ДЕНЬ 4В. Спецификации

4В.1. Создать класс, реализующий заданный интерфейс

Необходимо создать консольное приложение, позволяющее вводить числа с клавиатуры и выполнять с ними заданную операцию.

В коде программы должен быть класс, реализующий следующий интерфейс:

```
interface ICalculator
{
    void AddNumber(float number);
    float Calculate();
}
```

При запуске программы должен создаваться экземпляр класса, реализующего интерфейс *ICalculator*. Далее в цикле ввода чисел каждое новое число передается в метод *AddNumber*. Как только пользователь ввел пустую строку, ввод заканчивается, и программа должна вызывать метод *Calculate*, который выполняет заданную операцию с теми числами, которые передавались в метод *AddNumber*. После расчета полученный результат должен быть выведен в консоль.

В созданном классе, реализующем *ICalculator*, не должно быть обращения к консоли. Все члены этого класса, не указанные в интерфейсе, должны иметь модификатор доступа private.

Варианты

№	Операция	№	Операция
1	У каждого второго числа поменять знак и просуммировать все числа	11	Перемножить дробные части всех чисел
2	Просуммировать дробные части всех чисел	12	Просуммировать модули всех чисел
3	Просуммировать все числа, которые меньше, чем предыдущее число	13	Умножить числа на их порядковые номера и вычислить среднее
4	Вычислить среднее из квадратов чисел	14	Просуммировать квадраты всех чисел
5	Перемножить все отрицательные числа	15	У каждого второго числа поменять знак и посчитать среднее
6	Просуммировать все числа, которые больше, чем предыдущее число	16	Просуммировать все числа, которые больше первого из чисел
7	Просуммировать все четные числа	17	Просуммировать все нечетные числа
8	Извлечь квадратный корень из суммы квадратов чисел	18	Умножить числа на их порядковые номера и сложить результаты
9	Округлить числа и сложить результаты	19	Перемножить все положительные числа
10	Разделить все числа на первое из чисел и вычислить сумму результатов	20	Вычислить среднее значение

4В.2. Разработать модульный тест для функции проверки правильности ввода данных, бросающей пользовательское исключение в случае ошибки

Необходимо разработать функцию проверки правильности ввода данных для заданной записи и модульный тест, ее проверяющий. Функция проверки правильности ввода должна принимать на вход объект записи. Если функция обнаружила ошибку в данных, она должна бросить **исключение вами созданного типа**, который помимо текста ошибки, должен содержать номер поля (начиная с 1), в котором обнаружена ошибка.

Для создания собственного типа исключения необходимо создать класс, который наследуется от класса *Exception*. Добавьте в него дополнительное свойство – номер поля в записи.

Решение должно состоять из двух проектов:

- *Библиотека классов (.NET Framework)*, в которой должен быть класс с функцией проверки правильности ввода данных, класс, представляющий собой запись, и класс собственного типа исключения
- *Проект модульного теста (.NET Framework),* в котором должен быть реализован модульный тест.

Модульный тест должен проверить, как положительный исход функции, так и все варианты отрицательного исхода. В случае отрицательного исхода тест должен проверить как тип брошенного исключения (он должен быть вами созданным), так и номер поля.

Варианты

№	Поля записи и их проверка		
	Поле 1	**Поле 2**	**Поле 3**
1	Компания (строка), не меньше 10 символов	Сумма поступлений в млн. руб. (вещественное число): не меньше 0 и не больше 1000000	Сумма списаний в млн. руб. (вещественное число): не меньше 0
2	Номер телефона (строка): только цифры, скобки, знаки тире, плюса и пробела	Имя оператора (строка): не меньше 3 символов	баланс в копейках (целое число): не меньше -9999999
3	Фамилия (строка): не меньше 3 букв	Номер группы (строка): только цифры, знак дефиса и русские буквы	Номер в группе (целое число): больше 0
4	Номер заказа (строка): только латинские буквы и цифры	Описание (строка): не меньше 200 символов	Сумма заказа (целое число): не меньше 0
5	Название цеха (строка): только буквы и цифры	План выпуска деталей (целое число): не больше 1000	Фактический выпуск деталей (целое число): не меньше 0
6	Название товара (строка): не меньше 10 символов	Количество на складе (целое число): от 0 до 999 включительно	Количество зарезервированных (целое число): не меньше 0
7	Фамилия (строка): не меньше 3 букв	Число ролей (целое число): не меньше 0 и не больше 9999	Гонорар в млн. руб. (вещественное число): не меньше 0
8	Название материала (строка): не меньше 10 символов	Объем (вещественное число): не больше 1000	Вес (вещественное число): не меньше 0

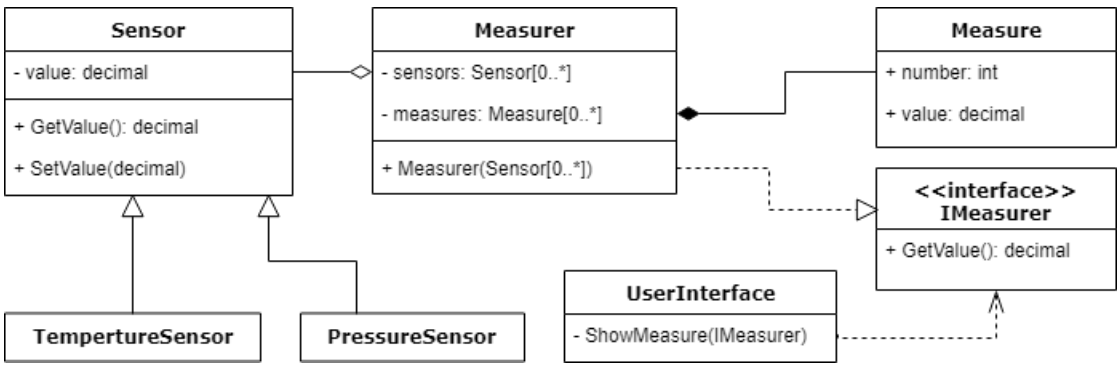
9	Адрес сайта (строка): только латинские буквы, точка, тире и цифры	Число посетителей (целое число): не меньше числа уникальных посетителей	Число уникальных посетителей (целое число): не меньше 0
10	Автомобильный номер (строка): только цифры и буквы A, B, E, K, M, H, O, P, C, T, У, X	Год выпуска (целое число): от 1980 до текущего года включительно	Пробег в км (целое число): больше или равно 0
11	Производитель (строка): не меньше 10 символов	Объем выпуска (вещественное число): не меньше 0	Средняя цена (вещественное число): от 1 до 99999 включительно
12	Фамилия (строка): не меньше 3 букв	Должность (строка): только русские буквы	Оклад в руб (целое число): больше 0
13	Фамилия (строка): не меньше 3 букв	Оценка за теорию (целое число): не меньше 3	Оценка за практику (целое число): не больше 5
14	Название (строка): не больше 255 символов	Число сезонов (целое число): больше или равно одному	Год выпуска первого сезона (целое число): от 2000 до текущего года включительно
15	Тема письма (строка): не больше 1024 символов	Адресат (строка): только латинские буквы, знак «@», точка, тире и цифры	Число слов (целое число): не меньше 1
16	Город (строка): не больше 255 символов	Улица (строка): только русские буквы, знак дефиса, пробела и цифры	Номер дома (число): больше 0
17	Дисциплина (строка): только русские буквы, знаки дефиса и пробела	номер курса (целое число): от 0 до 6 включительно	количество часов (целое число): от 0 до 9999 включительно
18	Фамилия (строка): не меньше 3 букв	Год поступления (целое число): не меньше 1952 и не больше текущего года	Средний балл (вещественное число): от 0 до 5 включительно
19	Фамилия (строка): не меньше 3 букв	Рост (вещественное число): не больше 300	Вес (вещественное число): больше 30
20	Адрес отправления (строка): не меньше 30 символов	Адрес доставки (строка): не равен адресу отправления	Вес (вещественное число): не больше 90

4В.3. Создать декларацию классов C# согласно заданной UML-диаграмме классов

Создать код декларации классов (без их реализации) на языке C# по заданной UML-диаграмме классов

Пример

Диаграмма классов:

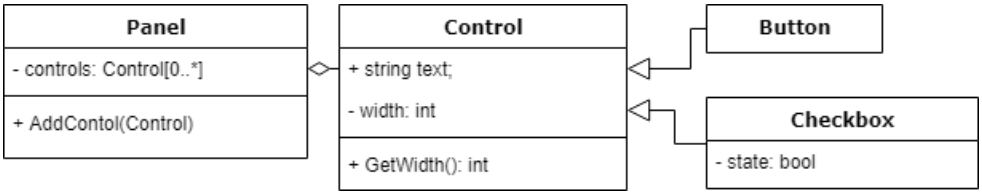
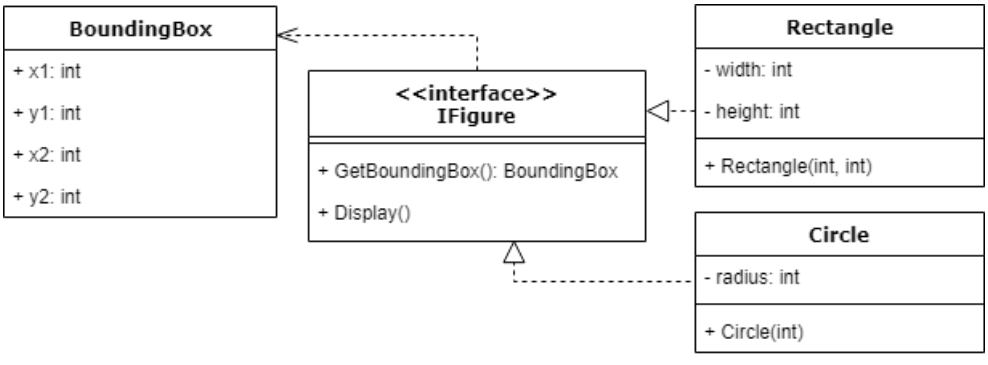


Код декларации классов:

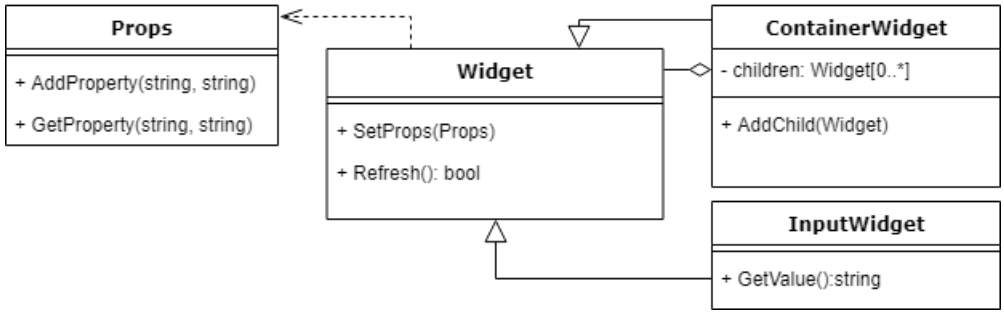
```
public class Sensor
{
    private decimal value;

    public decimal GetValue()
    {
        throw new NotImplementedException();
    }

    public void SetValue(decimal val)
    {
        throw new NotImplementedException();
    }
}

public class TempertureSensor: Sensor
{

}

public class PressureSensor: Sensor
{

}

public class Measure
{
    public int number;
    public decimal value;
}
```

```
public interface IMeasurer
{
    decimal GetValue();
}

public class Measurer: IMeasurer
{
    private List<Sensor> sensors;
    private List<Measure> measures;

    public Measurer(List<Sensor> sens)
    {
        throw new NotImplementedException();
    }

    public decimal GetValue()
    {
        throw new NotImplementedException();
    }
}

public class UserInterface
{
    private void ShowMeasure(IMeasurer)
    {
        throw new NotImplementedException();
    }
}
```

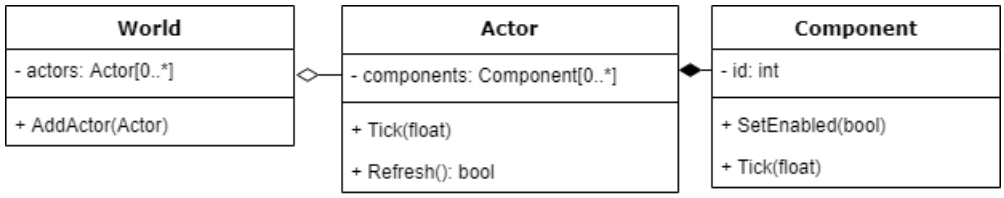
Варианты

№	Диаграмма
1	
2	

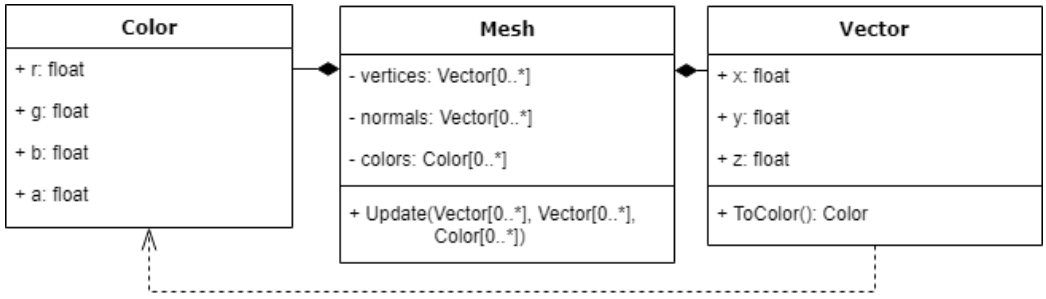
3


Props
+ AddProperty(string, string)
+ GetProperty(string, string)
Widget
+ SetProps(Props)
+ Refresh(): bool
ContainerWidget
- children: Widget[0..*]
+ AddChild(Widget)
InputWidget
+ GetValue():string

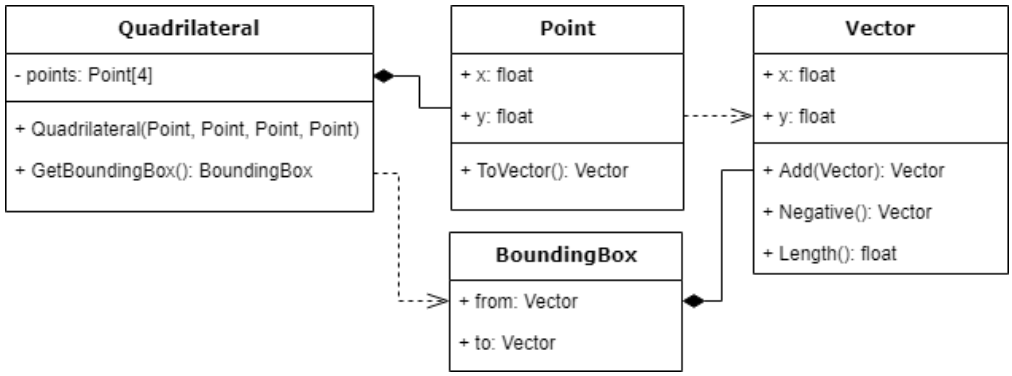
4


World
- actors: Actor[0..*]
+ AddActor(Actor)
Actor
- components: Component[0..*]
+ Tick(float)
+ Refresh(): bool
Component
- id: int
+ SetEnabled(bool)
+ Tick(float)

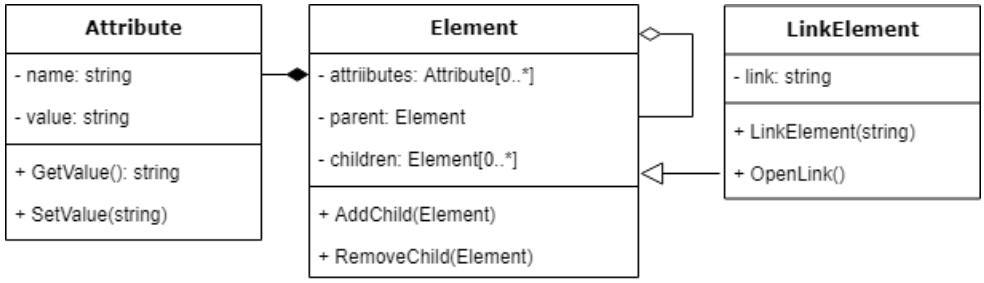
5

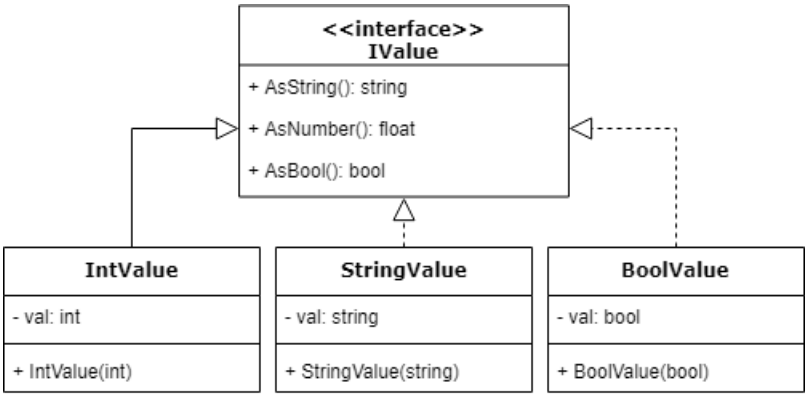
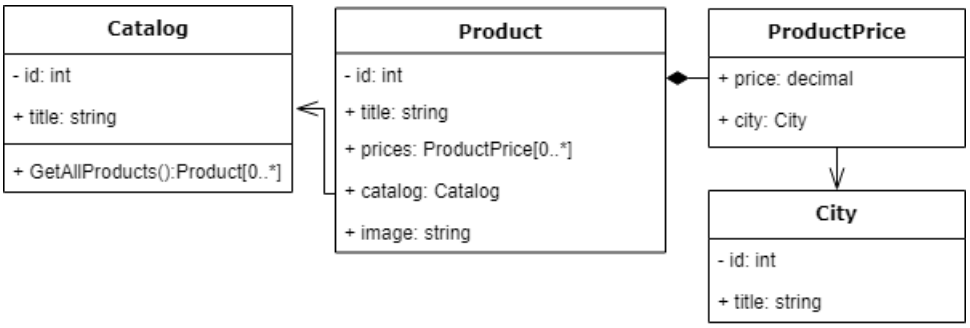
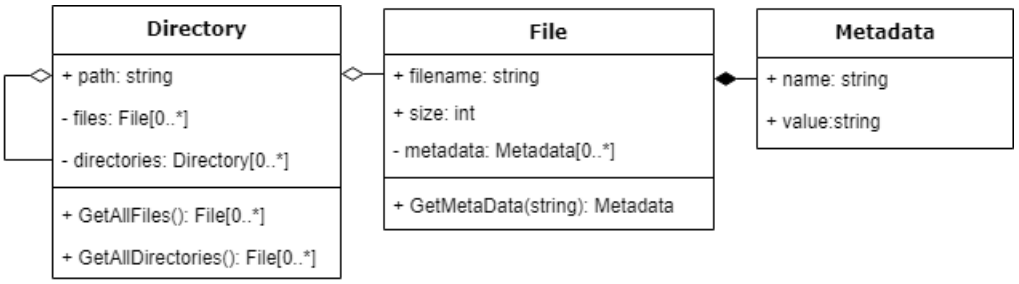

Color
+ r: float
+ g: float
+ b: float
+ a: float
Mesh
- vertices: Vector[0..*]
- normals: Vector[0..*]
- colors: Color[0..*]
+ Update(Vector[0..*], Vector[0..*], Color[0..*])
Vector
+ x: float
+ y: float
+ z: float
+ ToColor(): Color

6


Quadrilateral
- points: Point[4]
+ Quadrilateral(Point, Point, Point, Point)
+ GetBoundingBox(): BoundingBox
Point
+ x: float
+ y: float
+ ToVector(): Vector
Vector
+ x: float
+ y: float
+ Add(Vector): Vector
+ Negative(): Vector
+ Length(): float
BoundingBox
+ from: Vector
+ to: Vector

7


Attribute
- name: string
- value: string
+ GetValue(): string
+ SetValue(string)
Element
- attriibutes: Attribute[0..*]
- parent: Element
- children: Element[0..*]
+ AddChild(Element)
+ RemoveChild(Element)
LinkElement
- link: string
+ LinkElement(string)
+ OpenLink()

8	
9	
10	
11	см. вариант 1
12	см. вариант 2
13	см. вариант 3
14	см. вариант 4
15	см. вариант 5
16	см. вариант 6
17	см. вариант 7
18	см. вариант 8
19	см. вариант 9
20	см. вариант 10

4В.4. Сконвертировать входной CSV-файл в бинарный файл заданного формата

Вы должны разработать программу (консольную или оконную, на Ваш выбор), которая преобразует входной CSV-файл в бинарный файл заданного формата. Бинарный файл должен состоять из трех секций:

1. Заголовок
2. Оглавление
3. Данные

Секция «Заголовок» состоит из 6 байт:

Смещение	**Размер**	**Описание**
00-01	2	Два байта 0x50 (сигнатура)
02-05	4	Типы полей вашей структуры по варианту. Первый байт задает тип первого поля, второй – второго и т.д. Возможные значения типа: • 0x49 – целый • 0x44 – вещественный • 0x42 – логический • 0x53 – строковый • 0x00 – поля нет

Секция «Оглавление» состоит из двух элементов:

- количество записей в файле (4 байта)
- список, в котором для каждой записи указано смещение, по которому находятся данные этой записи. Смещения указываются относительно начала секции «Данные» и занимают 4 байта. Таким образом, если в файле N записей, размер списка – 4N байт

В секции «Данные» последовательно располагается содержимое записей. В зависимости от типа, поля в записи занимают разный объем памяти

- *Целое поле* занимает 4 байта
- *Вещественное поле* – 8 байт
- *Логическое поле* – 1 байт (0x01 – true, 0x00 – false)
- *Строковое поле* записывается в виде двух значений: размер строки в байтах (2 байта) и сама строка (в кодировке UTF-8)

Порядок записи байт в числах – от младшего к старшему (little-endian)

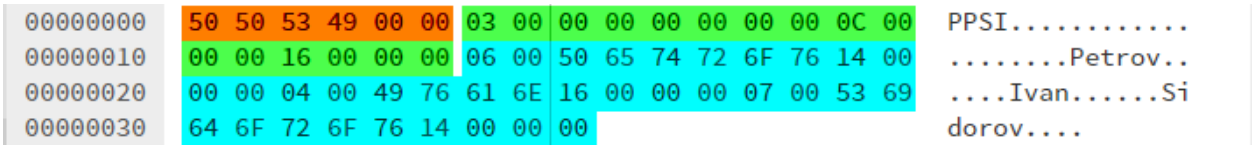
Пример

Запись: фамилия (строка), возраст (целое число)

Входной файл:

```
Petrov, 20
Ivan, 22
Sidorov, 20
```

Выходной бинарный файл (в шестнадцатеричном представлении)



Секция «Заголовок» обозначена оранжевым цветом:

Фрагмент	**Описание**
50 50	Сигнатура файла
53 49 00 00	Два поля в структуре: 1) строковое, 2) целочисленное

Секция «Оглавление» обозначена зеленым цветом

Фрагмент	**Описание**
03 00 00 00	Количество записей в файле (3)
00 00 00 00	Смещение первой записи относительно секции «Данные» (0)
0C 00 00 00	Смещение второй записи относительно секции «Данные» (12)
16 00 00 00	Смещение третьей записи относительно секции «Данные» (22)

Секция «Данные» обозначена голубым цветом

Фрагмент	**Описание**
06 00 50 65 74 72 6F 76	Поле «Фамилия» записи №1 (длина 6 байт, “Petrov”)
14 00 00 00	Поле «Возраст» записи №1 (20)
04 00 49 76 61 6E	Поле «Фамилия» записи №2 (длина 4 байта, “Ivan”)
16 00 00 00	Поле «Возраст» записи №2 (22)
07 00 53 69 64 6F 72 6F 76	Поле «Фамилия» записи №3 (длина 7 байт, “Sidorov”)
14 00 00 00	Поле «Возраст» записи №3 (20)

Варианты

№	**Запись**
1	Фамилия (строка), номер группы (строка), номер в группе (целое число), число выполненных заданий (целое число)
2	Тема письма (строка), адресат (строка), есть ли вложения (логический тип), число слов (целое число)
3	Фамилия (строка), число ролей (целое число), гонорар в млн. руб. (вещественное число)
4	Производитель (строка), объем выпуска (вещественное число), средняя цена (вещественное число)
5	Адрес отправления (строка), адрес доставки (строка), вес (вещественное число)
6	Название товара (строка), количество на складе (целое число), количество зарезервированных (целое число)
7	Город (строка), улица (строка), номер дома (число), номер этажа (целое число)
8	Номер заказа (строка), описание (строка), выполнен или нет (логический тип), сумма заказа (целое число)
9	Номер телефона (строка), имя оператора (строка), баланс в копейках (целое число)
10	Название материала (строка), объем (вещественное число), вес (вещественное число)

11	Фамилия (строка), год поступления (целое число), средний балл (вещественное число)
12	Название (строка), число сезонов (целое число), год выпуска первого сезона (целое число)
13	Фамилия (строка), оценка за теорию (целое число), оценка за практику (целое число)
14	Дисциплина (строка), номер курса (целое число), количество часов (целое число)
15	Адрес сайта (строка), число посетителей (целое число), число уникальных посетителей (целое число)
16	Фамилия (строка), рост (вещественное число), вес (вещественное число)
17	Компания (строка), сумма поступлений в млн. руб. (вещественное число), сумма списаний в млн. руб. (вещественное число)
18	Автомобильный номер (строка), год выпуска (целое число), пробег в км (целое число)
19	Название цеха (строка), план выпуска деталей (целое число), фактический выпуск деталей (целое число)
20	Фамилия (строка), должность (строка), оклад в руб (целое число)

4В.5. Разработать программу, выполняющую команды, полученные по сети

Вам необходимо разработать программу (консольную или оконную, на Ваш выбор) для обработки списка целых чисел. Команды для обработки чисел программа должна получать по сети путем создания TCP-подключения к серверу.

Программа-сервер уже написана, ее сборка и исходный код лежат по адресу https://github.com/Nordth/istu-priklad-practic-2022/tree/master/media/day4b/Task_NetServer. После запуска программа-сервер программа создаст локальный сервер на порту 3005 (порт можно изменить, указав новый порт в аргументах командной строки)

В зависимости от варианта, ваша программа должна реализовывать несколько команд из следующего списка:

1. *add* <*number*> - добавить число <number> в список для обработки
2. *range* <*number1*> <*number2*> - добавить числа от <*number1*> до <*number2*> с шагом 1 в список для обработки
3. *rand* – добавить случайное число от -100 до 100 в список для обработки и вывести его на экран
4. *copy* – добавить в список для обработки копию последнего из ранее добавленных. Если в список числа еще не добавлялись, то добавить в список число 0
5. *clear* – очистить список чисел для обработки
6. *pop* – убрать из списка для обработки последнее из добавленных чисел и вывести его на экран
7. *mul* <*number*> – умножить все числа в списке для обработки на <number>
8. *neg* – поменять знак у всех чисел в списке для обработки
9. *abs* – сделать все числа в списке для обработки положительными
10. *print* – вывести на экран все числа в списке для обработки
11. *top* – вывести на экран последнее из добавленных в список для обработки чисел. Если чисел не было, то вывести «Список пуст»
12. *count* – вывести количество чисел, находящихся в списке для обработки
13. *countodd* – вывести количество нечетных чисел, находящихся в списке для обработки

14. *counteven* – вывести количество четных чисел, находящихся в списке для обработки
15. *sum* – вывести сумму чисел, находящихся в списке для обработки
16. *sumodd* – просуммировать все нечетные числа в списке для обработки
17. *sumeven* – просуммировать все четные числа в списке для обработки
18. *avg* – вывести среднее значение чисел, находящихся в списке для обработки

Для того, чтобы создать подключение к серверу выполните следующее:

```
using (TcpClient client = new TcpClient())
{
    client.Connect("127.0.0.1", 3005);
    using (NetworkStream stream = client.GetStream())
    {
        // Действия
    }
}
```

Работа с TCP-подключением похожа на работу с файлом. Общение с сервером будет происходить по бинарному протоколу, поэтому для удобства вы можете использовать классы *BinaryReader* и *BinaryWriter* с потоком *NetworkStream.* Порядок записи байт в числах – от младшего к старшему (little-endian)

После подключения вам необходимо отправить информацию, какие команды, используются в Вашем варианте. Для этого вы должны отправить 5 номеров команд в виде 5-ти однобайтовых чисел (нумерация соответствует нумерации списка выше)

После отправки перечня команд сервер отправит 4 байта – кол-во команд, который вам пришлет сервер.

После отправки числа команд сервер будет случайным образом посылать команды на исполнение. Команда на исполнение состоит из номера команды (1 байт) и, если есть, из аргументов команды (каждый аргумент занимает 4 байта)

Вам необходимо принять команду, вывести на экран ее название, аргументы и выполнить ее.

После отправки всех команд сервер разорвет соединение.

Перед завершением работы вашей программы вы должны вывести текущий список чисел

Пример

1. После подключения вы отправили номера команд: 1, 2, 7, 8, 15 (0x010207080F)
2. Сервер вам в ответ прислал, что число команд равно 4 (0x04000000)
3. Сервер отправил команду *add 6* (0x0106000000)
4. У вас в списке на обработку теперь одно чисдо – шесть
5. Сервер отправил команду *range 10 15* (0x010A0000000F000000)
6. У вас в списке на обработку теперь следующие числа: 6, 10, 11, 12, 13, 14, 15

7. Сервер отправил команду *sum* (0x0F)
8. Вы выводите сумму, равную 81
9. Сервер отправил команду *mul 2* (0x0702000000)
10. У вас в списке на обработку теперь числа: 12, 20, 22, 24, 26, 28, 30

Варианты

№	**Команды**				
1	range	rand	pop	top	sumodd
2	add	rand	clear	countodd	avg
3	rand	copy	clear	print	sum
4	range	rand	mul	top	avg
5	add	copy	abs	print	sumeven
6	rand	copy	neg	print	avg
7	range	rand	abs	top	counteven
8	add	range	pop	counteven	sumodd
9	add	copy	abs	top	sum
10	range	rand	clear	counteven	avg
11	add	range	clear	print	counteven
12	add	rand	neg	sumodd	avg
13	range	copy	abs	top	countodd
14	add	rand	neg	countodd	sum
15	add	copy	neg	countodd	sumeven
16	rand	copy	mul	print	avg
17	range	copy	mul	sumeven	avg
18	range	copy	mul	top	count
19	add	rand	pop	print	count
20	add	range	pop	sumodd	avg