МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ
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 информационным технологиям и программированию

на тему: **«Аннотации. Stream API»**

Выполнил: студент группы БПИ2403

ФИО: Сон Владимир Сергеевич

Руководитель: Рыбаков Егор Дмитриевич

Москва, 2025

Цель работы: Изучить механизм аннотаций в Java, их назначение и способы создания пользовательских аннотаций, а также освоить основы Stream API для функциональной обработки данных, включая создание потоков, применение промежуточных и терминальных операций.

Задание 1.

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.

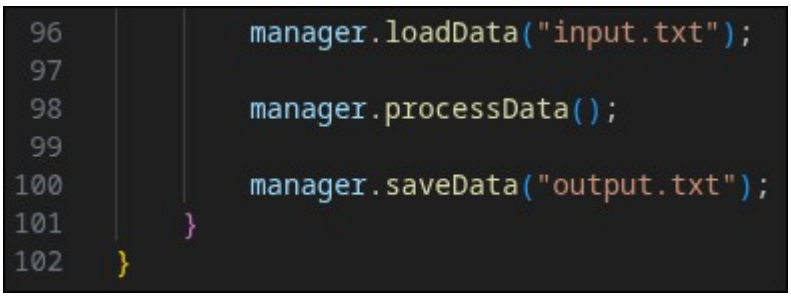
1. Создайте аннотацию @DataProcessor, которая будет использоваться для пометки методов обработки данных.

2. Создайте класс DataManager, который будет отвечать за многопоточную обработку данных. Этот класс должен иметь методы:
• registerDataProcessor(Object processor) — регистрирует объект — обработчик данных с аннотацией @DataProcessor;
• loadData(String source) — загружает данные из исходного источника;
• processData() — запускает многопоточную обработку данных, применяя методы с аннотацией @DataProcessor с использованием Stream API;
• 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.

Ход работы:

```java
import java.lang.annotation.*;
import java.lang.reflect.Method;
import java.util.*;
import java.util.stream.Collectors;
import java.util.concurrent.*;
import java.io.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@interface DataProcessor {
}

class DataManager {
    private List<String> data = new ArrayList<>();
    private List<Method> processors = new ArrayList<>();
    private Object processorObj;

    public void registerDataProcessor(Object processor) {
        this.processorObj = processor;
        for (Method m : processor.getClass().getMethods()) {
            if (m.isAnnotationPresent(DataProcessor.class)) {
                processors.add(m);
            }
        }
    }

    public void loadData(String source) {
        try (BufferedReader br = new BufferedReader(new FileReader(source))) {
            data = br.lines().collect(Collectors.toList());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void processData() {
        ExecutorService executor = Executors.newFixedThreadPool(2);

        try {
            List<String> result = executor.submit(() -> {
                return data.stream()
                    .map(item -> {
                        for (Method m : processors) {
                            try {
                                item = (String) m.invoke(processorObj, item);
                            } catch (Exception e) {}
                        }
                        return item;
```

```
                    })
                    .distinct()
                    .sorted()
                    .collect(Collectors.toList());
            }).get();

            data = result;
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            executor.shutdown();
        }
    }

    public void saveData(String destination) {
        try (PrintWriter pw = new PrintWriter(destination)) {
            data.forEach(pw::println);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class MyProcessor {
    @DataProcessor
    public String process1(String s) {
        return s.trim();
    }

    @DataProcessor
    public String process2(String s) {
        return s.toUpperCase();
    }
}

public class Main {
    public static void main(String[] args) throws Exception {
        PrintWriter pw = new PrintWriter("input.txt");
        pw.println("apple");
        pw.println("  banana  ");
        pw.println("sliva");
        pw.println("apple");
        pw.close();

        DataManager manager = new DataManager();

        manager.registerDataProcessor(new MyProcessor());

        manager.loadData("input.txt");
```



```
96          manager.loadData("input.txt");
97
98          manager.processData();
99
100         manager.saveData("output.txt");
101     }
102 }
```

Вывод: В ходе работы были рассмотрены аннотации как инструмент добавления метаданных к элементам программы, что позволяет управлять поведением кода на этапах компиляции и выполнения. Также изучен Stream API, предоставляющий декларативный и эффективный способ обработки коллекций данных с помощью промежуточных и терминальных операций. Использование Stream API способствует написанию более чистого, компактного и производительного кода в Java.