# Практическая работа 1

**Задание 1**

Был создан массив из 10000 случайных чисел.

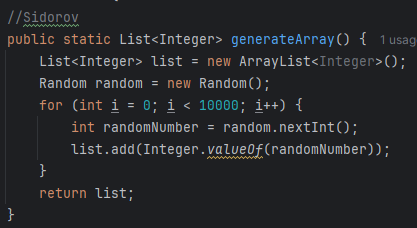


Рисунок 1 – Функция создания списка из 10000 элементов

В созданном массиве был начат поиск наименьшего элемента последовательно, многопоточно и с использованием ForkJoin.

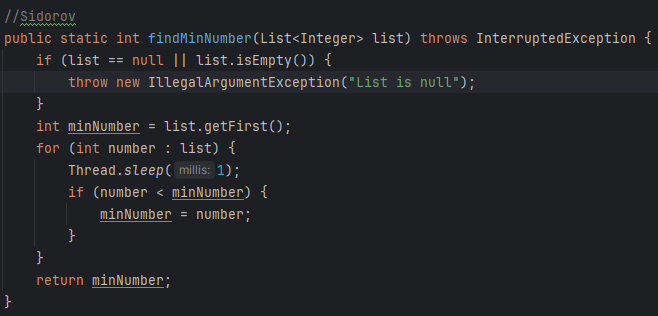


Рисунок 2 – Последовательный способ

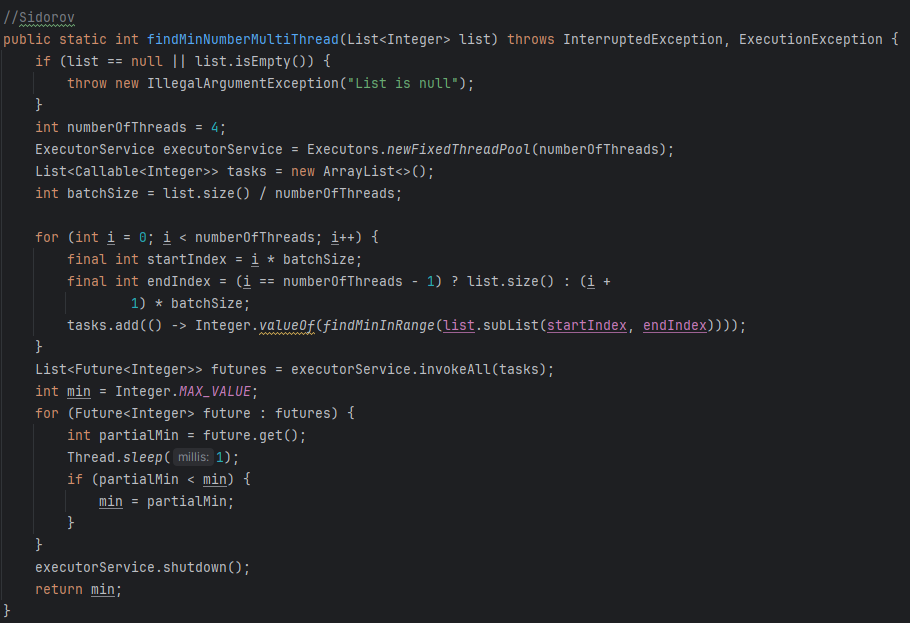


Рисунок 3 – Многопоточный способ

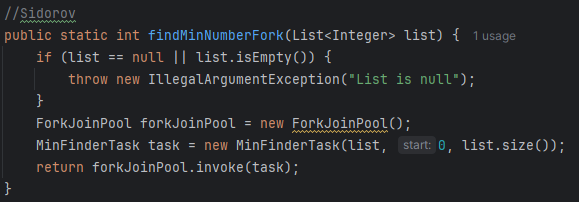


Рисунок 4 – Fork

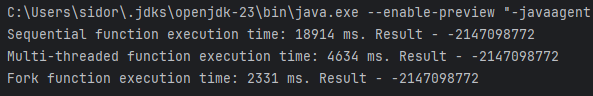


Рисунок 5 – Результат

Последовательный поиск наименьшего элемента оказался самым медленным.

**Задание 2**

Программа запрашивает у пользователя на вход число. Программа имитирует обработку запроса пользователя в виде задержки от 1 до 5 секунд выводит результат: число, возведенное в квадрат. В момент выполнения запроса пользователь имеет возможность отправить новый запрос. Реализовать с использованием Future.

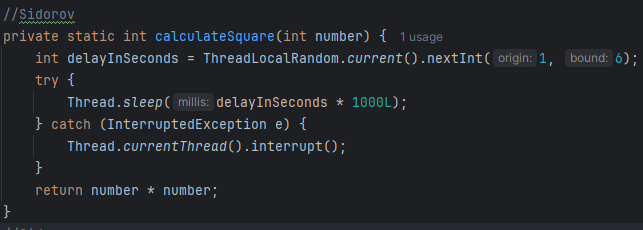


Рисунок 6 – Функция, высчитывающая квадрат

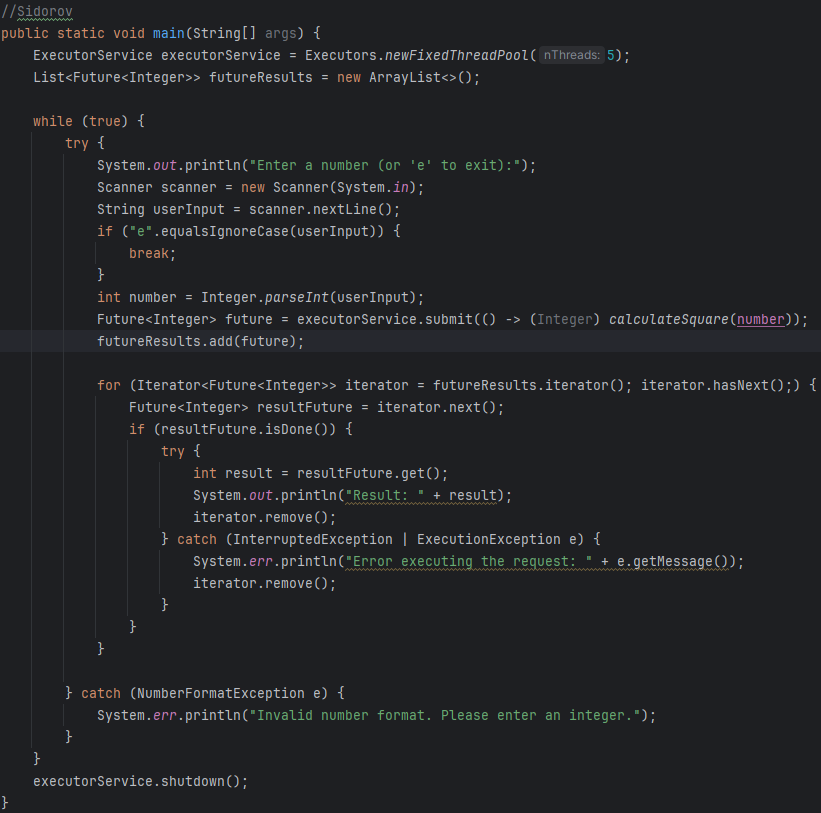


Рисунок 7 – Функция, обрабатывающая значения пользователя

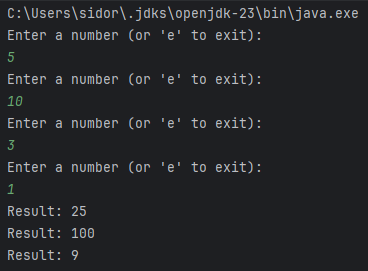


Рисунок 8 – Результат

В момент выполнения запроса пользователь имеет возможность отправить новый запрос.

**Задание 3**

Класс Файл имеет следующие характеристики:

1. Тип файла (например XML, JSON, XLS).

2. Размер файла — целочисленное значение от 10 до 100.

Генератор файлов — генерирует файлы с задержкой от 100 до 1000 мс. Очередь — получает файлы из генератора. Вместимость очереди — 5 файлов.

Обработчик файлов — получает файл из очереди. Каждый обработчик имеет параметр — тип файла, который он может обработать. Время обработки файла: «Размер файла\*7мс».

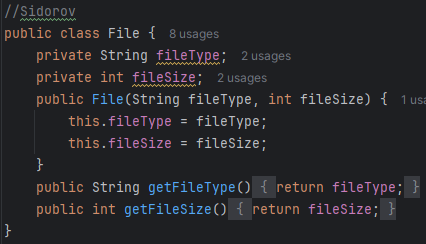


Рисунок 9 – Класс File

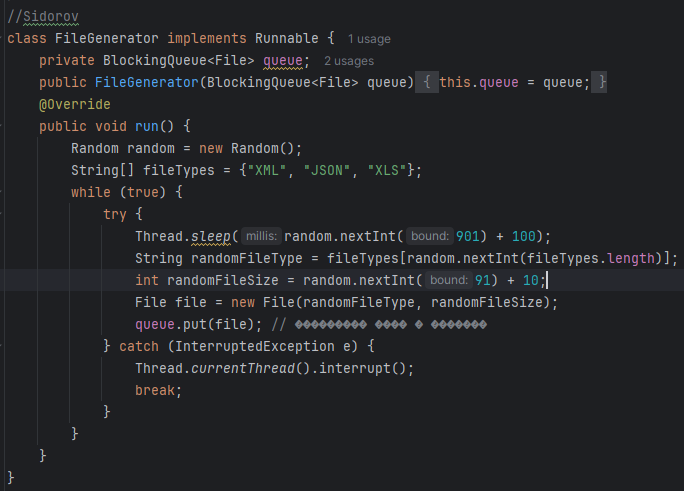


Рисунок 10 – Класс FileGenerator

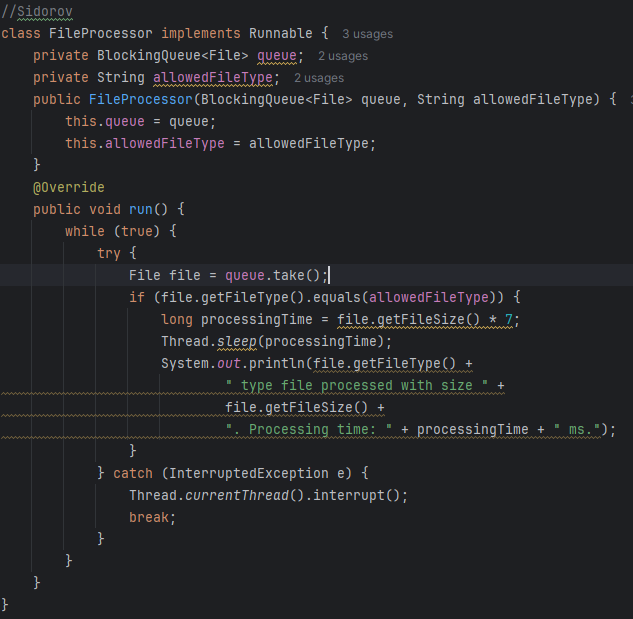


Рисунок 11 – Класс FileProcessor

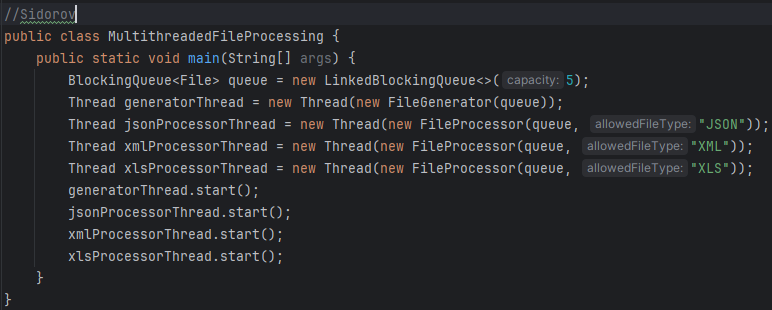


Рисунок 12 – Запуск потоков

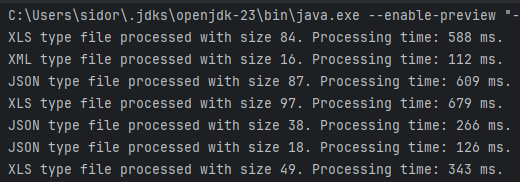


Рисунок 13 – Результат

В результате, обеспечена потокобезопасность, работа генератора не зависит от работы обработчиков, потоки не блокируют другие потоки при отсутствии задач

**Практическая работа №2**

Задание 1.

Для выполнения задания были созданы функции отвечающие за создание и чтение файла. Код представлен на рисунке 14.

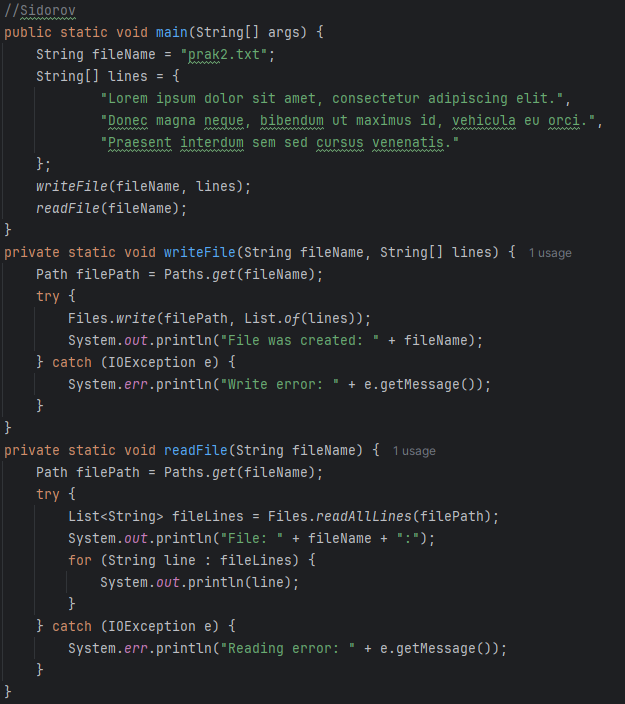


Рисунок 14 – Функции осуществляющие операции с файлом

Результат выполнения представлен на рисунке 15.

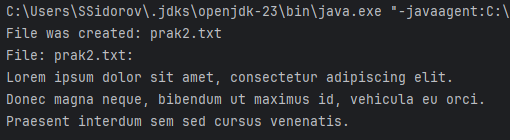


Рисунок 15 – результат вывода содержимого файла

Задание 2.

Для выполнения необходимых замеров и самих операций был написан метод main, представленный на рисунке 16.

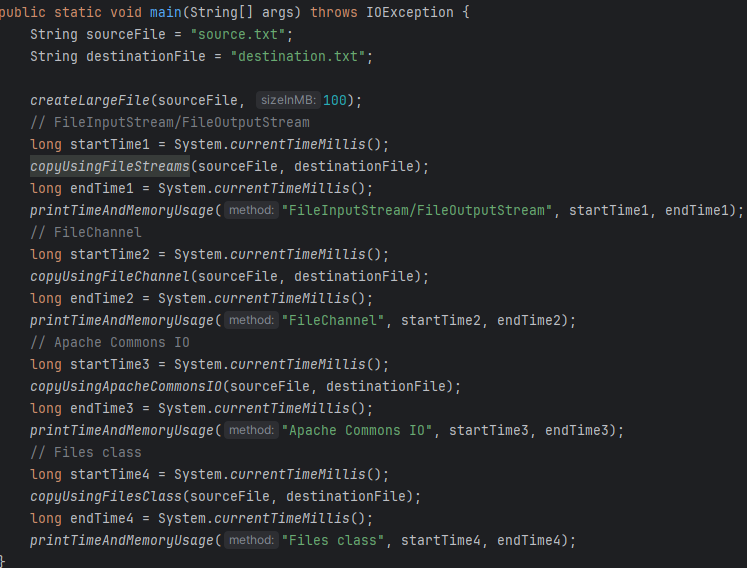


Рисунок 16 – Основной метод программы

В данном методе вызываются методы, отвечающие за выполнение копирования. Код методов представлен на рисунке 17.

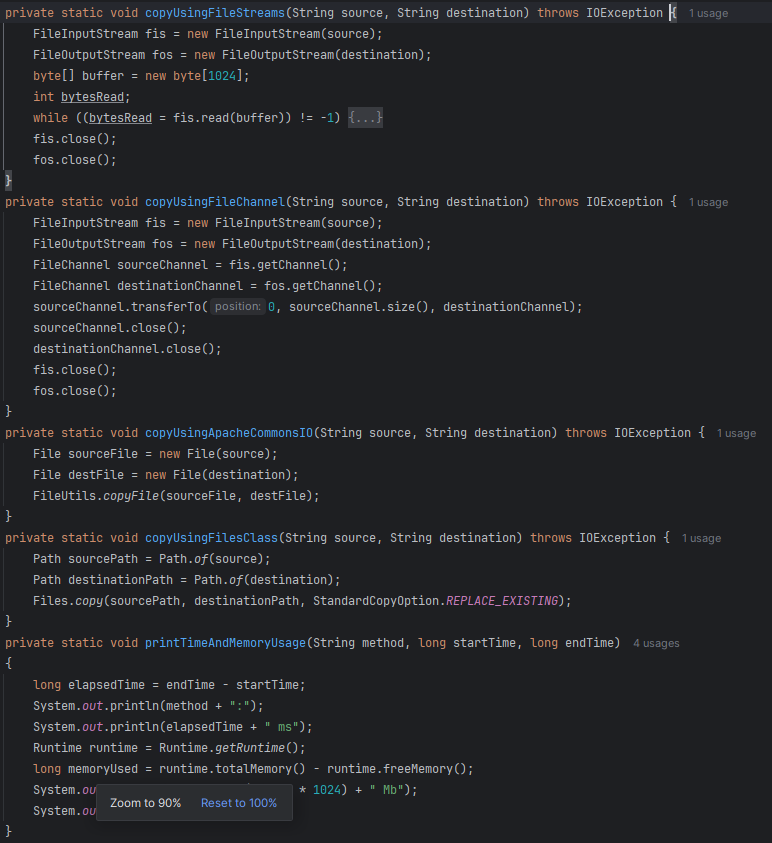


Рисунок 17 – Методы, отвечающие за копирование файла

Результат выполнения представлен на рисунке 18.

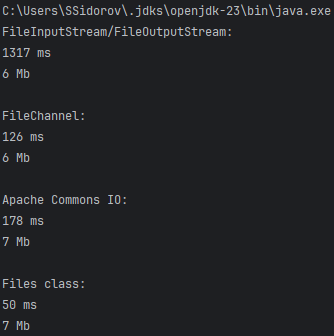


Рисунок 18 – Результат выполнения

Задание 3.

Для выполнения данного задания был написан метод calculateChecksum, вычисляющий 16-битную контрольную сумму файла, код которого представлен на рисунке 19.

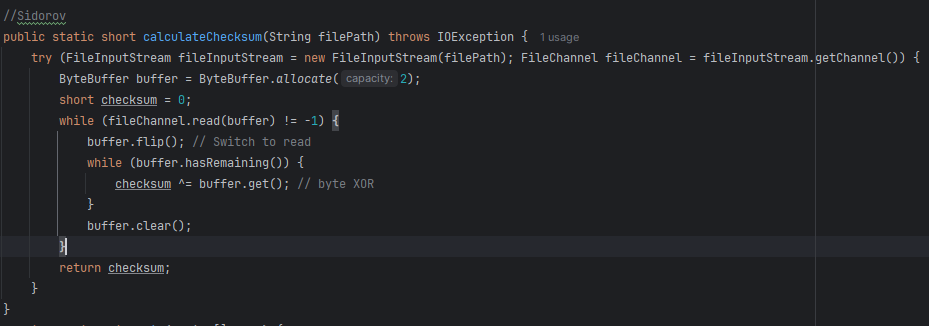


Рисунок 19 – Метод calculateChecksum

Результат вычисления для тестового текстового файла представлен на рисунке 20.

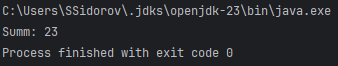


Рисунок 20 – Результат работы метода

**Задание 4**

Для четвертого задания были созданы функции для отслеживания изменений и чтения файла. Данные функции представлены на рисунке 22.

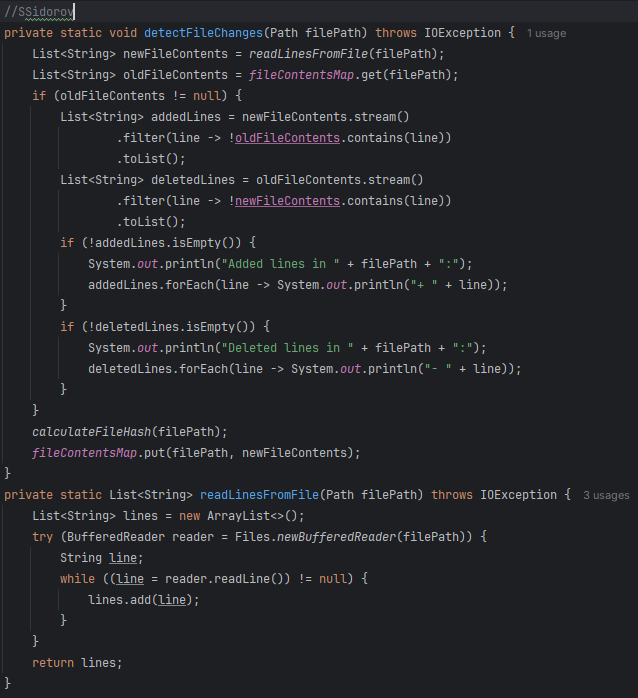


Рисунок 22 – Функции отслеживания изменений и чтения файла

Также была создана функция для подсчёта хэша, представленная на рисунке 23.

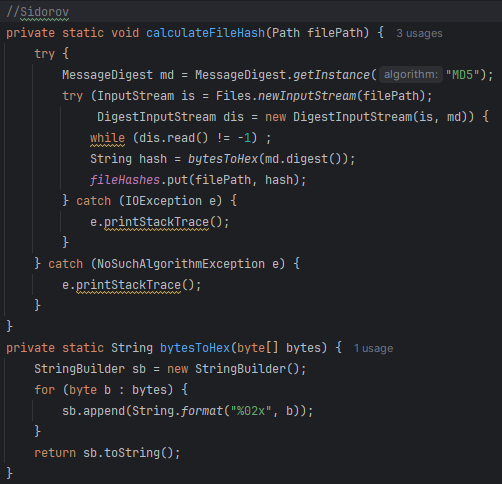


Рисунок 23 – Функция для подсчёта хэша

В функции main() происходит отслеживание изменений указанной директории, а также вызов соответствующих методов. Функция представлена на рисунке 24.

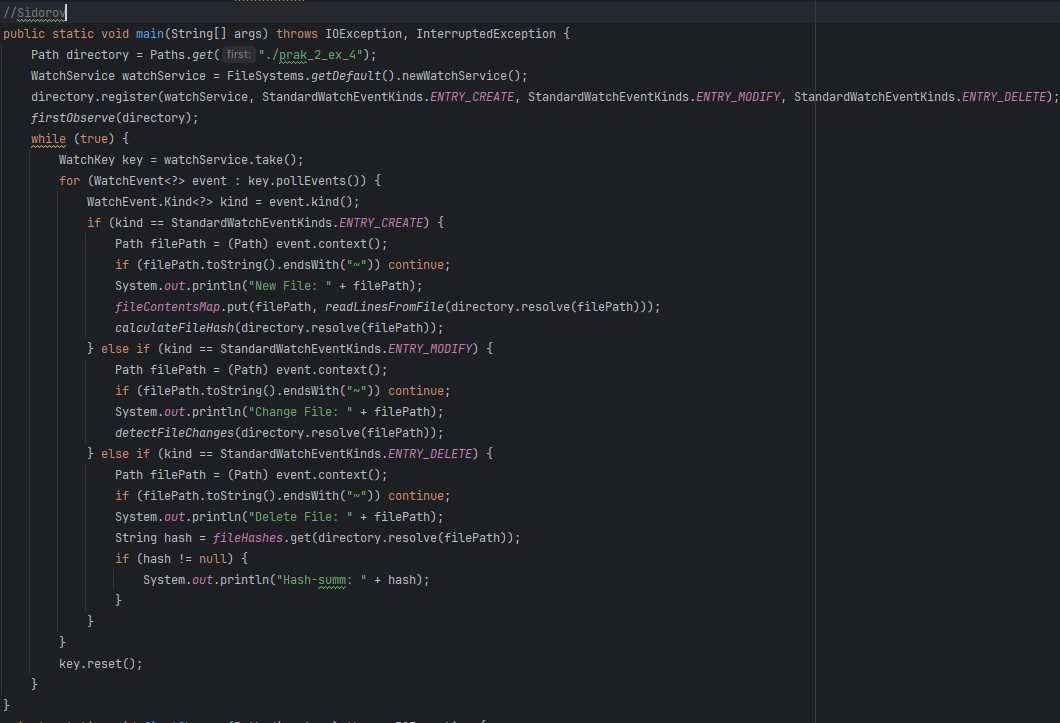


Рисунок 24 – Функция main

Результат работы программы представлен на рисунке 25.

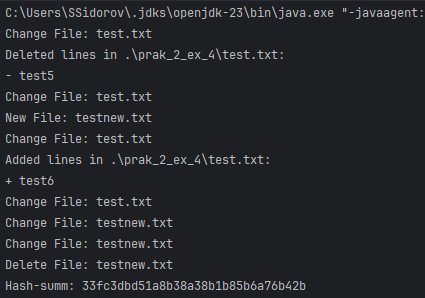


Рисунок 25 – Результат работы

**Практическая работа №3**

Задание 1.

Для выполнения данного задания был сформирован основной код, отвечающий за создание и организацию работу датчиков и сигнализаций. Код класса представлен на рисунке 26.

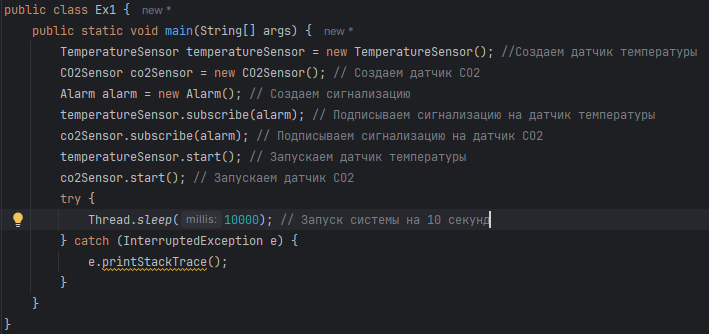


Рисунок 26 – Код основного класса для задания 1

Данный код использует вспомогательные классы, представленные на рисунках 27 – 29.

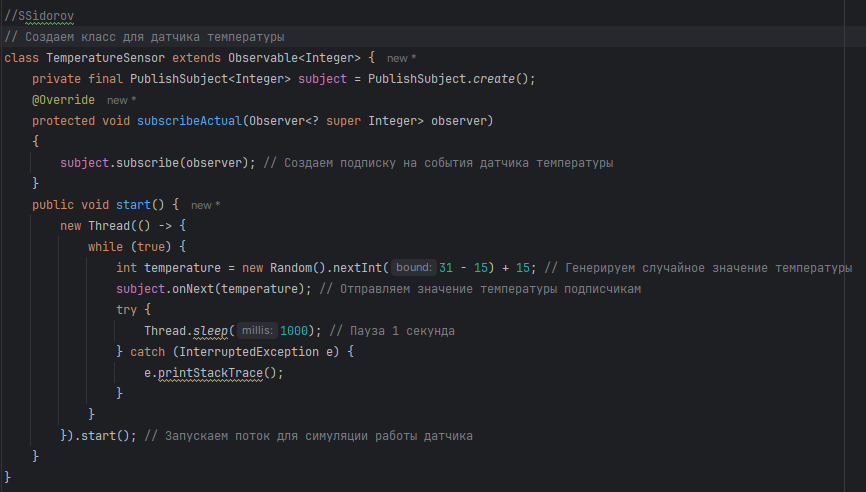


Рисунок 27 – Код для датчика температуры

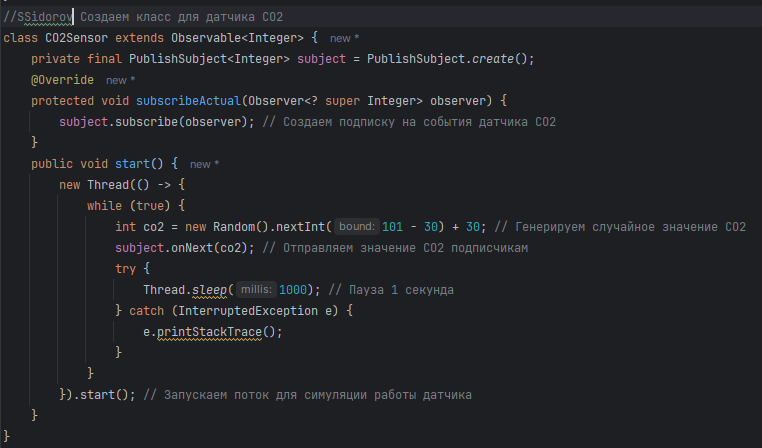


Рисунок 28 – Код для датчика CO2



Рисунок 29 – Код для сигнализации

Результат выполнения представлен на рисунке 30.

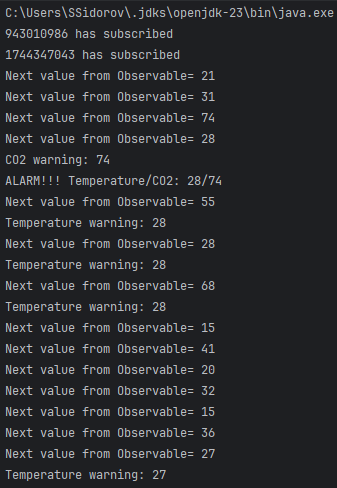


Рисунок 30 – Результат выполнения программы

Задание 2.

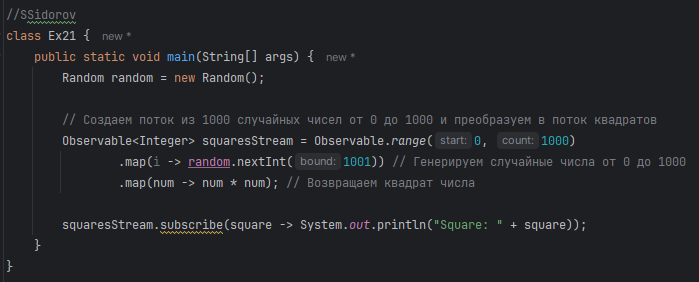
Код для выполнения задания 2.1 представлен на рисунке 31.   


Рисунок 31 – Работа с потоками случайных чисел

Результат выполнения представлен на рисунке 32.

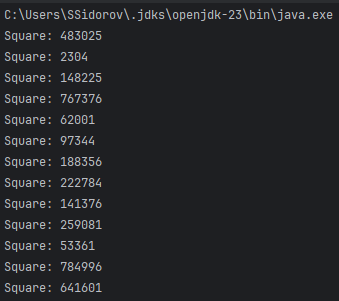


Рисунок 32 – Результат выполнения задания 2.1

Код для выполнения задания 2.2 представлен на рисунке 33.

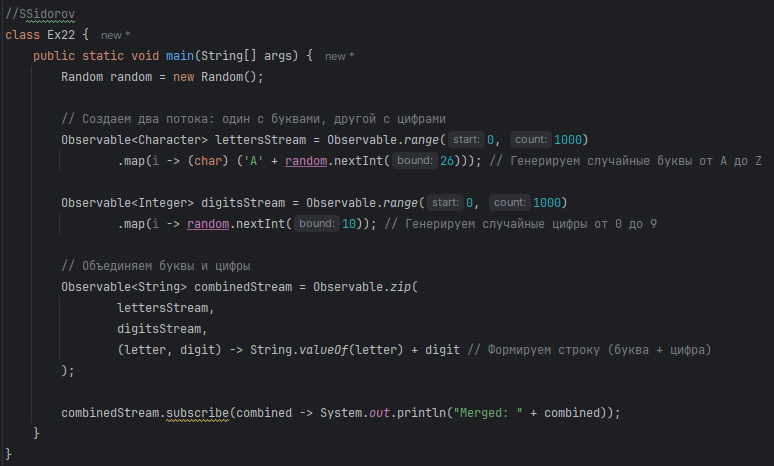


Рисунок 33 – Параллельная обработка потоков

Результат работы представлен на рисунке 34.

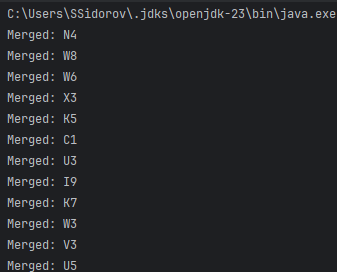


Рисунок 34 – Результат параллельной обработки потоков

Код для выполнения задания 2.3 представлен на рисунке 35.

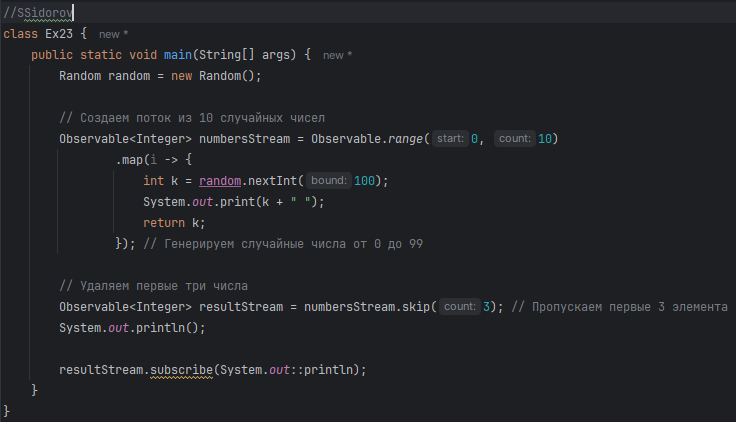


Рисунок 35 – Случайный поток в случайное число

Результат выполнения содержится на рисунке 36.

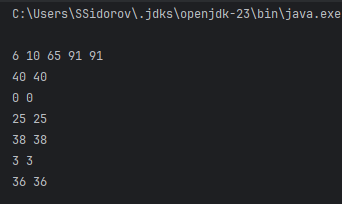


Рисунок 36 – Результат выполнения задания 2.3

Задание 3.

Для выполнения 3го задания был создан класс UserFriend, его код представлен на рисунке 37.



Рисунок 37 – Класс UserFriend

Результат выполнения представлен на рисунке 38.

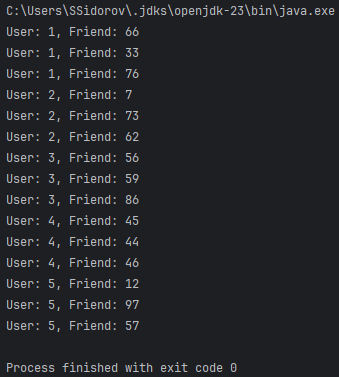


Рисунок 38 – Результат выполнения задания 3

Задание 4.

Класс File представлен на рисунке 39.

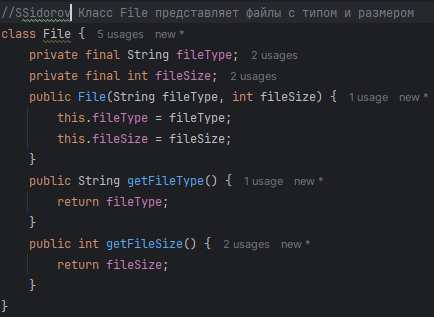


Рисунок 39 – Код класса File

Код класса FileGenerator представлен на рисунке 40.

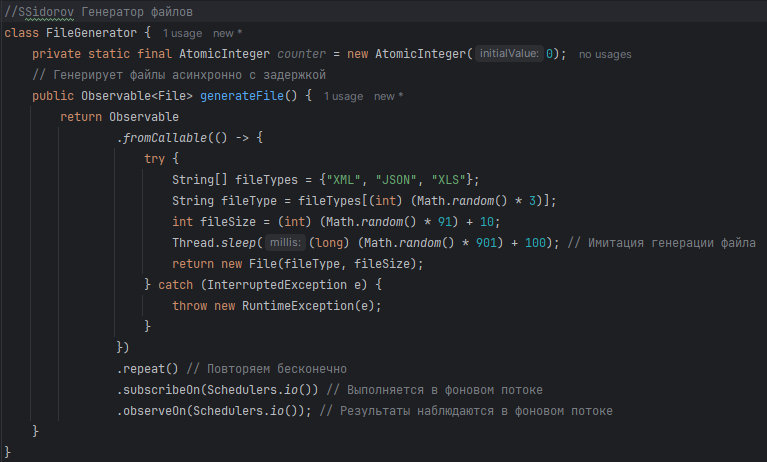


Рисунок 41 – Код FileGenerator

Код класса FileQueue представлен на рисунке 42.

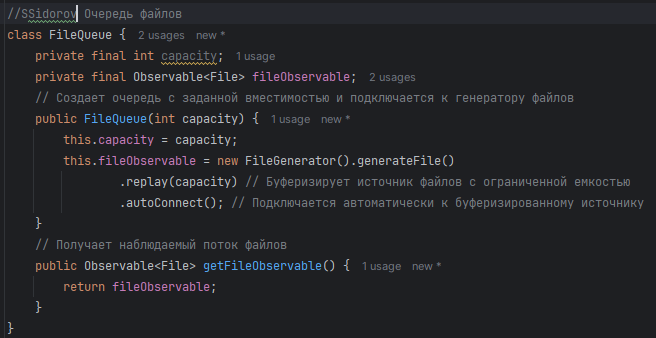


Рисунок 42 – Код FileQueue

Код класса FileProcessor представлен на рисунке 43.

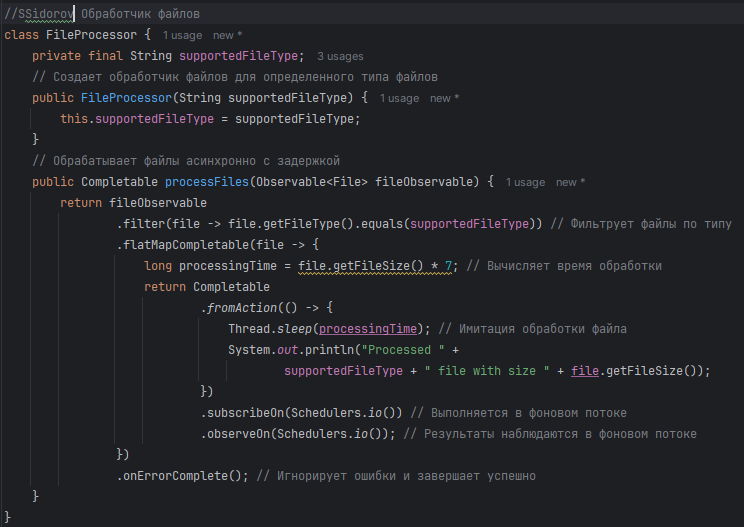


Рисунок 43 – Код FileProcessor

Код основного класса системы представлен на рисунке 44.

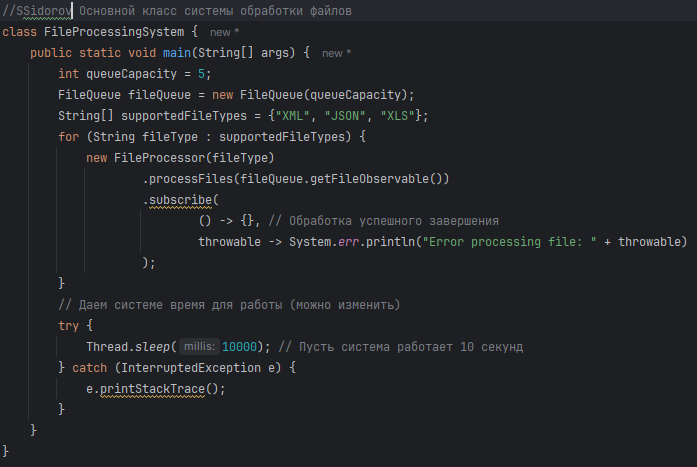


Рисунок 44 – Код основного класса

Вывод программы представлен на рисунке 45.

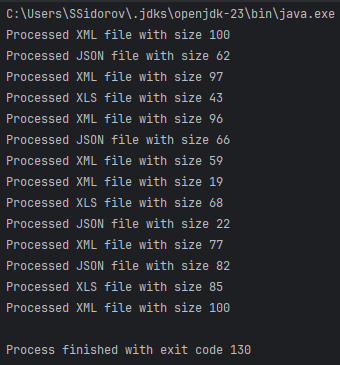


Рисунок 45 – Вывод программы