**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 1

**З дисципліни:** *“Кросплатформне програмування”*

**На тему:** *“* *Робота з колекціями”*

**Лектор:**

доц. каф. ПЗ

Дяконюк Л. М.

**Виконав:**

ст. гр. ПЗ-35

Хруставчук М.Л.

**Прийняв:**

ст. викл. каф. ПЗ

Шкраб Р. Р.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2024 р.

∑= \_\_\_\_\_ .

Львів – 2024

**Тема роботи:** Робота з колекціями.

**Мета роботи:** Навчитися працювати з колекціями в мові програмування Java.

**TЕОРЕТИЧНІ ВІДОМОСТІ**

*Колекції*:

Це об'єкти, які використовуються для зберігання групи елементів. Вони надають стандартизований спосіб роботи з даними, спрощуючи такі завдання, як додавання, видалення, сортування та пошук елементів. У Java колекції є частиною бібліотеки Java Collections Framework (JCF).

Типи колекцій:

* List:
  + ArrayList: Динамічний масив, який дозволяє довільний доступ до елементів. Швидкий для читання, але повільний для вставки, або видалення елементів всередині списку.
  + LinkedList: Реалізація на основі зв’язного списку.яка забезпечує ефективне додавання і видалення елементів в будь-якому місці, але доступ до елементів вимагає більше часу.
* Set:
  + - HashSet: Використовує хешування для забезпечення унікальності елементів і швидкого доступу.
    - TreeSet: Підтримує елементи у відсортованому порядку.
* Map:
  + HashMap: Використовує хешування для зберігання пар ключ-значення. Не гарантує порядку елементів.
  + TreeMap: Зберігає елементи у відсортованому за ключем порядку.

*Stream API:*

Надає інструменти для обробки даних у функціональному стилі. Це дозволяє виконувати такі операції як фільтрація, сортування, мапінг і агрегація даних над колекціями. Потоки не змінюють оригінальні колекції, а створюють нові послідовності оброблених даних.

Приклади використання:

* filter(): Фільтрує елементи за умовою.
* map(): Перетворює кожен елемент колекції.
* collect(): Збирає результати у нову колекцію.

*Optional:*

Клас Optional використовується для обгортання об'єкта, щоб уникнути можливості появи NullPointerException*.* Він допомагає безпечно працювати з відсутніми значеннями.

Основні методи:

* isPresent(): Перевіряє, чи містить об'єкт значення.
* ifPresent(): Виконує дію, якщо значення присутнє.
* orElse(): Повертає значення за замовчуванням, якщо об'єкт порожній.

**ЗАВДАННЯ**

Кожен варіант завдань передбачає використання різних видів колекцій. Покажіть 2 види реалізацій кожного з підзавданням, а саме – з використанням технології Stream.api та без складних операцій з Stream API та безпечну обробку даних через Optional.

Варіант 3: Управління студентськими гуртожитками

Опис: У вас є система обліку студентів, що проживають у гуртожитках. Кожен студент має:

* Ім'я (String)
* Прізвище (String)
* Гуртожиток (String)
* Номер кімнати (int)
* Плату за проживання (double)
* Вік (int)
* Деякі студенти є пільговиками, і мають право на знижку на проживання.

Завдання:

1. Розділити студентів на тих, хто є пільговиками, і тих, хто не є.
2. Згрупувати студентів за гуртожитками.
3. Створити нову колекцію, в якій для кожної кімнати вказати кількість студентів, що там проживають.
4. Відсортувати студентів за віком та кількістю пільг.
5. Вивести список всіх унікальних номерів кімнат.
6. Знайти студента, який сплачує найбільшу плату за проживання, і обробити відсутність таких студентів через Optional.

**ХІД ВИКОНАННЯ**

**1. Код розробленої програми**

Main.java

package org.example;  
  
import java.util.\*;  
  
public class Main {  
 public static void main(String[] args) {  
 List<Student> students = Arrays.*asList*(  
 new Student("Володимир", "Валла", "Гуртожиток №1", 89, 1000, 20, false),  
 new Student("Андрій", "Домчак", "Гуртожиток №3", 52, 1500, 18, false),  
 new Student("Дмитро", "Лесик", "Гуртожиток №7", 300, 1200, 17, true),  
 new Student("Катерина", "Стасенко", "Гуртожиток №1", 528, 1000, 19, true),  
 new Student("Роман", "Ферко", "Гуртожиток №3", 52, 1000, 20, true),  
 new Student("Максим", "Хруставчук", "Гуртожиток №1", 528, 1000, 19, false)  
 );  
 Scanner scanner = new Scanner(System.*in*);  
  
 // Виводимо опитування для користувача  
 System.*out*.println("Оберіть варіант: ");  
 System.*out*.println("1 - Використовувати Stream API");  
 System.*out*.println("2 - Використовувати звичайні методи без Stream API\n");  
 int choice = scanner.nextInt();  
  
 if (choice == 1) {  
 System.*out*.println("Ви вибрали Stream API:");  
 StreamAPI.*divideBeneficiaryAndNonBeneficiaryStudents*(students);  
 StreamAPI.*groupStudentsByDormitory*(students);  
 StreamAPI.*countStudentsInEachRoom*(students);  
 StreamAPI.*sortStudentsByAgeAndBeneficiary*(students);  
 StreamAPI.*allUniqueRoomNumbers*(students);  
 StreamAPI.*mostAccommodationFee*(students);  
 } else if (choice == 2) {  
 System.*out*.println("Ви вибрали звичайні методи:");  
 StudentManager.*divideBeneficiaryAndNonBeneficiaryStudents*(students);  
 StudentManager.*groupStudentsByDormitory*(students);  
 StudentManager.*countStudentsInEachRoom*(students);  
 StudentManager.*sortStudentsByAgeAndBeneficiary*(students);  
 StudentManager.*allUniqueRoomNumbers*(students);  
 StudentManager.*mostAccommodationFee*(students);  
 } else {  
 System.*out*.println("Неправильний вибір! Будь ласка, спробуйте ще раз.");  
 }  
  
 scanner.close();  
 }  
}

StreamAPI.java

package org.example;  
import java.util.\*;  
import java.util.stream.Collectors;  
  
public class StreamAPI  
{  
 static public void divideBeneficiaryAndNonBeneficiaryStudents(List<Student> students)  
 {  
 System.*out*.println("Студенти з пільгою: ");  
 students.stream()  
 .filter(Student::isBeneficiary)  
 .forEach(student -> System.*out*.println(student));  
  
 System.*out*.println("Студенти без пільги: ");  
 students.stream()  
 .filter(student -> !student.isBeneficiary())  
 .forEach(student -> System.*out*.println(student));  
 }  
  
 static public void groupStudentsByDormitory(List<Student> students)  
 {  
 Map<String, List<Student>> groupedByDormitory= students.stream()  
 .collect(Collectors.*groupingBy*(Student::getDormitory));  
  
 groupedByDormitory.forEach((dormitory, studentsInDormitory) -> {  
 System.*out*.println("Гуртожиток: " + dormitory);  
 studentsInDormitory.forEach(student -> System.*out*.println(" " + student));  
 });  
 }  
  
 static public void countStudentsInEachRoom(List<Student> students){  
 Map <Integer, Long> studentsPerRoom =  
 students.stream()  
 .collect(Collectors.*groupingBy*(Student::getRoomNumber, Collectors.*counting*()));  
  
 studentsPerRoom.forEach((roomNumber, count) ->  
 System.*out*.println("Кімната №" + roomNumber + ": " + count + " студент(ів)"));  
 }  
  
 static public void sortStudentsByAgeAndBeneficiary(List<Student> students){  
 System.*out*.println("\nСтуденти відсортовані за віком та наявністю пільг:");  
 students.stream()  
 .sorted(Comparator.*comparing*(Student::getAge).thenComparing(Student::isBeneficiary))  
 .forEach(student -> System.*out*.println(student));  
 }  
  
 static public void allUniqueRoomNumbers(List<Student> students){  
 Set<Integer> uniqueRoomNumbers = students.stream()  
 .map(Student::getRoomNumber)  
 .collect(Collectors.*toSet*());  
 System.*out*.println("Унікальні номери кімнат: " + uniqueRoomNumbers);  
 }  
  
 static public void mostAccommodationFee(List<Student> students){  
 Optional<Student> studentWithMaxFee = students.stream()  
 .max(Comparator.*comparing*(Student::getAccommodationFee));  
  
 studentWithMaxFee.ifPresentOrElse(  
 student -> System.*out*.println("Студент з найбільшою платою: \n" + student),  
 () -> System.*out*.println("Немає студентів у списку"));  
 }  
}

Student.java

package org.example;  
public class Student {  
 private String name;  
 private String surname;  
 private String dormitory;  
 private int roomNumber;  
 private double accommodationFee;  
 private int age;  
 private boolean isBeneficiary;  
  
 public Student(String name, String surname, String dormitory, int room, double price, int age, boolean isBeneficiary) {  
 this.name = name;  
 this.surname = surname;  
 this.dormitory = dormitory;  
 this.roomNumber = room;  
 this.accommodationFee = price;  
 this.age = age;  
 this.isBeneficiary = isBeneficiary;  
 }  
  
 public String getFirstName(){  
 return name;  
 }  
 public String getLastName() {  
 return surname;  
 }  
 public String getDormitory() {  
 return dormitory;  
 }  
 public int getRoomNumber(){  
 return roomNumber;  
 }  
 public double getAccommodationFee() {  
 return accommodationFee;  
 }  
 public int getAge(){  
 return age;  
 }  
 public boolean isBeneficiary() {  
 return isBeneficiary;  
 }  
  
 private double calculateFee() {  
 return isBeneficiary ? (accommodationFee \* 0.5) : accommodationFee;  
 }  
  
 @Override  
 public String toString(){  
 return "\tСтудент: " + name + " " + surname +  
 ", Гуртожиток: " + dormitory +  
 ", Кімната: " + roomNumber +  
 ", Плата за проживання: " + calculateFee() +  
 ", Вік: " + age +  
 ", Пільговик: " + (isBeneficiary ? "Так" : "Ні");  
 }}

StudentManager.java

package org.example;  
import java.util.\*;  
  
public class StudentManager {  
 static public void divideBeneficiaryAndNonBeneficiaryStudents(List<Student> students) {  
 System.*out*.println("Студенти з пільгою: ");  
 for (Student student : students) {  
 if (student.isBeneficiary()) {  
 System.*out*.println(student);  
 }  
 }  
  
 System.*out*.println("Студенти без пільги: ");  
 for (Student student : students) {  
 if (!student.isBeneficiary()) {  
 System.*out*.println(student);  
 }  
 }  
 }  
  
 static public void groupStudentsByDormitory(List<Student> students) {  
 Map<String, List<Student>> groupedByDormitory = new HashMap<>();  
  
 // Групування студентів за гуртожитком  
 for (Student student : students) {  
 groupedByDormitory  
 .computeIfAbsent(student.getDormitory(), k -> new ArrayList<>())  
 .add(student);  
 }  
  
 // Виведення груп  
 for (Map.Entry<String, List<Student>> entry : groupedByDormitory.entrySet()) {  
 System.*out*.println("Гуртожиток: " + entry.getKey());  
 for (Student student : entry.getValue()) {  
 System.*out*.println(" " + student);  
 }  
 }  
 }  
  
 static public void countStudentsInEachRoom(List<Student> students) {  
 Map<Integer, Integer> studentsPerRoom = new HashMap<>();  
  
 // Підрахунок кількості студентів у кожній кімнаті  
 for (Student student : students) {  
 int room = student.getRoomNumber();  
 studentsPerRoom.put(room, studentsPerRoom.getOrDefault(room, 0) + 1);  
 }  
  
 // Виведення кількості студентів по кімнатах  
 for (Map.Entry<Integer, Integer> entry : studentsPerRoom.entrySet()) {  
 System.*out*.println("Кімната №" + entry.getKey() + ": " + entry.getValue() + " студент(ів)");  
 }  
 }  
  
 static public void sortStudentsByAgeAndBeneficiary(List<Student> students) {  
 System.*out*.println("\nСтуденти відсортовані за віком та наявністю пільг:");  
  
 // Сортування за віком і пільгою  
 Collections.*sort*(students, new Comparator<Student>() {  
 @Override  
 public int compare(Student s1, Student s2) {  
 int ageComparison = Integer.*compare*(s1.getAge(), s2.getAge());  
 if (ageComparison != 0) {  
 return ageComparison;  
 }  
 return Boolean.*compare*(s1.isBeneficiary(), s2.isBeneficiary());  
 }  
 });  
  
 // Виведення відсортованих студентів  
 for (Student student : students) {  
 System.*out*.println(student);  
 }  
 }  
  
 static public void allUniqueRoomNumbers(List<Student> students) {  
 Set<Integer> uniqueRoomNumbers = new HashSet<>();  
  
 // Збирання унікальних номерів кімнат  
 for (Student student : students) {  
 uniqueRoomNumbers.add(student.getRoomNumber());  
 }  
  
 // Виведення унікальних номерів  
 System.*out*.println("Унікальні номери кімнат: " + uniqueRoomNumbers);  
 }  
  
 static public void mostAccommodationFee(List<Student> students) {  
 if (students.isEmpty()) {  
 System.*out*.println("Немає студентів у списку");  
 return;  
 }  
  
 // Пошук студента з найбільшою платою  
 Student studentWithMaxFee = students.get(0);  
 for (Student student : students) {  
 if (student.getAccommodationFee() > studentWithMaxFee.getAccommodationFee()) {  
 studentWithMaxFee = student;  
 }  
 }  
  
 System.*out*.println("Студент з найбільшою платою: \n" + studentWithMaxFee);  
 }  
}

**2. Результати виконання розробленої програми**

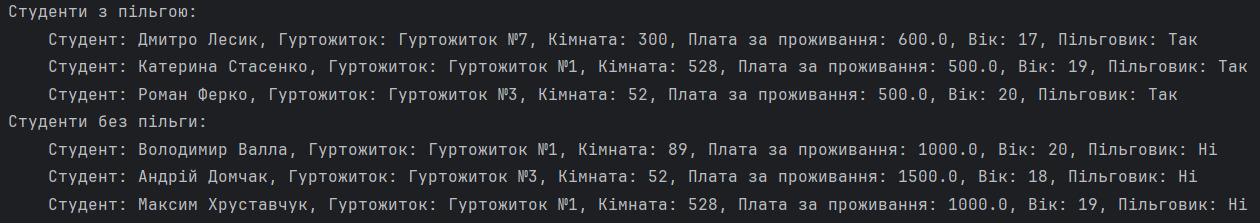
****

Рис. 1. Розподіл студентів на пільговиків і не пільговиків

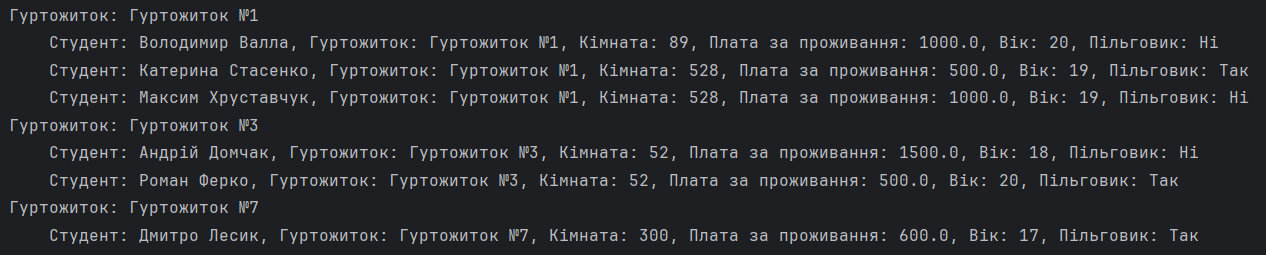


Рис. 2. Студети згруповані за гуртожитками

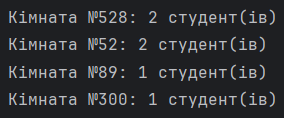


Рис. 3. Інформація про кількість студентів у кімнаті

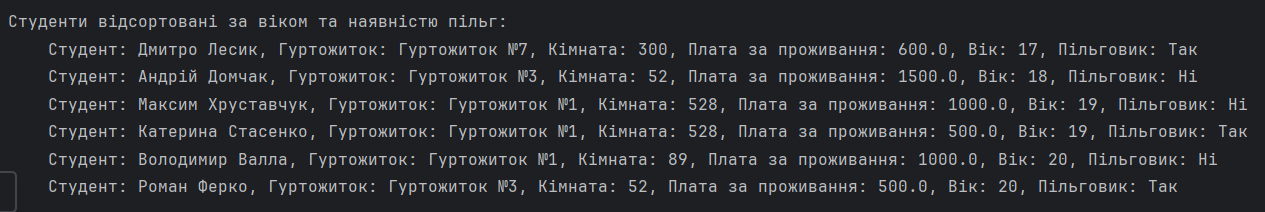


Рис. 4. Студенти відсортовані за віком та наявністю пільг



Рис. 5. Усі унікальні номери кімнат

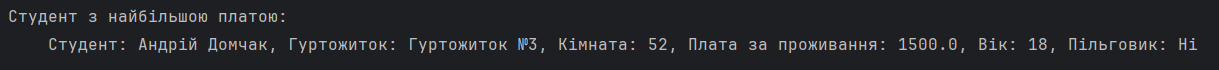


Рис. 6. Студент, який сплачує найбільшу плату за проживання

**ВИСНОВКИ**

У ході виконання лабораторної роботи я навчився працювати з колекціями в мові програмування Java. Розглянув основні типи колекцій, такі як List, Set та Map, а також їхні популярні реалізації, включаючи ArrayList, HashSet та HashMap. Було вивчено принципи роботи з колекціями, зокрема операції додавання, видалення, сортування та пошуку елементів. Особлива увага була приділена використанню StreamAPI для функціональної обробки даних та класу Optional для безпечної роботи з можливими відсутніми значеннями.