

Министерство науки и высшего образования Российской Федерации  
Казанский национальный исследовательский технический университет –  
КАИ им. А.Н. Туполева

Институт компьютерных технологий и защиты информации  
Отделение СПО ИКТЗИ «Колледж информационных технологий»

## **ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЕ**

Методические указания к лабораторной работе №7

Тема: «Введение в разработку графических пользовательских интерфейсов с  
использованием технологии Windows Forms»

Казань 2022

Составитель преподаватель СПО ИКТЗИ Мингалиев Заид Зульфатович

Методические указания к лабораторным работам по дисциплине «ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЕ» предназначены для студентов направления подготовки 09.02.07 «Информационные системы и программирование»

## **ПРОЦЕСС СДАЧИ ВЫПОЛНЕННОЙ РАБОТЫ**

По итогам выполнения работы студент:

1. демонстрирует преподавателю правильно работающие программы;
2. демонстрирует приобретённые знания и навыки отвечает на пару небольших вопросов преподавателя по составленной программе, возможностям её доработки;
3. демонстрирует отчет по выполненной лабораторной работе.

Итоговая оценка складывается из оценок по трем указанным составляющим.

## **ЛАБОРАТОРНАЯ РАБОТА №7**

### **ТЕМА: «ВВЕДЕНИЕ В РАЗРАБОТКУ ГРАФИЧЕСКИХ ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ WINDOWS FORMS»**

#### **ЦЕЛЬ РАБОТЫ**

Научиться размещать и настраивать внешний вид элементов управления на форме и создавать обработчики событий.

#### **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

Для Windows-приложений, ориентированных на выполнение в общезыковой исполняющей среде (CLR), в качестве основы построения графического пользовательского интерфейса используются классы Windows Forms из библиотек .NET Framework, т.е. Windows Forms – это набор средств для создания Windows-приложений, выполняющихся в среде CLR.

Средства Windows Forms обеспечивают быструю разработку пользовательского интерфейса, поскольку он создается из стандартных элементов, а соответствующий программный код генерируется автоматически. Затем требуется лишь доработать сгенерированный код, чтобы добиться необходимой функциональности.

Применяя Windows Forms, можно построить удобный пользовательский интерфейс, в том числе главное окно приложения, выбирая подходящие элементы управления, с которыми взаимодействует пользователь.

Чтобы создать Windows-приложение на основе Windows Forms, следует выбрать команду главного меню «Файл» (File) → «Создать» (New) → «Проект» (Project), затем выбрать тип проекта «Рабочий стол» (Windows Desktop), а в качестве шаблона проекта указать Windows Forms Application (Рисунок 1).

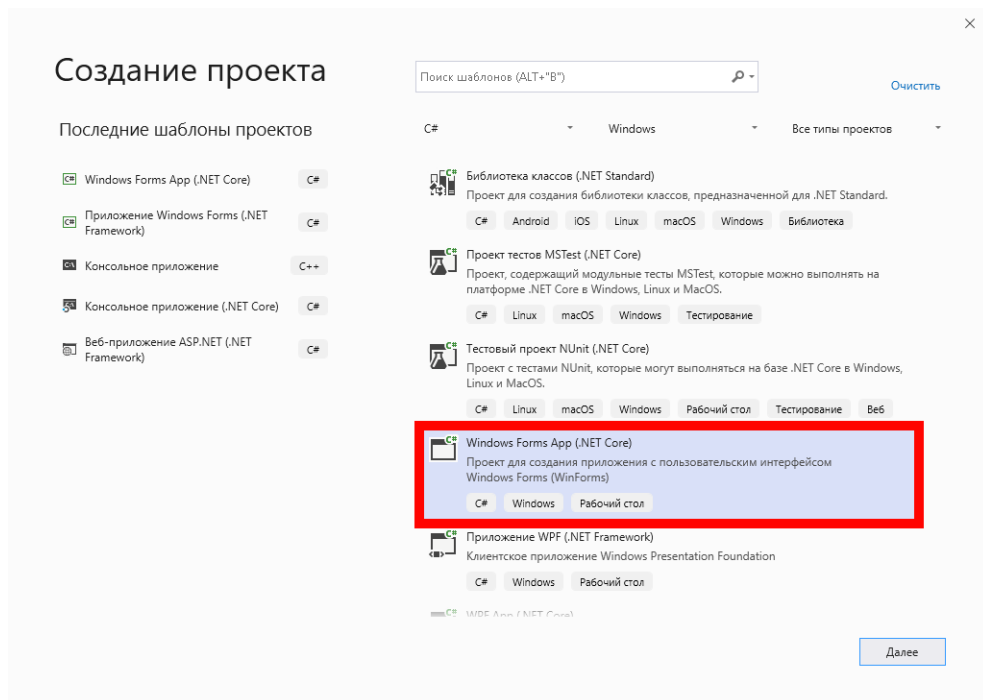


Рисунок 1

После этого следует ввести имя проекта, например, PR1, и указать папку для хранения проекта.

После этого Visual Studio откроет наш проект с созданными по умолчанию файлами:

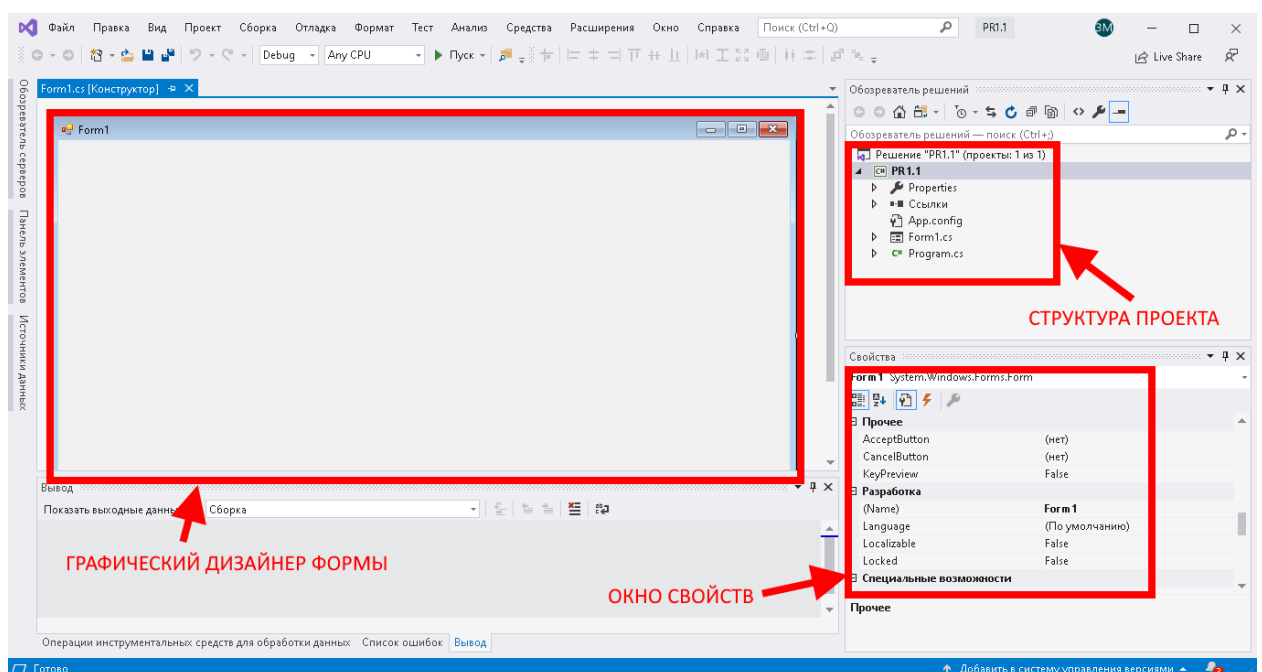


Рисунок 2

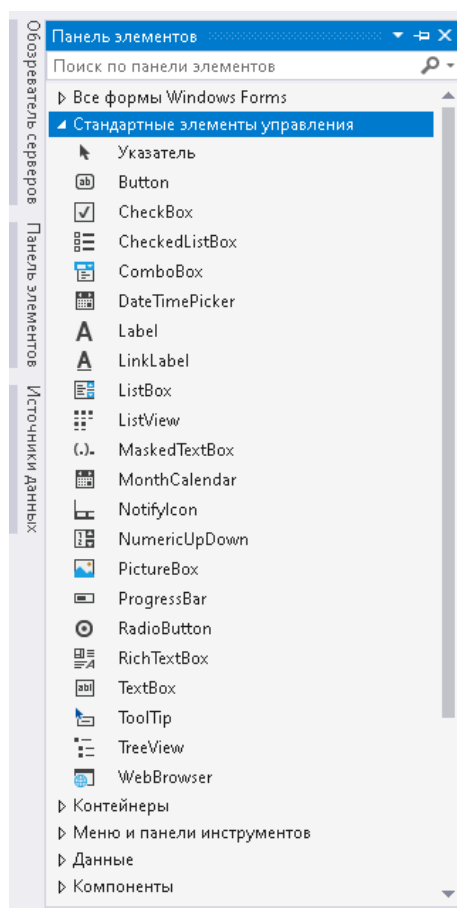
Большую часть пространства Visual Studio занимает графический дизайнер, который содержит форму будущего приложения. Пока она пуста и

имеет только заголовок Form1. Справа находится окно файлов решения/проекта - Solution Explorer (Обозреватель решений). Там и находятся все связанные с нашим приложением файлы, в том числе файлы формы Form1.cs.

Внизу справа находится окно свойств - Properties. Так как в данный момент выбрана форма как элемент управления, то в этом поле отображаются свойства, связанные с формой.

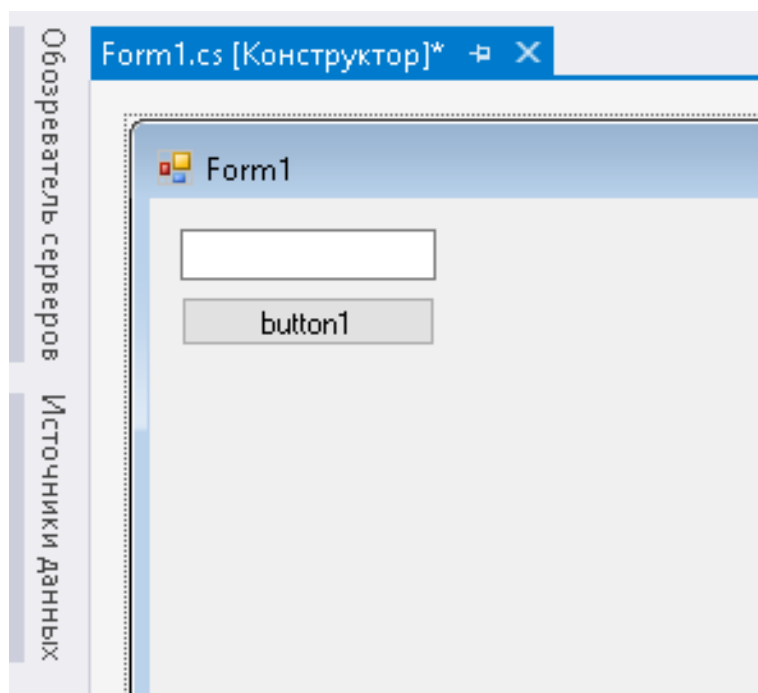
### **Размещение элементов управления на форме**

Для размещения различных элементов управления на форме используется Панель инструментов (Toolbox). Панель элементов содержит элементы управления, сгруппированные по типу. Каждую группу элементов можно свернуть, если она в настоящий момент не нужна. (Рисунок 3)



*Рисунок 3*

Чтобы добавить элементы управления на форму, необходимо щелкнуть на добавляемый элемент в панели инструментов, а затем щелкнуть в нужном месте формы. После этого элемент появится на форме (Рисунок 4).



*Рисунок 4 – Форма с добавленными элементами управления (TextBox и Button)*

Элемент можно перемещать по форме, схватившись за него левой кнопкой мыши. Если элемент управления позволяет изменять размеры, то на соответствующих его сторонах появятся белые кружки, ухватившись за которые и можно изменить размер. При размещении элемента управления на форме, его можно выделить щелчком мыши и получить доступ к его свойствам в окне свойств.

### **Размещение строки ввода**

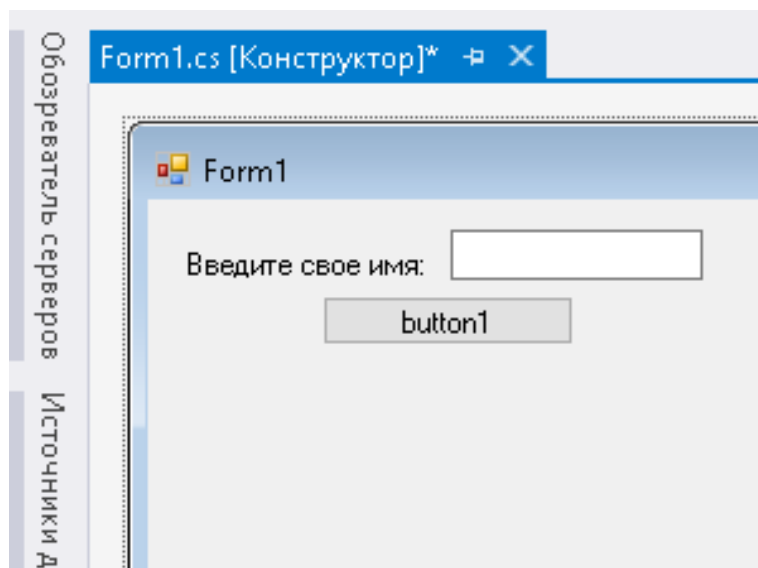
Если необходимо ввести из формы в программу или вывести в форму информацию, которая вмещается в **одну** строку, используют окно однострочного редактора текста, представляемого элементом управления TextBox.

Создадим поле для ввода имени пользователя. Выберем на панели инструментов пиктограмму с названием «TextBox», щелкните мышью в том месте формы, где хотите ее разметить. После размещения элемента можно в

тексте программы использовать переменную `textBox1`, которая соответствует добавленному элементу управления. В этой переменной, в свойстве `Text` будет содержаться строка символов (типа `string`) и отображаться в соответствующем окне `TextBox`. С помощью окна свойств можно установить шрифт и размер символов, отражаемых в строке `TextBox` (свойство `Font`).

### **Размещение надписей**

На форме могут размещаться пояснительные надписи. Для размещения таких надписей на форму используется элемент `Label`. Выберите на панели инструментов пиктограмму с названием `Label`, щелкните на ней мышью. После этого в нужном месте формы щелкните мышью, появится надпись `label1`. Щелкнув на ней мышью, можно отрегулировать размер и, изменив свойство `Text` в окне свойств, введите строку, например, «Введите свое имя:», а также выберите размер символов (свойство `Font`). Добавленный элемент представлен в форме на рисунке 5.



*Рисунок 5 – Форма с добавленным элементом `Label`*

В тексте программы можно обращаться к новой переменной типа `Label`. В ней хранится пояснительная строка, которую можно изменять в процессе работы программы.

### **Написание программы обработки события**

С каждым элементом управления на форме и с самой формой могут происходить события во время работы программы. Например, с кнопкой



может произойти событие – нажатие кнопки, а с окном, которое проектируется с помощью формы, может произойти ряд событий: создание окна, изменение размера окна, щелчок мыши на окне и т.п. Эти события могут обрабатываться программно. Для обработки таких событий используются специальные методы – обработчики событий. Создать такие методы можно двумя способами.

Первый способ – создать обработчик для события по умолчанию. Например, при нажатии кнопки на форме таким образом создается обработчик события нажатия.

### **Написание программы обработки события нажатия кнопки**

Поместите на форму кнопку, которая описывается элементом управления Button. С помощью окна свойств измените заголовок (Text) на слово «Привет» или другое по вашему желанию. Отрегулируйте положение и размер кнопки.

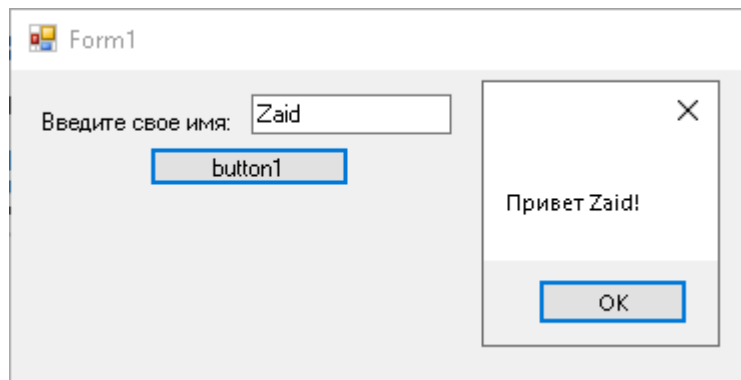
После этого два раза щелкните мышью на кнопке, появится текст программы:

```
private void button1_Click(object sender, EventArgs e)
{
}
}
```

Этот метод и будет являться обработчиком события нажатия кнопки. Вы можете добавлять свой код между скобками {...}. Например, наберите:


```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Привет " + textBox1.Text + "!");
}
```

При нажатии на кнопку выведется окно с сообщением «Привет», которое соединено с текстом, записанным пользователем в textBox1 (Рисунок 6).



*Рисунок 6 – Окно с сообщением, появившееся при нажатии кнопки button1*

### **Написание программы обработки события загрузки формы**

Второй способ создания обработчика события заключается в выборе соответствующего события для выделенного элемента на форме. При этом используется окно свойств и его закладка . Рассмотрим этот способ. Выделите форму щелчком по ней, чтобы вокруг нее появилась рамка из точек. В окне свойств найдите событие Load. Щелкните по данной строчке дважды мышкой. Появится метод:

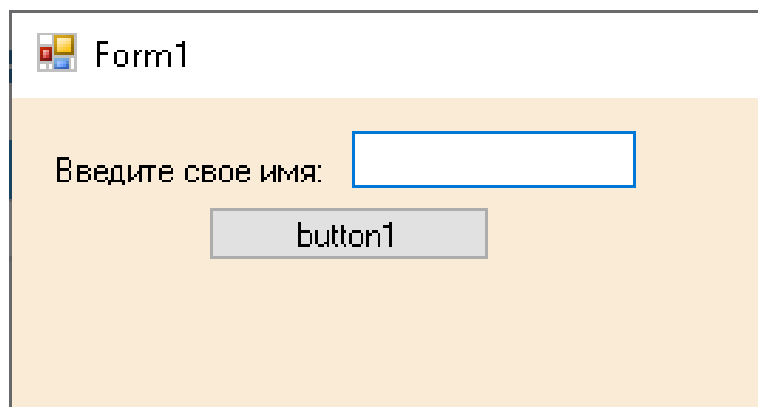
```
private void Form1_Load(object sender, EventArgs e)
{

}
```

Между скобками {...} вставим текст программы:

```
private void Form1_Load(object sender, EventArgs e)
{
    BackColor = Color.AntiqueWhite;
}
```

Свойство `BackColor` позволяет изменить цвет фона. Результат выполнения данного метода представлен на рисунке 7.



*Рисунок 7 – Форма с измененным фоном*

Каждый элемент управления имеет свой набор обработчиков событий, однако некоторые из них присущи большинству элементов управления. Наиболее часто применяемые события описаны ниже:

1) Load – событие, возникающее при загрузке формы. В обработчике данного события следует задавать действия, которые должны происходить в момент создания формы, например, установка начальных значений.

2) KeyPress – событие, возникающее при нажатии кнопки на клавиатуре. Параметр `e.KeyChar` имеет тип `char` и содержит код нажатой клавиши (например, клавиша Enter клавиатуры имеет код #13, клавиша Esc - #27 и т.д.). Обычно это событие используется в том случае, когда необходима реакция на нажатие одной из клавиш.

3) KeyDown – событие, возникающее при нажатии кнопки на клавиатуре. Обработчик этого события получает информацию о нажатой клавише и состоянии клавиш Shift, Alt и Ctrl, а также о нажатой кнопке мыши. Информация о клавише передается параметром `e.KeyCode`, который представляет собой перечисление `Keys` с кодами всех клавиш, а информацию о клавишах-модификаторах Shift и др. можно узнать из параметра `e.Modifiers`.

4) KeyUp – является парным событием для KeyDown и возникает при отпускании ранее нажатой клавиши.

5) Click – возникает при нажатии кнопкой мыши в области элемента управления.

6) DoubleClick – возникает при двойном нажатии кнопки мыши в области элемента управления.

**Важное примечание** - если какой-то обработчик был добавлен по ошибке или больше не нужен, то для его удаления нельзя просто удалить программный код обработчика. Сначала нужно удалить строку с именем обработчика в окне свойств в закладке с событиями. В противном случае программа может перестать компилироваться и даже отображать форму в дизайнера VS.

### **Динамическое изменение свойств**

Свойства элементов на окне могут быть изменены динамически во время выполнения программы. Например, можно изменять текст надписи или цвет формы. Изменение свойств происходит внутри обработчика события (например, обработчика события нажатия на кнопку). Для этого используют оператор присвоения вида:

`<имя элемента>.<свойство> = <значение>;`

Например,

`label1.Text = "Hello";`

`<имя элемента>` определяется на этапе проектирования формы, при размещении элемента управления на форме. Например, при размещении на форме ряда элементов TextBox, эти элемента получают имена `textBox1`, `textBox2`, `textBox3` и т.д., которые могут быть заменены в окне свойств в свойстве (Name) для текущего элемента. Список свойств для конкретного элемента можно посмотреть в окне свойств.

Если требуется изменить свойства формы, то никакое имя элемента перед точкой вставлять не нужно, как и саму точку. Например, чтобы задать цвет формы, нужно просто написать:

`BackColor = Color.Green;`

### **Ввод и вывод данных в программу**

Рассмотрим один из способов ввода данных через элементы, размещенные на форме. Для ввода данных чаще всего используют элемент

управления TextBox, через обращение к его свойству Text. Свойство Text хранит в себе строку введенных символов. Поэтому данные можно считать следующим образом:

```
private void button1_Click(object sender, EventArgs e)
{
    string s = textBox1.Text;
}
```

Однако со строкой символов трудно производить арифметические операции, поэтому лучше всего при вводе числовых данных перевести строку в целое или вещественное число. Для этого у типов int и double существуют методы Parse для преобразования строк в числа. С этими числами можно производить различные арифметические действия. Таким образом, предыдущий пример можно переделать следующим образом:

Перед выводом числовые данные следует преобразовать назад в строку. Для этого у каждой переменной существует метод ToString(), который возвращает в результате строку с символьным представлением значения. Вывод данных можно осуществлять в элементы TextBox или Label, используя свойство Text. Например,

```
private void button1_Click(object sender, EventArgs e)
{
    string s = textBox1.Text;
    int a = int.Parse(s);
    int b = a * a;
    label1.Text = b.ToString();
}
```

### **Кнопки-переключатели**

При создании программ в Visual Studio для организации разветвлений часто используются элементы управления в виде кнопок-переключателей. Кнопки-переключатели представлены в панели инструментов как RadioButton. Состояние такой кнопки (включено-выключено) визуально отражается на форме, а в программе можно узнать его помощью свойства Checked: если кнопка включена, это свойство будет содержать True, в противном случае

False. Если пользователь выбирает один из вариантов переключателя в группе, все остальные автоматически отключаются.

Группируются радиокнопки с помощью какого-либо контейнера – часто это бывает элемент `GroupBox`. Радиокнопки, размещенные в различных контейнерах, образуют независимые группы.

Контейнер с кнопками переключателями представлен на рисунке 8.

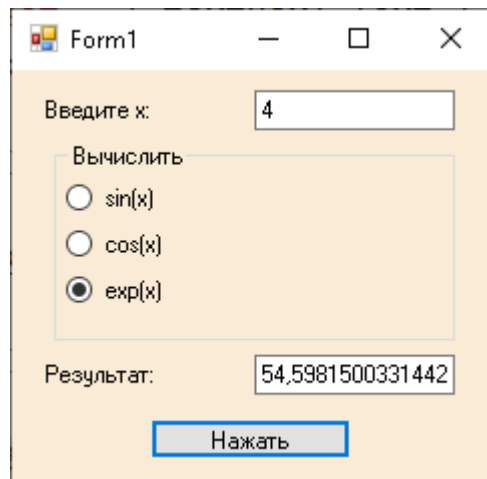


Рисунок 8. Группа кнопок-переключателей

Взаимодействие с кнопками-переключателями описывает следующий код:

```
if (radioButton1.Checked)
{
    textBox1.Text =
    Convert.ToString(Math.Sin(Convert.ToDouble(textBox2.Text)));
}
else if (radioButton2.Checked)
{
    textBox1.Text =
    Convert.ToString(Math.Cos(Convert.ToDouble(textBox2.Text)));
}
else if (radioButton3.Checked)
{
```

```

        textBox1.Text =
Convert.ToString(Math.Exp(Convert.ToDouble(textBox2.Text)));
    }

```

### Флажок для множественного выбора

Элемент CheckBox или флажок предназначен для установки одного из двух значений: отмечен или не отмечен. Чтобы отметить флажок, надо установить у его свойства Checked значение true.

Кроме свойства Checked у элемента CheckBox имеется свойство CheckState, которое позволяет задать для флажка одно из трех состояний - Checked (отмечен), Indeterminate (флажок не определен - отмечен, но находится в неактивном состоянии) и Unchecked (не отмечен).

Также следует отметить свойство AutoCheck - если оно имеет значение false, то мы не можем изменять состояние флажка. По умолчанию оно имеет значение true. Форма с элементами-флажками представлена на рисунке 9.

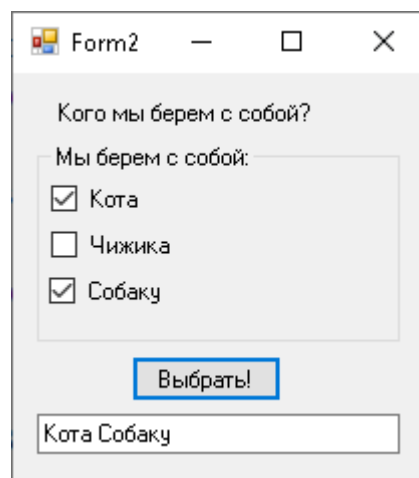


Рисунок 9. Форма с CheckBox

Взаимодействие с CheckBox позволяет описать следующий код:

```

private void button1_Click(object sender, EventArgs e)
//кнопка "Выбрать"
{
    textBox1.Text = "";
    switch(checkBox1.Checked)
    {
        case true:
        {
            textBox1.Text += checkBox1.Text + " ";

```

```

        break;
    }
    case false:
    {
        break;
    }
}
switch(checkBox2.Checked)
{
    case true:
    {
        textBox1.Text += checkBox2.Text + " ";
        break;
    }
    case false:
    {
        break;
    }
}
switch(checkBox3.Checked)
{
    case true:
    {
        textBox1.Text += checkBox3.Text + " ";
        break;
    }
    case false:
    {
        break;
    }
}
}

```

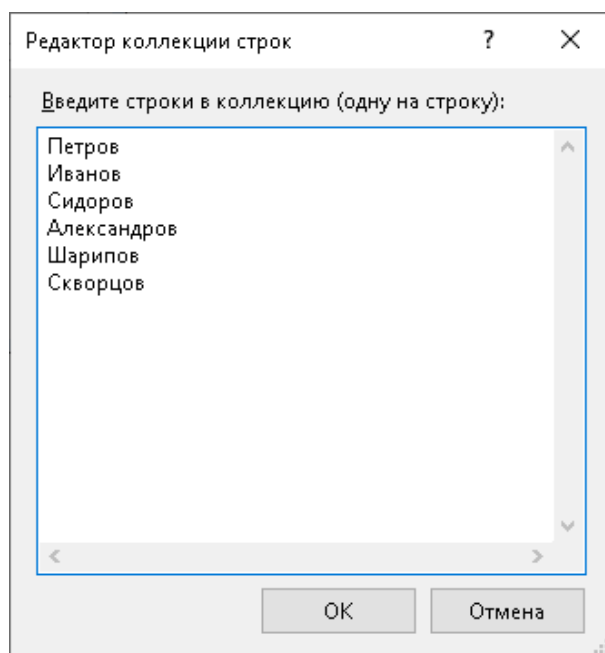
### Элемент **ListBox**

Элемент **ListBox** представляет собой простой список. Ключевым свойством этого элемента является свойство **Items**, которое как раз и хранит набор всех элементов списка.

Элементы в список могут добавляться как во время разработки, так и программным способом. В **Visual Studio** в окне **Properties** (Свойства) для элемента **ListBox** мы можем найти свойство **Items**. После двойного щелчка на

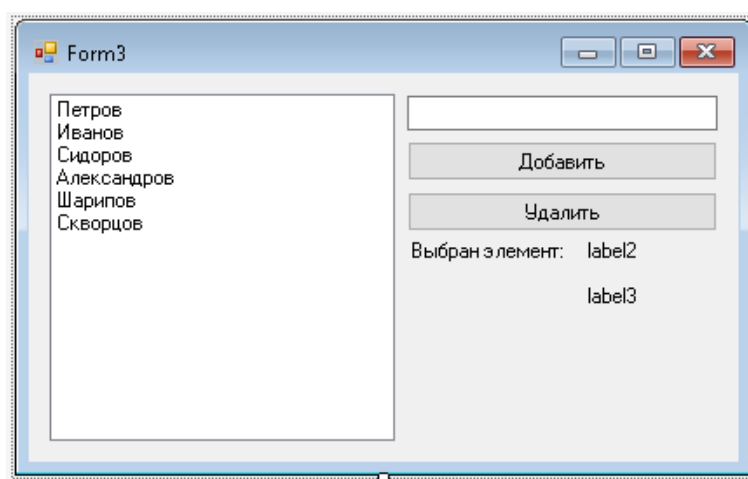


свойство нам отобразится окно для добавления элементов в список (Рисунок 10).



*Рисунок 10 – Редактор коллекции строк в ListBox*

В пустое поле мы вводим по одному элементу списка - по одному на каждой строке. После этого все добавленные нами элементы окажутся в списке, и мы сможем ими управлять. Создадим форму (Рисунок 11) для управления элементами формы. В качестве функционала добавим возможность добавления элемента в конец списка, удаление и отображение выделенного элемента.



*Рисунок 11 – Макет формы для управления содержимым списка*

Итак, все элементы списка входят в свойство `Items`, которое представляет собой коллекцию. Для добавления нового элемента в эту

коллекцию, а значит и в список, надо использовать метод `Add`, например:  
`listBox1.Items.Add("Новый элемент");`. При использовании этого метода каждый добавляемый элемент добавляется в конец списка.

Можно добавить сразу несколько элементов, например, массив. Для этого используется метод `AddRange`:

```
string[] students = {"Кондратьев", "Дормидонтова", "Валеева"};  
listBox1.Items.AddRange(students);
```

В отличие от простого добавления вставка производится по определенному индексу списка с помощью метода `Insert`:

```
listBox1.Items.Insert(listBox1.Items.Count, textBox1.Text);
```

Свойство `listBox1.Items.Count` определяет текущее количество элементов в списке. В вышеописанной строчке кода вставка элемента будет производиться постоянно в конец списка.

Для удаления элемента по его тексту используется метод `Remove`:

```
listBox1.Items.Remove("Петров");
```

Чтобы удалить элемент по его индексу в списке, используется метод `RemoveAt`:

```
listBox1.Items.RemoveAt(1);
```

В качестве параметра передается индекс удаляемого элемента.

Кроме того, можно очистить сразу весь список, применив метод `Clear`:

```
listBox1.Items.Clear();
```

Используя индекс элемента, можно получить доступ к самому элементу в списке. Например, получим первый элемент списка:

```
string firstElement = listBox1.Items[0];
```

При выделении элементов списка мы можем ими управлять как через индекс, так и через сам выделенный элемент. Получить выделенные элементы можно с помощью следующих свойств элемента `ListBox`:

- `SelectedIndex`: возвращает или устанавливает номер выделенного элемента списка. Если выделенные элементы отсутствуют, тогда свойство имеет значение `-1`

- **SelectedIndices**: возвращает или устанавливает коллекцию выделенных элементов в виде набора их индексов
- **SelectedItem**: возвращает или устанавливает текст выделенного элемента
- **SelectedItems**: возвращает или устанавливает выделенные элементы в виде коллекции

По умолчанию список поддерживает выделение одного элемента. Чтобы добавить возможность выделения нескольких элементов, надо установить у его свойства **SelectionMode** значение **MultiSimple**.

Чтобы выделить элемент программно, надо применить метод **SetSelected(int index, bool value)**, где **index** - номер выделенного элемента. Если второй параметр - **value** имеет значение **true**, то элемент по указанному индексу выделяется, если **false**, то выделение наоборот скрывается:

```
listBox1.SetSelected(2, true); // будет выделен третий
                               элемент
```

Чтобы снять выделение со всех выделенных элементов, используется метод **ClearSelected**.

Из всех событий элемента **ListBox** надо отметить в первую очередь событие **SelectedIndexChanged**, которое возникает при изменении выделенного элемента. Рассмотрим реализацию обработчика события на примере нашего задания.

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    label2.Text = Convert.ToString(listBox1.SelectedItem);
    label3.Text = Convert.ToString(listBox1.SelectedIndex);
}
```

Обработчик события изменяет текст в надписях. В **label2** выводится текст выбранного пользователем элемента, а в **label3** – его индекс.

При нажатии на кнопку «Добавить» будет происходить добавление нового элемента в конец списка. Конец элемента будет определять индекс, равный **listBox1.Items.Count** – текущему количеству элементов в списке.

Текстовое наполнение нового элемента будет извлекаться из данных, введенных в `textBox1`.

```
private void button1_Click(object sender, EventArgs e)
{
    listBox1.Items.Insert(listBox1.Items.Count,
textBox1.Text);
}
```

Кнопка «Удалить» будет производить удаление выбранного в списке элемента. `listBox1.SelectedIndex` будет возвращать индекс выбранного пользователем элемента.

```
private void button2_Click(object sender, EventArgs e)
{
    listBox1.Items.RemoveAt(listBox1.SelectedIndex);
}
```

### Пример написания программы

Выполним следующее задание, разработав для него интерфейс с помощью Windows Forms.

**Задание** – Поле шахматной доски определяется парой натуральных чисел, каждое из которых не превосходит 8: первое число — номер вертикали (при счете слева направо), второе—номер горизонтали (при счете снизу-вверх). Даны натуральные числа *a*, *b*, *c*, *d*, *e*, *f*, каждое из которых не превосходит 8. Определить, может ли белая ладья, расположенная на поле (*a*, *b*), одним ходом пойти на поле (*e*, *f*), не попав при этом под удар черной ладьи, находящейся на поле (*c*, *d*).

**Решение** – Упростим решение задачи. Поля шахматного поля будем представлять в виде флажков `CheckBox` и будем создавать их динамически, то есть явно через программный код. Финальный макет разрабатываемого приложения показан на рисунке 12.

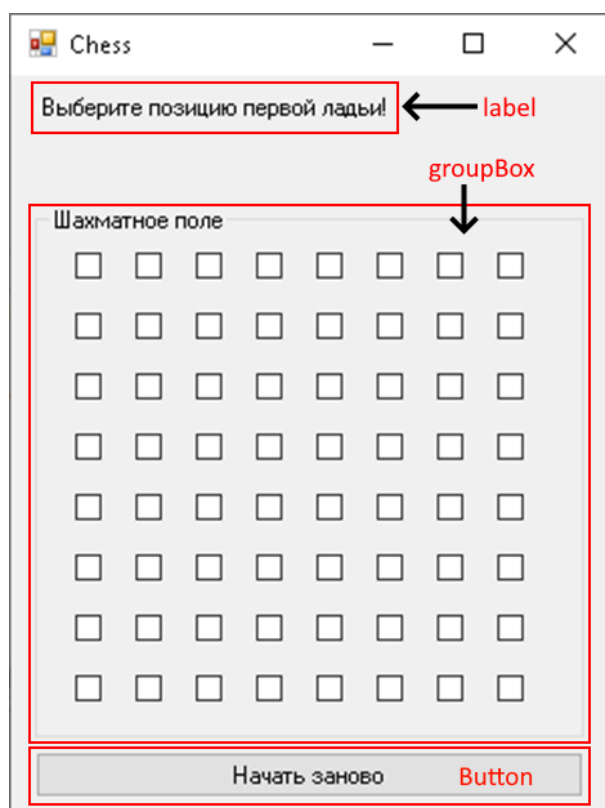


Рисунок 12

Промежуточный макет разрабатываемого приложения, который необходимо собрать вручную, представлен на рисунке 13.



Рисунок 13

Label будет хранить информацию для взаимодействия с пользователем: «Выберите позицию первой ладьи», «Выберите позицию первой ладьи» и «Выберите новую позицию первой ладьи». Контейнер GroupBox будет хранить флажки checkBox, каждый из которых будет представлять ячейку в шахматном поле. Кнопка «Начать заново» перезапустит ход игры, вернув в исходное положение все checkBox.

В разрабатываемой программе можно выделить три сущности, которые можно представить в виде классов:

1. Класс Rook (Ладья) – будет описывать объект «Ладья» со свойствами coord\_x (Позиция ладьи по горизонтали) и coord\_y (Позиция ладьи по вертикали). Конструктор класса будет инициализировать новые координаты ладьи в шахматном поле при ее создании. Создание будет происходить при выборе соответствующего флажка на игровом поле.

1	<code>public class Rook</code>
2	<code>{</code>
3	<code>    public int coord_x { get; set; }</code>
4	<code>    public int coord_y { get; set; }</code>
5	<code>    public Rook(int x, int y)</code>
6	<code>    {</code>
7	<code>        this.coord_x = x;</code>
8	<code>        this.coord_y = y;</code>
9	<code>    }</code>
10	<code>}</code>

2. Класс NewCheckBox – данный класс будет наследоваться от стандартного класса CheckBox и будет определять для него дополнительные свойства – это координаты (по горизонтали и вертикали) checkBox в шахматном поле. Координаты будут инициализироваться при создании элементов шахматного поля.

1	<code>public class NewCheckBox : CheckBox</code>
2	<code>{</code>
3	<code>    public int Coord_x { get; set; }</code>
4	<code>    public int Coord_y { get; set; }</code>
5	<code>    public NewCheckBox(int x, int y)</code>
6	<code>    {</code>
7	<code>        this.Coord_x = x;</code>

8	<code>this.Coord_y = y;</code>
9	<code>}</code>
10	<code>}</code>

3. Класс `ChessField` (Шахматное поле) – данный класс представляет описание объекта «Шахматное поле» и будет хранить следующие свойства: это массив флажков `checkBox` размерностью 8\*8 для дальнейшего их добавления в форму и переменная `state`, отвечающая за этап решения задачи (0 – выбор позиции первой ладьи, 1 – выбор позиции второй ладьи, 2 – выбор новой позиции первой ладьи).

Конструктор класса будет отвечать за динамическое создание элементов `checkBox` в форме.

1	<code>public class ChessField</code>
2	<code>{</code>
3	<code>    public NewCheckBox[, ] checkBoxes;</code>
4	<code>    public int state;</code>
5	<code>    public ChessField(int x, int y)</code>
6	<code>    {</code>
7	<code>        state = 0;</code>
8	<code>        checkBoxes = new NewCheckBox[x, y];</code>
9	<code>        for (int i = 0; i &lt; 8; i++)</code>
10	<code>        {</code>
11	<code>            for (int j = 0; j &lt; 8; j++)</code>
12	<code>            {</code>
13	<code>                checkBoxes[i, j] = new NewCheckBox(i, j)</code>
14	<code>                {</code>
15	<code>                    Checked = false,</code>
16	<code>                    Location = new Point(i * 30 + 20, j * 30</code>
17	<code>                    + 20),</code>
18	<code>                    Size = new Size(20, 20),</code>
19	<code>                    Name = "cell" + Convert.ToString(i + 1) +</code>
20	<code>                    Convert.ToString(j + 1),</code>
21	<code>                    Visible = true</code>
22	<code>                };</code>
23	<code>            }</code>
24	<code>        }</code>

Конструктор в качестве параметров принимает два целочисленных аргумента (строка 5), которые отвечают за количество элементов в каждом

измерении матрицы `checkboxes` при ее создании. Свойство `state` инициализируется нулем (строка 7), сигнализирующем о начальном состоянии программы (расстановка на шахматное поле первой ладьи).

В двойном цикле происходит обработка двумерного массива `checkboxes`. Во вложенном цикле динамически создается элемент `checkboxes` как экземпляр класса `NewCheckBox` (строка 13). Далее происходит инициализация свойств экземпляра класса `NewCheckBox` (строка 15- 19). Свойство `Checked` (строка 15) задает неотмеченное состояние флажка. Свойство `Location` (строка 16), которое создается динамически через вызов конструктора класса `Point`, определяет новое расположение элемента относительно верхнего левого края формы. Как аргументы конструктора задаются координаты по горизонтали и координаты по вертикали. Свойство `Size` (строка 17) задает размерность элемента по ширине и высоте. Свойство `Name` (строка 18) изменяет значение названия элемента, по которому в дальнейшем можно к нему обращаться. Свойство `Visible` (строка 19) изменяет видимость и невидимость элемента (если `True`, то элемент видимый).

После создания необходимых для решения задачи классов можно приняться за создание функциональности генерируемых событий в форме. При загрузке формы необходимо проинициализировать текстовое наполнение требуемых элементов формы и добавить в контейнер `groupBox1` все элементы, сгенерированные в классе `ChessField`. Обработчик события будет выглядеть следующим образом:

1	<code>private void Chess_Load(object sender, EventArgs e)</code>
2	<code>{</code>
3	<code>label1.Text = "Выберите позицию первой ладьи!";</code>
4	<code>groupBox1.Text = "Шахматное поле";</code>
5	<code>chessField = new ChessField(8, 8);</code>
6	<code>foreach (NewCheckBox i in chessField.checkBoxes)</code>
7	<code>{</code>
8	<code>groupBox1.Controls.Add(i);</code>



9	<code>i.CheckedChanged += new</code> <code>System.EventHandler(this.checkBox_CheckedChanged);</code>
10	<code>}</code>
11	<code>}</code>

В строке 5 происходит создание экземпляра класса `ChessField`, создается «Шахматное поле» размерностью 8\*8. Далее, используя цикл `foreach`, организуем обработку элементов двойного массива `checkboxes`, который представляет собой свойство экземпляра класса `ChessField`, где каждый элемент массива добавляем в контейнер `groupBox1` (строка 8). В строке 9 каждому элементу массива привязывается для его события `CheckedChanged` обработчик, представляющий собой метод `checkBox_CheckedChanged`. Название метода передается как аргумент при вызове конструктора обработчика события `System.EventHandler`.

Самое интересное кроется в методе обработчике события нажатия на флажок. При изменении состояния флажка он генерирует событие `CheckedChanged`. Обработывая это событие, мы можем получать измененный флажок и производить определенные действия.

1	<code>private void checkBox_CheckedChanged(object sender, EventArgs e)</code>
2	<code>{</code>
3	<code>    NewCheckBox checkBox = (NewCheckBox)sender; // приводим</code> <code>    отправителя к элементу типа CheckBox</code>

Здесь мы параметр `sender`, представляющий собой элемент отправитель события (флажок), преобразуем в пользовательский тип `NewCheckBox` и сохраняем в локальную переменную `checkbox`. Проще говоря, в переменную `checkbox` сохраняется тот элемент-флажок, который выбрал пользователь.

После инициализации переменной идет проверка состояния `Checked` флажка (выбран/не выбран).

4	<code>if (checkBox.Checked == true)</code>
5	<code>{</code>

Далее идет проверка значения свойства `chessField.state`. `state = 0` – это выбор позиции первой ладьи в шахматном поле, `state = 1` – это выбор позиции второй ладьи в шахматном поле, `state = 2` – это выбор новой позиции для первой ладьи.

6	<code>if (chessField.state == 0)</code>
7	<code>{</code>
8	<code>    checkBox.AutoCheck = false;</code>
9	<code>    checkBox.BackColor = Color.Aqua;</code>
10	<code>    rookFirst = new Rook(checkBox.Coord_x,</code> <code>    checkBox.Coord_y);</code>
11	<code>    chessField.state = 1;</code>
12	<code>    label1.Text = "Выберите позицию второй ладьи!";</code>
13	<code>}</code>
14	<code>else if (chessField.state == 1)</code>
15	<code>{</code>
16	<code>    checkBox.AutoCheck = false;</code>
17	<code>    checkBox.BackColor = Color.Blue;</code>
18	<code>    rookSecond = new Rook(checkBox.Coord_x,</code> <code>    checkBox.Coord_y);</code>
19	<code>    chessField.state = 2;</code>
20	<code>    label1.Text = "Выберите новую позицию первой ладьи!";</code>
21	<code>}</code>
22	<code>else if (chessField.state == 2)</code>
23	<code>{</code>
24	<code>    if (!(rookFirst.coord_x == checkBox.Coord_x   </code> <code>    rookFirst.coord_y == checkBox.Coord_y))</code>
25	<code>    {</code>
26	<code>        checkBox.Checked = false;</code>

27	<code>MessageBox.Show("Ладья так ходить не может!");</code>
28	<code>}</code>
29	<code>else</code>
30	<code>{</code>
31	<code>rookFirst.coord_x = checkBox.Coord_x;</code>
32	<code>rookFirst.coord_y = checkBox.Coord_y;</code>
33	<code>checkBox.BackColor = Color.Red;</code>
34	<code>if (rookFirst.coord_x == rookSecond.coord_x    rookFirst.coord_y == rookSecond.coord_y)</code>
35	<code>{</code>
36	<code>MessageBox.Show("Ладья под ударом!");</code>
37	<code>}</code>
38	<code>else</code>
39	<code>{</code>
40	<code>MessageBox.Show("Ладья в безопасности!");</code>
41	<code>}</code>
42	<code>checkBox.AutoCheck = false;</code>
43	<code>chessField.state = 0;</code>
44	<code>label1.Text = "";</code>
45	<code>}</code>
46	<code>}</code>

Свойство флажка `AutoCheck` равное `false` блокирует изменение состояния флажка. По умолчанию данное свойство имеет значение `true`.

Свойство `BackColor` позволяет изменить цветовую заливку элемента управления. Объект `Color` представляет собой цвет фона элемента управления.

При выборе позиции для первой и второй ладьи происходит динамическое создание двух экземпляров класса `Rook` с передачей в

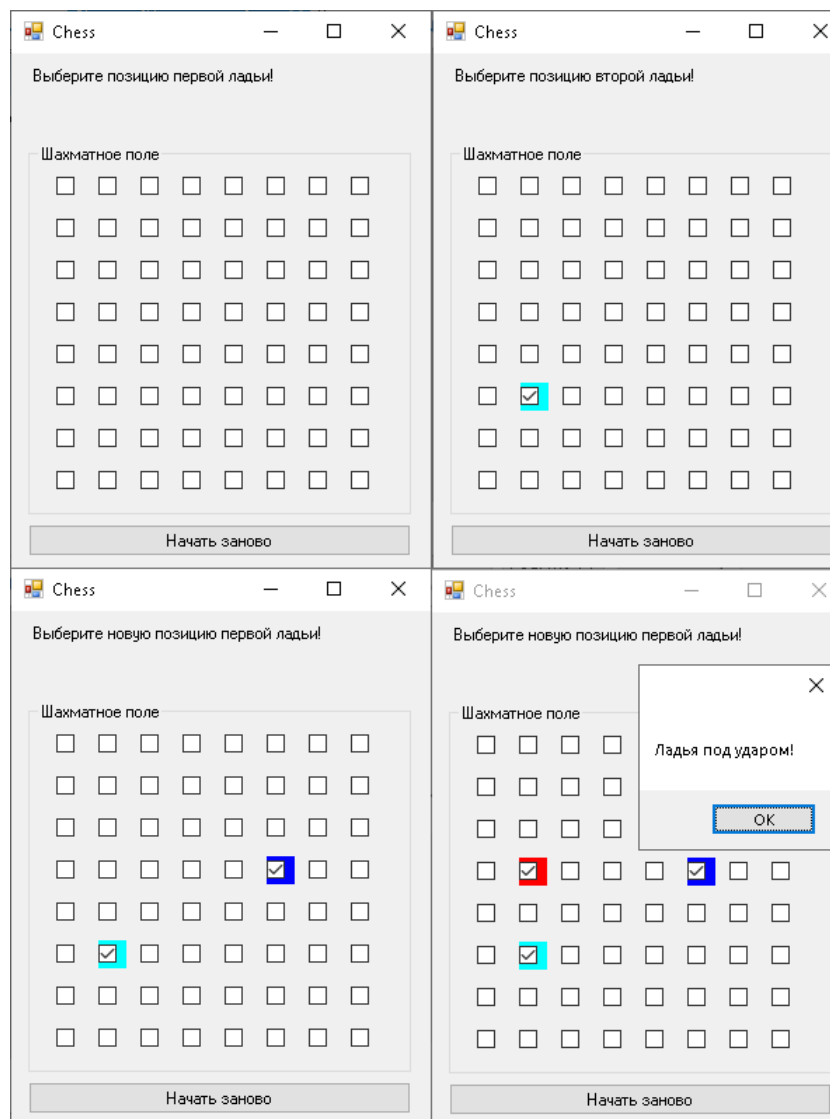
конструктор класса в качестве аргументов координаты по горизонтали и вертикали элементов управления, символизирующих ячейку в шахматном поле.

При достижении `state` значения 2 идет проверка поставленных в условии задачи ограничений нового расположения первой ладьи.

Последний метод, который необходимо добавить – это обработчик события нажатия кнопки «Нажать заново». При ее нажатии мы изменяем стартовое сообщение, обнуляем значение поля `state`, и с помощью цикла обходим все хранящиеся в массиве элементы-флажки и изменяем их свойства: цветовую заливку `BackColor` устанавливаем на значение `SystemColors.Control` (цвет по умолчанию), устанавливаем `AutoCheck` на значение `true`, чтобы разрешить изменение состояния всех флажков и устанавливаем `Checked` на значение `false`, чтобы изменить состояние всех флажков в контейнере на «Не отмеченный».

1	<code>private void button1_Click(object sender, EventArgs e)</code>
2	<code>{</code>
3	<code>label1.Text = "Выберите позицию первой ладьи!";</code>
4	<code>chessField.state = 0;</code>
5	<code>foreach (NewCheckBox i in chessField.checkBoxes)</code>
6	<code>{</code>
7	<code>    i.BackColor = SystemColors.Control;</code>
8	<code>    i.AutoCheck = true;</code>
9	<code>    i.Checked = false;</code>
10	<code>}</code>
11	<code>}</code>

Результат работы приложения представлен на рисунке 14.



*Рисунок 14*

### ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Выбрать пять заданий из лабораторных работ №2-6 (нельзя брать оба задания из одной лабораторной работы) и решить их, разработав графический интерфейс с использованием технологии Windows Forms.

**Вариант заданий необходимо согласовать у преподавателя перед непосредственной разработкой!**