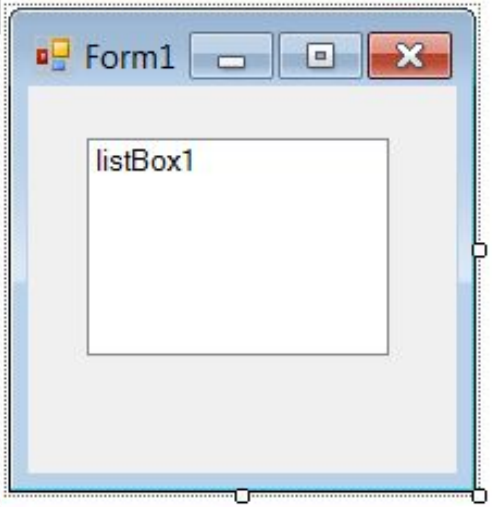
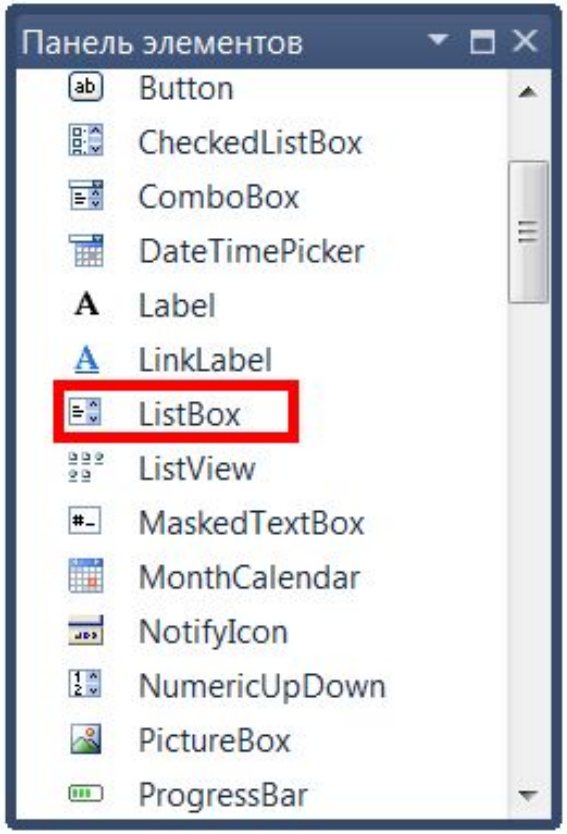
9. Элемент управления ListBox. Строки в C#.

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





Работа со списками

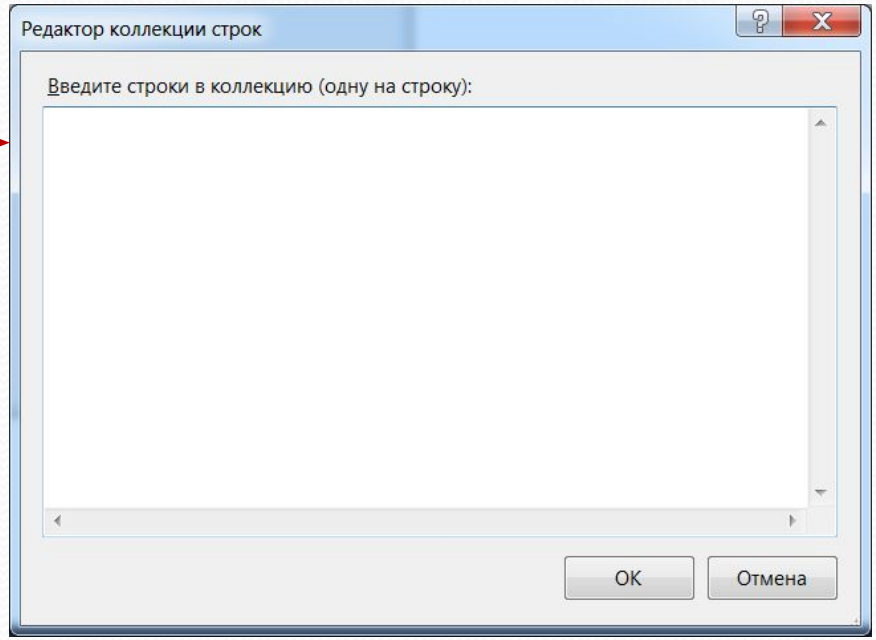
При работе со списками программисту обычно приходится решать следующие задачи:

- **добавление (вставка) в список новых элементов;**
- **удаление элементов из списка;**
- **замещение элемента списка новым значением;**
- **определение выделенного элемента списка (или нескольких, если список допускает множественное выделение).**

Записи в список добавляются одним из двух способов:

1. **В режиме конструктора с помощью коллекции Items из окна Свойства;**

При нажатии на многоточие появляется окно **Редактор коллекции строк** (String Collection Editor)



Список заполняется элементами так же, как и многострочное текстовое поле

2. В программном коде с помощью метода ADD.

Добавление элементов

Для добавления в конец списка новых элементов служит **метод Add() объекта Items списка.**

Формат вызова метода:

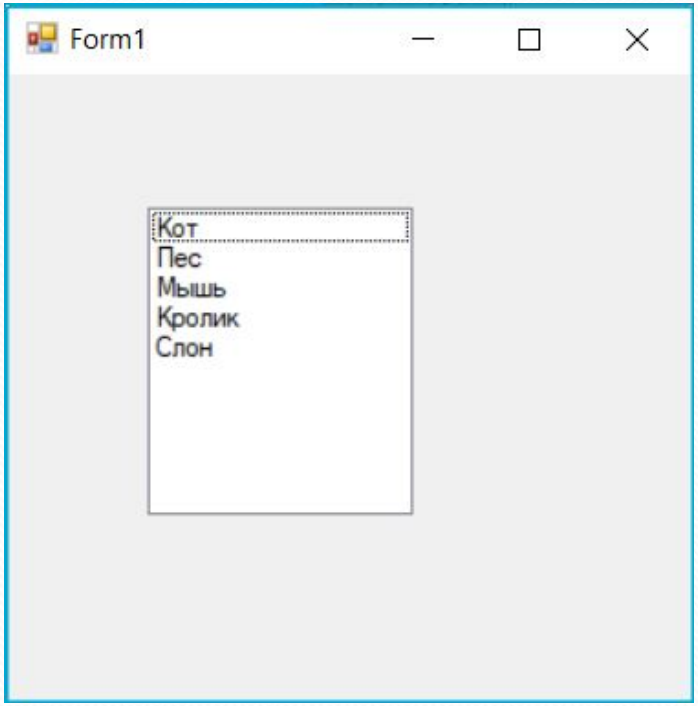
ИмяСписка.Items.Add(добавляемая_строка)

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[] animals = {"Кот", "Пес", "Мышь", "Кролик", "Слон"};
listBox1.Items.AddRange(animals);
```

Вставка элементов

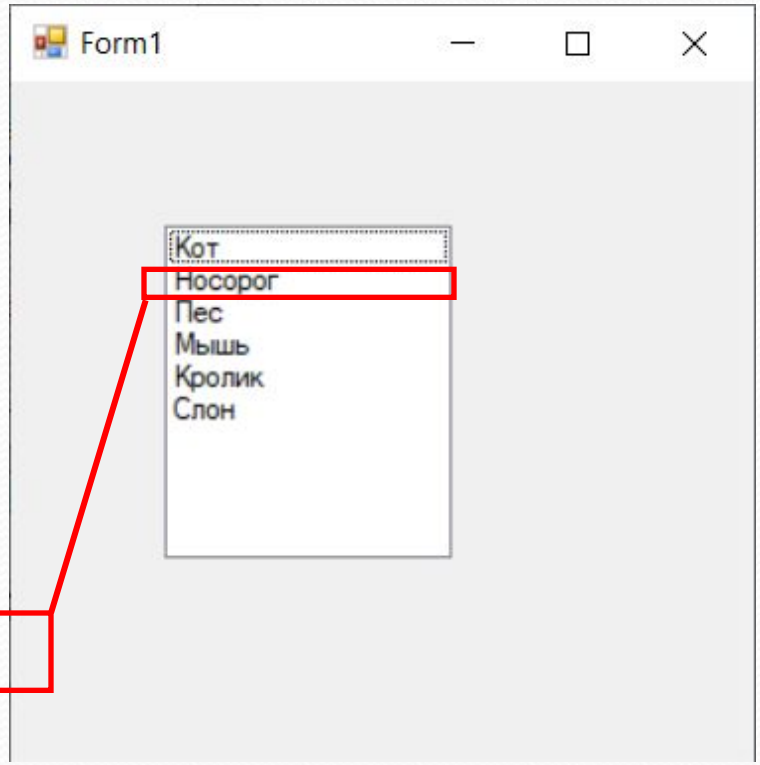
Чтобы вставить элемент в произвольное место списка (а не только в конец), можно воспользоваться **методом Insert():**

ИмяСписка.Items.Insert(Номер, вставляемая_строка)

где Номер — это порядковый номер позиции в списке, на которую будет помещена вставляемая строка.

Первая позиция имеет номер 0, вторая — 1 и т. д.

```
string[] animals = {"Кот", "Пес",
"Мышь", "Кролик", "Слон"};
listBox1.Items.AddRange(animals);
listBox1.Items.Insert(1, "Носорог");
```



В данном случае вставляем элемент на вторую позицию в списке, так как отсчет позиций начинается с нуля

Удаление элементов

Для удаления элемента списка служат два метода:

Remove() и RemoveAt()

Метод Remove() удаляет из списка строку, переданную ему в качестве аргумента:

ИмяСписка.Items.Pemove(удаляемая_строка)

Метод RemoveAt() удаляет строку, находящуюся на заданной позиции в списке:

ИмяСписка.Items.RemoveAt(порядковый_номер_удаляемоrо _элемента)

Иногда требуется **удалить из списка все элементы.**
Для этого можно воспользоваться **методом Clear():**

ИмяСписка.Clear()

Удаление элементов

- Для удаления элемента по его тексту используется метод Remove:

```
string[] animals = {"Кот", "Пес", "Мышь", "Кролик", "Слон"};
listBox1.Items.AddRange(animals);
listBox1.Items.Insert(1, "Носорог");
listBox1.Items.Remove(«Мышь");
```

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

```
listBox1.Items.RemoveAt(3);
```

- Можно очистить сразу весь список, применив метод Clear:

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

Основные свойства списков

HorizontalScrollbar - управляет отображением горизонтальной полосы прокрутки (ГПП).
Значение **False** – ГПП никогда не отображается.
True – ГПП будет отображаться, если хотя бы один элемент списка имеет ширину, большую, чем ширина самого списка.

ScrollAlwaysVisible - определяет, всегда ли отображаются полосы прокрутки:
True - всегда; **False** -по мере надобности.

Sorted - значение **True** означает, что элементы списка будут автоматически сортироваться в алфавитном порядке

SelectionMode - управляет возможностью выбора пользователем сразу нескольких элементов списка:

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

Значение свойства	Описание
None	Выбор элементов списка запрещен
One	Можно выбрать лишь один элемент списка
MultiSimple	Можно выбрать несколько элементов списка;
MultiExtended	Можно выбрать несколько элементов списка, используя клавиши Shift и Ctrl

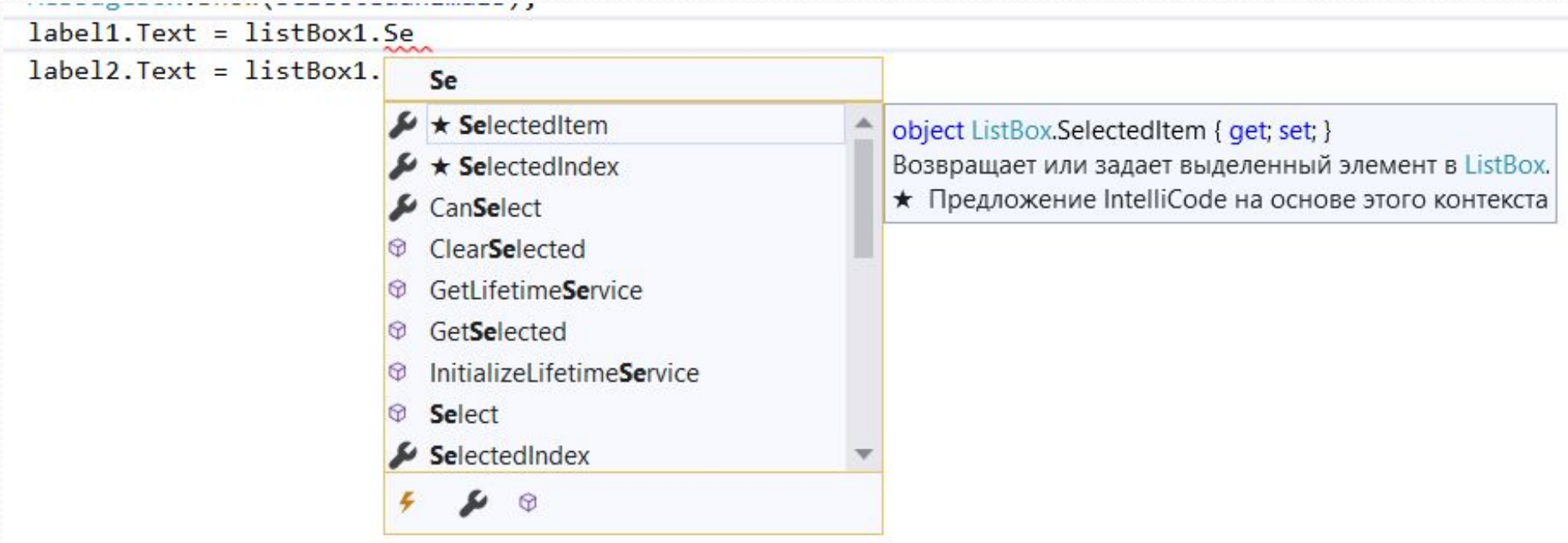
Метод **Count** позволяет определить количество элементов в списке:

Items.Count	Число элементов в списке

Определение выделенных элементов списка:

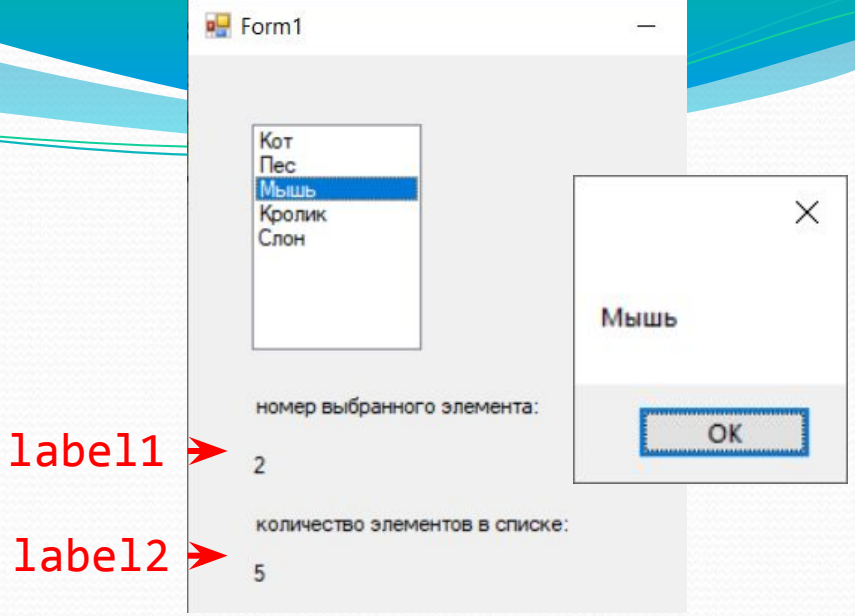
Свойства	Описание
SelectedIndex	возвращает или устанавливает номер выделенного элемента списка. Если выделенные элементы отсутствуют, тогда свойство имеет значение -1
SelectedItem	возвращает или устанавливает текст выделенного элемента
SelectedIndides	возвращает или устанавливает коллекцию выделенных элементов в виде набора их индексов
SelectedItems	возвращает или устанавливает выделенные элементы в виде коллекции

Обычно с системными свойствами работают через код программы и вызывают из справочной системы, в которой имеется пояснение их работы (Ctrl+Space):



Событие SelectedIndexChanged

Из всех событий элемента ListBox надо отметить в первую очередь событие SelectedIndexChanged, которое возникает при изменении выделенного элемента.

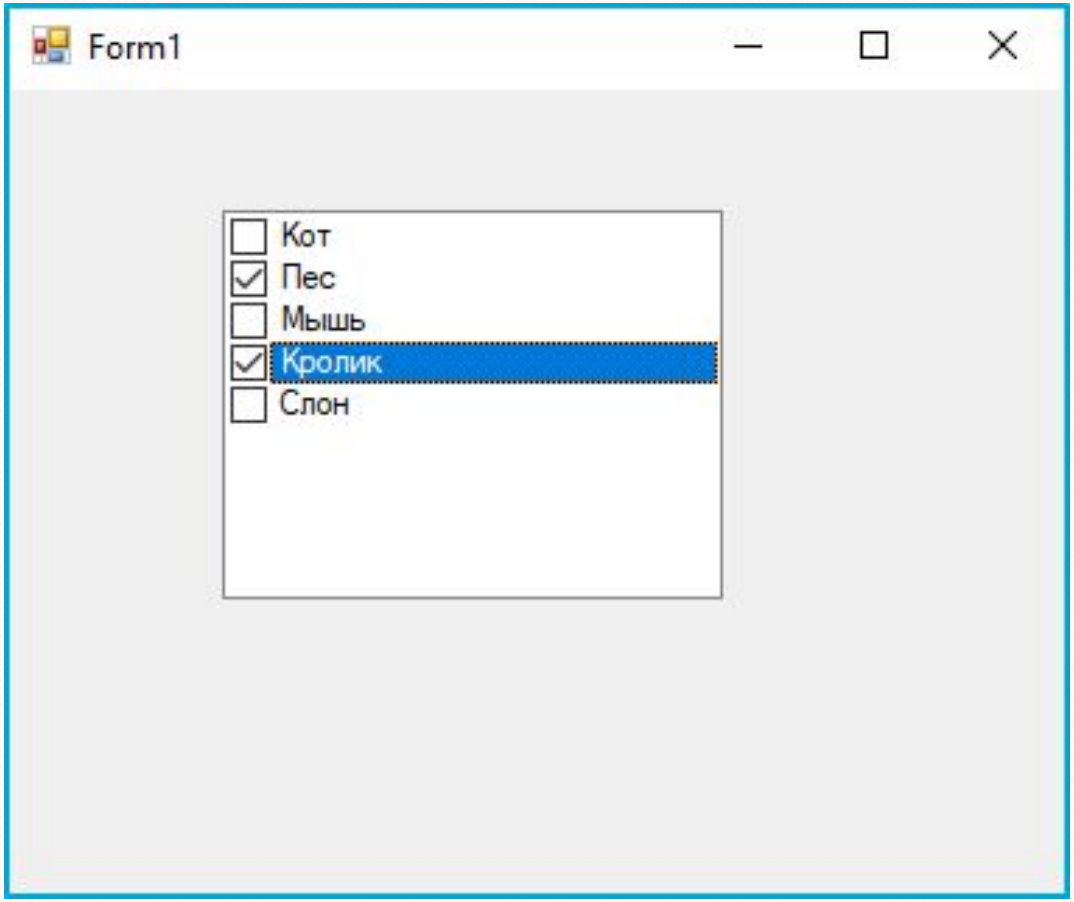



```
private void Form1_Load(object sender, EventArgs e)
{
string[] animals = {"Кот", "Пес", "Мышь", "Кролик", "Слон"};
listBox1.Items.AddRange(animals);
}
private void listBox1_SelectedIndexChanged(object sender,
EventArgs e)
{
string selectedanimals = listBox1.SelectedItem.ToString();
MessageBox.Show(selectedanimals);
label1.Text = listBox1.SelectedIndex.ToString();
label2.Text = listBox1.Items.Count.ToString();
}
```

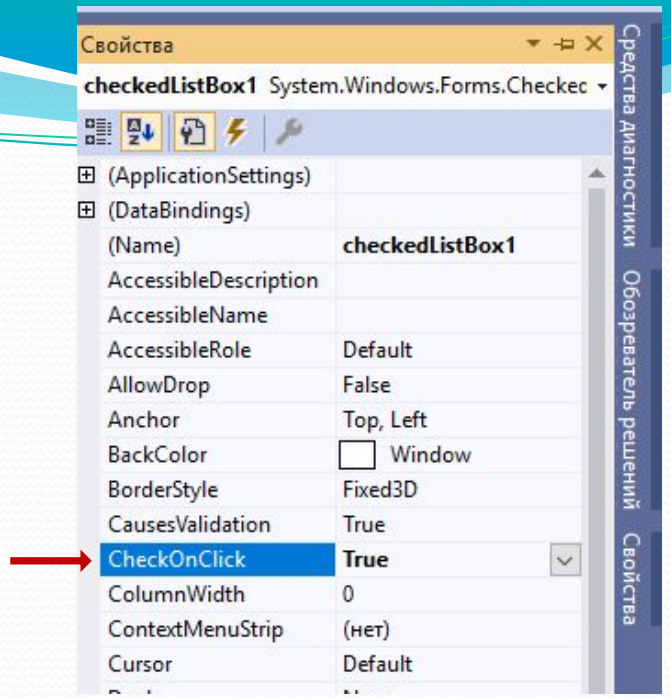
Элемент управления CheckedListBox

Элемент CheckedListBox представляет симбиоз компонентов ListBox и CheckBox. Для каждого элемента такого списка определено специальное поле CheckBox, которое можно отметить.

Элемент CheckedListBox обладает всеми свойствами элемента ListBox, добавляя при этом несколько новых – **CheckOnClick, CheckedIndides и CheckedItems**.



Чтобы поставить отметку в checkBox рядом с элементом в списке, нам надо сначала выделить элемент и дополнительным щелчком уже установить флажок. Однако это не всегда удобно, и с помощью свойства **CheckOnClick** и установке для него значения true мы можем определить сразу выбор элемента и установку для него флажка в один клик.



Свойство CheckOnClick

Значение данного свойства, равное True, означает, что флажок, соответствующий элементу списка, будет устанавливаться и сниматься первым же щелчком на элементе.

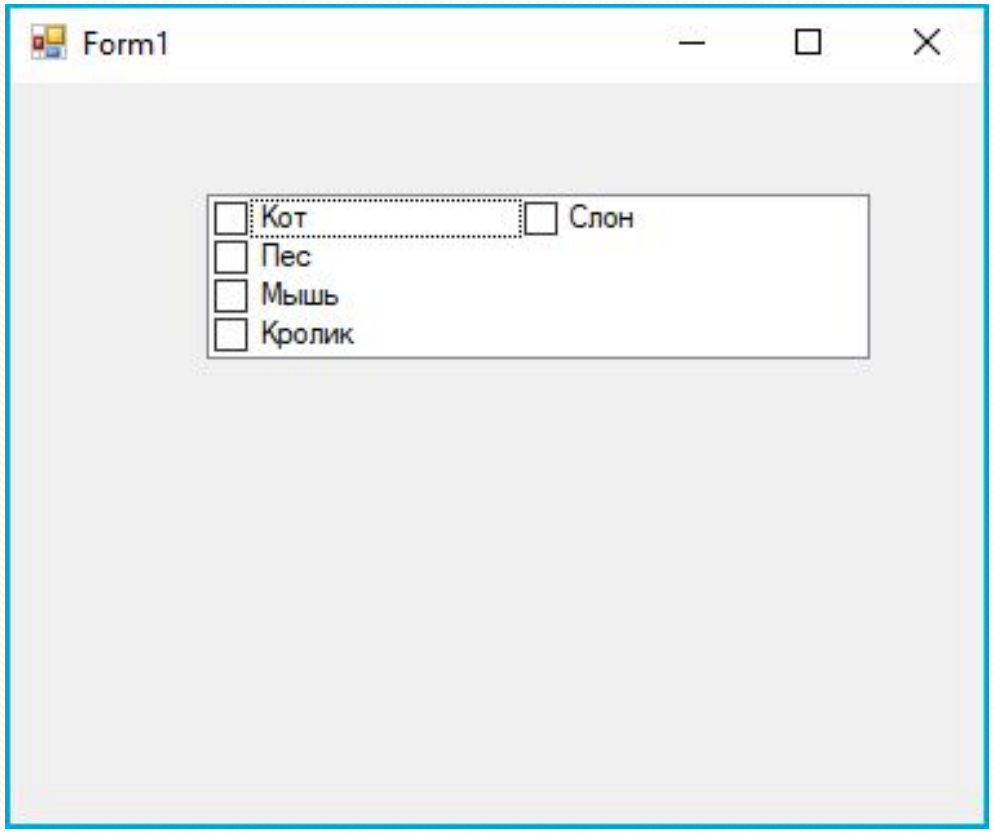
В противном случае первый щелчок лишь выделит элемент, второй щелчок изменит состояние сопоставленного элементу флажка

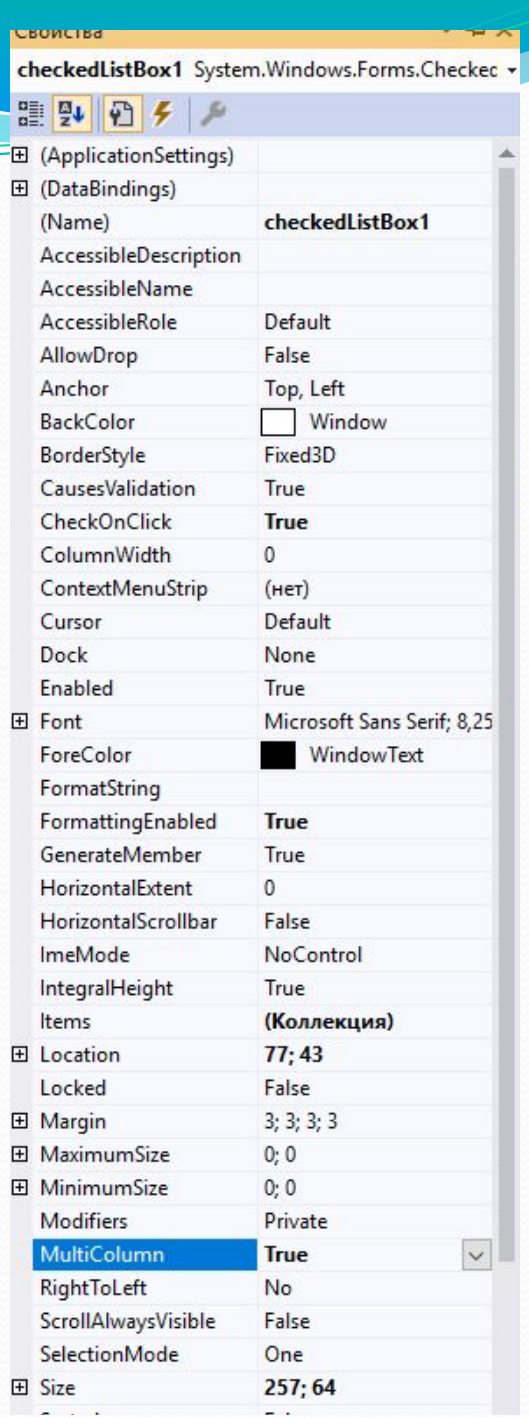
Свойства-коллекции CheckedIndicies и CheckedItems содержат порядковые номера и значения выбранных элементов (то есть тех элементов, чей флажок установлен).
Эти коллекции используются совершенно аналогично коллекциям SelectedIndides и SelectedItems.

Для добавления и удаления элементов в CheckedListBox определены все те же методы, что и в ListBox:

- **Add**(item): добавляет один элемент
- **AddRange**(array): добавляет в список массив элементов
- **Insert**(index, item): добавляет элемент по определенному индексу
- **Remove**(item): удаляет элемент
- **RemoveAt**(index): удаляет элемент по определенному индексу
- **Clear**(): полностью очищает список

Свойство **MultiColumn**
при значении true позволяет сделать многоколоночный список, если элементы не помещаются по длине:





Строковый тип данных

Довольно большое количество задач, которые могут встретиться при разработке приложений, так или иначе связано с обработкой строк - парсинг веб-страниц, поиск в тексте, какие-то аналитические задачи, связанные с извлечением нужной информации из текста и т.д. Поэтому в этом плане работе со строками уделяется особое внимание.

В языке C# строковые значения представляет тип **string**, а вся функциональность работы с данным типом сосредоточена в классе **System.String**. Собственно string является псевдонимом для класса System.String.

Строковый тип данных является ссылочным

Тип **string** является ссылочным, поэтому переменные этого типа хранят лишь адреса строк. Сами же строки являются динамическими структурами и размещаются в памяти уже в процессе выполнения программы. Длина строки ограничивается размером доступной оперативной памяти.

Переменная ссылочного типа содержит не данные, а ссылку на них. Сами данные в этом случае уже хранятся в куче. Куча - это область памяти, в которой размещаются управляемые объекты, и работает сборщик мусора. Сборщик мусора освобождает все ресурсы и объекты,

Постоянство строк

В языке C# существует понятие неизменяемый (immutable) класс. Для такого класса невозможно изменить значение объекта при вызове его методов. Динамические методы могут создавать новый объект, но не могут изменить значение существующего объекта. К таким неизменяемым классам относится и класс **String**. Ни один из методов этого класса не меняет значения существующих объектов. Конечно, некоторые из методов создают новые значения и возвращают в качестве результата новые строки.

Следует, однако, подчеркнуть, что переменные ссылки на строки (т.е. объекты типа string) подлежат изменению, а следовательно, они могут ссылаться на другой объект. Но содержимое самого объекта типа string не меняется после его создания.

Операции явного преобразования для данных строкового типа

Для всех встроенных типов данных возможно преобразование их значений в строковое представление, то есть приведение к строковому типу. Это достигается использованием метода **ToString()**. Синтаксис такого преобразования имеет вид:

```
<имя_переменной>.ToString()
```

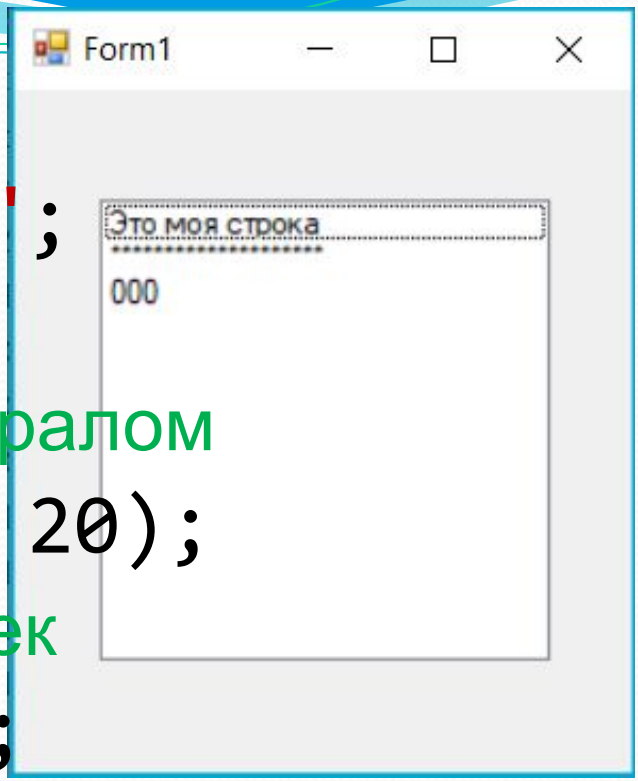
Отметим, что хотя этот метод не требует задания каких-либо параметров при своем вызове, пара круглых скобок после его имени должна быть указана. Обратное преобразование из строкового в любой встроенный тип значения называется также анализом строк. Анализ строки может быть выполнен с помощью метода **Parse()**, определенного для соответствующего типа. Синтаксис обращения к этому методу имеет вид:

```
<имя_типа>.Parse(<строка>)
```

Тип string не поддерживает ни одного варианта неявного преобразования

Создание строки

```
string s;
//инициализация отложена
string t = "Это моя строка";
listBox1.Items.Add(t);
//инициализация строковым литералом
string u = new string('*', 20);
//создание строки из 20 звездочек
listBox1.Items.Insert(1,u);
char[] a = {'0', '0', '0'};
//массив для инициализации строки
string v = new string(a);
//создание из массива символов
listBox1.Items.Insert(2, v);
```



Для строк определены следующие операции:

- **присваивание (=)**
- **проверка на равенство (==)**
- **проверка на неравенство (!=)**
- **обращение по индексу ([])**
- **сцепление (конкатенация) строк (+)**

Пример:

```
string a = "Текст";
string b = "строки";
string  c  =  a  +  "  "  +  b;
//  Результат:  Текст  строки
```

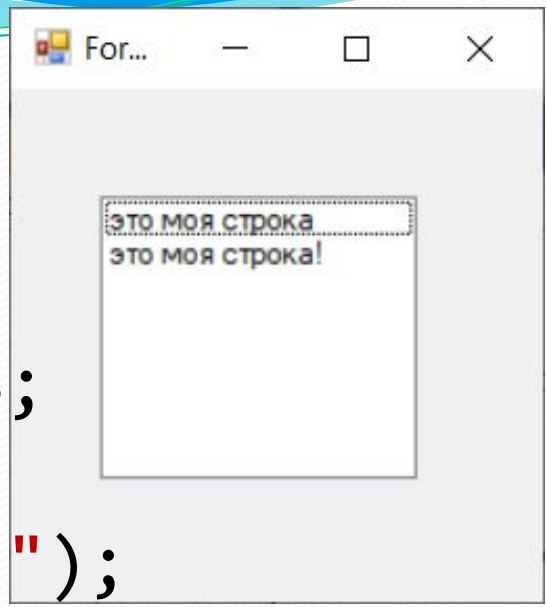
Основные методы строк

Compare: сравнивает две строки с учетом текущей культуры (локали) пользователя
CompareOrdinal: сравнивает две строки без учета локали
Contains: определяет, содержится ли подстрока в строке
Concat: соединяет строки
CopyTo: копирует часть строки, начиная с определенного индекса в массив
EndsWith: определяет, совпадает ли конец строки с подстрокой
Format: форматирует строку
IndexOf: находит индекс первого вхождения символа или подстроки в строке
Insert: вставляет в строку подстроку
Join: соединяет элементы массива строк
LastIndexOf: находит индекс последнего вхождения символа или подстроки в строке
Replace: замещает в строке символ или подстроку другим символом или подстрокой
Split: разделяет одну строку на массив строк
Substring: извлекает из строки подстроку, начиная с указанной позиции
ToLower: переводит все символы строки в нижний регистр
ToUpper: переводит все символы строки в верхний регистр
Trim: удаляет начальные и конечные пробелы из строки

Примеры использования методов строк

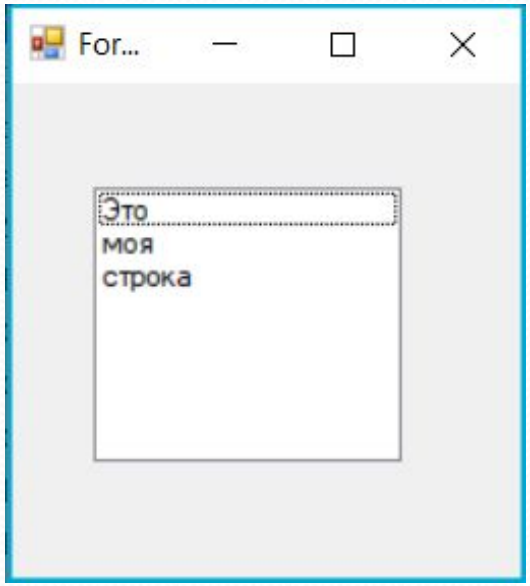
Конкантенация:

```
string s1 = "это";
string s2 = "моя";
string s3 = "строка";
string s4 = s1 + " " + s2+" "+s3;
listBox1.Items.Add(s4);
string s5 = String.Concat(s4, "!");
listBox1.Items.Insert(1, s5);
```



Разделение строк:

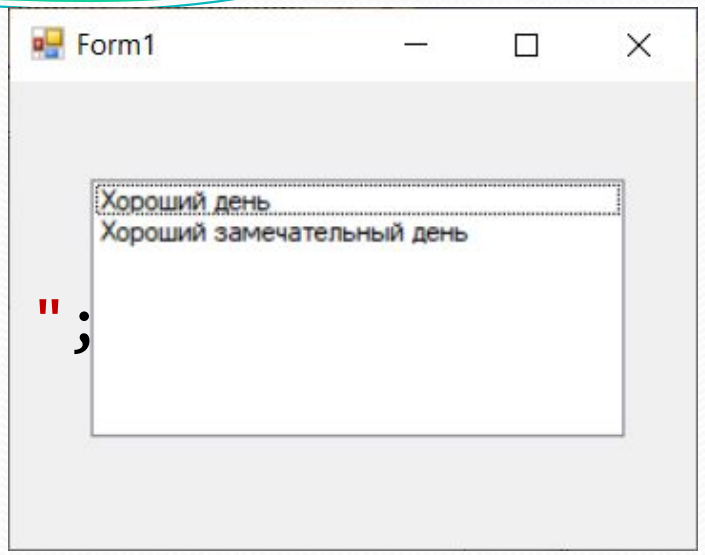
```
string t = "Это моя строка";
string[] Words = t.Split(' ');
listBox1.Items.AddRange(Words);
```



Примеры использования методов строк

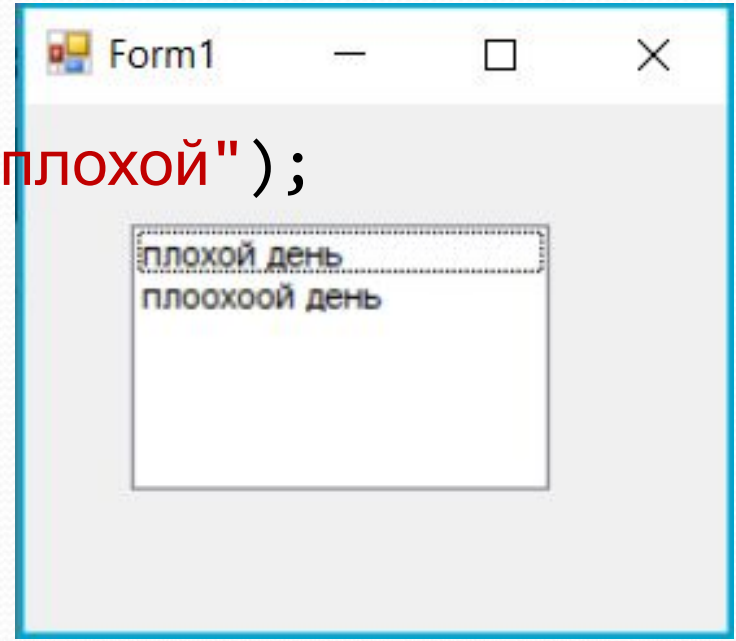
Вставка:

```
string text = "Хороший день";
listBox1.Items.Add(text);
string subString = "замечательный ";
text = text.Insert(8, subString);
listBox1.Items.Insert(1,text);
```



Замена:

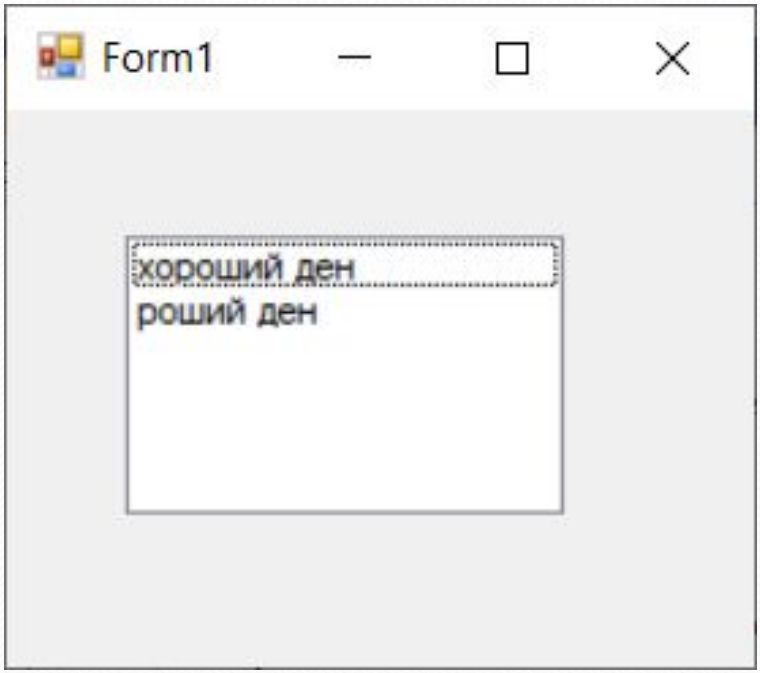
```
string text = "хороший день";
text = text.Replace("хороший", "плохой");
listBox1.Items.Add(text);
text = text.Replace("о", "оо");
listBox1.Items.Insert(1, text);
```



Примеры использования методов строк

Удаление:

```
string text = "хороший день";
// индекс последнего символа
int ind = text.Length - 1;
// вырезаем последний символ
text = text.Remove(ind);
listBox1.Items.Add(text);
// вырезаем первые два символа
text = text.Remove(0, 2);
listBox1.Items.Insert(1, text);
```



Класс StringBuilder

Хотя класс **System.String** предоставляет нам широкую функциональность по работе со строками, все таки он имеет свои недостатки. Прежде всего, объект **String** представляет собой **неизменяемую строку**. Когда мы выполняем какой-нибудь метод класса **String**, система создает новый объект в памяти с выделением ему достаточного места. Удаление первого символа - не самая затратная операция. Однако когда подобных операций множество, а объем текста, для которого надо выполнить данные операции, также не самый маленький, то издержки при потере производительности становятся более существенными.

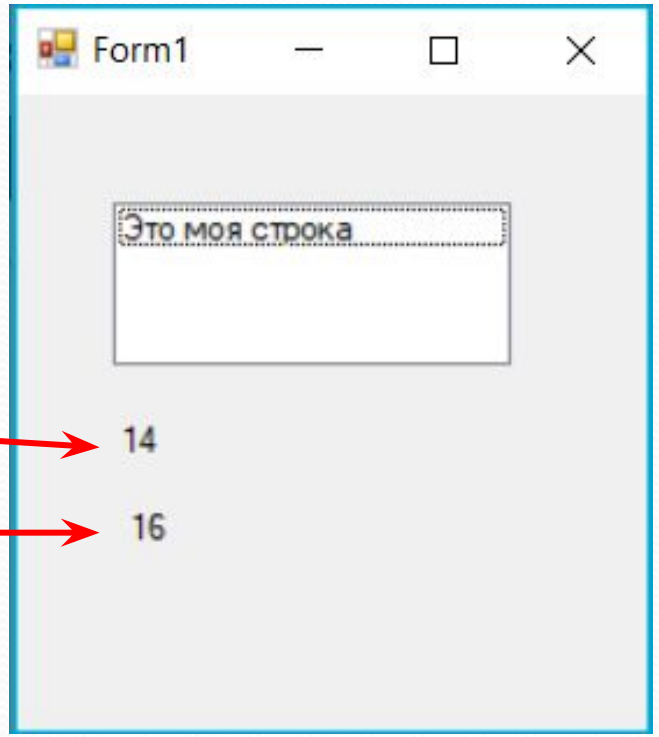
Чтобы выйти из этой ситуации во фреймворк .NET был добавлен новый класс **StringBuilder**, который находится в пространстве имен **System.Text**. Этот класс представляет динамическую (изменяемую) строку.

Класс StringBuilder

При создании строки **StringBuilder** выделяет памяти больше, чем необходимо этой строке:

```
StringBuilder sb = new StringBuilder("Это моя строка");
listBox1.Items.Add(sb);
label1.Text = sb.Length.ToString();
label2.Text = sb.Capacity.ToString();
```

При создании объекта StringBuilder выделяется память по умолчанию для 16 символов, так как длина начальной строки меньше 16.



длина строки

емкость выделенной памяти

Класс StringBuilder

Над строками этого класса определены практически те же операции с той же семантикой, что и над строками класса String:

- присваивание (=);
- две операции проверки эквивалентности (= =) и (!=);
- взятие индекса ([]).

Операция конкатенации (+) не определена над строками класса StringBuilder, ее роль играет метод Append, дописывающий новую строку в хвост уже существующей.

Со строкой этого класса можно работать как с массивом, но, в отличие от класса String, здесь уже все делается как надо: допускается не только чтение отдельного символа, но и его изменение.

Когда надо использовать класс String, а когда StringBulder?

Microsoft рекомендует использовать класс String в следующих случаях:

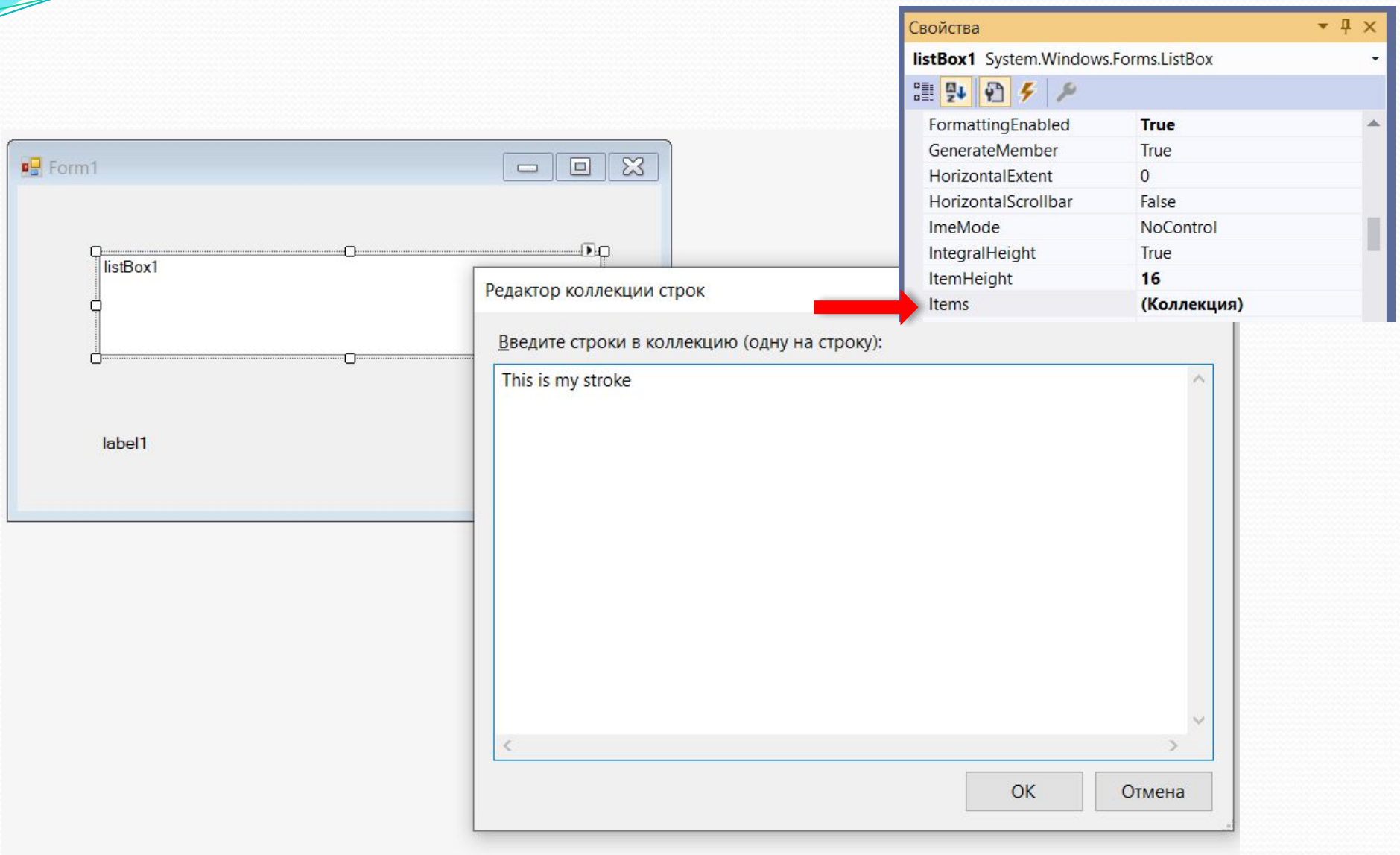
- При небольшом количестве операций и изменений над строками
- При выполнении фиксированного количества операций объединения. В этом случае компилятор может объединить все операции объединения в одну
- Когда надо выполнять масштабные операции поиска при построении строки, например **IndexOf** или **StartsWith**. Класс **StringBuilder** не имеет подобных методов.

Класс StringBuilder рекомендуется использовать в следующих случаях:

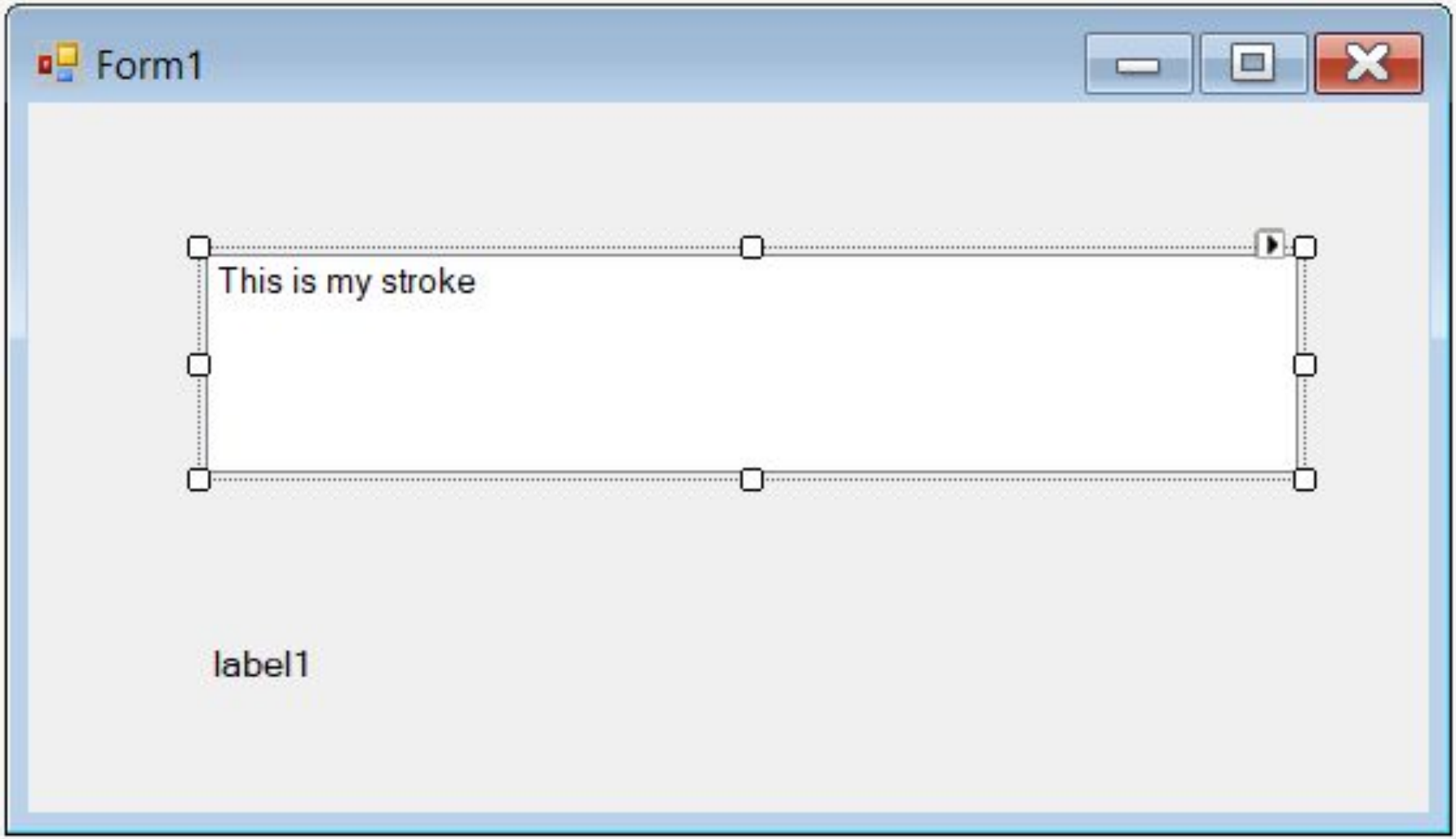
- При неизвестном количестве операций и изменений над строками во время выполнения программы
- Когда предполагается, что приложению придется сделать множество подобных операций

Пример: Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Сформировать новую строку, состоящую из чисел – длин слов в исходной строке.
Исходные данные вводить с помощью ListBox, строки вводятся на этапе проектирования формы, используя окно свойств. Вывод результата организовать в метку Label.

Форма в режиме конструирования



Форма в режиме конструирования



Листинг кода программы – обработчик события загрузки формы

```
private void Form1_Load(object sender, EventArgs e)
        {
            string Dlina = (string)listBox1.Items[0];
            string[] word = Dlina.Split(' ');
            string newstr = "";
            for (int i = 0; i < word.Length; i++)
            {
   newstr = newstr + " "+ word[i].Length.ToString();
            }
            label1.Text = newstr;
        }
```

Форма после запуска