

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Брянский государственный технический университет

***Утверждаю***

***Ректор университета***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О.Н.Федонин***

***«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2017 г.***

***Программирование в среде Microsoft .NET***

***Проектирование классов. События и исключения***

***Методические указания***

***к выполнению лабораторной работы №6***

***для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия»***

***Брянск 2017***

*УКД 004.43*

*Программирование в среде Microsoft .Net. Проектирование классов. События и исключения [Текст] + [Электронный ресурс]: Методические указания к выполнению лабораторной работы №6 для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия». – Брянск: БГТУ, 2017. – 27 с.*

*Разработали:*

*Д.А.Коростелёв*

*канд. техн. наук, доц.*

*Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол №2 от 16.09.2016)*

*Научный редактор А.А.Азарченков*

*Редактор издательства Л.И.Афонина*

*Компьютерный набор Д.А.Коростелёв*

*Темплан 2017 г., п.273*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Подписано в печать 23.09.17. Формат 60х84 1/16 Бумага офсетная. Офсетная печать. Усл.печ.л. 1,8. Уч.-изд.л. 1,8 Тираж 1 экз. Заказ Бесплатно.*

*Издательство Брянского государственного технического университета*

*241035, Брянск, бульвар 50-летия Октября, 7, БГТУ. 58-82-49.*

*Лаборатория оперативной полиграфии БГТУ, ул. Институтская, 16.*

***Цель работы***

*Целью работы является изучение принципов использования событий и исключений при проектировании классов.*

*Продолжительность работы – 2 часа.*

***Проектирование классов***

*Рассмотрим использование событий и исключений при проектировании классов на примере организации работы гостиницы. Для этой цели внесём некоторые изменения в библиотеку классов, полученную в предыдущей лабораторной работе.*

*Для начала опишем 3 новых класса исключительных ситуаций:*

1. *Исключение о некорректно заполненной информации о клиенте:*

*using System;*

*using System.Runtime.Serialization;*

*namespace ClassLibraryHotel.Exception*

*{*

*[Serializable]*

*public class InvalidClientException : System.Exception*

*{*

*public InvalidClientException()*

*{*

*}*

*public InvalidClientException(string message) : base(message)*

*{*

*}*

*public InvalidClientException(string message, System.Exception inner) : base(message, inner)*

*{*

*}*

*protected InvalidClientException(*

*SerializationInfo info,*

*StreamingContext context) : base(info, context)*

*{*

*}*

*}*

*}*

1. *Исключение о некорректно заполненной информации о номере:*

*using System;*

*using System.Runtime.Serialization;*

*namespace ClassLibraryHotel.Exception*

*{*

*[Serializable]*

*public class InvalidRoomException : System.Exception*

*{*

*public InvalidRoomException()*

*{*

*}*

*public InvalidRoomException(string message) : base(message)*

*{*

*}*

*public InvalidRoomException(string message, System.Exception inner) : base(message, inner)*

*{*

*}*

*protected InvalidRoomException(*

*SerializationInfo info,*

*StreamingContext context) : base(info, context)*

*{*

*}*

*}*

*}*

1. *Исключение о некорректно заполненной информации о поселении:*

*using System;*

*using System.Runtime.Serialization;*

*namespace ClassLibraryHotel.Exception*

*{*

*[Serializable]*

*public class InvalidSettlementException : System.Exception*

*{*

*public InvalidSettlementException()*

*{*

*}*

*public InvalidSettlementException(string message) : base(message)*

*{*

*}*

*public InvalidSettlementException(string message, System.Exception inner) : base(message, inner)*

*{*

*}*

*protected InvalidSettlementException(*

*SerializationInfo info,*

*StreamingContext context) : base(info, context)*

*{*

*}*

*}*

*}*

*Чтобы обеспечить целостность коллекций внесем некоторые изменения в класс Hotel. Организуем доступ к данным классу Hotel не через статические свойства, а через единственно возможный экземпляр класса, создаваемый с помощью приватного конструктора класса.*

*private static Hotel \_instance;*

*/// <summary>*

*/// Единственный экземпляр класса Отель*

*/// </summary>*

*public static Hotel Instance*

*{*

*get*

*{*

*if (\_instance == null)*

*{*

*\_instance = new Hotel();*

*}*

*return \_instance;*

*}*

*}*

*/// <summary>*

*/// Приватный конструктор*

*/// </summary>*

*private Hotel()*

*{*

*}*

*Сами статические свойства сделаем нестатическими, возвращающими интерфейсы IEnumerable<T>, а для хранения данных будем использовать приватные поля с данными.*

*/// <summary>*

*/// Словарь клиентов*

*/// </summary>*

*private Dictionary<int, Client> \_clients = new Dictionary<int, Client>();*

*/// <summary>*

*/// Словарь номеров*

*/// </summary>*

*private Dictionary<int, Room> \_rooms = new Dictionary<int, Room>();*

*/// <summary>*

*/// Спосок поселений*

*/// </summary>*

*private List<Settlement> \_settlements = new List<Settlement>();*

*/// <summary>*

*/// Коллекция клиентов*

*/// </summary>*

*public IEnumerable<Client> Clients*

*{*

*get { return \_clients.Values.AsEnumerable(); }*

*}*

*/// <summary>*

*/// Коллекция номеров*

*/// </summary>*

*public IEnumerable<Room> Rooms*

*{*

*get*

*{*

*return \_rooms.Values.AsEnumerable();*

*}*

*}*

*/// <summary>*

*/// Коллекция поселений*

*/// </summary>*

*public IEnumerable<Settlement> Settlements*

*{*

*get*

*{*

*return \_settlements;*

*}*

*}*

*В класс Hotel добавим описание событий, которые будут генерироваться при добавлении и удалении элементов.*

*public event EventHandler ClientAdded;*

*public event EventHandler RoomAdded;*

*public event EventHandler SettlementAdded;*

*public event EventHandler ClientRemoved;*

*public event EventHandler RoomRemoved;*

*public event EventHandler SettlementRemoved;*

*Для изменения коллекций реализуем специальные методы в классе Hotel.*

*/// <summary>*

*/// Добавление клиента*

*/// </summary>*

*/// <param name="client">Информация о клиенте</param>*

*public void AddClient(Client client)*

*{*

*if (!client.IsValid)*

*{*

*throw new InvalidClientException("Информация о клиент заполнена некорректно");*

*}*

*try*

*{*

*\_clients.Add(client.ClientId, client);*

*//Герерируем событие о том, что клиент добавлен*

*ClientAdded?.Invoke(client, EventArgs.Empty);*

*}*

*catch (System.Exception exception)*

*{*

*throw new InvalidClientException("При добавлении клиента произошла ошибка", exception);*

*}*

*}*

*/// <summary>*

*/// Добавление номера*

*/// </summary>*

*/// <param name="room">Информация о номере</param>*

*public void AddRoom(Room room)*

*{*

*if (!room.IsValid)*

*{*

*throw new InvalidRoomException("Информация о номере заполнена некорректно");*

*}*

*try*

*{*

*\_rooms.Add(room.Number, room);*

*//Герерируем событие о том, что номер добавлен*

*RoomAdded?.Invoke(room, EventArgs.Empty);*

*}*

*catch (System.Exception exception)*

*{*

*throw new InvalidRoomException("При добавлении номера произошла ошибка", exception);*

*}*

*}*

*/// <summary>*

*/// Информация о поселении*

*/// </summary>*

*/// <param name="settlement"></param>*

*public void AddSettlement(Settlement settlement)*

*{*

*if (!settlement.IsValid)*

*{*

*throw new InvalidSettlementException("Информация о заселении заполнена некорректно");*

*}*

*try*

*{*

*\_settlements.Add(settlement);*

*//Герерируем событие о том, что информация о поселении добавлена*

*SettlementAdded?.Invoke(settlement, EventArgs.Empty);*

*}*

*catch (System.Exception exception)*

*{*

*throw new InvalidSettlementException("При поселении произошла ошибка", exception);*

*}*

*}*

*/// <summary>*

*/// Удалить клиента по идентификатору*

*/// </summary>*

*/// <param name="clientKey">Идентификатор клиента</param>*

*public void RemoveClient(int clientKey)*

*{*

*\_clients.Remove(clientKey);*

*//Генерируем событие о том, что клиент удалён*

*ClientRemoved?.Invoke(clientKey, EventArgs.Empty);*

*//Получаем список сведений о поселении клиента*

*var settlementsForClient = Settlements.Where(s => s.Client.ClientId == clientKey).ToList();*

*for (int i = 0; i < settlementsForClient.Count; i++)*

*{*

*//Удаляем сведения о поселении клиента*

*RemoveSettlement(settlementsForClient[i]);*

*}*

*}*

*/// <summary>*

*/// Удалить номер по идентификатору*

*/// </summary>*

*/// <param name="roomKey"></param>*

*public void RemoveRoom(int roomKey)*

*{*

*\_rooms.Remove(roomKey);*

*//Генерируем событие о том, что номер удалён*

*RoomRemoved?.Invoke(roomKey, EventArgs.Empty);*

*//Получаем список сведений о поселении в номер*

*var settlementsForRoom = Settlements.Where(s => s.Room.Number == roomKey).ToList();*

*for (int i = 0; i < settlementsForRoom.Count; i++)*

*{*

*//Удаляем сведения о поселении в номер*

*RemoveSettlement(settlementsForRoom[i]);*

*}*

*}*

*/// <summary>*

*/// Удалить информацию о поселении*

*/// </summary>*

*/// <param name="settlement">Информация о поселении</param>*

*public void RemoveSettlement(Settlement settlement)*

*{*

*\_settlements.Remove(settlement);*

*//Генерируем событие о том, что информация о поселении удалена*

*SettlementRemoved?.Invoke(settlement, EventArgs.Empty);*

*}*

*Обратите внимание, что при реализации соответствующих методов генерируются исключений в случае некорректных данных с использованием описанных ранее классов исключений, а также реализован вызов соответствующих событий.*

*Дополнительно внесём изменения в классы диалоговых форм таким образом, чтобы эти формы можно было использовать повторно без пересоздания их экземпляров.*

*Класс формы добавления и редактирования клиента будет выглядеть следующим образом.*

*public partial class FormClient : Form*

*{*

*private Client \_client;*

*public Client Client*

*{*

*get { return \_client; }*

*set*

*{*

*\_client = value;*

*textBoxFirstName.Text = Client.FirstName;*

*textBoxMiddleName.Text = Client.MiddleName;*

*textBoxLastName.Text = Client.LastName;*

*textBoxIssuer.Text = Client.Passport.Issuer;*

*textBoxSeria.Text = Client.Passport.Seria;*

*textBoxNumber.Text = Client.Passport.Number;*

*dateTimePickerDate.Value = Client.Passport.Date;*

*}*

*}*

*public FormClient()*

*{*

*InitializeComponent();*

*}*

*private void buttonSave\_Click(object sender, EventArgs e)*

*{*

*\_client.FirstName = textBoxFirstName.Text;*

*\_client.MiddleName = textBoxMiddleName.Text;*

*\_client.LastName = textBoxLastName.Text;*

*\_client.Passport.Issuer = textBoxIssuer.Text;*

*\_client.Passport.Seria = textBoxSeria.Text;*

*\_client.Passport.Number = textBoxNumber.Text;*

*\_client.Passport.Date = dateTimePickerDate.Value;*

*}*

*}*

*Класс формы добавления и редактирования номера будет выглядеть следующим образом.*

*public partial class FormRoom : Form*

*{*

*private Room \_room;*

*public Room Room*

*{*

*get*

*{*

*return \_room;*

*}*

*set*

*{*

*\_room = value;*

*comboBoxCategory.SelectedItem = \_room.Category;*

*numericUpDownNumber.Value = \_room.Number;*

*numericUpDownPayByDay.Value = \_room.PayByDay;*

*numericUpDownPlaceCount.Value = \_room.PlaceCount;*

*}*

*}*

*public FormRoom()*

*{*

InitializeComponent();

comboBoxCategory.Items.Add(CategoryEnum.Econom);

comboBoxCategory.Items.Add(CategoryEnum.Standart);

comboBoxCategory.Items.Add(CategoryEnum.Comfort);

comboBoxCategory.Items.Add(CategoryEnum.Luxe);

}

private void buttonSave\_Click(object sender, EventArgs e)

{

\_room.Category = (CategoryEnum)comboBoxCategory.SelectedItem;

\_room.Number = (int)numericUpDownNumber.Value;

\_room.PayByDay = numericUpDownPayByDay.Value;

\_room.PlaceCount = (int)numericUpDownPlaceCount.Value;

}

}

В диалоговую форму редактирования и добавления информации о поселении необходимо добавить обработчики событий о добавлении или удалении клиентов и номеров. Эти обработчики необходимы для обеспечения актуальности выпадающих списков на этой форме. Сам класс будет выглядеть следующим образом.

public partial class FormSettlement : Form

{

private Settlement \_settlement;

public Settlement Settlement

{

get { return \_settlement; }

set

{

\_settlement = value;

comboBoxClient.SelectedItem = \_settlement.Client;

comboBoxRoom.SelectedItem = \_settlement.Room;

dateTimePickerStartDate.Value = \_settlement.StartDate;

dateTimePickerEndDate.Value = \_settlement.EndDate;

}

}

private readonly Hotel \_hotel = Hotel.Instance;

public FormSettlement()

{

InitializeComponent();

\_hotel.ClientAdded += \_hotel\_ClientAdded;

\_hotel.ClientRemoved += \_hotel\_ClientRemoved;

\_hotel.RoomAdded += \_hotel\_RoomAdded;

\_hotel.RoomRemoved += \_hotel\_RoomRemoved;

foreach (var client in \_hotel.Clients)

{

comboBoxClient.Items.Add(client);

}

foreach (var room in \_hotel.Rooms)

{

comboBoxRoom.Items.Add(room);

}

}

private void \_hotel\_RoomRemoved(object sender, EventArgs e)

{

int key = (int)sender;

for (int i = 0; i < comboBoxRoom.Items.Count; i++)

{

var room = comboBoxRoom.Items[i] as Room;

if (room?.Number == key)

{

comboBoxRoom.Items.RemoveAt(i);

break;

}

}

}

private void \_hotel\_RoomAdded(object sender, EventArgs e)

{

comboBoxRoom.Items.Add(sender);

}

private void \_hotel\_ClientRemoved(object sender, EventArgs e)

{

int key = (int)sender;

for (int i = 0; i < comboBoxClient.Items.Count; i++)

{

var client = comboBoxClient.Items[i] as Client;

if (client?.ClientId == key)

{

comboBoxClient.Items.RemoveAt(i);

break;

}

}

}

private void \_hotel\_ClientAdded(object sender, EventArgs e)

{

comboBoxClient.Items.Add(sender);

}

private void buttonSave\_Click(object sender, EventArgs e)

{

\_settlement.Client = comboBoxClient.SelectedItem as Client;

\_settlement.Room = comboBoxRoom.SelectedItem as Room;

\_settlement.StartDate = dateTimePickerStartDate.Value;

\_settlement.EndDate = dateTimePickerEndDate.Value;

}

}

В классе главной формы добавим и реализуем обработчики событий добавления и удаления объектов всех трех классов, а также избавимся от методов перестраивающих целиком списки элементов на вкладках. При реализации этих методов будем учитывать, что могут возникать исключительные ситуации и добавим их обработку в виде вывода информационных сообщений.

private readonly Hotel \_hotel = Hotel.Instance;

readonly FormRoom \_formRoom = new FormRoom();

readonly FormClient \_formClient = new FormClient();

readonly FormSettlement \_formSettlement = new FormSettlement();

public FormMain()

{

InitializeComponent();

\_hotel.ClientAdded += \_hotel\_ClientAdded;

\_hotel.RoomAdded += \_hotel\_RoomAdded;

\_hotel.SettlementAdded += \_hotel\_SettlementAdded;

\_hotel.ClientRemoved += \_hotel\_ClientRemoved;

\_hotel.RoomRemoved += \_hotel\_RoomRemoved;

\_hotel.SettlementRemoved += \_hotel\_SettlementRemoved;

}

private void \_hotel\_SettlementRemoved(object sender, EventArgs e)

{

var settlement = sender as Settlement;

for (int i = 0; i < listViewSettlement.Items.Count; i++)

{

if ((Settlement)listViewSettlement.Items[i].Tag == settlement)

{

listViewSettlement.Items.RemoveAt(i);

break;

}

}

}

private void \_hotel\_RoomRemoved(object sender, EventArgs e)

{

var roomNumber = (int)sender;

for (int i = 0; i < listViewRooms.Items.Count; i++)

{

if (((Room)listViewRooms.Items[i].Tag).Number == roomNumber)

{

listViewRooms.Items.RemoveAt(i);

break;

}

}

}

private void \_hotel\_ClientRemoved(object sender, EventArgs e)

{

var clientId = (int)sender;

for (int i = 0; i < listViewClients.Items.Count; i++)

{

if (((Client)listViewClients.Items[i].Tag).ClientId == clientId)

{

listViewClients.Items.RemoveAt(i);

break;

}

}

}

private void \_hotel\_SettlementAdded(object sender, EventArgs e)

{

var settlement = sender as Settlement;

if (settlement != null)

{

var listViewItem = new ListViewItem

{

Tag = settlement,

Text = settlement.Client.ToString()

};

listViewItem.SubItems.Add(settlement.Room.ToString());

listViewItem.SubItems.Add(settlement.StartDate.ToShortDateString());

listViewItem.SubItems.Add(settlement.EndDate.ToShortDateString());

listViewSettlement.Items.Add(listViewItem);

}

}

private void \_hotel\_RoomAdded(object sender, EventArgs e)

{

var room = sender as Room;

if (room != null)

{

var listViewItem = new ListViewItem

{

Tag = room,

Text = room.ToString()

};

listViewRooms.Items.Add(listViewItem);

}

}

private void \_hotel\_ClientAdded(object sender, EventArgs e)

{

var client = sender as Client;

if (client != null)

{

var listViewItem = new ListViewItem

{

Tag = client,

Text = client.ToString()

};

listViewClients.Items.Add(listViewItem);

}

}

private void addClientToolStripMenuItem\_Click(object sender, EventArgs e)

{

var client = new Client();

\_formClient.Client = client;

if (\_formClient.ShowDialog() == DialogResult.OK)

{

try

{

\_hotel.AddClient(client);

}

catch (Exception exception)

{

MessageBox.Show(exception.Message);

}

}

}

private void editClientToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

var client = listViewClients.SelectedItems[0].Tag as Client;

\_formClient.Client = client;

if (\_formClient.ShowDialog() == DialogResult.OK)

{

listViewClients.SelectedItems[0].Text = \_formClient.Client.ToString();

}

}

catch (Exception)

{

MessageBox.Show("Не выбрана строка с клиентом");

}

}

private void addRoomToolStripMenuItem\_Click(object sender, EventArgs e)

{

var room = new Room();

\_formRoom.Room = room;

if (\_formRoom.ShowDialog() == DialogResult.OK)

{

try

{

\_hotel.AddRoom(room);

}

catch (Exception exception)

{

MessageBox.Show(exception.Message);

}

}

}

private void editRoomToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

var room = listViewRooms.SelectedItems[0].Tag as Room;

\_formRoom.Room = room;

if (\_formRoom.ShowDialog() == DialogResult.OK)

{

listViewRooms.SelectedItems[0].Text = \_formRoom.Room.ToString();

}

}

catch (Exception)

{

MessageBox.Show("Не выбрана строка с номером");

}

}

private void addSettlementToolStripMenuItem\_Click(object sender, EventArgs e)

{

var settlement = new Settlement();

\_formSettlement.Settlement = settlement;

if (\_formSettlement.ShowDialog() == DialogResult.OK)

{

try

{

\_hotel.AddSettlement(settlement);

}

catch (Exception exception)

{

MessageBox.Show(exception.Message);

}

}

}

private void editSettlementToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

var settlement = listViewSettlement.SelectedItems[0].Tag as Settlement;

\_formSettlement.Settlement = settlement;

if (\_formSettlement.ShowDialog() == DialogResult.OK)

{

settlement = \_formSettlement.Settlement;

var listViewItem = listViewSettlement.SelectedItems[0];

listViewItem.Text = settlement.Client.ToString();

listViewItem.SubItems[1].Text = settlement.Room.ToString();

listViewItem.SubItems[2].Text = settlement.StartDate.ToShortDateString();

listViewItem.SubItems[3].Text = settlement.EndDate.ToShortDateString();

}

}

catch (Exception)

{

MessageBox.Show("Не выбрана строка с поселением");

}

}

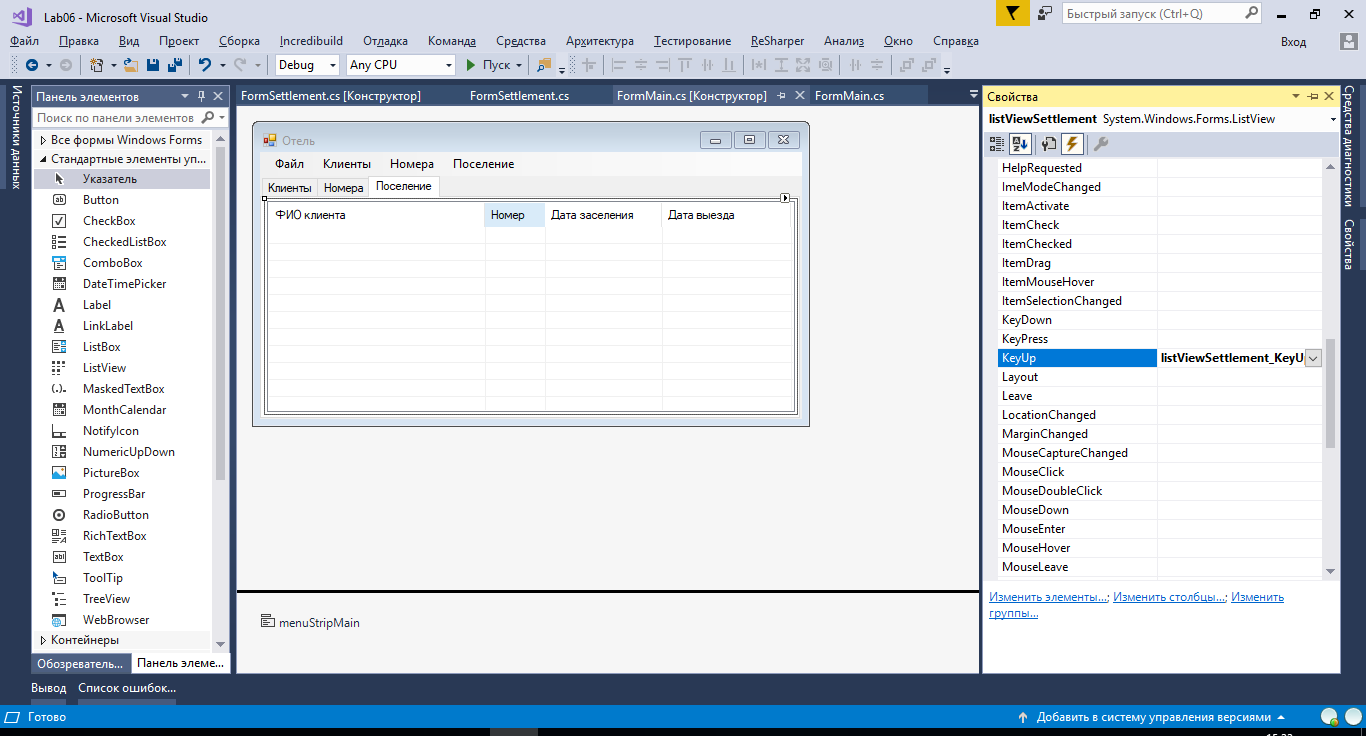
private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

Application.Exit();

}

В заключении добавим (рис. 1) и реализуем обработку события нажатия на клавишу «Delete» (KeyUp) для списочного элемента управления на каждой вкладке.



***Рис. 1. Добавление обработчика нажатия на клавишу***

private void listViewClients\_KeyUp(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Delete)

{

try

{

var client = listViewClients.SelectedItems[0].Tag as Client;

if (client != null)

{

\_hotel.RemoveClient(client.ClientId);

}

}

catch (Exception)

{

MessageBox.Show("Не выбрана строка с клиентом");

}

}

}

private void listViewRooms\_KeyUp(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Delete)

{

try

{

var room = listViewRooms.SelectedItems[0].Tag as Room;

if (room != null)

{

\_hotel.RemoveRoom(room.Number);

}

}

catch (Exception)

{

MessageBox.Show("Не выбрана строка с номером");

}

}

}

private void listViewSettlement\_KeyUp(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Delete)

{

try

{

var settlement = listViewSettlement.SelectedItems[0].Tag as Settlement;

if (settlement != null)

{

\_hotel.RemoveSettlement(settlement);

}

}

catch (Exception)

{

MessageBox.Show("Не выбрана строка с поселением");

}

}

}

**Задание на лабораторную работу**

1. Доработайте разработанный в предыдущей лабораторной работе класс, содержащий коллекции следующим образом/
   * Свойства с коллекциями сделайте не статическими, возвращающими интерфейсы IEnumerable<T> для всех 3-х классов. Для хранения коллекций используйте приватные поля данных соответствующих типов (словари для первых двух классов, список – для третьего).
   * Создайте статическое свойство для получения единственного экземпляра класса. Для этих целей реализуйте приватный конструктор класса.
   * Добавьте публичные методы для добавления и удаления объектов коллекций.
   * Добавьте описание событий добавления и удаления объектов коллекций.
   * Опишите классы исключений для каждого вида объектов, хранящихся в коллекциях.
   * При реализации методов добавления и удаления объектов коллекции используйте генерацию соответствующих исключений и вызов событий.
2. Внесите изменения в диалоговые и главную формы, позволяющие использовать единственный экземпляр каждой диалоговой формы при добавлении и редактировании соответствующей сущности. Для этого добавьте соответствующие приватные поля, переделайте публичные свойства и реализуйте только конструкторы диалоговых окон без параметров.
3. В третьем классе в конструкторе добавьте подключение к событиям добавления и удаления объектов первых двух классов. Реализуйте соответствующие обработчики событий.
4. В классе главной формы добавьте и реализуйте обработчики событий добавления и удаления объектов всех трех классов.
5. Избавьтесь в классе главной формы от методов перестраивающих целиком списки элементов на вкладках.
6. Добавьте и реализуйте обработку события нажатия на клавишу «Delete» (KeyUp) для списочного элемента управления на каждой вкладке.
7. Добавьте блоки обработки исключительных ситуаций в методах класса главной формы, которые потенциально могут генерировать исключения.

**Контрольные вопросы**

1. Что такое делегаты и как они объявляются?
2. Каким образом объявляются и используются анонимные методы?
3. Что такое событие и как оно генерируется?
4. Каким образом осуществляется объявление и привязка события?
5. Каким образом осуществляется обработка и генерация исключений в среде Microsoft .Net?

# Список рекомендуемой литературы

1. Нейгел, К. C# 5.0 и платформа .NET 4.5 для профессионалов / К.Нейгел, Б.Ивьен, Дж.Глинн, М.Скиннер, К.Уотсон. – М.: Диалектика, 2013. – 1440 с.
2. Неш, Т. C# 2010: ускоренный курс для профессионалов / Т.Неш – М.: Вильямс, 2010. – 592 с.
3. Снелл, М. Microsoft Visual Studio 2008 / М.Снелл, Л.Пауэрс. – СПб: БХВ-Петербург, 2009. – 1200 с.
4. Троелсен, Э. Язык программирования С# 5.0 и платформа .NET 4.5 / Э.Троелсен. – 6-е изд. – М.: Вильямс, 2015. – 1312 с.
5. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж.Рихтер. – СПб.: Питер, 2016. – 896 с.