Общие требования к организации входных и выходных данных

1. Предполагается, что файл с исходными данными расположен в папке проекта, рядом с exe-файлом приложения, другие расположения файла программа не проверяет.
2. Гарантируется, что пустых строк в файле с исходными данными нет.
3. Файл с выходными данными размещается в той же папке, что и исходные данные.
4. Для вариантов с обработкой символов гарантируется, что строки файла с исходными данными состоят из латинских символов, цифр и пробелов
5. Для вариантов с обработкой числовых значений гарантируется, что строки файла с исходными данными состоят из цифр, пробелов, запятых и точек (запятые и точки представляют десятичные разделители у вещественных значений, значения, представленные в виде мантиссы и порядка, отсутствуют). Выходные данные используют в качестве десятичного разделителя один из символов: точка или запятая.
6. Файлы должны быть представлены в кодировке **windows-1251**.
7. Пустые строки, сгенерированные программой в файлы с выходными данными, не попадают.

Вариант 1. Фильтруем и урезаем

Разработать консольное приложение, которое позволяет пользователю фильтровать строки, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования объектов-фильтров, описываемых иерархией фильтров из библиотеки классов. В результате применения фильтров программа создаёт новый файл, в котором каждая строка исходного файла состоит из двух копий:

1. урезанных до длины **N**, и преобразованных к верхнему регистру строк
2. урезанных до длины **N** и преобразованных к нижнему регистру строк

Пустые строки исходного файла не обрабатываются и в выходные данные не попадают. Целое число **N > 1** вводит с клавиатуры пользователь.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные

`Let it be`
`234`
	`LET`
	`let`
	`234`

let it snow HSE —---- N = 3	LET let HSE hse

Библиотека классов (каждый класс разместить в отдельном файле):

- Базовый класс **StringFilter** описывает фильтры для строк, методы класса должны допускать переопределение в наследниках. Поле класса **length** содержит максимальное количество символов фильтруемой строки. Метод **Filter**() с параметром – строка. Метод возвращает строку, урезанную до **length** символов. Попытка конструировать объект с **length < 0** должна приводить к выбрасыванию исключения **ArgumentOutOfRangeException**.
- Класс **UpperStringFilter** наследник **StringFilter**. Конструктор **UpperStringFilter**() содержит один параметр типа **int** – количество символов фильтруемой строки. Поместить в класс **UpperStringFilter** переопределённый метод **Filter**() с параметром – строка. Метод возвращает строку, преобразованную к верхнему регистру, если количество символов в ней не превосходит допустимого значения и пустую строку в противном случае.
- Класс **LowerStringFilter** наследник **StringFilter**. Конструктор **LowerStringFilter**() содержит один параметр типа int – количество символов фильтруемой строки. Поместить в класс **LowerStringFilter** переопределённый метод **Filter**() с параметром **S** строка. Метод возвращает строку **S**, преобразованную к нижнему регистру, если количество символов в ней не превосходит допустимого значения и пустую строку в противном случае.

Вариант 2. Уточняем и форматируем

Разработать консольное приложение, которое позволяет пользователю фильтровать численные данные, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования объектов-фильтров, описываемых иерархией фильтров из библиотеки классов. В результате применения фильтров программа создаёт два файла, в одном содержатся отфильтрованные целые, во втором - все остальные. Целое число **N > 1** вводит с клавиатуры пользователь.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходной файл с целыми	Выходной файл со всеми данными
-56 34,5 77.01 1.0 —-----	-056 001	-056 34,500 77,010 001

N = 3		

Библиотека классов (каждый класс разместить в отдельном файле):

- Базовый класс **PreciseFilter** описывает фильтры для строк, методы класса должны допускать переопределение в наследниках. Целочисленное поле класса **precise** содержит точность числовых значений. Метод **Filter**() с вещественным параметром **X** и конструирует строковое представление числа X с тремя знаками после десятичного разделителя вне зависимости от того, целое ли оно или вещественное. Попытка конструировать объект-фильтр с **precise < 0** должна приводить к выбрасыванию исключения **ArgumentOutOfRangeException**.
- Унаследовать от **PreciseFilter** класс **IntPreciseFilter**. Конструктор **IntPreciseFilter()** содержит один параметр типа **int** – количество позиций, отводимых под представление при печати целого числа. Переопределить метод **Filter()**. Метод возвращает строку с отформатированным числом **X**, под представление которого отведено **precise** позиций. Метод должен проверять наличие у параметра ненулевой дробной части и не обрабатывать значения, в которых она отлична от 0, возвращая пустую строку.
- Унаследовать от **PreciseFilter** класс **DoublePreciseFilter**. Конструктор **DoublePreciseFilter()** содержит один целочисленный параметр – количество знаков после десятичного разделителя. Переопределить метод **Filter()**. Метод возвращает строку с отформатированным числом **X**, под дробную часть которого отведено **precise** позиций. Метод работает как целыми, так и с вещественными значениями, не проверяя.

Вариант 3. Фильтруем фигуры

Разработать консольное приложение, которое позволяет пользователю фильтровать фигуры, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах фигур из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию обо всех треугольниках с площадью большей **N**. Второй – обо всех прямоугольниках с условной плотностью большей **M**.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные 1	Выходные данные 2
Triangle 3 4 5 12 Triangle 6 8 10 48 Rectangle 5 5 100 Rectangle 1 1 100 –----	Triangle 3 4 5 12 6 12.0 Triangle 6 8 10 24 2.0	Rectangle 5 5 100 25 4.0 Rectangle 1 1 100 1 100.0

N = 1 M = 2		

Библиотека классов (каждый класс разместить в отдельном файле):

- Базовый класс **Figure**, описывающий геометрические фигуры и содержащий:
  - Защищенное целочисленное поле – число вершин **n**.
  - Свойство **N**, предоставляющее защищенный от изменения пользователей доступ к числу вершин.
  - Защищенное целочисленное поле – масса фигуры m
  - Свойство **M**, предоставляющее защищенный от изменения пользователей доступ к массе фигуры
  - Виртуальное свойство **Area**, предоставляющее вещественное значение площади фигуры. По умолчанию, площадь – 0.
  - Конструктор с двумя параметрами: число вершин фигуры n и масса фигуры **m** (по умолчанию масса равна 1). (если длины сторон или масса отрицательные - выбрасываем **ArgumentException**)
  - Свойство **conditionalDensity**, возвращающее условную плотность фигуры $\frac{m}{Area}$.
  - Переопределение метода **ToString()**. Он должен возвращать строку, описывающую тип фигуры (**GetType()**), условную плотность и её площадь.

- Класс треугольника **Triangle**, наследник **Figure**:
  - Закрытые целочисленные поля **a**, **b** и **c** - длины сторон
  - Переопределение свойства подсчета площади треугольника.
  - Конструктор с четыремя параметрами: длины сторон (проверяем существование треугольника: если длины сторон отрицательные - выбрасываем **ArgumentException**, если треугольника не существует - выбрасываем **ArgumentOutOfRangeException**) и масса фигуры (по умолчанию масса равна 1, если масса отрицательная выбрасываем **ArgumentException**).
  - Переопределение метода **ToString()**. Он должен возвращать строку, описывающую длины всех сторон фигуры, тип фигуры (**GetType()**), условную плотность и её площадь.

- Класс прямоугольника **Rectangle**, наследник **Figure**:
  - Закрытые целочисленные поля **a**, **b** – длины сторон
  - Переопределение свойства подсчета площади прямоугольника.
  - Конструктор с тремя параметрами: длины сторон a и b (если длины сторон отрицательные - выбрасываем ArgumentException) и масса фигуры (по умолчанию масса равна 1, если масса отрицательная выбрасываем **ArgumentException**).
  - Переопределение метода **ToString()**, Он должен возвращать строку, описывающую длины сторон прямоугольника (достаточно двух сторон), тип фигуры (**GetType()**), условную плотность и её площадь.

Вариант 4. Фильтруем правильные фигуры

Разработать консольное приложение, которое позволяет пользователю фильтровать правильные фигуры, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах правильных фигур из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию обо всех треугольниках с площадью большей **N**. Второй – обо всех квадратах с условной плотностью большей **M**.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные 1	Выходные данные 2

`EqTriangle 3 12`
`Square 5 100`
`Square 1 100`
`-----`
`N = 1`
`M = 2`
	`EqTriangle 3 12 0.43 27.71`	`Square 5 100 25 4.0`
		`Square 1 100 1 100.0`

Библиотека классов (каждый класс разместить в отдельном файле):

- Базовый класс **Figure**, описывающий геометрические фигуры и содержащий:
  - Защищенное целочисленное поле – число вершин **n**.
  - Свойство **N**, предоставляющее защищенный от изменения пользователей доступ к числу вершин.
  - Автореализуемое свойство **a** – длина стороны.
  - Защищенное целочисленное поле – масса фигуры **m**
  - Свойство **M**, предоставляющее защищенный от изменения пользователей доступ к массе фигуры
  - Виртуальное свойство **Area**, предоставляющее вещественное значение площади фигуры. По умолчанию, площадь – 0.
  - Конструктор с двумя параметрами: число вершин фигуры **n** и масса фигуры **m** (по умолчанию масса равна 1). (если длины сторон или масса отрицательные - выбрасываем **ArgumentException**)
  - Свойство **conditionalDensity**, возвращающее условную плотность фигуры $\frac{m}{Area}$.
  - Переопределение метода **ToString()**. Он должен возвращать строку, описывающую тип фигуры (**GetType()**), условную плотность и её площадь.

- Класс треугольника **EqTriangle**, наследник **Figure**:
  - Переопределение свойства подсчета площади треугольника.
  - Конструктор с двумя параметрами: длина стороны (проверяем существование треугольника: если длины сторон отрицательные - выбрасываем **ArgumentException**) и масса фигуры (по умолчанию масса равна 1, если масса отрицательная выбрасываем **ArgumentException**).

- Переопределение метода **ToString()**. Он должен возвращать строку, описывающую длину стороны фигуры, тип фигуры (**GetType()**), условную плотность и её площадь.

- Класс квадрата **Square**, наследник **Figure**:
  - Закрытые целочисленные поля **a**, **b** – длины сторон
  - Переопределение свойства подсчета площади квадрата.
  - Конструктор с двумя параметрами: длина стороны a (если длина стороны отрицательная - выбрасываем **ArgumentException**) и масса фигуры (по умолчанию масса равна 1, если масса отрицательная выбрасываем **ArgumentException**).
  - Переопределение метода **ToString()**, Он должен возвращать строку, описывающую длину стороны квадрата, тип фигуры (**GetType()**), условную плотность и её площадь.

Вариант 5. Разные устройства

Разработать консольное приложение, которое позволяет пользователю фильтровать девайсы, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах девайсов из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию обо всех компьютерах с числом видеокарт **N**. Второй – обо всех телефонах с диагональю больше **M**.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные 1	Выходные данные 2

`PC 312 Model1 2 3`
`PC 313 Model2 3 3`
`Phone 314 Model3 10`
`Phone 315 Model4 5.4`
`-----`
`N = 2`
`M = 2`
	`PC 312 Model1 2 3`	`Phone 314 Model3 10`
		`Phone 315 Model4 5.4`

Библиотека классов (каждый класс разместить в отдельном файле):

- Класс **Device** со следующими членами:
  - Целочисленное свойство **Id**, возвращающее или задающее **id** устройства (из диапазона [1; 1000], иначе выбрасывать **ArgumentOutOfRangeException**);
  - Строковое свойство **Model**, возвращающее или задающее модель устройства;
  - Виртуальное строковое свойство **Type**, возвращающее тип устройства;

    - Виртуальный метод **PrintInfo**, выводящий на экран значения свойств **Id**, **Model** и **Type**;
- Класс **PC**, наследующий от класса **Device** и имеющий следующий члены:
    - Целочисленное свойство **GraphicsCardsCount**, возвращает или задает количество графических карт (0+);
    - Целочисленное свойство **MonitorsCount**, возвращающее или задающее количество мониторов (0+);
    - Метод **HasProblems**, возвращающий признак наличия проблем (true, если количество карт или мониторов равно нулю, false в противном случае);
    - Переопределённое свойство **Type**, возвращающее строку “Computer”;
    - Переопределённый метод **PrintInfo**, выводящий на экран значения свойств базового класса, текущего и результат метода **HasProblems**;
    - конструктор, принимающий на вход **id**, **model**, **graphicsCardsCount** и **monitorsCount** и устанавливающий соответствующие свойства в текущем и базовом классе;
- Класс **Phone**, наследующийся от класса **Device** и имеющий следующие члены:
    - Свойство **ScreenDiagonal**, вещественное значение – диагональ экрана в дюймах из диапазона [3; 8];
    - Переопределённое свойство Type, возвращающее строку “Phone”;
    - Открытый конструктор, принимающий на вход id, model и screenDiagonal и устанавливающий соответствующие свойства в базовом классе;
    - Переопределённый метод **PrintInfo()**, выводящий на экран значения всех доступных свойств класса.

Для созданных классов переопределить метод **ToString()**, возвращающий строку для метода **PrintInfo()**.

Вариант 6. Машины

Разработать консольное приложение, которое позволяет пользователю фильтровать машины, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах машины из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию обо всех быстрых машинах с шагом движения большим **N**. Второй – обо всех медленных машинах с шагом движения большим **M**.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные 1	Выходные данные 2

`SpeedCar Model1 20`
`SpeedCar Model2 50`
`SlowCar Model3 10`
`SlowCar Model4 5`
`-----`
`N = 40`
`M = 7`
	`SpeedCar Model2 50`	`SlowCar Model3 10`

Библиотека классов (каждый класс разместить в отдельном файле):

- Класс Car, имеющий следующие члены:
  - Целочисленное свойство **X**, возвращающее/устанавливающее позицию машины относительно оси **X** (начиная с 1);
  - Строковое свойство **Name**, возвращающее название машины;
  - Виртуальное символьное свойство **Symbol**, возвращающее символ, представляющий объект на карте. По умолчанию – a.
  - Виртуальный метод **Step**, изменяющий позицию машины. По умолчанию меняет на 1.
  - Метод **PrintOnMap**, принимающий на вход целочисленный параметр width и выводящий на экран представление на карте текущего объекта, используя ‘-‘ для обозначения пустой позиции, Symbol для обозначения позиции текущего объекта. Пример: { Name = “Speedy”; Symbol = ‘0’; X = 4; width = 10 } -> “Speedy: [---0------]”;
- Класс **SpeedCar**, наследующийся от Car и имеющий следующие члены:
  - Переопределённое свойство **Symbol**, возвращающее символ ‘>’;
  - Переопределённый метод **Step**, увеличивающий значение свойства **X** на заданное число;
  - конструктор, принимающий на вход параметр name, step и устанавливающий значение соответствующего свойства в базовом классе;
- Класс **SlowCar**, наследующийся от **Car** и имеющий следующие члены:
  - Переопределённое свойство **Symbol**, возвращающее символ ‘o’;

- Переопределённый метод **Step**, увеличивающий значение свойства **X** на 0.5 от заданного числа **step**;
- конструктор, принимающий на вход параметр **name**, **step** и устанавливающий значение соответствующего свойства в базовом классе.

Вариант 7. Движение машин

Разработать консольное приложение, которое позволяет пользователю фильтровать машины, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах машин из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию о позициях всех быстрых машин после **K** шагов. Второй – обо о всех позициях медленных машин после **K** шагов. Начальная позиция всех машин – 0.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные 1	Выходные данные 2

`SpeedCar Model1 20`
`SpeedCar Model2 50`
`SlowCar Model3 10`
`SlowCar Model4 5`
`-----`
`K = 10`
	`SpeedCar Model1`
	`200`
	`SpeedCar Model2`
	`500`
		`SlowCar Model3 100`
		`SlowCar Model4 50`

Библиотека классов (каждый класс разместить в отдельном файле):

- Класс **Car**, имеющий следующие члены:
  - Целочисленное свойство **X**, возвращающее/устанавливающее позицию машины на оси **X** (начиная с 1);
  - Строковое свойство Name, возвращающее название машины;
  - Виртуальное символьное свойство **Symbol**, возвращающее символ, представляющий объект на карте. По умолчанию – a.
  - Виртуальный метод **Step**, изменяющий позицию (координату) машины. По умолчанию меняет на 1.
  - Метод **PrintOnMap**, принимающий на вход целочисленный параметр width и выводящий на экран представление на карте текущего объекта, используя ‘-‘ для обозначения пустой позиции, **Symbol** для обозначения позиции текущего объекта. Пример: { Name = “Speedy”; Symbol = ‘0’; X = 4; width = 10 } -> “Speedy: [---0------]”;
- Класс **SpeedCar**, наследующийся от **Car** и имеющий следующие члены:
  - Переопределённое свойство **Symbol**, возвращающее символ ‘>’;

    - Переопределённый метод **Step**, увеличивающий значение свойства **X** на заданное число;
    - конструктор, принимающий на вход параметр **name**, **step** и устанавливающий значение соответствующего свойства в базовом классе;
- Класс **SlowCar**, наследующийся от **Car** и имеющий следующие члены:
    - Переопределённое свойство **Symbol**, возвращающее символ ‘o’;
    - Переопределённый метод **Step**, увеличивающий значение свойства **X** на 0.5 от заданного числа **step**;
    - конструктор, принимающий на вход параметр **name**, **step** и устанавливающий значение соответствующего свойства в базовом классе.

Вариант 8. Атлеты

Разработать консольное приложение, которое позволяет пользователю фильтровать атлетов, расположенных в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах атлетов из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию обо всех бегунах, чья скорость больше скорости Усэйна Болта. Второй – обо прыгунах с длиной прыжка большей мирового рекорда. Отсортировать по убыванию числовых значений.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные 1	Выходные данные 2

`Runner Name1 100`
`Runner Name2 10`
`Jumper Name3 10`
`Jumper Name4 5`
	`Runner Name2 10`	`Jumper Name3 10`

Библиотека классов (каждый класс разместить в отдельном файле):

- Класс **Athlete**, который содержит:
    - поле string name – имя спортсмена
    - конструктор **Athlete (string name, string country)** – задает значение соответствующим полям объекта класса
    - виртуальный метод **bool IsOnDoping()** – возвращает **true**, если спортсмена уличили в допинге, и **false** в противном случае
- Класс **Runner**, наследник класса **Athlete**. Содержит:
    - константа **int distance** – длина забега легкоатлета, равная 100 м
    - поле **double time** – время, за которое он пробежал стометровку

    - свойство **double Speed** с доступом только для чтения – скорость, с которой спортсмен преодолел эту дистанцию.
    - конструктор класса **Runner (string name, double time)** – присваивает соответствующим полям объекта класса передаваемые значения
    - переопределенный метод **IsOnDoping()** – возвращает true, если скорость спортсмена больше, чем средняя скорость Усэйна Болта, 12.42 м/с
    - переопределенный метод **ToString()** – возвращает строку, содержащую значения всех полей.
- Класс Jumper, наследник класса **Athlete**. Содержит:
    - поле **double distance** – длина прыжка, на которую прыгнул атлет
    - свойство **double Distance** с доступом только для чтения
    - конструктор класса **Jumper (string name, double distance)** – присваивает соответствующим полям объекта класса передаваемые значения
    - переопределенный метод **IsOnDoping()** – возвращает true, если длина прыжка больше, чем мировой рекорд, 8.90м
    - переопределенный метод **ToString()** – возвращает строку, содержащую значения всех полей

Вариант 9. Вкусная фильтрация

Разработать консольное приложение, которое позволяет пользователю фильтровать единицы товара (блюд), расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах единиц товаров из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию обо всех блюдах со стоимостью одного грамма большего **N**. Второй – обо всех напитках со стоимостью 1 мл большего **M**.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные 1	Выходные данные 2

`Food Name1 100 100`
`Food Name2 50 1000`
`Drink Name3 10 500`
`Drink Name4 50 1000`
`-----`
`N = 10`
`M = 10`
	`Food Name2 50 1000`	`Drink Name3 10 500`
		`Drink Name4 50 1000`

Библиотека классов (каждый класс разместить в отдельном файле):

- Класс **Item** - наименование товара. Содержит:
    - поле **string name** – наименование товара

    - поле **double price** – цена данного товара с точностью до двух знаков после запятой (т.е. с точностью до копеек)
    - свойство **double Price** для доступа к содержимому поля price, если содержимое поля price отрицательно – возвращать **ArgumentException**;
    - конструктор **Item(string name, double price)** – заполняет поля объекта класса значениями
- Класс **Food**, наследник класса **Item**. Содержит:
    - Автореализуемое свойство **int Weigh**t - вес данного пищевого продукта в граммах с доступом только для чтения.
    - Конструктор **Food(string name, double price, int weight)** – заполняет поля объекта класса значениями
    - Переопределенный метод **ToString()** – возвращает строку с полной информацию о продукте, т.е. в ней присутствуют значения всех полей
- Класс **Drink**, наследник класса **Item**. Содержит:
    - Автореализуемое свойство **int Volume** - объем данного напитка в мл с доступом только для чтения.
    - Конструктор **Drink(string name, double price, int volume)** – заполняет поля объекта класса значениями
    - Переопределенный метод **ToString()** – возвращает строку с полной информацию о напитке, т.е. в ней присутствуют значения всех полей

Вариант 10. Строения

Разработать консольное приложение, которое позволяет пользователю фильтровать правильные объекты собственности (загородные дома), расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах объектов собственности из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию обо всех апартаментах с налогом большим **N**. Второй – обо всех загородных домах с налогом большим **M**.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные 1	Выходные данные 2

`Apartment Name1 10 0.1 10`
`CountryHouse Name2 10 0.1 10`
`-----`
`N = 0.1`
`M = 0.2`
	`Apartment Name1 10 0.1 10`	`CountryHouse Name2 10 0.1 10`
		`.0`

Библиотека классов (каждый класс разместить в отдельном файле):

- Класс **Property**:

    - Свойство объекта: строка **Name** – англоязычное наименование налога. При недопустимых входных данных – генерировать исключение **ArgumentException** с сообщением об ошибке “Допускается последовательность только из букв латинского алфавита”).
    - Свойство объекта: вещественное число **Increase** – коэффициент увеличения налога (не менее 1).
    - Виртуальное свойство объекта: вещественное число **TaxRate** – процентная ставка налога в долях (не может превышать 0.4 для апартаментов и 1 для загородного дома).
    - Свойство объекта **Tenure**: целое число, показывающее количество месяцев владения объектом.
    - Конструктор с четырьмя параметрами – значением коэффициента для свойства **Increase**, **TaxRate** и **Tenure**, и значением имени для свойства Name. Не забывайте про обработку некорректных данных.
    - Виртуальный индексатор, принимающий в качестве входного параметра целочисленную стоимость имущества, и возвращающий вещественное значение налога, которое необходимо уплатить. В случае если параметр не является корректным для данной функции, необходимо генерировать исключение **ArgumentException** с соответствующим сообщением об ошибке.
    - Переопределить метод **ToString()**, чтобы он возвращал строку формата “Имущество: <Name>. Коэффициент увеличения налога: <Increase>, процентная ставка налога: <TaxRate>, число месяцев владения объектом: <Tenure>.”
- Члены класса **Apartment**, наследуемого от Property:
    - Переопределить метод **ToString()**, чтобы он возвращал строку со значением всех свойств базового класса, включая подстроку о виде имущества формата “Вид: Apartment.”
    - Налог для апартаментов вычисляется следующим образом: $Increase * TaxRate * ln(\frac{x}{Tenure})$, где x – стоимость апартаментов.
- Члены класса **CountryHouse**, наследуемого от Property:
    - Переопределить метод **ToString()**, чтобы он возвращал строку со значением всех свойств базового класса, включая подстроку о виде функции формата “Вид: CountryHouse.”
    - Налог для загородного дома вычисляется следующим образом: $Increase * TaxRate * \frac{x}{Tenure}$, где x – стоимость апартаментов.
- Самостоятельно определите дополнительные члены класса **Property**, **Apartment**, **CountryHouse** необходимые для решения задачи и корректного функционирования приложения.

Вариант 11. Транспортные средства

Разработать консольное приложение, которое позволяет пользователю фильтровать правильные данные о транспорте, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах правильных единиц транспорта из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию обо всех мотоциклах с расходом меньшим **N** на 1000 км пути. Второй – обо всех автомобилях с расходом меньшим **M** на 1000 км пути.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные 1	Выходные данные 2
Motorcycle Name1 10 10 Car Name2 100 20 ----- N = 200 M = 400	Motorcycle Name1 10 10	Car Name2 100 20

Библиотека классов (каждый класс разместить в отдельном файле):

- Члены базового класса **Transport**:
    - Свойство объекта: строка **Name** – англоязычное название транспортного средства. При недопустимых входных данных – генерировать исключение **ArgumentException** с сообщением об ошибке “Допускается последовательность только из букв латинского алфавита”).
    - Виртуальное свойство объекта: целое число **FuelTankCapacity**– объем бензобака (не может быть меньше 10 л для автомобиля и 5 л для мотоцикла).
    - Абстрактное свойство объекта: вещественное число FuelMileage – расход бензина на 100 км (не может быть больше 30 л для автомобиля и 15 л для мотоцикла).
    - Конструктор с четырьмя параметрами – значениями для инициализации свойств **FuelTankCapacity** и **FuelMileage**, и значением имени для свойства Name. Не забывайте про обработку некорректных данных.
    - Виртуальный индексатор, принимающий в качестве входного параметра вещественный аргумент – число километров, которые необходимо проехать, и возвращающий целое количество дозаправок, необходимое, чтобы проехать заданное расстояние. Считайте, что изначально в бензобаке транспортного средства бензин отсутствует. В случае если параметр не является корректным, необходимо генерировать исключение ArgumentException с соответствующим сообщением об ошибке.
    - Переопределить метод **ToString()**, чтобы он возвращал строку с информацией об объекте.

- Члены класса **Motorcycle**, представляющего двухколесные мотоциклы, наследуемого от **Transport**:
    - Переопределить метод **ToString()**, чтобы он возвращал строку со значением всех свойств класса.
- Члены класса **Car**, представляющего четырехколесные мотоциклы, наследуемого от **Transport**:
    - Переопределить метод **ToString()**, чтобы он возвращал строку со значением всех свойств класса.
    - Автомобиль может давать сбои, увеличивающие расход бензина. Автомобиль дает сбой и увеличивает расход бензина на коэффициент $ln(x)$, где x – число километров, которые необходимо проехать. При этом, для остальных поездок на этом же автомобиле расход бензина должен остаться прежним.
- Самостоятельно определите дополнительные члены класса **Transport**, **Motorcycle**, **Car** необходимые для решения задачи и корректного функционирования приложения.

Вариант 12. Правильные фигуры

Разработать консольное приложение, которое позволяет пользователю фильтровать правильные фигуры, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах правильных фигур из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию обо всех правильных треугольниках с радиусом описанной окружности большим **N**. Второй – обо всех квадратах с радиусом описанной окружности большим **M**.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные 1	Выходные данные 2
EqTriangle 0 0 10 Square 0 0 20 ----- N = 200 M = 400	EqTriangle 0 0 10	Square 0 0 20

Библиотека классов (каждый класс разместить в отдельном файле):

- Класс **Point**, содержащий:
    - Значения двух координат некоторой точки на плоскости: x и y.
    - Конструктор с двумя параметрами – значениями абсциссы и ординаты точки.

- Переопределенный метод **ToString()** для вывода координаты точки в удобном для пользователя виде.

- Класс **Figure**, реализующий ПРАВИЛЬНЫЕ фигуры и содержащий:
  - Поле **points** типа **Point[]** - одномерный массив, содержащий координаты всех вершин фигуры на плоскости.
  - Свойство **Length**, защищенное от изменений пользователем. Содержит длину стороны правильной фигуры.
  - Виртуальное свойство Area, предоставляющее значение площади фигуры
  - Конструктор класса с одним параметром, задающим массив координат. Hint: Массив тип ссылочный, нужно копировать значения, а не присваивать ссылку!
  - Виртуальный метод для нахождения длины радиуса описанной вокруг фигуры окружности.
  - Переопределить метод ToString(). Он должен возвращать строку, описывающую все **X** и **Y** координаты фигуры, длину стороны фигуры, а также тип фигуры (с помощью GetType()) и её площадь.
- Класс **EqTriangle**, реализующий ПРАВИЛЬНЫЙ треугольник, наследник Figure, содержащий:

  Переопределение свойства Area, подсчитывающего площадь фигуры

$$S = \frac{\sqrt{3}a^2}{4}$$

  - Переопределение метода Radius для подсчета радиуса описанной окружности вокруг правильного треугольника.

$$R = \frac{2a}{\sqrt{3}}$$

  - Метод, принимающий на вход два параметра: координату левой нижней вершины треугольника и длину его стороны. Метод возвращает массив, состоящий из переданной вершины и всех остальных вершин треугольника, координаты которых нужно посчитать самостоятельно. Высота равностороннего треугольника

$$h = \frac{\sqrt{3}a}{2}$$

  - Конструктор, принимающий на вход два параметра: координату левой нижней вершины треугольника и длину его стороны. Если длина стороны отрицательная, то выбрасываем соответствующее исключение.
- Класс Square, реализующий ПРАВИЛЬНЫЙ четырехугольник, наследник Figure:

  Переопределение свойства Area, подсчитывающего площадь фигуры

$$S = a^2$$

Переопределение метода Radius для подсчета радиуса описанной окружности вокруг квадрата.

$$R = \frac{a}{\sqrt{2}}$$

  - Метод, принимающий на вход два параметра: координату левой нижней вершины квадрата и длину его стороны. Метод возвращает массив, состоящий из переданной вершины и всех остальных вершин квадрата, координаты которых нужно посчитать самостоятельно.

- Конструктор, принимающий на вход два параметра: координату левой нижней вершины квадрата и длину его стороны. Если длина стороны отрицательная, то выбрасываем соответствующее исключение.

Вариант 13. Музыкальные инструменты

Разработать консольное приложение, которое позволяет пользователю фильтровать правильные инструменты, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах правильных инструментов из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию обо всех барабанах с результатом игры большим **N**. Второй – обо гитарах с результатом игры большим **M**.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные 1	Выходные данные 2
Drum 10 10 Guitar 5 5 ----- N = 2 M = 4	Drum 10 10	Guitar 5 5

Библиотека классов (каждый класс разместить в отдельном файле):

- Класс **Instrument**, который содержит:
  - поле **double volum**e – громкость инструмента
  - свойство к полю **volume** (только для чтения)
  - поле **int mastery** – мастерство музыканта при игре на инструменте
  - свойство к полю **mastery** (только для чтения)
  - виртуальный метод **void Play()** – метод возвращает результат игры – восторженность зрителя, равную произведению громкости на мастерство
- Класс **Drum**, наследник класса **Instrument**. Содержит:
  - конструктор класса **Drum** – задает громкость барабанов и присваивает уровень мастерства музыканта
  - переопределенный метод **Play()** – метод возвращает результат игры – восторженность зрителя, равную произведению удвоенной громкости на мастерство
  - переопределенный метод **ToString()** – возвращает строку, содержащую значения всех полей.
- Класс **Guitar**, наследник класса **Instrument**. Содержит:
  - конструктор класса **Guitar** – задает громкость гитары в интервале и присваивает уровень мастерства музыканта

- переопределенный метод **Play()** – метод возвращает результат игры – восторженность зрителя, равную произведению удвоенной громкости на утроенное мастерство
- переопределенный метод **ToString()** – возвращает строку, содержащую значения всех полей.

Вариант 14. Телефоны

Разработать консольное приложение, которое позволяет пользователю фильтровать правильные телефоны, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах правильных телефоны из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию обо всех мобильных телефонах с четной стоимостью. Второй – обо всех смартфонах с четной стоимостью. Отсортировать по убыванию стоимости.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные	Выходные данные
MobilePhone Name1 10 SmartPhone Name2 22	MobilePhone Name1 10	SmartPhone Name2 22

Библиотека классов (каждый класс разместить в отдельном файле):

- Члены базового класса **Phone** (стационарный телефон):
  - Свойство объекта: строка **Name**.
  - Свойство объекта: неотрицательная целочисленная стоимость телефона **Price**. Стоимость стационарного телефона не может быть менее 100. При недопустимых входных данных – генерировать исключение типа **ArgumentOutOfRangeException** с сообщением об ошибке.
  - Конструктор с двумя параметрами – значениями для свойств.
  - Переопределить метод **ToString()**, чтобы он возвращал строку с описанием телефона.
  - Переопределить метод **Equals**. Будем считать, что два объекта являются равными, если совпадают все значения их свойств объектов, а также тип объекта (используйте **GetType()**).
- Члены класса **MobilePhone** (мобильный телефон), наследуемого от **Phone**:
  - Переопределенная стоимость телефона **Price**. Стоимость мобильного телефона не может быть менее 1000. При недопустимых входных данных

– генерировать исключение типа **ArgumentOutOfRangeException** с сообщением об ошибке.
    - Переопределить метод **ToString()**, чтобы он возвращал подстроку “Mobile”, сконкатенированную со строкой базового класса.
- Члены класса **SmartPhone** (смартфон), наследуемого от **Phone**:
    - Переопределенная стоимость телефона Price. Стоимость смартфона не может быть менее 10000. При недопустимых входных данных – генерировать исключение типа **ArgumentOutOfRangeException** с сообщением об ошибке.

Вариант 15. Правильные фигуры

Разработать консольное приложение, которое позволяет пользователю фильтровать правильные фигуры, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах правильных фигур из библиотеки классов. В результате применения фильтрации программа создаёт два новых файла. Первый содержит информацию обо всех правильных треугольниках с площадью меньшей **N.** Второй – обо всех квадратах с площадью большей **M**. Отсортировать по убыванию площади фигур.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные	Выходные данные
EqTriangle 0 0 10 Square 0 0 20 ----- N = 20 M = 40	EqTriangle 0 0 10	Square 0 0 20

Библиотека классов (каждый класс разместить в отдельном файле):

- Класс **Point**, содержащий:
    - Значения двух координат некоторой точки на плоскости: **x** и **y**.
    - Конструктор с двумя параметрами – значениями абсциссы и ординаты точки.
    - Переопределенный метод **ToString()** для вывода координаты точки в удобном для пользователя виде.
- Класс **Figure**, реализующий ПРАВИЛЬНЫЕ фигуры и содержащий:
    - Поле **points** типа **Point[]** - одномерный массив, содержащий координаты всех вершин фигуры на плоскости.
    - Свойство **Length**, защищенное от изменений пользователем. Содержит длину стороны правильной фигуры.

- Виртуальное свойство **Area**, предоставляющее значение площади фигуры
- Конструктор класса с одним параметром, задающим массив координат. Hint: Массив тип ссылочный, нужно копировать значения, а не присваивать ссылку!
- Виртуальный метод для нахождения длины радиуса описанной вокруг фигуры окружности.
- Переопределить метод **ToString()**. Он должен возвращать строку, описывающую все **X** и **Y** координаты фигуры, длину стороны фигуры, а также тип фигуры (с помощью **GetType()**) и её площадь.

- Класс **EqTriangle**, реализующий ПРАВИЛЬНЫЙ треугольник, наследник Figure, содержащий:

  Переопределение свойства Area, подсчитывающего площадь фигуры

$$S = \frac{\sqrt{3}a^2}{4}$$

  - Переопределение метода Radius для подсчета радиуса описанной окружности вокруг правильного треугольника.

$$R = \frac{2a}{\sqrt{3}}$$

  - Метод, принимающий на вход два параметра: координату левой нижней вершины треугольника и длину его стороны. Метод возвращает массив, состоящий из переданной вершины и всех остальных вершин треугольника, координаты которых нужно посчитать самостоятельно. Высота равностороннего треугольника

$$h = \frac{\sqrt{3}a}{2}$$

  - Конструктор, принимающий на вход два параметра: координату левой нижней вершины треугольника и длину его стороны. Если длина стороны отрицательная, то выбрасываем соответствующее исключение.
- Класс Square, реализующий ПРАВИЛЬНЫЙ четырехугольник, наследник Figure:

  Переопределение свойства Area, подсчитывающего площадь фигуры

$$S = a^2$$

Переопределение метода **Radius** для подсчета радиуса описанной окружности вокруг квадрата.

$$R = \frac{a}{\sqrt{2}}$$

  - Метод, принимающий на вход два параметра: координату левой нижней вершины квадрата и длину его стороны. Метод возвращает массив, состоящий из переданной вершины и всех остальных вершин квадрата, координаты которых нужно посчитать самостоятельно.
  - Конструктор, принимающий на вход два параметра: координату левой нижней вершины квадрата и длину его стороны. Если длина стороны отрицательная, то выбрасываем соответствующее исключение.

Вариант 16. Правильные фигуры

Разработать консольное приложение, которое позволяет пользователю фильтровать правильные фигуры, расположенные в текстовых файлах. Имя файла (не путь) с данными пользователь подаёт на вход программе.

Фильтрация осуществляется за счёт использования методов, описываемых в классах правильных фигур из библиотеки классов. В результате применения фильтрации программа создаёт новый файл. Файл содержит информацию обо всех фигурах, лежащих в первой координатной четверти. Фигуры отсортированы по убыванию радиуса описанной окружности.

Пример содержимого файлов в результате обработки строк:

Исходные данные	Выходные данные	Выходные данные
EqTriangle 1 1 10 Square 2 2 20 ----- N = 200 M = 400	EqTriangle 1 1 10	Square 2 2 20

Библиотека классов (каждый класс разместить в отдельном файле):

- Класс **Point**, содержащий:
  - Значения двух координат некоторой точки на плоскости: **x** и **y**.
  - Конструктор с двумя параметрами – значениями абсциссы и ординаты точки.
  - Переопределенный метод **ToString()** для вывода координаты точки в удобном для пользователя виде.
- Класс **Figure**, реализующий ПРАВИЛЬНЫЕ фигуры и содержащий:
  - Поле **points** типа **Point[]** - одномерный массив, содержащий координаты всех вершин фигуры на плоскости.
  - Свойство **Length**, защищенное от изменений пользователем. Содержит длину стороны правильной фигуры.
  - Виртуальное свойство **Area**, предоставляющее значение площади фигуры
  - Конструктор класса с одним параметром, задающим массив координат. **Hint**: Массив тип ссылочный, нужно копировать значения, а не присваивать ссылку!
  - Виртуальный метод для нахождения длины радиуса описанной вокруг фигуры окружности.
  - Переопределить метод **ToString()**. Он должен возвращать строку, описывающую все **X** и **Y** координаты фигуры, длину стороны фигуры, а также тип фигуры (с помощью GetType()) и её площадь.
- Класс **EqTriangle**, реализующий ПРАВИЛЬНЫЙ треугольник, наследник Figure, содержащий:

  Переопределение свойства **Area**, подсчитывающего площадь фигуры

$$S = \frac{\sqrt{3}a^2}{4}$$

- Переопределение метода Radius для подсчета радиуса описанной окружности вокруг правильного треугольника.

$$R = \frac{2a}{\sqrt{3}}$$

- Метод, принимающий на вход два параметра: координату левой нижней вершины треугольника и длину его стороны. Метод возвращает массив, состоящий из переданной вершины и всех остальных вершин треугольника, координаты которых нужно посчитать самостоятельно. Высота равностороннего треугольника

$$h = \frac{\sqrt{3}a}{2}$$

- Конструктор, принимающий на вход два параметра: координату левой нижней вершины треугольника и длину его стороны. Если длина стороны отрицательная, то выбрасываем соответствующее исключение.

- Класс **Square**, реализующий ПРАВИЛЬНЫЙ четырехугольник, наследник Figure:

Переопределение свойства Area, подсчитывающего площадь фигуры

$$S = a^2$$

Переопределение метода Radius для подсчета радиуса описанной окружности вокруг квадрата.

$$R = \frac{a}{\sqrt{2}}$$

- Метод, принимающий на вход два параметра: координату левой нижней вершины квадрата и длину его стороны. Метод возвращает массив, состоящий из переданной вершины и всех остальных вершин квадрата, координаты которых нужно посчитать самостоятельно.
- Конструктор, принимающий на вход два параметра: координату левой нижней вершины квадрата и длину его стороны. Если длина стороны отрицательная, то выбрасываем соответствующее исключение.