Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

Делегаты, лямбда-выражения, события

тема

Вариант 14

Преподаватель \_\_\_\_\_\_\_\_\_\_\_ А.А. Чикизов

подпись, дата инициалы, фамилия

Студент КИ15–16Б, 031510065 \_\_\_\_\_\_\_\_\_\_\_ Т.В. Радионов

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2017

# Задание

В лабораторной работе требуется определить класс-коллекцию NoteCollection, содержащий в себе элементы типа Note. Для хранения элементов использовать стандартные коллекции.

NoteCollection изменяется при добавлении, удалении элементов Note (событие count change) или изменении одного из элементов (ссылки) в коллекции (событие reference change). В этом случае в соответствующих методах или свойствах класса NoteCollection вызываются(invoke) специально объявленные делегаты.

Для хранения и вызова в NoteCollection методов «подписанных» классов (в нашем случаем это будут методы класса Journal) необходимо использовать закрытые делегаты и открытые события. Подписку и отписку методов проводить только через события и их служебные методы (add, remove), никакого прямого доступа к закрытым делегатам быть не должно.

Для сортировки элементов коллекции необходимо использовать метод Sort (лямбда-выражение с полем сортировки). Использование лямбда-выражений для других целей приветствуется.

Определить класс хранения информации о событии NoteListHandlerEventArgs, производный от класса EventArgs включающий в себя

• открытое автосвойство типа string с названием коллекции, в которой произошло событие;

• открытое автосвойство типа string с информацией о типе изменений в коллекции; (типы изменений назвать самостоятельно)

• открытое автосвойство типа Note для хранения ссылки на объект, с которым связаны изменения;

• конструкторы для инициализации класса;

• перегруженную версию метода string ToString() для формирования строки с информацией обо всех полях класса.

Объявить новый тип данных - делегат NoteListHandler с сигнатурой:

delegate void NoteListHandler

(object source, NoteListHandlerEventArgs args);

Определить класс NoteCollection, который содержит в себе

• закрытое поле типа List<Note>;

• автосвойство типа string с названием коллекции;

• метод bool Add (int j, NOTE record) для добавления элемента record в позицию j списка List<Note>; если в списке нет элемента с номером j, метод ничего не добавляет и возвращает значение false;

• метод bool Remove (int j) для удаления элемента с номером j из списка List<Note>; если в списке нет элемента с номером j, метод возвращает значение false;

• индексатор this типа Note (с методами get и set) с целочисленным индексом для доступа к элементу списка List<Note> с заданным номером.

Названия событий в NoteCollection (тип делегата NoteListHandler)

• onNoteCountChanged - происходит при добавлении нового элемента в коллекцию или при удалении элемента из коллекции; через объект NoteListHandlerEventArgs cотбытие передает имя коллекции, строку с информацией о том, что в коллекцию был добавлен новый элемент или из нее был удален элемент, ссылку на добавленный или удаленный элемент Note;

• onNoteReferenceChanged - происходит, когда одной из ссылок, входящих в коллекцию, присваивается новое значение; через объект NoteListHandlerEventArgs событие передает имя коллекции, строку с информацией о том, что был заменен элемент в коллекции, и ссылку на новый элемент Note.

Событие NoteCountChanged вызывают следующие методы класса NoteCollection

• AddDefaults(); - заполнение коллекции произвольными заранее заданными данными

• Add (int j, Note record) ; - добавление в коллекцию элемента

• Remove (int j) – удаление j элемента

\*Событие NoteReferenceChanged вызывается в методе set индексатора, определенного в классе NoteCollection.

Определить класс Journal, который можно использовать для накопления информации об изменениях в коллекциях типа NoteCollection. Класс

Journal хранит информацию в списке объектов типа JournalEntry. Каждый элемент списка содержит информацию об отдельном изменении, которое произошло в коллекции.

Класс JournalEntry содержит

• открытое автосвойство типа string с названием коллекции, в которой произошло событие;

• открытое автосвойство типа date с информацией о времени, когда произошло событие;

• открытое автосвойство типа string с информацией о типе изменений в коллекции;

• открытое автосвойство типа string c данными объекта Note, с которым связаны изменения в коллекции;

• конструктор для инициализации полей класса;

• перегруженную версию метода string ToString().

Класс Journal содержит

• закрытое поле типа List<JournalEntry>;

• реализует обработчики NoteCountChanged и NoteReferenceChanged (для подписки на события в NoteCollection) которые при вызове добавляют новый элемент JournalEntry в список List<JournalEntry>; для инициализации JournalEntry используется информация из объекта NoteListHandlerEventArgs, который передается вместе с событием;

• перегруженную версию метода string ToString() для формирования строки с информацией обо всех элементах списка List<JournalEntry>.

В методе Main()

1. Создать две коллекции NoteCollection.

2. Создать объекта типа Journal и подписать его на события onNoteCountChanged и onNoteReferenceChanged всех коллекций NoteCollection

3. Внести изменения в коллекциях NoteCollection

• добавить элементы в коллекции;

• удалить некоторые элементы из коллекций;

• присвоить некоторым элементам коллекций новые значения;

• провести сортировку коллекции по разным полям

• провести сортировку журнала по типам операции

4. Вывод программы должен показать работу всех используемых технологий

# Исходный код

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

NoteCollection notes = new NoteCollection("Коллекция 1");

Journal journal = new Journal();

notes.OnNoteCountChanged += journal.NoteCountChanged; // Подписка метода NoteCountChanged класса Journal

notes.OnNoteReferenceChanged += journal.NoteReferenceChanged; // Подписка метода NoteReferenceChanged класса Journal

notes.AddDefaults(); // Добавляем заготовленные объекты

notes.Remove(1); // Удаляем объект из списка с индексом 1

notes[0] = notes[2]; // Заменяем данные объекта с индексом 0 на данные объекта с индексом 2

Console.WriteLine(journal.ToString()); // Вывод на консоль

notes.SortBySurnameName();

Console.WriteLine("Сортировка по фамилии и имени:\n" + notes.ToString());

notes.SortByPhone();

Console.WriteLine("Сортировка по телефону:\n" + notes.ToString());

notes.SortByDOB();

Console.WriteLine("Сортировка по дате рождения:\n" + notes.ToString());

journal.SortByTypeChange();

Console.WriteLine("Сортировка в журнале по типу изменения:\n" + journal.ToString());

Console.ReadKey();

}

}

class Journal

{

private List<JournalEntry> journal = new List<JournalEntry>();

/// <summary>

/// Запись в журнал события изменения количества объектов в коллекции

/// </summary>

/// <param name="sourse"></param>

/// <param name="args"></param>

public void NoteCountChanged(object sourse, NoteListHandlerEventArgs args)

{

journal.Add(new JournalEntry(args.NameCollection, DateTime.Now, args.TypeChange, args.Note.ToString()));

}

/// <summary>

/// Запись в журнал события изменения ссылки в коллекции

/// </summary>

/// <param name="sourse"></param>

/// <param name="args"></param>

public void NoteReferenceChanged(object sourse, NoteListHandlerEventArgs args)

{

journal.Add(new JournalEntry(args.NameCollection, DateTime.Now, args.TypeChange, args.Note.ToString()));

}

/// <summary>

/// Сортировка по типу изменения

/// </summary>

public void SortByTypeChange()

{

journal.Sort((x, y) => x.TypeChange.CompareTo(y.TypeChange));

}

/// <summary>

/// Перегрузка метода ToString

/// </summary>

/// <returns></returns>

public override string ToString()

{

var result = "";

for (var i = 0; i < journal.Count; i++)

result += journal[i].ToString() + "\n";

return result;

}

}

class JournalEntry

{

public string NameCollection { get; set; }

public DateTime Time { get; set; }

public string TypeChange { get; set; }

public string DataNote { get; set; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="nameCollection"></param>

/// <param name="time"></param>

/// <param name="typeChange"></param>

/// <param name="dataNote"></param>

public JournalEntry(string nameCollection, DateTime time, string typeChange, string dataNote)

{

NameCollection = nameCollection;

Time = time;

TypeChange = typeChange;

DataNote = dataNote;

}

/// <summary>

/// Перегрузка метода ToString

/// </summary>

/// <returns></returns>

public override string ToString()

{

return String.Format("Название коллекции: {0}; Время: {1}; Тип изменения: {2}; Данные о Note: {3}",

NameCollection, Time, TypeChange, DataNote);

}

}

class Note : IComparable<Note>

{

public string SurnameName { get; set; }

public string Phone { get; set; }

public string DOB { get; set; }

/// <summary>

/// Конструктор с кортежем

/// </summary>

/// <param name="note"></param>

public Note(Tuple<string, string, string> note)

{

SurnameName = note.Item1;

Phone = note.Item2;

DOB = note.Item3;

}

/// <summary>

/// Перегрузка метода ToString()

/// </summary>

/// <returns></returns>

public override string ToString()

{

return String.Format("Фамилия Имя: {0}; Телефон: {1}; Дата рождения: {2}", SurnameName, Phone, DOB);

}

/// <summary>

/// Реализация интерфейса IComparable

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

public int CompareTo(Note obj)

{

return Phone.CompareTo(obj.Phone);

}

}

class NoteCollection

{

// Свойства класса

public string NameCollection { get; set; } = "NoteCollection";

// Инициализация делегатов и событий

NoteListHandler NoteCountChanged; // Иницилизация делегата NoteListHandler

public event NoteListHandler OnNoteCountChanged // Подпискаи отписка метода через событие (обычное описание события с добавлением add и remove)

{

add { NoteCountChanged += value; }

remove { NoteCountChanged -= value; }

}

NoteListHandler NoteReferenceChanged;

public event NoteListHandler OnNoteReferenceChanged

{

add { NoteReferenceChanged += value; }

remove { NoteReferenceChanged -= value; }

}

// Список

List<Note> notes = new List<Note>();

/// <summary>

/// Конструктор

/// </summary>

/// <param name="nameCollection"></param>

public NoteCollection(string nameCollection)

{

NameCollection = nameCollection;

}

/// <summary>

/// Добавление элемента в список по заданному индексу

/// </summary>

/// <param name="j"></param>

/// <param name="record"></param>

/// <returns></returns>

public bool Add (int j, Note record)

{

try

{

notes.Insert(j, record);

}

catch (ArgumentOutOfRangeException)

{

return false;

}

NoteCountChanged?.Invoke(this, new NoteListHandlerEventArgs(NameCollection, "Добавление в список", notes[j]));

return true;

}

/// <summary>

/// Удаление элемента из списка по заданному индексу

/// </summary>

/// <param name="j"></param>

/// <returns></returns>

public bool Remove (int j)

{

Note oldNote = notes[j];

try

{

notes.RemoveAt(j);

}

catch (ArgumentOutOfRangeException)

{

return false;

}

NoteCountChanged?.Invoke(this, new NoteListHandlerEventArgs(NameCollection, "Удаление из списка", oldNote));

return true;

}

/// <summary>

/// Индекстатор this типа Note для доступа к элементу списка по индексу

/// </summary>

/// <param name="index"></param>

/// <returns></returns>

public Note this[int index]

{

get

{

return notes[index];

}

set

{

NoteReferenceChanged?.Invoke(this, new NoteListHandlerEventArgs(NameCollection, "Измененяется", notes[index]));

notes[index] = value;

NoteReferenceChanged?.Invoke(this, new NoteListHandlerEventArgs(NameCollection, "Изменили", notes[index]));

}

}

/// <summary>

/// Заполнение списка заготовленными объектами

/// </summary>

public void AddDefaults()

{

Note note = new Note(new Tuple<string, string, string>("Иван Васильевич", "+7 (987) 654-32-11", "01.11.1993"));

notes.Add(note);

NoteCountChanged?.Invoke(this, new NoteListHandlerEventArgs(NameCollection, "Добавление в список", note));

note = new Note(new Tuple<string, string, string>("Петр Петрович", "+7 (123) 456-78-90", "01.11.1997"));

notes.Add(note);

NoteCountChanged?.Invoke(this, new NoteListHandlerEventArgs(NameCollection, "Добавление в список", note));

note = new Note(new Tuple<string, string, string>("Иван Иванович", "+7 (111) 222-33-44", "01.11.1992"));

notes.Add(note);

NoteCountChanged?.Invoke(this, new NoteListHandlerEventArgs(NameCollection, "Добавление в список", note));

note = new Note(new Tuple<string, string, string>("Сергей Сергеевич", "+7 (222) 333-44-55", "01.11.1991"));

notes.Add(note);

NoteCountChanged?.Invoke(this, new NoteListHandlerEventArgs(NameCollection, "Добавление в список", note));

note = new Note(new Tuple<string, string, string>("Виталий Витальевич", "+7 (333) 444-55-66", "01.11.1995"));

notes.Add(note);

NoteCountChanged?.Invoke(this, new NoteListHandlerEventArgs(NameCollection, "Добавление в список", note));

}

/// <summary>

/// Сортировка по фамилии и имени

/// </summary>

public void SortBySurnameName()

{

notes.Sort((x, y) => x.SurnameName.CompareTo(y.SurnameName));

}

/// <summary>

/// Сортировка по телефону

/// </summary>

public void SortByPhone()

{

notes.Sort((x, y) => x.Phone.CompareTo(y.Phone));

}

/// <summary>

/// Сортировка по дате рождения

/// </summary>

public void SortByDOB()

{

notes.Sort((x, y) => x.DOB.CompareTo(y.DOB));

}

/// <summary>

/// Перегрузка метода ToString

/// </summary>

/// <returns></returns>

public override string ToString()

{

var result = "";

for (var i = 0; i < notes.Count; i++)

result += notes[i].ToString() + "\n";

return result;

}

}

class NoteListHandlerEventArgs : EventArgs

{

public string NameCollection { get; set; }

public string TypeChange { get; set; }

public Note Note { get; set; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="nameCollection"></param>

/// <param name="typeChange"></param>

/// <param name="note"></param>

public NoteListHandlerEventArgs (string nameCollection, string typeChange, Note note)

{

NameCollection = nameCollection;

TypeChange = typeChange;

Note = note;

}

/// <summary>

/// Перегрузка метода ToString

/// </summary>

/// <returns></returns>

public override string ToString()

{

return String.Format("название коллекции: {0}; Тип изменения: {1}; Объект Note: {2}", NameCollection, TypeChange, Note);

}

}

}

# Результы

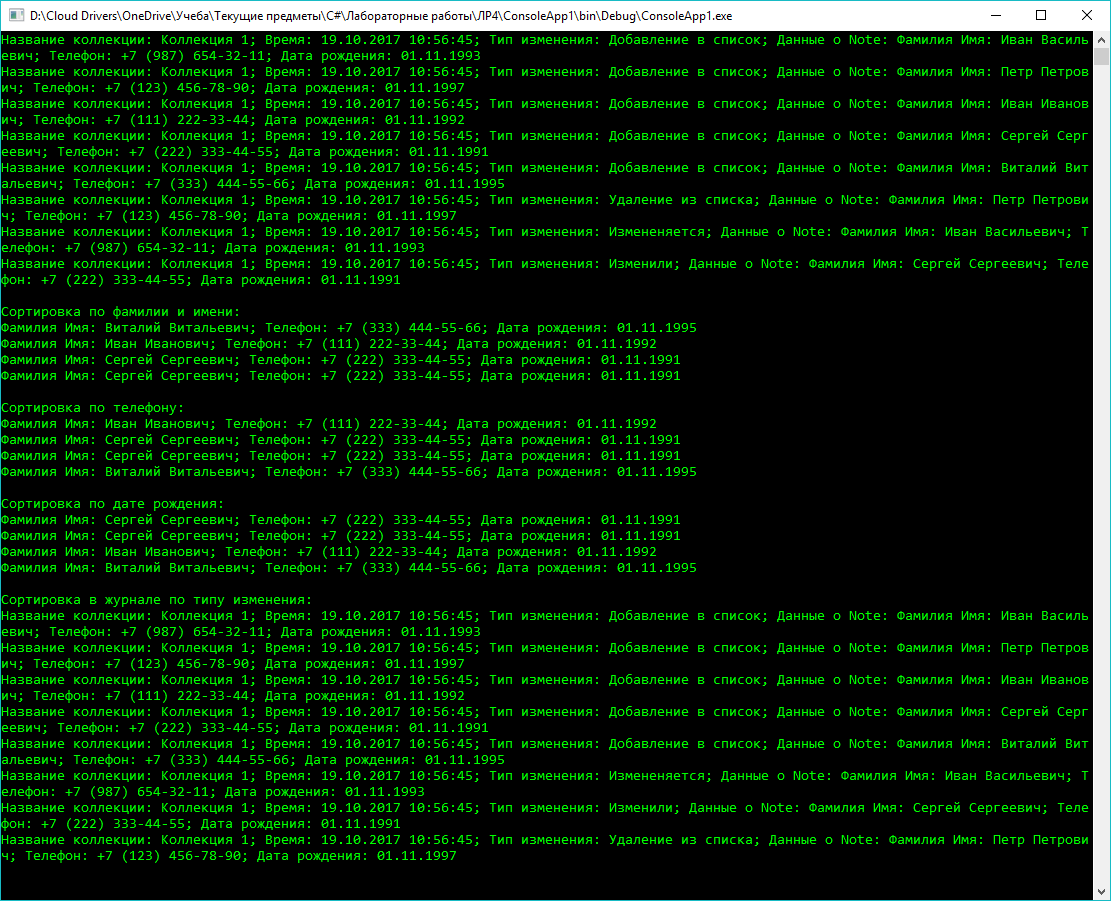


Рисунок 1 – Пример работы программы

ПРИЛОЖЕНИЕ А

Дополнительное задание: добавить метод Sort(), который будет принимать лямбда-выражение для сортировки по определенному полю.

Реализация метода в классе NoteCollection:

public void Sort(Comparison<Note> comparison)

{

notes.Sort(comparison);

}

Вызов метода:

notes.Sort((x, y) => x.SurnameName.CompareTo(y.SurnameName));

Console.WriteLine("Сортировка по фамилии и имени:\n" + notes.ToString());

notes.Sort((x, y) => x.Phone.CompareTo(y.Phone));

Console.WriteLine("Сортировка по телефону:\n" + notes.ToString());

notes.Sort((x, y) => x.DOB.CompareTo(y.DOB));

Console.WriteLine("Сортировка по дате рождения:\n" + notes.ToString());