**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

до лабораторної роботи №1

**на тему:** *“Робота з колекціями”*

**з дисципліни** *“Кросплатформне програмування”*

**Лектор:**

доц. каф. ПЗ

Дяконюк Л. М.

**Виконав:**

ст. гр. ПЗ-33

Юшкевич. А.І.

**Прийняв:**

доц. каф. ПЗ

Баштовий А. В.

«\_\_» \_\_\_\_\_\_\_\_\_ 2024р.

∑=\_\_\_\_\_

Львів – 2024

**Тема.** Робота з колекціями.

**Мета.** Навчитись користуватись колекціями y Java із використанням StreamApi та без.

# Теоретичні відомості

* **Основні типи колекцій**:
  + Множини (**Set**)
  + Списки (**List**)
  + Черги (**Queue**)
  + Асоційовані масиви (**Map**)
* **Інтерфейси**
  + дозволяють маніпулювати колекціями незалежно від типу реалізації
* **Реалізації** інтерфейсів колекції
  + надаються різноманітні реалізації для колекцій різних типів. Наприклад для колекцій типу List надаються реалізації ArrayList, LinkedList тощо
* **Алгоритми** пошуку, сортування тощо
  + для колекцій
  + для масивів
* **Пакети**
  + Пакет **java.util**
* Пакет **java.util.concurrent**
* Основні методи інтерфейсу **Collection**

add – додавання елементу

addAll – додавання колекції елементів

clear() – очистка колекції

contains(…) – чи містить колекція вказаний елемент

containsAll(…) – чи містить колекція усі вказані елементи

remove(…) – видалити із колекції вказаний елемент

removeAll(…) – видалити із колекції усі вказані елементи

retainAll(…) – залишити в колекції тільки вказані елементи

size() – кількість елементів

isEmpty() – чи порожня колекція

toArray() – перетворення у масив

* contains, containsAll, remove, removeAll, retainAll

При виконанні цих методів вибір елементів колекції - на основі методу **equals** для елементів

**Завдання до лабораторної роботи**

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 5: Спортивні результати**

Опис: Потрібно створити систему обліку спортивних змагань.

**Кожен спортсмен має:**

* Ім'я (String)
* Прізвище (String)
* Вид спорту (String)
* Кількість медалей (int)
* Вік (int)

Деякі спортсмени можуть представляти кілька видів спорту.

**Виконайте наступні завдання:**

1. Розділити спортсменів на тих, хто представляє кілька видів спорту, і тих, хто представляє один.
2. Згрупувати спортсменів за видами спорту.
3. Створити колекцію, яка покаже, скільки спортсменів у кожному виді спорту мають більше 5 медалей.
4. Відсортувати спортсменів за кількістю медалей та віком.
5. Вивести список всіх унікальних видів спорту.
6. Знайти спортсмена з найбільшою кількістю медалей в конкретному виді спорту або повернути повідомлення про його відсутність (Optional).

# Хід роботи

***Athlete.java***

import java.util.Set;  
  
class Athlete {  
 private String firstName;  
 private String lastName;  
 private Set<String> sports;  
 private int medals;  
 private int age;  
  
 public Athlete(String firstName, String lastName, Set<String> sports, int medals, int age) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.sports = sports;  
 this.medals = medals;  
 this.age = age;  
 }  
  
 // Getters  
 public String getFirstName() { return firstName; }  
 public String getLastName() { return lastName; }  
 public Set<String> getSports() { return sports; }  
 public int getMedals() { return medals; }  
 public int getAge() { return age; }  
}

***SportsCompetitionStreamApi.java***

import java.util.\*;  
import java.util.stream.\*;  
  
public class SportsCompetitionStreamApi {  
 private List<Athlete> athletes;  
  
 public SportsCompetitionStreamApi(List<Athlete> athletes) {  
 this.athletes = athletes;  
 }  
  
 public Map<Boolean, List<Athlete>> splitBySportsCount() {  
 return athletes.stream().collect(Collectors.*partitioningBy*(  
 athlete -> athlete.getSports().size() > 1  
 ));  
 }  
  
 public Map<String, List<Athlete>> groupBySport() {  
 return athletes.stream().flatMap(athlete ->  
 athlete.getSports().stream().map(sport ->  
 new AbstractMap.SimpleEntry<>(sport, athlete))  
 ).collect(Collectors.*groupingBy*(Map.Entry::getKey,  
 Collectors.*mapping*(Map.Entry::getValue, Collectors.*toList*())));  
 }  
  
 public Map<String, Long> countAthletesWithMoreThanFiveMedals() {  
 return athletes.stream()  
 .filter(athlete -> athlete.getMedals() > 5)  
 .flatMap(athlete -> athlete.getSports().stream()  
 .map(sport -> new AbstractMap.SimpleEntry<>(sport, 1)))  
 .collect(Collectors.*groupingBy*(Map.Entry::getKey, Collectors.*counting*()));  
 }  
  
 public List<Athlete> sortAthletesByMedalsAndAge() {  
 return athletes.stream()  
 .sorted(Comparator.*comparingInt*(Athlete::getMedals).reversed()  
 .thenComparingInt(Athlete::getAge))  
 .collect(Collectors.*toList*());  
 }  
  
 public Set<String> getUniqueSports() {  
 return athletes.stream()  
 .flatMap(athlete -> athlete.getSports().stream())  
 .collect(Collectors.*toSet*());  
 }  
  
 public String findTopAthleteInSport(String sport) {  
 return athletes.stream()  
 .filter(a -> a.getSports().contains(sport))  
 .max(Comparator.*comparingInt*(Athlete::getMedals))  
 .map(a -> a.getFirstName() + " " + a.getLastName() + " with " + a.getMedals() + " medals")  
 .orElse("No athlete found in " + sport);  
 }  
}

***SportsCompetitionNoStreamApi.java***

import java.util.\*;  
  
public class SportsCompetitionNoStreamApi {  
 private List<Athlete> athletes;  
  
 public SportsCompetitionNoStreamApi(List<Athlete> athletes) {  
 this.athletes = athletes;  
 }  
  
 public Map<Boolean, List<Athlete>> splitBySportsCount() {  
 Map<Boolean, List<Athlete>> result = new HashMap<>();  
 result.put(true, new ArrayList<>()); // Multiple sports  
 result.put(false, new ArrayList<>()); // Single sport  
  
 for (Athlete athlete : athletes) {  
 if (athlete.getSports().size() > 1) {  
 result.get(true).add(athlete);  
 } else {  
 result.get(false).add(athlete);  
 }  
 }  
 return result;  
 }  
  
 public Map<String, List<Athlete>> groupBySport() {  
 Map<String, List<Athlete>> sportGroups = new HashMap<>();  
 for (Athlete athlete : athletes) {  
 for (String sport : athlete.getSports()) {  
 sportGroups.computeIfAbsent(sport, k -> new ArrayList<>()).add(athlete);  
 }  
 }  
 return sportGroups;  
 }  
  
 public Map<String, Integer> countAthletesWithMoreThanFiveMedals() {  
 Map<String, Integer> countMap = new HashMap<>();  
 for (Athlete athlete : athletes) {  
 if (athlete.getMedals() > 5) {  
 for (String sport : athlete.getSports()) {  
 countMap.put(sport, countMap.getOrDefault(sport, 0) + 1);  
 }  
 }  
 }  
 return countMap;  
 }  
  
 public List<Athlete> sortAthletesByMedalsAndAge() {  
 athletes.sort(Comparator.*comparingInt*(Athlete::getMedals).reversed()  
 .thenComparingInt(Athlete::getAge));  
 return athletes;  
 }  
  
 public Set<String> getUniqueSports() {  
 Set<String> uniqueSports = new HashSet<>();  
 for (Athlete athlete : athletes) {  
 uniqueSports.addAll(athlete.getSports());  
 }  
 return uniqueSports;  
 }  
  
 public String findTopAthleteInSport(String sport) {  
 Athlete topAthlete = null;  
 for (Athlete athlete : athletes) {  
 if (athlete.getSports().contains(sport)) {  
 if (topAthlete == null || athlete.getMedals() > topAthlete.getMedals()) {  
 topAthlete = athlete;  
 }  
 }  
 }  
 return Optional.*ofNullable*(topAthlete)  
 .map(a -> a.getFirstName() + " " + a.getLastName() + " with " + a.getMedals() + " medals")  
 .orElse("No athlete found in " + sport);  
 }  
}

***Program.java***

import java.util.\*;  
  
public class Program {  
 public static void main(String[] args) {  
 Set<String> sports1 = new HashSet<>(Arrays.*asList*("Basketball"));  
 Set<String> sports2 = new HashSet<>(Arrays.*asList*("Football"));  
 Set<String> sports3 = new HashSet<>(Arrays.*asList*("Volleyball"));  
 Set<String> sports4 = new HashSet<>(Arrays.*asList*("Football", "Basketball"));  
 Set<String> sports5 = new HashSet<>(Arrays.*asList*("Football", "Volleyball"));  
 Set<String> sports6 = new HashSet<>(Arrays.*asList*("Basketball", "Volleyball"));  
 Set<String> sports7 = new HashSet<>(Arrays.*asList*("Football", "Basketball", "Volleyball"));  
  
 List<Athlete> athletes = Arrays.*asList*(  
 new Athlete("John", "Doe", sports1, 8, 25),  
 new Athlete("Jane", "Smith", sports2, 3, 30),  
 new Athlete("Alex", "Brown", sports3, 6, 22),  
 new Athlete("Michael", "Johnson", sports4, 10, 27),  
 new Athlete("Emily", "Davis", sports5, 4, 28),  
 new Athlete("Sarah", "Wilson", sports6, 5, 26),  
 new Athlete("James", "Taylor", sports7, 12, 32),  
 new Athlete("Oliver", "Miller", sports1, 7, 21),  
 new Athlete("Sophia", "Jones", sports2, 2, 24),  
 new Athlete("Liam", "Garcia", sports3, 9, 29),  
 new Athlete("Amelia", "Martinez", sports4, 6, 23),  
 new Athlete("Lucas", "Rodriguez", sports5, 8, 31),  
 new Athlete("Mia", "Hernandez", sports6, 5, 26),  
 new Athlete("Noah", "Lopez", sports7, 14, 30),  
 new Athlete("Ava", "Gonzalez", sports1, 6, 22),  
 new Athlete("Mason", "Wilson", sports2, 3, 27),  
 new Athlete("Isabella", "Anderson", sports3, 7, 24),  
 new Athlete("Ethan", "Thomas", sports4, 11, 29),  
 new Athlete("Charlotte", "Jackson", sports5, 5, 25),  
 new Athlete("William", "White", sports6, 4, 30),  
 new Athlete("Henry", "Harris", sports7, 13, 28),  
 new Athlete("Evelyn", "Clark", sports1, 9, 23),  
 new Athlete("Lily", "Lewis", sports2, 4, 26),  
 new Athlete("Jack", "Walker", sports3, 6, 32),  
 new Athlete("Elijah", "Hall", sports4, 8, 31),  
 new Athlete("Grace", "Young", sports5, 2, 27),  
 new Athlete("Logan", "King", sports6, 7, 28),  
 new Athlete("Chloe", "Wright", sports7, 10, 33),  
 new Athlete("Benjamin", "Scott", sports1, 3, 24),  
 new Athlete("Zoe", "Green", sports2, 5, 31)  
 );  
  
 SportsCompetitionStreamApi usingAPI = new SportsCompetitionStreamApi(athletes);  
 SportsCompetitionNoStreamApi noUsingAPI = new SportsCompetitionNoStreamApi(athletes);  
 }  
}

**Висновок:** Протягом виконання цієї лабораторної роботи я навчився користуватись колекціями у Java із використанням StreamApi та без. Також дослідив роботу з класом Optional.