МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра «ПРОГРАМНОЇ ІНЖЕНЕРІЇ ТА ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ УПРАВЛІННЯ»

ЗВІТ

з лабораторної роботи № 4

з навчальної дисципліни

«основи програмування Java (двв)»

ВИКОНАВ

Студент групи

ПЕРЕВІРИВ

доц каф. ПІІТУ к.т.н.

доцент Валерій ВОЛОВЩИКОВ

Харків 2022

**Тема:** Робота з винятками та файлами в Java.

**Завдання №1. Індивідуальне завдання**

Спроектувати та реалізувати класи для представлення сутностей попередньої лабораторної роботи. Рішення повинне базуватися на раніше створеній ієрархії класів. Слід створити абстрактний клас, похідний від класу, який представляє основну сутність. Цей клас, повинен декларувати відповідні операції читання і запису і реалізувати можливості демонстрації взаємодії з файлами різних форматів.

Слід також створити два похідних класи від класу. Один клас повинен бути доповненим можливостями створення текстового файлу, читання даних з цього текстового файлу та запису цих даних в інший файл після сортування. Другий клас повинен реалізовувати створення бінарного файлу даних, читання даних з цього файлу даних та запису цих даних в інший файл дних після сортування.

Окрім роботи з файлами повинно бути реалізоване виведення результатів у консольне вікно.

Мій варіант: **12**

Перша сутність: лікар.

Обов’язкові поля для першої сутності: прізвище, стаж.

Друга сутність: прийом.

Обов’язкові поля для другої сутності: день, кількість відвідувачів, коментар.

**Набір тестових даних завдання №1:**

Створення екземпляру doctor класу DoctorWithArrayList (surname: “TestSurname”, workExperience: 5, appointments: Appointment{day: “01.11”, visitors: 3, comment: “A”}, Appointment{ day: “02.11”, visitors: 1, comment: “C”}, Appointment{ day: “03.11”, visitors: 2, comment: “B”}).

* **Тестування класу DoctorWithTextFileOperations**

Створення об’єкту txtDoctor класу DoctorWithTextFileOperations з передачею у конструктор об’єкту doctor (очікується створення об’єкту з такими ж значеннями полів як і об’єкта doctor).

Виклик методу test() класу DoctorWithFileOperations з параметрами об’єктом txtDoctor та filename “Doctor.txt”.

У методі test():

Очікується виведення даних про об’єкт doctor

*Doctor: TestSurname*

*Work experience: 5 years*

*Appointments:*

*Day: 01.11*

*Amount of visitors: 3*

*Comment: “A”*

*Day: 02.11*

*Amount of visitors: 1*

*Comment: “C”*

*Day: 03.11*

*Amount of visitors: 2*

*Comment: “B”*

та створення у кореневій теці проекту файлу “Doctor.txt”, який міститиме наступні дані:

*TestSurname*

*5*

*01.11*

*3*

*A*

*02.11*

*1*

*C*

*03.11*

*2*

*B*

Здійснення сортування об’єкту doctor за першою ознакою.

Очікується виведення даних про об’єкт doctor

*Sorted by amount of visitors:*

*Doctor: TestSurname*

*Work experience: 5 years*

*Appointments:*

*Day: 02.11*

*Amount of visitors: 1*

*Comment: "C"*

*Day: 03.11*

*Amount of visitors: 2*

*Comment: "B"*

*Day: 01.11*

*Amount of visitors: 3*

*Comment: "A"*

та створення у кореневій теці проекту файлу “FirstSortedDoctor.txt”, який міститиме наступні дані:

*TestSurname*

*5*

*02.11*

*1*

*C*

*03.11*

*2*

*B*

*01.11*

*3*

*A*

Здійснення сортування об’єкту doctor за другою ознакою.

Очікується виведення даних про об’єкт doctor

*Sorted by comment alphabetically:*

*Doctor: TestSurname*

*Work experience: 5 years*

*Appointments:*

*Day: 01.11*

*Amount of visitors: 3*

*Comment: "A"*

*Day: 03.11*

*Amount of visitors: 2*

*Comment: "B"*

*Day: 02.11*

*Amount of visitors: 1*

*Comment: "C"*

та створення у кореневій теці проекту файлу “SecondSortedDoctor.txt”, який міститиме наступні дані:

*TestSurname*

*5*

*01.11*

*3*

*A*

*03.11*

*2*

*B*

*02.11*

*1*

*C*

Читання даних з файлу “Doctor.txt” до об’єкту doctor.

Очікується виведення даних про об’єкт doctor

*Doctor: TestSurname*

*Work experience: 5 years*

*Appointments:*

*Day: 01.11*

*Amount of visitors: 3*

*Comment: “A”*

*Day: 02.11*

*Amount of visitors: 1*

*Comment: “C”*

*Day: 03.11*

*Amount of visitors: 2*

*Comment: “B”*

Читання даних з файлу “FirstSortedDoctor.txt” до об’єкту doctor.

Очікується виведення даних про об’єкт doctor

*Sorted by amount of visitors:*

*Doctor: TestSurname*

*Work experience: 5 years*

*Appointments:*

*Day: 02.11*

*Amount of visitors: 1*

*Comment: "C"*

*Day: 03.11*

*Amount of visitors: 2*

*Comment: "B"*

*Day: 01.11*

*Amount of visitors: 3*

*Comment: "A"*

Читання даних з файлу “SecondSortedDoctor.txt” до об’єкту doctor.

Очікується виведення даних про об’єкт doctor

*Sorted by comment alphabetically:*

*Doctor: TestSurname*

*Work experience: 5 years*

*Appointments:*

*Day: 01.11*

*Amount of visitors: 3*

*Comment: "A"*

*Day: 03.11*

*Amount of visitors: 2*

*Comment: "B"*

*Day: 02.11*

*Amount of visitors: 1*

*Comment: "C"*

* **Тестування класу DoctorWithDataFileOperations**

Створення об’єкту datDoctor класу DoctorWithDataFileOperations з передачею у конструктор об’єкту doctor (очікується створення об’єкту з такими ж значеннями полів як і об’єкта doctor).

Виклик методу test() класу DoctorWithFileOperations з параметрами об’єктом datDoctor та filename “Doctor.dat”.

Очікується виведення даних про об’єкт doctor

Doctor: TestSurname

Work experience: 5 years

Appointments:

Day: 01.11

Amount of visitors: 3

Comment: “A”

Day: 02.11

Amount of visitors: 1

Comment: “C”

Day: 03.11

Amount of visitors: 2

Comment: “B”

та створення у кореневій теці проекту файлу “Doctor.dat”, який міститиме дані про об’єкт doctor у вигляді послідовності бітів.

Здійснення сортування об’єкту doctor за першою ознакою.

Очікується виведення даних про об’єкт doctor

Sorted by amount of visitors:

Doctor: TestSurname

Work experience: 5 years

Appointments:

Day: 02.11

Amount of visitors: 1

Comment: "C"

Day: 03.11

Amount of visitors: 2

Comment: "B"

Day: 01.11

Amount of visitors: 3

Comment: "A"

та створення у кореневій теці проекту файлу “FirstSortedDoctor.dat”, який міститиме дані про об’єкт doctor у вигляді послідовності бітів.

Здійснення сортування об’єкту doctor за другою ознакою.

Очікується виведення даних про об’єкт doctor

Sorted by comment alphabetically:

Doctor: TestSurname

Work experience: 5 years

Appointments:

Day: 01.11

Amount of visitors: 3

Comment: "A"

Day: 03.11

Amount of visitors: 2

Comment: "B"

Day: 02.11

Amount of visitors: 1

Comment: "C"

та створення у кореневій теці проекту файлу “SecondSortedDoctor.dat”, який міститиме дані про об’єкт doctor у вигляді послідовності бітів.

Читання даних з файлу “Doctor.dat” до об’єкту doctor.

Очікується виведення даних про об’єкт doctor

Doctor: TestSurname

Work experience: 5 years

Appointments:

Day: 01.11

Amount of visitors: 3

Comment: “A”

Day: 02.11

Amount of visitors: 1

Comment: “C”

Day: 03.11

Amount of visitors: 2

Comment: “B”

Читання даних з файлу “FirstSortedDoctor.dat” до об’єкту doctor.

Очікується виведення даних про об’єкт doctor

Sorted by amount of visitors:

Doctor: TestSurname

Work experience: 5 years

Appointments:

Day: 02.11

Amount of visitors: 1

Comment: "C"

Day: 03.11

Amount of visitors: 2

Comment: "B"

Day: 01.11

Amount of visitors: 3

Comment: "A"

Читання даних з файлу “SecondSortedDoctor.dat” до об’єкту doctor.

Очікується виведення даних про об’єкт doctor

Sorted by comment alphabetically:

Doctor: TestSurname

Work experience: 5 years

Appointments:

Day: 01.11

Amount of visitors: 3

Comment: "A"

Day: 03.11

Amount of visitors: 2

Comment: "B"

Day: 02.11

Amount of visitors: 1

Comment: "C"

**Програмний код до завдання №1:**

Клас Appointment

package task1;  
  
import java.util.Objects;  
  
*/\*\*  
 \* Клас для представлення другої сутності (Прийом)  
 \*/*class Appointment implements Comparable {  
 private String day;  
 private int visitors;  
 private String comment;  
 public Appointment(String day, int visitors, String comment) {  
 this.day = day;  
 this.visitors = visitors;  
 this.comment = comment;  
 }  
 public String getDay() {  
 return day;  
 }  
 public void setDay(String day) {  
 this.day = day;  
 }  
 public int getVisitors() {  
 return visitors;  
 }  
 public void setVisitors(int visitors) {  
 this.visitors = visitors;  
 }  
 public String getComment() {  
 return comment;  
 }  
 public void setComment(String comment) {  
 this.comment = comment;  
 }  
  
 @Override  
 public String toString() {  
 return String.*format*("\tDay: %s\n\tAmount of visitors: %d\n\tComment: \"%s\"", day, visitors, comment);  
 }  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj) return true;  
 if (obj == null || getClass() != obj.getClass()) return false;  
 Appointment appointment = (Appointment) obj;  
 return Objects.*equals*(visitors, appointment.visitors) && Objects.*equals*(day, appointment.day) && Objects.*equals*(comment, appointment.comment);  
 }  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(day, visitors, comment);  
 }  
  
 */\*\*  
 \* Перевизначення методу compareTo() інтерфейсу Comparable для природного порівняння об'єктів за кількістю відвідувачів на прийомі  
 \*  
 \** ***@param*** *o об'єкт для порівняння  
 \** ***@return*** *від'ємне ціле значення, нуль, або додатне ціле значення, якщо об'єкт, для якого викликаний метод менший, дорівнює, або більший за o  
 \*/* @Override  
 public int compareTo(Object o) {  
 Appointment a = (Appointment) o;  
 return Integer.*compare*(this.visitors, a.visitors);  
 }  
  
 public static void test() {  
 Appointment a = new Appointment("01.11", 1, "A");  
  
 // тестування методу toString()  
 System.*out*.println(a);  
  
 // тестування методу equals()  
 Appointment a2 = new Appointment("01.11", 1, "A");  
 System.*out*.println(a.equals(a2));  
 a2.setVisitors(2);  
 System.*out*.println(a.equals(a2));  
  
 // тестування методу hashCode()  
 System.*out*.printf("\nHashcode: %d\n", a.hashCode());  
  
 // тестування методу compareTo()  
 System.*out*.println(a.compareTo(a2));  
 System.*out*.println(a2.compareTo(a));  
 a2.setVisitors(1);  
 System.*out*.println(a.compareTo(a2));  
 }  
}

Клас AbstractDoctor

package task1;  
  
import java.util.Arrays;  
import java.util.Objects;  
  
*/\*\*  
 \* Абстрактний клас для представлення першої сутності (Лікар)  
 \*/*abstract class AbstractDoctor {  
 public abstract String getSurname();  
 public abstract void setSurname(String surname);  
 public abstract int getWorkExperience();  
 public abstract void setWorkExperience(int workExperience);  
 public abstract Appointment[] getAppointments();  
 public abstract void setAppointments(Appointment[] appointments);  
 public abstract Appointment getAppointment(int index);  
 public abstract void setAppointment(Appointment appointment, int index);  
  
 @Override  
 public String toString() {  
 String info = String.*format*("Doctor: %s\nWork experience: %d years\nAppointments:\n", getSurname(), getWorkExperience());  
 for (int i = 0; i < getAppointments().length; i++) {  
 info += getAppointment(i) + "\n";  
 }  
 return info;  
 }  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj) return true;  
 if (obj == null || getClass() != obj.getClass()) return false;  
 AbstractDoctor ad = (AbstractDoctor) obj;  
 return Objects.*equals*(this.getSurname(), ad.getSurname()) && Objects.*equals*(this.getWorkExperience(), ad.getWorkExperience()) && Arrays.*equals*(this.getAppointments(), ad.getAppointments());  
 }  
 @Override  
 public int hashCode() {  
 int result = Objects.*hash*(this.getSurname(), this.getWorkExperience());  
 result = 31 \* result + Arrays.*hashCode*(this.getAppointments());  
 return result;  
 }  
  
 */\*\*  
 \* Метод, що реалізує основне завдання (середня кількість відвідувачів)  
 \*  
 \** ***@return*** *середня кількість відвідувачів на всіх прийомах цього лікаря  
 \*/* public double getAverageVisitors() {  
 int sum = 0;  
 for (int i = 0; i < getAppointments().length; i++) {  
 sum += getAppointment(i).getVisitors();  
 }  
 return (double) sum / (double) getAppointments().length;  
 }  
 */\*\*  
 \* Метод, що реалізує основне завдання (прийом з найменшою кількістю відвідувачів)  
 \*  
 \** ***@return*** *прийом у цього лікаря, на якому була найменша кількість відвідувачів  
 \*/* public Appointment getAppointmentWithMinVisitors() {  
 Appointment min = getAppointments()[0];  
 for (int i = 0; i < getAppointments().length; i++) {  
 if (getAppointment(i).getVisitors() < min.getVisitors()) {  
 min = getAppointment(i);  
 }  
 }  
 return min;  
 }  
 */\*\*  
 \* Метод, що реалізує основне завдання (прийом з найдовшим коментарем)  
 \*  
 \** ***@return*** *прийом з найдовшим коментарем  
 \*/* public Appointment getAppointmentWithLongestComment() {  
 Appointment longestComment = getAppointments()[0];  
 for (int i = 0; i < getAppointments().length; i++) {  
 if (getAppointment(i).getComment().length() > longestComment.getComment().length()) {  
 longestComment = getAppointment(i);  
 }  
 }  
 return longestComment;  
 }  
  
 */\*\*  
 \* Метод, що здійснює сортування за першою ознакою (За збільшенням кількості відвідувачів) методом сортування бульбашкою  
 \*/* public void sortByVisitorsAscending() {  
 boolean mustSort = true;  
 while (mustSort) {  
 mustSort = false;  
 for (int i = 1; i < getAppointments().length; i++) {  
 if (getAppointment(i).getVisitors() < getAppointment(i - 1).getVisitors()) {  
 Appointment temp = getAppointment(i);  
 setAppointment(getAppointment(i - 1), i);  
 setAppointment(temp, i - 1);  
 mustSort = true;  
 }  
 }  
 }  
 }  
 */\*\*  
 \* Метод, що здійснює сортування за другою ознакою (За алфавітом коментаря) методом сортування включенням  
 \*/* public void sortByCommentAlphabetically() {  
 for (int left = 0; left < getAppointments().length; left++) {  
 Appointment value = getAppointment(left);  
 int i = left - 1;  
 for (; i >= 0; i--) {  
 if (value.getComment().compareTo(getAppointment(i).getComment()) < 0) {  
 setAppointment(getAppointment(i), i + 1);  
 } else {  
 break;  
 }  
 }  
 setAppointment(value, i + 1);  
 }  
 }  
  
 public static void test(){  
 // клас, що успадковує абстрактний клас для того, щоб його протестувати  
 class AbstractDoctorTestClass extends AbstractDoctor {  
 private String surname;  
 private int workExperience;  
 private Appointment[] appointments;  
 public AbstractDoctorTestClass(String surname, int workExperience, Appointment[] appointments) {  
 this.surname = surname;  
 this.workExperience = workExperience;  
 this.appointments = appointments;  
 }  
 @Override  
 public String getSurname() {  
 return surname;  
 }  
 @Override  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
 @Override  
 public int getWorkExperience() {  
 return workExperience;  
 }  
 @Override  
 public void setWorkExperience(int workExperience) {  
 this.workExperience = workExperience;  
 }  
 @Override  
 public Appointment[] getAppointments() {  
 return appointments;  
 }  
 @Override  
 public void setAppointments(Appointment[] appointments) {  
 this.appointments = appointments;  
 }  
 @Override  
 public Appointment getAppointment(int index) {  
 return appointments[index];  
 }  
 @Override  
 public void setAppointment(Appointment appointment, int index) {  
 appointments[index] = appointment;  
 }  
 }  
  
 AbstractDoctorTestClass t = new AbstractDoctorTestClass("TestSurname", 5, new Appointment[]{  
 new Appointment("01.11", 3, "A"),  
 new Appointment("02.11", 1, "C"),  
 new Appointment("03.11", 2, "BB")  
 });  
  
 // тестування методу toString()  
 System.*out*.println(t);  
  
 // тестування методу equals()  
 AbstractDoctorTestClass t2 = new AbstractDoctorTestClass("TestSurname", 5, new Appointment[]{  
 new Appointment("01.11", 3, "A"),  
 new Appointment("02.11", 1, "C"),  
 new Appointment("03.11", 2, "BB")  
 });  
 System.*out*.println(t.equals(t2));  
 t2.setWorkExperience(0);  
 System.*out*.println(t.equals(t2));  
  
 // тестування методу hashCode()  
 System.*out*.printf("\nHashcode: %d\n", t.hashCode());  
  
 System.*out*.printf("Average amount of visitors on appointment: %.2f\n", t.getAverageVisitors());  
 System.*out*.printf("Appointment with minimum amount of visitors:\n%s\n", t.getAppointmentWithMinVisitors());  
 System.*out*.printf("Appointment with longest comment:\n%s\n", t.getAppointmentWithLongestComment());  
  
 // тестування методів, що здійснюють сортування за ознаками  
 t.sortByVisitorsAscending();  
 System.*out*.printf("\nSorted by amount of visitors on appointments:\n%s", t);  
 t.sortByCommentAlphabetically();  
 System.*out*.printf("\nSorted by appointments comments alphabetically:\n%s", t);  
 }  
}

Клас DoctorWithArray

package task1;  
  
*/\*\*  
 \* Клас для представлення першої сутності (Лікар)  
 \*/*class DoctorWithArray extends AbstractDoctor {  
 private String surname;  
 private int workExperience;  
 private Appointment[] appointments;  
  
 public DoctorWithArray(String surname, int workExperience, Appointment[] appointments) {  
 this.surname = surname;  
 this.workExperience = workExperience;  
 this.appointments = appointments;  
 }  
  
 @Override  
 public String getSurname() {  
 return surname;  
 }  
 @Override  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
 @Override  
 public int getWorkExperience() {  
 return workExperience;  
 }  
 @Override  
 public void setWorkExperience(int workExperience) {  
 this.workExperience = workExperience;  
 }  
 @Override  
 public Appointment[] getAppointments() {  
 return appointments;  
 }  
 @Override  
 public void setAppointments(Appointment[] appointments) {  
 this.appointments = appointments;  
 }  
 @Override  
 public Appointment getAppointment(int index) {  
 return appointments[index];  
 }  
 @Override  
 public void setAppointment(Appointment appointment, int index) {  
 appointments[index] = appointment;  
 }  
  
 public static void test() {  
 DoctorWithArray t = new DoctorWithArray("TestSurname", 5, new Appointment[]{  
 new Appointment("01.11", 3, "A"),  
 new Appointment("02.11", 1, "C"),  
 new Appointment("03.11", 2, "BB")  
 });  
  
 // тестування методу toString()  
 System.*out*.println(t);  
  
 // тестування методу equals()  
 DoctorWithArray t2 = new DoctorWithArray("TestSurname", 5, new Appointment[]{  
 new Appointment("01.11", 3, "A"),  
 new Appointment("02.11", 1, "C"),  
 new Appointment("03.11", 2, "BB")  
 });  
 System.*out*.println(t.equals(t2));  
 t2.setWorkExperience(0);  
 System.*out*.println(t.equals(t2));  
  
 // тестування методу hashCode()  
 System.*out*.printf("\nHashcode: %d\n", t.hashCode());  
  
 System.*out*.printf("Average amount of visitors on appointment: %.2f\n", t.getAverageVisitors());  
 System.*out*.printf("Appointment with minimum amount of visitors:\n%s\n", t.getAppointmentWithMinVisitors());  
 System.*out*.printf("Appointment with longest comment:\n%s\n", t.getAppointmentWithLongestComment());  
  
 // тестування методів, що здійснюють сортування за ознаками  
 t.sortByVisitorsAscending();  
 System.*out*.printf("\nSorted by amount of visitors on appointments:\n%s", t);  
 t.sortByCommentAlphabetically();  
 System.*out*.printf("\nSorted by appointments comments alphabetically:\n%s", t);  
 }  
}

Клас DoctorWithArrayList

package task1;  
  
import java.util.\*;  
  
*/\*\*  
 \* Клас для представлення першої сутності (Лікар), у якому структурою зберігання даних про послідовність других сутностей  
 \* (прийомів) визначений список (ArrayList).  
 \*/*class DoctorWithArrayList extends AbstractDoctor {  
 private String surname;  
 private int workExperience;  
 private List<Appointment> appointments;  
  
 public DoctorWithArrayList(String surname, int workExperience, List<Appointment> appointments) {  
 this.surname = surname;  
 this.workExperience = workExperience;  
 this.appointments = appointments;  
 }  
  
 @Override  
 public String getSurname() {  
 return surname;  
 }  
 @Override  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
 @Override  
 public int getWorkExperience() {  
 return workExperience;  
 }  
 @Override  
 public void setWorkExperience(int workExperience) {  
 this.workExperience = workExperience;  
 }  
 @Override  
 public Appointment[] getAppointments() {  
 return appointments.toArray(new Appointment[0]);  
 }  
 @Override  
 public void setAppointments(Appointment[] appointments) {  
 this.appointments = new ArrayList<>(Arrays.*asList*(appointments));  
 }  
 @Override  
 public Appointment getAppointment(int index) {  
 return appointments.get(index);  
 }  
 @Override  
 public void setAppointment(Appointment appointment, int index) {  
 this.appointments.set(index, appointment);  
 }  
  
 */\*\*  
 \* Перевизначення методу сортування за першою ознакою (за збільшенням кількості відвідувачів) з використанням методу  
 \* sort() класу Collections. Позаяк у класі Appointment перевизначений метод compareTo() для порівняння об'єктів за  
 \* першою ознакою, сортування буде здійснюватись автоматично.  
 \*/* @Override  
 public void sortByVisitorsAscending() {  
 Collections.*sort*(appointments);  
 }  
 */\*\*  
 \* Перевизначення методу сортування за другою ознакою (за алфавітом коментаря) з використанням методу sort()  
 \* інтерфейсу List. Під час виклику методу comparing() інтерфейсу Comparator через посилання на метод явно вказується  
 \* ознака, за якою треба порівнювати об'єкти в ході сортування.  
 \*/* @Override  
 public void sortByCommentAlphabetically() {  
 appointments.sort(Comparator.*comparing*(Appointment::getComment));  
 }  
  
 public static void test() {  
 DoctorWithArrayList t = new DoctorWithArrayList("TestSurname", 5, Arrays.*asList*(  
 new Appointment("01.11", 3, "A"),  
 new Appointment("02.11", 1, "C"),  
 new Appointment("03.11", 2, "B")  
 ));  
  
 System.*out*.println(t);  
  
 t.sortByVisitorsAscending();  
 System.*out*.printf("\nSorted by amount of visitors on appointments:\n%s", t);  
  
 t.sortByCommentAlphabetically();  
 System.*out*.printf("\nSorted by appointments comments alphabetically:\n%s", t);  
 }  
}

Клас DoctorWithSortedSet

package task1;  
  
import java.util.Arrays;  
import java.util.Comparator;  
import java.util.SortedSet;  
import java.util.TreeSet;  
  
*/\*\*  
 \* Клас для представлення першої сутності (Лікар), у якому структурою зберігання даних про послідовність других сутностей  
 \* (прийомів) визначена множина (SortedSet).  
 \*/*public class DoctorWithSortedSet extends AbstractDoctor {  
 private String surname;  
 private int workExperience;  
 private SortedSet<Appointment> appointments;  
  
 public DoctorWithSortedSet(String surname, int workExperience, SortedSet<Appointment> appointments) {  
 this.surname = surname;  
 this.workExperience = workExperience;  
 this.appointments = appointments;  
 }  
  
 @Override  
 public String getSurname() {  
 return surname;  
 }  
 @Override  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
 @Override  
 public int getWorkExperience() {  
 return workExperience;  
 }  
 @Override  
 public void setWorkExperience(int workExperience) {  
 this.workExperience = workExperience;  
 }  
 @Override  
 public Appointment[] getAppointments() {  
 return appointments.toArray(new Appointment[0]);  
 }  
 @Override  
 public void setAppointments(Appointment[] appointments) {  
 this.appointments = new TreeSet<>(Arrays.*asList*(appointments));  
 }  
 @Override  
 public Appointment getAppointment(int index) {  
 return appointments.toArray(new Appointment[0])[index];  
 }  
 @Override  
 public void setAppointment(Appointment appointment, int index) {  
 this.appointments.toArray()[index] = appointment;  
 }  
  
 */\*\*  
 \* Перевизначення методу сортування за першою ознакою (за збільшенням кількості відвідувачів). Сортування здійснюється  
 \* шляхом присвоєння полю appointments нового об'єкту TreeSet, під час створення якого до методу comparing() інтерфейсу  
 \* Comparator через посилання на метод явно вказується ознака, за якою треба порівнювати об'єкти в ході сортування.  
 \*/* @Override  
 public void sortByVisitorsAscending() {  
 SortedSet<Appointment> temp = new TreeSet<>(Comparator.*comparing*(Appointment::getVisitors));  
 temp.addAll(appointments);  
 appointments = temp;  
 }  
  
 */\*\*  
 \* Перевизначення методу сортування за другою ознакою (за алфавітом коментаря). Сортування здійснюється шляхом присвоєння  
 \* полю appointments нового об'єкту TreeSet, під час створення якого до методу comparing() інтерфейсу Comparator через  
 \* посилання на метод явно вказується ознака, за якою треба порівнювати об'єкти в ході сортування.  
 \*/* @Override  
 public void sortByCommentAlphabetically() {  
 SortedSet<Appointment> temp = new TreeSet<>(Comparator.*comparing*(Appointment::getComment));  
 temp.addAll(appointments);  
 appointments = temp;  
 }  
  
 public static void test() {  
 DoctorWithSortedSet t = new DoctorWithSortedSet("TestSurname", 5, new TreeSet<>(Arrays.*asList*(  
 new Appointment("01.11", 3, "A"),  
 new Appointment("02.11", 1, "C"),  
 new Appointment("03.11", 2, "B")  
 )));  
  
 System.*out*.println(t);  
  
 t.sortByVisitorsAscending();  
 System.*out*.printf("\nSorted by amount of visitors on appointments:\n%s", t);  
  
 t.sortByCommentAlphabetically();  
 System.*out*.printf("\nSorted by appointments comments alphabetically:\n%s", t);  
 }  
}

Клас DoctorWithSorting

package task1;  
  
import java.util.Arrays;  
import java.util.Comparator;  
  
*/\*\*  
 \* Клас для представлення першої сутності (Лікар) з перевизначеними методами сортування за ознаками  
 \*/*class DoctorWithSorting extends DoctorWithArray {  
 public DoctorWithSorting(String surname, int workExperience, Appointment[] appointments) {  
 super(surname, workExperience, appointments);  
 }  
  
 */\*\*  
 \* Метод, що перевизначає сортування за першою ознакою (За збільшенням кількості відвідувачів)  
 \*/* @Override  
 public void sortByVisitorsAscending() {  
 Arrays.*sort*(getAppointments());  
 }  
  
 */\*\*  
 \* Метод, що перевизначає сортування за другою ознакою (За алфавітом коментаря)  
 \*/* @Override  
 public void sortByCommentAlphabetically() {  
 Arrays.*sort*(getAppointments(), new Comparator<Appointment>() {  
 @Override  
 public int compare(Appointment o1, Appointment o2) {  
 return o1.getComment().compareTo(o2.getComment());  
 }  
 });  
 }  
  
 public static void test() {  
 DoctorWithSorting t = new DoctorWithSorting("TestSurname", 5, new Appointment[]{  
 new Appointment("01.11", 3, "A"),  
 new Appointment("02.11", 1, "C"),  
 new Appointment("03.11", 2, "B")  
 });  
  
 // тестування методу toString()  
 System.*out*.printf("Unsorted:\n%s\n", t);  
  
 // тестування методів, що здійснюють сортування за ознаками  
 t.sortByVisitorsAscending();  
 System.*out*.printf("Sorted by amount of visitors on appointments:\n%s\n", t);  
 t.sortByCommentAlphabetically();  
 System.*out*.printf("Sorted by appointments comments alphabetically:\n%s\n", t);  
 }  
}

Клас DoctorWithFileOperations

package task1;  
  
public abstract class DoctorWithFileOperations extends DoctorWithArray {  
 public DoctorWithFileOperations(String surname, int workExperience, Appointment[] appointments) {  
 super(surname, workExperience, appointments);  
 }  
  
 public abstract void readFromFile(String fileName);  
  
 public abstract void writeToFile(String fileName);  
  
 public static void test(DoctorWithFileOperations doctor, String fileName) {  
 System.*out*.println(doctor);  
 doctor.writeToFile(fileName);  
  
 doctor.sortByVisitorsAscending();  
 System.*out*.println("Sorted by amount of visitors:\n" + doctor);  
 doctor.writeToFile("FirstSorted" + fileName);  
  
 doctor.sortByCommentAlphabetically();  
 System.*out*.println("Sorted by comment alphabetically:\n" + doctor);  
 doctor.writeToFile("SecondSorted" + fileName);  
  
 doctor.setSurname(null);  
 doctor.setWorkExperience(0);  
 doctor.setAppointments(null);  
  
 doctor.readFromFile(fileName);  
 System.*out*.println(doctor);  
  
 doctor.readFromFile("FirstSorted" + fileName);  
 System.*out*.println("Sorted by amount of visitors:\n" + doctor);  
  
 doctor.readFromFile("SecondSorted" + fileName);  
 System.*out*.println("Sorted by comment alphabetically:\n" + doctor);  
 }  
}

Клас DoctorWithTextFileOperations

package task1;  
  
import java.io.FileNotFoundException;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.PrintWriter;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
  
public class DoctorWithTextFileOperations extends DoctorWithFileOperations {  
 public DoctorWithTextFileOperations(DoctorWithArray doctor) {  
 this(doctor.getSurname(), doctor.getWorkExperience(), new ArrayList<>(Arrays.*stream*(doctor.getAppointments()).toList()).toArray(new Appointment[0]));  
 }  
 public DoctorWithTextFileOperations(String surname, int workExperience, Appointment[] appointments) {  
 super(surname, workExperience, appointments);  
 }  
  
 @Override  
 public void readFromFile(String fileName) {  
 try (Scanner scanner = new Scanner(new FileReader(fileName))) {  
 setSurname(scanner.nextLine());  
 setWorkExperience(scanner.nextInt());  
 List<Appointment> appointments = new ArrayList<>();  
 while (scanner.hasNext()) {  
 String day = scanner.next();  
 int visitors = scanner.nextInt();  
 String comment = scanner.next();  
 appointments.add(new Appointment(day, visitors, comment));  
 }  
 setAppointments(appointments.toArray(new Appointment[0]));  
 } catch (FileNotFoundException e) {  
 throw new RuntimeException(e);  
 }  
 }  
  
 @Override  
 public void writeToFile(String fileName) {  
 try (PrintWriter out = new PrintWriter(new FileWriter(fileName))) {  
 out.println(getSurname());  
 out.println(getWorkExperience());  
 for (Appointment appointment : getAppointments()) {  
 out.println(appointment.getDay());  
 out.println(appointment.getVisitors());  
 out.println(appointment.getComment());  
 }  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
}

Клас DoctorWithDataFileOperations

package task1;  
  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class DoctorWithDataFileOperations extends DoctorWithFileOperations {  
 public DoctorWithDataFileOperations(DoctorWithArray doctor) {  
 this(doctor.getSurname(), doctor.getWorkExperience(), new ArrayList<>(Arrays.*stream*(doctor.getAppointments()).toList()).toArray(new Appointment[0]));  
 }  
 public DoctorWithDataFileOperations(String surname, int workExperience, Appointment[] appointments) {  
 super(surname, workExperience, appointments);  
 }  
  
 @Override  
 public void readFromFile(String fileName) {  
 try (DataInputStream in = new DataInputStream(new FileInputStream(fileName))) {  
 setSurname(in.readUTF());  
 setWorkExperience(in.readInt());  
 List<Appointment> appointments = new ArrayList<>();  
 int count = in.readInt();  
 for (int i = 0; i < count; i++) {  
 String day = in.readUTF();  
 int visitors = in.readInt();  
 String comment = in.readUTF();  
 appointments.add(new Appointment(day, visitors, comment));  
 }  
 setAppointments(appointments.toArray(new Appointment[0]));  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
  
 @Override  
 public void writeToFile(String fileName) {  
 try (DataOutputStream out = new DataOutputStream(new FileOutputStream(fileName))) {  
 out.writeUTF(getSurname());  
 out.writeInt(getWorkExperience());  
 out.writeInt(getAppointments().length);  
 for (int i = 0; i < getAppointments().length; i++) {  
 out.writeUTF(getAppointments()[i].getDay());  
 out.writeInt(getAppointments()[i].getVisitors());  
 out.writeUTF(getAppointments()[i].getComment());  
 }  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

Клас Main

package task1;  
  
public class Main {  
 public static void main(String[] args) {  
 DoctorWithArray doctor = new DoctorWithArray("TestSurname", 5, new Appointment[] {  
 new Appointment("01.11", 3, "A"),  
 new Appointment("02.11", 1, "C"),  
 new Appointment("03.11", 2, "B")  
 });  
  
 DoctorWithTextFileOperations txtDoctor = new DoctorWithTextFileOperations(doctor);  
 DoctorWithDataFileOperations datDoctor = new DoctorWithDataFileOperations(doctor);  
  
 DoctorWithFileOperations.*test*(txtDoctor, "Doctor.txt");  
 DoctorWithFileOperations.*test*(datDoctor, "Doctor.dat");

}  
}

**Екранні форми за результатами роботи програмного коду відповідно до наборів тестових даних завдання №1:**

**1)**















**2)**













**Завдання №2. Сортування цілих**

Реалізувати програму читання з текстового файлу цілих додатних значень (числа розділені пробілами, слід читати до кінця файлу), занесення цих чисел у масив, сортування за зменшенням та за збільшенням суми цифр та зберігання обох результатів у двох нових текстових файлах. Перелічені дії реалізувати в окремій статичній функції. Для визначення порядку сортування створити класи, які реалізують інтерфейс Comparator.

**Набір тестових даних завдання №2:**

Послідовність чисел: 40, 23, 10, 54, 2, 73, 6, 43, що записана у файлі “in.txt”.

1) Сортування за збільшенням суми цифр.

Очікуваний результат: послідовність чисел: 10, 2, 40, 23, 6, 43, 54, 73, що записана у файлі “firstOut.txt”.

2) Сортування за зменшенням суми цифр.

Очікуваний результат: послідовність чисел: 73, 54, 43, 6, 23, 40, 2, 10, що записана у файлі “secondOut.txt”.

**Програмний код до завдання №2:**

Клас SortingIntsFromFile

package task2;  
  
import java.io.\*;  
import java.util.Arrays;  
import java.util.Comparator;  
import java.util.Scanner;  
  
public class SortingIntsFromFile {  
 public static void readAndWriteToFile(String inFileName, String firstOutFileName, String secondOutFileName) throws IOException {  
 Integer[] arr = new Integer[] { };  
  
 try (BufferedReader reader = new BufferedReader(new FileReader(inFileName));  
 Scanner scanner = new Scanner(reader)) {  
 while (scanner.hasNext()) {  
 int n = scanner.nextInt();  
 if (n > 0) {  
 Integer[] temp = new Integer[arr.length + 1];  
 System.*arraycopy*(arr, 0, temp, 0, arr.length);  
 temp[temp.length - 1] = n;  
 arr = temp;  
 }  
 }  
 }  
  
 PrintWriter writer1 = new PrintWriter(new FileWriter(firstOutFileName));  
 PrintWriter writer2 = new PrintWriter(new FileWriter(secondOutFileName));  
  
 try {  
 Arrays.*sort*(arr, new CompareAsc());  
 for (Integer n : arr) {  
 writer1.print(n + " ");  
 }  
  
 Arrays.*sort*(arr, new CompareDesc());  
 for (Integer n : arr) {  
 writer2.print(n + " ");  
 }  
 }  
 finally {  
 writer1.close();  
 writer2.close();  
 }  
 }  
  
 private static int sumOfDigits(int n) {  
 int sum = 0;  
 while (n > 9) {  
 sum += n % 10;  
 n /= 10;  
 }  
 sum += n;  
 return sum;  
 }  
  
 private static class CompareAsc implements Comparator<Integer> {  
 @Override  
 public int compare(Integer o1, Integer o2) {  
 return Integer.*compare*(SortingIntsFromFile.*sumOfDigits*(o1), SortingIntsFromFile.*sumOfDigits*(o2));  
 }  
 }  
  
 private static class CompareDesc implements Comparator<Integer> {  
 @Override  
 public int compare(Integer o1, Integer o2) {  
 return -1 \* Integer.*compare*(SortingIntsFromFile.*sumOfDigits*(o1), SortingIntsFromFile.*sumOfDigits*(o2));  
 }  
 }  
}

Клас Main

package task2;  
  
import java.io.IOException;  
  
public class Main {  
 public static void main(String[] args) {  
 try {  
 SortingIntsFromFile.*readAndWriteToFile*("in.txt", "firstOut.txt", "secondOut.txt");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**Екранні форми за результатами роботи програмного коду відповідно до наборів тестових даних завдання №2:**



1)



2)



**Завдання №3. Реалізація серіалізації й десеріалізації в XML**

Описати класи Студент і Академічна група (з полем – масивом студентів). Створити об'єкти, здійснити їх серіалізацію й десеріалізацію в XML.

**Набір тестових даних завдання №3:**

Створення об’єкту academGroup класу AcademGroup {name: “SomeGroup”, students: Student{name: “Alex”}, Student{name: “Michael”}, Student{name: ”Peter”}}.

Вивід даних про об’єкт academGroup. Очікується:

*Object serialized to XML-file:*

*Group: SomeGroup*

*Students:*

*Alex*

*Michael*

*Peter*

Серіалізація об’єкту academGroup до файлу “AcademGroup.xml”. Очікується створення XML-файлу з даними про об’єкт academGroup.

Створення порожнього об’єкту newAcademGroup класу AcademGroup.

Десеріалізація даних з XML-файлу “AcademGroup.xml” до об’єкту newAcademGroup.

Вивід даних про об’єкт newAcademGroup. Очікується:

*Object deserialized from XML-file:*

*Group: SomeGroup*

*Students:*

*Alex*

*Michael*

*Peter*

**Програмний код до завдання №3:**

Клас Student

package task3;  
  
public class Student {  
 private String name;  
  
 public Student() { }  
 public Student(String name) {  
 this.name = name;  
 }  
  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public String toString() {  
 return name;  
 }  
}

Клас AcademGroup

package task3;  
  
import java.beans.XMLDecoder;  
import java.beans.XMLEncoder;  
import java.io.FileInputStream;  
import java.io.FileOutputStream;  
import java.io.IOException;  
  
public class AcademGroup {  
 private String name;  
 private Student[] students;  
  
 public AcademGroup() {}  
 public AcademGroup(String name, Student[] students) {  
 this.name = name;  
 this.students = students;  
 }  
  
 public Student[] getStudents() {  
 return students;  
 }  
 public void setStudents(Student[] students) {  
 this.students = students;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public String toString() {  
 String result = "Group: " + name + "\nStudents:\n";  
 for (Student student : students) {  
 result += student + "\n";  
 }  
 return result;  
 }  
  
 public void writeToXMLFile(String fileName) {  
 try (XMLEncoder xmlEncoder = new XMLEncoder(new FileOutputStream(fileName))) {  
 xmlEncoder.writeObject(this);  
 xmlEncoder.flush();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 public void readFromXMLFile(String fileName) {  
 try (XMLDecoder xmlDecoder = new XMLDecoder(new FileInputStream(fileName))) {  
 AcademGroup academGroup = (AcademGroup) xmlDecoder.readObject();  
 this.name = academGroup.name;  
 this.students = academGroup.students;  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

Клас Main

package task3;  
  
public class Main {  
 public static void main(String[] args) {  
 // створення об'єкту  
 AcademGroup academGroup = new AcademGroup("SomeGroup", new Student[] {  
 new Student("Alex"),  
 new Student("Michael"),  
 new Student("Peter")  
 });  
  
 System.*out*.println("Object serialized to XML-file:\n" + academGroup);  
  
 String fileName = "AcademGroup.xml";  
  
 // серіалізація об'єкту в XML-файл  
 academGroup.writeToXMLFile(fileName);  
  
 // створення порожнього об'єкту  
 AcademGroup newAcademGroup = new AcademGroup();  
  
 // десеріалізація об'єкту з XML-файлу  
 newAcademGroup.readFromXMLFile(fileName);  
  
 System.*out*.println("Object deserialized from XML-file:\n" + newAcademGroup);  
 }  
}

**Екранні форми за результатами роботи програмного коду відповідно до наборів тестових даних завдання №3:**



**Висновки:**

У результаті виконання лабораторної роботи №4 з дисципліни «Основи програмування Java» я ознайомився з механізмами роботи з винятками та файлами в Java.