Работа с обобщенной коллекцией List<T>

Лабораторная работа №8

Цель работы. В данной работе с помощью класса **List<T>** реализуются основные операции с коллекцией или со списком, построенным из элементов различного типа, в том числе сортировка списка по разным критериям.

Класс **List<T>**, который (как и класс **ArrayList)** в языке C# называется **коллекцией**, способен хранить объекты произвольного типа (или любого класса). Класс **List<T>** объявлен в пространстве имен **System.Collections.Generic**.

- В отличие от фиксированных статических массивов, в коллекциях List<T> и ArrayList размер увеличивается по мере необходимости, то есть классы List<T> и ArrayList поддерживают динамические массивы, расширяющиеся и сокращающиеся по мере необходимости.
- Массив на основе коллекции List<T> или ArrayList создается с первоначальным размером. Если этот размер превышается, то массив автоматически расширяется. При удалении объектов из такого массива он автоматически сокращается.

В ходе выполнения работы необходимо подготовить и оформить отчет, содержащий последовательность разработки приложения и его программного кода, комментарии операторов, результаты выполнения, необходимые пояснения, краткий вывод о результатах работы.

Общая последовательность выполнения включает определенные этапы. При завершении очередного шага рекомендуется сохранить проект, выполнить компиляцию и отладку.

1. Структура проекта приложения и компоненты главной формы

Структура проекта приложения должна включать два модуля:

- первый модуль главной формы для создания интерфейса приложения (предусмотреть ввод наименования книги, автора и года издания, кнопки и другие компоненты);
- второй модуль классов (без формы), содержащий определения классов для реализации задания.

На главной форме приложения используются следующие компоненты:

- **ListBox** для вывода списка строк;
- **TextBox** для ввода наименования дисциплины;
- **MaskedTextBox** для ввода года издания;
- **GroupBox**, **RadioButton** для типа сортировки;
- **Buttun** для реализации операций;
- **StatusStrip** для вывода номера работы и варианта, фамилии и группы в элементах ToolStripStatusLabel1 и ToolStripStatusLabel2 соответственно.

Для примера используется список:

1. C#. Программирование на языке высокого уровня, Павловская Т.А., 2009.
2. Visual C# 2010. Полный курс, Уотсон К. 2011.
3. Библия C#, Фленов М.Е., 2016.
4. Изучаем C#. Стиллмен Э., Грин Д., 2014.
5. Программирование на языке C#, Фаронов В.В., 2007.
6. Язык программирования C#. Хейлсберг А., Торгерсен М., 2012.

Примерный внешний вид приложения показан на рисунке 1.

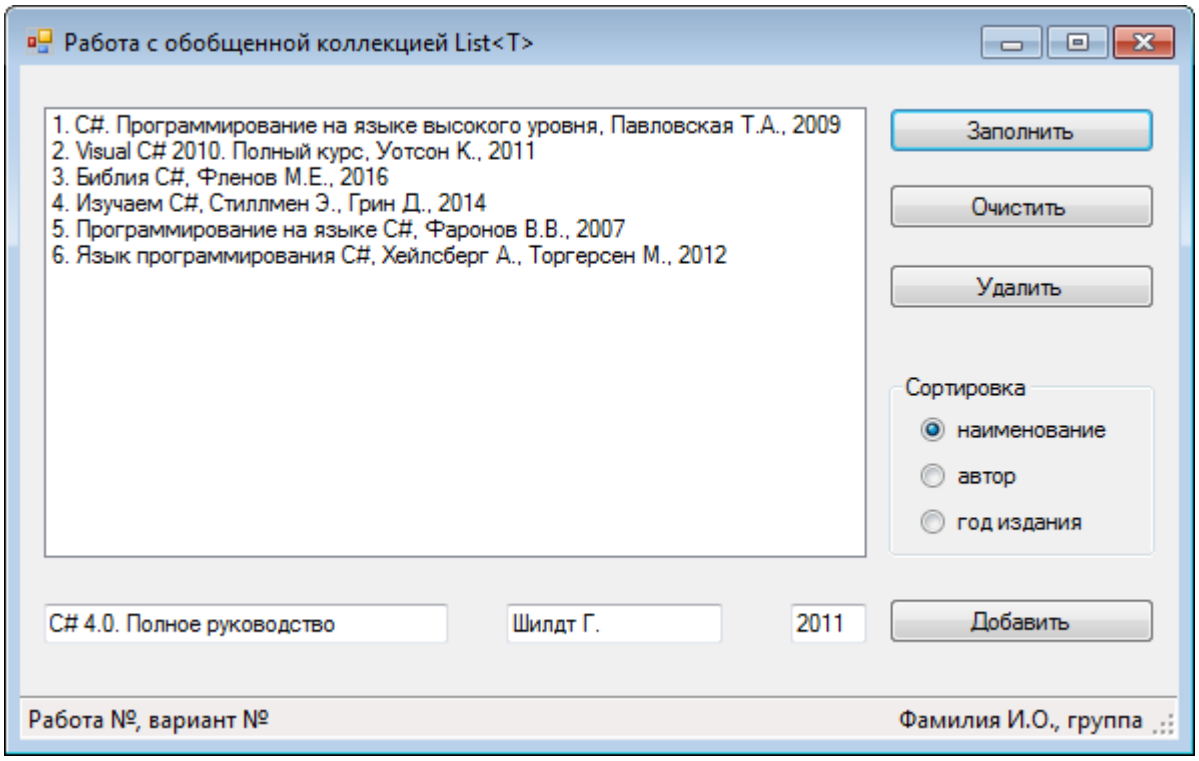

Рисунок 1 – Главная форма приложения

2. Объявление нового класса TBook, отписывающего одну книгу

Во втором модуле с классами создается новый класс **TBook**, содержащий параметры одной книги. Далее класс **TBook** будет использоваться в динамическом массиве, предназначенном для хранения списка книг.

```
// Определение класса (содержит описание одной книги)
public class TBook
{
    // поля класса
    internal string Title, Author;
    internal int Year;
    // конструкторы класса
    public TBook() { }   // умалчиваемый конструктор
    // конструктор с параметрами
    public TBook(string kTitle, string kAuthor, int kYear)
    {
        Title = kTitle;
        Author = kAuthor;
        Year = kYear;
    }
} // end of TBook
```

3. Объявление базового класса для коллекции книг

В модуле с классами определяется новый класс, например, **BaseBooks**, в котором создается динамический массив книг (коллекция книг) с помощью коллекции **List<T>** и класса **TBook.**

- Конструктор класса **BaseBooks** используется для заполнения коллекции книг.
- В классе **BaseBooks** также описывается метод **GetListBooks** для получения списка строк из коллекции для последующего вывода.

```
// Определение базового класса (содержит коллекцию книг)
public class BaseBooks
{
    // поле класса
    protected List<TBook> Books = new List<TBook>(); // коллекция книг
    public BaseBooks() // конструктор класса
    {
        // заполнить массив-список книг
        Books.Add(new TBook("C#. Программирование на языке высокого уровня",
                            "Павловская Т.А.", 2009));
        Books.Add(new TBook("Visual C# 2010. Полный курс", "Уотсон К.", 2011));
        Books.Add(new TBook("Библия C#", "Фленов М.Е.", 2016));
        Books.Add(new TBook("Изучаем C#", "Стиллмен Э., Грин Д.", 2014));
        Books.Add(new TBook("Программирование на языке C#", "Фаронов В.В.", 2007));
        Books.Add(new TBook("Язык программирования C#",
                            "Хейлсберг А., Торгерсен М.", 2012));
    }
    // вернуть список (массив) строк
    public string[] GetListBooks()
    {
        string[] ArrStr = new string[Books.Count]; // массив строк
        int i = 0;
        // цикл копирования Books в массив строк ArrStr
        foreach (TBook item in Books)
        {
        ArrStr[i] = (i+1) + ". " + item.Title +", "+ item.Author +", "+ item.Year;
        i++;
        }
        return ArrStr;
    }
} // end of BaseBooks
```

- В класс **BaseBooks** также необходимо добавить метод очистки коллекции **Books**.

4. Обработчик события для кнопки «Заполнить»

В первом модуле главной формы для создания обработчика кнопки «Заполнить» реализуется следующее:

- Объявление экземпляра (объекта), например, **ListBooks** класса **BaseBooks**.
- Инициализация объекта с помощью конструктора **BaseBooks().**
- Очистка **listBox1**.
- Вывод списка книг в **listBox1** с помощью методов **AddRange()** и **GetListBooks().**

5. Обработчик кнопки «Очистить»

Требуется очистить **listBox1** и коллекцию книг с помощью метода очистки коллекции **Books** класса **BaseBooks**.

Далее следует сохранить проект приложения, выполнить компиляцию, отладить код и проверить работу созданных обработчиков.

6. Объявление производного класса с новыми методами

Но основе **BaseBooks** необходимо определить новый производный класс, в котором определяются новые методы: удаление, добавление, сортировка по критериям:

- метод удаления по индексу (номеру) записи в коллекции книг
```
public void RemBook(int n)
{
    // удалить по индексу
    Books.RemoveAt(n);
}
```
- метод добавления книги без поиска и проверки повторного ввода
```
public void AddBook(string t, string a, int y)
{
    // добавить (без проверки повторного ввода)
    Books.Add(new TBook(t, a, y));
}
```
- метод для сортировки по критерию и формирования обновленного списка строк
```
public string[] GetListby(IComparer<TBook> compare)
{
    // сортировать и вернуть список
    Books.Sort(compare);
    return GetListBooks();
}
```

В дальнейшем реализацию всех операций задания следует выполнять с помощью созданного дочернего (производного) класса и его экземпляра.

7. Удаление книги из списка. Обработчик кнопки «Удалить»

Первая версия данного обработчика может содержать следующий код:

```
if (listBox1.SelectedItem != null)
{
    // определить индекс выбранной строки
    int index = listBox1.Items.IndexOf(listBox1.SelectedItem);
    // удалить книгу по индексу
    ListBooks.RemBook(index);

    listBox1.Items.Clear();
    // вывести список книг, используя метод GetListBooks
    listBox1.Items.AddRange(ListBooks.GetListBooks());
}
else MessageBox.Show("Для удаления не выбрана строка!", "Ошибка удаления");
```

Здесь **ListBooks** является экземпляром дочернего класса.

Если строка не выбрана, выдается сообщение об ошибке (рисунок 2).

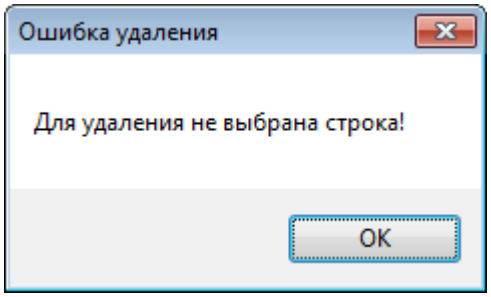

Рисунок 2 – Сообщение об ошибке удаления

8. Добавление книги. Обработчик кнопки «Добавить»

Первая версия данного обработчика (без проверки повторного ввода) может содержать следующий код:

```
string stitle = textBox1.Text.Trim();
string sauthor = textBox2.Text.Trim();

if (stitle == "" || sauthor.Length == 0 || maskedTextBox1.Text.Length == 0)
{
    MessageBox.Show("Пустая строка!", "Ошибка ввода");
    return; // выйти из обработчика
}
// вызвать метод добавления
ListBooks.AddBook(stitle, sauthor, int.Parse(maskedTextBox1.Text));

listBox1.Items.Clear();
// вывести список книг, используя метод GetListBooks
listBox1.Items.AddRange(ListBooks.GetListBooks());
```

Если хотя-бы одно из полей ввода не содержит вводимых данных, выдается соответсвующее сообщение об ошибке (рисунок 3).

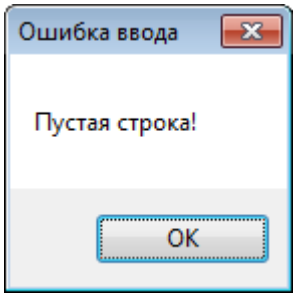

Рисунок 3 – Сообщение об ошибке ввода

9. Объявление компараторов для сравнения и сортировки

В модуле с классами объявляются компараторы, в которых с помощью метода **Compare** осуществляется сравнение элементов коллекции книг по разным критериям. Компараторы в дальнейшем применяются для сортировки коллекции книг. Классы компараторов наследуют интерфейс **IComparer.**

```
// классы компараторов для сравнения и сортировки
public class myIComparer : IComparer<TBook>
{
    virtual public int Compare(TBook a, TBook b)
    {
         return string.Compare(a.Title, b.Title);
    }
}
public class myIComparerAuthor : myIComparer
{
    override public int Compare(TBook a, TBook b)
    {
        return string.Compare(a.Author, b.Author);
    }
}
public class myIComparerYear : myIComparer
{
    override public int Compare(TBook a, TBook b)
    {
        if (a.Year > b.Year) return 1;
        if (a.Year < b.Year) return -1;
        else return 0;
    }
}
```

10. Сортировка списка книг по разным параметрам

- В главном модуле в классе **Form1** необходимо создать новый метод, который проверяет свойство **Checked** радиокнопок типа **RadioButton** и с помощью метода **GetListby()** с параметром сортирует список:

```
private void OutList(object sender, EventArgs e)
{
    listBox1.Items.Clear();

    if (radioButton1.Checked) // вывод списка по наименованию
      listBox1.Items.AddRange(ListBooks.GetListby(new myIComparer()));

    else if (radioButton2.Checked) // вывод списка по автору
      listBox1.Items.AddRange(ListBooks.GetListby(new myIComparerAuthor()));

    else if (radioButton3.Checked) // вывод списка по году
      listBox1.Items.AddRange(ListBooks.GetListby(new myIComparerYear()));
}
```

- Для каждой радиокнопки событие **Click** связать с созданным методом **OutList().**

На данном этапе следует сохранить проект приложения, выполнить компиляцию, отладить код и проверить работу созданных методов и обработчиков.

11. Дополнительное сообщение при удалении

Перед удалением необходимо вывести диалоговое окно, показанное на рисунке 4.

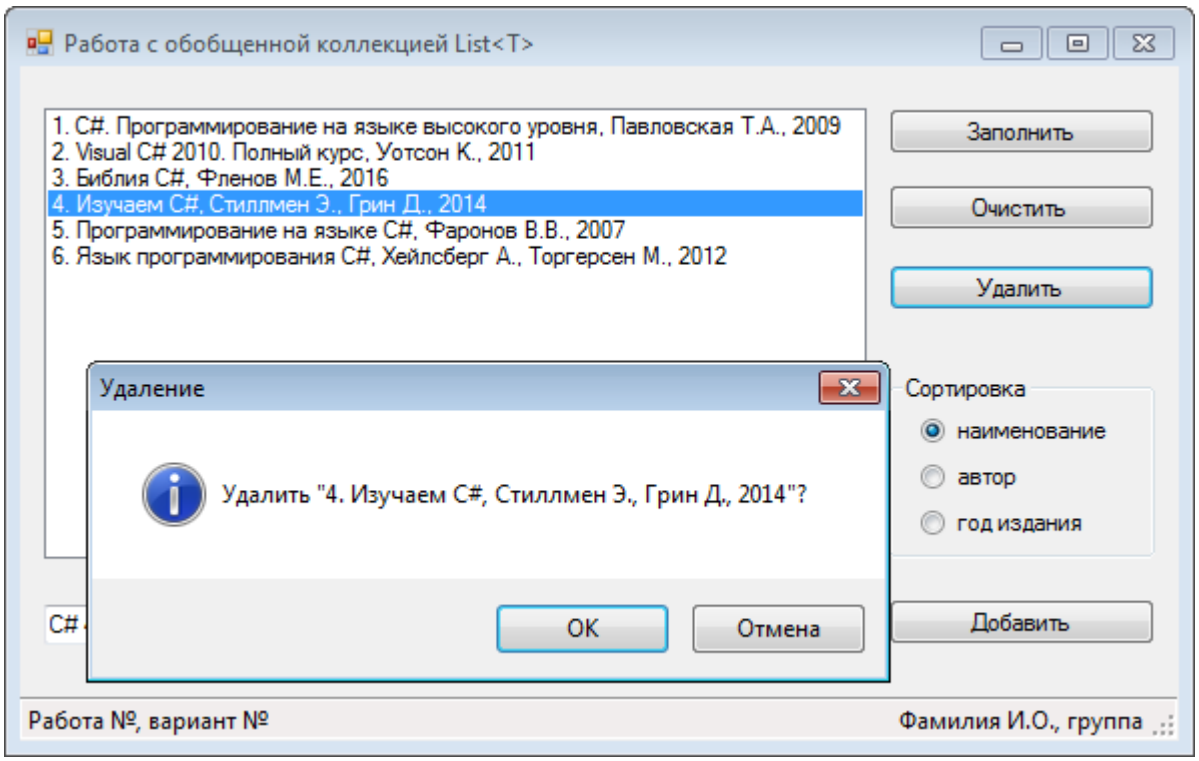

Рисунок 4 – Сообщение перед удалением

12. Дополнительная обработка ошибок ввода

- Обработать, исключить повторный ввод наименования книги. Для поиска существующего наименования рекомендуется использовать встроенный в коллекции **List<T>** метод **BinarySearch**. В обработчике кнопки «**Добавить**» необходимо предусмотреть вывод соответствующего сообщения об ошибке (рисунок 5).

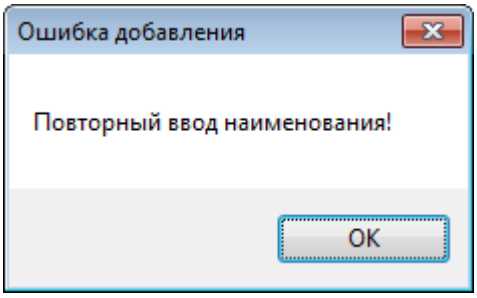

Рисунок 5 – Сообщение об ошибке повторного ввода

- Обработать ошибку добавления новой книги, если список книг не создан (рисунок 6).

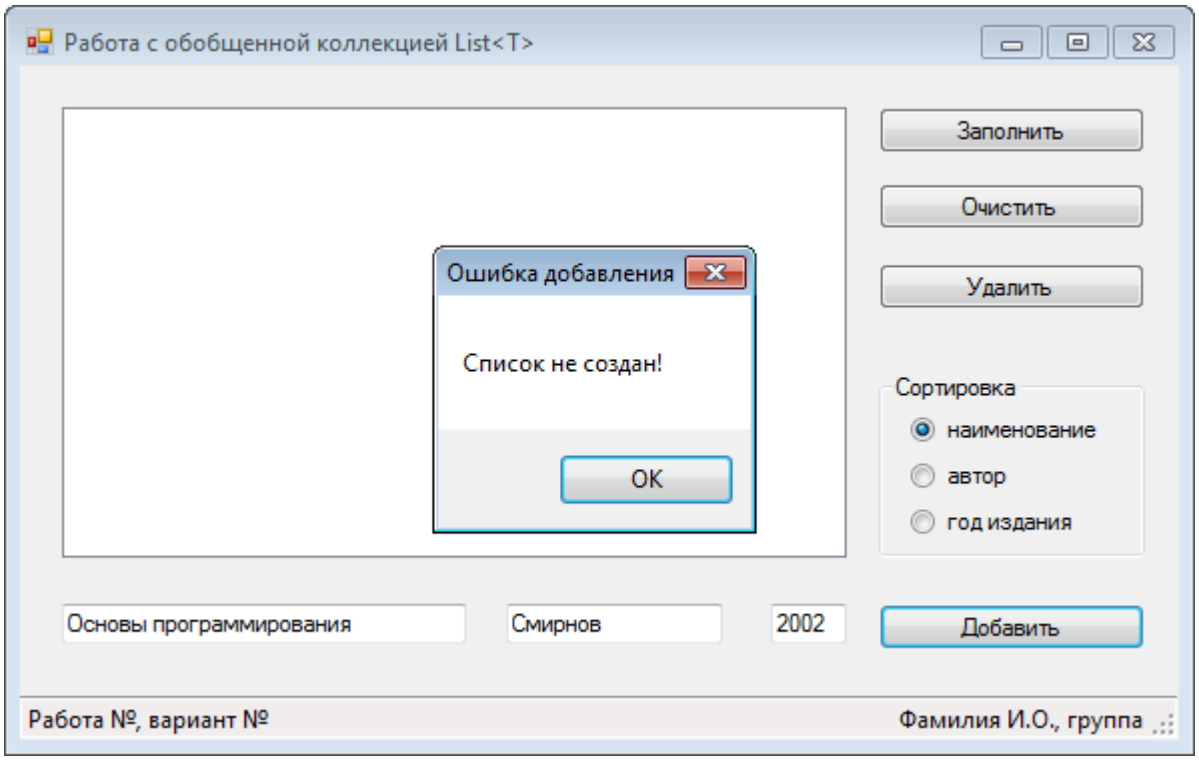

Рисунок 6 – Сообщение об ошибке добавления

Следует сохранить проект приложения, выполнить компиляцию, отладить код, проверить работу созданных методов и обработку ошибок ввода, удаления и добавления.

13. Построение диаграммы классов приложения

Необходимо создать и отобразить диаграмму (схему) классов приложения (рисунок 7).

Рисунок 7 – Схема классов приложения

В ходе выполнения заданий работы необходимо подготовить и оформить отчет, содержащий последовательность разработки приложения и его программного кода, комментарии операторов, результаты выполнения, необходимые пояснения, краткий вывод в заключении о результатах работы. При сохранении файла отчета и архива проекта приложения следует использовать фамилию студента и номер работы.