Справочный материал:

using System.Runtime.InteropServices;

namespace System

{

//

// Сводка:

// Определяет обобщенный метод сравнения для типа, который реализуется типом или

// классом значения, чтобы упорядочить или сортировать его экземпляры.

[ComVisible(true)]

public interface IComparable

{

//

// Сводка:

// Сравнивает текущий экземпляр с другим объектом того же типа и возвращает целое

// число, которое показывает, расположен ли текущий экземпляр перед, после или на

// той же позиции в порядке сортировки, что и другой объект.

//

// Параметры:

// obj:

// Объект для сравнения с данным экземпляром.

//

// Возврат:

// Значение, указывающее, каков относительный порядок сравниваемых объектов.Возвращаемые

// значения представляют следующие результаты сравнения.Значение Описание Меньше

// нуля Этот экземпляр меньше параметра obj. Нуль Этот экземпляр и параметр obj

// равны. Больше нуля Этот экземпляр больше параметра obj.

//

// Исключения:

// T:System.ArgumentException:

// Тип значения параметра obj отличается от типа данного экземпляра.

int CompareTo(object obj);

}

}

**Вторая часть (описываются поля и конструкторы для производных классов от головного).**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace WindowsFormsApplication1

{

public class Train : IComparable // класс Поезда

{

private string num; // номер

private int year; // год выпуска

// доступ к полям

public string Num { get { return num; } set { Num = num; } }

public int Year { get { return year; } set { Year = year; } }

public Train(string \_num, int \_year) // конструктор с параметрами

{

num = \_num;

year = \_year;

}

public int CompareTo(object obj) // для сортировки по номеру

{

Train o = obj as Train;

return num.CompareTo(o.num); // сравнение по номеру

}

}

// класс PassTrain (Пассажирские поезда) наследуется от класса Train

public class PassTrain : Train

{

private int kolm; // количество мест

private string color; // цвет

// доступ к полям

public int Kolm { get { return kolm; } set { Kolm = kolm; } }

public string Color { get { return color; } set { Color = color; } }

// конструктор с параметрами

public PassTrain(string \_num, int \_year, int \_kolm, string \_color)

: base(\_num, \_year)

{

kolm = \_kolm; // количество мест

color = \_color; // цвет

}

}

// класс FrTrain (Грузовые поезда) наследуется от класса Train

public class FrTrain : Train

{

private int cap; // вместимость

private string type; // тип груза

// доступ к полям

public int Cap { get { return cap; } set { Cap = cap; } }

public string Type { get { return type; } set { Type = type; } }

// конструктор с параметрами

public FrTrain(string \_num, int \_year, int \_cap, string \_type)

: base(\_num, \_year)

{

cap = \_cap;

type = \_type;

}

}

}

**Form 1 (некоторые выкладки на которые следует обратить внимание) :**

static public List<Train> trains = new List<Train>(); // инициализация списка поездов – trains

**Для загрузки изначально данных, нужно использовать метод: Load**

private void Form1\_Load(object sender, EventArgs e)

{

// положение формы1 на экране

// начальный набор данных

trains.Add(new PassTrain("004", 1990, 300, "синий"));

trains.Add(new PassTrain("010", 2015, 1000, "красный"));

trains.Add(new PassTrain("100", 2010, 200, "зеленый"));

trains.Add(new FrTrain("961", 1980, 2000, "опасный"));

**Обработка данных по нажатии кнопки “добавить”:**

private void button1\_Click(object sender, EventArgs e) // по кнопке "Добавить"

{

// проверка заполнения всех полей с данными

foreach (Control c in groupBox1.Controls)

if (c is TextBox)

if (((TextBox)c).Text == "")

{

MessageBox.Show("Не заполнены данные, добавление невозможно!");

return;

}

// проверяем уникальность номера - ищем введенный номер в списке

// Внимание!!!!! Использую тут лямбда –оператор. Ссылка про синтаксис и //использование его будет отдельным документом.

if (trains.Exists(x => x.Num == Num.Text))

{

MessageBox.Show("Такой номер уже существует");

return;

}

// получаем введенные данные

string num = Num.Text; // номер поезда

int year = int.Parse(IntYear.Text); // год выпуска

int pol3 = int.Parse(InttextBox1.Text); // количество мест (вместимость)

string pol4 = textBox2.Text; // цвет (тип груза)

// добавление данных

if (radioButton1.Checked == true) // пассажирские

{

// создаем новый экземпляр класса PassTrain и добавляем его в список

trains.Add(new PassTrain(num, year, pol3, pol4));

}

else if (radioButton2.Checked == true) // грузовые

{

// создаем новый экземпляр класса FrTrain и добавляем его в список

trains.Add(new FrTrain(num, year, pol3, pol4));

}

// сортировка списка по номеру поезда

trains.Sort();

}

**Что касается Валидации, и как она понимается. Это невозможность выхода, или появления предупреждения о том, что нужно покинуть данное поле. Информация об этом.**

private void tbIntValidating(object sender, CancelEventArgs e)

{

var tb = sender as TextBox;

int dummy;

// если не удается преобразовать в int, то не можем выйти из поля ввода

e.Cancel = !int.TryParse(tb.Text, out dummy);

}

Открытие модального окна с гридом, в котором отображаются введенные поезда, точнее классы

Поездов: пассажирские и грузовые:

private void button3\_Click(object sender, EventArgs e)

{

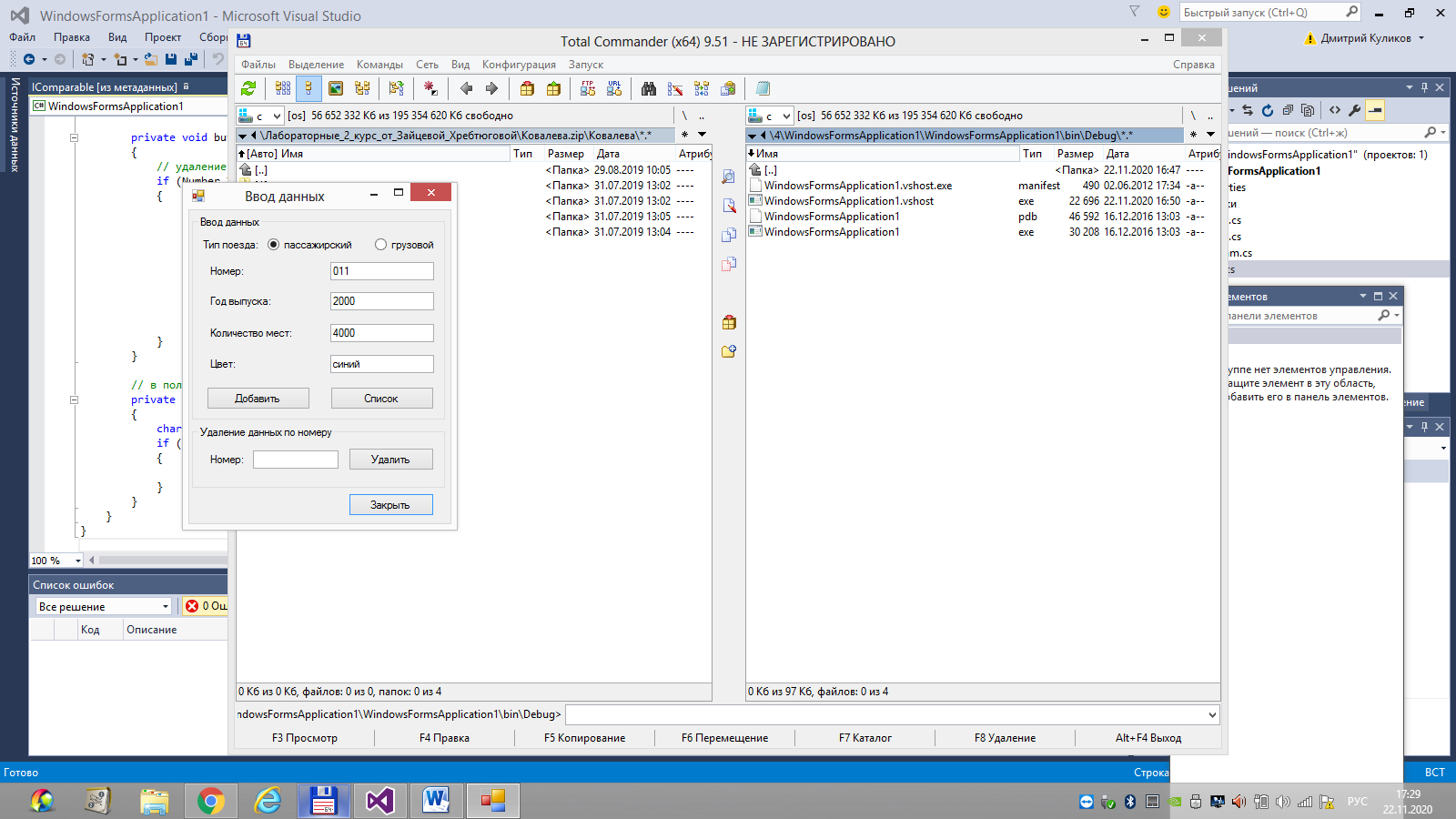
Form NewF = new Form3();

NewF.ShowDialog();

}

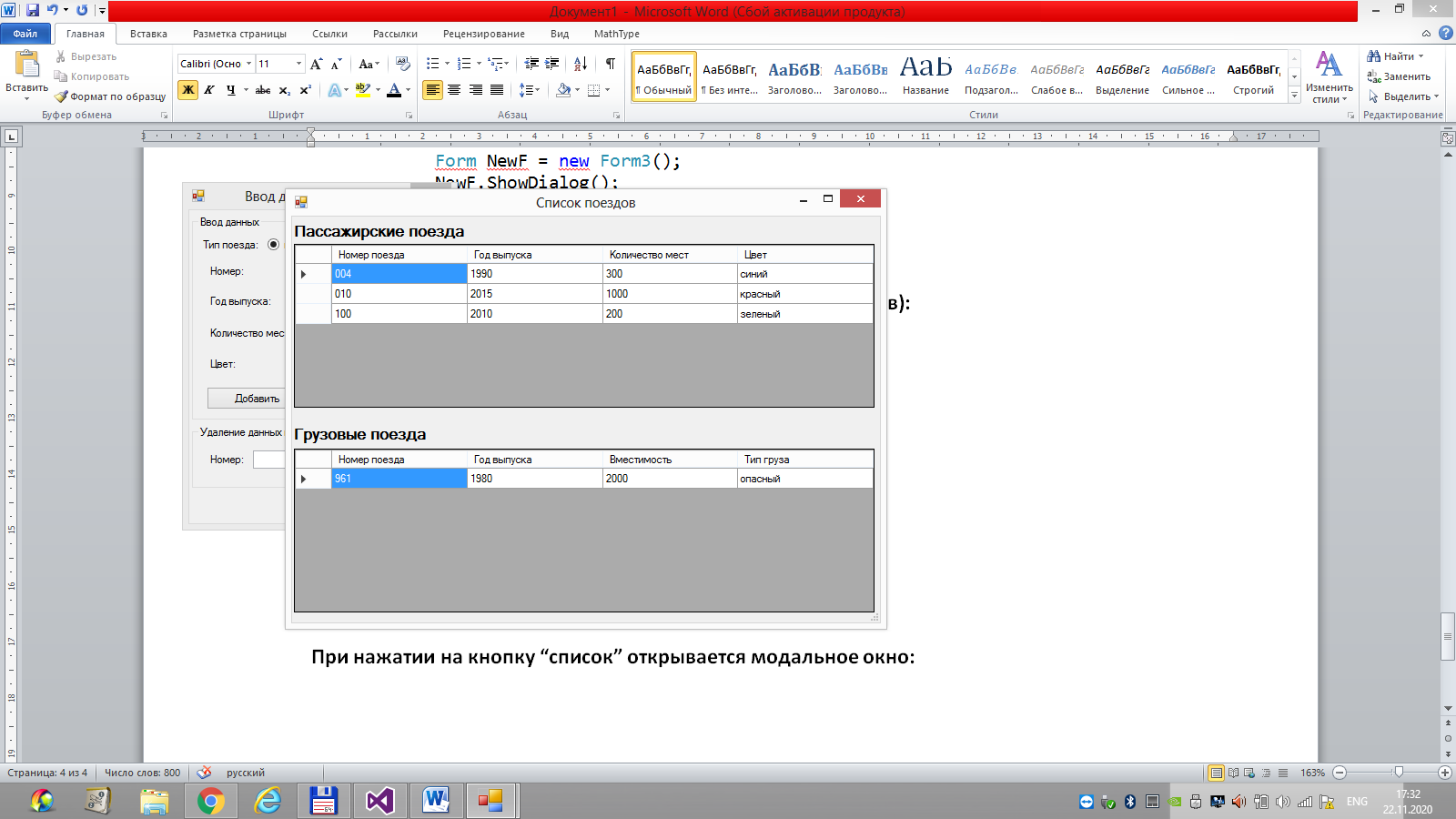
**Иллюстрация и пример как должно все работать:**

1. **Открывается форма (на примере работы двух классов):**



**Рис. 1**

**При нажатии на кнопку “список” открывается модальное окно:**

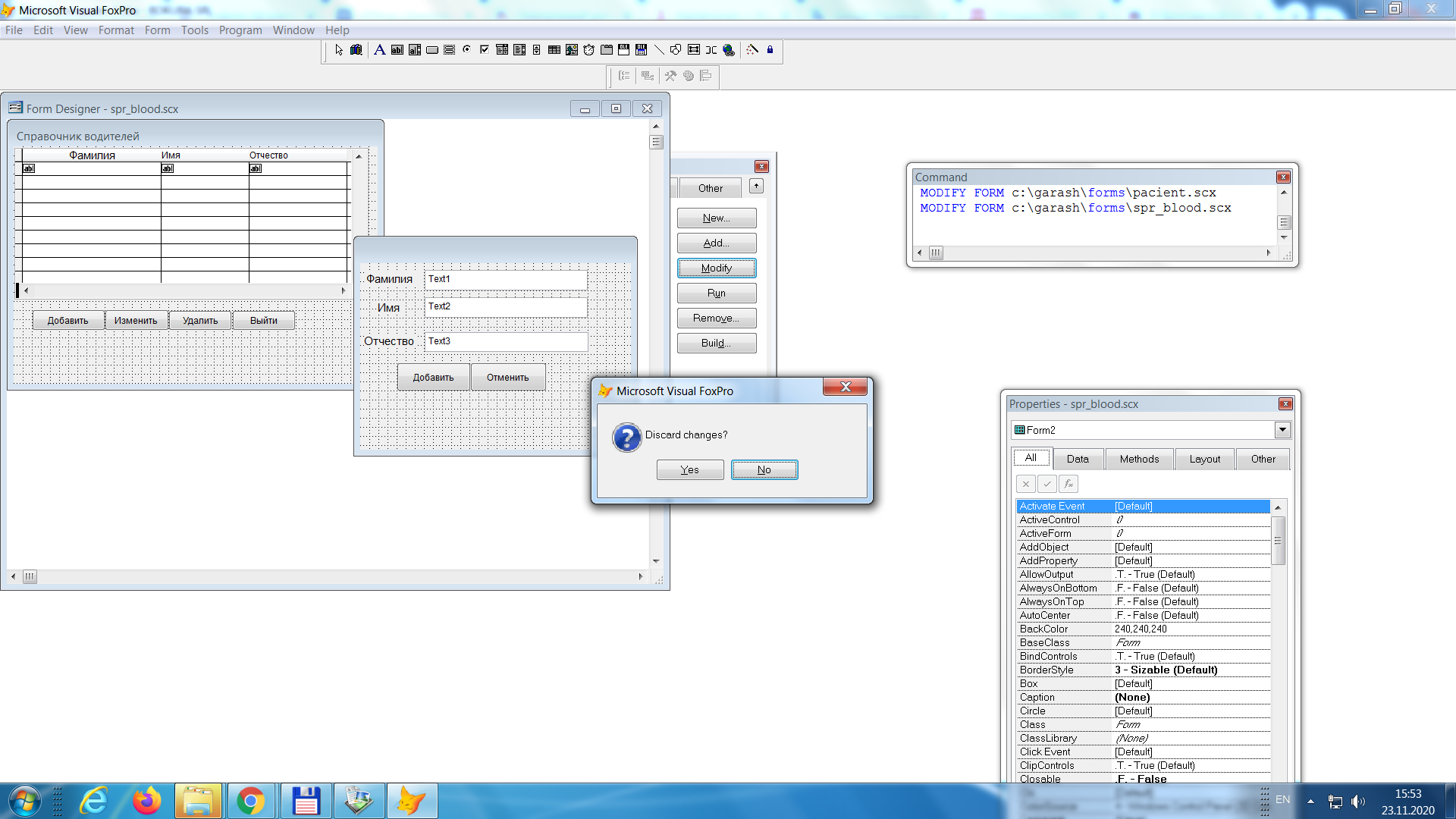


**Рис. 2**

**Советы по дизайну:**

**Есть головная форма, на ней 2 грида (если есть два дочерних класса, конечно, если больше то больше гридов). Внизу кнопки: добавить, изменить, удалить и поиск, при нажатии которых отображается модальное окно с вводом данных (тип (вид окна ) – см. выше – Рис. 1).**

**Т.е. в качестве примера советую и даже рекомендую, сделать следующее (т.е. такого сорта):**



**Информация про лямбда – выражения:**

**Лямбда-оператор**

В [лямбда-выражениях](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/lambda-expressions) лямбда-оператор => используется для отделения входных параметров с левой стороны от тела лямбда-выражения с правой стороны.

В следующих примерах используется функция [LINQ](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/linq/) с синтаксисом метода для демонстрации применения лямбда-выражений:

string[] words = { "bot", "apple", "apricot" };

int minimalLength = words

.Where(w => w.StartsWith("a"))

.Min(w => w.Length);

Console.WriteLine(minimalLength); // output: 5

int[] numbers = { 4, 7, 10 };

int product = numbers.Aggregate(1, (interim, next) => interim \* next);

Console.WriteLine(product); // output: 280

**Входные параметры лямбда-выражений строго типизируются во время компиляции. Если компилятор может выводить типы входных параметров, как в предыдущем примере, вы можете опустить объявления типа. Если требуется указать тип входных параметров, это необходимо делать для каждого такого параметра, как показано в следующем примере:**

int[] numbers = { 4, 7, 10 };

int product = numbers.Aggregate(1, (int interim, int next) => interim \* next);

Console.WriteLine(product); // output: 280

**В следующем примере показано, как определить лямбда-выражение без входных параметров:**

Func<string> greet = () => "Hello, World!";

Console.WriteLine(greet());

**Определение тела выражения имеет следующий общий синтаксис:**

member => expression;

где expression — любое допустимое выражение. Тип возвращаемого значения expression должен быть неявно преобразуемым в тип возвращаемого значения элемента. Если для элемента возвращается значение типа void или элемент является конструктором, методом завершения или методом доступа свойства или индексатора set, значение expression должно быть [*выражением оператора*](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/language-specification/statements#expression-statements). Поскольку результат выражения отбрасывается, возвращаемым типом этого выражения может быть любой тип.

**В следующем примере приводится определение тела выражения для метода Person.ToString:**

public override string ToString() => $"{fname} {lname}".Trim();

**Это сокращенная версия следующего определения метода:**

public override string ToString()

{

return $"{fname} {lname}".Trim();

}

**Задания**

**1.** описать 2 класса, которые настедуются от третьего.

- классы: кошки (вес, кличка, возраст, цвет, порода) и птицы (вес, вид, размах крыльев, цвет)

- реализовать добавление животных на одной форме (поля должны скрываться и показываться в зависимости от того, кого добавляем) с валидацией полей и проверкой на уникальность клички кошки и вида птицы

- справа от формы добавления отобразить список добавленных животных (сгруппированный по классу и отсортированный для кошек по кличке, а для птиц - по виду) либо отобразить как на рис. 3, что рекомендуется, так нагляднее. Т.е. на Ваш выбор, какая форма будет модальной – мне все равно.

- сделать начальный набор данных. Сохранять введенные данные не нужно.

- поиск по породе для кошек, поиск по виду для птиц

**2.** описать 2 класса, которые настедуются от третьего.

- классы: пассажирский поезд (номер, количество мест, год выпуска, цвет) и грузовой поезд (номер, вместимость, год выпуска, тип груза)

- реализовать добавление поездов на одной форме (поля должны скрываться и показываться в зависимости от того, что добавляем) с валидацией полей и проверкой на уникальность номера

- справа от формы добавления отобразить список добавленных поездов, отсортированный по номеру либо отобразить как на рис. 3, что рекомендуется, так нагляднее. Т.е. на Ваш выбор, какая форма будет модальной – мне все равно.

- реализовать удаление поезда по его номеру

- сделать начальный набор данных. Сохранять введенные данные не нужно.

- поиск по номеру поиска как грузового, так и пассажирского поездов.

**3.** описать 2 класса, которые наследуются от третьего.

- классы: картина (название, автор, цена, год создания, материал) и книга (название, автор, цена, год создания, кол-во страниц, тираж)

- реализовать добавление произведений на одной форме (поля должны скрываться и показываться в зависимости от того, что добавляем) с валидацией полей и проверкой на уникальность по совокупности свойств (название, автор, картина или книга)

- справа от формы добавления отобразить список добавленных произведений (отсортированный по цене) либо отобразить как на рис. 3, что рекомендуется, так нагляднее. Т.е. на Ваш выбор, какая форма будет модальной – мне все равно.

- реализовать поиск произведений по автору и названию (ввод поискового выражения в одном поле и поиск по вхождению заданной подстроки в название или имя автора). В результате вывести список строк с информацией (1 строка = 1 произведение).

- сделать начальный набор данных. Сохранять введенные данные не нужно.

**3.** описать 2 класса, которые наследуются от третьего.

- классы: футбольный клуб (название,страна (из какой страны), город, год создания, название стадиона на котором играет) и футболист (фио, возраст, амплуа, цена за трансфер)

- реализовать добавление футбольных клубов на одной форме (поля должны скрываться и показываться в зависимости от того, что добавляем) с валидацией полей и проверкой на уникальность по совокупности свойств (название,страна (из какой страны), город, год создания, название стадиона на котором играет)

- справа от формы добавления отобразить список добавленных произведений (отсортированный по цене) либо отобразить как на рис. 3, что рекомендуется, так нагляднее. Т.е. на Ваш выбор, какая форма будет модальной – мне все равно.

- реализовать поиск клубов по названию (ввод поискового выражения в одном поле и поиск по вхождению заданной подстроки в название). В результате вывести список строк с информацией (1 строка = 1 произведение).

- реализовать поиск по фио футболиста.

- сделать начальный набор данных. Сохранять введенные данные не нужно.