**МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

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

Кафедра «Информационные технологии»

Лабораторная работа по ИТП №8

Выполнил: Студент группы

БВТ2204

Сарыков Эрбол

Москва

2023

Задание для выполнения лабораторной работы:

Вам необходимо разработать приложение, которое считывает данные из

исходного источника (например, файл, база данных или сетевой ресурс),

применяет к ним различные операции с использованием Stream API, и

сохраняет результаты в новый источник данных.

1. Создайте аннотацию @DataProcessor, которая будет использоваться

для пометки методов обработки данных.

2. Создайте класс DataManager, который будет отвечать за

многопоточную обработку данных. Этот класс должен иметь методы:

o registerDataProcessor(Object processor): Регистрирует объект -

обработчик 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.

3. Создайте несколько классов, представляющих различные

обработчики данных, и пометьте их аннотацией @ DataProcessor.

Например, можно создать классы для фильтрации, трансформации

и агрегации данных.

4. Используйте многопоточность из java.util.concurrent для

эффективной обработки данных параллельно.

5. Протестируйте ваше приложение, загрузив данные из исходного

источника, применив различные обработчики с помощью Stream

API, и сохраните результаты в новый источник.

Реализация

DataManager.java

import java.util.\*;

import java.util.concurrent.\*;

import java.util.stream.\*;

import java.io.\*;

import java.nio.file.\*;

import java.lang.reflect.\*;

public class DataManager {

    private List<String> data = new ArrayList<>();

    private List<Method> processors = new ArrayList<>();

    public void registerDataProcessor(Object processor) {

        for (Method method : processor.getClass().getDeclaredMethods()) {

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

                processors.add(method);

            }

        }

    }

    public void loadData(String source) throws IOException {

        data = Files.readAllLines(Paths.get(source));

    }

    public void processData() {

        ExecutorService executor = Executors.newFixedThreadPool(processors.size());

        List<Future<String>> futures = processors.stream()

                .map(method -> executor.submit(() -> {

                    try {

                        Object result = method.invoke(null, data);

                        if (result instanceof List<?>) {

                            List<?> resultList = (List<?>) result;

                            if (resultList.stream().allMatch(item -> item instanceof String)) {

                                List<String> processedData = resultList.stream()

                                        .map(String::valueOf)

                                        .collect(Collectors.toList());

                                String comment = method.getAnnotation(DataProcessor.class).comment();

                                return comment + "\n" + processedData.toString() + "\n";

                            } else {

                                return "Error: Список содержит элементы, которые не являются строковыми.\n";

                            }

                        } else {

                            return "Error: Метод не вернул List<String>.\n";

                        }

                    } catch (IllegalAccessException | InvocationTargetException e) {

                        e.printStackTrace();

                        return "";

                    }

                }))

                .collect(Collectors.toList());

        data = futures.stream().flatMap(future -> {

            try {

                return Stream.of(future.get());

            } catch (InterruptedException | ExecutionException e) {

                e.printStackTrace();

            }

            return Stream.empty();

        }).collect(Collectors.toList());

        executor.shutdown();

    }

    public void saveData(String destination) throws IOException {

        Files.write(Paths.get(destination), data);

    }

}

DataProcessor.java

import java.lang.annotation.\*;

@Retention(RetentionPolicy.RUNTIME) // цикл жизни устанавливаем в рантайме

@Target(ElementType.METHOD) // помечаем метод

public @interface DataProcessor {

    String comment() default "";

}

DataProcessors.java

import java.util.\*;

import java.util.stream.\*;

public class DataProcessors {

    @DataProcessor(comment = "Удаляем из входного списка пустые строки или строки из пробелов")

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

        return data.stream()

            .filter(line -> !line.trim().isEmpty())

            .collect(Collectors.toList());

    }

    @DataProcessor(comment = "Преобразуем все строки из входного списка в верхний регистр")

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

        return data.stream()

            .map(String::toUpperCase)

            .collect(Collectors.toList());

    }

    @DataProcessor(comment = "// объединяем все строки во входном списке в одну строку разделённую запятыми и пробелами")

    public static List<String> aggregateData(List<String> data) {

        String aggregated = data.stream().collect(Collectors.joining(", "));

        return List.of(aggregated);

    }

}

Main.java

import java.util.List;

public class Main {

    public static void main(String[] args) throws Exception {

        List<String> inputData = List.of("text", "123123","" , " ", "date");

        java.nio.file.Files.write(java.nio.file.Paths.get("C:\\Users\\chiri\\Desktop\\2-8 LABS\\8\\input.txt"), inputData);

        DataManager dataManager = new DataManager();

        dataManager.registerDataProcessor(new DataProcessors());

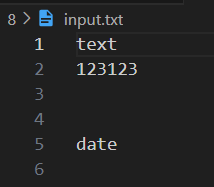
        dataManager.loadData("C:\\Users\\chiri\\Desktop\\2-8 LABS\\8\\input.txt");

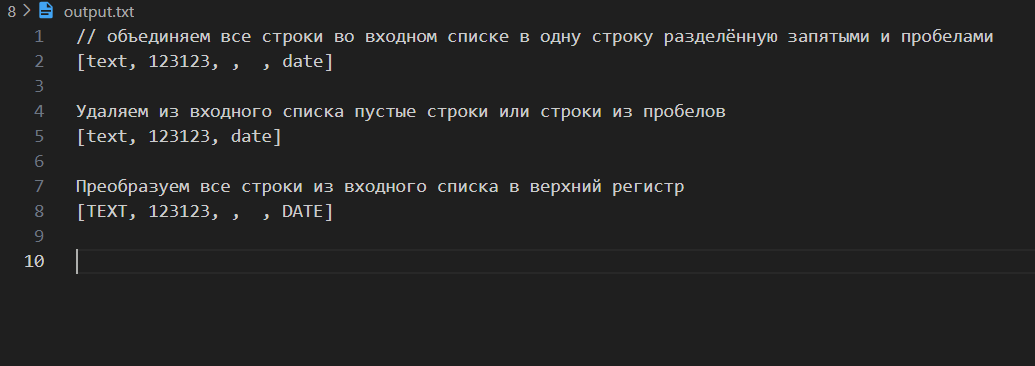
        dataManager.processData();

        dataManager.saveData("C:\\Users\\chiri\\Desktop\\2-8 LABS\\8\\output.txt");

    }

}





Класс DataManager

Основные методы

registerDataProcessor(Object processor): Регистрирует обработчик данных (методы с аннотацией @DataProcessor) из переданного объекта.

loadData(String source) throws IOException: Загружает данные из файла по указанному пути.

processData(): Обрабатывает данные, применяя зарегистрированные обработчики данных (методы с аннотацией @DataProcessor).

saveData(String destination) throws IOException: Сохраняет обработанные данные в файл по указанному пути.

Описание работы метода processData()

Создается пул потоков с фиксированным числом потоков, равным количеству зарегистрированных обработчиков данных.

Для каждого зарегистрированного обработчика данных выполняется следующее:

В отдельном потоке выполняется вызов метода обработчика.

Результат обработки проверяется на соответствие требованиям: метод должен вернуть List<String>, и все элементы этого списка должны быть строками.

Если результат соответствует требованиям, то результат добавляется в общий список данных с добавлением комментария из аннотации @DataProcessor.

Если результат не соответствует требованиям, генерируется сообщение об ошибке.

Результаты обработки собираются в список futures, содержащий объекты Future<String>, представляющие результаты выполнения методов обработчиков.

После завершения обработки данных из каждого потока результаты собираются в единый список.

Пул потоков закрывается.

Класс DataProcessors

Класс DataProcessors содержит методы для обработки данных, каждый из которых аннотирован аннотацией @DataProcessor. В данном проекте реализованы три метода обработки данных:

filterData: Удаляет пустые строки и строки, состоящие из пробелов.

transformData: Преобразует все строки в верхний регистр.

aggregateData: Объединяет все строки в одну строку, разделенную запятыми и пробелами.

Класс Main

Главный класс Main используется для демонстрации работы системы. В нем создается файл с исходными данными, затем создается объект DataManager, зарегистрированы обработчики данных из класса DataProcessors, данные загружаются, обрабатываются и сохраняются в другой файл.