УСЛОВИЯ ЗАДАЧ:

1. Создать класс Item со свойствами Name, ID, Price. Создать класс Manager с событием sale (распродажа). Создать вещи и добавить их в обобщенную коллекцию типа очередь. Подпишите некоторые вещи на событие sale
   Реакция на событие следующая, цена Item уменьшается на 70%. Продемонстрировать ситуацию события и вывести содержимое очереди на консоль

2. Реализовать обобщённый класс вектор. Вложить в него обобщённую коллекцию .NET. Наследовать интерфейс IAction с методами добавления, удаления, вывода и очистки. Методы реализовать в классе.
   Добавить обработку исключений с finally.
   Провести проверку с целочисленным типов и с типом студент (параметры определите сами).

3. Создать класс User с закрытыми полями login, password. переопределить в классе все Public методы object. перегрузить метод CompareTo стандартного унаследованного интерфейса IComparable который сравнивает пользователей по логину и паролю. создать и сравнить 3-х юзеров, создать LinkedList<user> с 5-ю юзерами. используя LINQ найти в коллекции юзеров, у которых длина пароля меньше 8 и содержит только цифры.

4. Создать абстрактный класс Transport (состав произвольно). Создайте интерфейс IAir с методами Fly и Check.Наследуйте их в классе Air, который содержит свойства Speed(скорость)
   CountOfPass (число пассажиров) и Status. Status принимает одно из значений перечисления fly, ready, error. При вызове метода Fly проверяется скорость, если она <220, генерируйте исключение.
   и выставляете статус error самолету. Check или ready (в пределах допустимого) или error (если их слишком мало или много). Создайте самолет и протестируйте его.

5. Есть класс Card, с приватными полями balance, класс ExDate, содержащий свойства Month, Year (два последние числа года), number. Реализован интерфейс IPay с методом Pay (int, ExDate). Привести явное преобразование интерфейса в классе Card. При условии, если баланс меньше - 100 вызвать исключение. Реализовать работу с Card. Создать массив значений баланса. Используя LINQ найти карту с максимальным балансом, вывести ее номер карты.

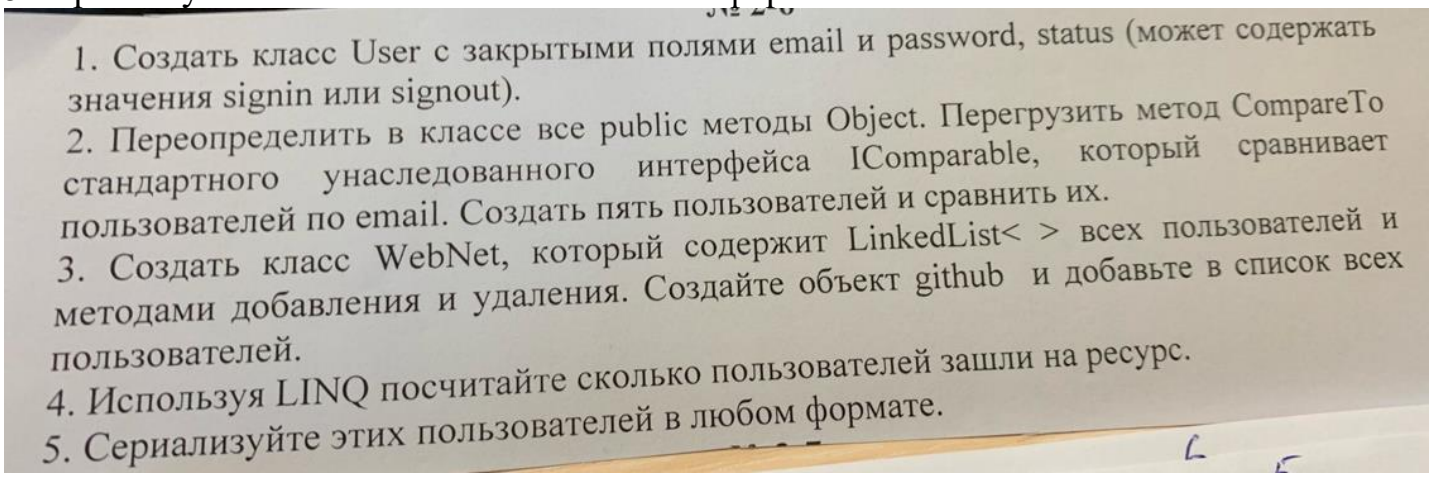
6. Создать производные классы: комплексные числа Mycomplex, вектор MyVect. определить функцию Norma для комплексных чисел -модуль в квадрате, для вектора - корень квадратный из суммы элементов.

7. 1-2. Сделать абстрактный класс Transport с вашей реализацией (я просто сделал string name). Наследовать его в классе Air. Добавить свойства CountOfPassengers и Speed, а

так же Status, который принимает значение из перечисления в классе с состояниями fly, ready, stop. Сделать интерфейс IAirable с методами Check() и Fly() и Наследовать его в Air. Метод Check(): Если CountOfPassengers = 0 и Speed = 0, то Status = stop; Если CountOfPassengers > 0 и Speed = 0, то Status = ready; Если CountOfPassengers > 0, Speed > 0 и Status = ready, то Status = fly. Метод Fly() выводит Flying, если Status = fly, если нет - выбрасывает исключение (можешь хоть базовое, я с сообщением делал). Продемонстрировать работу с объектом Air.
3. Сделать ВЕСЬ вывод еще и в файл.
4. Сделать интерфейс IAir... с таким же методом Check() и наследовать в Air. Метод из IAir... Должен выводить "Ready", если CountOfPassengers > 20 и <100. Продемонстрировать оба метода в программе. (2 интерфейса, 2 метода но с 1 названием (IAir...Check(), IAirable.Check())).
5. Создать коллекцию из Air и добавить 5 объектов. С помощью linq запросов вывести количество самолетов, находящихся в Status = fly, а так же посчитать среднюю их скорость.

8. 2-6 User

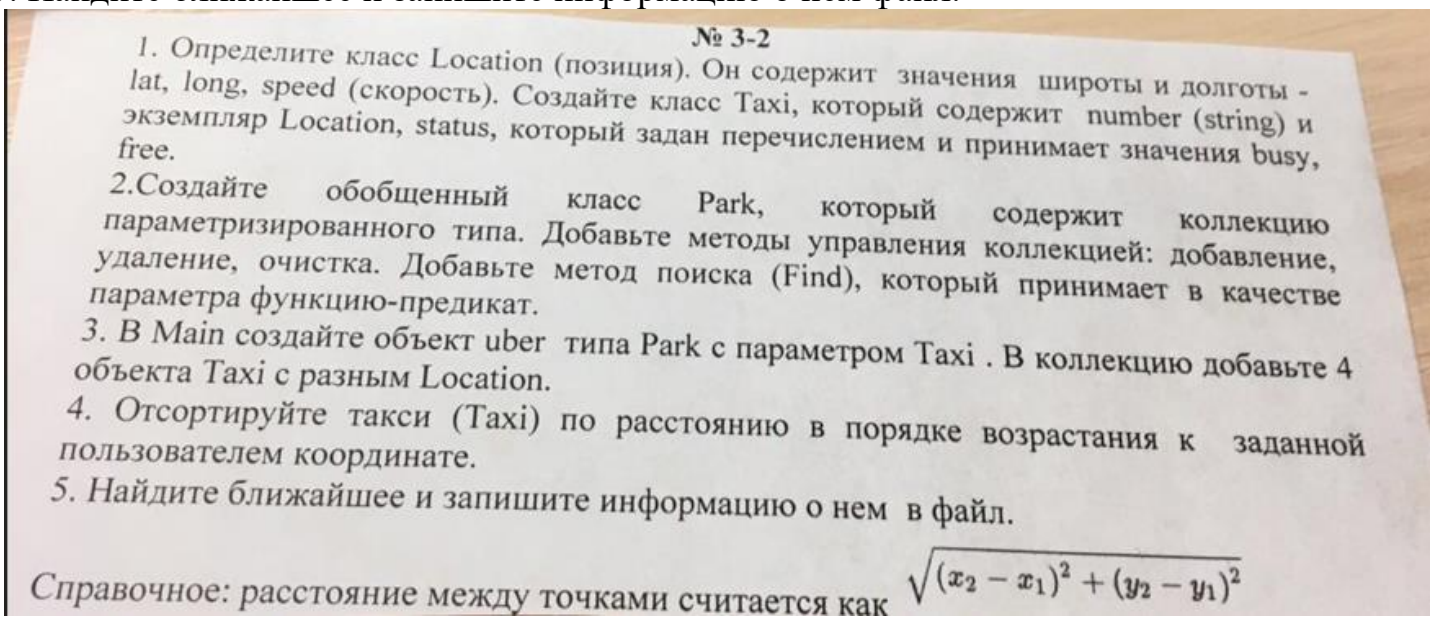
1.Создать класс User с закрытыми полями email и password, status(может содержать значения singin и singout)
2.Переопределить в классе все public методы Object. Перегрузить метод CompareTo стандартного унаследованного интерфейса IComparable, который сравнивает пользователей по email. Создать 5 пользователей и сравнить их.
3.Создать класс WebNet, который содержит LinkedList<> всех пользователей и методами добавления и удаления. Создать объект github и добавить в список всех пользователей.
4. Используя LINQ посчитайте сколько пользователей зашли на ресурс.
5.Сериализуёте этих пользователей в любом формате.


1. Создать класс User с закрытыми полями email и password, status (может содержать значения signin или signout).
2. Переопределить в классе все public методы Object. Перегрузить метод CompareTo стандартного унаследованного интерфейса IComparable, который сравнивает пользователей по email. Создать пять пользователей и сравнить их.
3. Создать класс WebNet, который содержит LinkedList< > всех пользователей и методами добавления и удаления. Создайте объект github и добавьте в список всех пользователей.
4. Используя LINQ посчитайте сколько пользователей зашли на ресурс.
5. Сериализуйте этих пользователей в любом формате.

9. 3-2 Location

1. Определите класс Location(позиция). Он содержит значения широты и долготы - lat, long, speed(скорость). Создайте класс Taxi, который содержит number(string) и экземпляр Lacation, status, который задан перечислением и принимает значения busy. free.
2. Создайте обобщённый класс Park, который содержит коллекцию параметризованного типа. Добавьте методы управления коллекцией: добавление, удаление, очистка. Добавьте метод поиска (Find), который принимает в качестве параметра функцию-предикат.

3. В Main создайте объект uber типа Park с параметром Taxi. В коллекцию добавьте 4 объекта Taxi с разными Location.
4. Отсортируйте такси (Taxi) по расстоянию в порядке возрастания к заданной пользователем координате.
5. Найдите ближайшее и запишите информацию о нём файл.

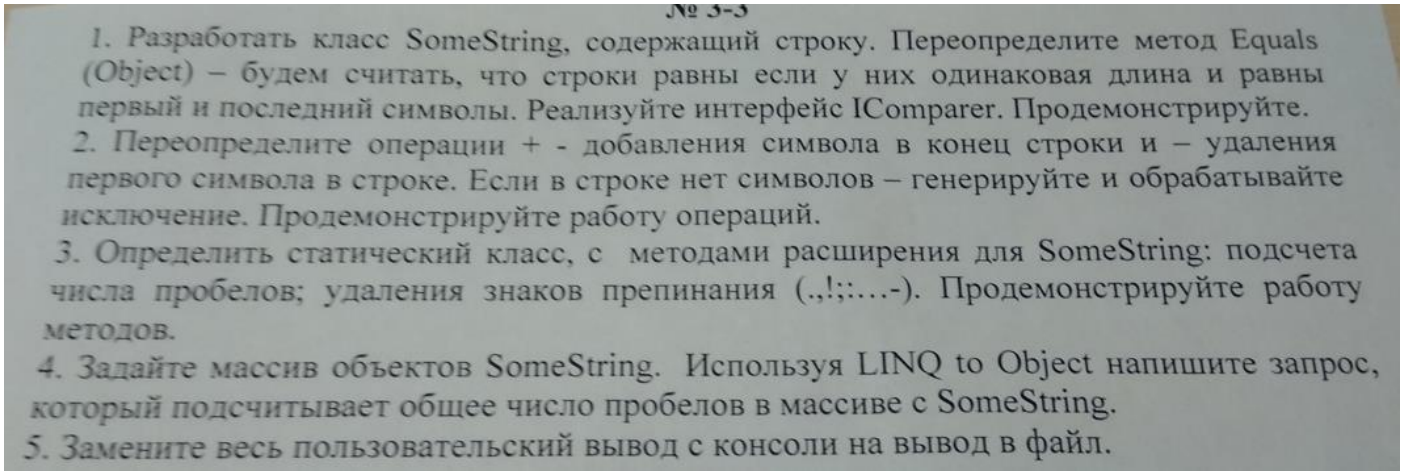


№ 3-2

1. Определите класс Location (позиция). Он содержит значения широты и долготы - lat, long, speed (скорость). Создайте класс Taxi, который содержит number (string) и экземпляр Location, status, который задан перечислением и принимает значения busy, free.
2. Создайте обобщенный класс Park, который содержит коллекцию параметризированного типа. Добавьте методы управления коллекцией: добавление, удаление, очистка. Добавьте метод поиска (Find), который принимает в качестве параметра функцию-предикат.
3. В Main создайте объект uber типа Park с параметром Taxi . В коллекцию добавьте 4 объекта Taxi с разным Location.
4. Отсортируйте такси (Taxi) по расстоянию в порядке возрастания к заданной пользователем координате.
5. Найдите ближайшее и запишите информацию о нем в файл.

Справочное: расстояние между точками считается как $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

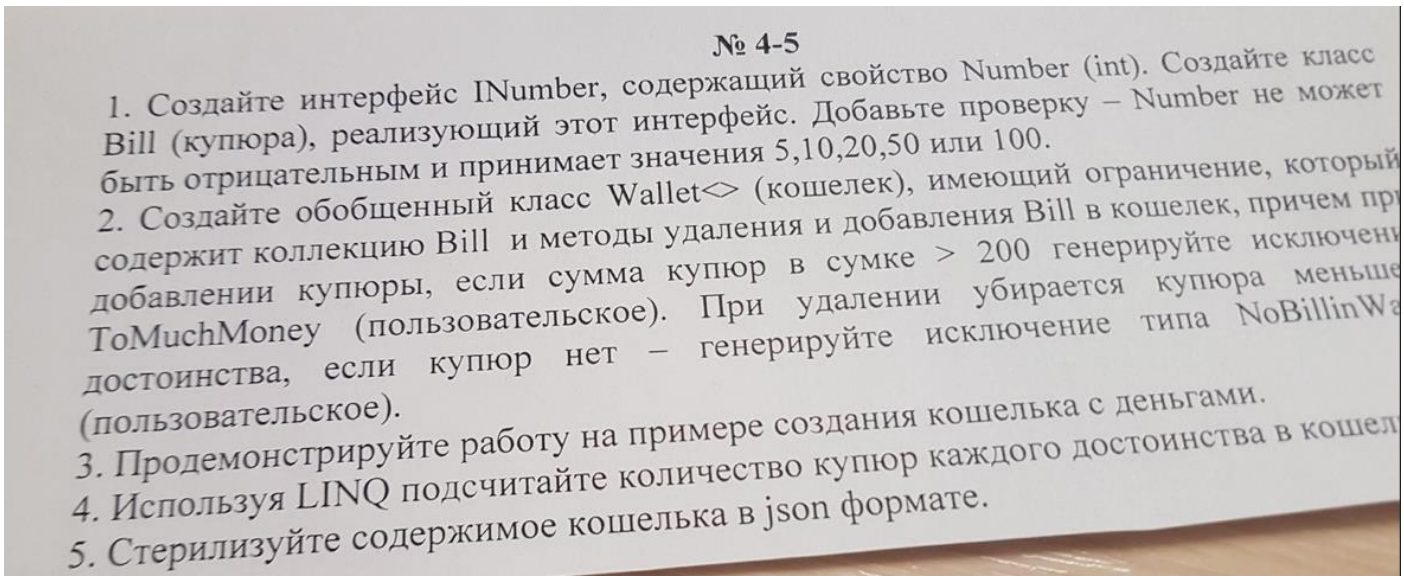
10.



№ 3-3

1. Разработать класс SomeString, содержащий строку. Переопределите метод Equals (Object) – будем считать, что строки равны если у них одинаковая длина и равны первый и последний символы. Реализуйте интерфейс IComparer. Продемонстрируйте.
2. Переопределите операции + - добавления символа в конец строки и – удаления первого символа в строке. Если в строке нет символов – генерируйте и обрабатывайте исключение. Продемонстрируйте работу операций.
3. Определить статический класс, с методами расширения для SomeString: подсчета числа пробелов; удаления знаков препинания (.,!;:…-). Продемонстрируйте работу методов.
4. Задайте массив объектов SomeString. Используя LINQ to Object напишите запрос, который подсчитывает общее число пробелов в массиве с SomeString.
5. Замените весь пользовательский вывод с консоли на вывод в файл.

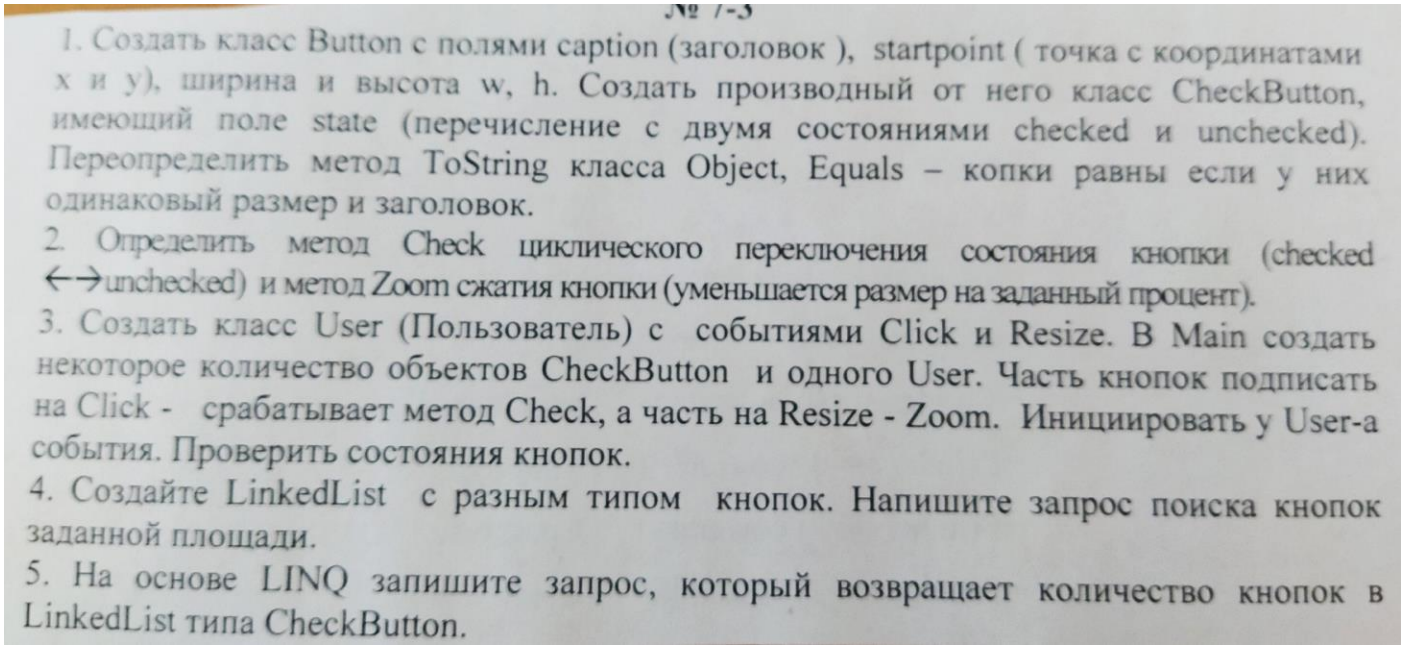
11.



№ 4-5

1. Создайте интерфейс INumber, содержащий свойство Number (int). Создайте класс Bill (купюра), реализующий этот интерфейс. Добавьте проверку – Number не может быть отрицательным и принимает значения 5,10,20,50 или 100.
2. Создайте обобщенный класс Wallet<> (кошелек), имеющий ограничение, который содержит коллекцию Bill и методы удаления и добавления Bill в кошелек, причем пр добавлении купюры, если сумма купюр в сумке > 200 генерируйте исключени ToMuchMoney (пользовательское). При удалении убирается купюра меньше достоинства, если купюр нет – генерируйте исключение типа NoBillinWa (пользовательское).
3. Продемонстрируйте работу на примере создания кошелька с деньгами.
4. Используя LINQ подсчитайте количество купюр каждого достоинства в кошел
5. Стерилизуйте содержимое кошелька в json формате.

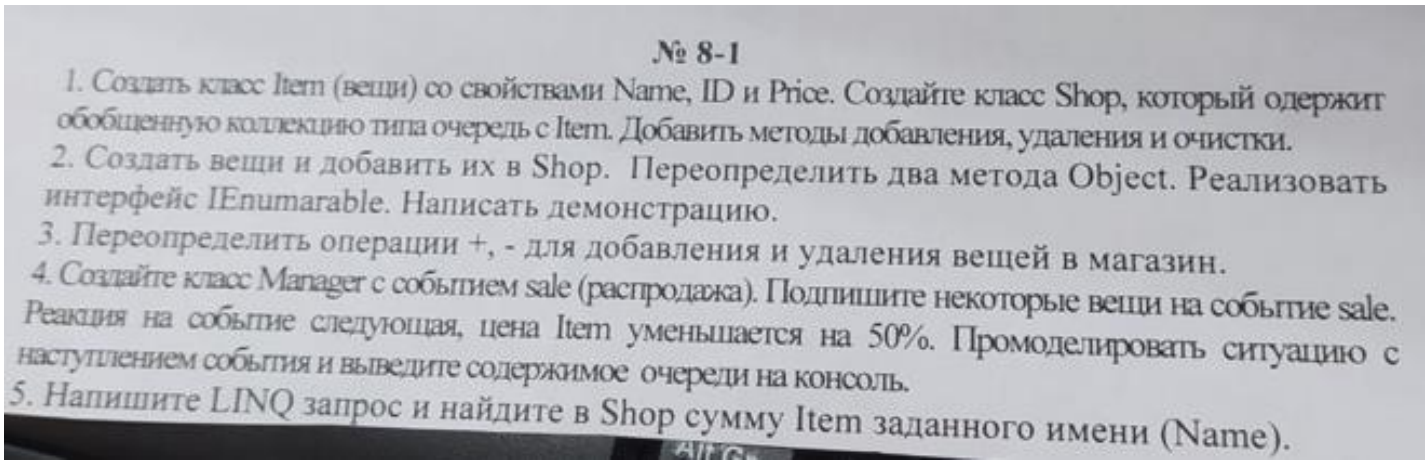
12.



№ 7-3

1. Создать класс Button с полями caption (заголовок), startpoint (точка с координатами x и y), ширина и высота w, h. Создать производный от него класс CheckButton, имеющий поле state (перечисление с двумя состояниями checked и unchecked). Переопределить метод ToString класса Object, Equals – копки равны если у них одинаковый размер и заголовок.
2. Определить метод Check циклического переключения состояния кнопки (checked ←→unchecked) и метод Zoom сжатия кнопки (уменьшается размер на заданный процент).
3. Создать класс User (Пользователь) с событиями Click и Resize. В Main создать некоторое количество объектов CheckButton и одного User. Часть кнопок подписать на Click - срабатывает метод Check, а часть на Resize - Zoom. Инициировать у User-а события. Проверить состояния кнопок.
4. Создайте LinkedList с разным типом кнопок. Напишите запрос поиска кнопок заданной площади.
5. На основе LINQ запишите запрос, который возвращает количество кнопок в LinkedList типа CheckButton.

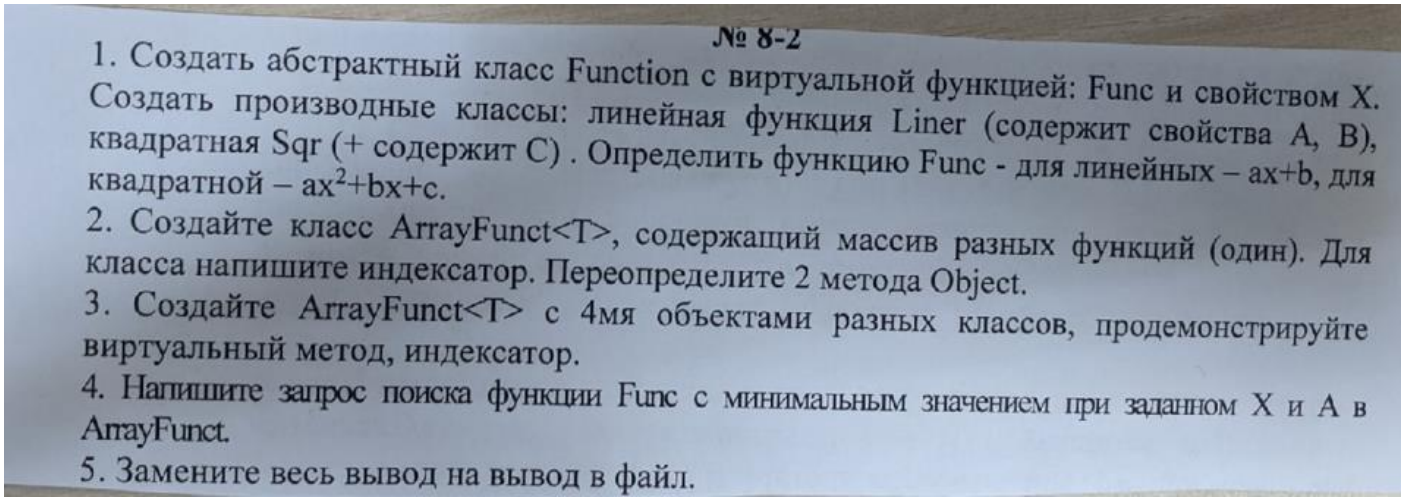
13.



№ 8-1

1. Создать класс Item (вещи) со свойствами Name, ID и Price. Создайте класс Shop, который одержит обобщенную коллекцию типа очередь с Item. Добавить методы добавления, удаления и очистки.
2. Создать вещи и добавить их в Shop. Переопределить два метода Object. Реализовать интерфейс IEnumarable. Написать демонстрацию.
3. Переопределить операции +, - для добавления и удаления вещей в магазин.
4. Создайте класс Manager с событием sale (распродажа). Подпишите некоторые вещи на событие sale. Реакция на событие следующая, цена Item уменьшается на 50%. Промоделировать ситуацию с наступлением события и выведите содержимое очереди на консоль.
5. Напишите LINQ запрос и найдите в Shop сумму Item заданного имени (Name).

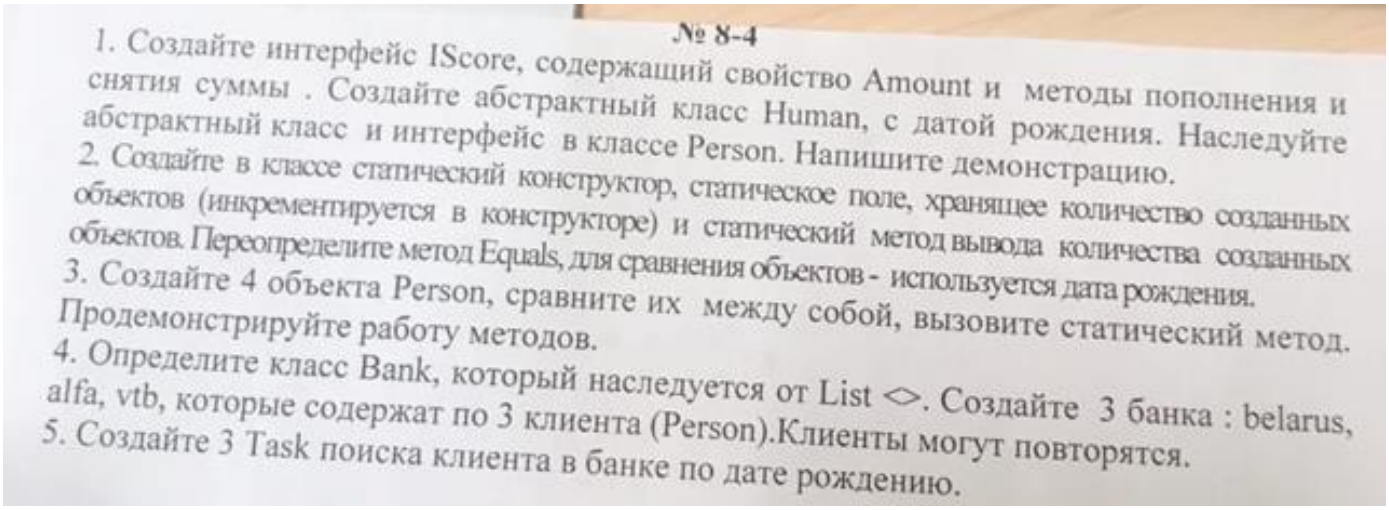
14.



№ 8-2

1. Создать абстрактный класс Function с виртуальной функцией: Func и свойством X. Создать производные классы: линейная функция Liner (содержит свойства A, B), квадратная Sqr (+ содержит C) . Определить функцию Func - для линейных – $ax+b$, для квадратной – ax^2+bx+c.
2. Создайте класс ArrayFunct<T>, содержащий массив разных функций (один). Для класса напишите индексатор. Переопределите 2 метода Object.
3. Создайте ArrayFunct<T> с 4мя объектами разных классов, продемонстрируйте виртуальный метод, индексатор.
4. Напишите запрос поиска функции Func с минимальным значением при заданном X и A в ArrayFunct.
5. Замените весь вывод на вывод в файл.

15.



№ 8-4

1. Создайте интерфейс IScore, содержащий свойство Amount и методы пополнения и снятия суммы . Создайте абстрактный класс Human, с датой рождения. Наследуйте абстрактный класс и интерфейс в классе Person. Напишите демонстрацию.
2. Создайте в классе статический конструктор, статическое поле, хранящее количество созданных объектов (инкрементируется в конструкторе) и статический метод вывода количества созданных объектов. Переопределите метод Equals, для сравнения объектов - используется дата рождения.
3. Создайте 4 объекта Person, сравните их между собой, вызовите статический метод. Продемонстрируйте работу методов.
4. Определите класс Bank, который наследуется от List <>. Создайте 3 банка : belarus, alfa, vtb, которые содержат по 3 клиента (Person).Клиенты могут повторятся.
5. Создайте 3 Task поиска клиента в банке по дате рождению.

16. Создать класс BSTUStudent, в котором следующие поля: имя, группа, курс, специальность- задача перечислением poit, isit, web, mobile. И 4 отметки за экзамены. Создать метод в классе BSTUStudent, которой возвращает кортеж, содержащий. Мин, макс и ср отметки за экзы. Создать класс Group, который хранит студентов в одной из необобщенных коллекций .Net и присвоить ей 4 студентов. С помощью Linq.Выведите два элемента с наибольшим средним баллом. Создать интерфейс IClearnable с методом Clearn , который очищает коллекцию в классе Group. Реализовать сам интерфейс.

17. 1. Создайте интерфейс Figure с виртуальным методом print.Создать класс Rectangle с координатами x, y, длиной и шириной h, l и color(строка) цветом и реализацией интерфейса.Метод print должен выводить прямоугольник на консоль.

2. Класс должен содержать: конструктор без параметров и с тремя, пятью параметрами (используйте делигирование конструкторов), переопределите метода ToString, оператор + int (добавление к ширине и высоте целого числа), метод вычисления площади.

3. Создайте коллекцию List<> из 6 прямоугольников.Продемонстрируйте работу оператора + и метода print.

4. Используя LINQ отсортируйте ее по x, а затем по y, затем по площади, возьмите первый и последний объекты и выедите их.

5. Сериализуйте коллекцию в формате json, координаты x, y – сделайте не сериализуемыми.

1.Создать generic класс ExamCard в котором можно объявить любую обобщенную коллекция .Net. Наследовать интерфейс IAction, в которой будут функции добавления удаления очистки и просмотра коллекции. Интерфейс реализовать явно.
2.Добавить исключения когда пытаемся очистить пустую коллекцию или когда пытаемся удалить элемент из пустой коллекции. Исключение называется NullSizeCollection (надо создать класс class NullSizeCollection : System.Exception). Реализовать для класса ограничение на конструктор.
3. Создать класс Student с полями Name, Mark, Subject, с которым будет работать ваша коллекция. Продемонстрировать работу.
4. Сделать запросы LINQ : 1) Вывести количество студентов, которые сдали экзамены выше либо равно 4. 2) Найти среднее значение среди оценок(Mark).
5. Создать метод расширения, который будет рандомно повышать оценку на 1,2,3. Продемонстрировать работу (не факт, что написал это задание правильно, ибо я его не успел сделать)

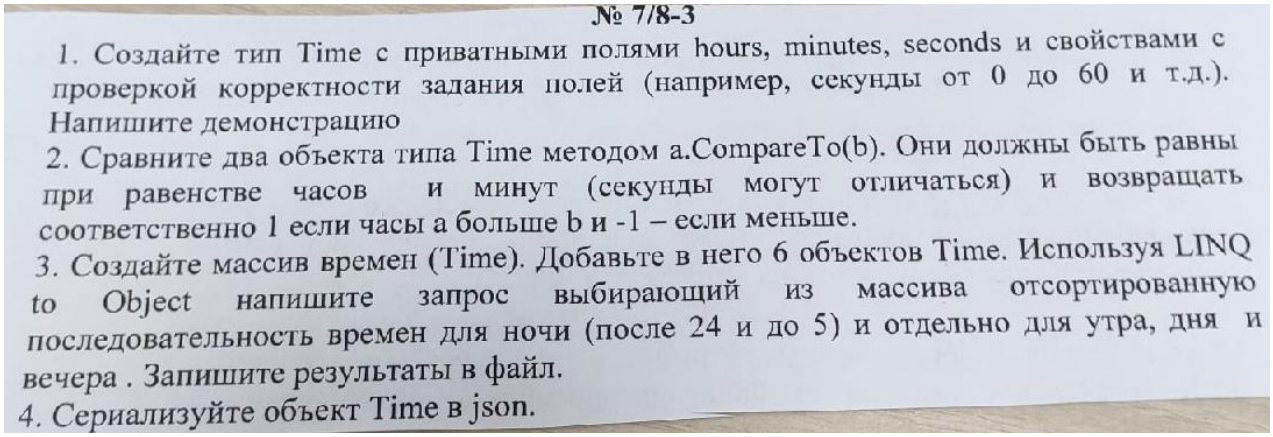
18. :) (рэд.)

19. 1.Создать абстрактный класс AbstractUser с полем Data. Создать клас User с полями password и login, который является потомком обстрактного класа.
    2-3. Создать исключения : 1)Если длинна пароля меньше 6 символов 2)если длинна пароля больше 12 символов 3) если пароль состоит из одних цифр.
    4. Создать 4 объекта типа user и поместить их в лист. Продемонстрировать работу исключений.
    5. На основе linq написать запрос : найти пользователя, добавленного раньше всех(по полю Data). Вроде как-то так.

20. 1. Создать тип (так и было написано) Car со свойствами: цена за сутки, номер и перечисление (free, busy). Создать тип LuxCar, который унаследовать от Car, со свойством "страховка". Реализовать метод GetPrice(int days). Для Car = возвращает days*цену за сутки, для LuxCar = days*цену за сутки+страховка. Продемонстрировать работу методов.
    2. Создать класс CarSharing и наследовать от обобщенной коллекции. Создать коллекцию из нескольких объектов. Переопределить методы Equals() и GetHashCode(). Показать, какие машины free, и какие busy. Продемонстрировать.
    3-4. Создать класс Manager с событием Discount. Для Car - цена за день становится на 10 ниже, для LuxCar - страховка становится = 0. Продемонстрировать.
    5. Создать запрос LINQ, который выводит 5 самых дорогих машин из коллекции объектов (как-то так).

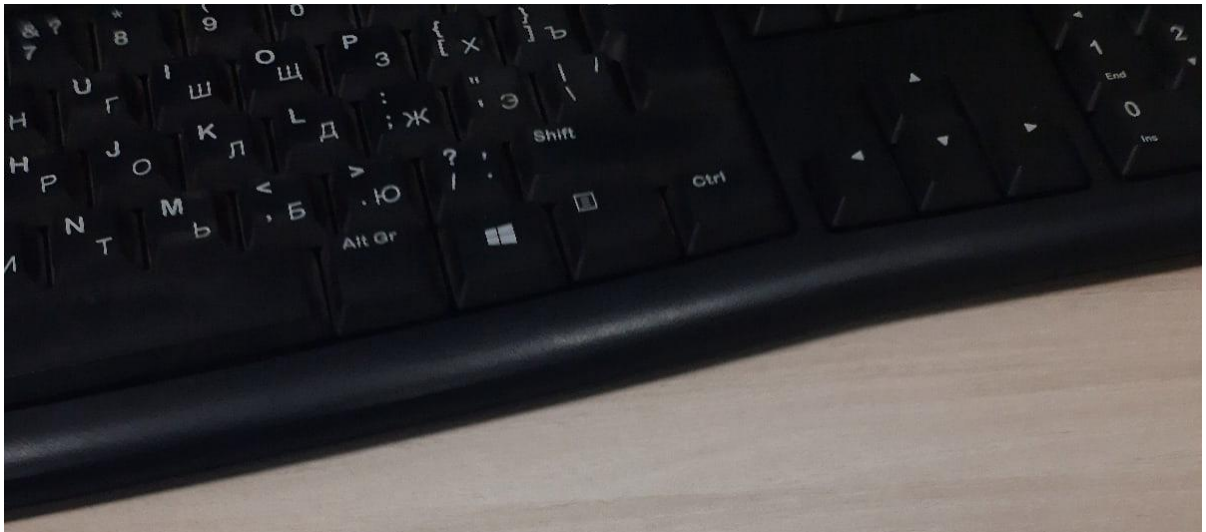
21. 1) Класс Human со свойствами name, age. От него наследовать Tutor со свойством level (может принимать значения либо 1, либо 2). Создать поле average. Создать объекты и продемонстрировать

2) переопределить операторы ++ и — на добавление к среднему баллу. Создать List и закинуть туда 3 объекта типа Tutor и 2 объекта Human. Продемонстрировать работу операторов
3) Создать класс директор. В нем события Up и Down. Привязать их к двум объектам типа Tutor. Вывести состояние объектов.
4) при помощи LINQ подсчитать средний балл в коллекции.
5) сериализовать коллекцию в json формате.

22. 

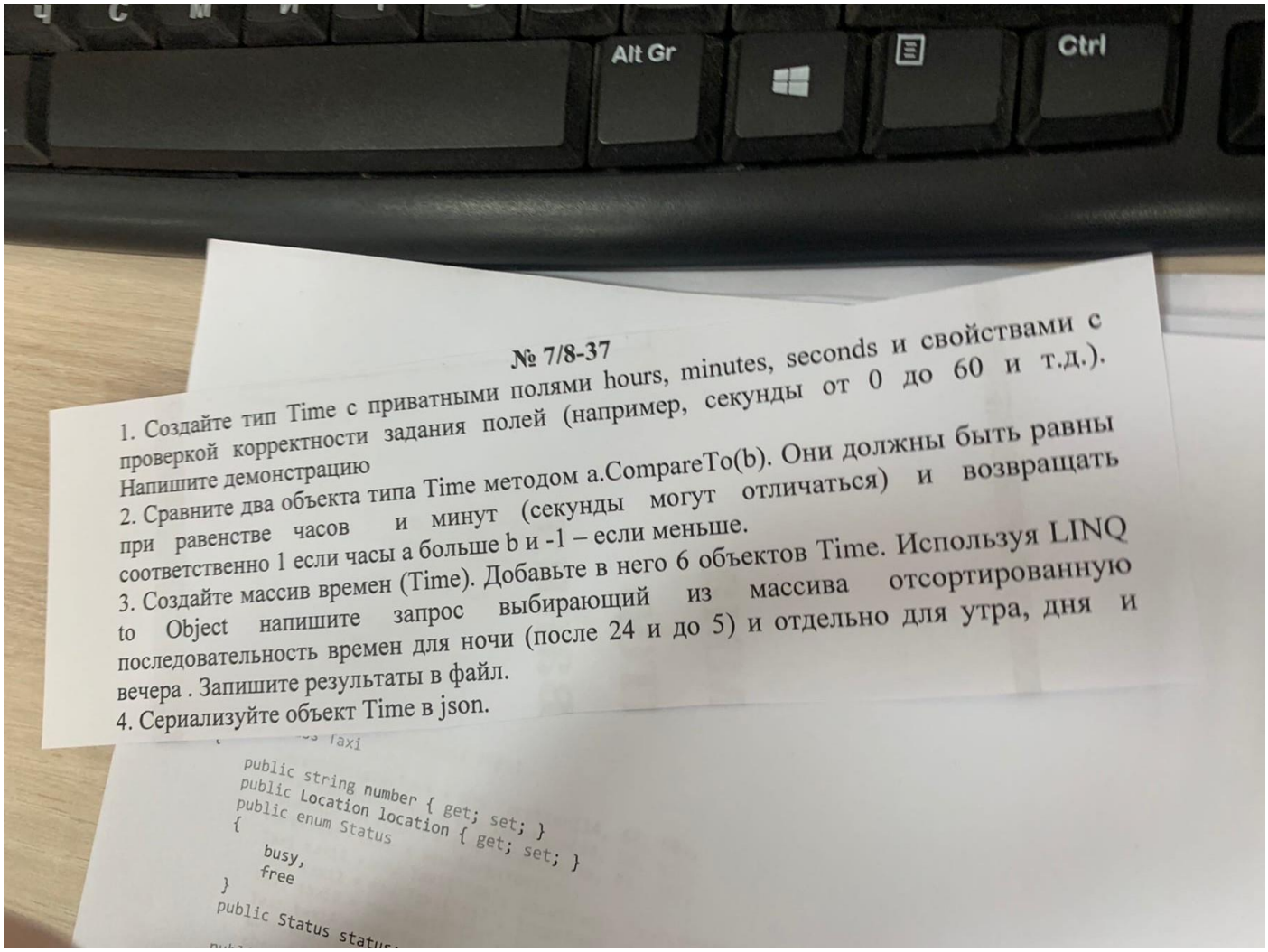
№ 7/8-3

1. Создайте тип Time с приватными полями hours, minutes, seconds и свойствами с проверкой корректности задания полей (например, секунды от 0 до 60 и т.д.). Напишите демонстрацию
2. Сравните два объекта типа Time методом a.CompareTo(b). Они должны быть равны при равенстве часов и минут (секунды могут отличаться) и возвращать соответственно 1 если часы a больше b и -1 – если меньше.
3. Создайте массив времен (Time). Добавьте в него 6 объектов Time. Используя LINQ to Object напишите запрос выбирающий из массива отсортированную последовательность времен для ночи (после 24 и до 5) и отдельно для утра, дня и вечера . Запишите результаты в файл.
4. Сериализуйте объект Time в json.



№ 5-2

1. Создать абстрактный класс Unit (состав произвольно). Наследуйте класс Own от Unit. Own содержит свойства Temp (температура), Time (время) и поле Status. Status принимает одно из значений перечисления ready, cooking, finish. Напишите демонстрацию.
2. Создайте интерфейс ICook с методами Cook и Check. Явно реализуйте ICook в классе Own. При вызове метода Cook проверяется статус и выводится сообщение в файл. Check проверяет температуру и время, если температура = 0, время >0 выставляйте статус ready, если температура >0 и время > 0 выставляет статус cooking, температура > 0, время =0 выставляйте статус finish.
3. Создайте массив Own, используя LINQ посчитайте сколько печей готовят (cooking).
4. Если в списке нет печей, которые готовят, сгенерируйте и обработайте пользовательское исключение NotWorkingException (наследуйте от стандартного).
5. Используя Task запустите для каждого объекта массива Own метод Cook.

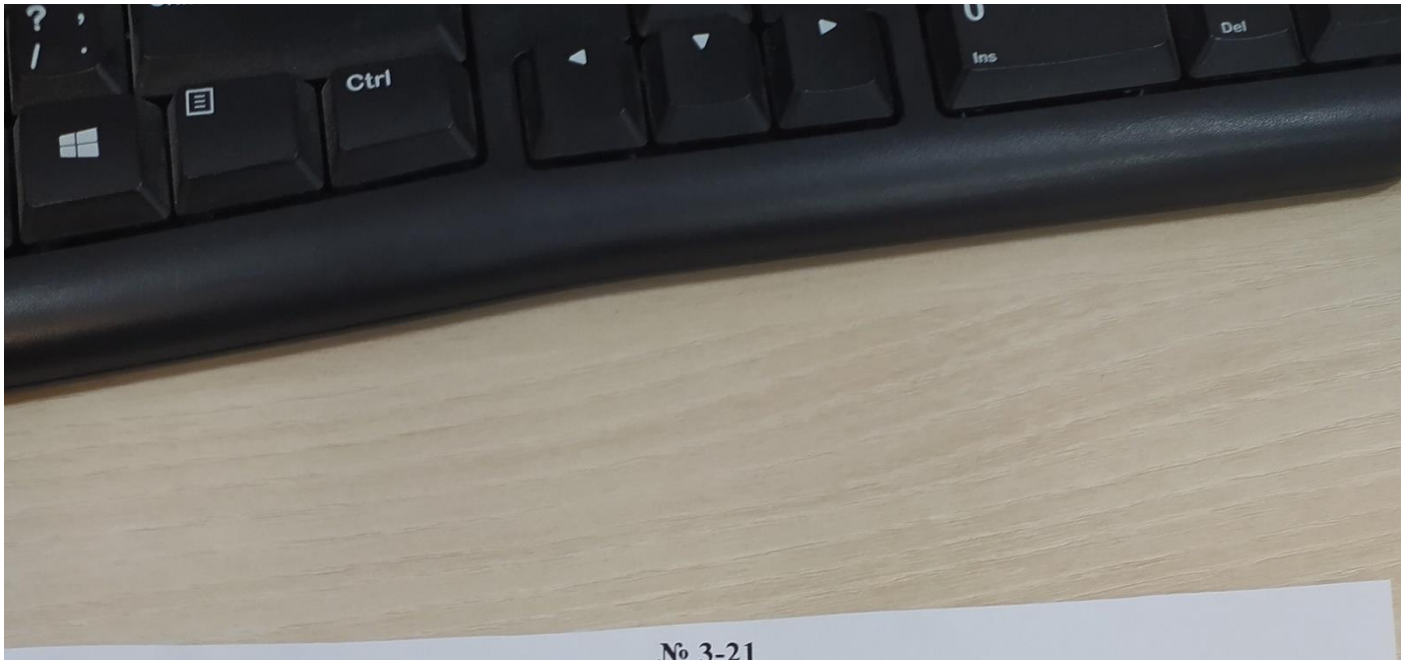


№ 7/8-37

1. Создайте тип Time с приватными полями hours, minutes, seconds и свойствами с проверкой корректности задания полей (например, секунды от 0 до 60 и т.д.). Напишите демонстрацию
2. Сравните два объекта типа Time методом a.CompareTo(b). Они должны быть равны при равенстве часов и минут (секунды могут отличаться) и возвращать соответственно 1 если часы a больше b и -1 – если меньше.
3. Создайте массив времен (Time). Добавьте в него 6 объектов Time. Используя LINQ to Object напишите запрос выбирающий из массива отсортированную последовательность времен для ночи (после 24 и до 5) и отдельно для утра, дня и вечера . Запишите результаты в файл.
4. Сериализуйте объект Time в json.

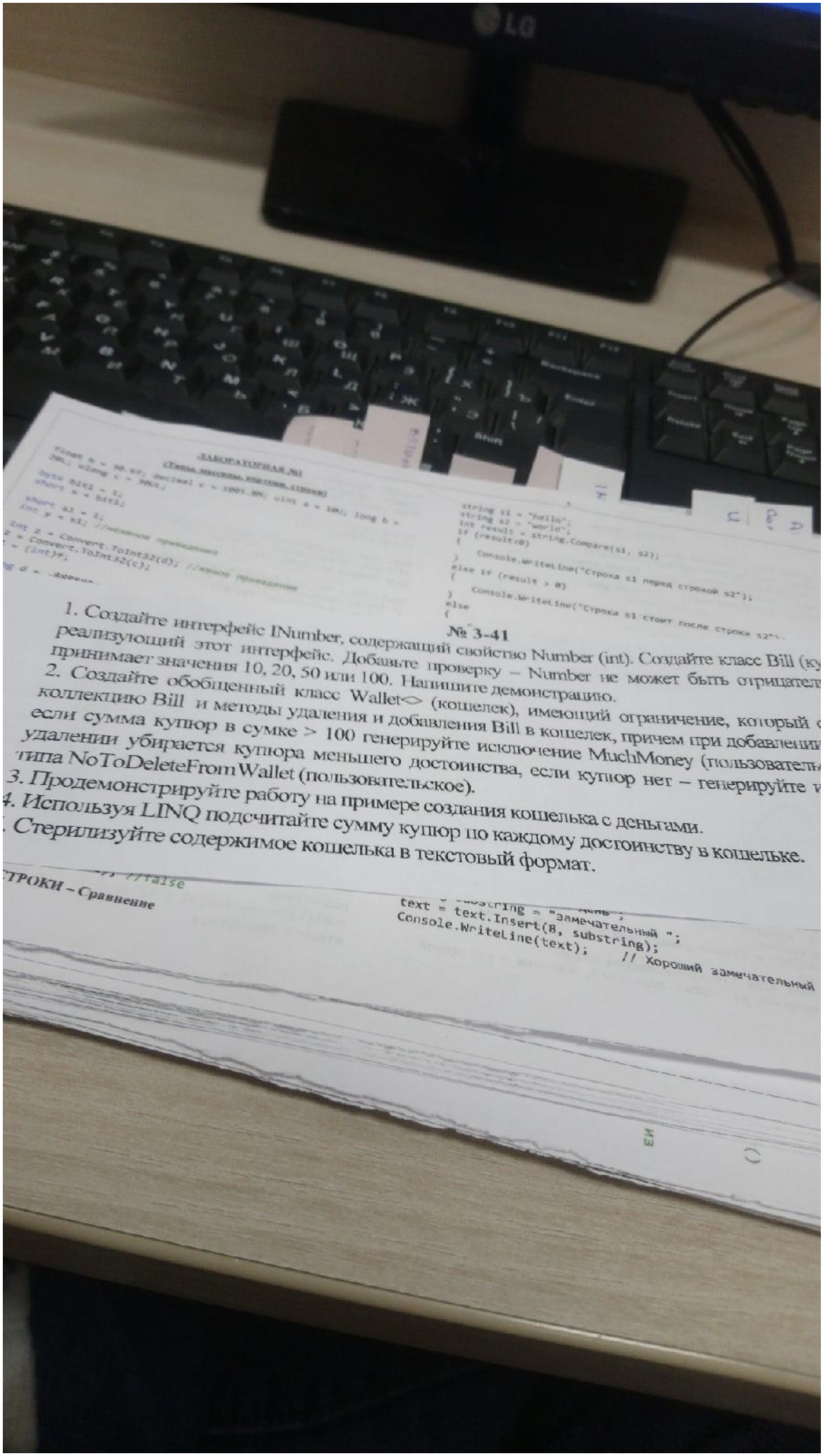
```
{ ... Taxi

    public string number { get; set; }
    public Location location { get; set; }
    public enum Status
    {
        busy,
        free
    }
    public Status status;
```



№ 3-21

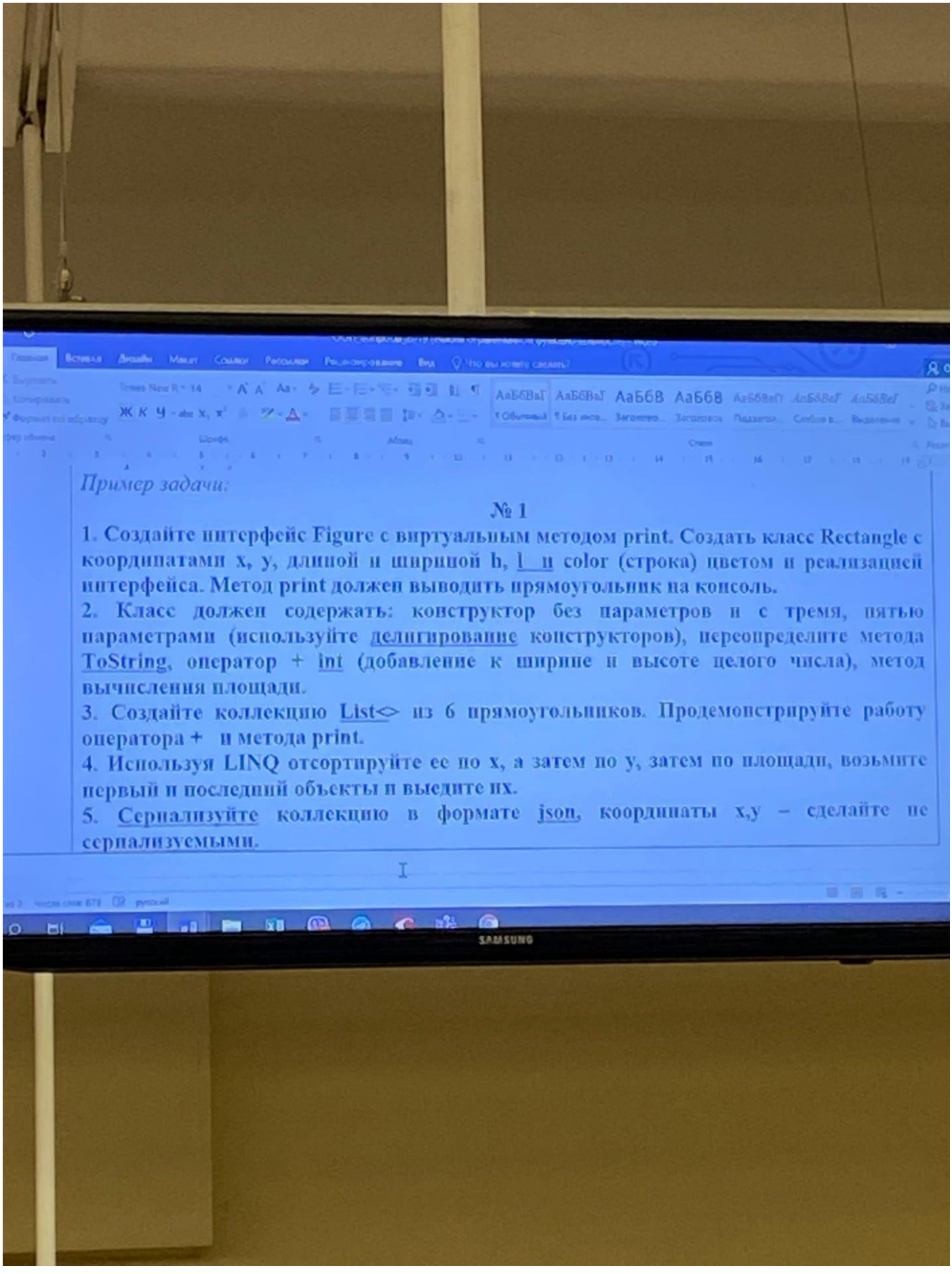
1. Разработать тип Stud, который хранит информацию о студенте (имя, номер группы, курс, специальность – задана перечислением и принимает значения poit, isit, mobile) и их оценках за 3 экзамена. Напишите демонстрацию.
2. Написать метод возвращающий кортеж с минимальной, максимальной оценкой за экзамен и средним баллом по всем экзаменам.
3. Создать класс Group, который содержит стандартную .Net коллекцию для хранения объектов Stud. Добавьте методы добавления элементов в коллекцию и вывода коллекции на консоль. Напишите демонстрацию.
4. Добавьте 5 объектов в Group и используя LINQ выведите объекты Stud для каждой специальности с максимальным средним баллом.
5. Создать интерфейс IClearnable с методом Clearn() (очистки) и реализовать его в Group. Если коллекция пуста, при попытке очистки генерируйте и обрабатывайте исключение.

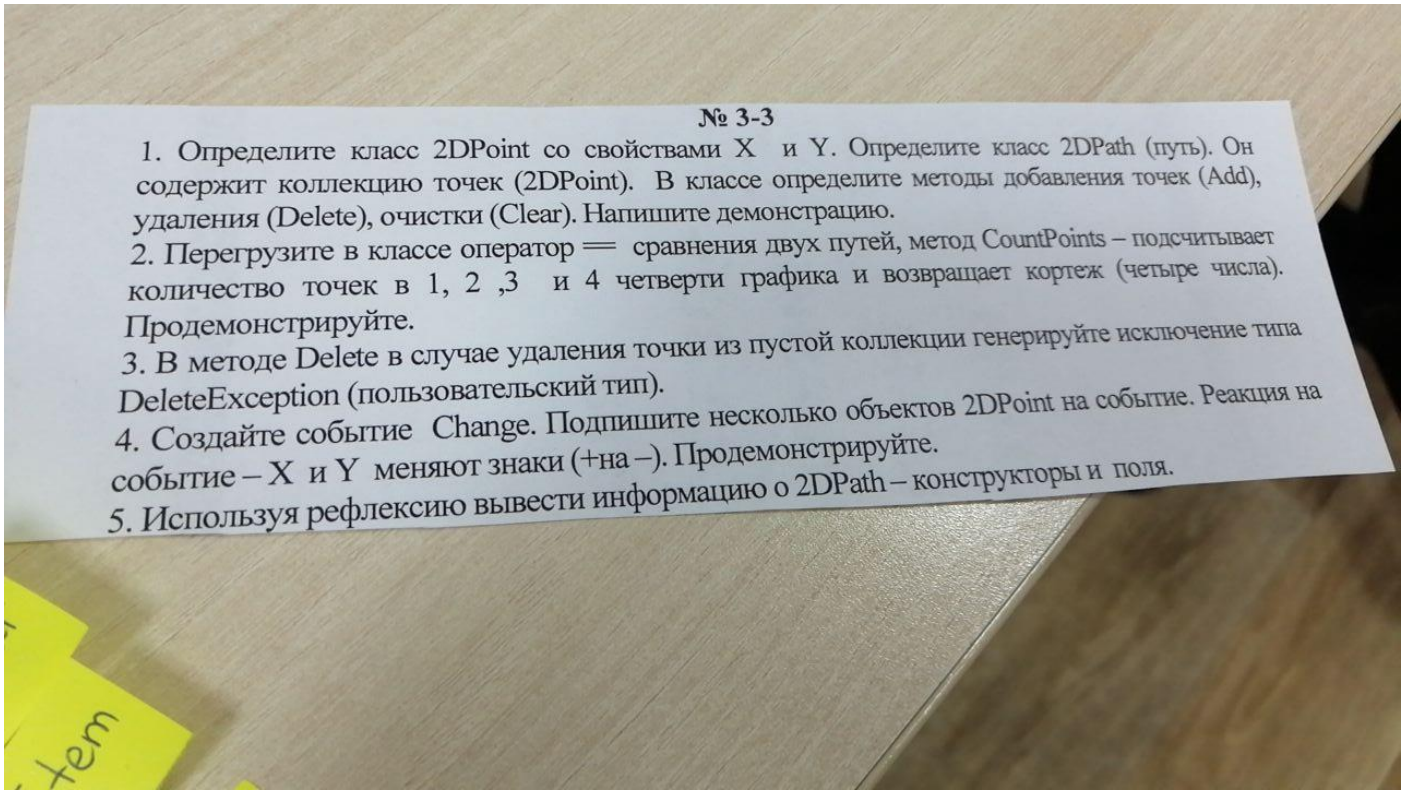

№ 3-41
1. Создайте интерфейс INumber, содержащий свойство Number (int). Создайте класс Bill (ку
реализующий этот интерфейс. Добавьте проверку – Number не может быть отрицател
принимает значения 10, 20, 50 или 100. Напишите демонстрацию.
2. Создайте обобщенный класс Wallet<> (кошелек), имеющий ограничение, который
коллекцию Bill и методы удаления и добавления Bill в кошелек, причем при добавлени
если сумма купюр в сумке > 100 генерируйте исключение MuchMoney (пользовател
удалении убирается купюра меньшего достоинства, если купюр нет – генерируйте
типа NoToDeleteFromWallet (пользовательское).
3. Продемонстрируйте работу на примере создания кошелька с деньгами.
4. Используя LINQ подсчитайте сумму купюр по каждому достоинству в кошельке.
. Стерилизуйте содержимое кошелька в текстовый формат.
СТРОКИ – Сравнение
text = text.Insert(8, substring);
Console.WriteLine(text);
// Хороший замечательный

№ 6-42

1. Создать интерфейс IEdit с методом Delete(). Создать абстрактный класс Redactor с методом Delete () и свойством Text типа StringBuilder. Наследовать IEdit и Redactor в классе Document. У методов должна быть разная реализация. В одном Delete – из Text удаляйте лишние пробелы, в другом Delete – удаляйте все слова кроме первого. Создать объект, продемонстрировать работу двух методов.
2. Переопределите в Document два любых метода из Object. Добавьте метод Print() который выводит на консоль содержимое Text в одну строку. Создайте производны от него класс Book, у которого Print() должен выводить каждое предложение с нов строки.
3.Создайте один объект archive типа List с 3-мя объектами Document и 2-мя Bo Циклом для каждого объекта из archive вызовите Print и Delete.
4. Замените в Print –ах вывод с вывода на консоль в вывод файл. Файлы дол создаваться каждый раз новые с уникальным именем (используйте время создан имени файла).
5.Добавьте для класса Book метод расширения ToBeContinue (в конце выводи Продемонстрируйте работу этого метода.

```
Dequeue();
ear()
ear();
numerator<Item> GetEnumerator()
urn Items.GetEnumerator();
IEnumerable.GetEnumerator()
GetEnumerator();
```


Пример задачи:
№ 1
1. Создайте интерфейс Figure с виртуальным методом print. Создать класс Rectangle с координатами x, y, длиной и шириной h, l и color (строка) цветом и реализацией интерфейса. Метод print должен выводить прямоугольник на консоль.
2. Класс должен содержать: конструктор без параметров и с тремя, пятью параметрами (используйте делигирование конструкторов), переопределите метода ToString, оператор + int (добавление к ширине и высоте целого числа), метод вычисления площади.
3. Создайте коллекцию List<> из 6 прямоугольников. Продемонстрируйте работу оператора + и метода print.
4. Используя LINQ отсортируйте ее по x, а затем по y, затем по площади, возьмите первый и последний объекты и выедите их.
5. Сериализуйте коллекцию в формате json, координаты x,y – сделайте не сериализуемыми.
SAMSUNG



№ 3-3

1. Определите класс 2DPoint со свойствами X и Y. Определите класс 2DPath (путь). Он содержит коллекцию точек (2DPoint). В классе определите методы добавления точек (Add), удаления (Delete), очистки (Clear). Напишите демонстрацию.
2. Перегрузите в классе оператор == сравнения двух путей, метод CountPoints – подсчитывает количество точек в 1, 2 ,3 и 4 четверти графика и возвращает кортеж (четыре числа). Продемонстрируйте.
3. В методе Delete в случае удаления точки из пустой коллекции генерируйте исключение типа DeleteException (пользовательский тип).
4. Создайте событие Change. Подпишите несколько объектов 2DPoint на событие. Реакция на событие – X и Y меняют знаки (+на –). Продемонстрируйте.
5. Используя рефлексию вывести информацию о 2DPath – конструкторы и поля.