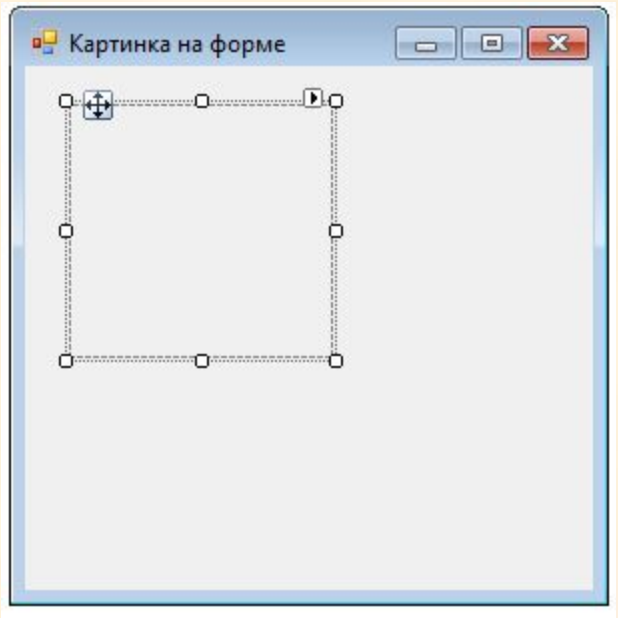
Специфическое свойство – **BorderStyle** – указывает на вид границы:

- **None** - нет границ
- **FixedSingle** – обычная рамка
- **Fixed3D** – объёмная рамка

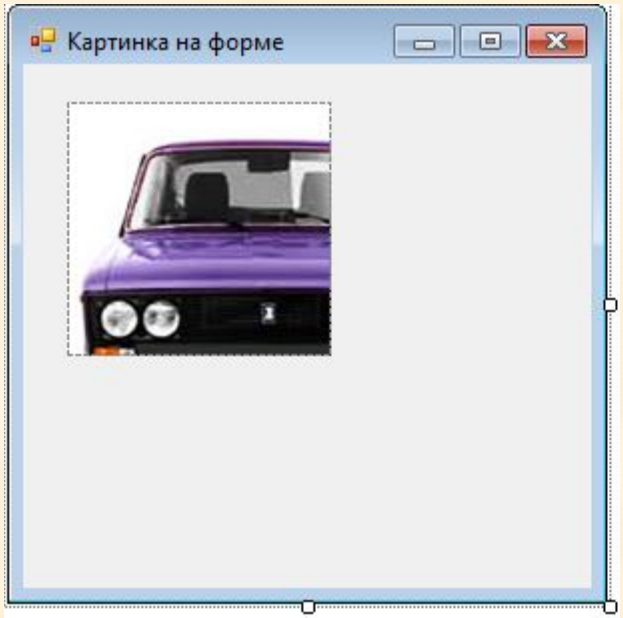
Пример. Используя компонент **Panel**, разместить на форме изображение из файла “2106.jpg” (с рабочего стола).

Решение.

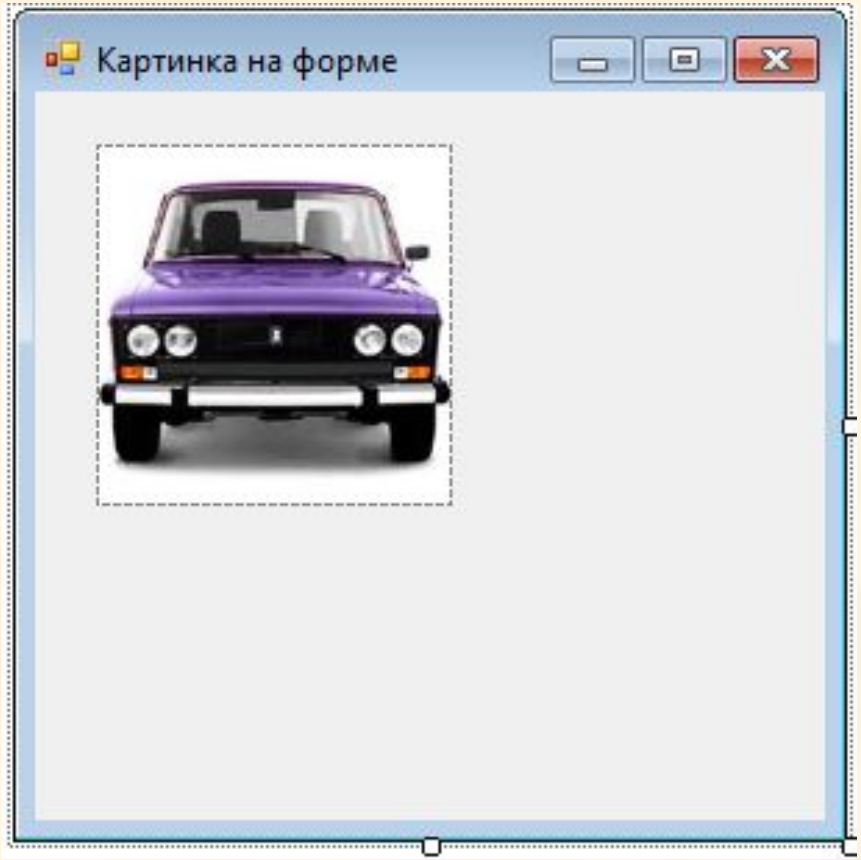
1. Располагаем на форме в предполагаемой позиции рисунка компонент **Panel**.

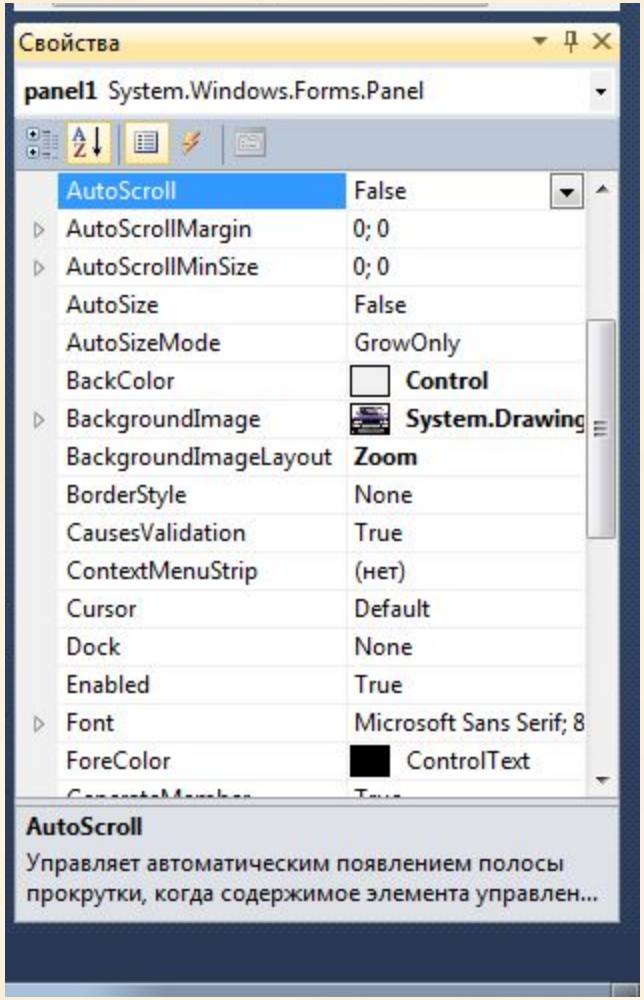


2. Свойству **BackgroundImage** компонента **Panel** ставим в соответствие выбранный нами рисунок (выполнив импорт из файла).



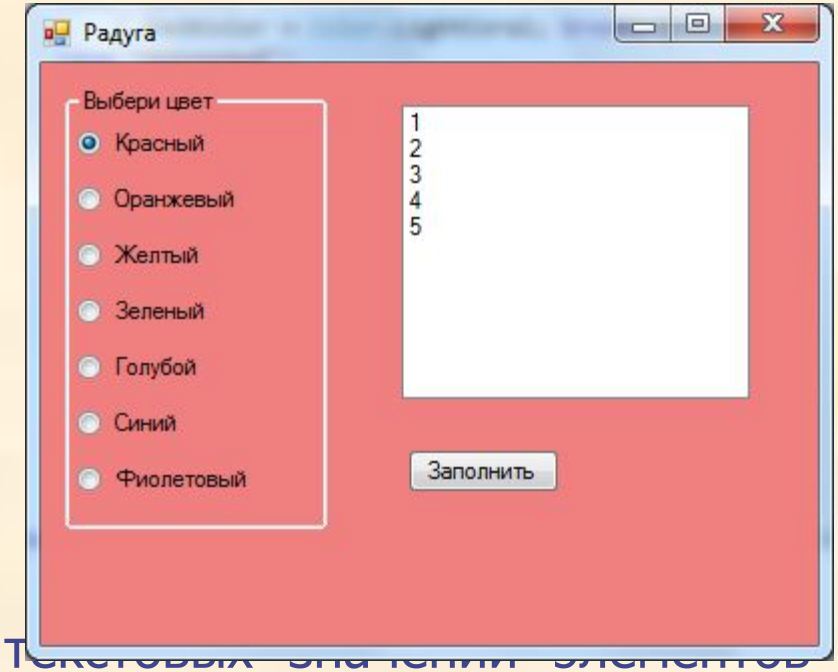
3. Выберем правило размещения рисунка на компоненте **Panel**, правильно настроив его свойство **BackgroundImageLayout** (в данном примере **BackgroundImageLayout=Zoom**).





5. ListBox

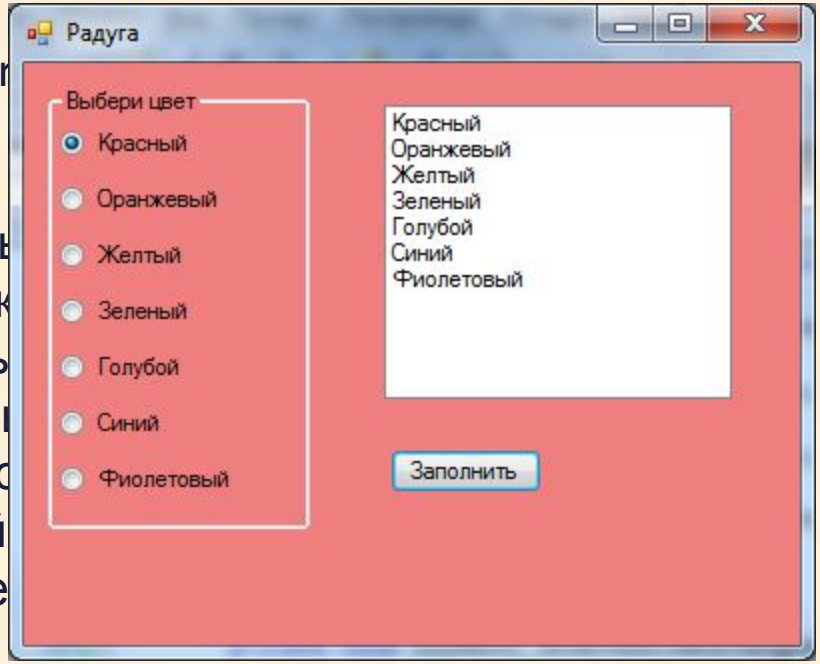
Компонент **ListBox** (простой список) представляет собой список, в котором можно выбрать (выделить) один или несколько элементов.



Для хранения текстовых значений элементов списка используется свойство **Items**. Заполнение выполняется либо на этапе макета, либо программно.

Так, в нашем примере заменить список строк можно следующим образом:

```
private void button1_Click(object ser
        {
            listBox1.Items.Clear();
            listBox1.Items.Add("Красны
            listBox1.Items.Add("Оранж
            listBox1.Items.Add("Желты
            listBox1.Items.Add("Зелени
            listBox1.Items.Add("Голубо
            listBox1.Items.Add("Синий
            listBox1.Items.Add("Фиоле
        }
```



Или так:

```
private void button1_Click(object sender, EventArgs e)
    {
        listBox1.Items.Clear();
        listBox1.Items.AddRange( new object[]
            { "Красный", "Оранжевый", "Желтый", "Зеленый",
              "Голубой", "Синий", "Фиолетовый" } );
     }
```

Свойства компонента **ListBox**

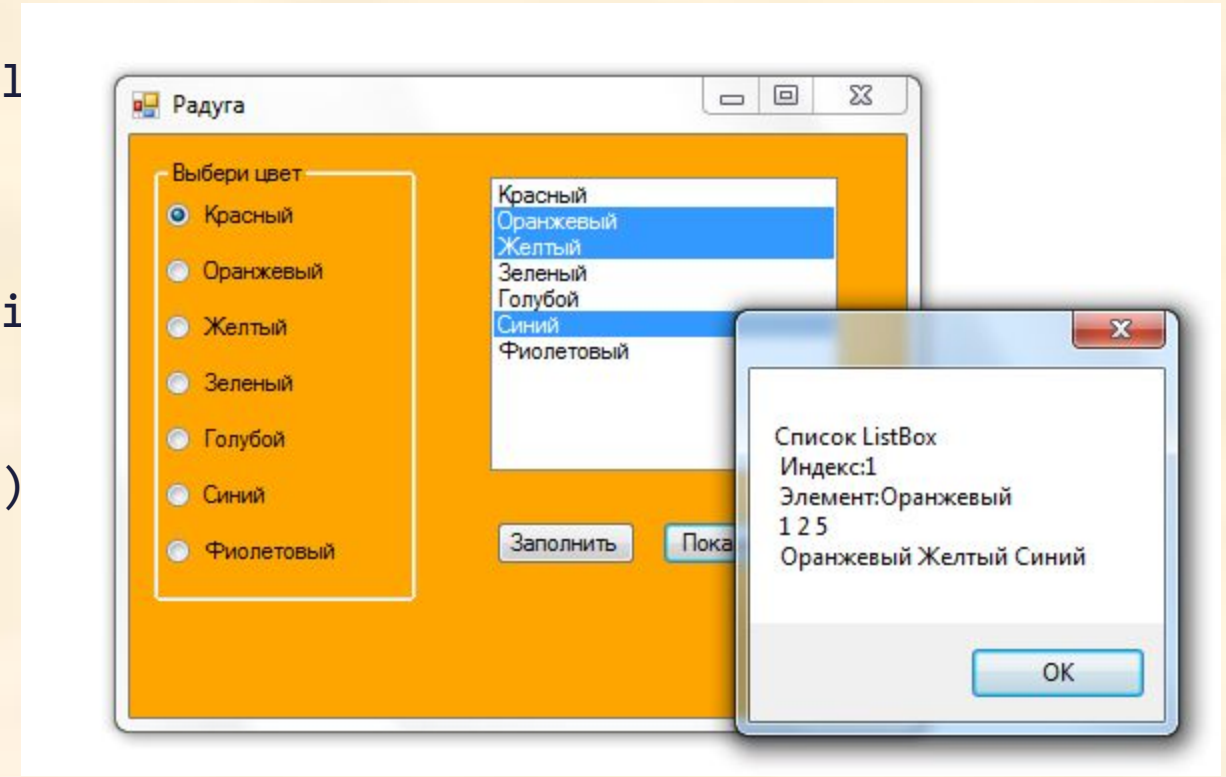
Свойство	*Тип*	*Описание*
Items	коллекция	Элементы списка – текстовые строки
Items.Count	int	Количество элементов списка
SelectedIndex	int	Номер выбранного элемента списка. Если ни один не выбран, то значение свойства = -1.
SelectedItem	string	Выделенная строка.
Sorted	bool	Если равно True, то выполняется автоматическая сортировка.
SelectionMode		Режим выбора элементов: **One** – только 1 элемент **MultiSelect** – несколько элементов щелчком мыши **MultiExtended** – несколько элементов щелчком мыши с **Ctrl**, или выделением диапазона щелчками мыши с **Shift** на первом и последнем элементах.
MultiColumn	bool	Вывод элементов в несколько колонок (с горизонтальной полосой прокрутки)

Будем менять цвет формы по щелчку мышкой по элементу списка:

```
private void listBox1_SelectedIndexChanged
             (object sender, EventArgs e)
        {
            switch (listBox1.SelectedIndex)
            {
                case 0: { BackColor = Color.LightCoral; break; }
                case 1: { BackColor = Color.Orange; break; }
                case 2: { BackColor = Color.Yellow; break; }
                case 3: { BackColor = Color.LightGreen; break; }
                case 4: { BackColor = Color.LightBlue; break; }
                case 5: { BackColor = Color.SteelBlue; break; }
                case 6: { BackColor = Color.DarkViolet; break; }

            }
        }
```

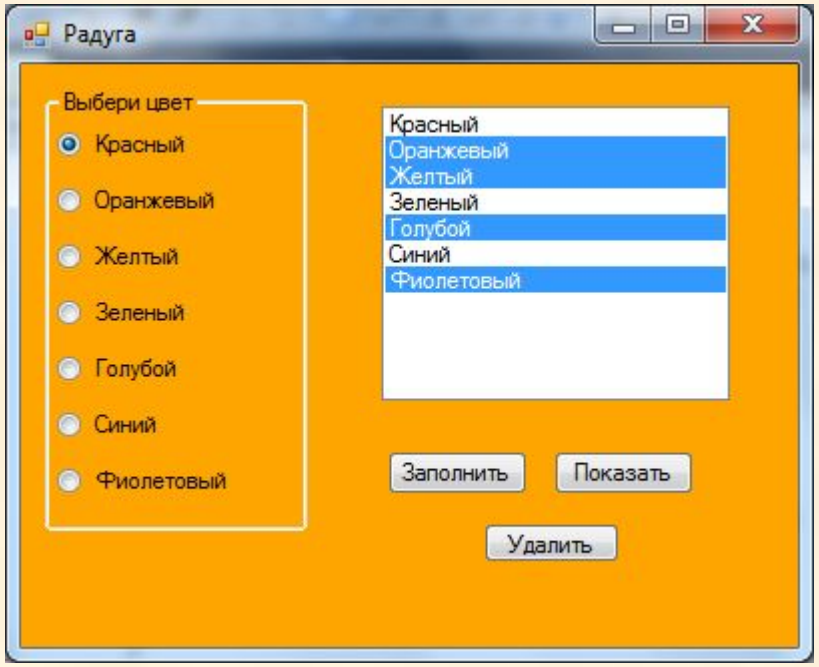
Для демонстрации множественного выбора добавим кнопку «показать выбранные», и присвоим ей обработчик события **Cilck**:

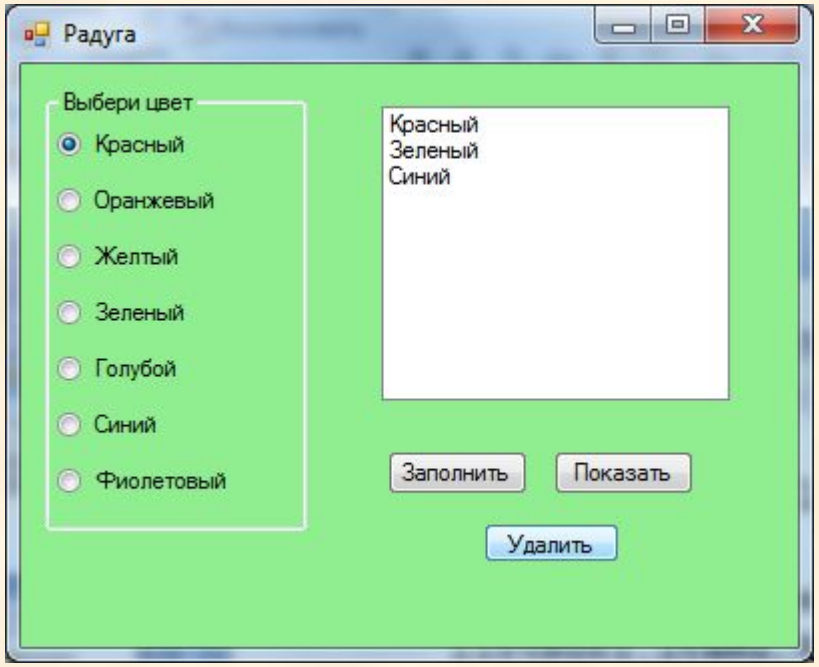
```
private void button2_Click(object sender, EventArgs e)
  {
     string st;
     st = "Список ListBox";
     st += "\n Индекс:"  + listBox1.SelectedIndex;
     st += "\n Элемент:" + listBox1.SelectedItem;
     st += "\n";

     foreach (int i in l
        st += " " + i;
     st += "\n";

     foreach (string s i
        st += " " + s;

     MessageBox.Show(st)
  }
```



Для демонстрации удаления выделенных элементов списка добавим кнопку «Удалить» и присвоим ей обработчик события **Cilck**:

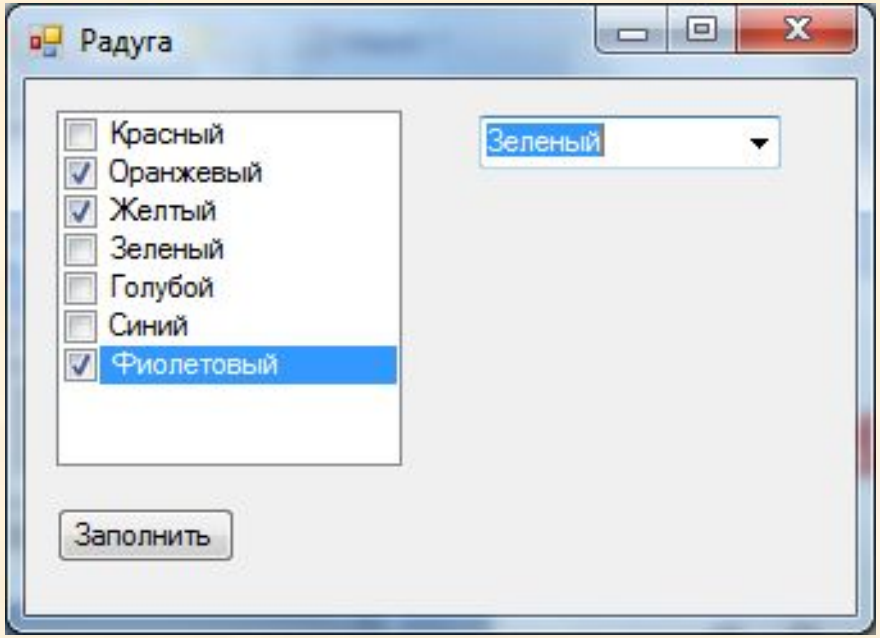
```
private void button3_Click(object sender, EventArgs e)
  {
     while (listBox1.SelectedIndices.Count>0)
       listBox1.Items.RemoveAt(listBox1.SelectedIndices[0]);
  }
```





6. CheckListBox

Компонент **CheckListBox** (список с пометками) представляет собой список, в котором выбор элементов выполняется установкой пометки.



Набор свойств аналогичен компоненту **ListBox**. При этом некоторые свойства меняют название (множественный выбор):

```
SelectedItems -> CheckedItems
SelectedIndices -> CheckedIndices
```

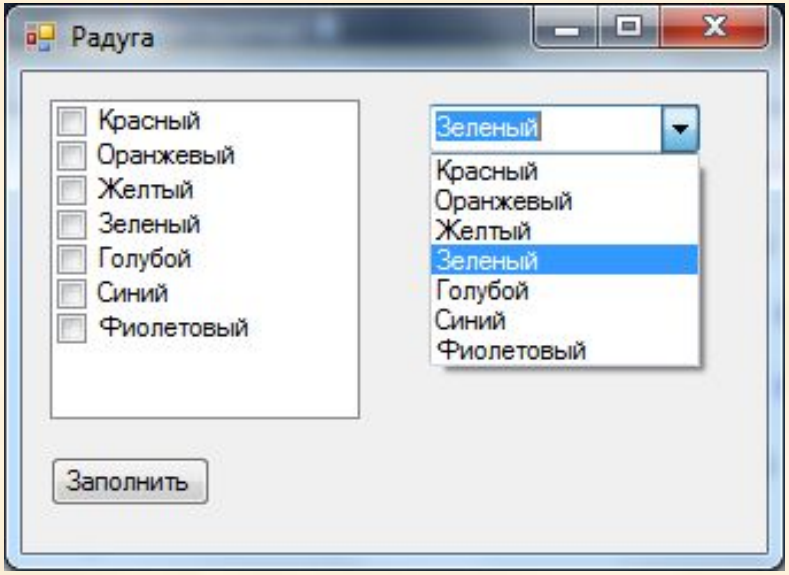
Работа с **CheckListBox** аналогична работе с **ListBox**.

Например, заполнить список можно так:

```
checkedListBox1.Items.Clear();
checkedListBox1.Items.Add("Красный");
checkedListBox1.Items.Add("Оранжевый");
checkedListBox1.Items.Add("Желтый");
checkedListBox1.Items.Add("Зеленый");
checkedListBox1.Items.Add("Голубой");
checkedListBox1.Items.Add("Синий");
checkedListBox1.Items.Add("Фиолетовый");
```

7. ComboBox

Компонент **ComboBox** (комбинированный список) представляет собой комбинацию простого списка и однострочного текстового редактора.



Для данного списка используются свойства и события, которые применялись для **ListBox** и **TextBox**.

Свойства компонента **ComboBox**:

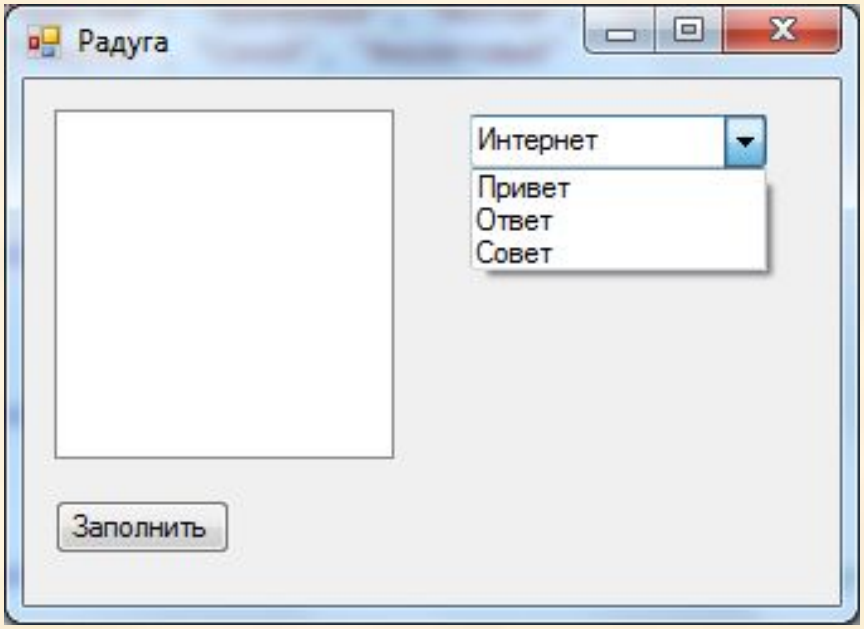
Свойство	*Тип*	*Описание*
Text	string	Текст в строке редактирования
DropDownStyle		Вид компонента: **Simple** – поле ввода со списком **DropDown** – поле ввода и раскрывающийся список **DropDownList** – просто раскрывающийся список
DropDownWidth	int	Ширина раскрывающейся части списка
MaxDropDownItems	int	Количество отображаемых элементов в раскрывающемся списке.

При выборе очередного элемента списка **ComboBox** происходит вывод его в строку редактирования **ComboBox.Text**.

При изменении свойства **ComboBox.Text** генерируется событие **TextChanged**, в обработчике которого можно прописать соответствующую реакцию.

Пример: Если ввод текста в поле ввода завершить нажатием клавиши <ENTER>, то введенный текст добавится к списку элементов:

```
private void comboBox1_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Enter)
            {
                comboBox1.Items.Add(comboBox1.Text);
                comboBox1.Text = "";
                e.Handled=true;
            }
        }
```



8. NumericUpDown

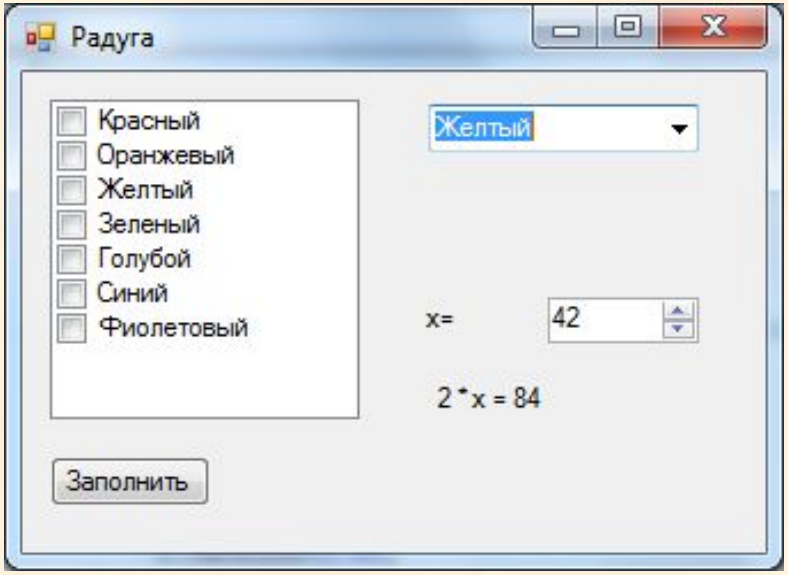
Компонент **NumericUpDown** предназначен для ввода или выбора числового значения из некоторого диапазона.

При этом компонент имеет ряд специфических свойств:

Свойство	*Тип*	*Описание*
Value	decimal	Текущее значение числа.
Maximum	decimal	Максимально допустимое значение числа.
Minimum	decimal	Минимально допустимое значение числа.
Increment	decimal	Шаг изменения значения числа при нажатии треугольников.
TextAlign		Расположение текста в поле редактирования.
ReadOnly	bool	Если true, то можно изменить текст, пользуясь только кнопками "СТРЕЛКА ВВЕРХ" и "СТРЕЛКА ВНИЗ".

Пример: При изменении значения числа в **NumericUpDown** будет вычисляться его удвоенное значение:

```
private void numericUpDown1_ValueChanged
                              (object sender, EventArgs e)
  {
      int res = Convert.ToInt32(numericUpDown1.Value) * 2;
      label2.Text = "2 * x = " + Convert.ToString(res);
  }
```



8. ProgressBar

Компонент **ProgressBar** предназначен для наглядного представления скорости протекания некоторого процесса в виде доли некоторого числового значения внутри диапазона.

Свойство	*Тип*	*Описание*
Value	int	Текущее значение числа.
Maximum	int	Максимально допустимое значение **Value**.
Minimum	int	Минимально допустимое значение **Value**.

9. TrackBar

Компонент **TrackBar** предназначен визуального выбора значения в виде его доли внутри некоторого диапазона.

Свойство	*Тип*	*Описание*
Value	int	Текущее значение числа.
Maximum	int	Максимально допустимое значение **Value**.
Minimum	int	Минимально допустимое значение **Value**.
SmallChange	int	Минимальный шаг изменения значения **Value**.
LargeChange	int	Шаг изменения значения **Value** при щелчке по полосе.

Пример: Применение «бегунков» позволяет настроить цвет формы по трём его компонентам:

```
private void trackBar1_ValueChanged
                              (object sender, EventArgs e)
        {
            int r = trackBar1.Value;
            int g = trackBar2.Value;
            int b = trackBar3.Value;
            BackColor = Color.FromArgb(r, g, b);
        }  // применить ко всем трём бегункам
```