**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Математическая Кибернетика и Информационные технологии»

Лабораторная работа №8

По дисциплине «Информационные технологии и программирование»

Выполнил: Студент группы

БВТ2203

Попов Александр

Москва

2023

**Цель работы:** Знакомство с аннотациями в ЯП Java.

**Задание 1.** Создайте аннотацию @DataProcessor, которая будет использоваться для пометки методов обработки данных.

**Задание** **2.** Создайте класс DataManager, который будет отвечать за многопоточную обработку данных. Этот класс должен иметь методы: o registerDataProcessor(Object processor): Регистрирует объектобработчик данных с аннотацией @DataProcessor. o loadData(String source): Загружает данные из исходного источника. o processData(): Запускает многопоточную обработку данных, применяя методы с аннотацией @DataProcessor с использованием Stream API. o saveData(String destination): Сохраняет обработанные данные в новый источник.

**Задание 3.** Создайте несколько классов, представляющих различные обработчики данных, и пометьте их аннотацией @DataProcessor. Например, можно создать классы для фильтрации, трансформации и агрегации данных.

**Задание 4.** Используйте многопоточность из java.util.concurrent для эффективной обработки данных параллельно.

**Задание 5.** Протестируйте ваше приложение, загрузив данные из исходного источника, применив различные обработчики с помощью Stream API, и сохраните результаты в новый источник

**Ход работы**

package main;

import java.io.IOException;

import java.lang.reflect.Method;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.stream.Collectors;

*public* *class* DataManager {

*private* List<Object> processors = *new* ArrayList<>();

*private* List<String> inputData = *new* ArrayList<>();

*private* List<String> processedData = *new* ArrayList<>();

*public* void registerDataProcessor(Object *processor*) {

        processors.add(*processor*);

    }

*public* void loadData(String *source*) {

*try* {

            inputData = Files.readAllLines(Paths.get(*source*));

        } *catch* (IOException *e*) {

            e.printStackTrace();

        }

    }

*public* void processData() {

        ExecutorService executorService = Executors.newCachedThreadPool();

        List<List<String>> allProcessedData = *new* ArrayList<>();

*for* (Object processor *:* processors) {

            Class<?> processorClass = processor.getClass();

*for* (Method method *:* processorClass.getDeclaredMethods()) {

*if* (method.isAnnotationPresent(DataProcessor.*class*)) {

                    Object finalProcessor = processor;

                    Method finalMethod = method;

                    executorService.submit(() -> {

                        List<List<String>> tempResultList = inputData.stream()

                                .map(*data* -> invokeDataProcessorMethod(finalProcessor, finalMethod, *data*))

                                .map(*innerList* -> *innerList*.stream().collect(Collectors.toList()))

                                .collect(Collectors.toList());

                        allProcessedData.addAll(tempResultList);

                    });

                }

            }

        }

        executorService.shutdown();

*while* (!executorService.isTerminated()) {

        }

        SortProcessor sortProcessor = *new* SortProcessor();

        List<String> sortedData = sortProcessor.sortData(allProcessedData.stream()

                .flatMap(List*::*stream)

                .collect(Collectors.toList()));

        System.out.println("Отсортированные данные: " + sortedData);

        FilterProcessor filterProcessor = *new* FilterProcessor();

        List<String> filteredData = filterProcessor.filterData(allProcessedData.stream()

                .flatMap(List*::*stream)

                .collect(Collectors.toList()));

        System.out.println("Отфильтрованные данные: " + filteredData);

        TransformProcessor transformProcessor = *new* TransformProcessor();

        List<String> transformedData = transformProcessor.transformData(allProcessedData.stream()

                .flatMap(List*::*stream)

                .collect(Collectors.toList()));

        System.out.println("Трансформированные данные: " + transformedData);

        processedData.addAll(sortedData);

        processedData.add("\n");

        processedData.addAll(filteredData);

        processedData.add("\n");

        processedData.addAll(transformedData);

    }

*private* List<String> invokeDataProcessorMethod(Object *processor*, Method *method*, String *data*) {

*try* {

            Object result = *method*.invoke(*processor*, Collections.singletonList(*data*));

*if* (result != null) {

*if* (result instanceof List<*?*>) {

*return* (List<String>) result;

                } *else* *if* (result instanceof String) {

*return* Collections.singletonList((String) result);

                } *else* {

*return* Collections.emptyList();

                }

            }

*return* Collections.emptyList();

        } *catch* (Exception *e*) {

            e.printStackTrace();

*return* Collections.emptyList();

        }

    }

*public* void saveData(String *destination*) {

*try* {

            Path outputPath = Paths.get(*destination*);

*if* (!Files.exists(outputPath.getParent())) {

                Files.createDirectories(outputPath.getParent());

            }

            String resultString = String.join(" ", processedData);

            Files.write(outputPath, Collections.singletonList(resultString));

        } *catch* (IOException *e*) {

            e.printStackTrace();

        }

    }

*public* *static* void main(String[] *args*) {

        String inputPath = "D:\\лабы\\итип\\лабы\\8лаба\\input.txt";

        String outputPath = "D:\\лабы\\итип\\лабы\\8лаба\\output.txt";

        DataManager dataManager = *new* DataManager();

        dataManager.registerDataProcessor(*new* SortProcessor());

        dataManager.loadData(inputPath);

        System.out.println("Данные до обработки: ");

        dataManager.inputData.forEach(System.out*::*println);

        System.out.println();

        dataManager.processData();

        dataManager.saveData(outputPath);

    }

}

*class* SortProcessor {

    @DataProcessor

*public* List<String> sortData(List<String> *data*) {

        String[] dataArray = *data*.stream()

                .flatMap(*s* -> Arrays.stream(*s*.split(" ")))

                .toArray(String[]*::new*);

        Arrays.sort(dataArray);

*return* Arrays.asList(dataArray);

    }

}

*class* FilterProcessor {

    @DataProcessor

*public* List<String> filterData(List<String> *data*) {

*return* *data*.stream()

                .map(*s* -> *s*.replaceAll("\\d", ""))

                .map(String*::*trim)

                .filter(*s* -> !*s*.isEmpty())

                .collect(Collectors.toList());

    }

}

*class* TransformProcessor {

    @DataProcessor

*public* List<String> transformData(List<String> *data*) {

        List<String> transformedData = *new* ArrayList<>();

        int offset = 0;

*for* (String s *:* *data*) {

            StringBuilder transformed = *new* StringBuilder();

*for* (int i = 0; i < s.length(); i++) {

                char currentChar = s.charAt(i);

*if* (Character.isDigit(currentChar)) {

*for* (char digit *:* String.valueOf(currentChar).toCharArray()) {

                        transformed.append(i + 1 + offset).append(")").append(digit).append(" ");

                    }

                } *else* {

                    transformed.append(i + 1 + offset).append(")").append(currentChar).append(" ");

                }

            }

            transformedData.add(transformed.toString().trim());

            offset += s.length();

        }

*return* transformedData;

    }

}

Файл 1. DataManager.java

package main;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD)

*public* @interface DataProcessor {

}

Файл 2. DataProcessor.java

1. **Аннотация @DataProcessor:**

Данная аннотация применяется к методам и используется для пометки методов обработки данных. Она имеет RetentionPolicy.RUNTIME, что означает, что аннотации будут доступны во время выполнения (например, через рефлексию).

1. **Класс DataManager:**

Поля: processors, inputData, и processedData - списки для хранения объектов-обработчиков данных, входных данных и обработанных данных соответственно.

Метод registerDataProcessor(Object processor): Регистрирует объект-обработчик данных, добавляя его в список processors.

Метод loadData(String source): Загружает данные из файла по указанному пути в inputData. В случае ошибки чтения из файла, выводится стек трейс ошибки.

Метод processData(): Создает ExecutorService для многопоточной обработки данных. Для каждого метода с аннотацией @DataProcessor в каждом объекте-обработчике создается отдельный поток, который обрабатывает данные с использованием Stream API. Результаты обработки объединяются в общий список allProcessedData. После завершения всех потоков, данные сортируются, фильтруются и трансформируются с использованием соответствующих обработчиков. Обработанные данные сохраняются в общий список processedData, разделенные строками **\n** между секциями.

Метод invokeDataProcessorMethod(Object processor, Method method, String data): Вызывает метод обработчика данных с использованием рефлексии. Обрабатывает результат метода и возвращает список строк.

Метод saveData(String destination): Сохраняет обработанные данные в файл по указанному пути, разделяя их пробелами. Создает директорию, если она не существует. В случае ошибки записи в файл, выводится стек трейс ошибки.

Метод main(String[] args): Создает объект DataManager. Регистрирует обработчик данных (в данном случае, SortProcessor). Загружает данные из файла. Выводит данные до обработки. Обрабатывает данные параллельно. Сохраняет обработанные данные в новый файл.

1. **Обработчики данных (SortProcessor, FilterProcessor, TransformProcessor):**

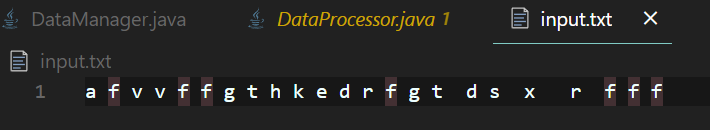
Каждый обработчик данных содержит метод с аннотацией @DataProcessor, который принимает список строк и возвращает обработанный список строк.****

Рис. 1. Исходные данные

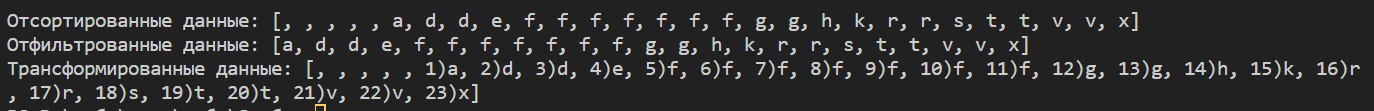


Рис. 2. Результат выполнения программы.

**Выводы**

1. Я научился работать с Аннотациями в ЯП Java. Они позволяют добавлять дополнительную информацию к коду и могут использоваться для анализа и автоматизации процесса компиляции.

2. Я научился работать со StreamAPI. Он упрощает многие типичные задачи обработки данных, такие как фильтрация, отображение, сортировка и агрегирование.

3. Я научился работать с java.util.concurrent. Он включает в себя средства для создания потоков, управления потоками, а также синхронизации и координации между потоками.