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»

Отчет по лабораторной работе №8

по дисциплине «Информационные технологии и программирование»

Выполнил: студент группы БПИ2401

Юлдашев Всеволод

Проверил: Харрасов Камиль Раисович

Москва

2025

Цель работы: Изучение современных средств обработки данных в Java (Stream API) и механизмов метапрограммирования (Reflection API, Аннотации). Разработка гибкого модульного приложения, использующего многопоточность для параллельного выполнения независимых задач обработки данных. Получение навыков создания собственных аннотаций для маркировки методов и динамического управления выполнением программы.

Ход работы:

Разработано приложение для обработки текстовых данных.

Используемые технологии:

- Stream API (фильтрация, преобразование, агрегация)
- Reflection API (динамический вызов методов)
- Annotations (маркировка методов обработки)
- Concurrency (ExecutorService для параллельного выполнения)

=== ИСХОДНЫЙ КОД ===

--- 1. Аннотация DataProcessor ---

Файл: src/main/java/com/example/dataprocessor/DataProcessor.java

```
package com.example.dataprocessor;

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 {
    String name() default "";
    int priority() default 0;
}
```

--- 2. Менеджер данных (DataManager) ---
Файл: src/main/java/com/example/dataprocessor/DataManager.java

```
// Отвечает за загрузку, регистрацию обработчиков и многопоточный запуск
public class DataManager {
    // ... (поля: data, processors, executorService)

    public void processData() throws Exception {
        System.out.println("\n=== Начало обработки данных ===");
        List<Future<List<String>>> futures = new ArrayList<>();

        for (Object processor : processors) {
            Method[] methods = processor.getClass().getDeclaredMethods();
            for (Method method : methods) {
                if (method.isAnnotationPresent(DataProcessor.class)) {
                    DataProcessor annotation = method.getAnnotation(DataProcessor.class);

                    // Запуск обработки в отдельном потоке
                    Future<List<String>> future = executorService.submit(() -> {
                        System.out.println("Поток " + Thread.currentThread().getName() +
                                           " обрабатывает: " + annotation.name());
                        return (List<String>) method.invoke(processor, data);
                    });
                    futures.add(future);
                }
            }
        }
        // ... (сбор результатов)
    }
    // ... (методы loadData, saveData)
}
```

--- 3. Обработчики (Processors) ---

Файлы: FilterProcessor.java, TransformProcessor.java, AggregationProcessor.java

```
public class FilterProcessor {
    @DataProcessor(name = "Фильтр длинных строк", priority = 1)
    public List<String> filterLongStrings(List<String> data) {
        return data.stream().filter(s -> s.length() > 5).collect(Collectors.toList());
    }
    // ... другие методы
}

public class TransformProcessor {
    @DataProcessor(name = "Преобразование в верхний регистр", priority = 1)
    public List<String> toUpperCase(List<String> data) {
        return data.stream().map(String::toUpperCase).collect(Collectors.toList());
    }
}

public class AggregationProcessor {
    @DataProcessor(name = "Подсчет статистики", priority = 2)
    public List<String> calculateStatistics(List<String> data) {
        // ... stream calculations (average, max)
        return List.of("Статистика:", "Всего элементов: " + data.size());
    }
}
```

--- 4. Главный класс (Main) ---

Файл: src/main/java/com/example/dataprocessor/Main.java

```
public class Main {
    public static void main(String[] args) {
        try {
            DataManager dataManager = new DataManager();
```

```
            // Регистрация модулей обработки
            dataManager.registerDataProcessor(new FilterProcessor());
            dataManager.registerDataProcessor(new TransformProcessor());
            dataManager.registerDataProcessor(new AggregationProcessor());

            dataManager.loadData("data/input.txt");
            dataManager.processData();
            dataManager.saveData("data/output.txt");
            dataManager.shutdown();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

=== ПРИМЕР РАБОТЫ ===

Входные данные (data/input.txt):
Hello World
Java 8
Stream API

Консольный вывод:
Регистрация обработчика: FilterProcessor
Регистрация обработчика: TransformProcessor
Регистрация обработчика: AggregationProcessor
Загрузка данных из: data/input.txt
Загружено строк: 3

=== Начало обработки данных ===
Поток pool-1-thread-1 обрабатывает: Фильтр длинных строк
Поток pool-1-thread-2 обрабатывает: Преобразование в верхний регистр

Поток pool-1-thread-3 обрабатывает: Подсчет статистики
Завершена обработка: Фильтр длинных строк, результат: 2 элементов
Завершена обработка: Преобразование в верхний регистр, результат: 3 элементов
=== Обработка завершена, всего результатов: ... ===

Сохранение данных в: data/output.txt
Приложение успешно завершено!

Вывод: В результате выполнения лабораторной работы было разработано модульное приложение для обработки данных. Мы закрепили навыки использования Stream API для лаконичной и эффективной работы с коллекциями данных (фильтрация, маппинг, агрегация).
Особое внимание было уделено механизмам рефлексии (Reflection API) и аннотациям, что позволило создать гибкую архитектуру, где методы обработки определяются декларативно. Интеграция ExecutorService обеспечила параллельное выполнение независимых задач обработки, демонстрируя преимущества многопоточности в задачах data processing. Приложение является расширяемым: добавление новой логики требует лишь создания метода с аннотацией @DataProcessor.

https://github.com/zevy3/java-labs/tree/main/java8