**Тестовое задание для курса по java**

Описание задачи: у*тилита фильтрации содержимого файлов.*

При запуске утилиты в командной строке подается несколько файлов, содержащих вперемешку целые числа, строки и вещественные числа. В качестве разделителя используется перевод строки. Строки из файлов читаются по очереди в соответствии с их перечислением в командной строке.

Задача утилиты записать разные типы данных в разные файлы. Целые числа в один выходной файл, вещественные в другой, строки в третий, значения записываются по порядку их прочтения из файла. По умолчанию файлы с результатами располагаются в текущей папке с именами *integers.txt, floats.txt, strings.txt*. Дополнительно с помощью опции

*-o* нужно уметь задавать путь для результатов. Опция -*p* задает префикс имен выходных файлов. Например *-o /some/path -p result\_* задают вывод в файлы

/some/path/result\_integers.txt, /some/path/result\_strings.txt и т.д. По умолчанию файлы

результатов перезаписываются. Если добавлена опция *-a*, то новые результаты добавляются в файлы с предыдущими. Файлы с результатами должны создаваться по мере необходимости. Если какого-то типа данных во входящих файлах нет, то и создавать исходящий файл, который будет заведомо пустым, не нужно.

В процессе фильтрации данных необходимо собирать статистику по каждому типу данных. Статистика должна поддерживаться двух видов: краткая или полная. Выбор статистики производится опциями *-s* или *-f* соответственно. Краткая статистика содержит только количество элементов записанных в исходящие файлы. Полная статистика для чисел дополнительно содержит минимальное и максимальное значения, сумма и среднее. Полная статистика для строк, помимо их количества, содержит также размер самой

короткой строки и самой длинной. Статистику по каждому типу отфильтрованных данных утилита должна вывести в консоль. Все возможные виды ошибок должны быть обработаны. Программа не должна «падать». Если после ошибки продолжить выполнение невозможно, программа должна сообщить об этом пользователю с указанием причины неудачи. Частичная обработка при наличии ошибок более предпочтительна чем остановка программы. Код программы должен быть читаемым и хорошо структурированным.

Для реализации необходимо использовать язык программирования Java, допустимо использовать стандартные системы сборки проекта (Maven, Gradle). Если система сборки не присутствует, то следует приложить инструкцию по сборке проекта.

Решение принимается в виде исходного кода всего проекта в **zip** архиве (папка проекта и все необходимые файлы **без результатов компиляции и сборки** проекта). К решению должна прилагаться инструкция по запуску. В ней можно отображать особенности реализации, не уточненные в задании. В частности, в инструкции необходимо указывать:

• версию Java;

• при использовании системы сборки – указать систему сборки и ее версию;

• при использовании сторонних библиотек указать их название и версию, приложить ссылки на такие библиотеки (можно в формате зависимостей системы сборки)

**Пример ожидаемого решения**

# **Пример входного файла in1.txt**

Lorem ipsum dolor sit amet 45

Пример

3.1415

consectetur adipiscing

-0.001

тестовое задание

100500

# **Пример входного файла in2.txt**

Нормальная форма числа с плавающей запятой

1.528535047E-25

Long

123456789

# **Пример запуска утилиты**

java -jar util.jar -s -a -p sample- in1.txt in2.txt

# **sample-integers.txt**

45

100500

123456789

# **sample-floats.txt**

3.1415

-0.001

1.528535047E-25

# **sample-strings.txt**

Lorem ipsum dolor sit amet

Пример

consectetur adipiscing

тестовое задание

Нормальная форма числа с плавающей запятой

Long

# **Пример краткой статистики**

sample-integers.txt short statistic: elements = 3

# **Пример полной статистики**

sample-integers.txt full statistic: elements = 3; min = 45; max = 123456789; sum = 123557334, average = 41185778;