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 лабораторной работе №8

Выполнил студент группы:

БПИ2401

Мещеряков Кирилл Владимироврич

Руководитель:

Харрасов Камиль Раисович

Москва, 2025

Цель работы

Изучить возможности языка Java по обработке данных с использованием Stream API, аннотаций и Reflection API, а также освоить принципы динамического вызова методов и организации конвейерной обработки данных.

Задание

Разработать приложение, которое:

- считывает данные из файла;
- обрабатывает их с использованием Stream API;
- использует пользовательскую аннотацию для маркировки методов обработки;
- применяет Reflection API для поиска и вызова методов;
- сохраняет результат обработки в файл.

Используемые технологии

- Java SE
- Stream API
- Аннотации
- Reflection API
- ExecutorService (многопоточность)

Описание решения

Приложение построено вокруг класса **DataManager**, который управляет загрузкой данных, регистрацией обработчиков, их выполнением и сохранением результатов. Обработчики данных представляют собой отдельные классы, содержащие методы, помеченные аннотацией @DataProcessor.

Каждый метод обработки:

- принимает список объектов Person;
- выполняет операцию над данными (фильтрация, трансформация, подсчёт);

- возвращает обновлённый список.

Порядок выполнения методов определяется приоритетом, указанным в аннотации.

Структура данных

Для хранения информации используется класс Person, содержащий имя, возраст и зарплату.

Листинг кода

Аннотация DataProcessor

```
@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD)

public @interface DataProcessor {

String description() default "";

int priority() default 0;

}
```

Класс DataManager (фрагмент)

```
public class DataManager {

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

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


public void registerDataProcessor(Object processor) {

processors.add(processor);

}


public void processData() {

List<MethodWithPriority> annotatedMethods = new ArrayList<>();
```

```
for (Object processor : processors) {
for (Method method : processor.getClass().getDeclaredMethods()) {
if (method.isAnnotationPresent(DataProcessor.class)) {
DataProcessor dp = method.getAnnotation(DataProcessor.class);
annotatedMethods.add(new MethodWithPriority(
method, processor, dp.priority(), dp.description()
));
}
}
}

annotatedMethods.sort((a, b) -> Integer.compare(b.priority, a.priority));

for (MethodWithPriority info : annotatedMethods) {
try {
data = (List<Person>) info.method.invoke(info.processor, data);
} catch (Exception e) {
e.printStackTrace();
}
}
}
}
```

Пример обработчика данных

```
public class AgeProcessor {
```

```
@DataProcessor(description = "Увеличение возраста на 1 год", priority = 30)
public List<Person> incrementAge(List<Person> people) {
return people.stream()
.map(p -> new Person(
p.getName(),
p.getAge() + 1,
p.getSalary()))
.toList();
}
}
```

Результаты работы

В результате выполнения лабораторной работы было создано приложение, позволяющее гибко подключать новые обработчики данных без изменения основной логики программы. Использование Stream API позволило упростить код обработки коллекций, а аннотации и Reflection API обеспечили расширяемость и модульность решения.

Контрольные вопросы

1. Что такое аннотации?

Аннотации — это специальные метки в Java, которые добавляют метаданные к классам, методам или полям и могут быть прочитаны во время выполнения программы.

2. Для чего нужны аннотации?

Аннотации используются для маркировки элементов программы, управления поведением фреймворков, валидации данных и динамического анализа кода.

3. Как создать собственную аннотацию?

Собственная аннотация создаётся с помощью ключевого слова @interface, а также аннотаций @Retention и @Target.

4. Для чего можно использовать Stream API?

Stream API применяется для обработки коллекций данных: фильтрации, преобразования, сортировки, агрегации и параллельной обработки.

5. С помощью каких инструментов в Java можно обрабатывать данные параллельно?

Параллельную обработку данных можно выполнять с помощью ExecutorService, Thread, ForkJoinPool и parallelStream().

6. В чем разница между Collection и Stream?

Collection хранит данные, а Stream предназначен для их обработки и не хранит элементы.

7. Как создать Stream из коллекции? Массива? Отдельных элементов?

- Из коллекции: collection.stream()
- Из массива: Arrays.stream(array)
- Из отдельных элементов: Stream.of(a, b, c)

8. Что такое Optional в Stream API? Как обрабатывать возможные null значения?

Optional — это контейнер, который может содержать значение или быть пустым. Для обработки используются методы orElse, orElseGet, ifPresent.

9. Какие существуют терминальные операции Stream API?

К терминальным операциям относятся: forEach, collect, reduce, count, findFirst, anyMatch, allMatch, average, sum.

Вывод

В ходе лабораторной работы были изучены современные возможности Java для обработки данных. Использование Stream API, аннотаций и Reflection API позволяет создавать гибкие, расширяемые и читаемые приложения.

Ссылка на GitHub репозиторий: https://github.com/M1ke0-0/ITiP