# Разработка клиент-серверных приложений

Мокин Антон, ИКБО-20-21

# Практическая работа 1

**Задание 1**

Был создан массив из 10000 случайных чисел.

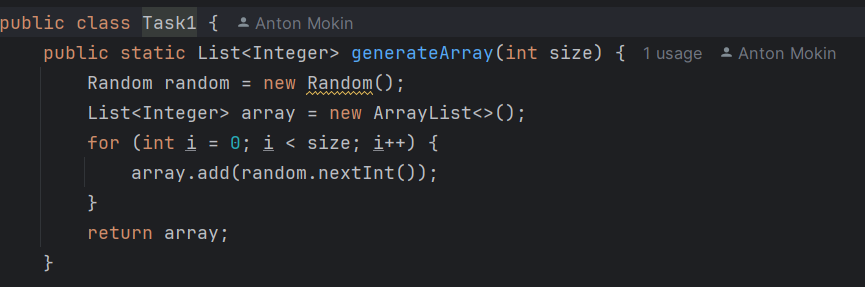


Рисунок 1 – Метод создания списка из 10000 элементов

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

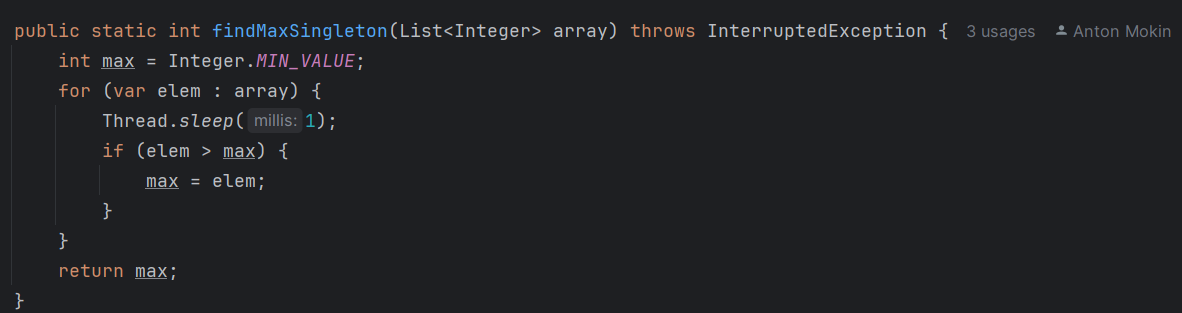


Рисунок 2 – Последовательный способ

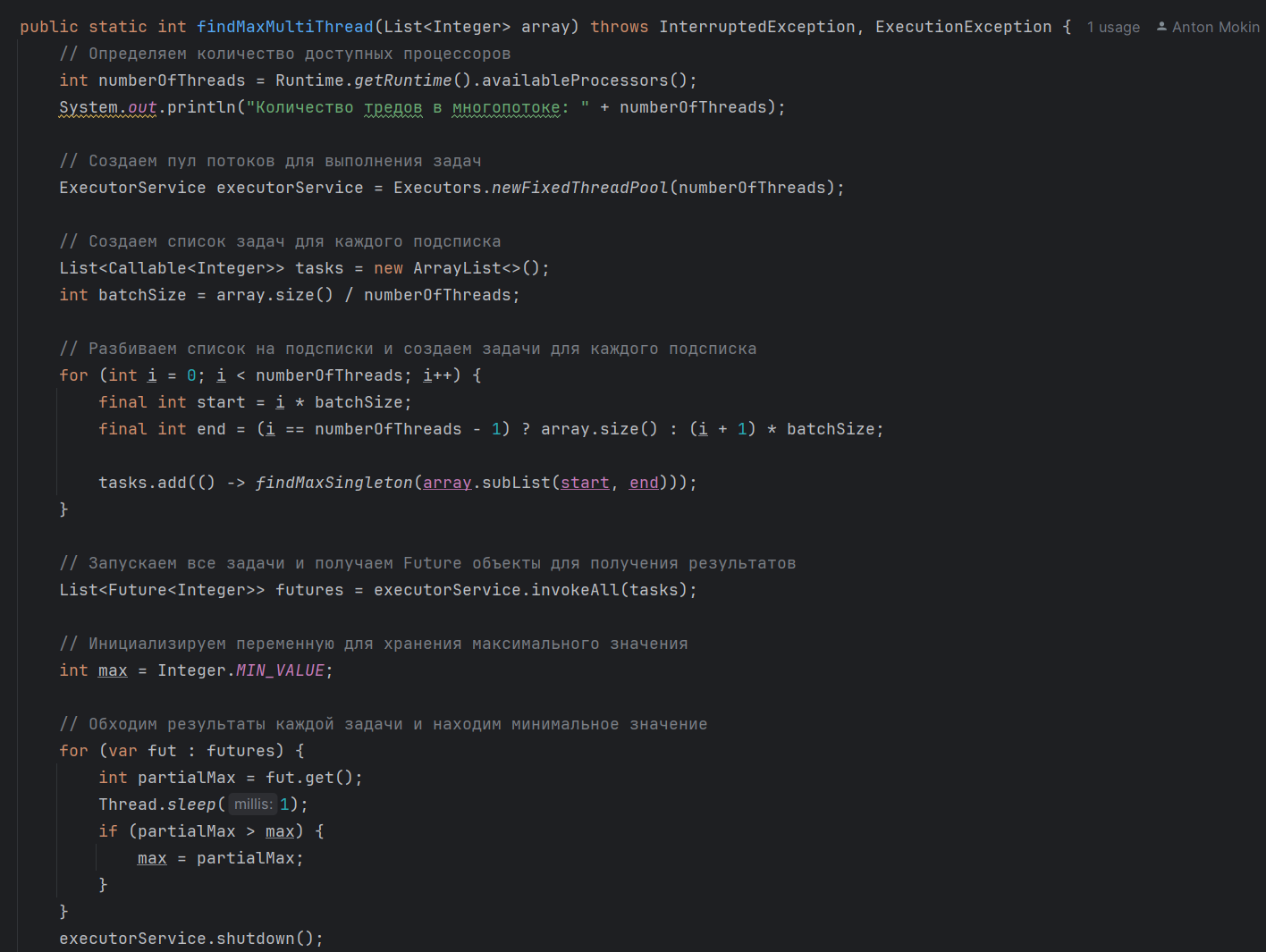


Рисунок 3 – Многопоточный способ

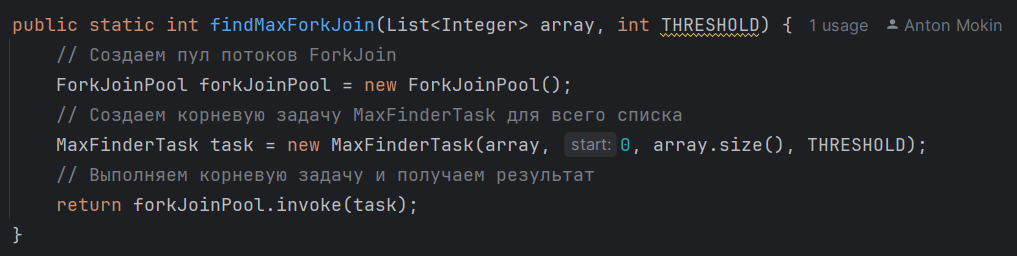


Рисунок 4 – Fork

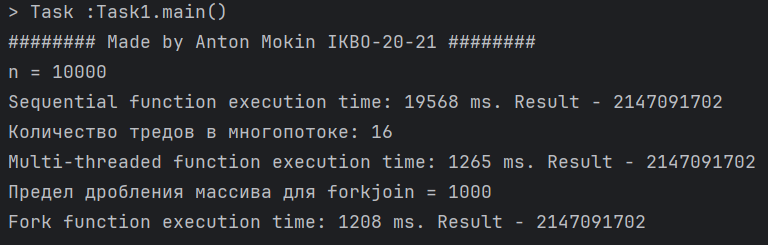


Рисунок 5 – Результат

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.

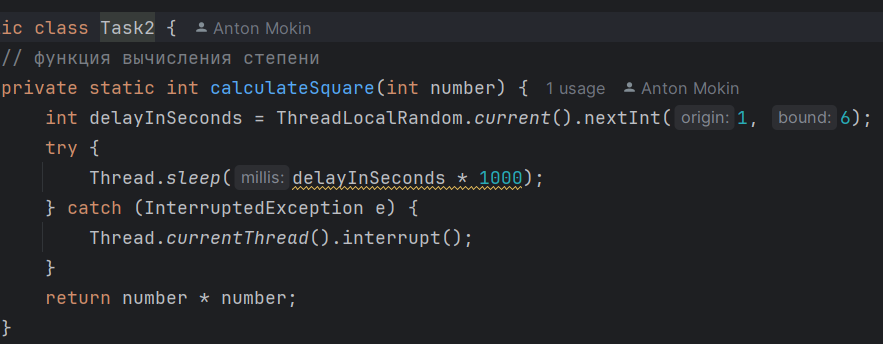


Рисунок 6 – Функция, высчитывающая квадрат

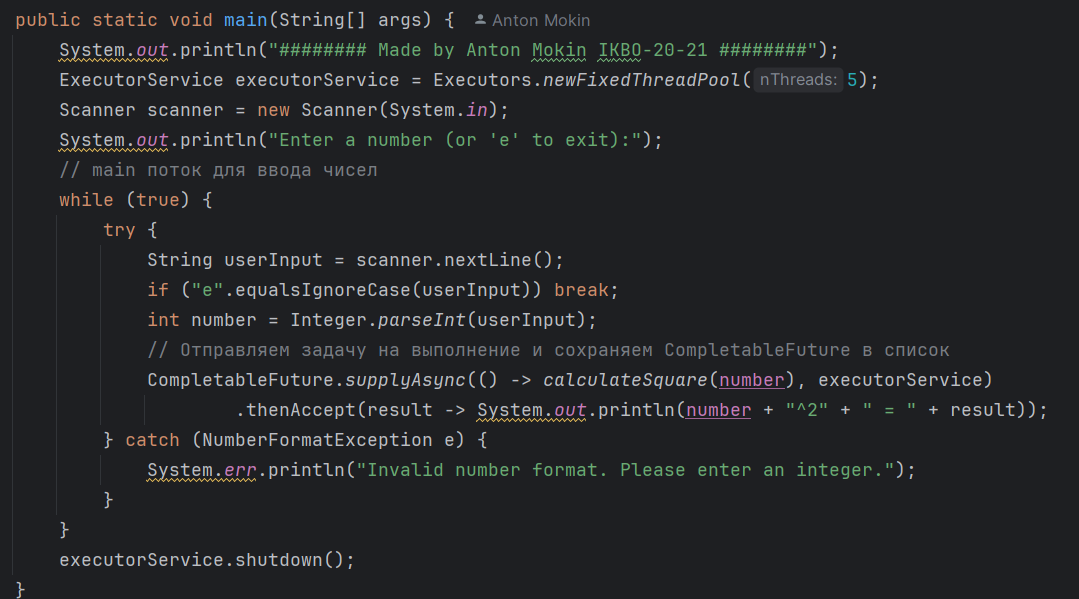


Рисунок 7 – Функция, обрабатывающая значения пользователя

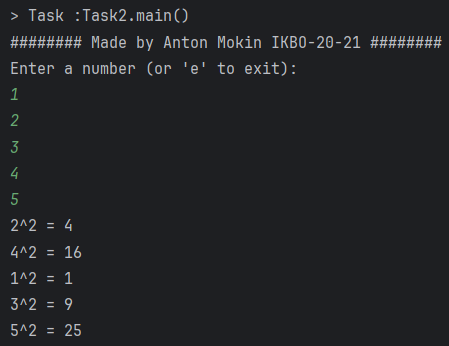


Рисунок 8 – Результат

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».

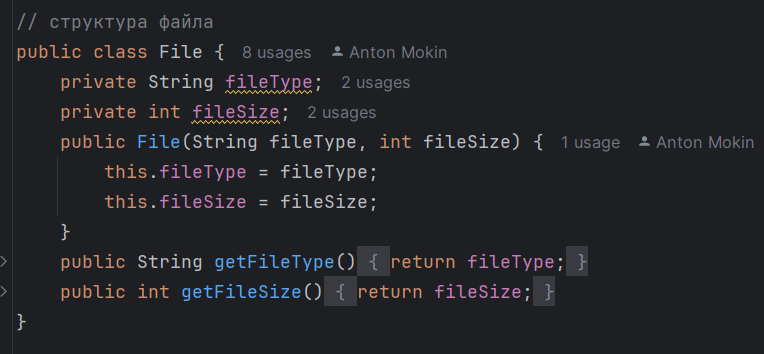


Рисунок 9 – Класс File

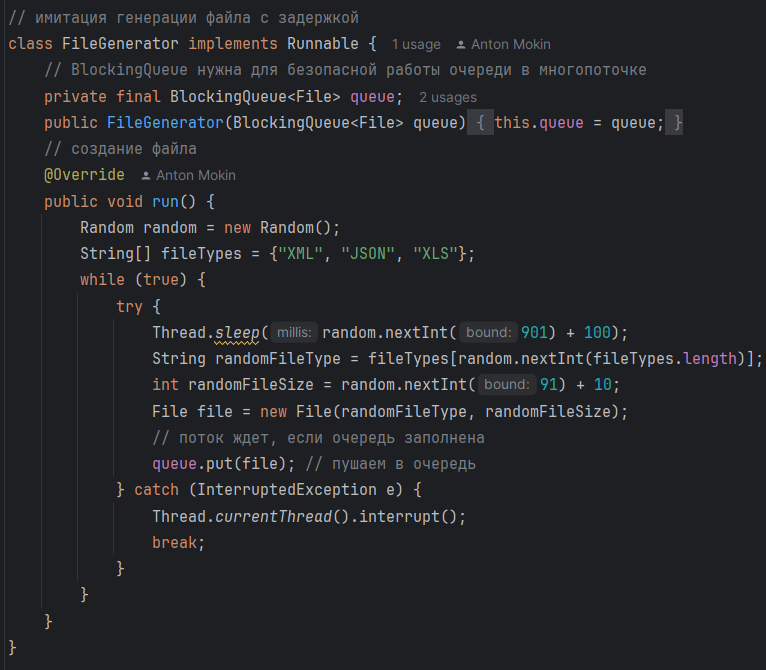


Рисунок 10 – Класс FileGenerator

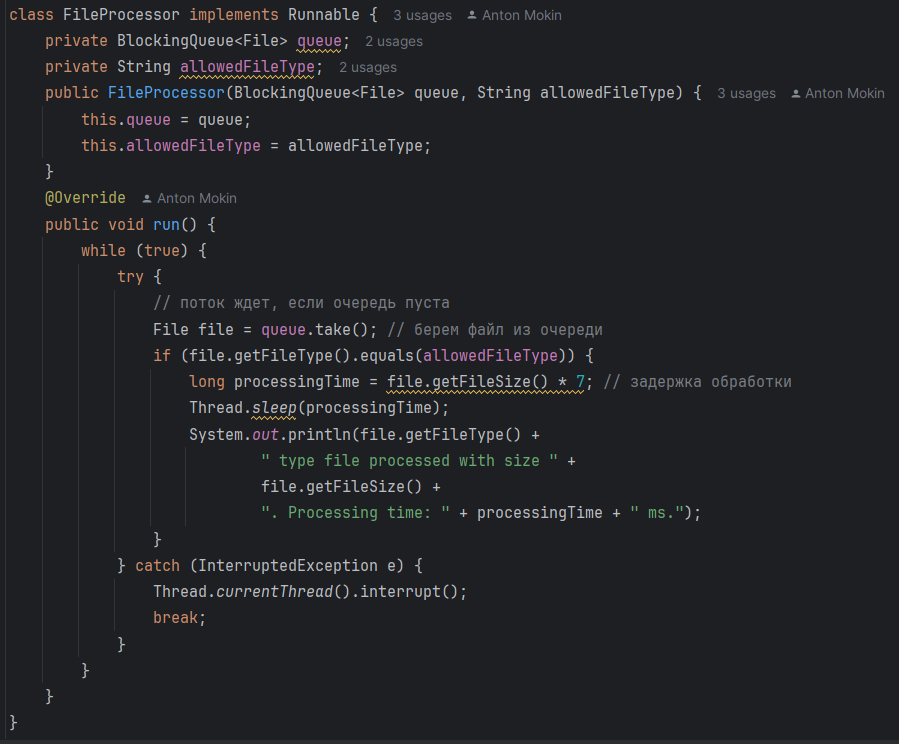


Рисунок 11 – Класс FileProcessor

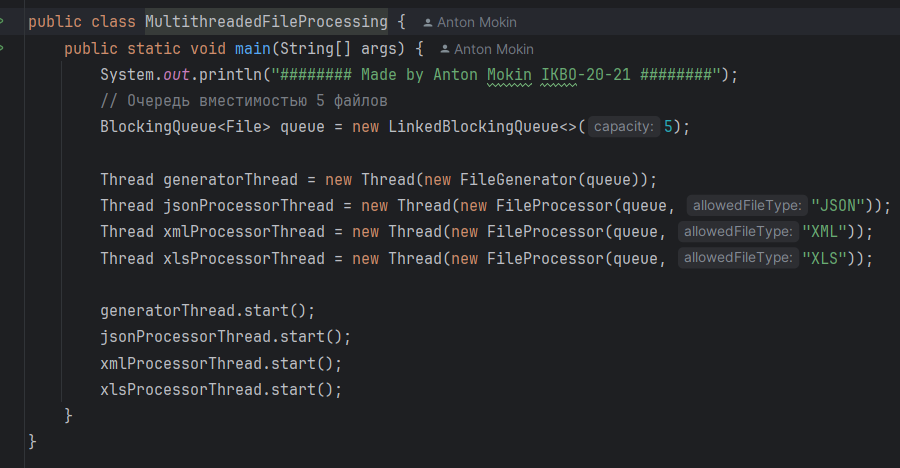


Рисунок 12 – Запуск потоков

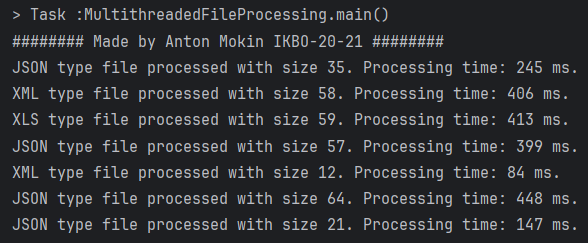


Рисунок 13 – Результат

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

# Практическая работа 2

**Задание 1**

Создать файл формата .txt, содержащий несколько строк текста. С помощью пакета java.nio нужно прочитать содержимое файла и вывести данные в стандартный поток вывода.

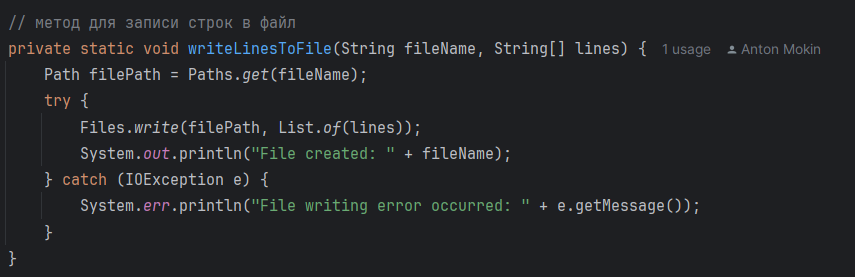


Рисунок 14 – Метод для записи строк в файл

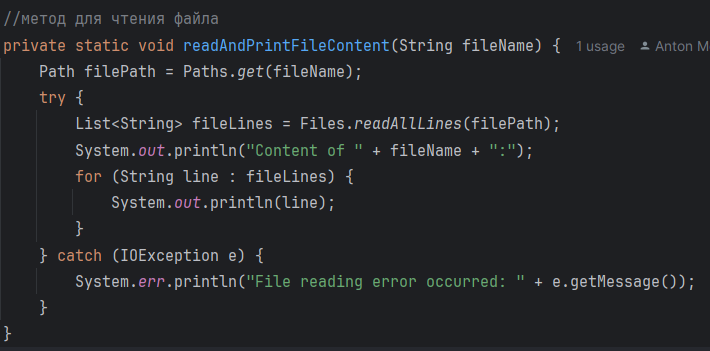


Рисунок 15 – Метод для чтения файла

Ну и в main мы создаем файл, заполняем строчками из списка и читаем файл.

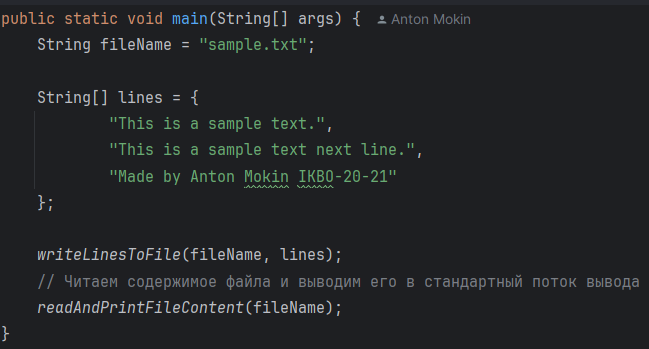


Рисунок 16 – Основная программа

Результат выполнения на рисунке 17.

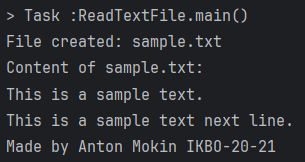


Рисунок 17 – Результат выполнения программы

**Задание 2**

Реализовать копирование файла размером 100 Мб 4 методами:

1) FileInputStream/FileOutputStream

2) FileChannel

3) Apache Commons IO

4) Files class

Замерить затраты по времени и памяти и провести сравнительный анализ.

Создадим для каждого способа по методу и протестируем.



Рисунок 18 – Метод с использованием FileInputStream/FileOutputStream

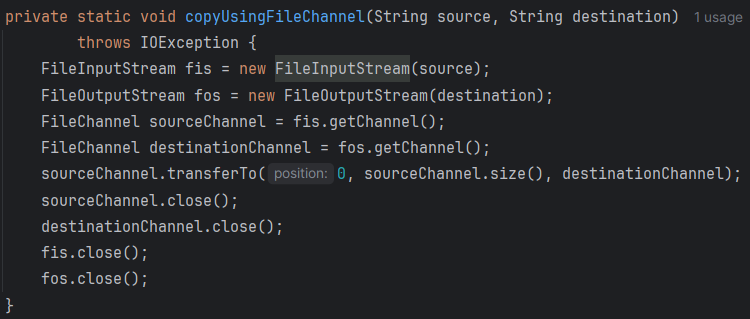


Рисунок 19 – Метод с использованием FileChannel

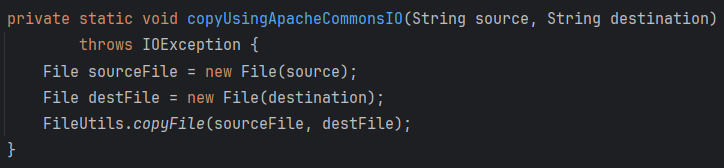


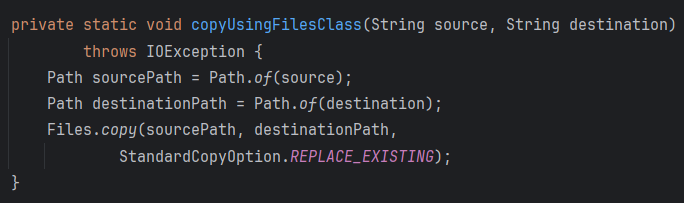
Рисунок 20 – Метод с использованием Apache Commons IO 

Рисунок 21 – Метод с использованием Files class

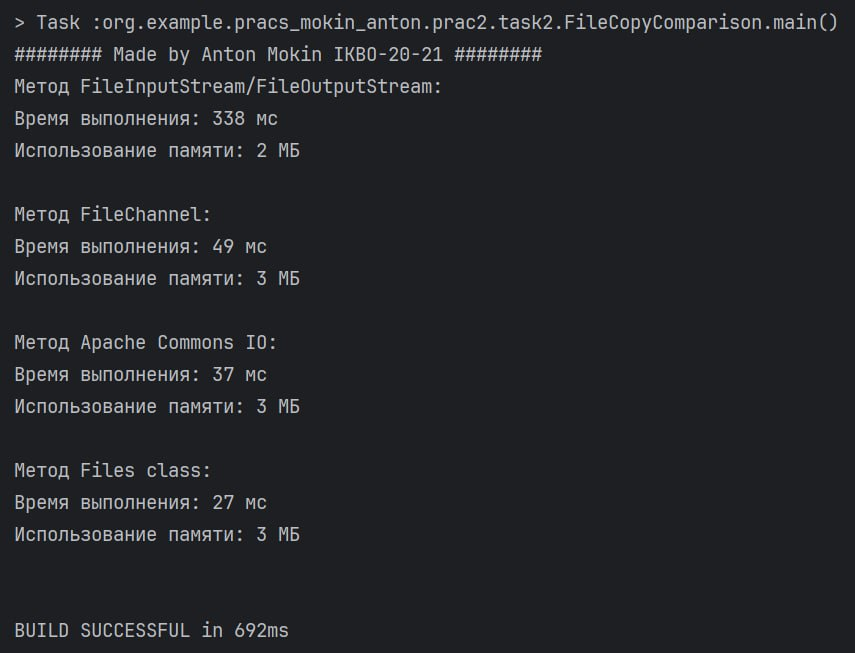


Рисунок 22 – Результат

Самый быстрый – Files class, но он затратный по памяти.

Самый экономный – FileIOStream, но он медленный.

**Задание 3**

Реализовать функцию нахождения 16-битной контрольной суммы файла с использованием бинарных операций и ByteBuffer.

Создадим метод и будем высчитывать контрольную сумму при помощи XOR.

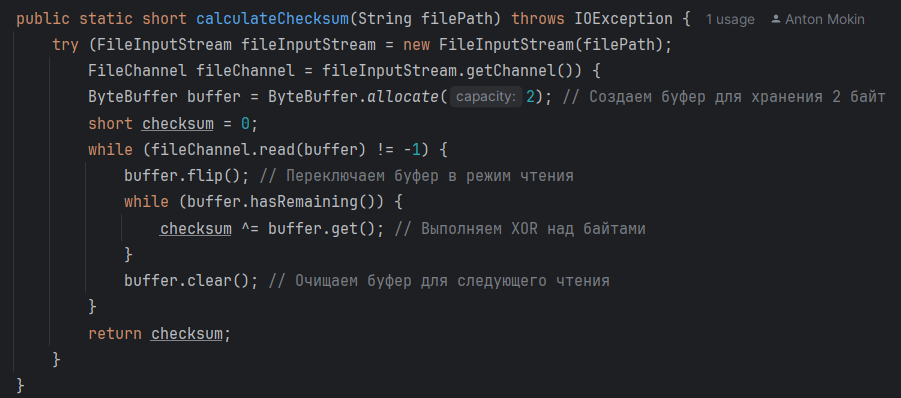


Рисунок 23 – Метод calculateChecksum

Результат представлен на рисунке 24.

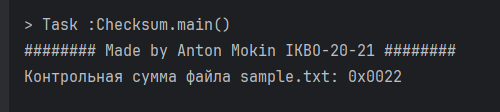


Рисунок 24 –Результат

**Задание 4**

При помощи WatchService реализовать наблюдение за каталогом:

1. При создании нового файла в этом каталоге вывести его название.

2. При изменении файла вывести список изменений (добавленных и удаленных строк).

3. При удалении файла вывести его размер и контрольную сумму (использовать реализацию из задания 3).

Если реализовать пункт 3 не представляется возможным – докажите это.

Указываем определенную директорию, за которой собираемся следить (в нашем случае prac2\_task4), далее создаем Сервис, который будет следить за директорией по 2 меткам (евентам): созданию файла (CREATE) и его изменению (MODIFY).

Вычислить контрольную сумму файла и его содержимое после срабатывания евента DELETE невозможно, потому что данный евент срабатывает тогда, когда файла уже не существует. Доступ к его размеру через стандартные методы ввода-вывода Java невозможен, так как файла физически больше не существует.

Первым делом нужно просканировать директорию, метод просматривает все файлы и все их строчки.

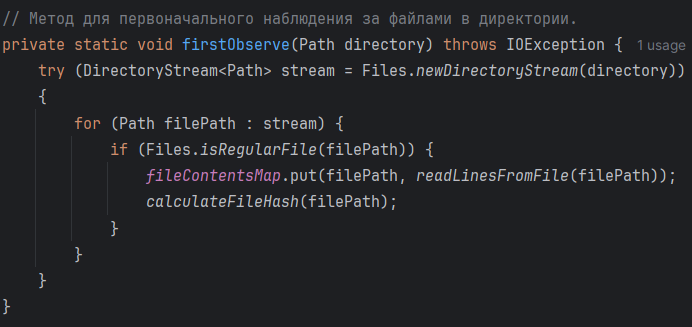


Рисунок 25 – Первоначальный осмотр

Затем создаем сервис отслеживания, регистрируем каталог и начинаем обрабатывать события.

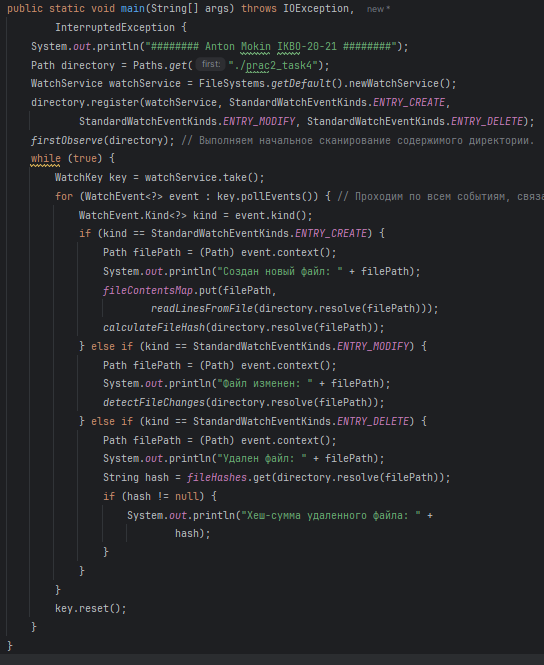


Рисунок 26 – Метод calculateChecksum

Результат представлен на рисунке 27. Создаем файл, вводим там три строчки, потом удаляем поочередно две из них. Видно, что отслеживание работает.

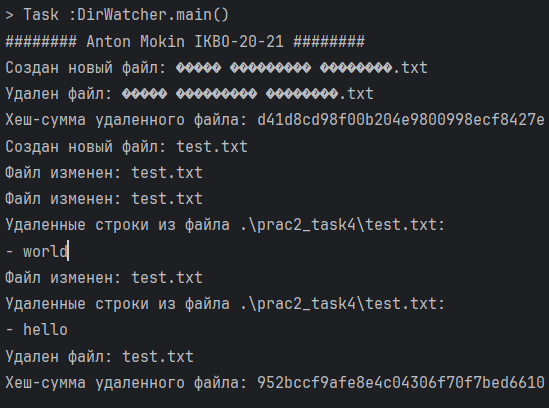


Рисунок 27 –Результат