Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

ЛАБОРАТОРНА РОБОТА № 12

з дисципліни «Основи програмування»

Тема: Stream API

Виконали:

Студентки групи ІА-31

Горлач Д., Макасєєва М.,

Соколова П.

Перевірив:

Степанов А.

Київ-2023

Тема: Stream API

Мета: Ознайомитись з наданими нам матеріалами, пригадати інформацію надану на лекціях з даної теми, також навчившись з минулих лабораторних робіт правильно використовувати знання та реалізовувати за допомогою них завдання, виконати поставлену нам задачу. А саме, скористатись новими знаннями та реалізувати за допомогою них подане завдання.

Хід роботи:

В цій лабораторній роботі використовуємо Stream API для розробки у стилі функціонального програмування. Не використовуючи жодних циклів типу for, while або перевірок за допомогою if чи switch зробимо обробку колекцій на сортування, ліміт, пропуск, фільтрацію і перетворення з одного типу об'єкта на інший в середині Stream API.

Для першого завдання створюємо клас Applicant(абітурієнт). Треба отримати список абітурієнтів, які не потрапили на бюджет, але які можуть потрапити на контракт. Створюємо потік об'єктів Applicant і фільтруємо всіх, хто має балів більше або дорівнює ніж 60 за допомогою метода filter і перевіркою кожного абітурієнта на кількість балів більша або дорівнює ніж 60, потім сортуємо по балах за допомогою метода sorted і статичного методу компаратора comparingInt і передаємо туди бали абітурієнтів для сортування, після чого викликаємо метод reverse, щоб отримати відсортований список від більшого до меншого, потім пропускаємо кількість абітурієнтів, які потрапляють на бюджет за допомогою метода skip, потім отримуємо тільки ту кількість на контракт, яка нам потрібна за допомогою методу limit, і термінальним методом collect, в який передаємо статичний метод toList класу Collectors, щоб отримати типізований список об'єктів Applicant.

Для другого завдання створюємо класи Institute(Інститут), Faculty(Факультет), Student(Студент). Треба отримати список студентів, які відсортовані за абеткою по прізвищах, у разі ідентичності прізвищ - по іменам, а у разі ідентичності імен - за номером залікової книжки. Створюємо потік об'єкту Institute, в конструктор якого передаємо список з об'єктів Faculty, у свою чергу які при створенні в конструктор отримують список з об'єктів Student. Замінюємо потік інститутів(в нашому випадку він один)на потік списків його інститутів за допомогою методу map і посиланням на метод getFaculties, потім із потоку списків інститутів перетворюємо на потік інститутів за допомогою методу flatMap, в який передаємо метод stream інтерфейсу Collection, потім отримуємо потік списків студентів за допомогою методу map і посиланням на метод getStudents, потім із потоку списків студентів перетворюємо на потік студентів за допомогою методу flatMap, в який передаємо метод stream інтерфейсу Collection, потім сортуємо студентів за допомогою методу sort і передаємо статичний метод comparing класу Comparator і порівнюємо спочатку по прізвищу за допомогою посилання на метод getLastName, наступним викликаємо метод у компаратора thenComparing і порівнюємо за іменем за допомогою посилання на метод getFirstName, наступним знову викликаємо метод у компаратора thenComparing, для того, щоб порівняти в останню чергу по номеру залікової книжки за допомогою посилання на метод getScoredBookNumber, і термінальним методом collect, в який передаємо статичний метод toList класу Collectors, щоб отримати типізований список об'єктів Student.

Для третього завдання використовуємо клас Applicant(Абітурієнт) із першого завдання. Треба отримати список абітурієнтів, які не можу бути зараховані в інститут (кількість балів менша ніж 60). Створюємо потік об'єктів Applicant і фільтруємо всіх, хто має балів менше ніж 60 за допомогою метода filter і перевіркою кожного абітурієнта на кількість балів менша ніж 60, потім

замінюємо потік абітурієнтів на потік їх прізвищ за допомогою методу map і посиланням на метод getLastName, і термінальним методом collect, в який передаємо статичний метод toList класу Collectors, щоб отримати типізований список об'єктів String(прізвища студентів).

2. Виконати завдання 1-3 відповідно до свого варіанту у таблиці 1. Для цього:
- проаналізувати завдання;
- створити зазначенні класи та тестові дані;
- усі списки мають бути типізованими (наприклад, ArrayList<Student>, а не просто ArrayList);
- завдання слід виконувати використовуючи функціональний стиль програмування (дозволяється використовувати Stream API, лямбда вирази та посилання на методи; заборонено використовувати такі конструкції як if, switch, for, while).

Рис. 1 завдання

1.2. Абітурієнтів, які не потрапили на «бюджет», але які можуть бути зараховані на «контракт» на дану кафедру (студенти, які не потрапили в список з п.1.1, M студентів з найвищими балами, де M – кількість місць на «контракт», кількість балів >= 60)

2.1. Список усіх студентів інституту (відсортований за абеткою по прізвищам, у разі ідентичності прізвищ – по іменам, а у разі ідентичності імен – за номером залікової книжки)

Завдання №3.
Для абітурієнтів із завдання №1, які не можуть бути зараховані в інститут (кількість балів < 60) отримати список:
3.1. Список абітурієнтів **List<Enrollee>**;
3.2. Список прізвищ абітурієнтів **List<String>**.

Рис. 2 завдання 1.2, 2.1, 3.2 відповідно

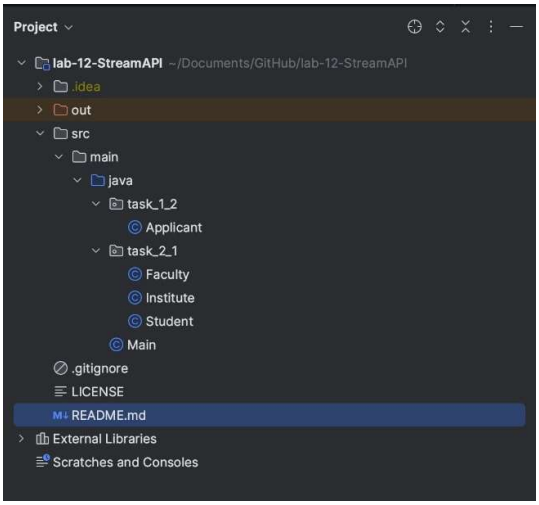
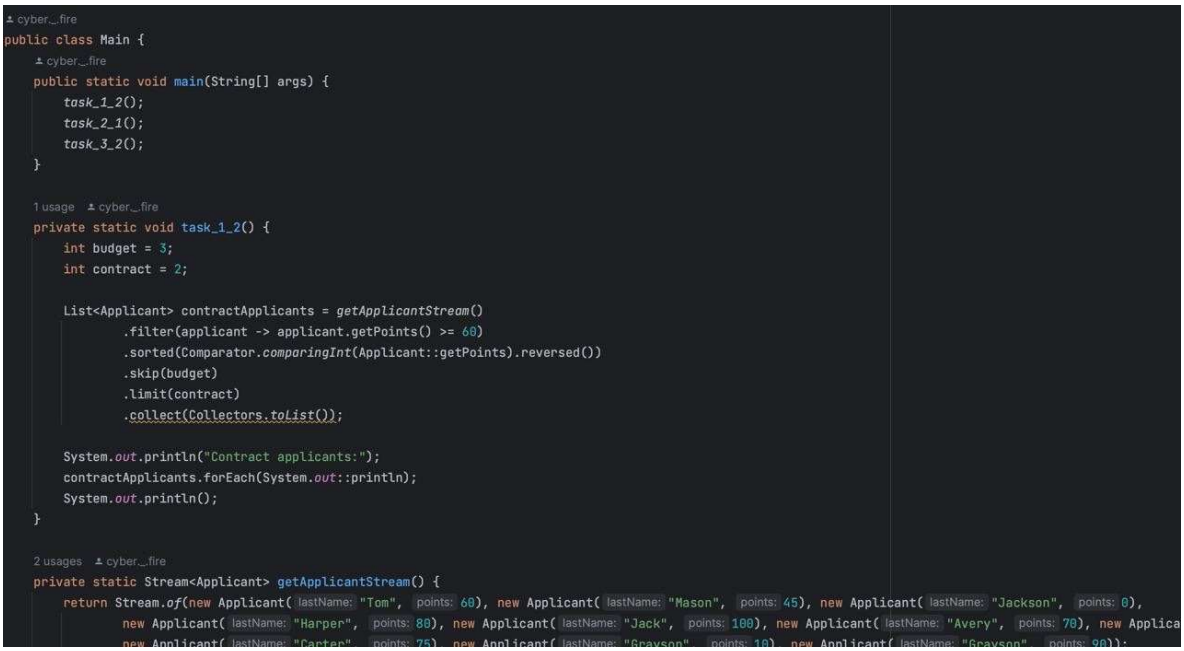


Рис 3. Структура програми



```java
cyber._.fire
public class Main {
    cyber._.fire
    public static void main(String[] args) {
        task_1_2();
        task_2_1();
        task_3_2();
    }

    1 usage  cyber._.fire
    private static void task_1_2() {
        int budget = 3;
        int contract = 2;

        List<Applicant> contractApplicants = getApplicantStream()
                .filter(applicant -> applicant.getPoints() >= 60)
                .sorted(Comparator.comparingInt(Applicant::getPoints).reversed())
                .skip(budget)
                .limit(contract)
                .collect(Collectors.toList());

        System.out.println("Contract applicants:");
        contractApplicants.forEach(System.out::println);
        System.out.println();
    }

    2 usages  cyber._.fire
    private static Stream<Applicant> getApplicantStream() {
        return Stream.of(new Applicant( lastName: "Tom",  points: 60), new Applicant( lastName: "Mason",  points: 45), new Applicant( lastName: "Jackson",  points: 0),
                new Applicant( lastName: "Harper",  points: 80), new Applicant( lastName: "Jack",  points: 100), new Applicant( lastName: "Avery",  points: 70), new Applica
                new Applicant( lastName: "Carter",  points: 75), new Applicant( lastName: "Grayson",  points: 10), new Applicant( lastName: "Grayson",  points: 90));
```

Рис. 4 Main

```
Contract applicants:
Applicant{lastName='Carter', points=75}
Applicant{lastName='Avery', points=70}

Sorted students:
Student{firstName='firstName1', lastName='lastName1', scoreBookNumber=10, averageScore=20}
Student{firstName='firstName2', lastName='lastName2', scoreBookNumber=30, averageScore=40}
Student{firstName='firstName3', lastName='lastName3', scoreBookNumber=10, averageScore=20}
Student{firstName='firstName4', lastName='lastName4', scoreBookNumber=150, averageScore=10}
Student{firstName='firstName5', lastName='lastName5', scoreBookNumber=10, averageScore=20}
Student{firstName='firstName6', lastName='lastName6', scoreBookNumber=30, averageScore=40}
Student{firstName='firstName7', lastName='lastName7', scoreBookNumber=10, averageScore=20}
Student{firstName='firstName8', lastName='lastName8', scoreBookNumber=150, averageScore=10}

Not included applicants last names:
Mason
Jackson
Grayson

Process finished with exit code 0
```

Рис. 5 output

Висновок:

StreamAPI (Stream Application Programming Interface) це прикладний потоковий інтерфейс, який є засобом роботи з потоками даних. Під поняттям "потік даних" мається на увазі канал передачі даних. Для потоку даних

визначається поняття джерела даних. Джерелами даних можуть бути масив, колекція, список тощо. Потік даних оперує цими джерелами. Потік даних ще можна визначити як послідовність об’єктів. Операції над потоками даних виконуються на основі формування відповідних запитів. Робота з засобами Stream API базується на використанні лямбда-виразів. У самому потоці дані не зберігаються, а тільки переміщуються при їх обробці (фільтрування, сортування, модифікація, пошуку тощо). При обробці потоку даних джерело даних не змінюється, це означає, що при сортуванні даних створюється новий відсортований потік даних, а початкове джерело залишається не сортоване.