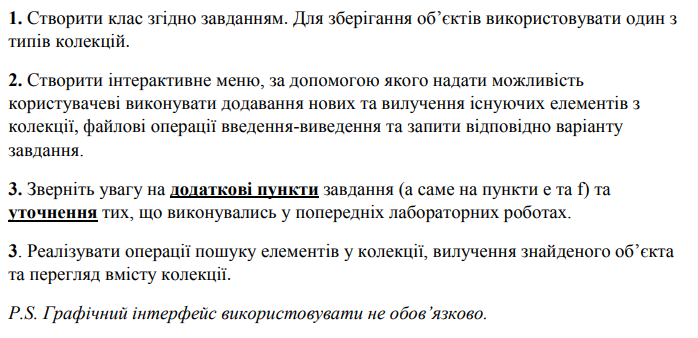
**Лабораторна робота №1**

Розробка та реалізація програм із використанням колекцій для зберігання даних та файлового введення-виведення.

**Мета роботи:** отримати навички створення та реалізації програм, що використовують колекції для зберігання та опрацювання даних та реалізують операції введення-виведення із файлами

*Завдання*



**Рішення**

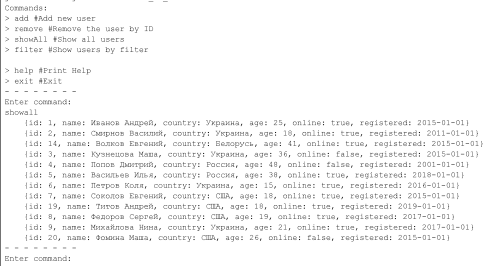
1. **Текст програми**

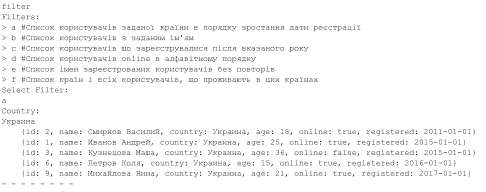
***Lab1.java:***

**public class** Lab1 {  
 **private static final** String ***USERS\_STORAGE\_URL*** = **"src/com/labs/lab\_s4\_1/users"**;  
  
 **private** HashSet<User> **users** = **new** HashSet<>();  
  
 **public static void** main(String[] args) {  
 **var** lab = **new** Lab1();  
 lab.run();  
 }  
  
 **public void** run() {  
  
 **try** {  
 loadUsers();  
 } **catch** (IOException | ClassNotFoundException err) {  
 System.***out***.println(**"Can't load users from file"**);  
 }  
  
 runCLI();  
 }  
  
 **private void** runCLI() {  
 printHelp();  
 *hr*();  
  
 **while** (**true**) {  
 String command = *promptLine*(**"Enter command: "**).trim();  
  
 **if** (command.equalsIgnoreCase(**"exit"**)) {  
 **break**;  
  
 } **else** {  
 useCommand(command);  
  
 }  
  
 *hr*();  
 }  
  
 }  
  
 **private void** useCommand(String command) {  
 **if** (command.equalsIgnoreCase(**"add"**)) {  
 addUser();  
  
 } **else if** (command.equalsIgnoreCase(**"remove"**)) {  
 removeElement();  
  
 } **else if** (command.equalsIgnoreCase(**"showAll"**)) {  
 printUsers(**users**);  
  
 } **else if** (command.equalsIgnoreCase(**"filter"**)) {  
 showUsersByFilter();  
  
 } **else if** (command.equalsIgnoreCase(**"help"**)) {  
 printHelp();  
  
 } **else** {  
 System.***out***.println(**"Command not found. Try again: "**);  
 }  
 }  
  
 **private void** addUser() {  
 **var** user = createNewUser();  
  
 **boolean** check = *promptBool*(**"Are you really want to add the user:\n"** + user);  
 **if** (!check) **return**;  
  
 **users**.add(user);  
  
 **try** {  
 saveUsers();  
 System.***out***.println(user);  
 } **catch** (IOException err) {  
 **users**.remove(user);  
 System.***out***.println(**"Can't save users"**);  
 }  
 }  
  
 **private void** removeElement() {  
 String id = *promptLine*(**"Enter user id: "**);  
  
 **var** found = **users**.stream().filter((user -> user.getId().equals(id))).findFirst();  
  
 **if** (!found.isPresent()) {  
 System.***out***.println(**"No user have the id"**);  
 **return**;  
 }  
  
 **var** user = found.get();  
  
 **boolean** check = *promptBool*(**"Are you really want to remove user:\n"** + user);  
 **if** (!check) **return**;;  
  
 **users**.removeIf((item) -> item.getId().equals(id));  
  
 **try** {  
 saveUsers();  
 System.***out***.println(**"Successfully"**);  
 } **catch** (IOException err) {  
 System.***out***.println(**"Can't apply changes"**);  
 }  
 }  
  
 **private void** showUsersByFilter() {  
 printFiltersHelp();  
 String filter = *promptLine*(**"Select Filter:"**);  
  
 **if** (filter.equalsIgnoreCase(**"a"**)) {  
 String country = *promptLine*(**"Country:"**);  
  
 **var** result = **users**.stream().filter(user -> user.getCountry().equalsIgnoreCase(country))  
 .sorted(Comparator.*comparing*(User::getRegistered, LocalDate::compareTo))  
 .collect(Collectors.*toCollection*(ArrayList::**new**));  
  
 printUsers(result);  
  
 } **else if** (filter.equalsIgnoreCase(**"b"**)) {  
 String name = *promptLine*(**"Name:"**);  
  
 **var** result = **users**.stream().filter(user -> user.getFirstName().equalsIgnoreCase(name))  
 .collect(Collectors.*toCollection*(ArrayList::**new**));  
  
 printUsers(result);  
  
 } **else if** (filter.equalsIgnoreCase(**"c"**)) {  
 **int** year = *promptInt*(**"Year:"**);  
  
 **var** result = **users**.stream().filter(user -> user.getRegistered().getYear() > year)  
 .collect(Collectors.*toCollection*(ArrayList::**new**));  
  
 printUsers(result);  
  
 } **else if** (filter.equalsIgnoreCase(**"d"**)) {  
 **var** result = **users**.stream().filter(User::isOnline)  
 .sorted(Comparator.*comparing*(User::getFullName, String::compareTo))  
 .collect(Collectors.*toCollection*(ArrayList::**new**));  
  
 printUsers(result);  
  
 } **else if** (filter.equalsIgnoreCase(**"e"**)) {  
 **var** result = **users**.stream().map(User::getFirstName)  
 .collect(Collectors.*toCollection*(HashSet::**new**));  
  
 System.***out***.println(result);  
  
 } **else if** (filter.equalsIgnoreCase(**"f"**)) {  
 **var** countries = **users**.stream().map(User::getCountry)  
 .collect(Collectors.*toCollection*(HashSet::**new**));  
  
 HashMap<String, HashSet<User>> map = **new** HashMap<>();  
  
 countries.forEach((country) -> {  
 HashSet<User> countryUsers = **users**.stream()  
 .filter(user -> user.getCountry().equals(country))  
 .collect(Collectors.*toCollection*(HashSet::**new**));  
  
 map.put(country, countryUsers);  
 });  
  
 map.forEach((country, countyUsers) -> {  
 System.***out***.println(**" - "** + country);  
 printUsers(countyUsers);  
 });  
  
 } **else** {  
 System.***out***.println(**"Can't find the filter"**);  
  
 }  
 }  
  
 **private void** printFiltersHelp() {  
 System.***out***.println(**"Filters:"**);  
 System.***out***.println(**"> a #Список користувачів заданої країни в порядку зростання дати реєстрації"**);  
 System.***out***.println(**"> b #Список користувачів з заданим ім'ям"**);  
 System.***out***.println(**"> c #Список користувачів що зареєструвалися після вказаного року"**);  
 System.***out***.println(**"> d #Список користувачів online в алфавітному порядку"**);  
 System.***out***.println(**"> e #Список імен зареєстрованих користувачів без повторів"**);  
 System.***out***.println(**"> f #Список країн і всіх користувачів, що проживають в цих країнах"**);  
 }  
  
 **private void** printHelp() {  
 System.***out***.println(**"Commands:"**);  
 System.***out***.println(**"> add #Add new user"**);  
 System.***out***.println(**"> remove #Remove the user by ID"**);  
 System.***out***.println(**"> showAll #Show all users"**);  
 System.***out***.println(**"> filter #Show users by filter"**);  
 System.***out***.println();  
 System.***out***.println(**"> help #Print Help"**);  
 System.***out***.println(**"> exit #Exit"**);  
 }  
  
 **private** User createNewUser() {  
 **return new** User(  
 *promptLine*(**"Id:"**),  
 *promptLine*(**"First Name:"**),  
 *promptLine*(**"Last Name:"**),  
 *promptLine*(**"Country"**),  
 *promptInt*(**"Age:"**),  
 *promptBool*(**"Online:"**),  
 LocalDate.*of*(*promptInt*(**"Registered year:"**), 1, 1)  
 );  
 }  
  
 **private void** printUsers(Collection<User> users) {  
 **if** (users.isEmpty()) {  
 System.***out***.println(**"There are no users"**);  
 **return**;  
 }  
  
 users.forEach((item) -> System.***out***.println(**"\t"** + item));  
 }  
  
 */\*\*  
 \* Загружает данные users из файла  
 \* <b>users при этом перезаписывается данными из файла</b>  
 \*/* **private void** loadUsers() **throws** IOException, ClassNotFoundException {  
 **try**(FileInputStream fis = **new** FileInputStream(***USERS\_STORAGE\_URL***);  
 ObjectInputStream ois = **new** ObjectInputStream(fis)) {  
  
 **users** = (HashSet<User>)ois.readObject();  
 }  
 }  
  
 */\*\*  
 \* Записывает коллекцию users в файл  
 \*/* **private void** saveUsers() **throws** IOException {  
 **try** (FileOutputStream fos = **new** FileOutputStream(***USERS\_STORAGE\_URL***);  
 ObjectOutputStream out = **new** ObjectOutputStream(fos)) {  
  
 out.writeObject(**users**);  
 }  
 }  
}

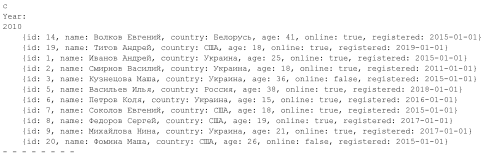
***User.java:***  
**public class** User **implements** Comparable<User>, Serializable {  
 **private** String **id**;  
 **private** String **firstName**;  
 **private** String **lastName**;  
 **private** String **country**;  
 **private boolean online**;  
 **private** LocalDate **registered**;  
 **private int age**;   
  
 **public** User(String id, String firstName, String lastName) {  
 setId(id);  
 setFirstName(firstName);  
 setLastName(lastName);  
 }  
  
 **public** User(String id, String firstName, String lastName, String country, **int** age, **boolean** online, LocalDate registered) {  
 setId(id);  
 setFirstName(firstName);  
 setLastName(lastName);  
 setCountry(country);  
 setOnline(online);  
 setRegistered(registered);  
 setAge(age);  
 }  
  
 **public** String getFullName() {  
 **return lastName** + **" "** + **firstName**;  
 }   
 **public** String toString() {  
 **return** String.*format*(  
 **"{id: %s, name: %s, country: %s, age: %d, online: %b, registered: %s}"**,  
 **id**, getFullName(), **country**, **age**, **online**, **registered** );  
 }   
 **public boolean** equals(Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (!(o **instanceof** User)) **return false**;  
 User user = (User) o;  
 **return id**.equals(user.**id**);  
 }   
 **public int** hashCode() {  
 **return** Objects.*hash*(**id**);  
 }  
 **public** String getId() {**return id**;}  
 **public void** setId(String id) {**this**.**id** = id;}  
 **public** String getFirstName() {**return firstName**;}  
 **public void** setFirstName(String firstName) {**this**.**firstName** = firstName;}  
 **public** String getLastName() {**return lastName**;}  
 **public void** setLastName(String lastName) {**this**.**lastName** = lastName;}  
 **public** String getCountry() {**return country**;}  
 **public void** setCountry(String country) {**this**.**country** = country;}  
 **public boolean** isOnline() {**return online**;}  
 **public void** setOnline(**boolean** online) {**this**.**online** = online;}  
 **public** LocalDate getRegistered() {**return registered**;}  
 **private void** setRegistered(LocalDate registered) {**this**.**registered** = registered;}  
 **public int** getAge() {**return age**;}  
 **public void** setAge(**int** age) {**this**.**age** = age;}  
  
 **public int** compareTo(@NotNull User o) {  
 **return** Comparator.*comparing*(User::getId, String::compareTo).compare(**this**, o);  
 }  
}

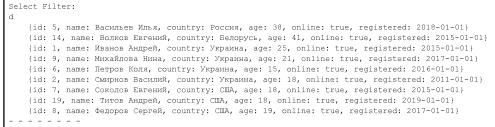
1. **Результат виконання роботи**



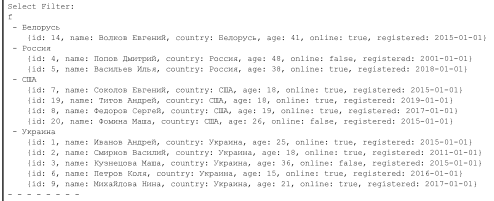












**Висновок:** на цій лабораторній роботі я отримав навички створення та реалізації програм, що використовують колекції для зберігання та опрацювання даних та реалізують операції введення-виведення із файлами.