

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

КАФЕДРА «Програмна інженерія та інтелектуальні технології
управління»

ЗВІТ
з лабораторної роботи № 3
з навчальної дисципліни
“ПОГЛИБЛЕНИЙ КУРС ПРОГРАМУВАННЯ JAVA”

ВИКОНАЛА

Студентка групи КН-222а

Репешко Інеса Віталіївна

ПЕРЕВІРИВ

асистент кафедри ПІТУ

Кондратов Олексій Михайлович

Харків 2024

Тема роботи

“Розширені можливості роботи з файлами”

1. Завдання №1 до лабораторної роботи

1.1. Індивідуальне завдання

Створити новий Maven-проект, в який перенести раніше створені класи, які представляють сутності індивідуальних завдань лабораторних робіт № 3 і № 4 курсу "Основи програмування Java".

Програма повинна демонструвати:

- відтворення реалізації завдань лабораторних робіт № 3 і № 4 курсу "Основи програмування Java";
- використання засобів Stream API для обробки та виведення послідовностей;
- виведення даних в текстовий файл засобами Stream API з подальшим читанням;
- серіалізацію об'єктів у XML-файл і JSON-файл і відповідну десеріалізацію із застосуванням бібліотеки XStream;
- запис подій, пов'язаних з виконанням програми, в системний журнал;
- тестування окремих класів з використанням JUnit.

Для представлення індивідуальних даних слід використати класи, які були створені під час виконання індивідуального завдання лабораторної роботи № 1 цього курсу.

Примітка: локалізацію і переклад текстів можна проводити за бажанням студента.

Умови завдання для лабораторних робіт № 3 та № 4 для варіанту 24 (номер 24 за порядком у списку групи) наведено нижче.

№ №	Перша сутність		Друга сутність		Основне завдання: знайти та вивести такі дані
	Сутність	Обов'язкові поля	Сутність	Обов'язкові поля	
8, 24	Станція метрополітену	Назва, рік відкриття	Година	Кількість пасажирів, коментар	Сумарна кількість пасажирів, години з найменшою кількістю пасажирів та найбільшою кількістю слів у коментарі

Рисунок 1.1.1 – Умова 1 завдання № 1 варіант 24

№№	Перша ознака	Друга ознака
8, 24	За зменшенням кількості пасажирів	За зменшенням довжини коментаря

Рисунок 1.1.2 – Умова 2 завдання № 1 варіант 24

1.2. Програмний код реалізації завдання № 1

1.2.1. Hour.java:

```
package part1.lab4.task1;

import java.util.Arrays;

/**
 * The {@code Hour} class performs hour with {@code ridership} and
 * {@code comment}.
 */
public class Hour implements Comparable<Hour> {
    /** Ridership is the number of passengers visiting a metro station
    per hour. */
    private int ridership;

    /** Comment on the {@code ridership} metric. */
    private String comment;

    /**
     * The constructor initialises the hour object with the default
     values.
     */
    public Hour() {
```

```

}

/**
 * The constructor initialises the hour object with the specified
 values.
 * @param ridership the ridership;
 * @param comment the comment.
 */
public Hour(int ridership, String comment) {
    if (ridership < 0) {
        this.ridership = 0;
    }

    if (comment == null) {
        this.comment = "";
    }

    this.ridership = ridership;
    this.comment = comment;
}

/**
 * Gets the {@code ridership} of the hour.
 * @return the {@code ridership}.
 */
public int getRidership() {
    if (ridership < 0) {
        return 0;
    }
    return ridership;
}

/**
 * Sets the {@code ridership} of the hour.
 * @param ridership the {@code ridership} to be set.
 */
public void setRidership(int ridership) {
    if (ridership < 0) {
        this.ridership = 0;
    }

    this.ridership = ridership;
}

/**
 * Gets the {@code comment} for the hour.
 * @return the {@code comment}.
 */
public String getComment() {

```

```

        if (comment == null) {
            return "";
        }

        return comment;
    }

    /**
     * Sets the {@code comment} for the hour.
     * @param comment the {@code comment} to be set.
     */
    public void setComment(String comment) {
        if (comment == null) {
            this.comment = "";
        }

        this.comment = comment;
    }

    /**
     * Gets the length of a comment in the hour.
     * @return the length of a comment.
     */
    public int getCommentLength() {
        if (comment == null) {
            return 0;
        }

        return getComment().length();
    }

    /**
     * Calculates the count of words of a comment in the hour.
     * @return the length of a comment.
     */
    public int calculateWordCountOfComment() {
        if (comment == null
            || comment.isEmpty()) {
            return 0;
        }

        String[] wordArray = comment.split(" ");

        return wordArray.length;
    }

    /**
     * Provides the string representing the Hour object.
     * @return the string representing the Hour object.

```

```

    */
    @Override
    public String toString() {
        return "Hour\t{ "
            + "ridership = " + getRidership()
            + ",\tcomment = \'' + getComment() + "\' }";
    }

    /**
     * Checks metro station this hour is equivalent to another.
     * @param obj the hour with which check the equivalence;
     * @return {@code true}, if two hours are the same and {@code false}
     otherwise.
    */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }

        if (!(obj instanceof Hour hour)) {
            return false;
        }

        return Integer.compare(hour.getRidership(), getRidership()) == 0
            && hour.getComment().equals(getComment());
    }

    /**
     * Calculates the hash code of the hour.
     * If two objects are equal, they must have the same hash code.
     * If this method is called multiple times on the same object, it
     must return the same number each time.
     * @return the hash code of the hour.
    */
    @Override
    public int hashCode() {
        return Integer.hashCode(getRidership()) *
getComment().hashCode();
    }

    /**
     * Compares this Hour object with another Hour object based on
     ridership.
     * @param h the object to be compared;
     * @return negative number, if this object is smaller, zero, if they
     are equal,
     * positive number, if this object is larger.
    */

```

```

@Override
public int compareTo(Hour h) {
    return Integer.compare(h.getRidership(), getRidership());
}

/**
 * Prints the array of hours.
 * @param hours array of hours to print.
 */
public void printHourArray(Hour[] hours) {
    System.out.println("Array of hours:");

    for (Hour hour : hours) {
        System.out.println(hour);
    }
}

/**
 * Tests of the functionality of the {@code Hour} class.
 */
public void testHour() {
    System.out.println("Create Hour with default constructor:");
    Hour hour = new Hour();
    System.out.println(hour);
    System.out.println("Length of comment:\t" +
hour.getCommentLength());
    System.out.println("Count of words in comment:\t" +
hour.calculateWordCountOfComment());

    System.out.println("\nCreate Hour with parameterized
constructor:");
    System.out.println("Valid data for hour:");
    hour = new Hour(100, "Low ridership");
    System.out.println(hour);
    System.out.println("Invalid data for hour:");
    Hour invalidHour = new Hour(-200, null);
    System.out.println(invalidHour);

    System.out.println("\nSet values for the Hour:");
    hour.setRidership(200);
    hour.setComment("Medium ridership");
    System.out.println(hour);

    System.out.println("\nGet values for the Hour:");
    System.out.println("Hour\t{ "
        + "ridership = " + hour.getRidership()
        + ",\tcomment = \"" + hour.getComment() + "\" }");
    System.out.println("Get length of comment:\t" +
hour.getCommentLength());

```

```

        System.out.println("Get count of words in comment:\t" +
hour.calculateWordCountOfComment() + "\n");

        Hour[] hours = { hour,
            new Hour(50, "Very low ridership"),
            new Hour(200, "Medium ridership"),
            new Hour(100, "Low ridership"),
            new Hour(700, "High ridership"),
            new Hour(1200, "Very high ridership"),
            invalidHour
        };
        printHourArray(hours);

        System.out.println("\nCheck for equal values of Hours at index 0
and 1:\t" + hours[0].equals(hours[1]));
        System.out.println("Hour at index 0:\t" + hours[0]);
        System.out.println("Hour at index 1:\t" + hours[1]);
        System.out.println("Check for equal values of Hours at index 0
and 2:\t" + hours[0].equals(hours[2]));
        System.out.println("Hour at index 0:\t" + hours[0]);
        System.out.println("Hour at index 2:\t" + hours[2]);

        System.out.println("\nComparison of Hours at index 0 and 1:\t" +
hours[0].compareTo(hours[1]));
        System.out.println("HashCode of Hour at index 0:\t" +
hours[0].hashCode());
        System.out.println("HashCode of Hour at index 1:\t" +
hours[1].hashCode());
        System.out.println("Comparison of Hours at index 0 and 2:\t" +
hours[0].compareTo(hours[2]));
        System.out.println("HashCode of Hour at index 0:\t" +
hours[0].hashCode());
        System.out.println("HashCode of Hour at index 2:\t" +
hours[2].hashCode());
        System.out.println("Comparison of Hours at index 1 and 2:\t" +
hours[1].compareTo(hours[2]));
        System.out.println("HashCode of Hour at index 1:\t" +
hours[1].hashCode());
        System.out.println("HashCode of Hour at index 2:\t" +
hours[2].hashCode());

        System.out.println("\nSort array of Hours by descending
ridership:");
        Arrays.sort(hours);
        printHourArray(hours);
    }
}

```


1.2.2. AbstractMetroStation.java:

```

package part1.lab4.task1;

import java.util.Arrays;

/**
 * Abstract class representing metro station with {@code name}, {@code
 * opened} year and operating hour data.
 * Access to the sequence of hours, {@code name} and {@code opened} year
 * is represented by abstract methods.
 */
public abstract class AbstractMetroStation {
    /**
     * Gets the {@code name} for the metro station.
     * The derived class must provide an implementation of this method.
     * @return the {@code name}.
     */
    public abstract String getName();

    /**
     * Sets the {@code name} for the metro station.
     * The derived class must provide an implementation of this method.
     * @param name the {@code name} to be set.
     */
    public abstract void setName(String name);

    /**
     * Gets the {@code opened} year for the metro station.
     * The derived class must provide an implementation of this method.
     * @return the {@code opened}.
     */
    public abstract int getOpened();

    /**
     * Sets the {@code opened} year for the metro station.
     * The derived class must provide an implementation of this method.
     * @param opened the {@code opened} year to be set.
     */
    public abstract void setOpened(int opened);

    /**
     * Gets the {@code hour} with index {@code i}.
     * The derived class must provide an implementation of this method.
     * @param i the index of hour array element;
     * @return the object of class {@code Hour} with index {@code i}.
     */
    public abstract Hour getHour(int i);
}

```

```

/**
 * Sets the {@code hour} with index {@code i}.
 * The derived class must provide an implementation of this method.
 * @param i index of {@code hour} in array of hours;
 * @param hour the object of class {@code Hour} with index {@code i}
to be set.
 */
public abstract void setHour(int i, Hour hour);

/**
 * Gets the array of operating hours for the metro station.
 * The derived class must provide an implementation of this method.
 * @return the array of operating hours.
 */
public abstract Hour[] getHours();

/**
 * Sets the array of operating hours for the metro station.
 * The derived class must provide an implementation of this method.
 * @param hours the array of operating hours to be set.
 */
public abstract void setHours(Hour[] hours);

/**
 * Adds a link to the new operating {@code hour} at the end of the
hour array.
 * The derived class must provide an implementation of this method.
 * @param hour the object of class {@code Hour} to be added;
 * @return {@code true}, if the link was added successfully, {@code
false} otherwise.
 */
public abstract boolean addHour(Hour hour);

/**
 * Creates a new operating {@code hour} and adds a link to it at the
end of the hour array.
 * The derived class must provide an implementation of this method.
 * @param ridership the ridership;
 * @param comment the comment;
 * @return {@code true}, if the link was added successfully, {@code
false} otherwise.
 */
public abstract boolean addHour(int ridership, String comment);

/**
 * Counts the number of hours in the hours array.
 * The derived class must provide an implementation of this method.
 * @return the number of hours.
 */

```

```

public abstract int countHours();

/**
 * Removes the sequence of hours from hours array.
 * The derived class must provide an implementation of this method.
 */
public abstract void removeHours();

/**
 * Provides the string representing the object that is inherited
 from this abstract class.
 * @return the string representing the object that is inherited from
 this abstract class.
 */
@Override
public String toString() {
    StringBuilder string = new StringBuilder();
    string.append("Station:\t")
        .append("Name: \'").append(getName()).append("\'.\t")
        .append("Opened: ").append(getOpened()).append(".\t")
        .append("Hours:\n");

    if (countHours() ≤ 0) {
        string.append("There are no hours for this station.\n");
    } else {
        for (Hour h : getHours()) {
            string.append(h).append("\n");
        }
    }

    return string.toString();
}

/**
 * Checks whether this metro station is equivalent to another.
 * @param obj the metro station with which check the equivalence.
 * @return {@code true}, if two weathers are the same, {@code false}
 otherwise.
 */
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }

    if (!(obj instanceof AbstractMetroStation ms)) {
        return false;
    }
}

```

```

        if (!ms.getName().equals(getName())
            || Integer.compare(ms.getOpened(), getOpened()) != 0) {
            return false;
        }

        return Arrays.equals(getHours(), ms.getHours());
    }

    /**
     * Calculates the hash code of the metro station.
     * If two objects are equal, they must have the same hash code.
     * If this method is called multiple times on the same object, it
     must return the same number each time.
     * @return the hash code of the metro station.
     */
    @Override
    public int hashCode() {
        return getName().hashCode() * Integer.hashCode(getOpened()) *
Arrays.hashCode(getHours());
    }

    /**
     * An additional static function for adding hour reference to the
     provided array of hours.
     * @param hours the array to which the hour is added;
     * @param hour the link that is added;
     * @return updated array of hours.
     */
    public static Hour[] addHourToArray(Hour[] hours, Hour hour) {
        if (hour.getRidership() < 0
            || hour.getComment() == null) {
            return hours;
        }

        Hour[] newHours;

        if (hours == null) {
            newHours = new Hour[1];
        } else {
            newHours = new Hour[hours.length + 1];
            System.arraycopy(hours, 0, newHours, 0, hours.length);
        }

        newHours[newHours.length - 1] = hour;

        return newHours;
    }

    /**

```

```

    * Calculates the total ridership for an array of metro station
    operating hours.
    * @return null, if there is no pointer to the hours array, or it is
    empty, the total ridership otherwise.
    */
    public Integer calculateTotalRidership() {
        if (countHours() == 0) {
            return null;
        }

        int totalRidership = 0;

        for (Hour hour : getHours()) {
            totalRidership += hour.getRidership();
        }

        return totalRidership;
    }

    /**
     * Finds the hours with the minimal ridership in the array of metro
     station operating hours.
     * @return null, if there is no pointer to the hours array, or it is
     empty,
     * array of hours with minimal ridership otherwise.
     */
    public Hour[] findHoursWithMinRidership() {
        if (countHours() == 0) {
            return null;
        }

        Hour minHour = getHours()[0];

        for (Hour hour : getHours()) {
            if (hour.getRidership() < minHour.getRidership()) {
                minHour = hour;
            }
        }

        Hour[] hours = null;

        for (Hour hour : getHours()) {
            if (hour.getRidership() == minHour.getRidership()) {
                hours = addHourToArray(hours, hour);
            }
        }

        return hours;
    }

```

```

    /**
     * Finds the hours with the maximum count of words in the comment in
     * the array of metro station operating hours.
     * @return null, if there is no pointer to the hours array, or it is
     * empty,
     * array of hours with the maximum word count in comment otherwise
     */
    public Hour[] findHoursWithMaxWordCountOfComment() {
        if (countHours() == 0) {
            return null;
        }

        Hour maxHour = getHours()[0];

        for (Hour hour : getHours()) {
            if (hour.calculateWordCountOfComment() >
maxHour.calculateWordCountOfComment()) {
                maxHour = hour;
            }
        }

        Hour[] hours = null;

        for (Hour hour : getHours()) {
            if (hour.calculateWordCountOfComment() ==
maxHour.calculateWordCountOfComment()) {
                hours = addHourToArray(hours, hour);
            }
        }

        return hours;
    }

    /**
     * Finds the total ridership for an array of metro station operating
     * hours and prints the result to the console.
     */
    public void printTotalRidership() {
        Integer totalRidership = calculateTotalRidership();
        System.out.print("Total ridership for station:\t");

        if (totalRidership == null) {
            System.out.println("There is no ridership hours.");
        } else {
            System.out.println(totalRidership);
        }
    }
}

```

```

/**
 * Prints the array of hours.
 * @param hours the array of hours to be printed.
 */
public void printHours(Hour[] hours) {
    for (Hour hour : hours) {
        System.out.println(hour);
    }
}

/**
 * Finds the hours with the minimal ridership in the array of metro
station operating hours
 * and prints the result to the console.
 */
public void printHoursWithMinRidership() {
    Hour[] hours = findHoursWithMinRidership();
    System.out.print("Hours with minimal ridership:\t");

    if (hours == null) {
        System.out.println("There is no ridership hours.");
    } else {
        System.out.println();
        printHours(hours);
    }
}

/**
 * Finds the hours with the maximum count of words in the comment in
the array of metro station operating hours
 * and prints the result to the console.
 */
public void printHoursWithMaxWordCountOfComment() {
    Hour[] hours = findHoursWithMaxWordCountOfComment();
    System.out.print("Hours with the maximum word count in a
comment:\t");

    if (hours == null) {
        System.out.println("There is no ridership hours.");
    } else {
        System.out.println();
        printHours(hours);
    }
}

/**
 * Sorts a sequence of hours by decreasing ridership using bubble
sorting.
 */

```

```

public void sortByDecreasingRidership() {
    if (countHours() == 0) {
        return;
    }

    boolean unsorted = true;

    while (unsorted) {
        unsorted = false;

        for (int i = 0; i < getHours().length - 1; i++) {
            if (getHours()[i].getRidership() < getHours()[i +
1].getRidership()) {
                Hour temp = getHours()[i];
                getHours()[i] = getHours()[i + 1];
                getHours()[i + 1] = temp;
                unsorted = true;
            }
        }
    }
}

/**
 * Sorts a sequence of hours by descending comment length using
insertion sorting.
 */
public void sortByDescendingCommentLength() {
    if (countHours() == 0) {
        return;
    }

    for (int i = 0; i < getHours().length; i++) {
        Hour key = getHours()[i];
        int j;

        for (j = i - 1; j ≥ 0
            && Integer.compare(getHours()[j].getCommentLength(),
key.getCommentLength()) < 0; j--) {
            getHours()[j + 1] = getHours()[j];
        }

        getHours()[j + 1] = key;
    }
}

/**
 * An additional function for adding hours to a sequence of hours in
hours array.
 * @return The object is inherited from this abstract class.

```



```

    */
    public AbstractMetroStation createMetroStationHours() {
        System.out.println("Add 6 valid Operating Hours at Metro
Station:");
        System.out.print(addHour(320, "Medium ridership") + "\t");
        Hour hour = new Hour(88, "Very low ridership");
        System.out.println(addHour(hour) + "\t"
            + addHour(107, "Low ridership") + "\t"
            + addHour(688, "High ridership") + "\t"
            + addHour(1234, "Very high ridership"));

        System.out.println("Add one Operating Hour with invalid data at
Metro Station:\t"
            + addHour(-1, null));

        System.out.println("Add one Operating Hour with duplicate data
at Metro Station:\t"
            + addHour(1234, "Very high ridership"));

        return this;
    }

    /**
     * Calls up search methods and print results of searching.
     */
    public void showSearchResults() {
        printTotalRidership();
        printHoursWithMinRidership();
        printHoursWithMaxWordCountOfComment();
    }

    /**
     * Performs testing of search methods.
     */
    public void testSearchData() {
        System.out.println("SEARCHING RESULTS:");
        setName("Universytet");
        setOpened(1984);
        System.out.println("Search data for Metro Station without
Operating Hours:");
        removeHours();
        showSearchResults();
        System.out.println();

        System.out.println("Create the Metro Station:");
        createMetroStationHours();
        System.out.println(this);
        showSearchResults();
        System.out.println();
    }

```

```

        System.out.println("Add new two Operating Hours with min
ridership and max word count in comment for searching:");
        System.out.println(addHour(75, "Very low ridership"));
        System.out.println(addHour(2000, "Maximum possible ridership for
station"));
        System.out.println(this);
        showSearchResults();
    }

    /**
     * Performs testing of sorting methods.
     */
    public void testSortingData() {
        System.out.println();
        System.out.println("SORTING RESULTS:");
        setName("Derzhprom");
        setOpened(1995);
        System.out.println("Sort data for Metro Station without
Operating Hours:");
        removeHours();
        sortByDecreasingRidership();
        sortByDescendingCommentLength();
        System.out.println(this);

        System.out.println("Create the Metro Station:");
        createMetroStationHours();
        System.out.println(this);

        System.out.println("Sort Operating Hours by decreasing
ridership:");
        sortByDecreasingRidership();
        System.out.println(this);

        System.out.println("Sort Operating Hours by descending comment
length:");
        sortByDescendingCommentLength();
        System.out.println(this);
    }
}

```

1.2.3. MetroStationWithCollection.java:

```

package part1.lab4.task1;

/**
 * An abstract class {@link MetroStationWithCollection} representing a
Metro Station with a collection of operating

```

```

* hours. Extends the {@link AbstractMetroStation} class.
*/
public abstract class MetroStationWithCollection extends
AbstractMetroStation {
    /** The name of the metro station. */
    private String name;

    /** The opened year of the metro station. */
    private int opened;

    /**
     * The constructor initialises the object with the default values.
     */
    public MetroStationWithCollection() {}

    /**
     * The constructor initialises the object with the specified values
with metro station {@code name}
     * and {@code opened} year.
     * @param name the name of metro station;
     * @param opened the opened year of metro station.
     */
    public MetroStationWithCollection(String name, int opened) {
        this.name = name;
        this.opened = opened;
    }

    public abstract void setHour(int i, Hour hour);

    /**
     * Gets the {@code name} for the metro station.
     * @return the {@code name} of metro station.
     */
    @Override
    public String getName() {
        return name;
    }

    /**
     * Sets the {@code name} for the metro station.
     * @param name the {@code name} of metro station to be set.
     */
    @Override
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Gets the {@code opened} year for the metro station.

```

```

    * @return the {@code opened} year of metro station.
    */
    @Override
    public int getOpened() {
        return opened;
    }

    /**
     * Sets the {@code opened} year for the metro station.
     * @param opened the {@code opened} year of metro station to be set.
     */
    @Override
    public void setOpened(int opened) {
        this.opened = opened;
    }

    /**
     * Performs testing of the functionality of the {@code
MetroStationWithCollection} class.
     */
    public void testMetroStationWithCollection() {
        System.out.println("Initial Metro Station data:");
        System.out.println(this);

        Hour[] hoursArray = {
            new Hour(23, "Very low ridership"),
            new Hour(345, "Medium ridership"),
            new Hour(87, "Low ridership"),
            new Hour(1007, "Very high ridership")
        };

        System.out.println("Get Metro Station Name and Opened Year:");
        System.out.println("Name:\t" + getName() + "\tOpened:\t" +
getOpened());
        System.out.println();

        System.out.println("Reset the Operating Hours for the Metro
Station:");
        setHours(hoursArray);
        System.out.println(this);

        System.out.println("Set the Operating Hour by index and get all
Operating Hours:");
        setHour(0, new Hour(250, "Medium ridership"));
        hoursArray = getHours();
        for (Hour hour : hoursArray) {
            System.out.println(hour);
        }
        System.out.println();
    }

```

```

        System.out.println("Get Operating Hour by index:");
        System.out.println(getHour(1));
        System.out.println("Get count of all Operating Hours:\t" +
countHours());
        System.out.println();
    }
}

```

1.2.4. MetroStationWithList.java:

```

package part1.lab4.task1;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.Collections;
import java.util.Set;
import java.util.LinkedHashSet;

/**
 * Represents metro station data with an ArrayList of operating hours.
 * This class is inherited from the abstract {@link
MetroStationWithCollection}.
 */
public class MetroStationWithList extends MetroStationWithCollection {
    /** List of operating hours of the metro station. */
    private List<Hour> hours = new ArrayList<>();

    /**
     * The constructor initialises the metro station object with the
default values.
     */
    public MetroStationWithList() {}

    /**
     * The constructor initialises the metro station object with the
specified values with {@code name},
     * {@code opened} year and operating {@code hours}.
     * @param name the name of metro station;
     * @param opened the opened year of metro station;
     * @param hours the operating hours of metro station.
     */
    public MetroStationWithList(String name, int opened, ArrayList<Hour>
hours) {
        super(name, opened);
    }
}

```

```

        Set<Hour> uniqueSet = new LinkedHashSet<>(hours);
        this.hours = new ArrayList<>(uniqueSet);
    }

    /**
     * The constructor initialises the metro station object with the
     * specified values with {@code name} and {@code opened} year.
     * @param name the name of metro station;
     * @param opened the opened year of metro station.
     */
    public MetroStationWithList(String name, int opened) {
        super(name, opened);
    }

    /**
     * Gets the array of operating hours for the metro station.
     * @return the array of hours.
     */
    @Override
    public Hour[] getHours() {
        return hours.toArray(new Hour[0]);
    }

    /**
     * Gets the list of operating hours for the metro station.
     * @return the list of operating hours for the metro station.
     */
    public List<Hour> getHoursList() {
        return hours;
    }

    /**
     * Sets the list of operating hours for the metro station.
     * @param hours the array of hours to be set.
     */
    @Override
    public void setHours(Hour[] hours) {
        Set<Hour> uniqueSet = new LinkedHashSet<>(Arrays.asList(hours));
        this.hours = new ArrayList<>(uniqueSet);
    }

    /**
     * Sets the list of Operating Hours for the Metro Station.
     * @param hours the list of Hours
     */
    protected void setHoursList(List<Hour> hours) {
        this.hours = hours;
    }

```

```

/**
 * Gets the {@code hour} with index {@code i} from the hours list.
 * @return the object of class {@code Hour} with index {@code i}.
 */
@Override
public Hour getHour(int i) {
    return hours.get(i);
}

/**
 * Sets the {@code hour} with index {@code i} to hours list.
 * @param i index of {@code hour} in hours list;
 * @param hour the object of class {@code Hour} with index {@code i}
to be set.
 */
@Override
public void setHour(int i, Hour hour) {
    if (hours.contains(hour)) {
        return;
    }

    hours.set(i, hour);
}

/**
 * Adds a link to the new {@code hour} at the end of the hours list.
 * @param hour the object of class {@code Hour} to be added to the
hours list;
 * @return {@code true}, if the link was added successfully, {@code
false} otherwise.
 */
@Override
public boolean addHour(Hour hour) {
    if (hours.contains(hour)) {
        return false;
    }

    return hours.add(hour);
}

/**
 * Creates a new {@code hour} and adds a link to it at the end of
the sequence at the hours list.
 * @param ridership the ridership;
 * @param comment the comment;
 * @return {@code true}, if the link was added successfully, {@code
false} otherwise.
 */
@Override

```

```

public boolean addHour(int ridership, String comment) {
    return addHour(new Hour(ridership, comment));
}

/**
 * Counts the number of hours in the sequence at hours list.
 * @return the number of hours.
 */
@Override
public int countHours() {
    return hours.size();
}

/**
 * Removes the sequence of hours from hours list.
 */
@Override
public void removeHours() {
    hours.clear();
}

/**
 * Overridden decreasing ridership sorting method using the standard
sort function of class {@code Collections}.
 * Is provided by the implementation of the Comparable interface for
the {@code Hour} class.
 */
@Override
public void sortByDecreasingRidership() {
    Collections.sort(hours);
}

/**
 * Overridden descending comment length sorting method using the
default sort function of interface {@code List}.
 * Is provided by {@code Comparator}.
 */
@Override
public void sortByDescendingCommentLength() {
    hours.sort(Comparator.comparing(Hour::getCommentLength).reversed());
}
}

```

1.2.5. MetroStationWithStreams.java:

```
package part2.lab1.task1;
```



```

import part1.lab4.task1.Hour;
import part1.lab4.task1.MetroStationWithList;

import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

/**
 * Represents Metro Station data with a list of Hours.
 * Stream API tools are used to process sequences of elements.
 * This class is inherited from the {@link MetroStationWithList}.
 */
public class MetroStationWithStreams extends MetroStationWithList {
    /** The constructor initialises the Metro Station object with the
    default values. */
    public MetroStationWithStreams() {

        /**
         * The constructor initialises the Metro Station object with the
         specified values with {@code name},
         * {@code opened} year and operating {@code hours}.
         * @param name the name of Metro Station;
         * @param opened the opened year of Metro Station;
         * @param hours the Operating Hours of Metro Station.
         */
        public MetroStationWithStreams(String name, int opened, List<Hour>
hours) {
            super(name, opened);
            setHoursList(hours);
        }

        /**
         * The constructor initialises the Metro Station object with the
         specified values with {@code name} and {@code opened} year.
         * @param name the name of metro station;
         * @param opened the opened year of metro station.
         */
        public MetroStationWithStreams(String name, int opened) {
            super(name, opened);
        }

        /**
         * Sets the list of Hours for the Metro Station.
         * @param hours the list of Operating Hours.
         */
        @Override

```

```

public void setHoursList(List<Hour> hours) {
    super.setHoursList(hours.stream().collect(Collectors.toList()));
}

/**
 * Sets the list of Operating Hours for the Metro Station.
 * @param hours the array of Operating Hours.
 */
@Override
public void setHours(Hour[] hours) {
    setHoursList(Arrays.asList(hours));
}

/** Overridden decreasing ridership sorting method using Stream API
with the help of the {@link Comparator} interface. */
@Override
public void sortByDecreasingRidership() {
    setHoursList(getHoursList().stream()

.sorted(Comparator.comparing(Hour::getRidership).reversed())
    .collect(Collectors.toList()));
}

/** Overridden descending comment length sorting method using Stream
API with the help of the {@link Comparator} interface. */
@Override
public void sortByDescendingCommentLength() {
    setHoursList(getHoursList().stream()

.sorted(Comparator.comparing(Hour::getCommentLength).reversed())
    .collect(Collectors.toList()));
}

/**
 * Calculates the total ridership for an array of Metro Station
Operating Hours.
 * @return null if the list of Hours is empty or if there will be
problems with the calculation;
 * the total ridership otherwise.
 */
@Override
public Integer calculateTotalRidership() {
    if (getHoursList().isEmpty()) {
        return null;
    }

    return
getHoursList().stream().mapToInt(Hour::getRidership).sum();
}

```

```

    /**
     * Finds the hours with the minimal ridership in the list of Metro
     Station Operating Hours.
     * @return an array of {@code Hour} objects with the minimal
     ridership;
     * if the list of Hours is empty, returns null.
     */
    @Override
    public Hour[] findHoursWithMinRidership() {
        Hour hourWithMinRidership = getHoursList().stream()
            .min(Comparator.comparing(Hour::getRidership))
            .orElse(null);

        if (hourWithMinRidership == null) {
            return null;
        }

        return getHoursList().stream()
            .filter(hour → hour.getRidership() =
hourWithMinRidership.getRidership())
            .toArray(Hour[]::new);
    }

    /**
     * Finds the hours with the maximum count of words in the comment in
     the list of Metro Station Operating Hours.
     * @return An array of {@code Hour} objects with the maximum count
     of words in the comment;
     * if the list of Hours is empty, returns null.
     */
    @Override
    public Hour[] findHoursWithMaxWordCountOfComment() {
        Hour hourWithMaxWordCountOfComment = getHoursList().stream()
            .max(Comparator.comparing(Hour::calculateWordCountOfComment))
            .orElse(null);

        if (hourWithMaxWordCountOfComment == null) {
            return null;
        }

        return getHoursList().stream()
            .filter(hour → hour.calculateWordCountOfComment() =
hourWithMaxWordCountOfComment.calculateWordCountOfComment())
            .toArray(Hour[]::new);
    }

    /**

```

```

    * Demonstrates the functionality of the {@code
TramStationWithStreams} class.
    * Prints the created Metro Station, performs a search test and a
sorting test.
    */
    public void testMetroStationWithStreams() {
        System.out.println("The Metro Station created:\n" + this);
        this.testSearchData();
        this.testSortingData();
    }

    /**
     * Creates a new instance of {@code TramStationWithStreams} with
predefined values.
     * @return the object of class {@code TramStationWithStreams}.
     */
    public static MetroStationWithStreams
createMetroStationWithStreams() {
        return new MetroStationWithStreams("Politekhnichna", 1984,
            Arrays.asList(
                new Hour(1100, "Very high ridership"),
                new Hour(110, "Low ridership"),
                new Hour(650, "High ridership"),
                new Hour(532, "High ridership"),
                new Hour(60, "Very low ridership"),
                new Hour(188, "Low ridership"),
                new Hour(200, "Medium ridership"),
                new Hour(200, "Medium ridership")
            ));
    }
}

```

1.2.6. FileUtils.java:

```

package part2.lab3.task1;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.json.JettisonMappedXmlDriver;
import com.thoughtworks.xstream.security.AnyTypePermission;
import org.apache.logging.log4j.Logger;
import part1.lab4.task1.Hour;

import java.io.*;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.stream.Collectors;

```

```

/**
 * Utility class for handling file operations related to MetroStation
 * objects.
 * It implements the following approaches to working with files:
 * <p> - outputting data to a text file using Stream API with subsequent
 * reading;
 * <p> - serialization of objects into an XML file and a JSON file and
 * corresponding deserialization using the XStream library;
 * <p> - recording events related to program execution in the system
 * log;
 */
public class FileUtils {
    private static Logger logger = null;

    /**
     * Sets the logger to be used by this utility class.
     * @param logger The logger instance.
     */
    public static void setLogger(Logger logger) {
        FileUtils.logger = logger;
    }

    /**
     * Writes MetroStationWithFiles object to a TXT file using Stream
     * API.
     * @param outputFilePath The path of the output TXT file.
     * @param metroStation The MetroStationWithFiles object to write.
     */
    public static void writeToTXT(String outputFilePath,
        MetroStationWithFiles metroStation) {
        if (logger != null) {
            logger.info("Write to TXT file");
        }

        try {
            Files.write(Path.of(outputFilePath),
                metroStation.toListOfStrings());
        } catch (IOException e) {
            if (logger != null) {
                logger.error(e.toString());
            }

            throw new RuntimeException(e);
        }
    }

    /**
     * Reads MetroStationWithFiles object from a TXT file using Stream
     * API.

```

```

    * @param inputFilePath The path of the input TXT file.
    * @return The MetroStationWithFiles object read from the file.
    */
    public static MetroStationWithFiles readFromTXT(String
inputFilePath) {
        if (logger != null) {
            logger.info("Read from TXT file");
        }

        try {
            MetroStationWithFiles metroStation = new
MetroStationWithFiles();
            metroStation.fromListOfStrings(
                Files.lines(Path.of(inputFilePath))
                    .collect(Collectors.toList())
            );

            return metroStation;
        } catch (IOException e) {
            if (logger != null) {
                logger.error(e.toString());
            }

            throw new RuntimeException(e);
        }
    }

    /**
     * Serializes MetroStationWithFiles object to an XML file using the
XStream library.
     * @param outputFilePath The path of the output XML file.
     * @param metroStation The MetroStationWithFiles object to
serialize.
     */
    public static void serializeToXML(String outputFilePath,
MetroStationWithFiles metroStation) {
        if (logger != null) {
            logger.info("Serializing to XML");
        }

        XStream xStream = new XStream();
        xStream.alias("metroStation", MetroStationWithFiles.class);
        xStream.alias("hour", Hour.class);
        String xml = xStream.toXML(metroStation);

        try {
            Files.write(Path.of(outputFilePath), xml.getBytes());
        } catch (IOException e) {
            if (logger != null) {

```

```

        logger.error(e.toString());
    }

    throw new RuntimeException(e);
}

/**
 * Deserializes MetroStationWithFiles object from a JSON file using
 * the XStream library.
 * @param inputFilePath The path of the input JSON file.
 * @return The MetroStationWithFiles object deserialized from the
 * file.
 */
public static MetroStationWithFiles deserializeFromXML(String
inputFilePath) {
    if (logger != null) {
        logger.info("Deserializing from XML");
    }

    try {
        XStream xStream = new XStream();
        xStream.addPermission(AnyTypePermission.ANY);

        xStream.alias("metroStation", MetroStationWithFiles.class);
        xStream.alias("hour", Hour.class);

        String xmlContent =
Files.lines(Path.of(inputFilePath)).collect(Collectors.joining());
        MetroStationWithFiles metroStation = (MetroStationWithFiles)
xStream.fromXML(xmlContent);

        return metroStation;
    } catch (Exception e) {
        if (logger != null) {
            logger.error(e.toString());
        }

        throw new RuntimeException(e);
    }
}

/**
 * Serializes MetroStationWithFiles object to a JSON file using the
 * XStream library.
 * @param outputFilePath The path to the output JSON file.
 * @param metroStation The MetroStationWithFiles object to be
 * serialized.
 */

```

```

    public static void serializeToJSON(String outputPath,
MetroStationWithFiles metroStation) {
        if (logger != null) {
            logger.info("Serializing to JSON");
        }

        XStream xStream = new XStream(new JettisonMappedXmlDriver());

        xStream.alias("metroStation", MetroStationWithFiles.class);
        xStream.alias("hour", Hour.class);

        String json = xStream.toXML(metroStation);

        try {
            Files.write(Path.of(outputFilePath), json.getBytes());
        } catch (IOException e) {
            if (logger != null) {
                logger.error(e.toString());
            }

            throw new RuntimeException(e);
        }
    }

    /**
     * Deserializes MetroStationWithFiles object from a JSON file using
     the XStream library.
     * @param inputFilePath The path to the input JSON file.
     * @return The deserialized MetroStationWithFiles object.
     */
    public static MetroStationWithFiles deserializeFromJSON(String
inputFilePath) {
        if (logger != null) {
            logger.info("Deserializing from JSON");
        }

        try {
            XStream xStream = new XStream(new
JettisonMappedXmlDriver());
            xStream.addPermission(AnyTypePermission.ANY);

            xStream.alias("metroStation", MetroStationWithFiles.class);
            xStream.alias("hour", Hour.class);

            String jsonContent =
Files.lines(Path.of(inputFilePath)).collect(Collectors.joining());
            MetroStationWithFiles metroStation = (MetroStationWithFiles)
xStream.fromXML(jsonContent);

```



```

        return metroStation;
    } catch (Exception e) {
        if (logger != null) {
            logger.error(e.toString());
        }

        throw new RuntimeException(e);
    }
}
}

```

1.2.7. MetroStationWithFiles.java:

```

package part2.lab3.task1;

import part1.lab4.task1.Hour;
import part2.lab1.task1.MetroStationWithStreams;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Represents a metro station with file-related operations.
 * Stream API tools are used for processing and outputting sequences.
 * This class is inherited from the {@link MetroStationWithStreams}.
 */
public class MetroStationWithFiles extends MetroStationWithStreams {
    /**
     * Default constructor.
     */
    public MetroStationWithFiles() {

    }

    /**
     * Constructor with parameters.
     * @param name The name of the metro station.
     * @param opened The year the metro station was opened.
     * @param hours The list of hours for the metro station.
     */
    public MetroStationWithFiles(String name, int opened, List<Hour>
hours) {
        super(name, opened, hours);
    }
}

```

```

/**
 * Converts the metro station information to a list of strings using
 * Stream API.
 * @return The list of strings representing the metro station
 * information.
 */
public List<String> toListOfStrings() {
    ArrayList<String> stringList = new ArrayList<>();
    stringList.add("Station: Name:\t'" + getName() + "'.\tOpened:\t"
+ getOpened() + ".\tHours:");

    if (!getHoursList().isEmpty()) {
        getHoursList().forEach(hour →
            stringList.add("Hour\t{ ridership = " +
hour.getRidership() + ",\tcomment = '" + hour.getComment() + "' }"));
    } else {
        stringList.add("There are no hours for this station.");
    }

    return stringList;
}

/**
 * Populates the metro station information from a list of strings
 * using Stream API.
 * @param stringList The list of strings containing the metro
 * station information.
 */
public void fromListOfStrings(List<String> stringList) {
    Matcher matcher =
Pattern.compile("Name:\\s*'([^']*)'.").matcher(stringList.get(0));
    setName(matcher.find() ? matcher.group(1) : "");
    matcher =
Pattern.compile("Opened:\\s*(\\d{4}).").matcher(stringList.get(0));
    setOpened(Integer.parseInt(matcher.find() ? matcher.group(1) :
"0"));
    stringList.remove(0);

    stringList.forEach(s → {
        Matcher match =
Pattern.compile("ridership\\s*=\\s*(\\d+)").matcher(s);
        String ridership = match.find() ? match.group(1) : "0";
        match =
Pattern.compile("comment\\s*=\\s*'([^']*)'").matcher(s);
        String comment = match.find() ? match.group(1) : "";
        addHour(Integer.parseInt(ridership), comment);
    });
}

```

```

    /**
     * Creates a sample MetroStationWithFiles object for demonstration
     purposes.
     * @return A sample MetroStationWithFiles object.
     */
    public static MetroStationWithFiles createMetroStationWithFiles() {
        MetroStationWithFiles metroStation = new
MetroStationWithFiles("Politekhnikhna", 1984, /*new ArrayList<>()*/
        Arrays.asList(
            new Hour(1100, "Very high ridership"),
            new Hour(110, "Low ridership"),
            new Hour(650, "High ridership"),
            new Hour(532, "High ridership"),
            new Hour(60, "Very low ridership"),
            new Hour(188, "Low ridership"),
            new Hour(200, "Medium ridership")
        ));
        metroStation.sortByDecreasingRidership();

        return metroStation;
    }
}

```

1.2.8. MetroStationWithFilesDemo.java:

```

package part2.lab3.task1;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import static part2.lab3.task1.FileUtils.*;

/**
 * The {@code MetroStationWithFilesDemo} class is a demonstration of
 functionality of the {@link MetroStationWithFiles}.
 */
public class MetroStationWithFilesDemo {
    private static final String TXT_FILE_PATH =
"src/main/resources/part2/lab3/task1/MetroStation.txt";
    private static final String XML_FILE_PATH =
"src/main/resources/part2/lab3/task1/MetroStation.xml";
    private static final String JSON_FILE_PATH =
"src/main/resources/part2/lab3/task1/MetroStation.json";

    /**
     * Performs demonstration of functionality of the {@link
MetroStationWithFiles}.

```

```

* The {@code args} are not used.
* @param args the command-line arguments.
*/
public static void main(String[] args) {
    Logger logger =
LogManager.getLogger(MetroStationWithFilesDemo.class);
    FileUtils.setLogger(logger);
    logger.info("Program started");

    MetroStationWithFiles metroStation =
MetroStationWithFiles.createMetroStationWithFiles();

    writeToTXT(TXT_FILE_PATH, metroStation);
    metroStation = readFromTXT(TXT_FILE_PATH);
    logger.info("Metro Station read from TXT file:\n" +
metroStation);

    serializeToXML(XML_FILE_PATH, metroStation);
    metroStation = deserializeFromXML(XML_FILE_PATH);
    logger.info("Metro Station deserialize from XML file:\n" +
metroStation);

    serializeToJSON(JSON_FILE_PATH, metroStation);
    metroStation = deserializeFromJSON(JSON_FILE_PATH);
    logger.info("Metro Station deserialize from JSON file:\n" +
metroStation);

    logger.info("Program finished");
}
}

```

1.3. Программний код модульного тестування з використанням Junit завдання № 1

1.3.1. MetroStationWithFilesTest.java:

```

package part2.lab3.task1;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import part1.lab4.task1.Hour;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;

```

```

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

import static org.junit.jupiter.api.Assertions.*;

public class MetroStationWithFilesTest {
    private MetroStationWithFiles metroStation;

    @BeforeEach
    public void setup() {
        metroStation = new MetroStationWithFiles("Politekhnichna", 1984,
            Arrays.asList(
                new Hour(1100, "Very high ridership"),
                new Hour(110, "Low ridership"),
                new Hour(650, "High ridership"),
                new Hour(532, "High ridership"),
                new Hour(60, "Very low ridership"),
                new Hour(188, "Low ridership"),
                new Hour(200, "Medium ridership")
            ));
        metroStation.sortByDecreasingRidership();
    }

    @Nested
    class TestMetroStationWithFiles {
        @Test
        @DisplayName("Should create Metro Station with sorted by
decreasing ridership Hours")
        public void testSortedMetroStationCreation() {
            MetroStationWithFiles actual =
MetroStationWithFiles.createMetroStationWithFiles();

            assertEquals(
                metroStation,
                actual,
                "The Metro Station objects should match");
            assertEquals(
                metroStation.getName(),
                actual.getName(),
                "The Names of Metro Stations should match");
            assertEquals(
                metroStation.getOpened(),
                actual.getOpened(),
                "The Opened years of Metro Stations should match");
            assertEquals(
                metroStation.getHoursList().toArray(),
                actual.getHoursList().toArray(),
                "The list of Hours of Metro Stations should match");
        }
    }
}

```

```

    }

    @Test
    @DisplayName("Should match the lists of strings (lines) with the
data of Metro Stations")
    public void testToListOfStrings() {
        MetroStationWithFiles actual =
MetroStationWithFiles.createMetroStationWithFiles();
        assertLinesMatch(
            metroStation.toListOfStrings(),
            actual.toListOfStrings());
        assertEquals(
            metroStation.toListOfStrings().toString(),
            actual.toListOfStrings().toString());
    }

    @Test
    @DisplayName("Should match the MetroStationWithFiles objects
parsed from the list of strings (lines)")
    public void testFromListOfStrings() {
        String file =
"src/main/resources/part2/lab3/task1/MetroStation.txt";
        List<String> lines = null;

        try {
            lines =
Files.lines(Path.of(file)).collect(Collectors.toList());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        MetroStationWithFiles actual = new MetroStationWithFiles();
        actual.fromListOfStrings(lines);

        assertEquals(metroStation, actual);
    }
}

@Nested
class TestTXTFileIO {
    @Test
    @DisplayName("Should match the input TXT file contents when
writing to TXT")
    public void testWriteToTXT() throws IOException {
        String actualFile =
"src/main/resources/part2/lab3/task1/testWriteToTXT.txt";
        Path actualPath = Path.of(actualFile);
        FileUtils.writeToTXT(actualFile, metroStation);
        assertTrue(Files.exists(actualPath));
    }
}

```

```

        List<String> lines = null;

        try {
            lines =
Files.lines(actualPath).collect(Collectors.toList());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        assertFalse(lines.isEmpty());
        assertLinesMatch(metroStation.toListOfStrings(), lines);

        Files.deleteIfExists(actualPath);
    }

    @Test
    @DisplayName("Should throw exception when writing to TXT with
invalid path to file")
    public void testWriteToTXTException() {
        String invalidPath = "invalid/path/MetroStation.txt";
        assertThrows(RuntimeException.class, () →
FileUtils.writeToTXT(invalidPath, metroStation));
    }

    @Test
    @DisplayName("Should match the MetroStationWithFiles objects
when reading from TXT file")
    public void testReadFromTXT() throws IOException {
        String actualFile =
"src/main/resources/part2/lab3/task1/MetroStation.txt";
        MetroStationWithFiles actual =
FileUtils.readFromTXT(actualFile);

        assertEquals(
            metroStation,
            actual,
            "The Metro Station objects should match");
        assertEquals(
            metroStation.getName(),
            actual.getName(),
            "The Names of Metro Stations should match");
        assertEquals(
            metroStation.getOpened(),
            actual.getOpened(),
            "The Opened years of Metro Stations should match");
        assertEquals(
            metroStation.getHoursList().toArray(),
            actual.getHoursList().toArray(),

```

```

        "The list of Hours of Metro Stations should match");
    }

    @Test
    @DisplayName("Should throw exception when reading from TXT with
invalid path to file")
    public void testReadFromTXTException() {
        String invalidPath = "invalid/path/MetroStation.txt";
        assertThrows(RuntimeException.class, () →
FileUtils.readFromTXT(invalidPath));
    }
}

@Nested
class TestXMLFileSerialization {
    @Test
    @DisplayName("Should match the input XML file contents when
serializing to XML")
    public void testSerializeToXML() throws IOException {
        String actualFile =
"src/main/resources/part2/lab3/task1/testSerializeToXML.txt";
        Path actualPath = Path.of(actualFile);
        FileUtils.serializeToXML(actualFile, metroStation);
        assertTrue(Files.exists(actualPath));

        String expectedFile =
"src/main/resources/part2/lab3/task1/MetroStation.xml";
        Path expectedPath = Path.of(expectedFile);
        assertTrue(Files.exists(expectedPath));

        List<String> expectedLines = null;
        List<String> actualLines = null;

        try {
            expectedLines =
Files.lines(expectedPath).collect(Collectors.toList());
            actualLines =
Files.lines(actualPath).collect(Collectors.toList());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        assertFalse(expectedLines.isEmpty());
        assertFalse(actualLines.isEmpty());
        assertLinesMatch(expectedLines, actualLines);

        Files.deleteIfExists(actualPath);
    }
}

```



```

    @Test
    @DisplayName("Should throw exception when serialize to XML with
invalid path to file")
    public void testSerializeToXMLException() {
        String invalidPath = "invalid/path/MetroStation.xml";
        assertThrows(RuntimeException.class, () →
FileUtils.serializeToXML(invalidPath, metroStation));
    }

    @Test
    @DisplayName("Should match the MetroStationWithFiles objects
when deserializing from XML file")
    public void testDeserializeFromXML() throws IOException {
        String actualFile =
"src/main/resources/part2/lab3/task1/MetroStation.xml";
        MetroStationWithFiles actual =
FileUtils.deserializeFromXML(actualFile);

        assertEquals(
            metroStation,
            actual,
            "The Metro Station objects should match");
        assertEquals(
            metroStation.getName(),
            actual.getName(),
            "The Names of Metro Stations should match");
        assertEquals(
            metroStation.getOpened(),
            actual.getOpened(),
            "The Opened years of Metro Stations should match");
        assertEquals(
            metroStation.getHoursList().toArray(),
            actual.getHoursList().toArray(),
            "The list of Hours of Metro Stations should match");
    }

    @Test
    @DisplayName("Should throw exception when deserializing from XML
with invalid path to file")
    public void testDeserializeFromXMLException() {
        String invalidPath = "invalid/path/MetroStation.xml";
        assertThrows(RuntimeException.class, () →
FileUtils.deserializeFromXML(invalidPath));
    }
}

@Nested
class TestJSONFileSerialization {
    @Test

```

```

        @DisplayName("Should match the input JSON file contents when
serializing to JSON")
        public void testSerializeToJSON() throws IOException {
            String actualFile =
"src/main/resources/part2/lab3/task1/testSerializeToJSON.json";
            Path actualPath = Path.of(actualFile);
            FileUtils.serializeToJSON(actualFile, metroStation);
            assertTrue(Files.exists(actualPath));

            String expectedFile =
"src/main/resources/part2/lab3/task1/MetroStation.json";
            Path expectedPath = Path.of(expectedFile);
            assertTrue(Files.exists(expectedPath));

            List<String> expectedLines = null;
            List<String> actualLines = null;

            try {
                expectedLines =
Files.lines(expectedPath).collect(Collectors.toList());
                actualLines =
Files.lines(actualPath).collect(Collectors.toList());
            } catch (IOException e) {
                throw new RuntimeException(e);
            }

            assertFalse(expectedLines.isEmpty());
            assertFalse(actualLines.isEmpty());
            assertLinesMatch(expectedLines, actualLines);

            Files.deleteIfExists(actualPath);
        }

        @Test
        @DisplayName("Should throw exception when serialize to JSON with
invalid path to file")
        public void testSerializeToJSONException() {
            String invalidPath = "invalid/path/MetroStation.json";
            assertThrows(RuntimeException.class, () →
FileUtils.serializeToJSON(invalidPath, metroStation));
        }

        @Test
        @DisplayName("Should match the MetroStationWithFiles objects
when deserializing from JSON file")
        public void testDeserializeFromJSON() throws IOException {
            String outputFile =
"src/main/resources/part2/lab3/task1/MetroStation.json";
            MetroStationWithFiles actual =

```

```

FileUtils.deserializeFromJSON(outputFile);

    assertEquals(
        metroStation,
        actual,
        "The Metro Station objects should match");
    assertEquals(
        metroStation.getName(),
        actual.getName(),
        "The Names of Metro Stations should match");
    assertEquals(
        metroStation.getOpened(),
        actual.getOpened(),
        "The Opened years of Metro Stations should match");
    assertEquals(
        metroStation.getHoursList().toArray(),
        actual.getHoursList().toArray(),
        "The list of Hours of Metro Stations should match");
}

@Test
@DisplayName("Should throw exception when deserializing from
JSON with invalid path to file")
public void testDeserializeFromJSONException() {
    String invalidPath = "invalid/path/MetroStation.json";
    assertThrows(RuntimeException.class, () →
FileUtils.deserializeFromJSON(invalidPath));
}
}
}

```

1.4. Екранні форми за результатами роботи програмного коду завдання № 1

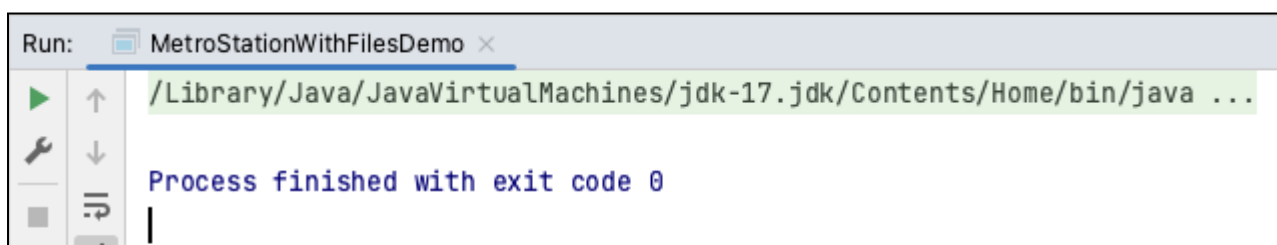


Рисунок 1.4.1 – Результати роботи програмного коду

```

1 Station: Name: 'Politekhnichna'. Opened: 1984. Hours:
2 Hour { ridership = 1100, comment = 'Very high ridership' }
3 Hour { ridership = 650, comment = 'High ridership' }
4 Hour { ridership = 532, comment = 'High ridership' }
5 Hour { ridership = 200, comment = 'Medium ridership' }
6 Hour { ridership = 188, comment = 'Low ridership' }
7 Hour { ridership = 110, comment = 'Low ridership' }
8 Hour { ridership = 60, comment = 'Very low ridership' }
9 |

```

Рисунок 1.4.2 – Вміст файлу “MetroStation.txt”

```

1 <metroStation>
2   <name>Politekhnichna</name>
3   <opened>1984</opened>
4   <hours>
5     <hour>
6       <ridership>1100</ridership>
7       <comment>Very high ridership</comment>
8     </hour>
9     <hour>
10      <ridership>650</ridership>
11      <comment>High ridership</comment>
12    </hour>
13    <hour>
14      <ridership>532</ridership>
15      <comment>High ridership</comment>
16    </hour>
17    <hour>
18      <ridership>200</ridership>
19      <comment>Medium ridership</comment>
20    </hour>
21    <hour>
22      <ridership>188</ridership>
23      <comment>Low ridership</comment>
24    </hour>
25    <hour>
26      <ridership>110</ridership>
27      <comment>Low ridership</comment>
28    </hour>
29    <hour>
30      <ridership>60</ridership>
31      <comment>Very low ridership</comment>
32    </hour>
33  </hours>
34 </metroStation>

```

Рисунок 1.4.3 – Вміст файлу “MetroStation.xml”

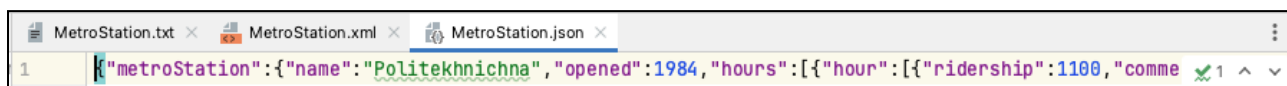


Рисунок 1.4.4 – Вміст файлу “MetroStation.json”

Рисунок 1.4.5 – Вміст файлу “MetroStation.json” у структурованому
представленні

```

MetroStation-2024-05-10.log  log4j2.xml
1  2024-05-10 13:31:39.886 [main] part2.lab3.task1.MetroStationWithFilesDemo - Program started
2  2024-05-10 13:31:39.895 [main] part2.lab3.task1.MetroStationWithFilesDemo - Write to TXT file
3  2024-05-10 13:31:39.914 [main] part2.lab3.task1.MetroStationWithFilesDemo - Read from TXT file
4  2024-05-10 13:31:39.922 [main] part2.lab3.task1.MetroStationWithFilesDemo - Metro Station read from TXT file:
5  Station:   Name: 'Politekhnichna'. Opened: 1984.   Hours:
6  Hour      { ridership = 1100, comment = 'Very high ridership' }
7  Hour      { ridership = 650,  comment = 'High ridership' }
8  Hour      { ridership = 532,  comment = 'High ridership' }
9  Hour      { ridership = 200,  comment = 'Medium ridership' }
10 Hour      { ridership = 188,  comment = 'Low ridership' }
11 Hour      { ridership = 110,  comment = 'Low ridership' }
12 Hour      { ridership = 60,   comment = 'Very low ridership' }
13
14 2024-05-10 13:31:39.922 [main] part2.lab3.task1.MetroStationWithFilesDemo - Serializing to XML
15 2024-05-10 13:31:40.062 [main] part2.lab3.task1.MetroStationWithFilesDemo - Deserializing from XML
16 2024-05-10 13:31:40.070 [main] part2.lab3.task1.MetroStationWithFilesDemo - Metro Station deserialize from XML file:
17 Station:   Name: 'Politekhnichna'. Opened: 1984.   Hours:
18 Hour      { ridership = 1100, comment = 'Very high ridership' }
19 Hour      { ridership = 650,  comment = 'High ridership' }
20 Hour      { ridership = 532,  comment = 'High ridership' }
21 Hour      { ridership = 200,  comment = 'Medium ridership' }
22 Hour      { ridership = 188,  comment = 'Low ridership' }
23 Hour      { ridership = 110,  comment = 'Low ridership' }
24 Hour      { ridership = 60,   comment = 'Very low ridership' }
25
26 2024-05-10 13:31:40.070 [main] part2.lab3.task1.MetroStationWithFilesDemo - Serializing to JSON
27 2024-05-10 13:31:40.077 [main] part2.lab3.task1.MetroStationWithFilesDemo - Deserializing from JSON
28 2024-05-10 13:31:40.080 [main] part2.lab3.task1.MetroStationWithFilesDemo - Metro Station deserialize from JSON file:
29 Station:   Name: 'Politekhnichna'. Opened: 1984.   Hours:
30 Hour      { ridership = 1100, comment = 'Very high ridership' }
31 Hour      { ridership = 650,  comment = 'High ridership' }
32 Hour      { ridership = 532,  comment = 'High ridership' }
33 Hour      { ridership = 200,  comment = 'Medium ridership' }
34 Hour      { ridership = 188,  comment = 'Low ridership' }
35 Hour      { ridership = 110,  comment = 'Low ridership' }
36 Hour      { ridership = 60,   comment = 'Very low ridership' }
37
38 2024-05-10 13:31:40.080 [main] part2.lab3.task1.MetroStationWithFilesDemo - Program finished
39

```

Рисунок 1.4.6 – Вміст файлу “MetroStation.log”

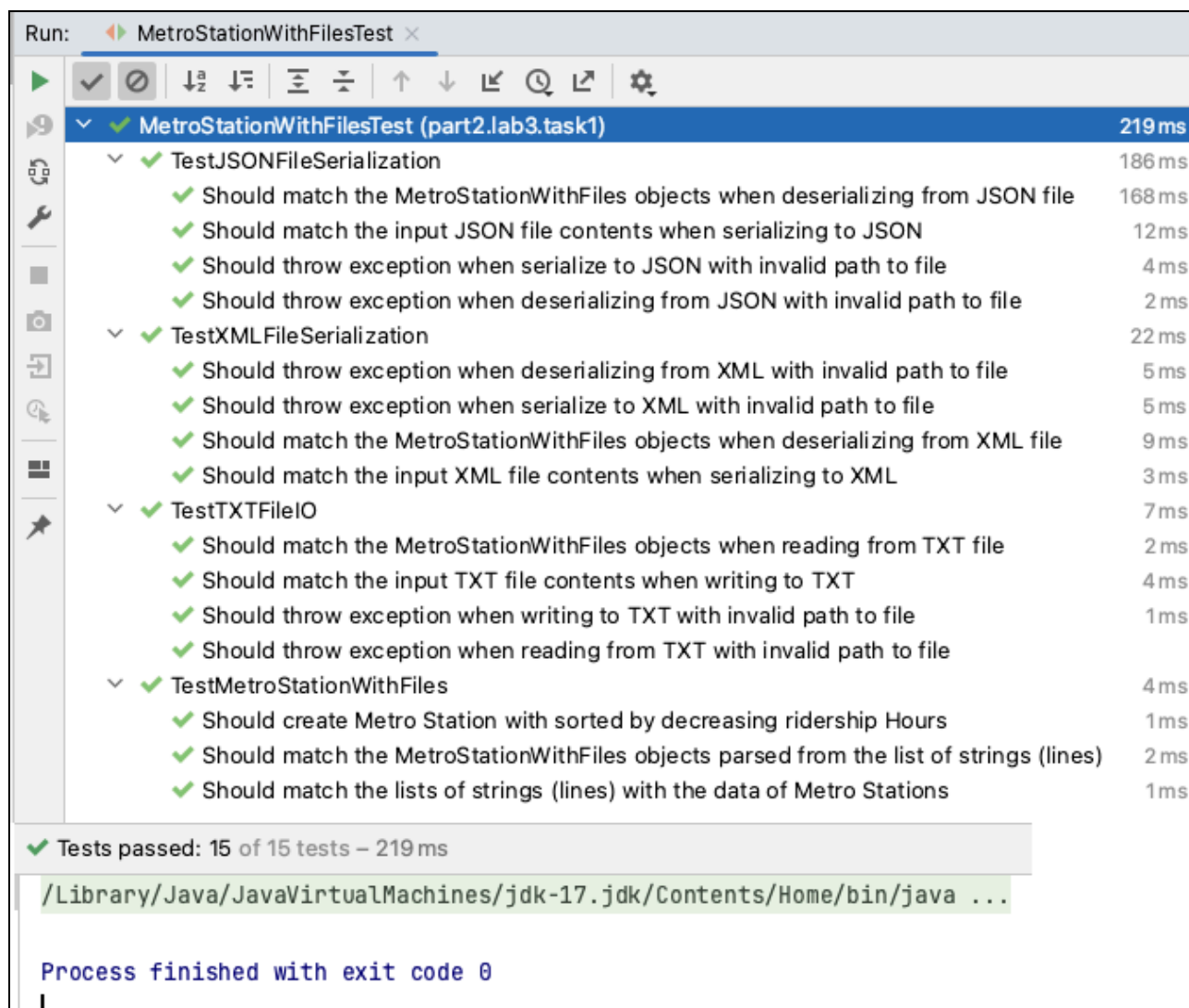


Рисунок 1.4.7 – Результати модульного тестування з використанням JUnit програмного коду

2. Завдання №2 до лабораторної роботи

2.1. Список файлів усіх підкаталогів

Увести з клавіатури ім'я певної теки. Вивести на екран імена усіх файлів цієї теки, а також усіх файлів підкаталогів, їхніх підкаталогів тощо. Реалізувати два підходи:

- пошук за допомогою класу `java.io.File` через рекурсивну функцію;
- пошук засобами пакету `java.nio.file`.

Обидва результати послідовно вивести на екран. Якщо тека не існує, вивести повідомлення про помилку.

2.2. Програмний код реалізації завдання № 2

2.2.1. DirectoryLister.java:

```
package part2.lab3.task2;

import java.io.File;
import java.io.IOException;
import java.nio.file.*;
import java.nio.file.attribute.BasicFileAttributes;
import java.util.Arrays;

public class DirectoryLister {
    public static void listFilesRecursivelyByIOFileClass(File directory)
    throws NullPointerException {
        Arrays.stream(directory.listFiles())
            .forEach(file → {
                if (file.isDirectory()) {
                    System.out.println("Directory:\t" +
file.getAbsolutePath());
                    listFilesRecursivelyByIOFileClass(file);
                } else {
                    System.out.println("\tFile:\t" +
file.getAbsolutePath());
                }
            });
    }

    public static void listFilesRecursivelyByNIO(Path directory) throws
IOException {
        Files.walkFileTree(directory, new SimpleFileVisitor<Path>() {
            @Override
            public FileVisitResult preVisitDirectory(Path dir,
BasicFileAttributes attrs) {
                System.out.println("Directory:\t" + dir);
                return FileVisitResult.CONTINUE;
            }

            @Override
            public FileVisitResult visitFile(Path file,
BasicFileAttributes attrs) {
                System.out.println("\tFile:\t" + file);
                return FileVisitResult.CONTINUE;
            }

            @Override
            public FileVisitResult visitFileFailed(Path file,
IOException e) {
                System.err.println(e);
                return FileVisitResult.CONTINUE;
            }
        });
    }
}
```



```

        }
    });
}
}

```

2.2.2. DirectoryListerDemo.java:

```

package part2.lab3.task2;

import java.io.File;
import java.io.IOException;
import java.util.Arrays;

import static
part2.lab3.task2.DirectoryLister.listFilesRecursivelyByIOFileClass;
import static
part2.lab3.task2.DirectoryLister.listFilesRecursivelyByNIO;

public class DirectoryListerDemo {
    private static String[] DIR_PATHES = {
        "src/main/java/part2/lab2",
        "invalid_path",
        "src/main/resources/part2/lab3/task2"
    };

    public static void main(String[] args) {
        Arrays.stream(DIR_PATHES)
            .forEach(path → {
                System.out.println("\nThe directory path:\t" +
path);

                File directory = new File(path);

                if (!directory.exists()
                    || !directory.isDirectory()) {
                    System.out.println("The given directory does not
exist");
                    return;
                }

                System.out.println("List subdirectories and files
recursively by java.io.File:");

                try {
                    listFilesRecursivelyByIOFileClass(directory);
                } catch (NullPointerException e) {
                    System.err.println(e.getMessage());
                }
            });
    }
}

```

```

        System.out.println("List subdirectories and files
recursively by java.nio.file:");

        try {
            listFilesRecursivelyByNIO(directory.toPath());
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    });
}
}

```

2.3. Екранні форми за результатами роботи програмного коду завдання № 2

```

inessarepeshko@MacBook-Air task2 % chmod 000 test/restricted_file.txt
inessarepeshko@MacBook-Air task2 % ls -laR
total 0
drwxr-xr-x  4 inessarepeshko  staff  128 May  8 13:49 .
drwxr-xr-x  9 inessarepeshko  staff  288 May  8 13:46 ..
-rwxrwxrwx  1 inessarepeshko  staff    0 May  8 13:47 restricted_file.txt
drwxrwxrwx  3 inessarepeshko  staff   96 May  8 13:47 test

./test:
total 0
drwxrwxrwx  3 inessarepeshko  staff   96 May  8 13:47 .
drwxr-xr-x  4 inessarepeshko  staff  128 May  8 13:49 ..
-----  1 inessarepeshko  staff    0 May  8 13:47 restricted_file.txt
inessarepeshko@MacBook-Air task2 % chmod 000 test
inessarepeshko@MacBook-Air task2 % ls -laR
total 0
drwxr-xr-x  4 inessarepeshko  staff  128 May  8 13:49 .
drwxr-xr-x  9 inessarepeshko  staff  288 May  8 13:46 ..
-rwxrwxrwx  1 inessarepeshko  staff    0 May  8 13:47 restricted_file.txt
d-----  3 inessarepeshko  staff   96 May  8 13:47 test

./test:
total 0
ls: ./test: Permission denied
inessarepeshko@MacBook-Air task2 %

```

Рисунок 2.3.1 – Структура тестових файлів із відображенням модифікаторів доступу до них

```

Run: DirectoryListerDemo x
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...

The directory path: src/main/java/part2/lab2
List subdirectories and files recursively by java.io.File:
Directory: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task1
File: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task1/MetroStationWithLocalization.java
File: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task1/HourWithDates.java
Directory: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task_control
File: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task_control/TimeZonesPrinter.java
Directory: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task3
File: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task3/PhoneNumberValidator.java
File: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task3/PhoneNumberValidatorDemo.java
Directory: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task4
File: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task4/PasswordValidatorDemo.java
File: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task4/PasswordValidator.java
Directory: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task5
File: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task5/SubstringsParser.java
File: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task5/SubstringsParserDemo.java
Directory: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task2
File: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task2/DateValidator.java
File: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/java/part2/lab2/task2/DateValidatorDemo.java

List subdirectories and files recursively by java.nio.file:
Directory: src/main/java/part2/lab2
Directory: src/main/java/part2/lab2/task1
File: src/main/java/part2/lab2/task1/MetroStationWithLocalization.java
File: src/main/java/part2/lab2/task1/HourWithDates.java
Directory: src/main/java/part2/lab2/task_control
File: src/main/java/part2/lab2/task_control/TimeZonesPrinter.java
Directory: src/main/java/part2/lab2/task3
File: src/main/java/part2/lab2/task3/PhoneNumberValidator.java
File: src/main/java/part2/lab2/task3/PhoneNumberValidatorDemo.java
Directory: src/main/java/part2/lab2/task4
File: src/main/java/part2/lab2/task4/PasswordValidatorDemo.java
File: src/main/java/part2/lab2/task4/PasswordValidator.java
Directory: src/main/java/part2/lab2/task5
File: src/main/java/part2/lab2/task5/SubstringsParser.java
File: src/main/java/part2/lab2/task5/SubstringsParserDemo.java
Directory: src/main/java/part2/lab2/task2
File: src/main/java/part2/lab2/task2/DateValidator.java
File: src/main/java/part2/lab2/task2/DateValidatorDemo.java

The directory path: invalid_path
The given directory does not exist

The directory path: src/main/resources/part2/lab3/task2
List subdirectories and files recursively by java.io.File:
Directory: /Users/inessarepeshko/university_code/semester4/advanced_java_programming_course/java-advanced/src/main/resources/part2/lab3/task2/test
List subdirectories and files recursively by java.nio.file:
Directory: src/main/resources/part2/lab3/task2
File: src/main/resources/part2/lab3/task2/restricted_file.txt
Cannot read the array length because "array" is null
java.nio.file.AccessDeniedException: src/main/resources/part2/lab3/task2/test

Process finished with exit code 0

```

Рисунок 2.3.2 – Результати роботи програмного коду

3. Завдання №3 до лабораторної роботи

3.1. Робота з текстовими файлами засобами Stream API

Прочитати функцією `Files.lines()` рядки з текстового файлу, розсортувати за збільшенням довжини й вивести в інший файл рядки, які містять літеру "a".

3.2. Програмний код реалізації завдання № 3

3.2.1. TextFileProcessor.java:

```

package part2.lab3.task3;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Comparator;
import java.util.List;

```

```

import java.util.stream.Collectors;

public class TextFileProcessor {
    public static List<String> readLines(String inputFilePath) throws
IOException {
        return Files.lines(Paths.get(inputFilePath))
            .collect(Collectors.toList());
    }

    public static List<String> sortLinesByLength(List<String> lines) {
        return lines.stream()
            .sorted(Comparator.comparingInt(String::length))
            .collect(Collectors.toList());
    }

    public static List<String> filterLinesContainingWordFragment(List<String>
lines, String fragment) {
        return lines.stream()
            .filter(line → line.contains(fragment))
            .collect(Collectors.toList());
    }

    public static void writeLinesToFile(String outputFilePath, List<String>
lines) throws IOException {
        Files.write(Paths.get(outputFilePath), lines);
    }

    public static void printContentsOfFile(String filePath) throws IOException
{
        System.out.println("The contents of the file \"" +
getFileName(filePath) + "\"");

        try {
            Files.readAllLines(Paths.get(filePath)).stream().forEach(System.out::println);
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }

    public static String getFileName(String filePath) {
        return new File(filePath).getName();
    }
}

```

3.2.2. TextFileProcessorDemo.java:

```

package part2.lab3.task3;

import java.io.IOException;
import java.util.List;

```

```

import static part2.lab3.task3.TextFileProcessor.*;

public class TextFileProcessorDemo {
    private static String inputFilePath =
"src/main/resources/part2/lab3/task3/input.txt";
    private static String outputFilePath =
"src/main/resources/part2/lab3/task3/output.txt";

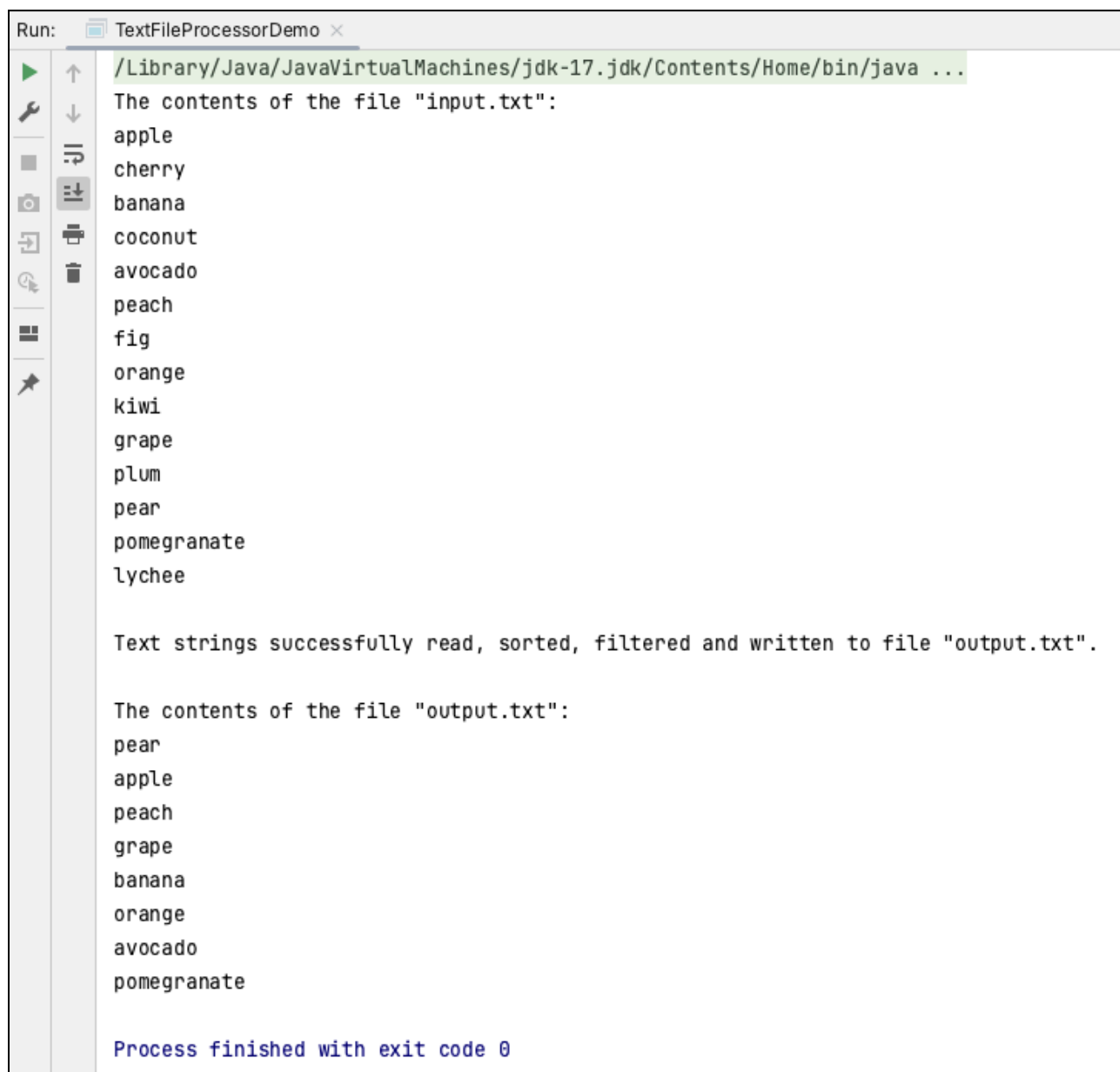
    public static void main(String[] args) {
        try {
            printContentsOfFile(inputFilePath);

            List<String> filteredLines = filterLinesContainingWordFragment(
                sortLinesByLength(readLines(inputFilePath)),
                "a");
            writeLinesToFile(outputFilePath, filteredLines);
            System.out.println("\nText strings successfully read, sorted,
filtered and written to file \"" + getFileName(outputFilePath) + "\".\n");

            printContentsOfFile(outputFilePath);
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }
}

```

3.3. Екранні форми за результатами роботи програмного коду завдання № 3



```
Run: TextFileProcessorDemo x
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...
The contents of the file "input.txt":
apple
cherry
banana
coconut
avocado
peach
fig
orange
kiwi
grape
plum
pear
pomegranate
lychee

Text strings successfully read, sorted, filtered and written to file "output.txt".

The contents of the file "output.txt":
pear
apple
peach
grape
banana
orange
avocado
pomegranate

Process finished with exit code 0
```

Рисунок 3.3.1 – Результаты работы программного коду

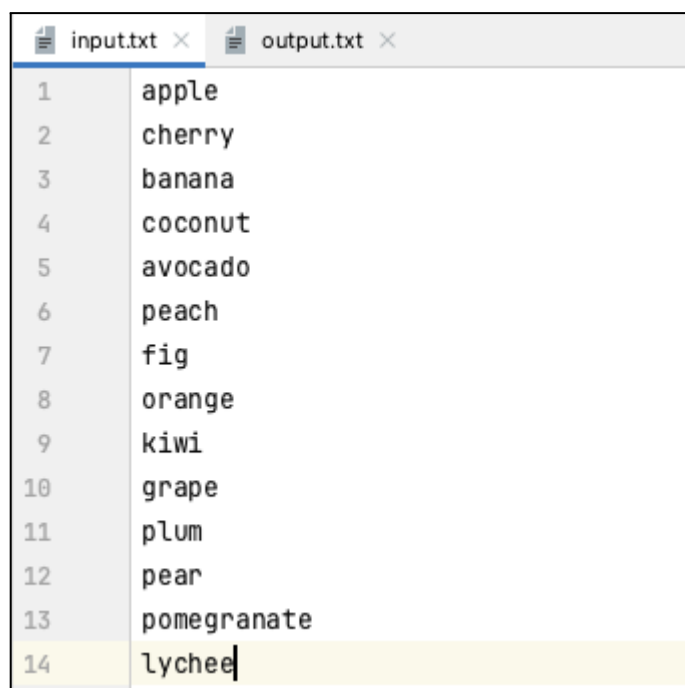


Рисунок 3.3.2 – Вміст файлу “input.txt”

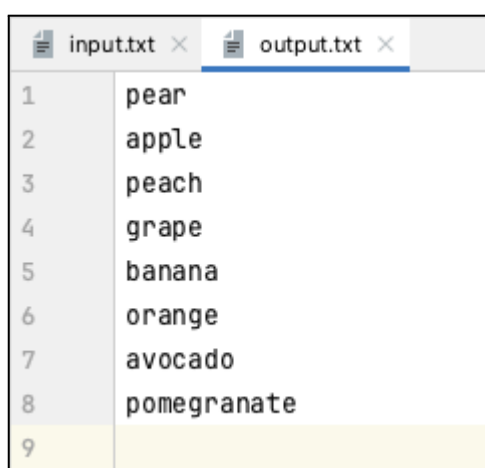


Рисунок 3.3.3 – Вміст файлу “output.txt”

4. Завдання №4 до лабораторної роботи

4.1. Створення файлів і читання з файлів даних про студента й академічну групу

Описати класи Студент і Академічна група (з полем – масивом студентів). Створити об'єкти. Забезпечити створення файлів і читання з файлів, застосувавши такі підходи:

- використання засобів Stream API для роботи з текстовими файлами;

- серіалізація й десеріалізація в XML і JSON(засобами XStream).

4.2. Програмний код реалізації завдання № 4

4.2.1. Student.java:

```
package part2.lab3.task4;

import java.util.regex.Pattern;

public class Student {
    private String fullName;

    private Double ratingScore;

    public Student() {}

    public Student(String fullName, Double ratingScore) throws
    IllegalArgumentException {
        if (fullName.isEmpty()) {
            throw new IllegalArgumentException("The full name cannot be
null.");
        }

        if (!Pattern.compile("[A-Z][a-z]+
[A-Z][a-z]+$").matcher(fullName).matches()) {
            throw new IllegalArgumentException("The full name must be in
the format \"Surname Name\".");
        }

        if (ratingScore == null || ratingScore ≤ 0) {
            throw new IllegalArgumentException("The rating score must be
greater than zero.");
        }

        this.fullName = fullName;
        this.ratingScore = ratingScore;
    }

    public String getFullName() {
        return fullName;
    }

    public Double getRatingScore() {
        return ratingScore;
    }

    public void setFullName(String fullName) {
        this.fullName = fullName;
    }
}
```



```

    public void setRatingScore(Double ratingScore) {
        this.ratingScore = ratingScore;
    }

    @Override
    public String toString() {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append("\tFull name: \t").append(getFullName())
            .append("\t");
        stringBuilder.append("\tRating score:\t").append(getRatingScore()).append(".");

        return stringBuilder.toString();
    }
}

```

4.2.2. AcademicGroup.java:

```

package part2.lab3.task4;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.regex.Pattern;
import java.util.stream.Collectors;

public class AcademicGroup {
    private String groupNumber;

    private List<Student> studentList = new ArrayList<>();

    public AcademicGroup() {
    }

    public AcademicGroup(String groupNumber) throws
    IllegalArgumentException {
        if (groupNumber.isEmpty()) {
            throw new IllegalArgumentException("The group number cannot
            be null.");
        }

        if
        (!Pattern.compile("[A-Z]{1,3}-\\d{3}[a-z]$").matcher(groupNumber).matc
        hes()) {
            throw new IllegalArgumentException("The group number must be
            in the format \"NNN-000n\");
        }
    }
}

```

```

        setGroupNumber(groupNumber);
    }

    public AcademicGroup(String groupNumber,
                        List<Student> studentList) throws
IllegalArgumentException {
        if (groupNumber.isEmpty()) {
            throw new IllegalArgumentException("The group number cannot
be null.");
        }

        if
(!Pattern.compile("[A-Z]{1,3}-\\d{3}[a-z]$", groupNumber).matc
hes()) {
            throw new IllegalArgumentException("The group number must be
in the format \"NNN-000n\".");
        }

        if (studentList.isEmpty()) {
            throw new IllegalArgumentException("The list of students
cannot be empty.");
        }

        setGroupNumber(groupNumber);
        studentList.forEach(this::addStudent);
    }

    public String getGroupNumber() {
        return this.groupNumber;
    }

    private List<Student> getStudentList() {
        return this.studentList;
    }

    public void setGroupNumber(String groupNumber) {
        this.groupNumber = groupNumber;
    }

    private void addStudent(Student student) {
        this.studentList.add(student);
    }

    @Override
    public String toString() {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append("The group
number:\\t").append(getGroupNumber()).append("\\n")

```

```

        .append("The student list:\n")
        .append(getStudentList().stream()
            .map(Student::toString)
            .collect(Collectors.joining("\n")));

    return stringBuilder.toString();
}

public static AcademicGroup createAcademicGroup() {
    return new AcademicGroup(
        "CS-222a",
        Arrays.asList(
            new Student("Blits Anna", 90.0),
            new Student("Chizh Alexander", 92.30),
            new Student("Ermolov Vladislav", 89.7),
            new Student("Khorunzhii Tetiana", 95.02)
        )
    );
}
}

```

4.2.3. FileUtils.java:

```

package part2.lab3.task4;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.json.JettisonMappedXmlDriver;
import com.thoughtworks.xstream.io.json.JsonHierarchicalStreamDriver;
import com.thoughtworks.xstream.security.AnyTypePermission;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.List;
import java.util.stream.Collectors;

public class FileUtils {
    public static List<String> readFromFile(String filePath) {
        try {
            return
Files.lines(Paths.get(filePath)).collect(Collectors.toList());
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }
}

```

```

        return null;
    }

    public static void printLines(List<String> lines) throws IOException
    {
        if (lines.isEmpty()) {
            System.out.println("The file is empty.");
        } else {
            System.out.println("The contents of the file:");
            lines.forEach(System.out::println);
        }
    }

    public static void writeToFile(String filePath, String data) throws
    IOException {
        try {
            Files.write(Paths.get(filePath), data.getBytes());
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }

    public static void serializeToXML(String inputFilePath,
    AcademicGroup academicGroup) {
        System.out.println("The data to be serialised:\n" +
    academicGroup);

        XStream xStream = new XStream();
        xStream.alias("academicGroup", AcademicGroup.class);
        xStream.alias("student", Student.class);
        String xml = xStream.toXML(academicGroup);

        try {
            Files.write(Paths.get(inputFilePath), xml.getBytes());
            System.out.println("The file \"" + new
    File(inputFilePath).getName() + "\" successfully serialized.\n");
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }

    public static void deserializeFromXML(String outputFilePath, String
    inputFilePath) {
        XStream xStream = new XStream();
        xStream.addPermission(AnyTypePermission.ANY);

        xStream.alias("academicGroup", AcademicGroup.class);
        xStream.alias("student", Student.class);
    
```

```

        AcademicGroup academicGroup = (AcademicGroup)
xStream.fromXML(new File(inputFilePath));

        if (academicGroup != null) {
            try {
                writeFile(outputFilePath, academicGroup.toString());
                System.out.println("The file \"" + new
File(inputFilePath).getName() + "\" successfully deserialized.");
                System.out.println("The deserialized data:\n" +
academicGroup);
            } catch (IOException e) {
                System.err.println(e.getMessage());
            }
        }

        public static void serializeToJSON(String inputFilePath,
AcademicGroup academicGroup) {
            System.out.println("The data to be serialised:\n" +
academicGroup);

            XStream xStream = new XStream(new
JsonHierarchicalStreamDriver());
            xStream.alias("academicGroup", AcademicGroup.class);
            xStream.alias("studentList", Student.class);

            String json = xStream.toXML(academicGroup);

            try {
                Files.write(Paths.get(inputFilePath), json.getBytes());
                System.out.println("The file \"" + new
File(inputFilePath).getName() + "\" successfully serialized.\n");
            } catch (IOException e) {
                System.err.println(e.getMessage());
            }
        }

        public static void deserializeFromJSON(String outputFilePath, String
inputFilePath) {
            XStream xStream = new XStream(new JettisonMappedXmlDriver());
            xStream.addPermission(AnyTypePermission.ANY);

            xStream.alias("academicGroup", AcademicGroup.class);
            xStream.addImplicitCollection(AcademicGroup.class,
"studentList");
            xStream.alias("studentList", Student.class);

            AcademicGroup academicGroup = (AcademicGroup)

```

```

xStream.fromXML(new File(inputFilePath));

    if (academicGroup != null) {
        try {
            writeToFile(outputFilePath, academicGroup.toString());
            System.out.println("The file \" + new
File(inputFilePath).getName() + "\" successfully deserialized.");
            System.out.println("The deserialized data:\n" +
academicGroup);
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }
}
}

```

4.2.4. FileUtilsDemo.java:

```

package part2.lab3.task4;

import java.io.IOException;

import static part2.lab3.task4.FileUtils.*;

public class FileUtilsDemo {
    private static final String XML_INPUT_FILE_PATH =
"src/main/resources/part2/lab3/task4/AcademicGroup.xml";
    private static final String JSON_INPUT_FILE_PATH =
"src/main/resources/part2/lab3/task4/AcademicGroup.json";
    private static final String OUTPUT_FILE_PATH =
"src/main/resources/part2/lab3/task4/AcademicGroup.txt";

    public static void printOutputFiles(String inputFilePath, String
outputFilePath) {
        try {
            System.out.println();
            printLines(readFromFile(inputFilePath));
            System.out.println();
            printLines(readFromFile(outputFilePath));
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }

    public static void testXMLSerialization() {

System.out.println("*****

```

```

*****\n"
        + "XML serialization and deserialization:\n"
        +
"*****\n");

        serializeToXML(XML_INPUT_FILE_PATH,
AcademicGroup.createAcademicGroup());
        deserializeFromXML(OUTPUT_FILE_PATH, XML_INPUT_FILE_PATH);
        printOutputFiles(XML_INPUT_FILE_PATH, OUTPUT_FILE_PATH);
    }

    public static void testJSONbyXStreamSerialization() {

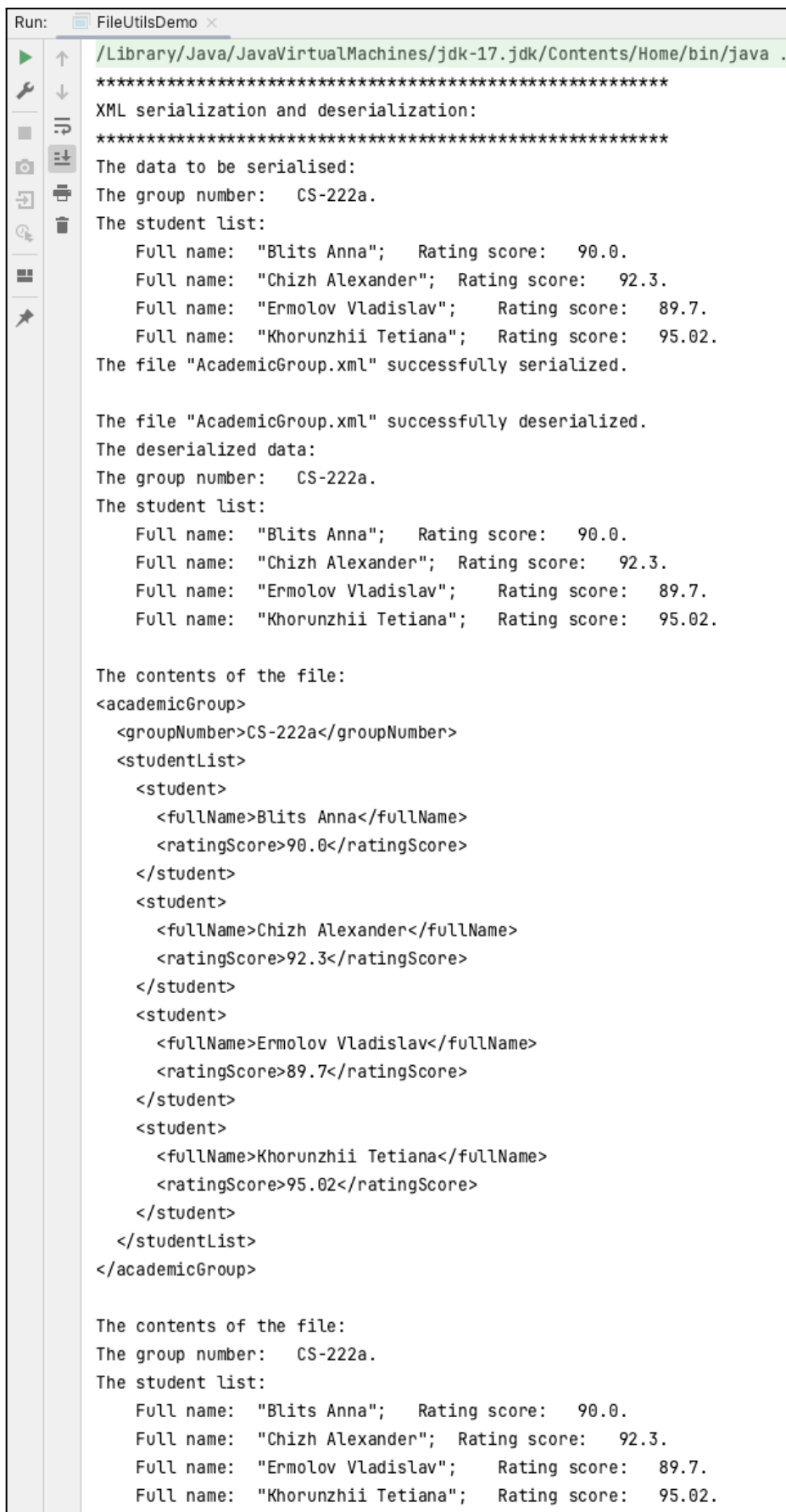
System.out.println("*****\n"
        + "JSON by XStream serialization and deserialization:\n"
        +
"*****\n");

        serializeToJSON(JSON_INPUT_FILE_PATH,
AcademicGroup.createAcademicGroup());
        deserializeFromJSON(OUTPUT_FILE_PATH, JSON_INPUT_FILE_PATH);
        printOutputFiles(JSON_INPUT_FILE_PATH, OUTPUT_FILE_PATH);
    }

    public static void main(String[] args) {
        testXMLSerialization();
        System.out.println("\n");
        testJSONbyXStreamSerialization();
    }
}

```

4.3. Екранні форми за результатами роботи програмного коду завдання № 4



```

Run: FileUtilsDemo x
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java .
*****
XML serialization and deserialization:
*****
The data to be serialised:
The group number:  CS-222a.
The student list:
    Full name:  "Blits Anna";    Rating score:  90.0.
    Full name:  "Chizh Alexander"; Rating score:  92.3.
    Full name:  "Ermolov Vladislav"; Rating score:  89.7.
    Full name:  "Khorunzhii Tetiana"; Rating score:  95.02.
The file "AcademicGroup.xml" successfully serialized.

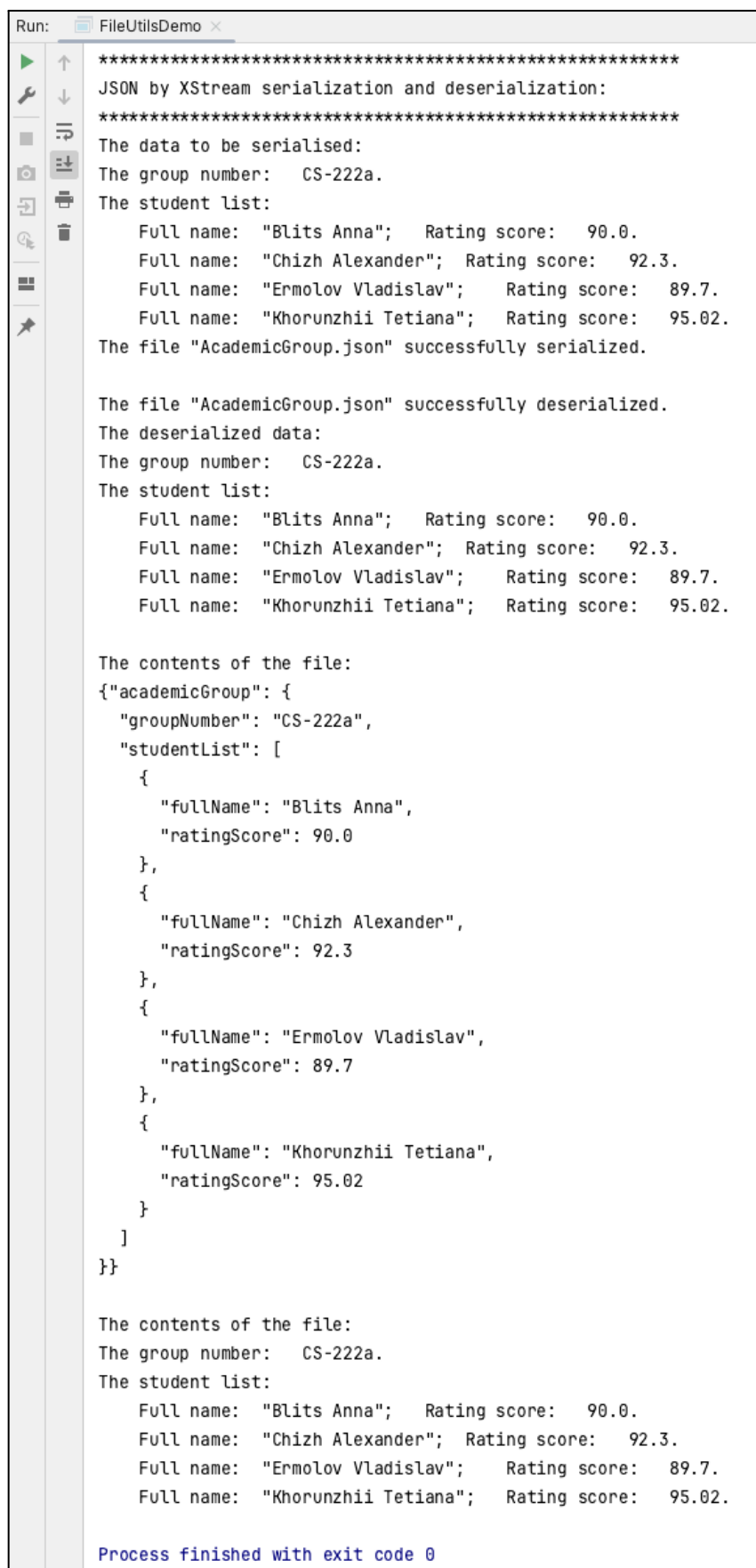
The file "AcademicGroup.xml" successfully deserialized.
The deserialized data:
The group number:  CS-222a.
The student list:
    Full name:  "Blits Anna";    Rating score:  90.0.
    Full name:  "Chizh Alexander"; Rating score:  92.3.
    Full name:  "Ermolov Vladislav"; Rating score:  89.7.
    Full name:  "Khorunzhii Tetiana"; Rating score:  95.02.

The contents of the file:
<academicGroup>
  <groupNumber>CS-222a</groupNumber>
  <studentList>
    <student>
      <fullName>Blits Anna</fullName>
      <ratingScore>90.0</ratingScore>
    </student>
    <student>
      <fullName>Chizh Alexander</fullName>
      <ratingScore>92.3</ratingScore>
    </student>
    <student>
      <fullName>Ermolov Vladislav</fullName>
      <ratingScore>89.7</ratingScore>
    </student>
    <student>
      <fullName>Khorunzhii Tetiana</fullName>
      <ratingScore>95.02</ratingScore>
    </student>
  </studentList>
</academicGroup>

The contents of the file:
The group number:  CS-222a.
The student list:
    Full name:  "Blits Anna";    Rating score:  90.0.
    Full name:  "Chizh Alexander"; Rating score:  92.3.
    Full name:  "Ermolov Vladislav"; Rating score:  89.7.
    Full name:  "Khorunzhii Tetiana"; Rating score:  95.02.

```

Рисунок 4.3.1 – Результаты № 1 роботи програмного коду



```

Run: FileUtilsDemo x
*****
JSON by XStream serialization and deserialization:
*****
The data to be serialised:
The group number:  CS-222a.
The student list:
    Full name: "Blits Anna";   Rating score:  90.0.
    Full name: "Chizh Alexander"; Rating score:  92.3.
    Full name: "Ermolov Vladislav"; Rating score:  89.7.
    Full name: "Khorunzhii Tetiana"; Rating score:  95.02.
The file "AcademicGroup.json" successfully serialized.

The file "AcademicGroup.json" successfully deserialized.
The deserialized data:
The group number:  CS-222a.
The student list:
    Full name: "Blits Anna";   Rating score:  90.0.
    Full name: "Chizh Alexander"; Rating score:  92.3.
    Full name: "Ermolov Vladislav"; Rating score:  89.7.
    Full name: "Khorunzhii Tetiana"; Rating score:  95.02.

The contents of the file:
{"academicGroup": {
  "groupNumber": "CS-222a",
  "studentList": [
    {
      "fullName": "Blits Anna",
      "ratingScore": 90.0
    },
    {
      "fullName": "Chizh Alexander",
      "ratingScore": 92.3
    },
    {
      "fullName": "Ermolov Vladislav",
      "ratingScore": 89.7
    },
    {
      "fullName": "Khorunzhii Tetiana",
      "ratingScore": 95.02
    }
  ]
}}

The contents of the file:
The group number:  CS-222a.
The student list:
    Full name: "Blits Anna";   Rating score:  90.0.
    Full name: "Chizh Alexander"; Rating score:  92.3.
    Full name: "Ermolov Vladislav"; Rating score:  89.7.
    Full name: "Khorunzhii Tetiana"; Rating score:  95.02.

Process finished with exit code 0

```

Рисунок 4.3.2 – Результати № 2 роботи програмного коду

```

1 <academicGroup>
2   <groupNumber>CS-222a</groupNumber>
3   <studentList>
4     <student>
5       <fullName>Blits Anna</fullName>
6       <ratingScore>90.0</ratingScore>
7     </student>
8     <student>
9       <fullName>Chizh Alexander</fullName>
10      <ratingScore>92.3</ratingScore>
11    </student>
12    <student>
13      <fullName>Ermolov Vladislav</fullName>
14      <ratingScore>89.7</ratingScore>
15    </student>
16    <student>
17      <fullName>Khorunzhii Tetiana</fullName>
18      <ratingScore>95.02</ratingScore>
19    </student>
20  </studentList>
21 </academicGroup>

```

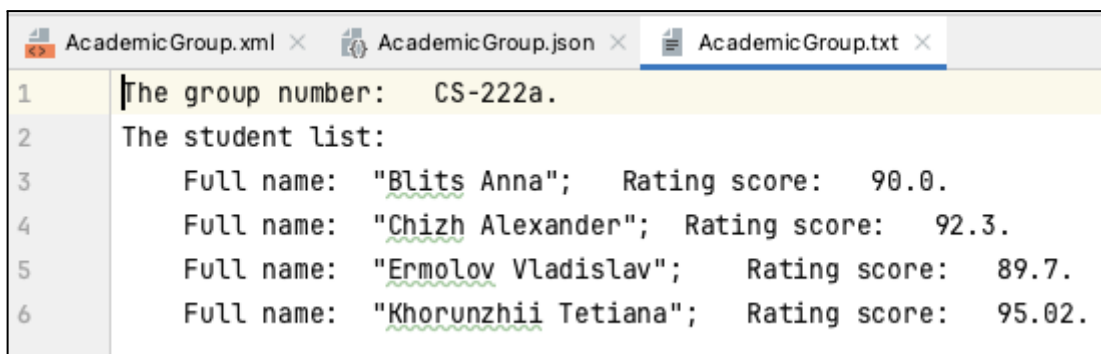
Рисунок 4.3.3 – Вміст файлу “AcademicGroup.xml”

```

1 { "academicGroup": {
2   "groupNumber": "CS-222a",
3   "studentList": [
4     {
5       "fullName": "Blits Anna",
6       "ratingScore": 90.0
7     },
8     {
9       "fullName": "Chizh Alexander",
10      "ratingScore": 92.3
11    },
12    {
13      "fullName": "Ermolov Vladislav",
14      "ratingScore": 89.7
15    },
16    {
17      "fullName": "Khorunzhii Tetiana",
18      "ratingScore": 95.02
19    }
20  ]
21 } }

```

Рисунок 4.3.3 – Вміст файлу “AcademicGroup.json”



```

1 The group number: CS-222a.
2 The student list:
3 Full name: "Blits Anna"; Rating score: 90.0.
4 Full name: "Chizh Alexander"; Rating score: 92.3.
5 Full name: "Ermolov Vladislav"; Rating score: 89.7.
6 Full name: "Khorunzhii Tetiana"; Rating score: 95.02.

```

Рисунок 4.3.3 – Вміст файлу “AcademicGroup.txt”

5. Завдання №5 до лабораторної роботи

5.1. Робота з бібліотекою org.json (додаткове завдання)

Виконати завдання 1.4 із застосуванням засобів для роботи з JSON-файлами бібліотеки org.json.

5.2. Програмний код реалізації завдання № 5

5.2.1. StudentWithJSON.java:

```

package part2.lab3.task5;

import org.json.JSONObject;
import part2.lab3.task4.Student;

public class StudentWithJSON extends Student {
    public StudentWithJSON() {}

    public StudentWithJSON(String fullName, Double ratingScore) throws
    IllegalArgumentException {
        super(fullName, ratingScore);
    }

    public JSONObject toJSON() {
        JSONObject json = new JSONObject();

        json.put("full_name", getFullName());
        json.put("rating_score", getRatingScore());

        return json;
    }

    public StudentWithJSON fromJSON(JSONObject json) {
        return new StudentWithJSON(
            json.getString("full_name"),
            json.getDouble("rating_score")
        );
    }
}

```

```

    }
}

```

5.2.2. AcademicGroupWithJSON.java:

```

package part2.lab3.task5;

import org.json.JSONArray;
import org.json.JSONObject;
import part2.lab3.task4.AcademicGroup;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class AcademicGroupWithJSON extends AcademicGroup {
    private List<StudentWithJSON> studentList = new ArrayList<>();
    public AcademicGroupWithJSON() {
    }

    public AcademicGroupWithJSON(String groupNumber) {
        super(groupNumber);
    }

    public AcademicGroupWithJSON(String groupNumber,
                                   List<StudentWithJSON> studentList)
throws IllegalArgumentException {
        super(groupNumber);

        if (studentList.isEmpty()) {
            throw new IllegalArgumentException("The list of students
cannot be empty.");
        }

        studentList.forEach(this::addStudent);
    }

    public List<StudentWithJSON> getStudentList() {
        return this.studentList;
    }

    public void addStudent(StudentWithJSON student) {
        studentList.add(student);
    }
}

```

```

@Override
public String toString() {
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append("The group
number:\t").append(getGroupNumber()).append(".\n")
        .append("The student list:\n")
        .append(getStudentList().stream()
            .map(StudentWithJSON::toString)
            .collect(Collectors.joining("\n")));

    return stringBuilder.toString();
}

public JSONObject toJSON() {
    JSONObject json = new JSONObject();
    JSONArray jsonArray = new JSONArray();

    getStudentList().forEach(student →
jsonArray.put(student.toJSON()));
    json.put("group_number", getGroupNumber());
    json.put("student_list", jsonArray);

    return json;
}

public AcademicGroupWithJSON fromJSON(JSONObject json) {
    AcademicGroupWithJSON academicGroup = new
AcademicGroupWithJSON(json.getString("group_number"));
    JSONArray jsonArray = json.getJSONArray("student_list");

    for (int i = 0; i < jsonArray.length(); i++) {
        academicGroup.addStudent(
            new StudentWithJSON().fromJSON(
                jsonArray.getJSONObject(i)
            );
    }

    return academicGroup;
}

public static AcademicGroupWithJSON createAcademicGroupWithJSON() {
    return new AcademicGroupWithJSON(
        "CS-222a",
        Arrays.asList(
            new StudentWithJSON("Blits Anna", 90.0),
            new StudentWithJSON("Chizh Alexander", 92.30),
            new StudentWithJSON("Ermolov Vladislav", 89.7),
            new StudentWithJSON("Khorunzhii Tetiana", 95.02)
        )
    );
}

```

```

    );
}
}

```

5.2.3. FileUtilsWithJSON.java:

```

package part2.lab3.task5;

import org.json.JSONObject;
import part2.lab3.task4.FileUtils;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class FileUtilsWithJSON extends FileUtils {
    public static void serializeToJSON(String inputFilePath,
        AcademicGroupWithJSON academicGroup) {
        System.out.println("The data to be serialised:\n" +
            academicGroup);

        try (FileWriter writer = new FileWriter(inputFilePath)) {
            writer.write(academicGroup.toJSON().toString(1));
            System.out.println("The file \"" + new
                File(inputFilePath).getName() + "\" successfully serialized.\n");
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }

    public static void deserializeFromJSON(String outputFilePath, String
        inputFilePath) {
        try {
            String content = new
                String(Files.readAllBytes(Paths.get(inputFilePath)));
            JSONObject json = new JSONObject(content);
            AcademicGroupWithJSON academicGroup = new
                AcademicGroupWithJSON().fromJSON(json);

            if (academicGroup != null) {
                writeToFile(outputFilePath, academicGroup.toString());
                System.out.println("The file \"" + new
                    File(inputFilePath).getName() + "\" successfully deserialized.");
                System.out.println("The deserialized data:\n" +
                    academicGroup);
            }
        }
    }
}

```

```

    }
} catch (IOException e) {
    System.err.println(e.getMessage());
}
}
}

```

5.2.4. FileUtilsWithJSONDemo.java:

```

package part2.lab3.task5;

import part2.lab3.task4.FileUtilsDemo;

import static part2.lab3.task5.FileUtilsWithJSON.*;

public class FileUtilsWithJSONDemo {
    private static final String INPUT_FILE_PATH =
"src/main/resources/part2/lab3/task5/AcademicGroup.json";
    private static final String OUTPUT_FILE_PATH =
"src/main/resources/part2/lab3/task5/AcademicGroup.txt";

    public static void testJSONSerialization() {

System.out.println("*****\n"
        + "JSON serialization and deserialization:\n"
        + "*****");

        serializeToJSON(INPUT_FILE_PATH,
AcademicGroupWithJSON.createAcademicGroupWithJSON());
        deserializeFromJSON(OUTPUT_FILE_PATH, INPUT_FILE_PATH);
        FileUtilsDemo.printOutputFiles(INPUT_FILE_PATH,
OUTPUT_FILE_PATH);
    }

    public static void main(String[] args) {
        testJSONSerialization();
    }
}

```

5.3. Екранні форми за результатами роботи програмного коду завдання № 5

```

Run: FileUtilsWithJSONDemo x
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...
*****
JSON serialization and deserialization:
*****
The data to be serialised:
The group number:   CS-222a.
The student list:
    Full name: "Blits Anna";   Rating score:   90.0.
    Full name: "Chizh Alexander"; Rating score:  92.3.
    Full name: "Ermolov Vladislav"; Rating score: 89.7.
    Full name: "Khorunzhii Tetiana"; Rating score: 95.02.
The file "AcademicGroup.json" successfully serialized.

The file "AcademicGroup.json" successfully deserialized.
The deserialized data:
The group number:   CS-222a.
The student list:
    Full name: "Blits Anna";   Rating score:   90.0.
    Full name: "Chizh Alexander"; Rating score:  92.3.
    Full name: "Ermolov Vladislav"; Rating score: 89.7.
    Full name: "Khorunzhii Tetiana"; Rating score: 95.02.

The contents of the file:
{
  "group_number": "CS-222a",
  "student_list": [
    {
      "full_name": "Blits Anna",
      "rating_score": 90
    },
    {
      "full_name": "Chizh Alexander",
      "rating_score": 92.3
    },
    {
      "full_name": "Ermolov Vladislav",
      "rating_score": 89.7
    },
    {
      "full_name": "Khorunzhii Tetiana",
      "rating_score": 95.02
    }
  ]
}

The contents of the file:
The group number:   CS-222a.
The student list:
    Full name: "Blits Anna";   Rating score:   90.0.
    Full name: "Chizh Alexander"; Rating score:  92.3.
    Full name: "Ermolov Vladislav"; Rating score: 89.7.
    Full name: "Khorunzhii Tetiana"; Rating score: 95.02.

Process finished with exit code 0

```

Рисунок 5.3.1 – Результати роботи програмного коду

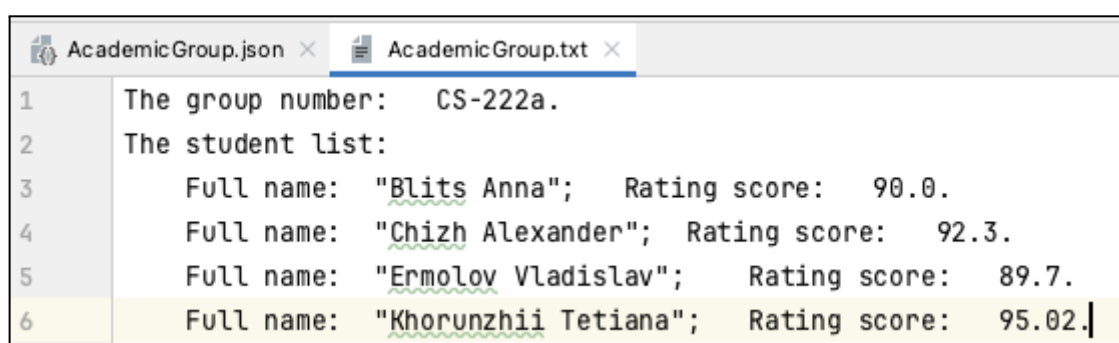


```

1 {
2   "group_number": "CS-222a",
3   "student_list": [
4     {
5       "full_name": "Blits Anna",
6       "rating_score": 90
7     },
8     {
9       "full_name": "Chizh Alexander",
10      "rating_score": 92.3
11    },
12    {
13      "full_name": "Ermolov Vladislav",
14      "rating_score": 89.7
15    },
16    {
17      "full_name": "Khorunzhii Tetiana",
18      "rating_score": 95.02
19    }
20  ]
21 }

```

Рисунок 5.3.2 – Вміст файлу “AcademicGroup.json”



```

1 The group number:  CS-222a.
2 The student list:
3   Full name:  "Blits Anna";   Rating score:   90.0.
4   Full name:  "Chizh Alexander"; Rating score:  92.3.
5   Full name:  "Ermolov Vladislav"; Rating score: 89.7.
6   Full name:  "Khorunzhii Tetiana"; Rating score: 95.02.

```

Рисунок 5.3.3 – Вміст файлу “AcademicGroup.txt”

6. Завдання №6 до лабораторної роботи

6.1. Використання технологій SAX і DOM (додаткове завдання)

Підготувати XML-документ з даними про студентів академічної групи. За допомогою технології SAX здійснити читання даних з XML-документа і виведення даних на консоль. За допомогою технології DOM здійснити

читання даних з того ж XML-документа, модифікацію даних і запис їх в новий документ.

6.2. Програмний код реалізації завдання № 6

6.2.1. FileUtils.java:

```
package part2.lab3.task6;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;
import java.io.IOException;

public class FileUtils {
    public static class DOMFileUtils {
        public Document parseXMLFile(String fileName) throws
ParserConfigurationException, SAXException, IOException {
            File inputFile = new File(fileName);

            DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder docBuilder = factory.newDocumentBuilder();

            return docBuilder.parse(inputFile);
        }

        public void printAcademicGroupInfo(Document document) {
            document.getDocumentElement().normalize();
            System.out.println("Group number:\t"
+
document.getElementsByTagName("groupNumber").item(0).getTextContent()
+ "\nStudent list:");

            NodeList nList = document.getElementsByTagName("student");

            for (int i = 0; i < nList.getLength(); i++) {
                Node nNode = nList.item(i);
```

```

        System.out.print("\tStudent:\t");

        if (nNode.getNodeType() == Node.ELEMENT_NODE) {
            Element eElement = (Element) nNode;

            System.out.println(
                "Full name: \t\" +
eElement.getElementsByTagName("fullName").item(0).getTextContent() +
                "\";\tRating score:\t" +
eElement.getElementsByTagName("ratingScore").item(0).getTextContent() + "."
            );
        }
    }

    public void updateRatingScore(Document document) {
        NodeList nodeList = document.getElementsByTagName("ratingScore");

        for (int i = 0; i < nodeList.getLength(); i++) {
            Node node = nodeList.item(i);

            node.setTextContent(String.valueOf(Double.parseDouble(node.getTextContent()) +
3.5));
        }
        printAcademicGroupInfo(document);
    }

    public void writeModifiedToXML(Document document, String fileName)
throws TransformerException {
        TransformerFactory factory = TransformerFactory.newInstance();
        Transformer transformer = factory.newTransformer();

        DOMSource source = new DOMSource(document);
        StreamResult result = new StreamResult(new File(fileName));
        transformer.transform(source, result);
    }
}

public static class SAXFileUtils extends DefaultHandler {
    private boolean inGroupNumber = false;
    private boolean inStudentList = false;
    private boolean inStudent = false;
    private boolean inFullName = false;
    private boolean inRatingScore = false;

    @Override
    public void startDocument() {
        System.out.println("\nParsed info from document:");
    }

    @Override
    public void startElement(String uri, String localName, String qName,

```

```

Attributes attributes) throws SAXException {
    if (qName.equalsIgnoreCase("groupNumber")) {
        inGroupNumber = true;
        System.out.print("Group number:\t");
    } else if (qName.equalsIgnoreCase("studentList")) {
        inStudentList = true;
        System.out.print("\nStudent list:");
    } else if (qName.equalsIgnoreCase("student")) {
        inStudent = true;
        System.out.print("\n\tStudent:");
    } else if (qName.equalsIgnoreCase("fullName")) {
        inFullName = true;
        System.out.print("\tFull name:\t");
    } else if (qName.equalsIgnoreCase("ratingScore")) {
        inRatingScore = true;
        System.out.print("\tRating score:\t");
    }
}

@Override
public void characters(char[] ch, int start, int length) throws
SAXException {
    String value = new String(ch)
        .substring(start, start + length)
        .trim();

    if (!value.isEmpty()
        && (inGroupNumber
            || inFullName
            || inRatingScore)) {
        System.out.print(value);
    }
}
}
}

```

6.2.2. FileUtilsDemo.java:

```

package part2.lab3.task6;

import org.w3c.dom.Document;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.transform.TransformerException;
import java.io.IOException;

import static part2.lab3.task4.FileUtils.*;

```

```

import static part2.lab3.task6.FileUtils.*;

public class FileUtilsDemo {
    private static final String INPUT_FILE_PATH =
"src/main/resources/part2/lab3/task6/AcademicGroup.xml";
    private static final String OUTPUT_FILE_PATH =
"src/main/resources/part2/lab3/task6/ModifiedAcademicGroup.xml";

    public static void testFileUtilsByDOM() {

System.out.println("*****\n"
n"
        + "Academic group XML file processor by DOM:\n"
        + "*****");

    try {
        printLines(readFromFile(INPUT_FILE_PATH));

        DOMFileUtils parser = new DOMFileUtils();
        Document document = parser.parseXMLFile(INPUT_FILE_PATH);

        System.out.println("\nParsed info from document:");
        parser.printAcademicGroupInfo(document);

        System.out.println("\nModified info from document:");
        parser.updateRatingScore(document);

        parser.writeModifiedToXML(document, OUTPUT_FILE_PATH);
        System.out.println("\nModified XML file saved.\n");

        printLines(readFromFile(OUTPUT_FILE_PATH));
    } catch (ParserConfigurationException | SAXException | IOException |
TransformerException e) {
        System.err.println(e.getMessage());
    }
}

    public static void testFileUtilsBySAX() {

System.out.println("*****\n"
n"
        + "Academic group XML file processor by SAX:\n"
        + "*****");

    try {
        printLines(readFromFile(INPUT_FILE_PATH));

        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser saxParser = factory.newSAXParser();

        if (saxParser != null) {
            InputSource input = new InputSource(INPUT_FILE_PATH);
            saxParser.parse(input, new SAXFileUtils());
        }
    }
}

```

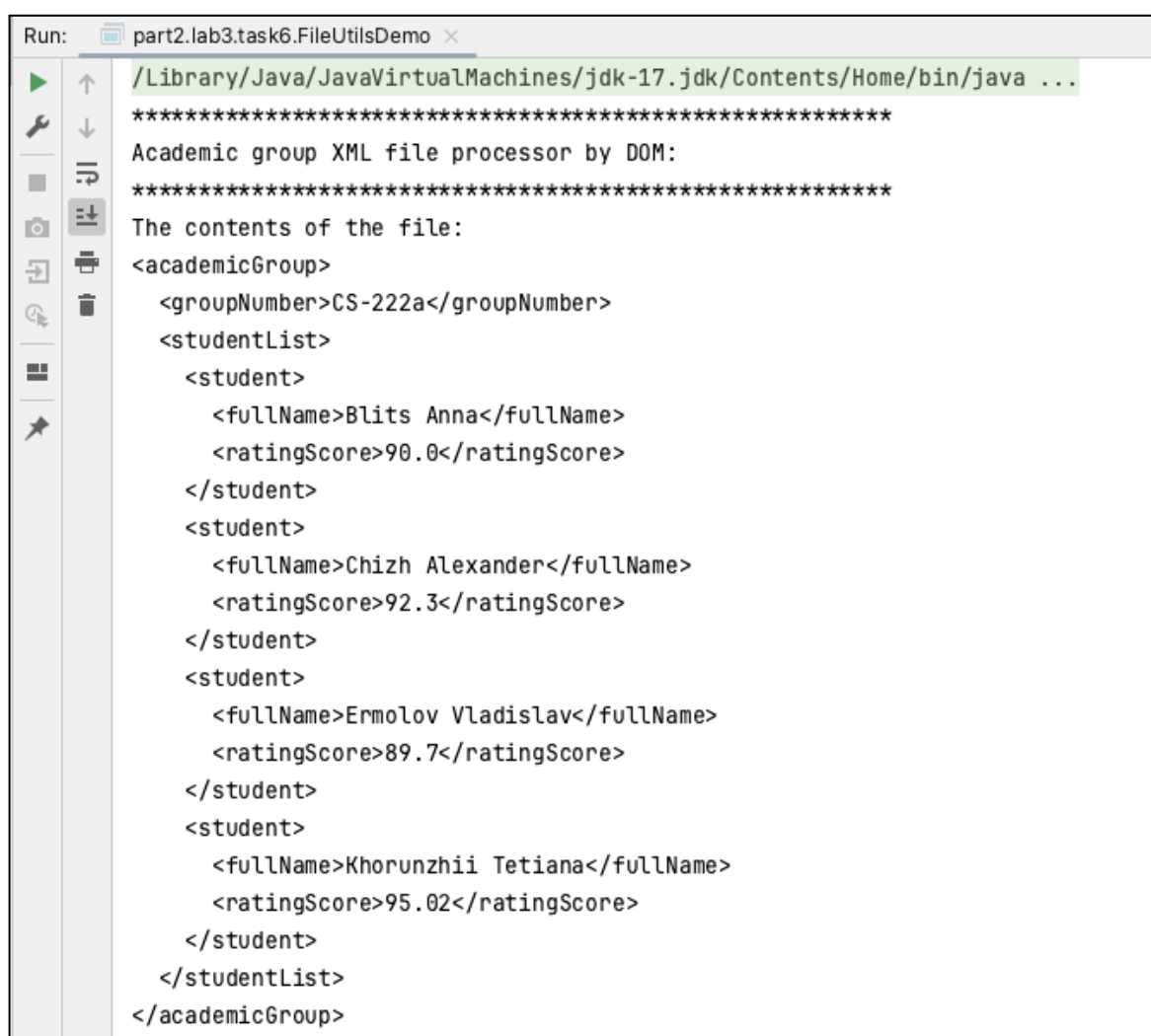
```

    } catch (ParserConfigurationException | SAXException | IOException e) {
        System.err.println(e.getMessage());
    }
}

public static void main(String[] args) {
    testFileUtilsByDOM();
    System.out.println("\n");
    testFileUtilsBySAX();
}
}

```

6.3. Екранні форми за результатами роботи програмного коду завдання № 6



```

Run: part2.lab3.task6.FileUtilsDemo x
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...
*****
Academic group XML file processor by DOM:
*****
The contents of the file:
<academicGroup>
  <groupNumber>CS-222a</groupNumber>
  <studentList>
    <student>
      <fullName>Blits Anna</fullName>
      <ratingScore>90.0</ratingScore>
    </student>
    <student>
      <fullName>Chizh Alexander</fullName>
      <ratingScore>92.3</ratingScore>
    </student>
    <student>
      <fullName>Ermolov Vladislav</fullName>
      <ratingScore>89.7</ratingScore>
    </student>
    <student>
      <fullName>Khorunzhii Tetiana</fullName>
      <ratingScore>95.02</ratingScore>
    </student>
  </studentList>
</academicGroup>

```

Рисунок 6.3.1 – Результати № 1 роботи програмного коду

	<p>Parsed info from document:</p> <p>Group number: CS-222a</p> <p>Student list:</p> <p>Student: Full name: "Blits Anna"; Rating score: 90.0.</p> <p>Student: Full name: "Chizh Alexander"; Rating score: 92.3.</p> <p>Student: Full name: "Ermolov Vladislav"; Rating score: 89.7.</p> <p>Student: Full name: "Khorunzhii Tetiana"; Rating score: 95.02.</p> <p>Modified info from document:</p> <p>Group number: CS-222a</p> <p>Student list:</p> <p>Student: Full name: "Blits Anna"; Rating score: 93.5.</p> <p>Student: Full name: "Chizh Alexander"; Rating score: 95.8.</p> <p>Student: Full name: "Ermolov Vladislav"; Rating score: 93.2.</p> <p>Student: Full name: "Khorunzhii Tetiana"; Rating score: 98.52.</p> <p>Modified XML file saved.</p>
--	---

Рисунок 6.3.2 – Результаты № 2 работы программного коду

	<p>The contents of the file:</p> <pre><?xml version="1.0" encoding="UTF-8" standalone="no"?><academicGroup> <groupName>CS-222a</groupName> <studentList> <student> <fullName>Blits Anna</fullName> <ratingScore>93.5</ratingScore> </student> <student> <fullName>Chizh Alexander</fullName> <ratingScore>95.8</ratingScore> </student> <student> <fullName>Ermolov Vladislav</fullName> <ratingScore>93.2</ratingScore> </student> <student> <fullName>Khorunzhii Tetiana</fullName> <ratingScore>98.52</ratingScore> </student> </studentList> </academicGroup></pre>
--	---

Рисунок 6.3.3 – Результаты № 3 работы программного коду

```

*****
Academic group XML file processor by SAX:
*****
The contents of the file:
<academicGroup>
  <groupNumber>CS-222a</groupNumber>
  <studentList>
    <student>
      <fullName>Blits Anna</fullName>
      <ratingScore>90.0</ratingScore>
    </student>
    <student>
      <fullName>Chizh Alexander</fullName>
      <ratingScore>92.3</ratingScore>
    </student>
    <student>
      <fullName>Ermolov Vladislav</fullName>
      <ratingScore>89.7</ratingScore>
    </student>
    <student>
      <fullName>Khorunzhii Tetiana</fullName>
      <ratingScore>95.02</ratingScore>
    </student>
  </studentList>
</academicGroup>

Parsed info from document:
Group number:   CS-222a
Student list:
  Student:      Full name: "Blits Anna";   Rating score:   90.0
  Student:      Full name: "Chizh Alexander"; Rating score:  92.3
  Student:      Full name: "Ermolov Vladislav"; Rating score: 89.7
  Student:      Full name: "Khorunzhii Tetiana"; Rating score: 95.02
Process finished with exit code 0

```

Рисунок 6.3.4 – Результаты № 4 работы программного коду


```

1 <academicGroup>
2   <groupNumber>CS-222a</groupNumber>
3   <studentList>
4     <student>
5       <fullName>Blits Anna</fullName>
6       <ratingScore>90.0</ratingScore>
7     </student>
8     <student>
9       <fullName>Chizh Alexander</fullName>
10      <ratingScore>92.3</ratingScore>
11    </student>
12    <student>
13      <fullName>Ermolov Vladislav</fullName>
14      <ratingScore>89.7</ratingScore>
15    </student>
16    <student>
17      <fullName>Khorunzhii Tetiana</fullName>
18      <ratingScore>95.02</ratingScore>
19    </student>
20  </studentList>
21 </academicGroup>

```

Рисунок 6.3.5 – Вміст файлу “AcademicGroup.xml”

```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"?><academicGroup>
2   <groupNumber>CS-222a</groupNumber>
3   <studentList>
4     <student>
5       <fullName>Blits Anna</fullName>
6       <ratingScore>93.5</ratingScore>
7     </student>
8     <student>
9       <fullName>Chizh Alexander</fullName>
10      <ratingScore>95.8</ratingScore>
11    </student>
12    <student>
13      <fullName>Ermolov Vladislav</fullName>
14      <ratingScore>93.2</ratingScore>
15    </student>
16    <student>
17      <fullName>Khorunzhii Tetiana</fullName>
18      <ratingScore>98.52</ratingScore>
19    </student>
20  </studentList>
21 </academicGroup>

```

Рисунок 6.3.6 – Вміст файлу “ModifiedAcademicGroup.xml”

7. Вправа для контролю до лабораторної роботи

7.1. Умови завдань

7.1.1. Прочитати з текстового файлу дійсні значення (до кінця файлу), знайти їх суму та вивести в інший текстовий файл. Застосувати засоби Stream API.

7.1.2. Прочитати з текстового файлу цілі значення, замінити від'ємні значення модулями, додатні - нулями та вивести отримані значення в інший текстовий файл. Застосувати засоби Stream API.

7.1.3. Прочитати з текстового файлу цілі значення, розділити парні елементи на 2, непарні – збільшити у 2 рази та вивести отримані значення в інший текстовий файл. Застосувати засоби Stream API.

7.2. Програмний код реалізації вправи для контролю

7.2.1. NumberProcessor.java:

```
package part2.lab3.task_control;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.stream.Stream;

public class NumberProcessor {
    /**
     * Control Task #1. Read real values from a text file (to the end of
     * the file),
     * find their sum and output to another text file. Use the Stream
     * API tools.
     */
    public static void findAndWriteToFileRealNumbersSum(String
outputFilePath, String inputFilePath) {
        try (Stream<String> stream =
Files.lines(Paths.get(inputFilePath));
            PrintWriter writer = new PrintWriter(new
FileWriter(outputFilePath))) {
            Double realNumbersSum =
stream.mapToDouble(Double::parseDouble).sum();
            writer.println("The real numbers sum:\t" + realNumbersSum);
            System.out.println("Successfully read real numbers from the
text file, " +
```

```

        "\n\ncalculated its sum " +
        "\n\ncand wrote them to another text file.\n");
    } catch (IOException e) {
        System.err.println(e.getMessage());
    }
}

/**
 * Control Task #2. Read integer values from a text file,
 * replace negative values with modules, positive values with zeros,
 * and output the resulting values to another text file. Use the
 * Stream API tools.
 */
public static void findAndWriteToFileAbsoluteIntegers(String
outputFilePath, String inputFilePath) {
    try (Stream<String> stream =
Files.lines(Paths.get(inputFilePath));
        PrintWriter writer = new PrintWriter(new
FileWriter(outputFilePath))) {
        stream.mapToInt(Integer::parseInt)
            .map(number → number < 0 ? Math.abs(number) : 0)
            .forEach(writer::println);
        System.out.println("Successfully read integer numbers from
the text file, " +
            "\n\nreplaced negative values with modules, positive
values with zeros" +
            "\n\ncand wrote them to another text file.\n");
    } catch (IOException e) {
        System.err.println(e.getMessage());
    }
}

/** Control Task #3. Read integer values from a text file,
 * divide even elements by 2, increase odd elements by 2
 * and output the resulting values to another text file. Use the
 * Stream API tools.*/
public static void findAndWriteToFileOddAndEvenIntegers(String
outputFilePath, String inputFilePath) {
    try (Stream<String> stream =
Files.lines(Paths.get(inputFilePath));
        PrintWriter writer = new PrintWriter(new
FileWriter(outputFilePath))) {
        stream.mapToInt(Integer::parseInt)
            .map(number → number % 2 == 0 ? number / 2 : number
* 2)
            .forEach(writer::println);
        System.out.println("Successfully read integer numbers from
the text file, " +
            "\n\ndivided even values by 2, increase odd values by

```

```

2" +
        "\nand wrote them to another text file.");
    } catch (IOException e) {
        System.err.println(e.getMessage());
    }
}
}

```

7.2.2. NumberProcessorDemo.java:

```

package part2.lab3.task_control;

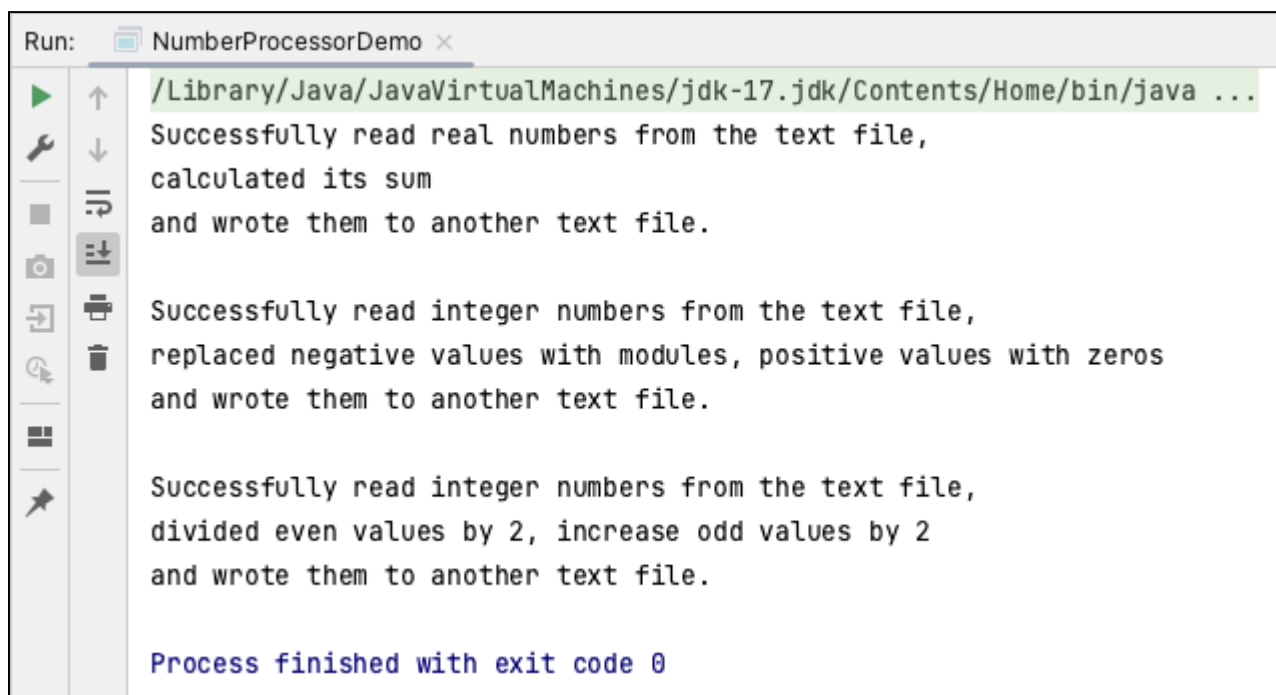
import static part2.lab3.task_control.NumberProcessor.*;

public class NumberProcessorDemo {
    private static final String INPUT_REAL_NUMENRS =
"src/main/resources/part2/lab3/task_control/inputRealNumbers.txt";
    private static final String INPUT_INTEGERS =
"src/main/resources/part2/lab3/task_control/inputIntegers.txt";
    private static final String OUTPUT_REAL_NUMBERS =
"src/main/resources/part2/lab3/task_control/outputRealNumbers.txt";
    private static final String OUTPUT_ABS_AND_ZERO_INTEGERS =
"src/main/resources/part2/lab3/task_control/outputAbsoluteAndZeroIntegers.txt";
    private static final String OUTPUT_ODD_AND_EVEN_INTEGERS =
"src/main/resources/part2/lab3/task_control/outputOddAndEvenIntegers.txt";

    public static void main(String[] args) {
        findAndWriteToFileRealNumbersSum(OUTPUT_REAL_NUMBERS,
INPUT_REAL_NUMENRS);
        findAndWriteToFileAbsoluteIntegers(OUTPUT_ABS_AND_ZERO_INTEGERS,
INPUT_INTEGERS);
        findAndWriteToFileOddAndEvenIntegers(OUTPUT_ODD_AND_EVEN_INTEGERS,
INPUT_INTEGERS);
    }
}

```

7.3. Екранні форми за результатами роботи програмного коду вправи для контролю



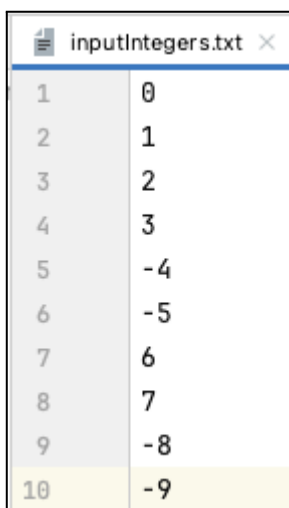
```
Run: NumberProcessorDemo x
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...
Successfully read real numbers from the text file,
calculated its sum
and wrote them to another text file.

Successfully read integer numbers from the text file,
replaced negative values with modules, positive values with zeros
and wrote them to another text file.

Successfully read integer numbers from the text file,
divided even values by 2, increase odd values by 2
and wrote them to another text file.

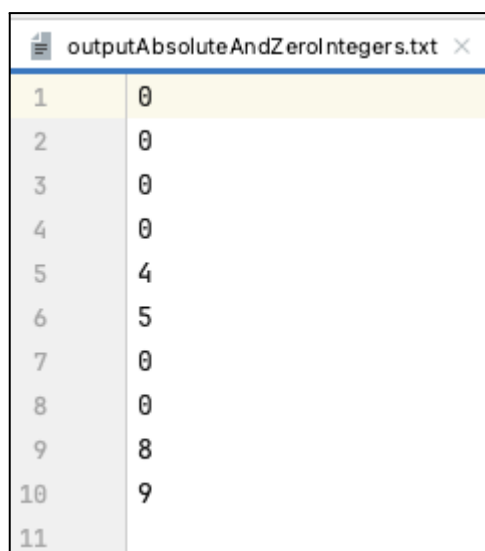
Process finished with exit code 0
```

Рисунок 7.3.1 – Результати роботи програмного коду



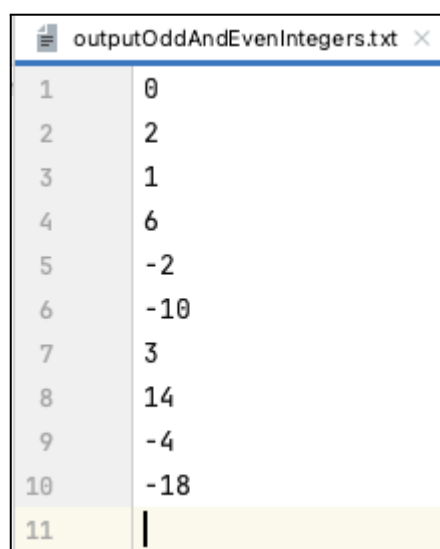
1	0
2	1
3	2
4	3
5	-4
6	-5
7	6
8	7
9	-8
10	-9

Рисунок 7.3.2 – Вміст файлу “inputIntegers.txt”



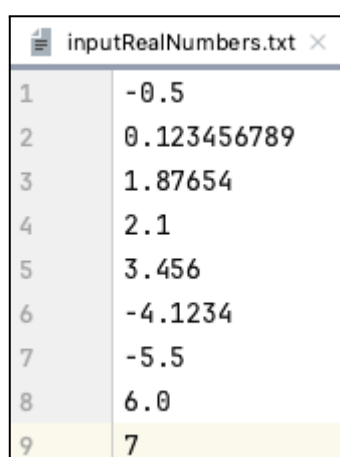
1	0
2	0
3	0
4	0
5	4
6	5
7	0
8	0
9	8
10	9
11	

Рисунок 7.3.3 – Вміст файлу “outputAbsoluteAndZeroIntegers.txt”



1	0
2	2
3	1
4	6
5	-2
6	-10
7	3
8	14
9	-4
10	-18
11	

Рисунок 7.3.4 – Вміст файлу “outputOddAndEvenIntegers.txt”



1	-0.5
2	0.123456789
3	1.87654
4	2.1
5	3.456
6	-4.1234
7	-5.5
8	6.0
9	7

Рисунок 7.3.5 – Вміст файлу “inputRealNumbers.txt”

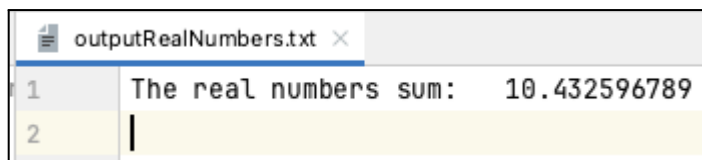


Рисунок 7.3.6 – Вміст файлу “outputRealNumbers.txt”

8. Висновки до лабораторної роботи

Виконання даної лабораторної роботи надало можливість збагатити знання і навички у роботі з файлами та різними інструментами Java. Перш за все, завдяки цій роботі було отримано розширені знання про роботу з файлами в Java, що є важливим аспектом в програмуванні. Також було опановано різні способи читання та запису даних у текстові файли за допомогою Stream API, що дозволило набути кращого розуміння принципів їх роботи та ефективно використовувати ці інструменти у реалізовуваних програмах.

Другим важливим аспектом було використання серіалізації у форматах XML та JSON засобами бібліотеки XStream. Це дозволило перетворювати об'єкти Java у рядки XML та JSON для зберігання або передачі даних, а також зчитувати ці дані для подальшого використання у наших програмах. Це є важливою навичкою для взаємодії з іншими системами та сервісами.

Нарешті, було отримано досвід у написанні тестів за допомогою JUnit, що є стандартним інструментом для тестування Java-програм. Це дозволило переконатися у коректності реалізованого коду та його відповідності вимогам, що забезпечило більшу надійність та стабільність програм.