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

КАФЕДРА «Програмна інженерія та інтелектуальні технології управління»

ЗВІТ

з лабораторної роботи № 2

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

“Робота датами та текстом. Локалізація”

1. Завдання №1 до лабораторної роботи

1.1. Індивідуальне завдання

Спроєктувати та реалізувати класи для представлення сутностей третьої лабораторної роботи курсу "Основи програмування Java". Рішення повинне базуватися на раніше створеній ієрархії класів.

Слід створити похідний клас, який представляє основну сутність. Як базовий використати клас, створений у попередній лабораторній роботі. Клас повинен бути доповненим можливостями підтримки різних локалізацій, зокрема, української та американської. Необхідно передбачити переклад тексту, виведення чисел, а також дат і часу з урахуванням різних локалізацій. Додати (модифікувати) пошук слів у коментарях (або іншому тексті) за допомогою регулярних виразів. Здійснити сортування сутностей за алфавітом з використанням класу Collator.

Створити похідний клас від класу, який представляє другу сутність, в якому додати поле – час і дата, коли відбувається певна подія. Для представлення часу й дати використовувати класи пакету java.time (з урахуванням поясного часу). Підрахувати проміжки часу між подіями та знайти й вивести найменший з проміжків. Якщо клас, який представляв другу сутність індивідуального завдання, не містив поля, типу String, слід додати поле – коментар до події. Для нового (або такого, що існує) текстового поля передбачити можливість виведення українською або англійською мовою, залежно від локалізації.

Програма повинна демонструвати:

- відтворення реалізації завдань лабораторних робіт № 3 і № 4 курсу "Основи програмування Java";
- форматування числових даних різними варіантами, а також з урахуванням локалізації;

- виведення даних про дати й час подій з урахуванням локалізації;
- виведення тексту українською та англійською мовою;
- можливості сортування за алфавітом з використанням класу Collator;
- підрахування та виведення проміжків часу між подіями, пов'язаними з другою сутністю завдання; знаходження й виведення найменшого з проміжків;
- варіанти складного пошуку в тексті (із застосуванням регулярних виразів), зокрема, пошук фрагмента тексту на початку (наприкінці) слова.

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

№ №	Перша сутність		Другиа 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;
```

```java
/**
* 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;
   }
```

```java
/**
 * 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. HourWithDates.java:

```java
package part2.lab2.task1;

import part1.lab4.task1.Hour;

import java.text.NumberFormat;
import java.time.Duration;
import java.time.Month;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.util.Arrays;
import java.util.Locale;
import java.util.ResourceBundle;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.stream.Stream;

/**
* The {@code HourWithDates} class represents an hour with additional
date information.
* It extends the {@link Hour} class and includes methods for managing
{@link ZonedDateTime} operating hour field,
* calculating {@link Duration} intervals and displaying results.
*/
public class HourWithDates extends Hour {
   /** Represents the operating hour date and time for this instance.
*/
   private ZonedDateTime operatingHour;

   /** Constructs a new default {@link HourWithDates} object. */
   public HourWithDates() {}

   /**
    * Constructs a new {@link HourWithDates} object with the specified
parameters.
    * @param ridership The number of ridership.
    * @param comment The comment associated with this hour.
    * @param operatingHour The operating hour.
    */
   public HourWithDates(int ridership, String comment, ZonedDateTime
operatingHour) {
       super(ridership, comment);
       this.operatingHour = operatingHour;
   }
```

```
    /**
     * Gets the {@link HourWithDates#operatingHour}.
     * @return The operating hour date and time.
     */
    public ZonedDateTime getOperatingHour() {
        return operatingHour;
    }

    /**
     * Overrides the {@link Hour#getComment()} method to retrieve
localized comment.
     * @return Localized comment.
     */
    @Override
    public String getComment() {
        ResourceBundle bundle = ResourceBundle.getBundle("hours");
        return bundle.getString(super.getComment());
    }

    /**
     * Sets the {@link HourWithDates#operatingHour}.
     * @param operatingHour The operating hour to set.
     */
    public void setOperatingHour(ZonedDateTime operatingHour) {
        this.operatingHour = operatingHour;
    }

    /**
     * Provides the string representing the {@link HourWithDates}
object.
     * @return the string representing the {@link HourWithDates} object.
     */
    @Override
    public String toString() {
        ResourceBundle bundle = ResourceBundle.getBundle("hours");
        DateTimeFormatter dateTimeFormatter =
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG).withLocale(Loca
le.getDefault());
        NumberFormat numberFormat =
NumberFormat.getInstance(Locale.getDefault());

        return String.format(
                Locale.getDefault(),
                "\t" + bundle.getString("hour") + "\t{ "
                        + bundle.getString("operationHour") + ": " +
getOperatingHour().format(dateTimeFormatter) + ";\t"
                        + bundle.getString("ridership") + " = " +
numberFormat.format(getRidership()) + ";\t"
                        + bundle.getString("comment") + " = \'" +
```

```
getComment() + "\' }");
    }

    /**
     * Calculates the word count of the comment using regular
expressions.
     * @return The word count of the comment.
     */
    @Override
    public int calculateWordCountOfComment() {
        String comment = getComment();

        if (comment == null || comment.isEmpty()) {
            return 0;
        }

        String regex = "\\s+";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(comment);
        String[] wordArray = pattern.split(comment);

        return wordArray.length;
    }

    /**
     * Calculates the time interval in {@link Duration} between two
operating hours.
     * @param hour1 The first HourWithDates object.
     * @param hour2 The second HourWithDates object.
     * @return The duration between the two hours.
     */
    public Duration getInterval(HourWithDates hour1, HourWithDates
hour2) {
        return Duration.between(hour1.getOperatingHour(),
hour2.getOperatingHour()).abs();
    }

    /**
     * Finds the smallest interval in {@link Duration} between hours in
the given array.
     * @param hours An array of HourWithDates objects.
     * @return The smallest interval in {@link Duration}.
     */
    public Duration findSmallestIntervalBetweenHours(HourWithDates[]
hours) {
        return Stream.of(hours)
                .flatMap(hour -> Stream.of(hours)
                        .filter(other -> hour != other)
                        .map(other -> getInterval(hour, other))
```

```
                )
                .min(Duration::compareTo)
                .orElse(Duration.ZERO);
    }

    /**
     * Finds the largest interval in {@link Duration} between hours in
the given array.
     * @param hours An array of HourWithDates objects.
     * @return The largest interval in {@link Duration}.
     */
    public Duration findLargestIntervalBetweenHours(HourWithDates[]
hours) {
        return Stream.of(hours)
                .flatMap(hour -> Stream.of(hours)
                        .filter(other -> hour != other)
                        .map(other -> getInterval(hour, other))
                )
                .max(Duration::compareTo)
                .orElse(Duration.ZERO);
    }


    /**
     * Generates a new {@link HourWithDates} object with predefined
values.
     * @return The created {@link HourWithDates} object.
     */
    public HourWithDates createHour() {
        setOperatingHour(
                ZonedDateTime.of(
                        2024,
                        Month.MAY.getValue(),
                        3,
                        17, 0, 0, 0,
                        ZoneId.of("Europe/Kiev")
                )
        );
        setRidership(1234);
        setComment("veryHighRidership");

        return this;
    }

    /**
     * Generates an array of {@link HourWithDates} objects with
predefined values.
     * @return An array of {@link HourWithDates} objects.
     */
```

```
    public HourWithDates[] createHours() {
        return new HourWithDates[] {
            new HourWithDates(1100, "veryHighRidership",
                    ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 8,
0, 0, 0, ZoneId.of("Europe/Kiev"))),
            new HourWithDates(110, "lowRidership",
                    ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 12,
0, 0, 0, ZoneId.of("Europe/Kiev"))),
            new HourWithDates(650, "highRidership",
                    ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 14,
0, 0, 0, ZoneId.of("Europe/Kiev"))),
            new HourWithDates(532, "highRidership",
                    ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 15,
0, 0, 0, ZoneId.of("Europe/Kiev"))),
            new HourWithDates(60, "veryLowRidership",
                    ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 22,
0, 0, 0, ZoneId.of("Europe/Kiev"))),
            new HourWithDates(188, "lowRidership",
                    ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 10,
0, 0, 0, ZoneId.of("Europe/Kiev"))),
            new HourWithDates(200, "mediumRidership",
                    ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 5,
0, 0, 0, ZoneId.of("Europe/Kiev"))),
            new HourWithDates(200, "mediumRidership",
                    ZonedDateTime.of(2024, Month.MAY.getValue(), 18, 17,
8, 32, 0, ZoneId.of("Europe/Kiev"))),
        };
    }

    /**
     * Prints the interval in {@link Duration} in a human-readable
format.
     * @param interval The duration to print.
     */
    public void printInterval(Duration interval) {
        ResourceBundle bundle = ResourceBundle.getBundle("hours");
        NumberFormat numberFormat =
NumberFormat.getInstance(Locale.getDefault());
        System.out.println(
                numberFormat.format(interval.toDays()) + " " +
bundle.getString("timeDays") + "    "
                + numberFormat.format(interval.toHours() % 24) + " " +
bundle.getString("timeHours") + "    "
                + numberFormat.format(interval.toMinutes() % 60) + " " +
bundle.getString("timeMinutes") + "    "
                + numberFormat.format(interval.toSeconds() % 60 ) + " "
+ bundle.getString("timeSeconds") + "."
        );
    }
```

```
    /**
     * Shows the interval results between predefined hours.
     */
    public void showIntervalsResults() {
        ResourceBundle bundle = ResourceBundle.getBundle("hours");

        HourWithDates hour1 = new HourWithDates(1234,
"veryHighRidership",
                ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 7, 0, 0,
0, ZoneId.of("Europe/Kiev")));
        HourWithDates hour2 = new HourWithDates(123, "lowRidership",
                        ZonedDateTime.of(2024, Month.JANUARY.getValue(),
1, 7, 0, 0, 0, ZoneId.of("Europe/Kiev")));
        HourWithDates hour3 = new HourWithDates(654, "highRidership",
                        ZonedDateTime.of(2024, Month.JUNE.getValue(),
15, 18, 15, 34, 0, ZoneId.of("Europe/Kiev")));
        System.out.println(bundle.getString("createdHours") + ":");
        System.out.println(hour1 + "\n" + hour2 + "\n" + hour3);

        DateTimeFormatter dateTimeFormatter =
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG).withLocale(Loca
le.getDefault());
        System.out.print(bundle.getString("timeIntervalBetweenHours") +
" [ "
                + hour1.getOperatingHour().format(dateTimeFormatter) +
"; "
                + hour2.getOperatingHour().format(dateTimeFormatter) + "
]:\t");
        printInterval(getInterval(hour1, hour2));
        System.out.print(bundle.getString("timeIntervalBetweenHours") +
" [ "
                + hour1.getOperatingHour().format(dateTimeFormatter) +
"; "
                + hour3.getOperatingHour().format(dateTimeFormatter) + "
]:\t");
        printInterval(getInterval(hour1, hour3));
        System.out.print(bundle.getString("timeIntervalBetweenHours") +
" [ "
                + hour2.getOperatingHour().format(dateTimeFormatter) +
"; "
                + hour3.getOperatingHour().format(dateTimeFormatter) + "
]:\t");
        printInterval(getInterval(hour2, hour3));


        HourWithDates[] hours = createHours();
        System.out.println("\n" + bundle.getString("createdHourArray") +
":");
```

```
        Arrays.stream(createHours()).forEach(System.out::println);

        Duration smallestInterval =
findSmallestIntervalBetweenHours(hours);
        Duration largestInterval =
findLargestIntervalBetweenHours(hours);

        System.out.print(bundle.getString("smallestInterval") + ":\t");
        printInterval(smallestInterval);
        System.out.print(bundle.getString("largestInterval") + ":\t");
        printInterval(largestInterval);
    }

    /**
     * Performs testing the functionality of the {@link HourWithDates}
class:
     * <p>- the extended {@link Hour} class methods;
     * <p>- methods for managing {@link ZonedDateTime} {@link
HourWithDates#operatingHour} field,
     * <p>- methods for managing localized {@link HourWithDates} fields,
     * <p>- methods for calculating {@link Duration} time intervals and
displaying results.
     */
    public void testHour() {
        ResourceBundle bundle = ResourceBundle.getBundle("hours");
        System.out.println(
"\n**************************************************************************
***************\n\t\t\t\t\t\t\t"
                + bundle.getString("localizationText")
                +
"\n**************************************************************************
***************");

        System.out.println(bundle.getString("createdHour") + ":");
        System.out.println(this + "\n");

        showIntervalsResults();
    }

    /**
     * Performs testing of the {@link HourWithDates} class
functionality. The {@code args} are not used.
     * @param args the command-line arguments.
     */
    public static void main(String[] args) {
        Locale.setDefault(Locale.US);
        new HourWithDates().createHour().testHour();

        Locale.setDefault(new Locale("uk"));
```

```
        new HourWithDates().createHour().testHour();
    }
}
```

1.2.7. MetroStationWithLocalization.java:

```
package part2.lab2.task1;

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

import java.text.Collator;
import java.text.NumberFormat;
import java.time.Month;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.util.Arrays;
import java.util.Comparator;
import java.util.Locale;
import java.util.ResourceBundle;

/**
* The {@code MetroStationWithLocalization} class represents a Metro
Station with localized information.
* It extends the {@link MetroStationWithStreams} class and includes
methods for managing opening dates,
* sorting and searching for specific hours based on ridership or
comments.
*/
public class MetroStationWithLocalization extends
MetroStationWithStreams {
    /**
     * Represents the {@link ZonedDateTime} opening date of the metro
station.
     */
    private ZonedDateTime openingDate;

    /**
     * Constructs a new default {@link MetroStationWithLocalization}
object.
     */
    MetroStationWithLocalization() {}

    /**
     * Constructs a new {@link MetroStationWithLocalization} object with
the specified parameters.
```

```
     * @param name The name of the metro station.
     * @param openingDate The opening date of the metro station.
     */
    MetroStationWithLocalization(String name, ZonedDateTime openingDate)
{
        setName(name);
        setOpeningDate(openingDate);
    }

    /**
     * Gets the opening date of the metro station.
     * @return The opening date.
     */
    public ZonedDateTime getOpeningDate() {
        return openingDate;
    }

    /**
     * Sets the opening date of the metro station.
     * @param openingDate The opening date to set.
     */
    public void setOpeningDate(ZonedDateTime openingDate) {
        this.openingDate = openingDate;
    }


    /**
     * Overrides the {@link MetroStationWithStreams#toString()} method
and provides the string representing
     * the {@link MetroStationWithLocalization} object with included
localized information about the metro station.
     * @return A string representation of the {@link
MetroStationWithLocalization} object.
     */
    @Override
    public String toString() {
        ResourceBundle bundle = ResourceBundle.getBundle("hours");
        StringBuilder result  = new StringBuilder(
                bundle.getString("name") + ":\t\'" + getName() + "\'.\t"
                + bundle.getString("openingDate") + ":\t"
                        + getOpeningDate().format(

DateTimeFormatter.ofLocalizedDate(FormatStyle.LONG)

.withLocale(Locale.getDefault()))
        );

        Arrays.stream(getHours()).forEach( hour -> {
            result.append("\n").append(hour);
```

```
        });

        return result + "";
    }

    /**
     * Overridden descending length of the comment sorting method using
{@link Comparator#compare(Object, Object)} interface.
     */
    @Override
    public void sortByDescendingCommentLength() {
        Hour[] hours = getHours();
        Arrays.sort(hours, new Comparator<Hour>() {
            @Override
            public int compare(Hour hour1, Hour hour2) {
                return Integer.compare(hour2.getCommentLength(),
hour1.getCommentLength());
            }
        });
        setHours(hours);
    }

    /**
     * Overridden the alphabetically comment sorting method using {@link
Collator#compare(Object, Object)} interface.
     */
    public void sortByCommentAlphabetically() {
        Hour[] hours = getHours();
        Arrays.sort(hours, new Comparator<Hour>() {
            Collator collator =
Collator.getInstance(Locale.getDefault());

            @Override
            public int compare(Hour hour1, Hour hour2) {
                return collator.compare(hour1.getComment(),
hour2.getComment());
            }
        });
        setHours(hours);
    }

    /**
     * Finds hours with a word fragment at the start or end of the
comment.
     * @param text The word fragment to search for.
     * @return An array of HourWithDates objects.
     */
    public HourWithDates[]
findHoursWithWordFragmentAtStartOrEndOfComment(String text) {
```

```
        return Arrays.stream(getHours())
                .filter((hour -> {
                    for (String word : hour.getComment().split("\\s")) {
                        if (word.startsWith(text) ||
word.endsWith(text)) {
                            return true;
                        }
                    }

                    return false;
                }))
                .toArray(HourWithDates[]::new);
    }

    /**
     * Creates a new {@link MetroStationWithLocalization} object with
default values.
     * @return The created {@link MetroStationWithLocalization} object.
     */
    public MetroStationWithLocalization createMetroStation() {
        ResourceBundle bundle = ResourceBundle.getBundle("hours");

        setName(bundle.getString("metroStationName"));
        setOpeningDate(ZonedDateTime.of(1984, Month.AUGUST.getValue(),
10, 8, 0, 0, 0, ZoneId.of("Europe/Kiev")));

        addHour(new HourWithDates(1100, "veryHighRidership",
                ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 8, 0, 0,
0, ZoneId.of("Europe/Kiev"))));
        addHour(new HourWithDates(110, "lowRidership",
                ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 12, 0,
0, 0, ZoneId.of("Europe/Kiev"))));
        addHour(new HourWithDates(650, "highRidership",
                ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 14, 0,
0, 0, ZoneId.of("Europe/Kiev"))));
        addHour(new HourWithDates(532, "highRidership",
                ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 15, 0,
0, 0, ZoneId.of("Europe/Kiev"))));
        addHour(new HourWithDates(60, "veryLowRidership",
                ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 22, 0,
0, 0, ZoneId.of("Europe/Kiev"))));
        addHour(new HourWithDates(188, "lowRidership",
                ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 10, 0,
0, 0, ZoneId.of("Europe/Kiev"))));
        addHour(new HourWithDates(200, "mediumRidership",
                ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 5, 0, 0,
0, ZoneId.of("Europe/Kiev"))));
        addHour(new HourWithDates(200, "mediumRidership",
                ZonedDateTime.of(2024, Month.MAY.getValue(), 3, 4, 0, 0,
```

```
0, ZoneId.of("Europe/Kiev"))));

        return this;
    }

    /**
     * Prints the results of calculation of the total ridership for the
metro station.
     */
    @Override
    public void printTotalRidership() {
        Integer totalRidership = calculateTotalRidership();

        ResourceBundle bundle = ResourceBundle.getBundle("hours");
        NumberFormat numberFormat =
NumberFormat.getInstance(Locale.getDefault());

System.out.print(bundle.getString("totalRidershipForMetroStation") +
":\t");

        if (totalRidership == null) {
            System.out.print(bundle.getString("noHoursWithRidership") +
".");
        } else {
            System.out.println(numberFormat.format(totalRidership));
        }
    }

    /**
     * Prints hours with minimum ridership.
     */
    @Override
    public void printHoursWithMinRidership() {
        Hour[] hours = findHoursWithMinRidership();

        ResourceBundle bundle = ResourceBundle.getBundle("hours");
        System.out.print(bundle.getString("findHoursWithMinRidership") +
":\t");

        if (hours == null) {
            System.out.print(bundle.getString("noHoursWithRidership") +
".");
        } else {
            System.out.println();
            printHours(hours);
        }
    }

    /**
```

```
    * Prints hours with maximum word count of comment.
    */
   @Override
   public void printHoursWithMaxWordCountOfComment() {
       Hour[] hours = findHoursWithMaxWordCountOfComment();

       ResourceBundle bundle = ResourceBundle.getBundle("hours");

System.out.print(bundle.getString("findHoursWithMaxWordCountInComment")
+ ":\t");

       if (hours == null) {
           System.out.print(bundle.getString("noHoursWithRidership") +
".");
       } else {
           System.out.println();
           printHours(hours);
       }
   }

   /**
    * Prints hours with a word fragment in the comment.
    */
   public void printHoursWithWordFragmentInComment() {
       ResourceBundle bundle = ResourceBundle.getBundle("hours");
       HourWithDates[] hours =
findHoursWithWordFragmentAtStartOrEndOfComment(bundle.getString("findWo
rdAtStart"));


System.out.print(bundle.getString("findHoursWithWordFragmentInComment")
+ " [\""
               + bundle.getString("findWordAtStart") + "\"]:\t");

       if (hours == null) {
           System.out.print(bundle.getString("noHoursWithComment") +
".");
       } else {
           System.out.println();
           printHours(hours);
       }

       hours =
findHoursWithWordFragmentAtStartOrEndOfComment(bundle.getString("findWo
rdAtEnd"));


System.out.print(bundle.getString("findHoursWithWordFragmentInComment")
+ " [\""
```

```
                + bundle.getString("findWordAtEnd") + "\"]:\t");

        if (hours == null) {
            System.out.print(bundle.getString("noHoursWithComment") +
".");
        } else {
            System.out.println();
            printHours(hours);
        }
    }

    /**
     * Shows the creating results of the metro station.
     */
    public void showCreatingResults() {
        ResourceBundle bundle = ResourceBundle.getBundle("hours");
        System.out.println(bundle.getString("creatingResults") + ":");
        System.out.println(this);
    }

    /**
     * Shows the search results of the metro station.
     */
    @Override
    public void showSearchResults() {
        ResourceBundle bundle = ResourceBundle.getBundle("hours");
        System.out.println(bundle.getString("searchingResults") + ":");
        printTotalRidership();
        printHoursWithMinRidership();
        printHoursWithMaxWordCountOfComment();
        printHoursWithWordFragmentInComment();
    }

    /**
     * Shows the sorting results of the metro station.
     */
    public void showSortingResults() {
        ResourceBundle bundle = ResourceBundle.getBundle("hours");
        System.out.println(bundle.getString("sortingResults") + ":");


System.out.println(bundle.getString("sortedHoursByDecrRidership") +
":");
        sortByDecreasingRidership();
        System.out.println(this);


System.out.println(bundle.getString("sortedHoursByDescCommentLength") +
":");
```

```
        sortByDescendingCommentLength();
        System.out.println(this);


System.out.println(bundle.getString("sortedHoursCommentAlphabetically")
+ ":");
        sortByCommentAlphabetically();
        System.out.println(this);
    }

    /**
     * Tests the MetroStationWithLocalization class.
     */
    public void testMetroStation() {
        ResourceBundle bundle = ResourceBundle.getBundle("hours");
        System.out.println(
"\n**********************************************************************
***************\n\t\t\t\t\t\t"
                + bundle.getString("localizationText")
                +
"\n**********************************************************************
***************" );
        showCreatingResults();
        System.out.println();
        showSearchResults();
        System.out.println();
        showSortingResults();
        System.out.println();
    }

    /**
     * The main method to run the {@link MetroStationWithLocalization}
class.
     * @param args Command-line arguments.
     */
    public static void main(String[] args) {
        Locale.setDefault(Locale.US);
        new MetroStationWithLocalization()
                .createMetroStation()
                .testMetroStation();

        Locale.setDefault(new Locale("uk"));
        new MetroStationWithLocalization()
                .createMetroStation()
                .testMetroStation();
    }
}
```

1.2.8. hours.properties:

```
hour=Hour
operationHour=Operating Hour
ridership=Ridership
comment=Comment
opened=Opened
veryHighRidership=Very high ridership
veryLowRidership=Very low ridership
mediumRidership=Medium ridership
lowRidership=Low ridership
highRidership=High ridership
name=Metro Station Name
metroStationName=Politekhnichna
totalRidershipForMetroStation=Total Ridership for Metro Station
findHoursWithMinRidership=Hours with minimal ridership
findHoursWithMaxWordCountInComment=Hours with the maximum word count in
a comment
searchingResults=SEARCHING RESULTS
noHoursWithRidership=There are no hours with ridership
sortingResults=SORTING RESULTS
sortedHoursByDecrRidership=Sorted Hours by decreasing ridership
sortedHoursByDescCommentLength=Sorted Hours by descending comment
length
creatingResults=CREATED METRO STATION
localizationText=RESULTS FOR DEFAULT LOCALIZATION
openingDate=Opening Date
sortedHoursCommentAlphabetically=Sorted Hours by comment alphabetically
durationResults=Results of calculating time intervals
smallestInterval=The smallest time interval between Hours
largestInterval=The largest time interval between Hours
createdHour=Created Hour
createdHourArray=Created Hour Array
timeDays=days
timeHours=h
timeMinutes=min
timeSeconds=sec
createdHours=Created Hours
timeIntervalBetweenHours=Time interval between Hours
findWordAtStart=Med
findWordAtEnd=ry
findHoursWithWordFragmentInComment=Hours with the word fragment in the
comment
noHoursWithComment=There are no hours with comment
```

1.2.9. hours_en_US.properties:

```
hour=Hour
operationHour=Operating Hour
ridership=Ridership
comment=Comment
opened=Opened
veryHighRidership=Very high ridership
veryLowRidership=Very low ridership
mediumRidership=Medium ridership
lowRidership=Low ridership
highRidership=High ridership
name=Metro Station Name
metroStationName=Politekhnichna
totalRidershipForMetroStation=Total Ridership for Metro Station
findHoursWithMinRidership=Hours with minimal ridership
findHoursWithMaxWordCountInComment=Hours with the maximum word count in
a comment
searchingResults=SEARCHING RESULTS
noHoursWithRidership=There are no hours with ridership
sortingResults=SORTING RESULTS
sortedHoursByDecrRidership=Sorted Hours by decreasing ridership
sortedHoursByDescCommentLength=Sorted Hours by descending comment
length
creatingResults=CREATED METRO STATION
localizationText=RESULTS FOR EN(US) LOCALIZATION
openingDate=Opening Date
sortedHoursCommentAlphabetically=Sorted Hours by comment alphabetically
durationResults=Results of calculating time intervals
smallestInterval=The smallest time interval between Hours
largestInterval=The largest time interval between Hours
createdHour=Created Hour
createdHourArray=Created Hour Array
timeHours=h
timeMinutes=min
timeSeconds=sec
timeDays=days
createdHours=Created Hours
timeIntervalBetweenHours=Time interval between Hours
findWordAtStart=Med
findWordAtEnd=ry
findHoursWithWordFragmentInComment=Hours with the word fragment in the
comment
noHoursWithComment=There are no hours with comment
```

1.2.10. hours_uk.properties.java:

```
hour=Година
operationHour=Операційна Година
ridership=Пасажиропотік
comment=Коментар
opened=Дата Відкриття
veryHighRidership=Дуже високий пасажиропотік
veryLowRidership=Дуже низький пасажиропотік
mediumRidership=Середній пасажиропотік
lowRidership=Низький пасажиропотік
highRidership=Високий пасажиропотік
name=Назва Станції Метро
metroStationName=Політехнічна
totalRidershipForMetroStation=Загальний пасажиропотік Станції Метро
findHoursWithMinRidership=Години з мінімальним пасажиропотоком
findHoursWithMaxWordCountInComment=Години з максимальною кількістю слів
в коментарі
searchingResults=РЕЗУЛЬТАТИ ПОШУКУ
noHoursWithRidership=Немає годин із пасажиропотоком
sortingResults=РЕЗУЛЬТАТИ СОРТУВАННЯ
sortedHoursByDecrRidership=Відсортовані Години за зменшенням
пасажиропотоку
sortedHoursByDescCommentLength=Відсортовані Години за зменшенням
довжини коментаря
creatingResults=СТВОРЕНА СТАНЦІЯ МЕТРО
localizationText=РЕЗУЛЬТАТИ ДЛЯ UK ЛОКАЛІЗАЦІЇ
openingDate=Дата Відкриття
sortedHoursCommentAlphabetically=Відсортовані Години за алфавітом
коментаря
durationResults=Результати обрахунків проміжків часу
smallestInterval=Найменший проміжок часу між Годинами
largestInterval=Найбільший проміжок часу між Годинами
createdHour=Створена Година
createdHourArray=Створений масив Годин
timeHours=год
timeMinutes=хв
timeSeconds=сек
timeDays=днів
createdHours=Створені Години
timeIntervalBetweenHours=Інтервал часу між Годинами
findWordAtStart=Сер
findWordAtEnd=же
findHoursWithWordFragmentInComment=Години з фрагментом слова в
коментарі
noHoursWithComment=Немає годин з коментарем
```

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

```
*************************************************************************
                    RESULTS FOR EN(US) LOCALIZATION
*************************************************************************
Created Hour:
    Hour    { Operating Hour: May 3, 2024 at 5:00:00 PM EEST;   Ridership = 1,234;  Comment = 'Very high ridership' }

Created Hours:
    Hour    { Operating Hour: May 3, 2024 at 7:00:00 AM EEST;   Ridership = 1,234;  Comment = 'Very high ridership' }
    Hour    { Operating Hour: January 1, 2024 at 7:00:00 AM EET;    Ridership = 123;    Comment = 'Low ridership' }
    Hour    { Operating Hour: June 15, 2024 at 6:15:34 PM EEST; Ridership = 654;    Comment = 'High ridership' }
Time interval between Hours [ May 3, 2024 at 7:00:00 AM EEST; January 1, 2024 at 7:00:00 AM EET ]:  122 days    23 h    0 min    0 sec.
Time interval between Hours [ May 3, 2024 at 7:00:00 AM EEST; June 15, 2024 at 6:15:34 PM EEST ]:   43 days    11 h    15 min    34 sec.
Time interval between Hours [ January 1, 2024 at 7:00:00 AM EET; June 15, 2024 at 6:15:34 PM EEST ]:    166 days    10 h    15 min    34 sec.

Created Hour Array:
    Hour    { Operating Hour: May 3, 2024 at 8:00:00 AM EEST;   Ridership = 1,100;  Comment = 'Very high ridership' }
    Hour    { Operating Hour: May 3, 2024 at 12:00:00 PM EEST;  Ridership = 110;    Comment = 'Low ridership' }
    Hour    { Operating Hour: May 3, 2024 at 2:00:00 PM EEST;   Ridership = 650;    Comment = 'High ridership' }
    Hour    { Operating Hour: May 3, 2024 at 3:00:00 PM EEST;   Ridership = 532;    Comment = 'High ridership' }
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 PM EEST;  Ridership = 60; Comment = 'Very low ridership' }
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 AM EEST;  Ridership = 188;    Comment = 'Low ridership' }
    Hour    { Operating Hour: May 3, 2024 at 5:00:00 AM EEST;   Ridership = 200;    Comment = 'Medium ridership' }
    Hour    { Operating Hour: May 18, 2024 at 5:08:32 PM EEST;  Ridership = 200;    Comment = 'Medium ridership' }
The smallest time interval between Hours:   0 days    1 h    0 min    0 sec.
The largest time interval between Hours:    15 days    12 h    8 min    32 sec.
```

Рисунок 1.3.1 – Результати № 1 роботи програмного коду класу HourWithDates.java

```
*************************************************************************
                    РЕЗУЛЬТАТИ ДЛЯ UK ЛОКАЛІЗАЦІЇ
*************************************************************************
Створена Година:
    Година  { Операційна Година: 3 травня 2024 р. о 17:00:00 EEST;  Пасажиропотік = 1 234;  Коментар = 'Дуже високий пасажиропотік' }

Створені Години:
    Година  { Операційна Година: 3 травня 2024 р. о 07:00:00 EEST;  Пасажиропотік = 1 234;  Коментар = 'Дуже високий пасажиропотік' }
    Година  { Операційна Година: 1 січня 2024 р. о 07:00:00 EET;    Пасажиропотік = 123;    Коментар = 'Низький пасажиропотік' }
    Година  { Операційна Година: 15 червня 2024 р. о 18:15:34 EEST; Пасажиропотік = 654;    Коментар = 'Високий пасажиропотік' }
Інтервал часу між Годинами [ 3 травня 2024 р. о 07:00:00 EEST; 1 січня 2024 р. о 07:00:00 EET ]:    122 днів    23 год    0 хв    0 сек.
Інтервал часу між Годинами [ 3 травня 2024 р. о 07:00:00 EEST; 15 червня 2024 р. о 18:15:34 EEST ]: 43 днів    11 год    15 хв    34 сек.
Інтервал часу між Годинами [ 1 січня 2024 р. о 07:00:00 EET; 15 червня 2024 р. о 18:15:34 EEST ]:   166 днів    10 год    15 хв    34 сек.

Створений масив Годин:
    Година  { Операційна Година: 3 травня 2024 р. о 08:00:00 EEST;  Пасажиропотік = 1 100;  Коментар = 'Дуже високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 12:00:00 EEST;  Пасажиропотік = 110;    Коментар = 'Низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 14:00:00 EEST;  Пасажиропотік = 650;    Коментар = 'Високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 15:00:00 EEST;  Пасажиропотік = 532;    Коментар = 'Високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 22:00:00 EEST;  Пасажиропотік = 60; Коментар = 'Дуже низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 10:00:00 EEST;  Пасажиропотік = 188;    Коментар = 'Низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 05:00:00 EEST;  Пасажиропотік = 200;    Коментар = 'Середній пасажиропотік' }
    Година  { Операційна Година: 18 травня 2024 р. о 17:08:32 EEST; Пасажиропотік = 200;    Коментар = 'Середній пасажиропотік' }
Найменший проміжок часу між Годинами:   0 днів    1 год    0 хв    0 сек.
Найбільший проміжок часу між Годинами:  15 днів    12 год    8 хв    32 сек.

Process finished with exit code 0
```

Рисунок 1.3.2 – Результати № 2 роботи програмного коду класу HourWithDates.java

```
********************************************************************************
                        RESULTS FOR EN(US) LOCALIZATION
********************************************************************************
CREATED METRO STATION:
Metro Station Name: 'Politekhnichna'.   Opening Date:   August 10, 1984
    Hour    { Operating Hour: May 3, 2024 at 8:00:00 AM EEST;   Ridership = 1,100;  Comment = 'Very high ridership' }
    Hour    { Operating Hour: May 3, 2024 at 12:00:00 PM EEST;  Ridership = 110;    Comment = 'Low ridership' }
    Hour    { Operating Hour: May 3, 2024 at 2:00:00 PM EEST;   Ridership = 650;    Comment = 'High ridership' }
    Hour    { Operating Hour: May 3, 2024 at 3:00:00 PM EEST;   Ridership = 532;    Comment = 'High ridership' }
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 PM EEST;  Ridership = 60; Comment = 'Very low ridership' }
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 AM EEST;  Ridership = 188;    Comment = 'Low ridership' }
    Hour    { Operating Hour: May 3, 2024 at 5:00:00 AM EEST;   Ridership = 200;    Comment = 'Medium ridership' }

SEARCHING RESULTS:
Total Ridership for Metro Station:  2,840
Hours with minimal ridership:
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 PM EEST;  Ridership = 60; Comment = 'Very low ridership' }
Hours with the maximum word count in a comment:
    Hour    { Operating Hour: May 3, 2024 at 8:00:00 AM EEST;   Ridership = 1,100;  Comment = 'Very high ridership' }
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 PM EEST;  Ridership = 60; Comment = 'Very low ridership' }
Hours with the word fragment in the comment ["Med"]:
    Hour    { Operating Hour: May 3, 2024 at 5:00:00 AM EEST;   Ridership = 200;    Comment = 'Medium ridership' }
Hours with the word fragment in the comment ["ry"]:
    Hour    { Operating Hour: May 3, 2024 at 8:00:00 AM EEST;   Ridership = 1,100;  Comment = 'Very high ridership' }
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 PM EEST;  Ridership = 60; Comment = 'Very low ridership' }

SORTING RESULTS:
Sorted Hours by decreasing ridership:
Metro Station Name: 'Politekhnichna'.   Opening Date:   August 10, 1984
    Hour    { Operating Hour: May 3, 2024 at 8:00:00 AM EEST;   Ridership = 1,100;  Comment = 'Very high ridership' }
    Hour    { Operating Hour: May 3, 2024 at 2:00:00 PM EEST;   Ridership = 650;    Comment = 'High ridership' }
    Hour    { Operating Hour: May 3, 2024 at 3:00:00 PM EEST;   Ridership = 532;    Comment = 'High ridership' }
    Hour    { Operating Hour: May 3, 2024 at 5:00:00 AM EEST;   Ridership = 200;    Comment = 'Medium ridership' }
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 AM EEST;  Ridership = 188;    Comment = 'Low ridership' }
    Hour    { Operating Hour: May 3, 2024 at 12:00:00 PM EEST;  Ridership = 110;    Comment = 'Low ridership' }
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 PM EEST;  Ridership = 60; Comment = 'Very low ridership' }
Sorted Hours by descending comment length:
Metro Station Name: 'Politekhnichna'.   Opening Date:   August 10, 1984
    Hour    { Operating Hour: May 3, 2024 at 8:00:00 AM EEST;   Ridership = 1,100;  Comment = 'Very high ridership' }
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 PM EEST;  Ridership = 60; Comment = 'Very low ridership' }
    Hour    { Operating Hour: May 3, 2024 at 5:00:00 AM EEST;   Ridership = 200;    Comment = 'Medium ridership' }
    Hour    { Operating Hour: May 3, 2024 at 2:00:00 PM EEST;   Ridership = 650;    Comment = 'High ridership' }
    Hour    { Operating Hour: May 3, 2024 at 3:00:00 PM EEST;   Ridership = 532;    Comment = 'High ridership' }
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 AM EEST;  Ridership = 188;    Comment = 'Low ridership' }
    Hour    { Operating Hour: May 3, 2024 at 12:00:00 PM EEST;  Ridership = 110;    Comment = 'Low ridership' }
Sorted Hours by comment alphabetically:
Metro Station Name: 'Politekhnichna'.   Opening Date:   August 10, 1984
    Hour    { Operating Hour: May 3, 2024 at 2:00:00 PM EEST;   Ridership = 650;    Comment = 'High ridership' }
    Hour    { Operating Hour: May 3, 2024 at 3:00:00 PM EEST;   Ridership = 532;    Comment = 'High ridership' }
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 AM EEST;  Ridership = 188;    Comment = 'Low ridership' }
    Hour    { Operating Hour: May 3, 2024 at 12:00:00 PM EEST;  Ridership = 110;    Comment = 'Low ridership' }
    Hour    { Operating Hour: May 3, 2024 at 5:00:00 AM EEST;   Ridership = 200;    Comment = 'Medium ridership' }
    Hour    { Operating Hour: May 3, 2024 at 8:00:00 AM EEST;   Ridership = 1,100;  Comment = 'Very high ridership' }
    Hour    { Operating Hour: May 3, 2024 at 10:00:00 PM EEST;  Ridership = 60; Comment = 'Very low ridership' }
```

Рисунок 1.3.3 – Результати № 3 роботи програмного коду класу MetroStationWithLocalization.java

```
********************************************************************************
                        РЕЗУЛЬТАТИ ДЛЯ UK ЛОКАЛІЗАЦІЇ
********************************************************************************
СТВОРЕНА СТАНЦІЯ МЕТРО:
Назва Станції Метро:    'Політехнічна'. Дата Відкриття: 10 серпня 1984 р.
    Година  { Операційна Година: 3 травня 2024 р. о 08:00:00 EEST;  Пасажиропотік = 1 100;  Коментар = 'Дуже високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 12:00:00 EEST;  Пасажиропотік = 110;    Коментар = 'Низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 14:00:00 EEST;  Пасажиропотік = 650;    Коментар = 'Високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 15:00:00 EEST;  Пасажиропотік = 532;    Коментар = 'Високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 22:00:00 EEST;  Пасажиропотік = 60; Коментар = 'Дуже низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 10:00:00 EEST;  Пасажиропотік = 188;    Коментар = 'Низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 05:00:00 EEST;  Пасажиропотік = 200;    Коментар = 'Середній пасажиропотік' }

РЕЗУЛЬТАТИ ПОШУКУ:
Загальний пасажиропотік Станції Метро:  2 840
Години з мінімальним пасажипотоком:
    Година  { Операційна Година: 3 травня 2024 р. о 22:00:00 EEST;  Пасажиропотік = 60; Коментар = 'Дуже низький пасажиропотік' }
Години з максимальною кількістю слів в коментарі:
    Година  { Операційна Година: 3 травня 2024 р. о 08:00:00 EEST;  Пасажиропотік = 1 100;  Коментар = 'Дуже високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 22:00:00 EEST;  Пасажиропотік = 60; Коментар = 'Дуже низький пасажиропотік' }
Години з фрагментом слова в коментарі ["Сер"]:
    Година  { Операційна Година: 3 травня 2024 р. о 05:00:00 EEST;  Пасажиропотік = 200;    Коментар = 'Середній пасажиропотік' }
Години з фрагментом слова в коментарі ["же"]:
    Година  { Операційна Година: 3 травня 2024 р. о 08:00:00 EEST;  Пасажиропотік = 1 100;  Коментар = 'Дуже високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 22:00:00 EEST;  Пасажиропотік = 60; Коментар = 'Дуже низький пасажиропотік' }

РЕЗУЛЬТАТИ СОРТУВАННЯ:
Відсортовані Години за зменшенням пасажипотоку:
Назва Станції Метро:    'Політехнічна'. Дата Відкриття: 10 серпня 1984 р.
    Година  { Операційна Година: 3 травня 2024 р. о 08:00:00 EEST;  Пасажиропотік = 1 100;  Коментар = 'Дуже високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 14:00:00 EEST;  Пасажиропотік = 650;    Коментар = 'Високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 15:00:00 EEST;  Пасажиропотік = 532;    Коментар = 'Високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 05:00:00 EEST;  Пасажиропотік = 200;    Коментар = 'Середній пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 10:00:00 EEST;  Пасажиропотік = 188;    Коментар = 'Низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 12:00:00 EEST;  Пасажиропотік = 110;    Коментар = 'Низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 22:00:00 EEST;  Пасажиропотік = 60; Коментар = 'Дуже низький пасажиропотік' }
Відсортовані Години за зменшенням довжини коментаря:
Назва Станції Метро:    'Політехнічна'. Дата Відкриття: 10 серпня 1984 р.
    Година  { Операційна Година: 3 травня 2024 р. о 08:00:00 EEST;  Пасажиропотік = 1 100;  Коментар = 'Дуже високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 22:00:00 EEST;  Пасажиропотік = 60; Коментар = 'Дуже низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 05:00:00 EEST;  Пасажиропотік = 200;    Коментар = 'Середній пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 14:00:00 EEST;  Пасажиропотік = 650;    Коментар = 'Високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 15:00:00 EEST;  Пасажиропотік = 532;    Коментар = 'Високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 10:00:00 EEST;  Пасажиропотік = 188;    Коментар = 'Низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 12:00:00 EEST;  Пасажиропотік = 110;    Коментар = 'Низький пасажиропотік' }
Відсортовані Години за алфавітом коментаря:
Назва Станції Метро:    'Політехнічна'. Дата Відкриття: 10 серпня 1984 р.
    Година  { Операційна Година: 3 травня 2024 р. о 14:00:00 EEST;  Пасажиропотік = 650;    Коментар = 'Високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 15:00:00 EEST;  Пасажиропотік = 532;    Коментар = 'Високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 08:00:00 EEST;  Пасажиропотік = 1 100;  Коментар = 'Дуже високий пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 22:00:00 EEST;  Пасажиропотік = 60; Коментар = 'Дуже низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 10:00:00 EEST;  Пасажиропотік = 188;    Коментар = 'Низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 12:00:00 EEST;  Пасажиропотік = 110;    Коментар = 'Низький пасажиропотік' }
    Година  { Операційна Година: 3 травня 2024 р. о 05:00:00 EEST;  Пасажиропотік = 200;    Коментар = 'Середній пасажиропотік' }


Process finished with exit code 0
```

Рисунок 1.3.4 – Результати № 4 роботи програмного коду класу MetroStationWithLocalization.java

2. Завдання №2 до лабораторної роботи

2.1. Уведення дати

Реалізувати програму, в якій користувач вводить рядок. Програма перевіряє, чи відповідає рядок представленню дати, прийнятому в Україні.

Перевірка здійснюється за допомогою регулярних виразів. Якщо рядок не відповідає вимогам, виводиться повідомлення про помилку. В іншому випадку створюються і виводяться на консоль об'єкти Date, GregorianCalendar і LocalDate.

2.2. Програмний код реалізації завдання № 2

2.2.1. DateValidator.java:

```
package part2.lab2.task2;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
* The {@code DateParser} class provides utility methods for validating
and parsing date strings.
*/
public class DateValidator {
   /**
    * Validates if the provided input string adheres to the expected
date format (dd.MM.yyyy) with optional spaces.
    * @param input the date string to be validated;
    * @return {@code true} if the input matches the expected format,
{@code true} otherwise.
    */
   public static boolean isValidDateFormat(String input) {
       String regex =
"\\s*(0[1-9]|[12][0-9]|3[01])[.](0[1-9]|1[012])[.](\\d{4})\\s*";
       Pattern pattern = Pattern.compile(regex);
       Matcher matcher = pattern.matcher(input);

       return matcher.matches();
   }

   /**
    * Parses the provided input string into the {@link Date} object in
the format dd.MM.yyyy
    * by removing spaces from both ends of the string before parsing.
    * @param input The date string to be parsed;
    * @return a Date object representing the parsed date;
    * @throws ParseException if the input string does not conform to
```

```
the expected format.
     */
    public static Date parseStringToDate(String input) throws
ParseException {
        SimpleDateFormat dateFormat = new
SimpleDateFormat("dd.MM.yyyy");
        Date date = dateFormat.parse(input.trim());

        return date;
    }

    /**
     * Parses the provided input string into the {@link
GregorianCalendar} object in the format dd.MM.yyyy
     * by splitting the entered string into parts using dots,
     * extracts the day, month (Months in the Calendar class start from
0) and year,
     * and then creates an object of type GregorianCalendar with these
values of day, month and year.
     * @param input The date string to be parsed.
     * @return A GregorianCalendar object representing the parsed date.
     */
    public static GregorianCalendar
parseStringToGregorianCalendar(String input) {
        String[] parts = input.trim().split("\\.");
        int day = Integer.parseInt(parts[0]);
        int month = Integer.parseInt(parts[1]) - 1;
        int year = Integer.parseInt(parts[2]);
        GregorianCalendar calendar = new GregorianCalendar(year, month,
day);

        return calendar;
    }

    /**
     * Parses the provided input string into the {@link LocalDate}
object in the format dd.MM.yyyy
     * by removing spaces from both ends of the string before parsing.
     * @param input the date string to be parsed;
     * @return a LocalDate object representing the parsed date.
     */
    public static LocalDate parseStringToLocalDate(String input) {
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd.MM.yyyy");
        LocalDate localDate = LocalDate.parse(input.trim(), formatter);

        return localDate;
    }
}
```

2.2.2. DateValidatorDemo.java:

```java
package part2.lab2.task2;

import java.text.ParseException;
import java.time.LocalDate;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.stream.IntStream;

/**
* The {@code DateParserTester} class is a demonstration of
functionality of the {@link DateValidator}.
*/
public class DateValidatorDemo {
   /** The constant array representing the date strings with valid and
invalid data and date formats used in the tests. */
   public static final String[] DATE_STRINGS = {
           "25.12.2023",
           " 01.01.2024 ",
           "29.02.2024",
           "31.03.2025",
           "25/12/2023",
           "5.3.2023",
           "32.12.2023",
           "15.13.2023",
           "31.02.202",
           "opened",
           "12.02/2023",
           "2023.12.31",
           "32.12.2023",
           "12.2023",
           "abc.12.2024",
           "20022024"
   };

   /**
    * Performs testing of the functionality of the {@link
DateValidator}. The {@code args} are not used.
    * @param args the command-line arguments (not used).
    */
   public static void main(String[] args) {
       IntStream.range(0, DATE_STRINGS.length).forEach(i -> {
           System.out.println("\nThe date string is:\t\"" +
DATE_STRINGS[i] + "\"");

           System.out.println("The results:");
```

```
            if (DateValidator.isValidDateFormat(DATE_STRINGS[i])) {
                try {
                    Date date =
DateValidator.parseStringToDate(DATE_STRINGS[i]);
                    System.out.println("\tParsed String to Date
object:\t" + date);

                    GregorianCalendar calendar =
DateValidator.parseStringToGregorianCalendar(DATE_STRINGS[i]);
                    System.out.println("\tParsed String to
GregorianCalendar Object:\t" + calendar.getTime());

                    LocalDate localDate =
DateValidator.parseStringToLocalDate(DATE_STRINGS[i]);
                    System.out.println("\tParsed String to LocalDate
Object:\t" + localDate);
                } catch (ParseException e) {
                    System.out.println("Unable to identify the date");
                    System.err.println(e.getMessage());
                }
            } else {
                System.out.println("Invalid date format entered");
            }
        });
    }
}
```

2.3. Програмний код модульного тестування з використанням Junit завдання № 2

2.3.1. DateValidatorTest.java:

```
package part2.lab2.task2;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import java.text.ParseException;
import java.time.LocalDate;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.stream.IntStream;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static part2.lab2.task2.DateValidatorDemo.DATE_STRINGS;

/**
* The {@code DateValidatorTest} class provides unit tests for the
```

```
{@link DateValidator} class.
* It tests the parsing the provided string into the {@link Date},
{@link GregorianCalendar}
* and {@link LocalDate} (in the format dd.MM.yyyy) objects.
*/
class DateValidatorTest {
   /** The constant array representing strings with valid results of
parsing strings to Date and GregorianCalendar
    *  objects used in the tests.
    */
   private static final String[] DATE_AND_GCALNEDAR_OBJS = {
           "Mon Dec 25 00:00:00 EET 2023",
           "Mon Jan 01 00:00:00 EET 2024",
           "Thu Feb 29 00:00:00 EET 2024",
           "Mon Mar 31 00:00:00 EEST 2025"
   };

   /** The constant array representing strings with valid results of
parsing strings to LocalDate objects used in the tests. */
   private static final String[] LOCAL_DATE_OBJS = {
           "2023-12-25",
           "2024-01-01",
           "2024-02-29",
           "2025-03-31"
   };

   /** Tests the parsing of date strings to Date objects
    * by {@link DateValidator#parseStringToDate(String)} method.
    */
   @Test
   @DisplayName("Should verify parsed Date object from String")
   public void testParseStringToDate() {
       IntStream.range(0, DATE_STRINGS.length)
               .forEach(i -> {
                   if
(DateValidator.isValidDateFormat(DATE_STRINGS[i])) {
                       try {
                           assertEquals(
                                   DATE_AND_GCALNEDAR_OBJS[i],

DateValidator.parseStringToDate(DATE_STRINGS[i]).toString()
                           );
                       } catch (ParseException e) {
                           System.err.println(e.getMessage());
                       }
                   }
               });
   }
```

```
    /** Tests the parsing of date strings to GregorianCalendar objects
     * by {@link DateValidator#parseStringToGregorianCalendar(String)}
method.
     */
    @Test
    @DisplayName("Should verify parsed GregorianCalendar object from
String")
    public void testParseStringToGregorianCalendar() {
        IntStream.range(0, DATE_STRINGS.length)
                .forEach(i -> {
                    if
(DateValidator.isValidDateFormat(DATE_STRINGS[i])) {
                        assertEquals(
                                DATE_AND_GCALNEDAR_OBJS[i],

DateValidator.parseStringToGregorianCalendar(DATE_STRINGS[i]).getTime()
.toString()
                        );
                    }
                });
    }

    /** Tests the parsing of date strings to LocalDate objects in the
format dd.MM.yyyy
     * by {@link DateValidator#parseStringToLocalDate(String)} method.
     */
    @Test
    @DisplayName("Should verify parsed LocalDate object from String")
    public void testParseStringToLocalDate() {
        IntStream.range(0, DATE_STRINGS.length)
                .forEach(i -> {
                    if
(DateValidator.isValidDateFormat(DATE_STRINGS[i])) {
                        assertEquals(
                                LOCAL_DATE_OBJS[i],

DateValidator.parseStringToLocalDate(DATE_STRINGS[i]).toString()
                        );
                    }
                });
    }
}
```

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

```
The date string is: "25.12.2023"
The results:
    Parsed String to Date object:   Mon Dec 25 00:00:00 EET 2023
    Parsed String to GregorianCalendar Object:  Mon Dec 25 00:00:00 EET 2023
    Parsed String to LocalDate Object:  2023-12-25

The date string is: " 01.01.2024 "
The results:
    Parsed String to Date object:   Mon Jan 01 00:00:00 EET 2024
    Parsed String to GregorianCalendar Object:  Mon Jan 01 00:00:00 EET 2024
    Parsed String to LocalDate Object:  2024-01-01

The date string is: "29.02.2024"
The results:
    Parsed String to Date object:   Thu Feb 29 00:00:00 EET 2024
    Parsed String to GregorianCalendar Object:  Thu Feb 29 00:00:00 EET 2024
    Parsed String to LocalDate Object:  2024-02-29

The date string is: "31.03.2025"
The results:
    Parsed String to Date object:   Mon Mar 31 00:00:00 EEST 2025
    Parsed String to GregorianCalendar Object:  Mon Mar 31 00:00:00 EEST 2025
    Parsed String to LocalDate Object:  2025-03-31

The date string is: "25/12/2023"
The results:    Invalid date format entered

The date string is: "5.3.2023"
The results:    Invalid date format entered

The date string is: "32.12.2023"
The results:    Invalid date format entered

The date string is: "15.13.2023"
The results:    Invalid date format entered

The date string is: "31.02.202"
The results:    Invalid date format entered

The date string is: "opened"
The results:    Invalid date format entered

The date string is: "12.02/2023"
The results:    Invalid date format entered

The date string is: "2023.12.31"
The results:    Invalid date format entered

The date string is: "32.12.2023"
The results:    Invalid date format entered

The date string is: "12.2023"
The results:    Invalid date format entered

The date string is: "abc.12.2024"
The results:    Invalid date format entered

The date string is: "20022024"
The results:    Invalid date format entered

Process finished with exit code 0
```

Рисунок 2.4.1 – Результати роботи програмного коду класу DateValidatorDemo.java

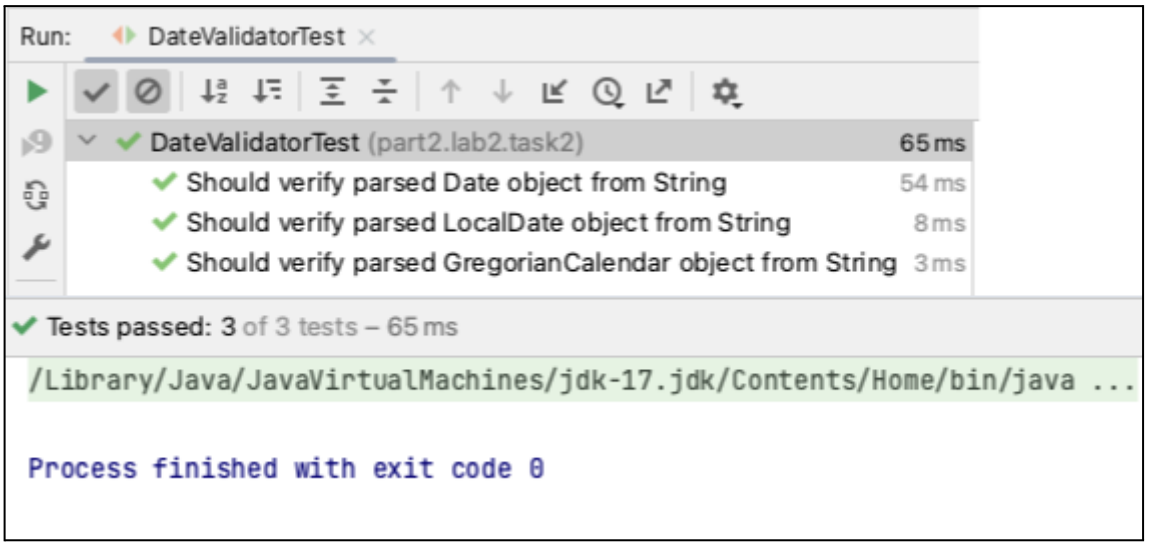

Рисунок 2.4.2 – Результати модульного тестування з використанням Junit програмного коду класу DateValidatorTest.java

3. Завдання №3 до лабораторної роботи

3.1. Перевірка номера телефону

Розробити програму перевірки правильності того, що рядок є номером телефону оператора Київстар. Слід скористатися регулярними виразами.

3.2. Програмний код реалізації завдання № 3

3.2.1. PhoneNumberValidator.java:

```
package part2.lab2.task3;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
* The {@code PhoneNumberParser} class provides utility methods for validating
of phone number strings
* is a number of the Kyivstar telecommunications company.
*/
public class PhoneNumberValidator {
   /** Checks if the phone number is a number of the Kyivstar
telecommunications company. */
   public static boolean verifyIsKyivstarNumber(String phoneNumber) {
       String regex = "^\\+380(67|96|97|98)\\d{7}$";
       Pattern pattern = Pattern.compile(regex);
       Matcher matcher = pattern.matcher(phoneNumber);

       return matcher.matches();
   }
}
```

3.2.2. PhoneNumberValidatorDemo.java:

```
package part2.lab2.task3;

import java.util.stream.IntStream;

/**
* The {@code PhoneNumberValidatorDemo} class is a demonstration of
functionality of the {@link PhoneNumberValidator}.
*/
public class PhoneNumberValidatorDemo {
   /** The constant array representing the phone number strings with valid and
invalid data and phone number formats used in the tests. */
   public static final String[] PHONE_NUMBER_STRINGS = {
           "+380671234567",
           "+380967654321",
           "+380979876543",
           "+380981112233",
           "+380661234567",
           " 0957654321 ",
           "0999876543",
           "1234567890",
           "06712345",
           "opened",
           "+38 (067) 12 34 567",
           "+38-067-12-34-567"
   };

   /**
    * Performs testing of the functionality of the {@link
PhoneNumberValidator}. The {@code args} are not used.
    * @param args the command-line arguments (not used).
    */
   public static void main(String[] args) {
       IntStream.range(0, PHONE_NUMBER_STRINGS.length).forEach(i -> {
           System.out.println("\nThe phone number is:\t\"" +
PHONE_NUMBER_STRINGS[i] + "\"");

           if
(PhoneNumberValidator.verifyIsKyivstarNumber(PHONE_NUMBER_STRINGS[i])) {
               System.out.println("The phone number belongs to Kyivstar
operator.");
           } else {
               System.out.println("The phone number does NOT belong to
Kyivstar operator.");
           }
       });
   }
}
```

3.3. Програмний код модульного тестування з використанням Junit завдання № 3

3.3.1. PhoneNumberValidatorTest.java:

```
package part2.lab2.task3;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import java.util.stream.IntStream;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static part2.lab2.task3.PhoneNumberValidatorDemo.PHONE_NUMBER_STRINGS;

/**
* The {@code PhoneNumberValidatorTest} class provides unit tests for the
{@link PhoneNumberValidator} class.
* It tests the validating of phone number strings is a number of the Kyivstar
telecommunications company.
*/
class PhoneNumberValidatorTest {
   /** Tests the validating of phone number strings is a number of the
Kyivstar telecommunications company
    * by {@link PhoneNumberValidator#verifyIsKyivstarNumber(String)} method.
    */
   @Test
   @DisplayName("Should verify phone number strings is Kyivstar number")
   public void testVerifyIsKyivstarNumber() {
       IntStream.range(0, PHONE_NUMBER_STRINGS.length)
               .forEach(i -> {
                   if (i < 4) {

assertTrue(PhoneNumberValidator.verifyIsKyivstarNumber(PHONE_NUMBER_STRINGS[i]
));
                   } else {

assertFalse(PhoneNumberValidator.verifyIsKyivstarNumber(PHONE_NUMBER_STRINGS[i
]));
                   }
               });
   }
}
```

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

```
The phone number is:    "+380671234567"
The phone number belongs to Kyivstar operator.

The phone number is:    "+380967654321"
The phone number belongs to Kyivstar operator.

The phone number is:    "+380979876543"
The phone number belongs to Kyivstar operator.

The phone number is:    "+380981112233"
The phone number belongs to Kyivstar operator.

The phone number is:    "+380661234567"
The phone number does NOT belong to Kyivstar operator.

The phone number is:    " 0957654321 "
The phone number does NOT belong to Kyivstar operator.

The phone number is:    "0999876543"
The phone number does NOT belong to Kyivstar operator.

The phone number is:    "1234567890"
The phone number does NOT belong to Kyivstar operator.

The phone number is:    "06712345"
The phone number does NOT belong to Kyivstar operator.

The phone number is:    "opened"
The phone number does NOT belong to Kyivstar operator.

The phone number is:    "+38 (067) 12 34 567"
The phone number does NOT belong to Kyivstar operator.

The phone number is:    "+38-067-12-34-567"
The phone number does NOT belong to Kyivstar operator.

Process finished with exit code 0
```

Рисунок 3.4.1 – Результати роботи програмного коду класу PhoneNumberValidatorDemo.java

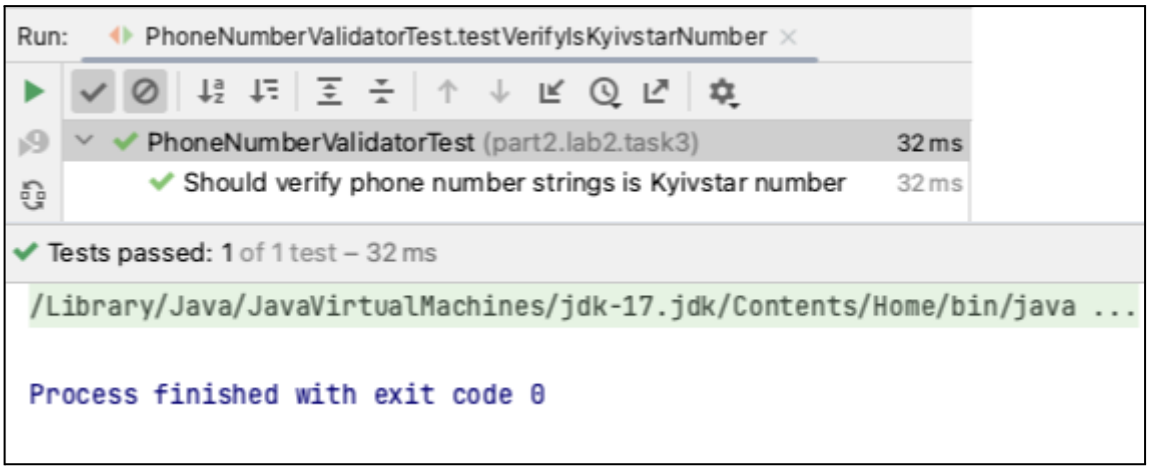

Рисунок 3.4.2 – Результати модульного тестування з використанням Junit програмного коду класу PhoneNumberValidatorTest.java

4. Завдання №4 до лабораторної роботи

4.1. Перевірка рядка пароля

Розробити програму перевірки відповідності пароля вимогам:

- пароль може містити літери латинського алфавіту, цифри та спеціальні символи: _ - *;
- має бути мінімум одна маленька літера;
- має бути мінімум одна велика літера;
- має бути мінімум одна цифра;
- має бути мінімум один спеціальний символ.

Слід скористатися регулярними виразами.

4.2. Програмний код реалізації завдання № 4

4.2.1. PasswordValidator.java:

```
package part2.lab2.task4;

import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

/**
* The {@code PasswordParser} class provides utility methods for
validating of password strings by certain rules
* and printing results of validation.
```

```
*/
public class PasswordValidator {
   /** The array of descriptions of the rules of password validation.
*/
   public static final String[] RULES = {
           "\t- there must be at least one lowercase letter",
           "\t- there must be at least one capital letter",
           "\t- there must be at least one digit",
           "\t- there must be at least one special character: _ - *",
           "\t- there must be at least 8 characters long"
   };

   /** The array of regular expressions corresponding to the rules to
be checked. */
   public static final String[] REGEXES = {
           ".*[a-z].*",
           ".*[A-Z].*",
           ".*\\d.*",
           ".*[_\\-*].*",
           ".{8,}"
   };

   /**
    * Verifies whether the password matches all the {@link
PasswordValidator#RULES} using a regular expression.
    * @param password the password string to be validated;
    * @return {@code true} if the password matches all the rules,
{@code false} otherwise.
    */
   public static boolean verifyPasswordMatchingAllRules(String
password) {
       String regex =
"^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[_\\-*]).{8,}$";
       Pattern pattern = Pattern.compile(regex);
       Matcher matcher = pattern.matcher(password);

       return matcher.matches();
   }

   /**
    * Retrieves descriptions of rules that the password does not match.
    * @param password the password string to be validated;
    * @return a list of descriptions of rules that the password does
not match.
    */
   public static List<String>
getPasswordMismatchRulesDescriptions(String password) {
       return IntStream.range(0, REGEXES.length)
               .filter(i -> !password.matches(REGEXES[i]))
```

```
                .mapToObj(i -> RULES[i])
                .collect(Collectors.toList());
    }

    /**
     * Gets the indexes of the rules of the {@link
PasswordValidator#RULES} array that the password does not match.
     * @param password the password string to be validated;
     * @return an array of indexes of rules that the password does not
match.
     */
    public static int[] getPasswordMismatchRulesIndexes(String password)
{
        return IntStream.range(0, PasswordValidator.REGEXES.length)
                .filter(i ->
!password.matches(PasswordValidator.REGEXES[i]))
                .toArray();
    }

    /**
     * Prints the result of password verification by certain rules.
     * @param password The password string to be validated.
     */
    public static void printPasswordVerificationResult(String password)
{
        if (verifyPasswordMatchingAllRules(password)) {
            System.out.println("The password matches all the rules.");
        } else {
            System.out.println("The password does not match the
following rules:");

getPasswordMismatchRulesDescriptions(password).forEach(System.out::prin
tln);
        }
    }
}
```

4.2.2. PasswordValidatorDemo.java:

```
package part2.lab2.task4;

import java.util.stream.IntStream;

/**
* The {@code PasswordValidatorDemo} class is a demonstration of
functionality of the {@link PasswordValidator} class.
*/
public class PasswordValidatorDemo {
```

```
   /** The constant array representing the password strings with valid
and invalid data used in the tests. */
   public static final String[] PASSWORD_STRINGS = {
           "Password123*",
           "qweRTY123_",
           "LongPassword123_-*",
           "Passw123* ",
           "",
           "PASSWORD123*",
           "password123*",
           "Password*",
           "Password123",
           "Pass1*",
           "PASSWORD123",
           "password_",
           "12345678",
           "password",
           "short",

   };

   /**
    * Performs testing of the functionality of the {@link
PasswordValidator}. The {@code args} are not used.
    * @param args the command-line arguments (not used).
    */
   public static void main(String[] args) {
       IntStream.range(0, PASSWORD_STRINGS.length)
               .forEach(i -> {
                   System.out.println("\nThe password is:\t\"" +
PASSWORD_STRINGS[i] + "\"");

PasswordValidator.printPasswordVerificationResult(PASSWORD_STRINGS[i]);
               });
   }
}
```

4.3. Програмний код модульного тестування з використанням Junit завдання № 4

4.3.1. PasswordValidatorTest.java:

```
package part2.lab2.task4;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import java.util.List;
```

```
import java.util.stream.Collectors;
import java.util.stream.IntStream;

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

/**
* The {@code PasswordValidatorTest} class provides unit tests for the
{@link PasswordValidator} class.
* It tests the validating of password strings by certain rules.
*/
class PasswordValidatorTest {
    /**
     * The constant array representing the indexes of the rules of the
{@link PasswordValidator#RULES} array that
     * the password does not match in {@link
PasswordValidatorDemo#PASSWORD_STRINGS} test data.
     */
    public static final int[][] RULES_INDEXES = {
            new int[] {0, 1, 2, 3, 4},
            new int[] {0},
            new int[] {1},
            new int[] {2},
            new int[] {3},
            new int[] {4},
            new int[] {0, 3},
            new int[] {1, 2},
            new int[] {0, 1, 3},
            new int[] {1, 2, 3},
            new int[] {1, 2, 3, 4}
    };

    /**
     * Generates indexed descriptions based on the provided {@link
PasswordValidator#RULES} rules
     * and their corresponding indexes at {@link
PasswordValidatorTest#RULES_INDEXES}.
     * @param rules an array of rule descriptions;
     * @param rulesIndexes a two-dimensional array where each inner
array contains indexes of rules for a specific category.
     * @return an array of lists of rule descriptions indexed according
to the provided indexes.
     */
    public static List<String>[] getIndexedDescriptions(String[] rules,
int[][] rulesIndexes) {
        List<String>[] descriptions = new List[rulesIndexes.length];

        IntStream.range(0, rulesIndexes.length).forEach(i -> {
            descriptions[i] = IntStream.of(rulesIndexes[i])
                    .filter(ruleIndex -> ruleIndex >= 0 && ruleIndex <
```

```
rules.length)
                    .mapToObj(ruleIndex -> rules[ruleIndex])
                    .collect(Collectors.toList());
        });

        return descriptions;
    }


    /** Tests the validating of password strings by all certain rules.
     * by {@link
PasswordValidator#verifyPasswordMatchingAllRules(String)} method
     */
    @Test
    @DisplayName("Should verify password by all rules")
    public void verifyPasswordMatchingAllRules() {
        IntStream.range(0, 4)
                .forEach(i -> {
                    assertTrue(

PasswordValidator.verifyPasswordMatchingAllRules(

PasswordValidatorDemo.PASSWORD_STRINGS[i]
                            )
                    );
                });
    }

    /** Tests getting the indexes of the rules of the {@link
PasswordValidator#RULES} array that
     * the password does not match by {@link
PasswordValidator#getPasswordMismatchRulesIndexes(String)} method.
     */
    @Test
    @DisplayName("Should verify getting indexes of the rules that the
password does not match")
    public void getPasswordMismatchRulesIndexes() {
        IntStream.range(4,
PasswordValidatorDemo.PASSWORD_STRINGS.length)
                .forEach(i -> {
                    assertArrayEquals(
                            RULES_INDEXES[i - 4],

PasswordValidator.getPasswordMismatchRulesIndexes(

PasswordValidatorDemo.PASSWORD_STRINGS[i]
                            )
                    );
                });
```

```
    }

    /** Tests getting the descriptions of the rules of the {@link
PasswordValidator#RULES} array that
     * the password does not match by {@link
PasswordValidator#getPasswordMismatchRulesDescriptions(String)} method.
     */
    @Test
    @DisplayName("Should verify getting indexes of the rules that the
password does not match")
    public void getPasswordMismatchRulesDescriptions() {
        IntStream.range(4,
PasswordValidatorDemo.PASSWORD_STRINGS.length)
                .forEach(i -> {
                    assertEquals(

getIndexedDescriptions(PasswordValidator.RULES, RULES_INDEXES)[i - 4],

PasswordValidator.getPasswordMismatchRulesDescriptions(PasswordValidato
rDemo.PASSWORD_STRINGS[i])
                    );
                });
    }
}
```

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

```
The password is:    "Password123*"
The password matches all the rules.

The password is:    "qweRTY123_"
The password matches all the rules.

The password is:    "LongPassword123_-*"
The password matches all the rules.

The password is:    "Passw123* "
The password matches all the rules.

The password is:    ""
The password does not match the following rules:
    - there must be at least one lowercase letter
    - there must be at least one capital letter
    - there must be at least one digit
    - there must be at least one special character: _ - *
    - there must be at least 8 characters long

The password is:    "PASSWORD123*"
The password does not match the following rules:
    - there must be at least one lowercase letter

The password is:    "password123*"
The password does not match the following rules:
    - there must be at least one capital letter

The password is:    "Password*"
The password does not match the following rules:
    - there must be at least one digit

The password is:    "Password123"
The password does not match the following rules:
    - there must be at least one special character: _ - *

The password is:    "Pass1*"
The password does not match the following rules:
    - there must be at least 8 characters long
```

Рисунок 4.4.1 – Результати № 1 роботи програмного коду класу PasswordValidatorDemo.java

```
The password is:    "PASSWORD123"
The password does not match the following rules:
    - there must be at least one lowercase letter
    - there must be at least one special character: _ - *

The password is:    "password_"
The password does not match the following rules:
    - there must be at least one capital letter
    - there must be at least one digit

The password is:    "12345678"
The password does not match the following rules:
    - there must be at least one lowercase letter
    - there must be at least one capital letter
    - there must be at least one special character: _ - *

The password is:    "password"
The password does not match the following rules:
    - there must be at least one capital letter
    - there must be at least one digit
    - there must be at least one special character: _ - *

The password is:    "short"
The password does not match the following rules:
    - there must be at least one capital letter
    - there must be at least one digit
    - there must be at least one special character: _ - *
    - there must be at least 8 characters long

Process finished with exit code 0
```

Рисунок 4.4.2 – Результати № 2 роботи програмного коду класу PasswordValidatorDemo.java

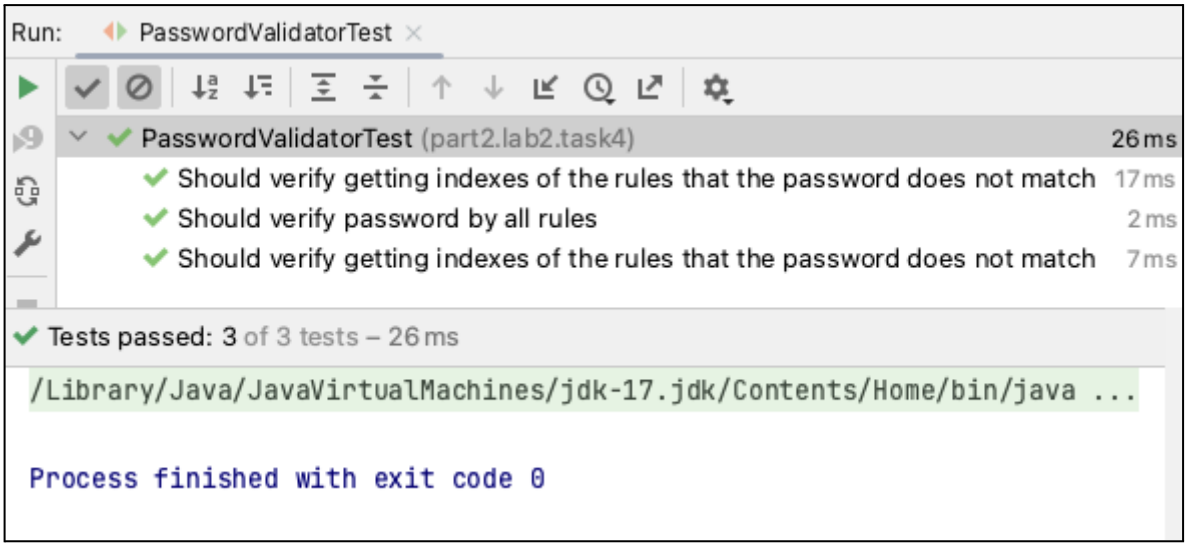

Рисунок 4.4.3 – Результати модульного тестування з використанням Junit програмного коду класу PasswordValidatorTest.java

5. Завдання №5 до лабораторної роботи

5.1. Отримання масиву підрядків (додаткове завдання)

Рядок довжиною понад 20 символів містить літери та цифри. Отримати з цього рядка масив підрядків, які містять літери між цифрами (групами цифр), визначити цифри як розділювачі.

5.2. Програмний код реалізації вправи для контролю

5.2.1. SubstringsParser.java:

```
package part2.lab2.task5;

import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.stream.Collectors;

/**
* The {@code SubstringsParser} class provides methods for extracting
substrings from a given input string.
*/
public class SubstringsParser {
   /**
    * Extracts substrings from the input string that are enclosed
between digits.
    * @param input the input string from which substrings are to be
extracted;
    * @return a list of substrings extracted from the input string;
```

```
    * @throws NullPointerException if the input string is {@code null}.
    */
   public static List<String> getSubstrings(String input) throws
NullPointerException {
       String regex = "(?<=\\d)([a-zA-Z]+)(?=\\d)";
       Pattern pattern = Pattern.compile(regex);
       Matcher matcher = pattern.matcher(input);

       return matcher.results()
               .map(matchResult -> matchResult.group())
               .collect(Collectors.toList());
   }


   /**
    * Prints the list of substrings.
    * @param substringsList the list of substrings to be printed.
    */
   public static void printSubstringsList(List<String> substringsList)
{
       if (substringsList.isEmpty()) {
           System.out.print("Missing substrings");
       } else {
           substringsList.forEach(substring -> System.out.print("\"" +
substring + "\"\t"));
       }
   }
}
```

5.2.2. SubstringsParserDemo.java:

```
package part2.lab2.task5;

import java.util.Arrays;
import java.util.List;

/**
* The {@code SubstringsParserDemo} class demonstrates the usage of the
{@link SubstringsParser} class
* by generating valid and invalid input strings and displaying the
resulting substrings.
*/
public class SubstringsParserDemo {
   /**
    * Generates a list of valid input strings containing letters
enclosed between digits.
    * @return a list of valid input strings.
    */
```

```
    public static List<String> createValidStrings() {
        return Arrays.asList(
                "123abc456def789",
                "1a2b3c4d5e6f7g8",
                "12a34b56c78d",
                "1abc2",
                "a1b2c3d4e5f6g7",
                "ab123cd456ef"
        );
    }

    /**
     * Generates a list of invalid input strings without letters
enclosed between digits.
     * @return a list of invalid input strings.
     */
    public static List<String> createInvalidStrings() {
        return Arrays.asList(
                "123456789",
                "abcdef",
                "1234abcdef",
                "abcdef1234",
                "  123 kjf 987 ",
                "!@#$%^&*()!@#$%^&*(",
                ""
        );
    }

    /**
     * The main method generates input strings, extracts substrings, and
displays the results.
     * @param args command-line arguments (not used).
     */
    public static void main(String[] args) {
        System.out.print("The resulting substrings of strings:");
        try {
            createValidStrings().forEach(input -> {
                System.out.print("\n\nThe string:\t\"" + input + "\"\n"
                        + "The substrings:\t");
                SubstringsParser.printSubstringsList(
                        SubstringsParser.getSubstrings(input)
                );
            });

            createInvalidStrings().forEach(input -> {
                System.out.print("\n\nThe string:\t\"" + input + "\"\n"
                        + "The substrings:\t");
                SubstringsParser.printSubstringsList(
                        SubstringsParser.getSubstrings(input)
```

```
            );
        });

        System.out.print("\n\nThe string:\t\"" + null + "\"\n"
                + "The substrings:\t");
        SubstringsParser.printSubstringsList(
                SubstringsParser.getSubstrings(null)
        );
    } catch (NullPointerException e) {
        System.err.println(e.getMessage());
    }
  }
}
```

5.3. Програмний код модульного тестування з використанням Junit вправи для контролю

5.3.1. SubstringsParserTest.java:

```
package part2.lab2.task5;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.IntStream;
import java.util.stream.Stream;

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

/**
* The {@code SubstringsParserTest} class contains unit tests for the
{@link SubstringsParser} class.
* It tests the validating of getting a substrings from a strings and
outputting results.
*/
class SubstringsParserTest {
   /**
    * Creates an array of valid substrings for testing purposes.
    * @return an array of valid substrings.
    */
   public static String[][] createValidSubstrings() {
       return Stream.of(
               Arrays.asList("abc", "def"),
               Arrays.asList("a", "b", "c", "d", "e", "f", "g"),
               Arrays.asList("a", "b", "c"),
```

```
                Arrays.asList("abc"),
                Arrays.asList("b", "c", "d", "e", "f", "g"),
                Arrays.asList("cd"))
                .map(list -> list.toArray(String[]::new))
                .toArray(String[][]::new);
    }

    /**
     * Tests the {@link SubstringsParser#getSubstrings(String)} method
for valid input strings.
     */
    @Test
    @DisplayName("Should verify getting substrings from valid strings")
    public void getSubstringsFromValidStrings() {
        IntStream.range(0,
SubstringsParserDemo.createValidStrings().size())
                .forEach(i -> {
                    assertArrayEquals(
                            SubstringsParser.getSubstrings(

SubstringsParserDemo.createValidStrings()
                                            .toArray(String[]::new)[i])
                                    .toArray(),

Arrays.stream(createValidSubstrings()[i]).toArray()
                    );
                });
    }

    /**
     * Tests the {@link SubstringsParser#getSubstrings(String)} method
for invalid input strings.
     */
    @Test
    @DisplayName("Should verify getting substrings from invalid
strings")
    public void getSubstringsFromInvalidStrings() {
        IntStream.range(0,
SubstringsParserDemo.createInvalidStrings().size())
                .forEach(i -> {
                    assertTrue(
                            SubstringsParser.getSubstrings(

SubstringsParserDemo.createInvalidStrings()
                                            .toArray(String[]::new)[i])
                                    .isEmpty()
                    );
                });
    }
```

```
    /**
     * Tests the {@link SubstringsParser#getSubstrings(String)} method
with null input (without string).
     */
    @Test
    @DisplayName("Should verify getting substrings from null (without
string)")
    public void getSubstringsWithNull() {
        assertThrows(NullPointerException.class, () -> {
            SubstringsParser.getSubstrings(null);
        });
    }

    /**
     * Tests the {@link SubstringsParser#printSubstringsList(List)}
method with valid input strings.
     */
    @Test
    @DisplayName("Should verify output of substrings from valid
strings")
    public void printSubstringsListWithValidStrings() {
        IntStream.range(0,
SubstringsParserDemo.createValidStrings().size())
                .forEach(i -> {
                    assertEquals(
                            SubstringsParser.getSubstrings(

SubstringsParserDemo.createValidStrings()

.toArray(String[]::new)[i])
                                    .toString(),
                            Arrays.toString(createValidSubstrings()[i])
                    );
                });
    }

    /**
     * Tests the {@link SubstringsParser#printSubstringsList(List)}
method with invalid input strings.
     */
    @Test
    @DisplayName("Should verify output of substrings from invalid
strings")
    public void printSubstringsListWithInvalidStrings() {
        IntStream.range(0,
SubstringsParserDemo.createInvalidStrings().size())
                .forEach(i -> {
                    assertEquals(
```

```
                            SubstringsParser.getSubstrings(

SubstringsParserDemo.createInvalidStrings()

.toArray(String[]::new)[i])
                                        .toString(),
                                new ArrayList<String>().toString()
                    );
                });
    }
}
```

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

```
The resulting substrings of strings:

The string: "123abc456def789"
The substrings: "abc"   "def"

The string: "1a2b3c4d5e6f7g8"
The substrings: "a" "b" "c" "d" "e" "f" "g"

The string: "12a34b56c78d"
The substrings: "a" "b" "c"

The string: "1abc2"
The substrings: "abc"

The string: "a1b2c3d4e5f6g7"
The substrings: "b" "c" "d" "e" "f" "g"

The string: "ab123cd456ef"
The substrings: "cd"

The string: "123456789"
The substrings: Missing substrings

The string: "abcdef"
The substrings: Missing substrings

The string: "1234abcdef"
The substrings: Missing substrings

The string: "abcdef1234"
The substrings: Missing substrings

The string: "  123 kjf 987 "
The substrings: Missing substrings

The string: "!@#$%^&*()!@#$%^&*("
The substrings: Missing substrings

The string: ""
The substrings: Missing substrings

The string: "null"
The substrings: Cannot invoke "java.lang.CharSequence.length()" because "this.text" is null

Process finished with exit code 0
```

Рисунок 5.4.1 – Результати роботи програмного коду класу SubstringsParserDemo.java

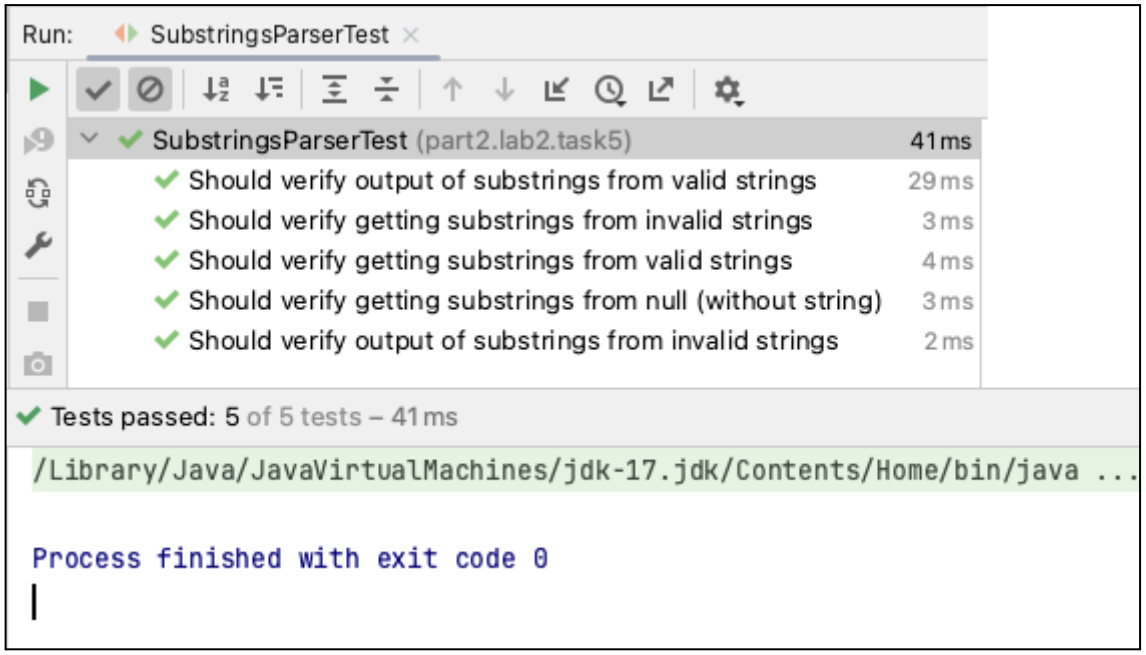

Рисунок 5.4.2 – Результати модульного тестування з використанням Junit програмного коду класу SubstringsParserTest.java

6. Вправа для контролю до лабораторної роботи

6.1. Умова завдання

Вивести на екран інформацію про всі часові пояси.

6.2. Програмний код реалізації вправи для контролю

6.2.1. TimeZonesPrinter.java:

```
package part2.lab2.task_control;

import java.util.Arrays;
import java.util.List;
import java.util.TimeZone;

/**
* TimeZonesPrinter class provides functionality to retrieve and print
the list of available time zones.
*/
public class TimeZonesPrinter {
   /**
    * Retrieves the list of available time zones.
    * @return A list of available time zone IDs.
    */
   public static List<String> getTimeZones() {
       return Arrays.stream(TimeZone.getAvailableIDs()).toList();
```

```
    }

    /**
     * Main method to print the list of available time zones.
     * @param args Command-line arguments (not used).
     */
    public static void main(String[] args) {
        System.out.println("The list of available time zones:");
        getTimeZones().forEach(System.out::println);
    }
}
```

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

```
The list of available time zones:
Africa/Abidjan                  Africa/Ndjamena                 America/Cancun                  America/Indiana/Vevay
Africa/Accra                    Africa/Niamey                   America/Caracas                 America/Indiana/Vincennes
Africa/Addis_Ababa              Africa/Nouakchott               America/Catamarca               America/Indiana/Winamac
Africa/Algiers                  Africa/Ouagadougou              America/Cayenne                 America/Indianapolis
Africa/Asmara                   Africa/Porto-Novo               America/Cayman                  America/Inuvik
Africa/Asmera                   Africa/Sao_Tome                 America/Chicago                 America/Iqaluit
Africa/Bamako                   Africa/Timbuktu                 America/Chihuahua               America/Jamaica
Africa/Bangui                   Africa/Tripoli                  America/Ciudad_Juarez           America/Jujuy
Africa/Banjul                   Africa/Tunis                    America/Coral_Harbour           America/Juneau
Africa/Bissau                   Africa/Windhoek                 America/Cordoba                 America/Kentucky/Louisville
Africa/Blantyre                 America/Adak                    America/Costa_Rica              America/Kentucky/Monticello
Africa/Brazzaville              America/Anchorage               America/Creston                 America/Knox_IN
Africa/Bujumbura                America/Anguilla                America/Cuiaba                  America/Kralendijk
Africa/Cairo                    America/Antigua                 America/Curacao                 America/La_Paz
Africa/Casablanca               America/Araguaina               America/Danmarkshavn            America/Lima
Africa/Ceuta                    America/Argentina/Buenos_Aires  America/Dawson                  America/Los_Angeles
Africa/Conakry                  America/Argentina/Catamarca     America/Dawson_Creek            America/Louisville
Africa/Dakar                    America/Argentina/ComodRivadavia America/Denver                 America/Lower_Princes
Africa/Dar_es_Salaam            America/Argentina/Cordoba       America/Detroit                 America/Maceio
Africa/Djibouti                 America/Argentina/Jujuy         America/Dominica                America/Managua
Africa/Douala                   America/Argentina/La_Rioja      America/Edmonton                America/Manaus
Africa/El_Aaiun                 America/Argentina/Mendoza       America/Eirunepe                America/Marigot
Africa/Freetown                 America/Argentina/Rio_Gallegos  America/El_Salvador             America/Martinique
Africa/Gaborone                 America/Argentina/Salta         America/Ensenada                America/Matamoros
Africa/Harare                   America/Argentina/San_Juan      America/Fort_Nelson             America/Mazatlan
Africa/Johannesburg             America/Argentina/San_Luis      America/Fort_Wayne              America/Mendoza
Africa/Juba                     America/Argentina/Tucuman       America/Fortaleza               America/Menominee
Africa/Kampala                  America/Argentina/Ushuaia       America/Glace_Bay               America/Merida
Africa/Khartoum                 America/Aruba                   America/Godthab                 America/Metlakatla
Africa/Kigali                   America/Asuncion                America/Goose_Bay               America/Mexico_City
Africa/Kinshasa                 America/Atikokan                America/Grand_Turk              America/Miquelon
Africa/Lagos                    America/Atka                    America/Grenada                 America/Moncton
Africa/Libreville               America/Bahia                   America/Guadeloupe              America/Monterrey
Africa/Lome                     America/Bahia_Banderas          America/Guatemala               America/Montevideo
Africa/Luanda                   America/Barbados                America/Guayaquil               America/Montreal
Africa/Lubumbashi               America/Belem                   America/Guyana                  America/Montserrat
Africa/Lusaka                   America/Belize                  America/Halifax                 America/Nassau
Africa/Malabo                   America/Blanc-Sablon            America/Havana                  America/New_York
Africa/Maputo                   America/Boa_Vista               America/Hermosillo              America/Nipigon
Africa/Maseru                   America/Bogota                  America/Indiana/Indianapolis    America/Nome
Africa/Mbabane                  America/Boise                   America/Indiana/Knox            America/Noronha
Africa/Mogadishu                America/Buenos_Aires            America/Indiana/Marengo         America/North_Dakota/Beulah
Africa/Monrovia                 America/Cambridge_Bay           America/Indiana/Petersburg      America/North_Dakota/Center
Africa/Nairobi                  America/Campo_Grande            America/Indiana/Tell_City       America/North_Dakota/New_Salem
```

Рисунок 6.3.1 – Результати № 1 роботи програмного коду класу TimeZonesPrinter.java

```
America/Nuuk                Antarctica/Casey            Asia/Ho_Chi_Minh        Asia/Taipei
America/Ojinaga             Antarctica/Davis            Asia/Hong_Kong          Asia/Tashkent
America/Panama              Antarctica/DumontDUrville   Asia/Hovd              Asia/Tbilisi
America/Pangnirtung         Antarctica/Macquarie        Asia/Irkutsk           Asia/Tehran
America/Paramaribo          Antarctica/Mawson           Asia/Istanbul          Asia/Tel_Aviv
America/Phoenix             Antarctica/McMurdo          Asia/Jakarta           Asia/Thimbu
America/Port-au-Prince      Antarctica/Palmer           Asia/Jayapura          Asia/Thimphu
America/Port_of_Spain       Antarctica/Rothera          Asia/Jerusalem         Asia/Tokyo
America/Porto_Acre          Antarctica/South_Pole       Asia/Kabul             Asia/Tomsk
America/Porto_Velho         Antarctica/Syowa            Asia/Kamchatka         Asia/Ujung_Pandang
America/Puerto_Rico         Antarctica/Troll            Asia/Karachi           Asia/Ulaanbaatar
America/Punta_Arenas        Antarctica/Vostok           Asia/Kashgar           Asia/Ulan_Bator
America/Rainy_River         Arctic/Longyearbyen         Asia/Kathmandu         Asia/Urumqi
America/Rankin_Inlet        Asia/Aden                   Asia/Katmandu          Asia/Ust-Nera
America/Recife              Asia/Almaty                 Asia/Khandyga          Asia/Vientiane
America/Regina              Asia/Amman                  Asia/Kolkata           Asia/Vladivostok
America/Resolute            Asia/Anadyr                 Asia/Krasnoyarsk       Asia/Yakutsk
America/Rio_Branco          Asia/Aqtau                  Asia/Kuala_Lumpur      Asia/Yangon
America/Rosario             Asia/Aqtobe                 Asia/Kuching           Asia/Yekaterinburg
America/Santa_Isabel        Asia/Ashgabat               Asia/Kuwait            Asia/Yerevan
America/Santarem            Asia/Ashkhabad              Asia/Macao             Atlantic/Azores
America/Santiago            Asia/Atyrau                 Asia/Macau             Atlantic/Bermuda
America/Santo_Domingo       Asia/Baghdad                Asia/Magadan           Atlantic/Canary
America/Sao_Paulo           Asia/Bahrain                Asia/Makassar          Atlantic/Cape_Verde
America/Scoresbysund        Asia/Baku                   Asia/Manila            Atlantic/Faeroe
America/Shiprock            Asia/Bangkok                Asia/Muscat            Atlantic/Faroe
America/Sitka               Asia/Barnaul                Asia/Nicosia           Atlantic/Jan_Mayen
America/St_Barthelemy       Asia/Beirut                 Asia/Novokuznetsk      Atlantic/Madeira
America/St_Johns            Asia/Bishkek                Asia/Novosibirsk       Atlantic/Reykjavik
America/St_Kitts            Asia/Brunei                 Asia/Omsk              Atlantic/South_Georgia
America/St_Lucia            Asia/Calcutta               Asia/Oral              Atlantic/St_Helena
America/St_Thomas           Asia/Chita                  Asia/Phnom_Penh        Atlantic/Stanley
America/St_Vincent          Asia/Choibalsan             Asia/Pontianak         Australia/ACT
America/Swift_Current       Asia/Chongqing              Asia/Pyongyang         Australia/Adelaide
America/Tegucigalpa         Asia/Chungking              Asia/Qatar             Australia/Brisbane
America/Thule               Asia/Colombo                Asia/Qostanay          Australia/Broken_Hill
America/Thunder_Bay         Asia/Dacca                  Asia/Qyzylorda         Australia/Canberra
America/Tijuana             Asia/Damascus               Asia/Rangoon           Australia/Currie
America/Toronto             Asia/Dhaka                  Asia/Riyadh            Australia/Darwin
America/Tortola             Asia/Dili                   Asia/Saigon            Australia/Eucla
America/Vancouver           Asia/Dubai                  Asia/Sakhalin          Australia/Hobart
America/Virgin              Asia/Dushanbe               Asia/Samarkand         Australia/LHI
America/Whitehorse          Asia/Famagusta              Asia/Seoul             Australia/Lindeman
America/Winnipeg            Asia/Gaza                   Asia/Shanghai          Australia/Lord_Howe
America/Yakutat             Asia/Harbin                 Asia/Singapore         Australia/Melbourne
America/Yellowknife         Asia/Hebron                 Asia/Srednekolymsk     Australia/NSW
```

Рисунок 6.3.2 – Результати № 2 роботи програмного коду класу TimeZonesPrinter.java

```
Australia/North          Etc/GMT-10          Europe/London          Indian/Christmas
Australia/Perth          Etc/GMT-11          Europe/Luxembourg      Indian/Cocos
Australia/Queensland     Etc/GMT-12          Europe/Madrid          Indian/Comoro
Australia/South          Etc/GMT-13          Europe/Malta           Indian/Kerguelen
Australia/Sydney         Etc/GMT-14          Europe/Mariehamn       Indian/Mahe
Australia/Tasmania       Etc/GMT-2           Europe/Minsk           Indian/Maldives
Australia/Victoria       Etc/GMT-3           Europe/Monaco          Indian/Mauritius
Australia/West           Etc/GMT-4           Europe/Moscow          Indian/Mayotte
Australia/Yancowinna     Etc/GMT-5           Europe/Nicosia         Indian/Reunion
Brazil/Acre              Etc/GMT-6           Europe/Oslo            Iran
Brazil/DeNoronha         Etc/GMT-7           Europe/Paris           Israel
Brazil/East              Etc/GMT-8           Europe/Podgorica       Jamaica
Brazil/West              Etc/GMT-9           Europe/Prague          Japan
CET                      Etc/GMT0            Europe/Riga            Kwajalein
CST6CDT                  Etc/Greenwich       Europe/Rome            Libya
Canada/Atlantic          Etc/UCT             Europe/Samara          MET
Canada/Central           Etc/UTC             Europe/San_Marino      MST7MDT
Canada/Eastern           Etc/Universal       Europe/Sarajevo        Mexico/BajaNorte
Canada/Mountain          Etc/Zulu            Europe/Saratov         Mexico/BajaSur
Canada/Newfoundland      Europe/Amsterdam    Europe/Simferopol      Mexico/General
Canada/Pacific           Europe/Andorra      Europe/Skopje          NZ
Canada/Saskatchewan      Europe/Astrakhan    Europe/Sofia           NZ-CHAT
Canada/Yukon             Europe/Athens       Europe/Stockholm       Navajo
Chile/Continental        Europe/Belfast      Europe/Tallinn         PRC
Chile/EasterIsland       Europe/Belgrade     Europe/Tirane          PST8PDT
Cuba                     Europe/Berlin       Europe/Tiraspol        Pacific/Apia
EET                      Europe/Bratislava   Europe/Ulyanovsk       Pacific/Auckland
EST5EDT                  Europe/Brussels     Europe/Uzhgorod        Pacific/Bougainville
Egypt                    Europe/Bucharest    Europe/Vaduz           Pacific/Chatham
Eire                     Europe/Budapest     Europe/Vatican         Pacific/Chuuk
Etc/GMT                  Europe/Busingen     Europe/Vienna          Pacific/Easter
Etc/GMT+0                Europe/Chisinau     Europe/Vilnius         Pacific/Efate
Etc/GMT+1                Europe/Copenhagen   Europe/Volgograd       Pacific/Enderbury
Etc/GMT+10               Europe/Dublin       Europe/Warsaw          Pacific/Fakaofo
Etc/GMT+11               Europe/Gibraltar    Europe/Zagreb          Pacific/Fiji
Etc/GMT+12               Europe/Guernsey     Europe/Zaporozhye      Pacific/Funafuti
Etc/GMT+2                Europe/Helsinki     Europe/Zurich          Pacific/Galapagos
Etc/GMT+3                Europe/Isle_of_Man  GB                     Pacific/Gambier
Etc/GMT+4                Europe/Istanbul     GB-Eire                Pacific/Guadalcanal
Etc/GMT+5                Europe/Jersey       GMT                    Pacific/Guam
Etc/GMT+6                Europe/Kaliningrad  GMT0                   Pacific/Honolulu
Etc/GMT+7                Europe/Kiev         Greenwich              Pacific/Johnston
Etc/GMT+8                Europe/Kirov        Hongkong               Pacific/Kanton
Etc/GMT+9                Europe/Kyiv         Iceland                Pacific/Kiritimati
Etc/GMT-0                Europe/Lisbon       Indian/Antananarivo    Pacific/Kosrae
Etc/GMT-1                Europe/Ljubljana    Indian/Chagos          Pacific/Kwajalein
```

Рисунок 6.3.3 – Результати № 3 роботи програмного коду класу TimeZonesPrinter.java

```
Pacific/Majuro          US/East-Indiana
Pacific/Marquesas       US/Eastern
Pacific/Midway          US/Hawaii
Pacific/Nauru           US/Indiana-Starke
Pacific/Niue            US/Michigan
Pacific/Norfolk         US/Mountain
Pacific/Noumea          US/Pacific
Pacific/Pago_Pago       US/Samoa
Pacific/Palau           UTC
Pacific/Pitcairn        Universal
Pacific/Pohnpei         W-SU
Pacific/Ponape          WET
Pacific/Port_Moresby    Zulu
Pacific/Rarotonga       EST
Pacific/Saipan          HST
Pacific/Samoa           MST
Pacific/Tahiti          ACT
Pacific/Tarawa          AET
Pacific/Tongatapu       AGT
Pacific/Truk            ART
Pacific/Wake            AST
Pacific/Wallis          BET
Pacific/Yap             BST
Poland                  CAT
Portugal                CNT
ROK                     CST
Singapore               CTT
SystemV/AST4            EAT
SystemV/AST4ADT         ECT
SystemV/CST6            IET
SystemV/CST6CDT         IST
SystemV/EST5            JST
SystemV/EST5EDT         MIT
SystemV/HST10           NET
SystemV/MST7            NST
SystemV/MST7MDT         PLT
SystemV/PST8            PNT
SystemV/PST8PDT         PRT
SystemV/YST9            PST
SystemV/YST9YDT         SST
Turkey                  VST
UCT
US/Alaska               Process finished with exit code 0
US/Aleutian
US/Arizona
US/Central
```

Рисунок 6.3.4 – Результати № 4 роботи програмного коду класу TimeZonesPrinter.java

7. Висновки до лабораторної роботи

Під час виконання лабораторної роботи були опановані різноманітні технології Java, такі як робота з датами та текстом, локалізація, регулярні

вирази та класи пакету java.time для роботи з датою й часом. Використання цих технологій дозволило ефективно реалізувати індивідуальне завдання лабораторної роботи.

Основне завдання передбачало створення класів для представлення сутностей, з реалізацією можливостей підтримки різних локалізацій. Було успішно здійснено переклад тексту, виведення чисел, дат і часу з урахуванням різних локалізацій, а також реалізовано пошук слів у коментарях за допомогою регулярних виразів та сортування сутностей за алфавітом з використанням класу Collator.

Додатково, виконання інших завдань лабораторної роботи дозволило розширити знання про роботу з регулярними виразами для перевірки введених даних, наприклад, перевірка правильності введення дати та номера телефону, а також розробка програми для перевірки відповідності пароля встановленим критеріям.