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

КАФЕДРА «Програмна інженерія та інтелектуальні технології управління»

ЗВІТ
з лабораторної роботи № 1
з навчальної дисципліни
“ПОГЛИБЛЕНИЙ КУРС ПРОГРАМУВАННЯ JAVA”

ВИКОНАЛА
Студентка групи КН-222а
Репешко Інеса Віталіївна

ПЕРЕВІРИВ
асистент кафедри ПІІТУ
Кондратов Олексій Михайлович

Харків 2024

Тема роботи

“Робота з великими числами та наборами даних”

1. Завдання №1 до лабораторної роботи

1.1. Індивідуальне завдання

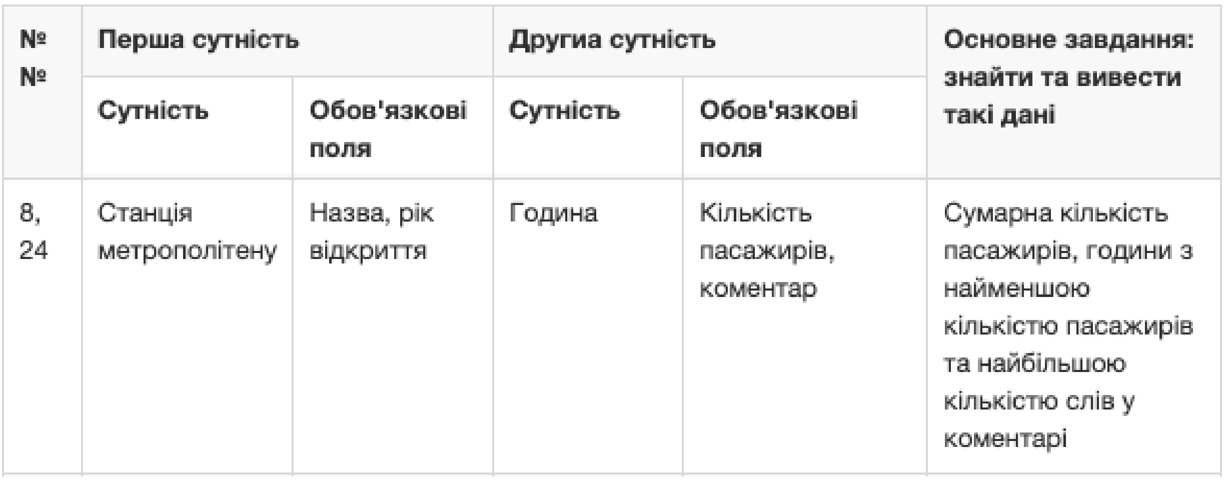
Спроєктувати та реалізувати класи для представлення сутностей третьої лабораторної роботи курсу "Основи програмування Java". Рішення повинне базуватися на раніше створеній ієрархії класів.

Слід створити похідний клас, який представляє основну сутність. Як базовий використати клас, створений у четвертій лабораторній роботі курсу "Основи програмування Java". Слід скористатися класом, який представляє послідовність списком. Створити похідні класи, в яких перевизначити реалізацію всіх методів, пов'язаних з обробкою послідовностей через використання засобів Stream API. Якщо в класі, який представляє другу сутність, немає обробки послідовностей, клас можна не перевизначати й користуватися базовим класом.

Програма повинна демонструвати:

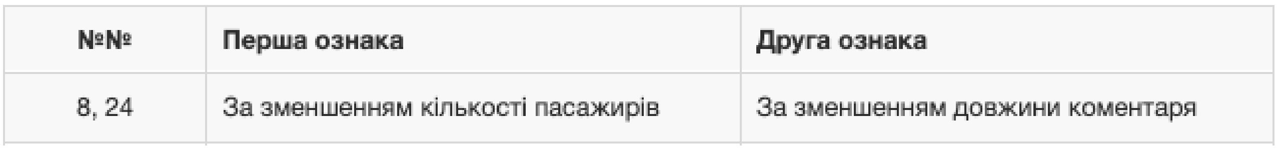
- відтворення функціональності лабораторних робіт № 3 і № 4 курсу "Основи програмування Java";
- використання засобів Stream API для всіх функцій обробки та виведення послідовностей;
- тестування методів окремих класів з використанням JUnit.

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:

```java
package part1.lab4.task1;

import java.util.Arrays;

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

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

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

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

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

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

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

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

       this.ridership = ridership;
   }

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

```java
        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:

```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;
    }
```

```java
   /**
    * 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.
    */
   private 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.
    */
   private 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.
    */
   private 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.
    */
   private 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.
    */
   private 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
MetroStationWithStreams}  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 MetroStationWithStreams} with
predefined values.
     * @return the object of class {@code MetroStationWithStreams}.
     */
    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. MetroStationWithStreamsDemo.java:

```
package part2.lab1.task1;

/**
* The {@code MetroStationWithStreamsDemo} class is a demonstration of
functionality
* of the {@link MetroStationWithStreams}.
*/
public class MetroStationWithStreamsDemo {
    /**
     * Performs demonstration of functionality of the {@link
MetroStationWithStreams}.
    * The {@code args} are not used.
    * @param args the command-line arguments.
    */
   public static void main(String[] args) {

MetroStationWithStreams.createMetroStationWithStreams().testMetroStatio
nWithStreams();
   }
}
```

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

1.3.1. MetroStationWithStreamsTest.java:

```
package part2.lab1.task1;

import part1.lab4.task1.Hour;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.EmptySource;

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

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.junit.jupiter.api.Assertions.assertNull;
import static org.junit.jupiter.api.Assertions.assertArrayEquals;

/**
* The {@code MetroStationWithStreamsTest} class provides unit tests for
the {@link MetroStationWithStreams} class.
* It includes tests for setting Operating Hours, sorting and finding
Operating Hours based on certain conditions.
* Each test method in this class corresponds to a method in the {@link
MetroStationWithStreams} class.
*/
class MetroStationWithStreamsTest {
   private MetroStationWithStreams metroStationWithStreams;
   /**
    * Assistive method to get riderships from a list of Operating
Hours.
    * @param hours the list of Operating Hours;
    * @return an array of riderships.
    */
   private static int[] getRiderships(List<Hour> hours) {
       return hours.stream()
               .mapToInt(Hour::getRidership)
               .toArray();
   }

   /**
    * Assistive method to get riderships from an array of Operating
Hours.
    * @param hours the array of Operating Hours;
    * @return an array of riderships.
    */
   private static int[] getRiderships(Hour[] hours) {
       return Arrays.stream(hours)
               .mapToInt(Hour::getRidership)
               .toArray();
   }

   @BeforeEach
   public void setup() {
       metroStationWithStreams = new
MetroStationWithStreams("Politekhnichna", 1984,
               Arrays.asList(
                       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")
               ));
   }

   @Nested
   class TestSettingHours {
       /**
        * Tests the {@link MetroStationWithStreams#setHoursList(List)}
method.
        */
       @Test
       @DisplayName("Should Set Hours")
       public void setHours() {
           List<Hour> hours = new
ArrayList<>(metroStationWithStreams.getHoursList());
```

```
            hours.add(new Hour(27, "Very low ridership"));
            hours.add(new Hour(250, "Medium ridership"));
            metroStationWithStreams.setHoursList(hours);
            assertEquals(hours.size(),
metroStationWithStreams.getHoursList().size());
            hours.forEach(hour ->
assertTrue(metroStationWithStreams.getHoursList().contains(hour)));
       }

       /**
        * Tests the {@link MetroStationWithStreams#setHoursList(List)}
method.
        * Checks if the method handles duplicate Operating Hours
correctly.
        */
       @Test
       @DisplayName("Should Handle Duplicate Operating Hours When
Setting")
       public void setDaysWithDuplicate() {
            List<Hour> hours = new
ArrayList<>(metroStationWithStreams.getHoursList());
            Hour duplicateHour = new Hour(200, "Medium ridership");
            hours.add(duplicateHour);
            metroStationWithStreams.setHoursList(hours);
            long count = metroStationWithStreams.getHoursList().stream()
                    .filter(hour -> hour.equals(duplicateHour))
                    .count();
            assertEquals(2, count);
       }
   }

   @Nested
   class TestSortingHours {
       /**
        * Tests the {@link
MetroStationWithStreams#sortByDecreasingRidership()} method.
        */
       @Test
       @DisplayName("Should Sort By Decreasing Ridership")
       public void sortByDecreasingRidership() {
            metroStationWithStreams.sortByDecreasingRidership();
            int[] expected = new int[]{650, 532, 200, 188, 110, 60};
            int[] actual =
getRiderships(metroStationWithStreams.getHoursList());
            assertArrayEquals(expected, actual);
       }

       /**
        * Tests the {@link
```

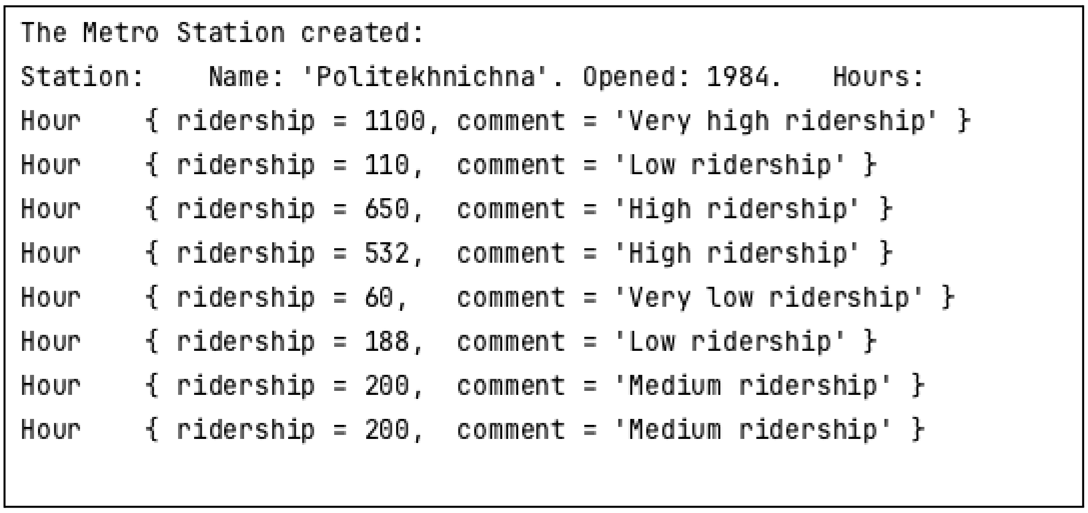
```
MetroStationWithStreams#sortByDescendingCommentLength()} method.
         */
        @Test
        @DisplayName("Should Sort By Descending Comment Length")
        public void sortByDescendingCommentLength() {
            metroStationWithStreams.sortByDescendingCommentLength();
            int[] expected = new int[]{60, 200, 650, 532, 110, 188};
            int[] actual =
getRiderships(metroStationWithStreams.getHoursList());
            assertArrayEquals(expected, actual);
        }
    }

    @Nested
    class TestTotalRidershipCalculations {
        /**
         * Tests the {@link
MetroStationWithStreams#calculateTotalRidership()} method.
         */
        @Test
        @DisplayName("Should Calculate Total Ridership")
        public void calculateTotalRidership() {
            assertEquals(1740,
metroStationWithStreams.calculateTotalRidership(), 0);
        }

        /**
         * Tests the {@link
MetroStationWithStreams#calculateTotalRidership()} method.
         * Checks if the method returns null when there are no Operating
Hours.
         */
        @ParameterizedTest
        @DisplayName("Should Return Null When Calculating Total
Ridership Without Operating Hours")
        @EmptySource
        public void calculateTotalRidershipWithoutHours(List<Hour>
emptyList) {
            metroStationWithStreams.setHoursList(emptyList);

assertNull(metroStationWithStreams.calculateTotalRidership());
        }
    }

    @Nested
    class TestSearchHours {
        /**
         * Tests the {@link
MetroStationWithStreams#findHoursWithMinRidership()} method.
```

```
         * Checks if the method finds the one Operating Hour with the
minimal ridership correctly.
         */
        @Test
        @DisplayName("Should Find One Operating Hour With Minimum
Ridership")
        public void findHoursWithMinRidershipWithOneMinRidership() {
            int[] expected = new int[]{60};
            int[] actual =
getRiderships(metroStationWithStreams.findHoursWithMinRidership());
            assertArrayEquals(expected, actual);
        }

        /**
         * Tests the {@link
MetroStationWithStreams#findHoursWithMinRidership()} method.
         * Checks if the method finds multiple Operating Hours with the
minimal ridership correctly.
         */
        @Test
        @DisplayName("Should Find Two Operating Hours With Minimum
Ridership")
        public void findHoursWithMinRidershipWithMultipleMinRidership()
{
            List<Hour> hours = new
ArrayList<>(metroStationWithStreams.getHoursList());
            Hour duplicateMinRidership = new Hour(60, "Very low
ridership");
            hours.add(duplicateMinRidership);
            metroStationWithStreams.setHoursList(hours);

            int[] expected = new int[]{60, 60};
            int[] actual =
getRiderships(metroStationWithStreams.findHoursWithMinRidership());
            assertArrayEquals(expected, actual);
        }

        /**
         * Tests the {@link
MetroStationWithStreams#findHoursWithMinRidership()} method.
         * Checks if the method returns null when there are no Operating
Hours.
         */
        @ParameterizedTest
        @DisplayName("Should Return Null When Finding Minimum Ridership
Operating Hours Without Operating Hours")
        @EmptySource
        public void findHoursWithMinRidershipWithoutHours(List<Hour>
emptyList) {
```

```
           metroStationWithStreams.setHoursList(emptyList);

assertNull(metroStationWithStreams.findHoursWithMinRidership());
       }

       /**
        * Tests the {@link
MetroStationWithStreams#findHoursWithMaxWordCountOfComment()} method.
        * Checks if the method finds the one Operating Hour with the
max word count of comment correctly.
        */
       @Test
       @DisplayName("Should Find One Operating Hour With Maximum Word
Count Of Comment")
       public void
findHoursWithMaxWordCountOfCommentWithOneMaxWordCountOfComment() {
           int[] expected = new int[]{60};
           int[] actual =
getRiderships(metroStationWithStreams.findHoursWithMaxWordCountOfCommen
t());
           assertArrayEquals(expected, actual);
       }

       /**
        * Tests the {@link
MetroStationWithStreams#findHoursWithMaxWordCountOfComment()} method.
        * Checks if the method finds multiple Operating Hours with the
max word count of comment correctly.
        */
       @Test
       @DisplayName("Should Find Two Operating Hours With Maximum Word
Count Of Comment")
       public void
findHoursWithMaxWordCountOfCommentWithMultipleMaxWordCountOfComment() {
           List<Hour> hours = new
ArrayList<>(metroStationWithStreams.getHoursList());
           Hour duplicateHourWithMaxWordCountOfComment = new Hour(1100,
"Very high ridership");
           hours.add(duplicateHourWithMaxWordCountOfComment);
           metroStationWithStreams.setHoursList(hours);

           int[] expected = new int[]{60, 1100};
           int[] actual =
getRiderships(metroStationWithStreams.findHoursWithMaxWordCountOfCommen
t());
           assertArrayEquals(expected, actual);
       }

       /**
```

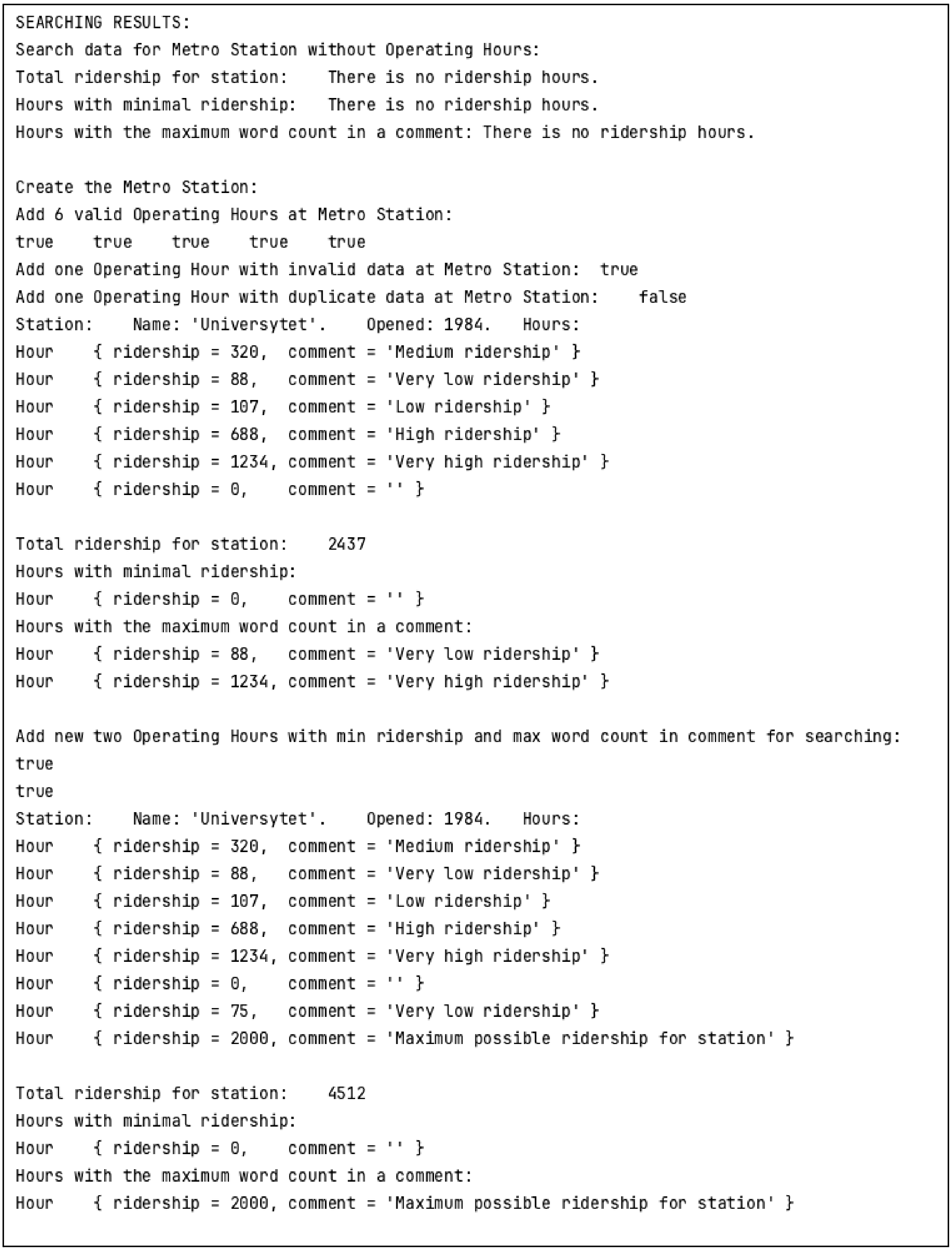
```
         * Tests the {@link
MetroStationWithStreams#findHoursWithMaxWordCountOfComment()} ()}
method.
         * Checks if the method returns null when there are no Operating
Hours.
         */
        @ParameterizedTest
        @DisplayName("Should Return Null When Finding Operating Hours
With Maximum Word Count Of Comment Without Operating Hours")
        @EmptySource
        public void
findHoursWithMaxWordCountOfCommentWithoutHours(List<Hour> emptyList) {
            metroStationWithStreams.setHoursList(emptyList);

assertNull(metroStationWithStreams.findHoursWithMaxWordCountOfComment()
);
        }
    }
}
```

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



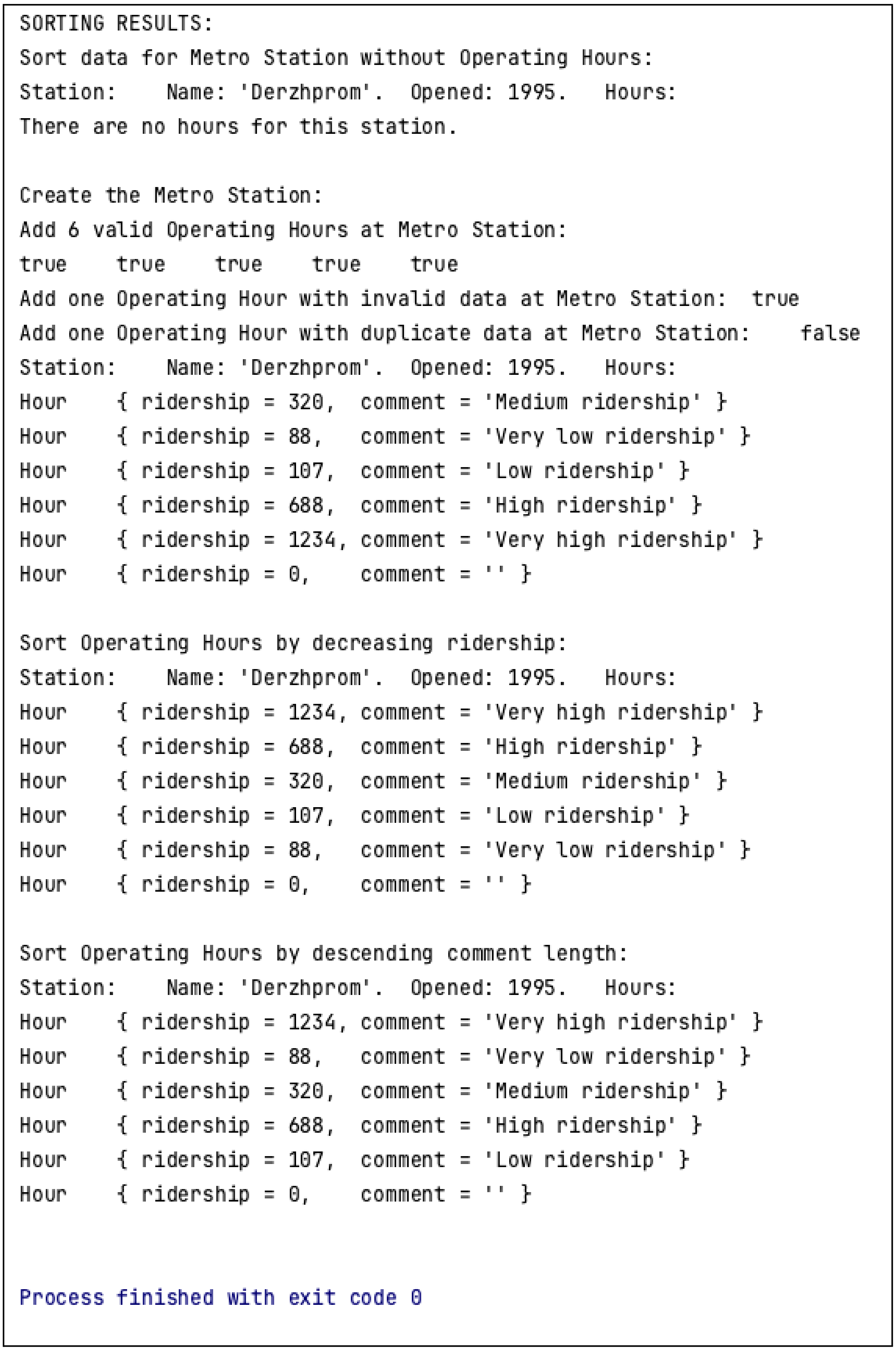
```
The Metro Station created:
Station:    Name: 'Politekhnichna'. Opened: 1984.   Hours:
Hour    { ridership = 1100, comment = 'Very high ridership' }
Hour    { ridership = 110,  comment = 'Low ridership' }
Hour    { ridership = 650,  comment = 'High ridership' }
Hour    { ridership = 532,  comment = 'High ridership' }
Hour    { ridership = 60,   comment = 'Very low ridership' }
Hour    { ridership = 188,  comment = 'Low ridership' }
Hour    { ridership = 200,  comment = 'Medium ridership' }
Hour    { ridership = 200,  comment = 'Medium ridership' }
```

Рисунок 1.4.1 – Результати №1 роботи програмного коду класу MetroStationWithStreamsDemo.java



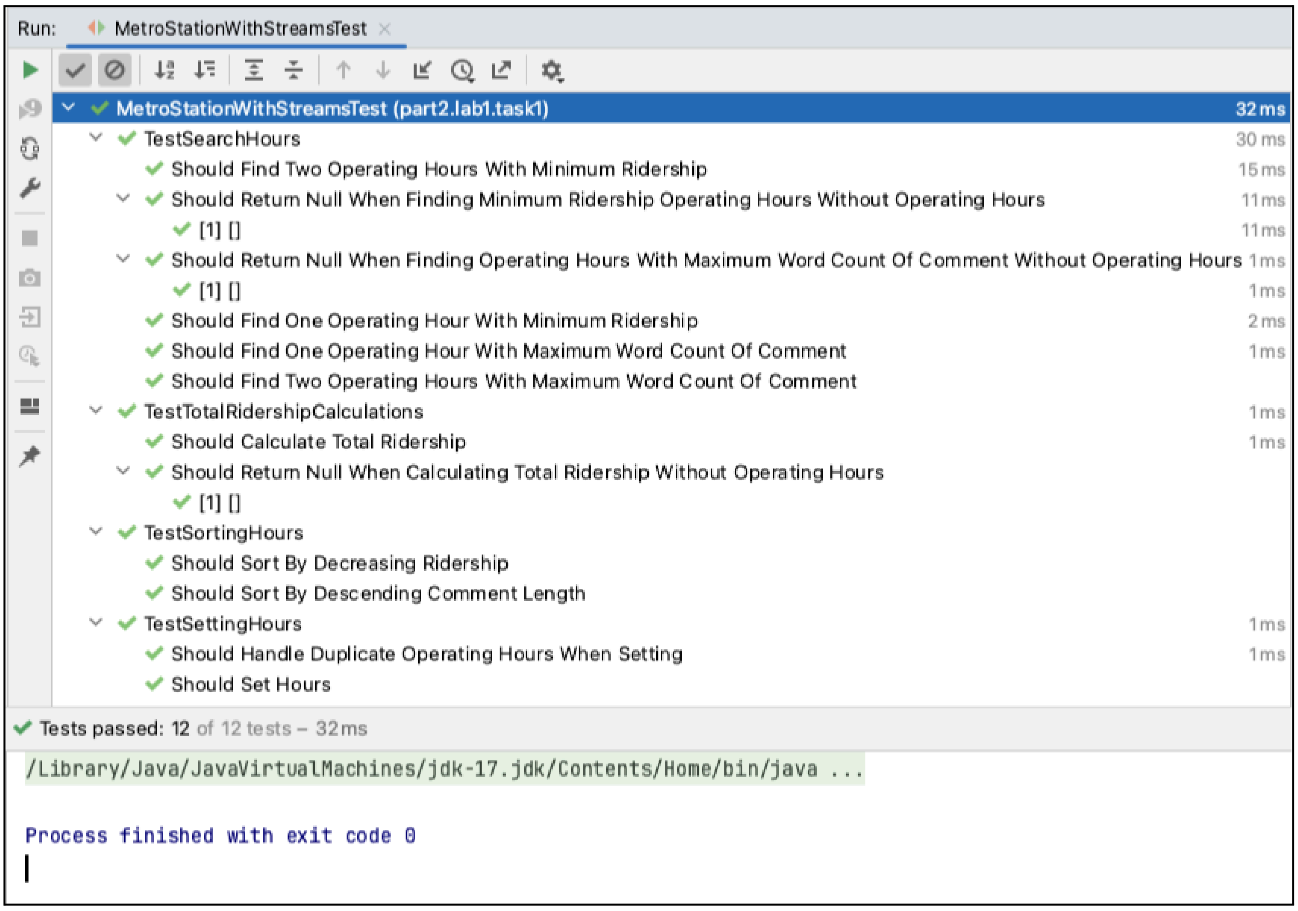
```
SEARCHING RESULTS:
Search data for Metro Station without Operating Hours:
Total ridership for station:    There is no ridership hours.
Hours with minimal ridership:   There is no ridership hours.
Hours with the maximum word count in a comment: There is no ridership hours.

Create the Metro Station:
Add 6 valid Operating Hours at Metro Station:
true    true    true    true    true
Add one Operating Hour with invalid data at Metro Station:  true
Add one Operating Hour with duplicate data at Metro Station:    false
Station:    Name: 'Universytet'.    Opened: 1984.   Hours:
Hour    { ridership = 320,  comment = 'Medium ridership' }
Hour    { ridership = 88,   comment = 'Very low ridership' }
Hour    { ridership = 107,  comment = 'Low ridership' }
Hour    { ridership = 688,  comment = 'High ridership' }
Hour    { ridership = 1234, comment = 'Very high ridership' }
Hour    { ridership = 0,    comment = '' }

Total ridership for station:    2437
Hours with minimal ridership:
Hour    { ridership = 0,    comment = '' }
Hours with the maximum word count in a comment:
Hour    { ridership = 88,   comment = 'Very low ridership' }
Hour    { ridership = 1234, comment = 'Very high ridership' }

Add new two Operating Hours with min ridership and max word count in comment for searching:
true
true
Station:    Name: 'Universytet'.    Opened: 1984.   Hours:
Hour    { ridership = 320,  comment = 'Medium ridership' }
Hour    { ridership = 88,   comment = 'Very low ridership' }
Hour    { ridership = 107,  comment = 'Low ridership' }
Hour    { ridership = 688,  comment = 'High ridership' }
Hour    { ridership = 1234, comment = 'Very high ridership' }
Hour    { ridership = 0,    comment = '' }
Hour    { ridership = 75,   comment = 'Very low ridership' }
Hour    { ridership = 2000, comment = 'Maximum possible ridership for station' }

Total ridership for station:    4512
Hours with minimal ridership:
Hour    { ridership = 0,    comment = '' }
Hours with the maximum word count in a comment:
Hour    { ridership = 2000, comment = 'Maximum possible ridership for station' }
```

Рисунок 1.4.2 – Результати №2 роботи програмного коду класу MetroStationWithStreamsDemo.java



```
SORTING RESULTS:
Sort data for Metro Station without Operating Hours:
Station:    Name: 'Derzhprom'.  Opened: 1995.   Hours:
There are no hours for this station.

Create the Metro Station:
Add 6 valid Operating Hours at Metro Station:
true    true    true    true    true
Add one Operating Hour with invalid data at Metro Station:  true
Add one Operating Hour with duplicate data at Metro Station:    false
Station:    Name: 'Derzhprom'.  Opened: 1995.   Hours:
Hour    { ridership = 320,  comment = 'Medium ridership' }
Hour    { ridership = 88,   comment = 'Very low ridership' }
Hour    { ridership = 107,  comment = 'Low ridership' }
Hour    { ridership = 688,  comment = 'High ridership' }
Hour    { ridership = 1234, comment = 'Very high ridership' }
Hour    { ridership = 0,    comment = '' }

Sort Operating Hours by decreasing ridership:
Station:    Name: 'Derzhprom'.  Opened: 1995.   Hours:
Hour    { ridership = 1234, comment = 'Very high ridership' }
Hour    { ridership = 688,  comment = 'High ridership' }
Hour    { ridership = 320,  comment = 'Medium ridership' }
Hour    { ridership = 107,  comment = 'Low ridership' }
Hour    { ridership = 88,   comment = 'Very low ridership' }
Hour    { ridership = 0,    comment = '' }

Sort Operating Hours by descending comment length:
Station:    Name: 'Derzhprom'.  Opened: 1995.   Hours:
Hour    { ridership = 1234, comment = 'Very high ridership' }
Hour    { ridership = 88,   comment = 'Very low ridership' }
Hour    { ridership = 320,  comment = 'Medium ridership' }
Hour    { ridership = 688,  comment = 'High ridership' }
Hour    { ridership = 107,  comment = 'Low ridership' }
Hour    { ridership = 0,    comment = '' }


Process finished with exit code 0
```

Рисунок 1.4.3 – Результати №3 роботи програмного коду класу MetroStationWithStreamsDemo.java


Run: MetroStationWithStreamsTest

```
MetroStationWithStreamsTest (part2.lab1.task1)                                                                      32 ms
  TestSearchHours                                                                                                   30 ms
    Should Find Two Operating Hours With Minimum Ridership                                                          15 ms
    Should Return Null When Finding Minimum Ridership Operating Hours Without Operating Hours                        11 ms
      [1] []                                                                                                        11 ms
    Should Return Null When Finding Operating Hours With Maximum Word Count Of Comment Without Operating Hours        1 ms
      [1] []                                                                                                         1 ms
    Should Find One Operating Hour With Minimum Ridership                                                            2 ms
    Should Find One Operating Hour With Maximum Word Count Of Comment                                                1 ms
    Should Find Two Operating Hours With Maximum Word Count Of Comment
  TestTotalRidershipCalculations                                                                                     1 ms
    Should Calculate Total Ridership                                                                                 1 ms
    Should Return Null When Calculating Total Ridership Without Operating Hours
      [1] []
  TestSortingHours
    Should Sort By Decreasing Ridership
    Should Sort By Descending Comment Length
  TestSettingHours                                                                                                   1 ms
    Should Handle Duplicate Operating Hours When Setting                                                             1 ms
    Should Set Hours
Tests passed: 12 of 12 tests – 32 ms
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Рисунок 1.4.4 – Результати модульного тестування з використанням Junit програмного коду класу MetroStationWithStreamsTest.java

2. Завдання №2 до лабораторної роботи

2.1. Знаходження цілого степеня

Написати програму, яка здійснює заповнення числа типу BigInteger випадковими цифрами та обчислення цілого степеня цього числа. Для результату використати BigInteger. Реалізувати два варіанти – з використанням функції pow() класу та функції, що забезпечує множення довгих цілих. Порівняти результати.

Забезпечити тестування методів класу з використанням JUnit.

2.2. Програмний код реалізації завдання № 2

2.2.1. BigInetgerPower.java:

```
package part2.lab1.task2;
```

```java
import java.math.BigInteger;
import java.util.Random;
import java.util.function.BiFunction;

/**
* This class provides methods for working with BigInteger numbers,
* including generating a random BigInteger, raising a BigInteger to a
power,
* and comparing the results and durations of different methods of
raising a BigInteger to a power.
*/
public class BigIntegerPower {
   /** The number of digits of the number. */
   public static final int NUM_DIGITS = 30;

   /** The maximum number of digits to be displayed in the console. */
   public static final int MAX_DISPLAY_DIGITS = 60;

   /**
    * Generates a random BigInteger with a '{@value NUM_DIGITS}' of
digits.
    * @return a random BigInteger with '{@value NUM_DIGITS}' digits.
    */
   public static BigInteger generateRandomBigInteger() {
       Random random = new Random();
       StringBuilder sb = new StringBuilder(NUM_DIGITS);
       sb.append(random.nextInt(9) + 1);

       for (int i = 1; i < NUM_DIGITS; i++) {
           sb.append(random.nextInt(10));
       }

       return new BigInteger(sb.toString());
   }

   /**
    * Raises a BigInteger to a power using the pow function {@link
BigInteger#pow(int)}.
    * @param base the base BigInteger;
    * @param exponent the exponent to raise the base to;
    * @return the result of raising the base to the exponent.
    */
   public static BigInteger powerUsingPowFunction(BigInteger base, int
exponent) {
       return base.pow(exponent);
   }

   /**
```

```
    * Raises a BigInteger to a power using method {@link
BigInteger#multiply(BigInteger)}.
    * @param base     the base BigInteger;
    * @param exponent the exponent to raise the base to;
    * @return the result of raising the base to the exponent;
    * @throws ArithmeticException {@code exponent} is negative
    * (this would cause the operation to yield a non-integer value).
    */
   public static BigInteger powerUsingMultiplication(BigInteger base,
int exponent) {
       if (exponent < 0) {
           throw new ArithmeticException("Negative exponent");
       }

       BigInteger result = BigInteger.ONE;

       for (int i = 0; i < exponent; i++) {
           result = result.multiply(base);
       }

       return result;
   }

   /** A class to hold a result and the duration it took to compute
that result. */
   public static class ResultAndDurationPair {
       private final BigInteger result;
       private final long duration;

       public ResultAndDurationPair(BigInteger result, long duration) {
           this.result = result;
           this.duration = duration;
       }

       public BigInteger getResult() {
           return result;
       }

       public long getDuration() {
           return duration;
       }
   }

   /**
    * Calculates the result of a function and measures the time it took
to compute the result.
    * @param base the base BigInteger;
    * @param exponent the exponent to raise the base to;
    * @param function {@code BiFunction} - the function to apply to the
```

```
base and exponent;
    * @return a ResultAndDurationPair containing the result and the
duration it took to compute the result.
    */
   public static ResultAndDurationPair
calculateAndMeasureTime(BigInteger base,
                                                               int
exponent,

BiFunction<BigInteger, Integer, BigInteger> function) {
       long startTime = System.nanoTime();
       BigInteger result = function.apply(base, exponent);
       long endTime = System.nanoTime();
       long duration = endTime - startTime;

       return new ResultAndDurationPair(result, duration);
   }

   /**
    * Formats a BigInteger for display, truncating it if it has more
than '{@value MAX_DISPLAY_DIGITS}' digits.
    * @param number the BigInteger to format;
    * @return a string representation of the BigInteger, truncated if
necessary.
    */
   public static String formatBigInteger(BigInteger number) {
       String str = number.toString();

       if (str.length() > MAX_DISPLAY_DIGITS) {
           str = str.substring(0, MAX_DISPLAY_DIGITS) + "...";
       }

       return str;
   }

   /**
    * Compares two BigIntegers and prints a message indicating whether
they are equal.
    * @param resultUsingPow the first BigInteger to compare;
    * @param resultUsingMultiplication the second BigInteger to
compare.
    */
   public static void printComparingResults(BigInteger resultUsingPow,
                                            BigInteger
resultUsingMultiplication) {
       if (resultUsingPow.equals(resultUsingMultiplication)) {
           System.out.println("The results are equal.");
       } else {
           System.out.println("The results are not equal.");
```

```
        }
    }

    /**
     * Compares two durations and prints a message indicating which is
shorter.
     * @param durationUsingPow the first duration to compare;
     * @param durationUsingMultiplication the second duration to
compare.
     */
    public static void printComparingDurations(long durationUsingPow,
                                                long
durationUsingMultiplication) {
        if (durationUsingPow < durationUsingMultiplication) {
            System.out.println("Pow function is faster.");
        } else if (durationUsingPow > durationUsingMultiplication) {
            System.out.println("Multiplication is faster.");
        } else {
            System.out.println("Both methods took the same amount of
time.");
        }
    }
}
```

2.2.2. BigIntegerPowerDemo.java:

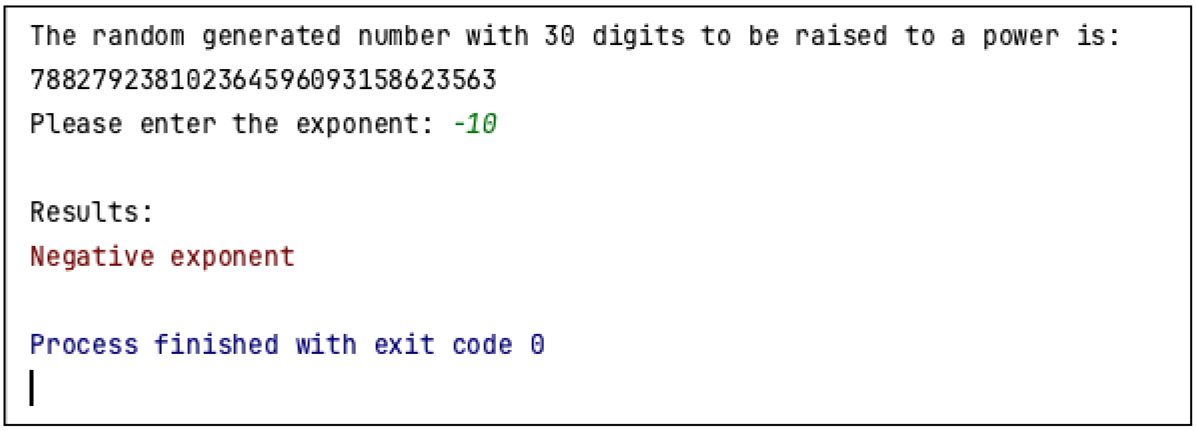
```
package part2.lab1.task2;

import java.math.BigInteger;
import java.util.Scanner;

import static part2.lab1.task2.BigIntegerPower.*;

/**
* The {@code BigIntegerPowerDemo} class represents showing of the
functionality of the {@link BigIntegerPower}.
* It generates a random BigInteger, asks the user for an exponent,
raises the BigInteger to the power of the exponent
* using two different methods, and compares the results and the time
taken by each method.
*/
public class BigIntegerPowerDemo {
    /**
     * Carries out showing of the functionality of the {@link
BigIntegerPower}.
     * {@code args} are not used;
     * @param args the command-line arguments.
     */
    public static void main(String[] args) {
        BigInteger base = generateRandomBigInteger();
        System.out.println("The random generated number with "
                + NUM_DIGITS + " digits to be raised to a power is:\n" +
base);
        System.out.print("Please enter the exponent: ");
        int exponent = new Scanner(System.in).nextInt();

        System.out.println("\nResults:");

        try {
            BigIntegerPower.ResultAndDurationPair resultUsingPow =
                    calculateAndMeasureTime(base, exponent,
BigIntegerPower::powerUsingPowFunction);
            System.out.println("Result using pow function: " +
formatBigInteger(resultUsingPow.getResult()));
            System.out.println("Time taken using pow function: " +
resultUsingPow.getDuration() + " nanoseconds");

            BigIntegerPower.ResultAndDurationPair
resultUsingMultiplication =
                    calculateAndMeasureTime(base, exponent,
BigIntegerPower::powerUsingMultiplication);
            System.out.println("\nResult using multiplication: "
                    +
formatBigInteger(resultUsingMultiplication.getResult()));
            System.out.println("Time taken using multiplication: "
                    + resultUsingMultiplication.getDuration() + "
nanoseconds");

            System.out.println("\nConclusions:");
            printComparingResults(resultUsingPow.getResult(),
resultUsingMultiplication.getResult());
            printComparingDurations(resultUsingPow.getDuration(),
resultUsingMultiplication.getDuration());
        } catch (ArithmeticException e) {
            System.err.println(e.getMessage());
        }
    }
}
```

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

2.3.1. BigIntegerPowerTest.java:

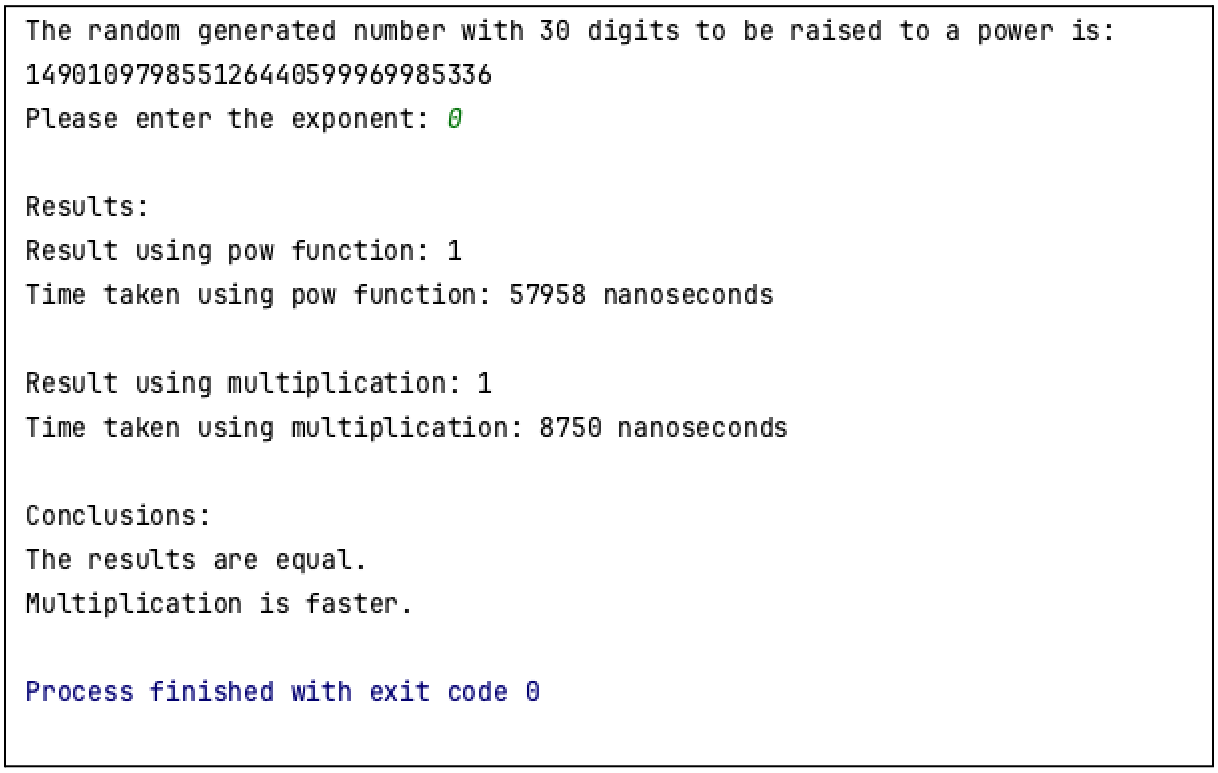
```java
package part2.lab1.task2;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;

import java.math.BigInteger;
import java.util.function.BiFunction;
import java.util.stream.Stream;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

/**
* The {@code BigIntegerPowerTest} class provides unit tests for the
{@link BigIntegerPower} class.
* It tests the generation of random BigIntegers, the calculation of
powers using two different methods,
* and the formatting of BigIntegers for display.
*/
class BigIntegerPowerTest {
   /** A constant representing the base BigInteger used in the tests.
*/
   private static final BigInteger BASE = new
BigInteger("98273491827349812734981273498l");

   /** A constant representing the fifth power of {@code BASE}. */
   private static final BigInteger FIFTH_POWER_BASE = new
BigInteger("91660440028948881638815 5" +

"03476828492344299720577607828648822739360218174865096693195415188766 80
501512369891817 0" +
           "01041479958588407184322529218444371989 01");

   /**
    * Provides data for the power calculation tests.
    * @return a stream of arguments for the power calculation tests.
    */
   private static Stream<Arguments>
provideDataForPowerCalculationTests() {
       BiFunction<BigInteger, Integer, BigInteger> powFunction =
BigIntegerPower::powerUsingPowFunction;
       BiFunction<BigInteger, Integer, BigInteger>
multiplicationFunction = BigIntegerPower::powerUsingMultiplication;

       return Stream.of(
               Arguments.of(powFunction, -1, null),
               Arguments.of(powFunction, 0, BigInteger.ONE),
               Arguments.of(powFunction, 5, FIFTH_POWER_BASE),
               Arguments.of(multiplicationFunction, -1, null),
               Arguments.of(multiplicationFunction, 0, BigInteger.ONE),
               Arguments.of(multiplicationFunction, 5,
FIFTH_POWER_BASE)
       );
   }

   /**
    * Tests the {@link BigIntegerPower#generateRandomBigInteger()}
method.
    * Checks if the generated BigInteger has the expected number of
digits.
    */
   @Test
   @DisplayName("Should generate BigInteger with the expected number of
digits")
   public void generateRandomBigInteger() {
       BigInteger result = BigIntegerPower.generateRandomBigInteger();
       int expectedNumDigits = BigIntegerPower.NUM_DIGITS;
       int actualNumDigits = result.toString().length();

       assertEquals(expectedNumDigits, actualNumDigits);
   }

   /** Tests the calculation of powers using two different methods. */
   @ParameterizedTest(name = "Test {index}: exponent = {1}, expected =
{2}")
   @DisplayName("Should calculate power. First 3 tests using function
powerUsingPowFunction, second 3 tests - powerUsingMultiplication")
   @MethodSource("provideDataForPowerCalculationTests")
   public void testPowerCalculation(BiFunction<BigInteger, Integer,
BigInteger> function,
                                    int exponent,
                                    BigInteger expected) {
       if (exponent < 0) {
           assertThrows(ArithmeticException.class, () ->
function.apply(BASE, exponent));
       } else {
           BigInteger actual = function.apply(BASE, exponent);
           assertEquals(expected, actual);
       }
   }

   /** Tests the {@link BigIntegerPower#formatBigInteger(BigInteger)}
method. */
   @Test
   @DisplayName("Should format BigInteger correctly")
   public void formatBigInteger() {
       String actualString = BigIntegerPower.formatBigInteger(BASE);
       int actualLength = actualString.length();
       int expectedLength = BASE.toString().length();

       if (BASE.toString().length() >
BigIntegerPower.MAX_DISPLAY_DIGITS) {
           expectedLength = BigIntegerPower.MAX_DISPLAY_DIGITS + 3;
       }

       assertEquals(expectedLength, actualLength);
   }
}
```

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



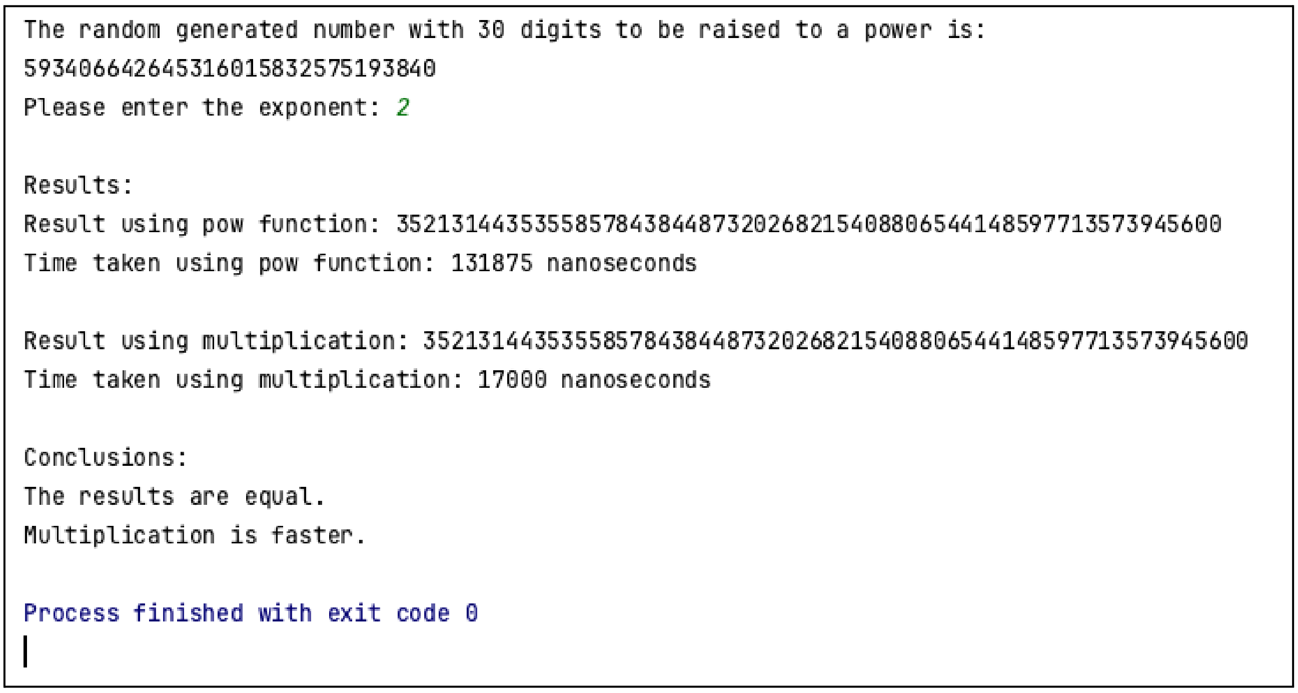
```
The random generated number with 30 digits to be raised to a power is:
788279238102364596093158623563
Please enter the exponent: -10

Results:
Negative exponent

Process finished with exit code 0
```

Рисунок 2.4.1 – Результати №1 роботи програмного коду класу BigIntegerPowerDemo.java



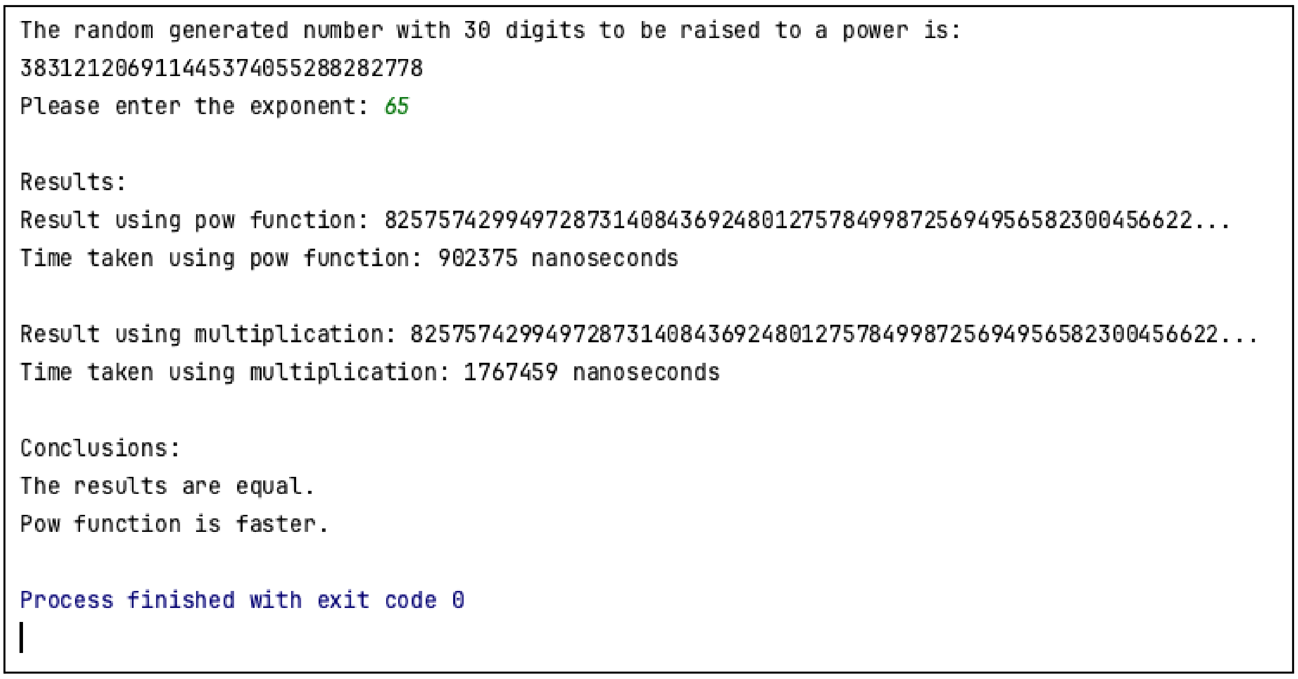
```
The random generated number with 30 digits to be raised to a power is:
149010979855126440599969985336
Please enter the exponent: 0

Results:
Result using pow function: 1
Time taken using pow function: 57958 nanoseconds

Result using multiplication: 1
Time taken using multiplication: 8750 nanoseconds

Conclusions:
The results are equal.
Multiplication is faster.

Process finished with exit code 0
```

Рисунок 2.4.2 – Результати №2 роботи програмного коду класу BigIntegerPowerDemo.java



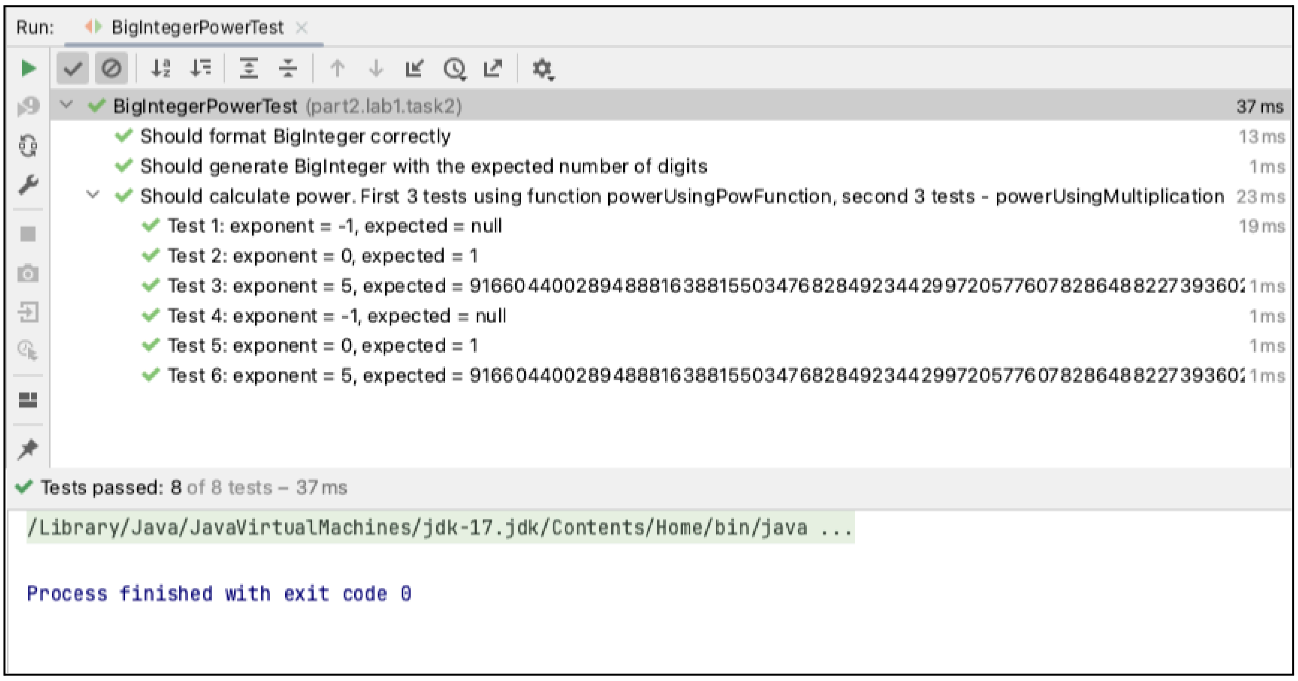
```
The random generated number with 30 digits to be raised to a power is:
593406642645316015832575193840
Please enter the exponent: 2

Results:
Result using pow function: 352131443535585784384487320268215408806544148597713573945600
Time taken using pow function: 131875 nanoseconds

Result using multiplication: 352131443535585784384487320268215408806544148597713573945600
Time taken using multiplication: 17000 nanoseconds

Conclusions:
The results are equal.
Multiplication is faster.

Process finished with exit code 0
```

Рисунок 2.4.3 – Результати №3 роботи програмного коду класу BigIntegerPowerDemo.java



```
The random generated number with 30 digits to be raised to a power is:
383121206911445374055288282778
Please enter the exponent: 65

Results:
Result using pow function: 8257574299497287314084369248012757849987256949565823004566 22...
Time taken using pow function: 902375 nanoseconds

Result using multiplication: 825757429949728731408436924801275784998725694956582300456622...
Time taken using multiplication: 1767459 nanoseconds

Conclusions:
The results are equal.
Pow function is faster.

Process finished with exit code 0
```

Рисунок 2.4.4 – Результати №4 роботи програмного коду класу BigIntegerPowerDemo.java



```
Run:  BigIntegerPowerTest
BigIntegerPowerTest (part2.lab1.task2)                                                                                    37 ms
  ✓ Should format BigInteger correctly                                                                                    13 ms
  ✓ Should generate BigInteger with the expected number of digits                                                          1 ms
  ✓ Should calculate power. First 3 tests using function powerUsingPowFunction, second 3 tests - powerUsingMultiplication 23 ms
      ✓ Test 1: exponent = -1, expected = null                                                                            19 ms
      ✓ Test 2: exponent = 0, expected = 1
      ✓ Test 3: exponent = 5, expected = 91660440028948881638815503476828492344299720577607828648822739360  1 ms
      ✓ Test 4: exponent = -1, expected = null                                                                             1 ms
      ✓ Test 5: exponent = 0, expected = 1                                                                                 1 ms
      ✓ Test 6: exponent = 5, expected = 91660440028948881638815503476828492344299720577607828648822739360  1 ms
✓ Tests passed: 8 of 8 tests – 37 ms
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Рисунок 2.4.5 – Результати модульного тестування з використанням Junit програмного коду класу BigIntegerPowerTest.java

3. Завдання №3 до лабораторної роботи

3.1. Фільтрація та сортування

Створити список об'єктів типу BigDecimal. Заповнити список випадковими значеннями. Здійснити сортування за зменшенням абсолютної величини. Знайти добуток додатних чисел. Реалізувати три підходи:

- з використанням циклів і умовних тверджень (без засобів, доданих у Java 8);
- без явних циклів і розгалужень, з використанням функцій, які були визначені в інтерфейсах Java Collection Framework починаючи з Java 8;
- з використанням засобів Stream API.

Забезпечити тестування класів з використанням JUnit.

3.2. Програмний код реалізації завдання № 3

3.2.1. BigDecimalProcessor.java:

```
package part2.lab1.task3;

import java.math.BigDecimal;
import java.util.List;

/**
* Interface for processing lists of BigDecimal values.
* @see BigDecimalProcessorWithLoops ;
* @see BigDecimalProcessorWithCollection ;
* @see BigDecimalProcessorWithStreams .
*/
public interface BigDecimalProcessor {
   /**
    * Fills the given list with random BigDecimal values.
    * @param list the list to fill;
    * @param size the number of elements.
    */
   void fillWithRandomValues(List<BigDecimal> list, int size);

   /**
    * Sorts the given list in descending order by absolute value.
    * @param list the list to sort;
    * @throws UnsupportedOperationException if the method is not
implemented.
    */
   static void sortDescendingByAbsoluteValue(List<BigDecimal> list) {
       throw new UnsupportedOperationException("Not implemented yet");
   }

   /**
    * Calculates the product of positive numbers in the given list.
    * @param list the list to process;
    * @return the product of positive numbers;
    * @throws UnsupportedOperationException if the method is not
implemented.
    */
   static BigDecimal calculateProductOfPositiveNumbers(List<BigDecimal>
list) {
       throw new UnsupportedOperationException("Not implemented yet");
   }

   /**
    * Prints the given list.
    * @param list the list to print;
    * @throws UnsupportedOperationException if the method is not
implemented.
    */
   static void printList(List<BigDecimal> list) {
       throw new UnsupportedOperationException("Not implemented yet");
   }
}
```

3.2.2. BigDecimalProcessorWithLoops.java:

```
package part2.lab1.task3;

import java.math.BigDecimal;
import java.util.List;
import java.util.Random;

/**
* This class implements the {@link BigDecimalProcessor} interface using
loops.
* It provides methods to fill a list with random BigDecimal values,
sort the list
* in descending order by absolute value, calculate the product of
positive numbers in the list, and print the list.
*/
public class BigDecimalProcessorWithLoops implements
BigDecimalProcessor {
    private final Random random = new Random();

    private final int bound;
```

```
   /**
    * Constructs a new BigDecimalProcessorWithLoops with the specified
bound for random values.
    * @param bound the bound for random values.
    */
   BigDecimalProcessorWithLoops(int bound) {
       this.bound = bound;
   }

   /**
    * Returns the bound for random values.
    * @return the bound for random values.
    */
   public int getBound() {
       return bound;
   }

   /**
    * Fills the given list with random BigDecimal values within the
bound.
    * @param list the list to fill;
    * @param size the number of elements.
    */
   public void fillWithRandomValues(List<BigDecimal> list, int size) {
       list.clear();

       for (int i = 0; i < size; i++) {
//            BigDecimal value = BigDecimal.valueOf(random.nextDouble()
* 500 - 250);
           BigDecimal value = new BigDecimal(random.nextDouble() * 2 *
bound - bound);
           list.add(value);
       }
   }

   /**
    * Sorts the given list in descending order by absolute value using
a bubble sort algorithm.
    * @param list the list to sort.
    */
   public static void sortDescendingByAbsoluteValue(List<BigDecimal>
list) {
       boolean mustSorted = true;

       while (mustSorted) {
           mustSorted = false;

           for (int i = 0; i < list.size() - 1; i++) {
               if (list.get(i).abs().compareTo(list.get(i + 1).abs())
== -1) {
                   BigDecimal temp = list.get(i);
                   list.set(i, list.get(i + 1));
                   list.set(i + 1, temp);
                   mustSorted = true;
               }
           }
       }
   }

   /**
    * Calculates the product of positive numbers in the given list.
    * @param list the list to process;
    * @return the product of positive numbers.
    */
   public static BigDecimal
calculateProductOfPositiveNumbers(List<BigDecimal> list) {
       BigDecimal product = BigDecimal.ONE;
       BigDecimal value;

       for (int i = 0; i < list.size(); i++) {
           value = list.get(i);

           if (value.compareTo(BigDecimal.ZERO) == 1) {
               product = product.multiply(value);
           }
       }

       return product;
   }

   /**
    * Prints the given list to the standard output.
    * @param list the list to print.
    */
   public static void printList(List<BigDecimal> list) {
       for (int i = 0; i < list.size(); i++) {
           System.out.println(list.get(i));
       }
   }
}
```

3.2.3. BigDecimalProcessorWithCollection.java:

```
package part2.lab1.task3;

import java.math.BigDecimal;
import java.util.Comparator;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

/**
* This class implements the {@link BigDecimalProcessor} interface using
collection functions.
* It provides methods to fill a list with random BigDecimal values,
sort the list
* in descending order by absolute value, calculate the product of
positive numbers in the list, and print the list.
*/
public class BigDecimalProcessorWithCollection implements
BigDecimalProcessor {
   private final Random random = new Random();

   private final int bound;

   /**
    * Constructs a new {@code BigDecimalProcessorWithCollection} with
the specified bound for random values.
    * @param bound the bound for random values.
    */
   BigDecimalProcessorWithCollection(int bound) {
       this.bound = bound;
   }

   /**
    * Returns the bound for random values.
    * @return the bound for random values.
    */
   public int getBound() {
       return bound;
   }

   /**
    * Fills the given list with random BigDecimal values within the
bound.
    * @param list the list to fill;
    * @param size the number of elements.
    */
   public void fillWithRandomValues(List<BigDecimal> list, int size) {
       list.clear();
       list.addAll(IntStream.range(0, size)
                       .mapToObj(i ->
                               new BigDecimal(random.nextDouble() * 2 *
bound - bound))
```

```
                                .collect(Collectors.toList())
        );
    }

    /**
     * Sorts the given list in descending order by absolute value using
collection functions.
     * @param list the list to sort.
     */
    public static void sortDescendingByAbsoluteValue(List<BigDecimal>
list) {
        list.sort(Comparator.comparing((BigDecimal bd) ->
bd.abs()).reversed());
    }

    /**
     * Calculates the product of positive numbers in the given list
using collection functions.
     * @param list the list to process;
     * @return the product of positive numbers.
     */
    public static BigDecimal
calculateProductOfPositiveNumbers(List<BigDecimal> list) {
        BigDecimal[] product = {BigDecimal.ONE};
        list.forEach(value -> {
            if (value.compareTo(BigDecimal.ZERO) == 1) {
                product[0] = product[0].multiply(value);
            }
        });

        return product[0];
    }

    /**
     * Prints the given list to the standard output.
     * @param list the list to print.
     */
    public static void printList(List<BigDecimal> list) {
        list.forEach(System.out::println);
    }
}
```

3.2.4. BigDecimalProcessorWithStreams.java:

```
package part2.lab1.task3;

import java.math.BigDecimal;
import java.util.Comparator;
import java.util.List;
import java.util.Random;
import java.util.function.Supplier;
import java.util.stream.Collectors;
import java.util.stream.Stream;

/**
* This class implements the {@link BigDecimalProcessor} interface using
Java Streams API.
* It provides methods to fill a list with random BigDecimal values
within the bound
* using {@link Stream#generate(Supplier)} sort the list in descending
order by absolute value
* using the sorted method of the Stream interface, calculate the
product of positive numbers in the list
* using the filter and reduce methods of the Stream interface, and
print the list to the standard output
* using the forEach method of the Stream interface.
*/
public class BigDecimalProcessorWithStreams implements
BigDecimalProcessor {
   private final Random random = new Random();

   private final int bound;

   /**
    * Constructs a new BigDecimalProcessorWithStreams with the
specified bound for random values.
    * @param bound the bound for random values.
    */
   BigDecimalProcessorWithStreams(int bound) {
       this.bound = bound;
   }

   /**
    * Returns the bound for random values.
    * @return the bound for random values.
    */
   public int getBound() {
       return bound;
   }

   /**
    * Fills the given list with random BigDecimal values within the
bound using Stream.generate.
    * @param list the list to fill;
    * @param size the number of elements.
    */
   public void fillWithRandomValues(List<BigDecimal> list, int size) {
       list.clear();
       list.addAll(Stream.generate(() ->
                       new BigDecimal(random.nextDouble() * 2 * bound -
bound)).limit(size)
               .collect(Collectors.toList()));
   }

   /**
    * Sorts the given list in descending order by absolute value using
the sorted method of the Stream interface.
    * @param list the list to sort.
    */
   public static void sortDescendingByAbsoluteValue(List<BigDecimal>
list) {
       List<BigDecimal> sortedList = list.stream()
               .sorted(Comparator.comparing((BigDecimal bd) ->
bd.abs()).reversed())
               .collect(Collectors.toList());
       list.clear();
       list.addAll(sortedList);
   }

   /**
    * Calculates the product of positive numbers in the given list
using the filter and reduce methods of the Stream interface.
    * @param list the list to process;
    * @return the product of positive numbers.
    */
   public static BigDecimal
calculateProductOfPositiveNumbers(List<BigDecimal> list) {
       return list.stream()
               .filter(value -> value.compareTo(BigDecimal.ZERO) == 1)
               .reduce(BigDecimal.ONE, BigDecimal::multiply);
   }

   /**
    * Prints the given list to the standard output using the forEach
method of the Stream interface.
    * @param list the list to print.
    */
   public static void printList(List<BigDecimal> list) {
       list.stream().forEach(System.out::println);
   }
}
```

3.2.5. BigDecimalProcessorDemo.java:

```
package part2.lab1.task3;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;
import java.util.function.Function;

/**
* This is the main class for demonstrating the functionality of {@link
BigDecimalProcessor} implementations.
* It provides methods to demonstrate the functionality of {@link
BigDecimalProcessor} implementations such as
* filling a list with random BigDecimal values, sorting the list in
descending order by absolute value,
* calculating the product of positive numbers in the list, and printing
the list.
* <p> The class uses Java's functional interfaces, Consumer and
Function, to accept the methods as parameters
* and execute them. This allows for a high degree of flexibility, as
any method that fits the signature
* of the functional interfaces can be passed in and executed.
*/
public class BigDecimalProcessorDemo {
   /** The upper limit for generating random BigDecimal values. */
   public static int BOUND = 200;

   /** The scale to be used for the BigDecimal product calculation, the
number of digits to the right of the decimal point. */
   public static int PRODUCT_SCALE = 10;

   /** The size of the list to be filled with random BigDecimal values.
*/
   public static int LIST_SIZE = 4;

   /**
    * Demonstrates the methods of a {@link BigDecimalProcessor}
implementation.
    * @param processor the {@link BigDecimalProcessor} implementation;
    * @param ListModifier a Consumer that modifies a list of BigDecimal
values;
    * @param ProductCalculator a Function that calculates the product
of positive numbers in a list;
    * @param ListPrinter a Consumer that prints a list of BigDecimal
values.
    */
   public static void demonstrateMethods(BigDecimalProcessor processor,
                                         Consumer<List<BigDecimal>>
ListModifier,
                                         Function<List<BigDecimal>,
BigDecimal> ProductCalculator,
                                         Consumer<List<BigDecimal>>
ListPrinter) {
       List<BigDecimal> list = new ArrayList<>();
       processor.fillWithRandomValues(list, LIST_SIZE);
       System.out.println("List after filling with random values:");
       ListPrinter.accept(list);

       ListModifier.accept(list);
       System.out.println("\nList after sorting by absolute value in
descending order:");
       ListPrinter.accept(list);

       BigDecimal product = ProductCalculator.apply(list);
       product = product.setScale(PRODUCT_SCALE, RoundingMode.HALF_UP);
       System.out.println("\nProduct of positive numbers: " + product);
   }

   /**
    * The main method for this class. Creates instances of {@link
BigDecimalProcessor} implementations
    * and demonstrates their methods. The {@code args} are not used.
    * @param args the command-line arguments.
    */
   public static void main(String[] args) {
       BigDecimalProcessorWithLoops processor1 = new
BigDecimalProcessorWithLoops(BOUND);
       BigDecimalProcessorWithCollection processor2 = new
BigDecimalProcessorWithCollection(BOUND);
       BigDecimalProcessorWithStreams processor3 = new
BigDecimalProcessorWithStreams(BOUND);

       System.out.println("Demonstrating methods for
BigDecimalProcessorWithLoops:");
       demonstrateMethods(processor1,

BigDecimalProcessorWithLoops::sortDescendingByAbsoluteValue,

BigDecimalProcessorWithLoops::calculateProductOfPositiveNumbers,
               BigDecimalProcessorWithLoops::printList);
       System.out.println("\n\nDemonstrating methods for
BigDecimalProcessorWithCollection:");
       demonstrateMethods(processor2,

BigDecimalProcessorWithCollection::sortDescendingByAbsoluteValue,

BigDecimalProcessorWithCollection::calculateProductOfPositiveNumbers,
                BigDecimalProcessorWithCollection::printList);

        System.out.println("\n\nDemonstrating methods for
BigDecimalProcessorWithStreams:");
        demonstrateMethods(processor3,

BigDecimalProcessorWithStreams::sortDescendingByAbsoluteValue,

BigDecimalProcessorWithStreams::calculateProductOfPositiveNumbers,
                BigDecimalProcessorWithStreams::printList);
    }
}
```

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

3.3.1. BigDecimalProcessorTest.java:

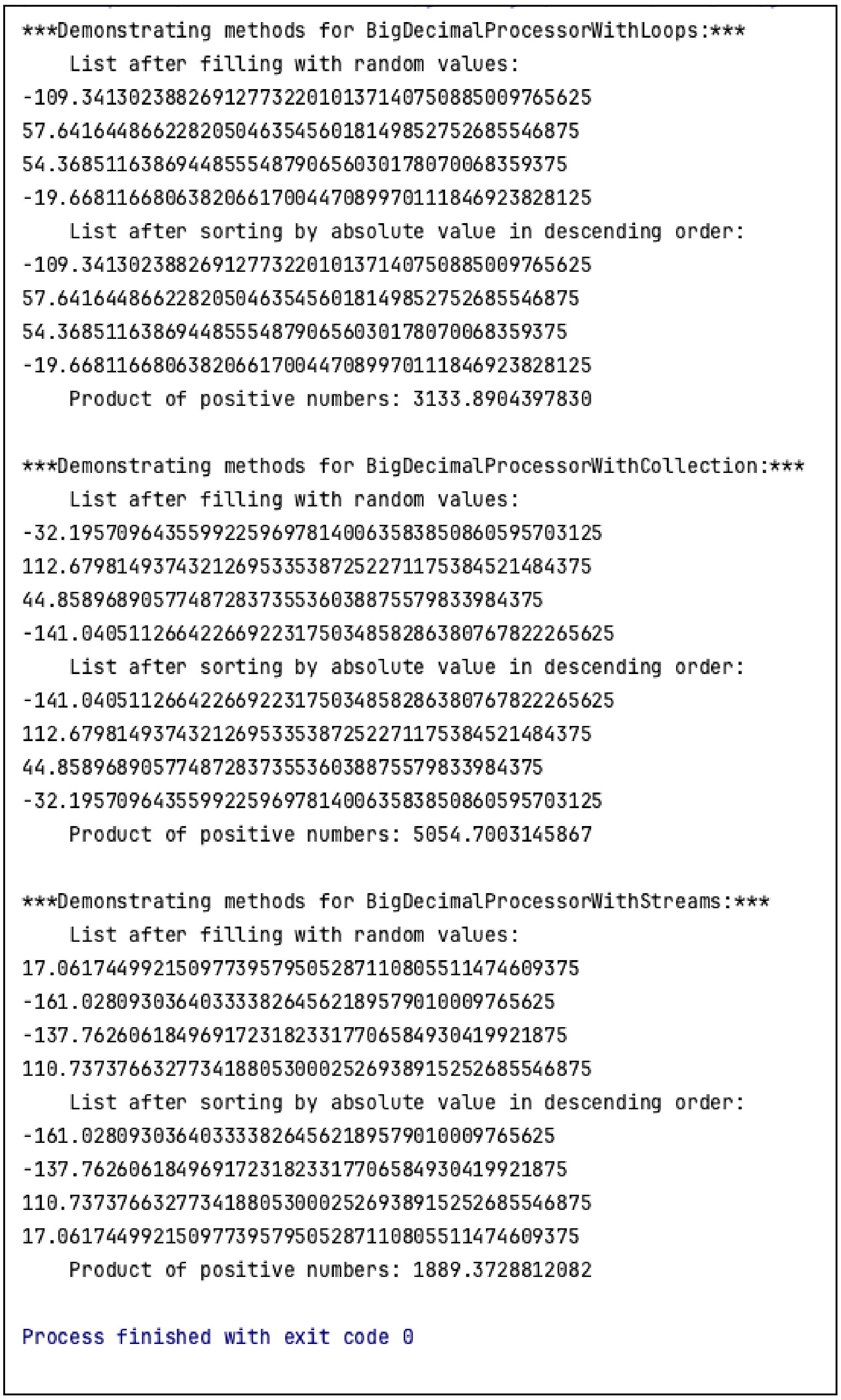
```
package part2.lab1.task3;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;
import java.util.function.Function;
import java.util.stream.Stream;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

/**
* The {@code BigDecimalProcessorTest} class provides unit tests for the
implementations of {@link BigDecimalProcessor}
* interface. It tests the generation of random BigDecimal values,
sorting, the calculation of products positive values
* using different methods, and the formatting of BigDecimal values for
```

```
display.
* @see BigDecimalProcessorWithLoops ;
* @see BigDecimalProcessorWithCollection ;
* @see BigDecimalProcessorWithStreams .
*/
class BigDecimalProcessorTest {
   private static final BigDecimal FIRST_POSITIVE_VALUE = new
BigDecimal("1.2");

   private static final BigDecimal SECOND_NEGATIVE_VALUE = new
BigDecimal("-3.7");

   private static final BigDecimal THIRD_POSITIVE_VALUE = new
BigDecimal("4.3");

   private static final BigDecimal FOURTH_NEGATIVE_VALUE = new
BigDecimal("-4.7");

   private static final BigDecimal FIFTH_NEGATIVE_VALUE = new
BigDecimal("-7.1");

   private static final BigDecimal PRODUCT_OF_FIRST_AND_THIRD = new
BigDecimal("5.16");

   /** The size of the list to be filled with random BigDecimal values.
*/
   private static final int SIZE = 10;

   /** The bound for the random BigDecimal values. */
   private static final int BOUND = 100;

   private List<BigDecimal> list;

   @BeforeEach
   public void setUp() {
       list = new ArrayList<>();
   }

   private static Stream<Consumer<List<BigDecimal>>>
sortDescendingByAbsoluteValueProvider() {
       return Stream.of(

BigDecimalProcessorWithLoops::sortDescendingByAbsoluteValue,

BigDecimalProcessorWithCollection::sortDescendingByAbsoluteValue,

BigDecimalProcessorWithStreams::sortDescendingByAbsoluteValue
       );
   }
   /**
    * The {@code FillWithRandomValuesTests} class provides unit tests
for the implementations
    * {@link BigDecimalProcessor#fillWithRandomValues} method. It tests
the generation of random BigDecimal values
    * and checks if the list is filled with the correct number of
elements.
    */
   @Nested
   class FillWithRandomValuesTests {
       private static Stream<BigDecimalProcessor> processorProvider() {
           return Stream.of(
                   new BigDecimalProcessorWithLoops(BOUND),
                   new BigDecimalProcessorWithCollection(BOUND),
                   new BigDecimalProcessorWithStreams(BOUND)
           );
       }

       /**
        * Tests the implementations of {@link
BigDecimalProcessor#fillWithRandomValues} method.
        * Checks if the list is filled with the correct number of
elements.
        * @param processor the BigDecimalProcessor to be tested.
        */
       @ParameterizedTest(name = "Test #{index}")
       @DisplayName("Should ensure that the list is filled with the
correct number of elements with 3 different implementations")
       @MethodSource("processorProvider")
       public void testFillWithRandomValuesSize(BigDecimalProcessor
processor) {
           processor.fillWithRandomValues(list, SIZE);
           assertEquals(SIZE, list.size());
       }

       /**
        * Tests the implementations of {@link
BigDecimalProcessor#fillWithRandomValues method.
        * Checks if the range of values is correct.
        * @param processor the BigDecimalProcessor to be tested.
        */
       @ParameterizedTest(name = "Test #{index}")
       @DisplayName("Should ensure that range of values is correct with
3 different implementations")
       @MethodSource("processorProvider")
       public void
testFillWithRandomValuesRangeReplacesOldValues(BigDecimalProcessor
processor) {
```

```
            int repeatTimes = 3;

            for (int i = 0; i < repeatTimes; i++) {
                processor.fillWithRandomValues(list, SIZE);
                BigDecimal LOWER_BOUND = BigDecimal.valueOf(-BOUND);
                BigDecimal UPPER_BOUND = BigDecimal.valueOf(BOUND);
                assertTrue(list.stream().
                        allMatch(item -> item.compareTo(LOWER_BOUND) ==
1
                                && item.compareTo(UPPER_BOUND) == -1));
            }
        }
    }

    /**
     * Tests the implementations of {@link
BigDecimalProcessor#sortDescendingByAbsoluteValue} method.
     * Checks if the list is sorted in descending order by absolute
value.
     * @param sortDescendingByAbsoluteValue the method to be tested
     */
    @ParameterizedTest(name = "Test #{index}")
    @DisplayName("Should sort descending by absolute value with 3
different implementations")
    @MethodSource("sortDescendingByAbsoluteValueProvider")
    public void
testSortDescendingByAbsoluteValue(Consumer<List<BigDecimal>>
sortDescendingByAbsoluteValue) {
        list.add(FIRST_POSITIVE_VALUE);
        list.add(SECOND_NEGATIVE_VALUE);
        list.add(THIRD_POSITIVE_VALUE);
        sortDescendingByAbsoluteValue.accept(list);
        assertEquals(THIRD_POSITIVE_VALUE, list.get(0));
        assertEquals(SECOND_NEGATIVE_VALUE, list.get(1));
        assertEquals(FIRST_POSITIVE_VALUE, list.get(2));
    }

    /**
     * The {@code CalculateProductOfPositiveNumbersTests} class provides
unit tests for the implementations
     * of {@link BigDecimalProcessor#calculateProductOfPositiveNumbers}
method. It tests the calculation of
     * the product of positive numbers and checks if the result is
correct.
     */
    @Nested
    class CalculateProductOfPositiveNumbersTests {
        private static Stream<Function<List<BigDecimal>, BigDecimal>>
calculateProductOfPositiveNumbersProvider() {
```

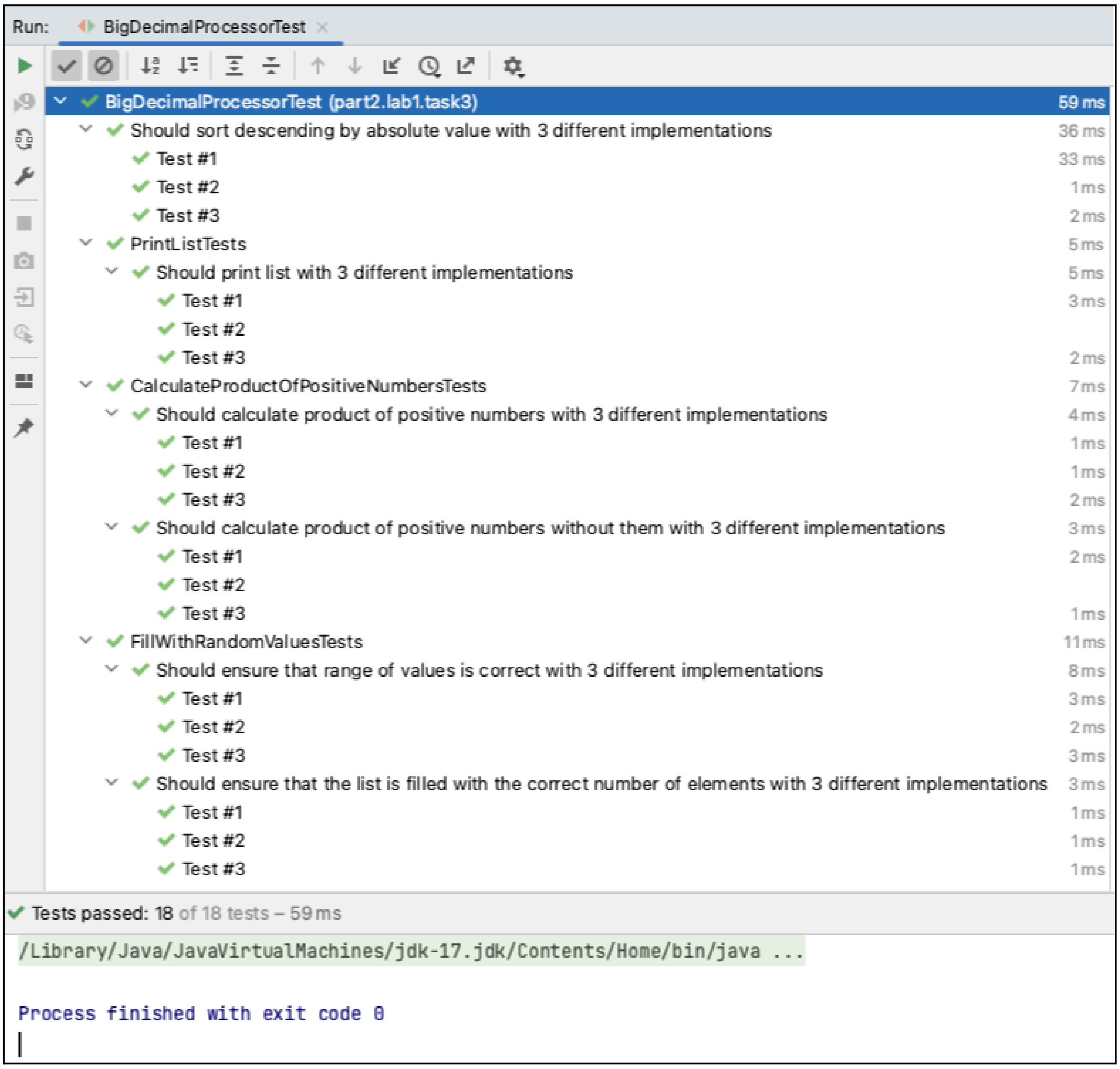
```
            return Stream.of(

BigDecimalProcessorWithLoops::calculateProductOfPositiveNumbers,

BigDecimalProcessorWithCollection::calculateProductOfPositiveNumbers,

BigDecimalProcessorWithStreams::calculateProductOfPositiveNumbers
            );
        }

        /**
         * Tests the implementations of {@link
BigDecimalProcessor#calculateProductOfPositiveNumbers} method.
         * Checks if the product of positive numbers is calculated
correctly.
         * @param calculateProductOfPositiveNumbers the method to be
tested.
         */
        @ParameterizedTest(name = "Test #{index}")
        @DisplayName("Should calculate product of positive numbers with
3 different implementations")
        @MethodSource("calculateProductOfPositiveNumbersProvider")
        public void testCalculateProductOfPositiveNumbers(
                Function<List<BigDecimal>, BigDecimal>
calculateProductOfPositiveNumbers) {
            list.add(FIRST_POSITIVE_VALUE);
            list.add(SECOND_NEGATIVE_VALUE);
            list.add(THIRD_POSITIVE_VALUE);
            BigDecimal product =
calculateProductOfPositiveNumbers.apply(list);
            assertEquals(PRODUCT_OF_FIRST_AND_THIRD, product);
        }

        /**
         * Tests the implementations of {@link
BigDecimalProcessor#calculateProductOfPositiveNumbers} method.
         * Checks if the product of positive numbers is calculated
correctly when there are no positive numbers.
         * @param calculateProductOfPositiveNumbers the method to be
tested.
         */
        @ParameterizedTest(name = "Test #{index}")
        @DisplayName("Should calculate product of positive numbers
without them with 3 different implementations")
        @MethodSource("calculateProductOfPositiveNumbersProvider")
        public void testCalculateProductWithoutPositiveNumbers(
                Function<List<BigDecimal>, BigDecimal>
calculateProductOfPositiveNumbers) {
            list.add(SECOND_NEGATIVE_VALUE);
            list.add(FOURTH_NEGATIVE_VALUE);
            list.add(FIFTH_NEGATIVE_VALUE);
            BigDecimal product =
calculateProductOfPositiveNumbers.apply(list);
            assertEquals(BigDecimal.ONE, product);
        }
    }

    /**
     * The {@code PrintListTests} class provides unit tests for the
implementations of
     * {@link BigDecimalProcessor#printList} method. It tests the
printing of the list and checks if the output is correct.
     * <p> ByteArrayOutputStream and PrintStream are used to capture
console output.
     * ByteArrayOutputStream stores the output in a byte array, which
allows easy checking of the output content later.
     * PrintStream is used with System.setOut to redirect output, which
usually goes to the console, to ByteArrayOutputStream.
     */
    @Nested
    class PrintListTests {
        private final ByteArrayOutputStream outContent = new
ByteArrayOutputStream();

        private final PrintStream originalOut = System.out;

        /**
         * Sets up the streams before each test.
         * Redirect output, which usually goes to the console, to
ByteArrayOutputStream
         */
        @BeforeEach
        public void setUpStreams() {
            System.setOut(new PrintStream(outContent));
        }

        /**
         * Provides a stream of Consumers that print a list of
BigDecimal values.
         * Each Consumer is a method reference to a different
implementation of the printList method.
         * @return a stream of Consumers.
         */
        private static Stream<Consumer<List<BigDecimal>>>
printListProvider() {
            return Stream.of(BigDecimalProcessorWithLoops::printList,
                    BigDecimalProcessorWithCollection::printList,
                    BigDecimalProcessorWithStreams::printList);
        }

        /**
         * Tests the printList method of different BigDecimalProcessor
implementations. Checks if the list is
         * printed correctly. The split(System.lineSeparator()) method
is used to split this string
         * into an array of strings (lines), using the system's line
separator as the delimiter.
         * This allows the lines to be printed properly on any OS.
         * @param printList a Consumer that prints a list of BigDecimal
values.
         */
        @ParameterizedTest(name = "Test #{index}")
        @DisplayName("Should print list with 3 different
implementations")
        @MethodSource("printListProvider")
        public void testPrintList(Consumer<List<BigDecimal>> printList)
{
            List<BigDecimal> list = new ArrayList<>();
            list.add(FIRST_POSITIVE_VALUE);
            list.add(SECOND_NEGATIVE_VALUE);
            list.add(THIRD_POSITIVE_VALUE);

            printList.accept(list);
            String[] lines =
outContent.toString().split(System.lineSeparator());
            assertEquals(FIRST_POSITIVE_VALUE.toString(), lines[0]);
            assertEquals(SECOND_NEGATIVE_VALUE.toString(), lines[1]);
            assertEquals(THIRD_POSITIVE_VALUE.toString(), lines[2]);
        }

        /**
         * Restores the standard output to the console after each test.
         * <p> Note: ByteArrayOutputStream does not require closing.
         * It does not use any system resources that require closing, so
it can be safely left open.
         * It differs from other I/O streams, such as FileOutputStream
or PrintStream, which use system resources,
         * such as files or network connections, and therefore require
closing to free these resources.
         */
        @AfterEach
        public void restoreStreams() {
            System.setOut(originalOut);
        }
    }
}
```

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



```
***Demonstrating methods for BigDecimalProcessorWithLoops:***
    List after filling with random values:
-109.3413023882691277322010137140750885009765625
57.64164486622820504635456018149852752685546875
54.3685116386944855548790656030178070068359375
-19.66811668063820661700447089970111846923828125
    List after sorting by absolute value in descending order:
-109.3413023882691277322010137140750885009765625
57.64164486622820504635456018149852752685546875
54.3685116386944855548790656030178070068359375
-19.66811668063820661700447089970111846923828125
    Product of positive numbers: 3133.8904397830

***Demonstrating methods for BigDecimalProcessorWithCollection:***
    List after filling with random values:
-32.195709643559922596978140063583850860595703125
112.67981493743212695335387252271175384521484375
44.85896890577487283735536038875579833984375
-141.040511266422669223175034858286380767822265625
    List after sorting by absolute value in descending order:
-141.040511266422669223175034858286380767822265625
112.67981493743212695335387252271175384521484375
44.85896890577487283735536038875579833984375
-32.195709643559922596978140063583850860595703125
    Product of positive numbers: 5054.7003145867

***Demonstrating methods for BigDecimalProcessorWithStreams:***
    List after filling with random values:
17.06174499215097739579505287110805511474609375
-161.0280930364033338264562189579010009765625
-137.76260618496917231823317706584930419921875
110.73737663277341880530002526938915252685546875
    List after sorting by absolute value in descending order:
-161.0280930364033338264562189579010009765625
-137.76260618496917231823317706584930419921875
110.73737663277341880530002526938915252685546875
17.06174499215097739579505287110805511474609375
    Product of positive numbers: 1889.3728812082

Process finished with exit code 0
```

Рисунок 3.4.1 – Результати №1 роботи програмного коду класу BigDecimalProcessorDemo.java



```
Run:   BigDecimalProcessorTest ×
✓ BigDecimalProcessorTest (part2.lab1.task3)                                                          59 ms
  ✓ Should sort descending by absolute value with 3 different implementations                         36 ms
    ✓ Test #1                                                                                          33 ms
    ✓ Test #2                                                                                           1 ms
    ✓ Test #3                                                                                           2 ms
  ✓ PrintListTests                                                                                      5 ms
    ✓ Should print list with 3 different implementations                                                5 ms
      ✓ Test #1                                                                                         3 ms
      ✓ Test #2
      ✓ Test #3                                                                                         2 ms
  ✓ CalculateProductOfPositiveNumbersTests                                                              7 ms
    ✓ Should calculate product of positive numbers with 3 different implementations                     4 ms
      ✓ Test #1                                                                                         1 ms
      ✓ Test #2                                                                                         1 ms
      ✓ Test #3                                                                                         2 ms
    ✓ Should calculate product of positive numbers without them with 3 different implementations        3 ms
      ✓ Test #1                                                                                         2 ms
      ✓ Test #2
      ✓ Test #3                                                                                         1 ms
  ✓ FillWithRandomValuesTests                                                                          11 ms
    ✓ Should ensure that range of values is correct with 3 different implementations                    8 ms
      ✓ Test #1                                                                                         3 ms
      ✓ Test #2                                                                                         2 ms
      ✓ Test #3                                                                                         3 ms
    ✓ Should ensure that the list is filled with the correct number of elements with 3 different implementations   3 ms
      ✓ Test #1                                                                                         1 ms
      ✓ Test #2                                                                                         1 ms
      ✓ Test #3                                                                                         1 ms
✓ Tests passed: 18 of 18 tests – 59 ms
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

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

4. Завдання №4 до лабораторної роботи

4.1. Пошук усіх дільників (додаткове завдання)

З використанням засобів Stream API організувати пошук усіх дільників цілого додатного числа. Створити окрему статичну функцію, яка приймає ціле число і повертає масив цілих чисел. Всередині функції створювати потік IntStream. Застосувати функцію range() і фільтр. Не використовувати явних циклів.

4.2. Програмний код реалізації завдання № 4

4.2.1. DivisiorFinder.java:

```
package part2.lab1.task4;

import java.util.stream.IntStream;

/**
* The {@code DivisorFinder} class provides a method for finding all
positive divisors of a number.
* <p> The {@link #findAllPositiveNumberDivisors(int)} method takes a
positive integer as input and
* returns an array of all its positive divisors.
*/
public class DivisorFinder {
   /**
    * Finds all positive divisors of a number. It uses the Java 8
Stream API to generate a range of numbers
    * from 1 to the input number (inclusive), and then filters this
stream to include only those numbers
    * that divide the input number without a remainder. The resulting
stream is then converted to an array.
    * @param number the number to find the divisors of, must be
positive;
    * @return an array of all positive divisors of the number;
    * @throws IllegalArgumentException if the number is not positive.
    */
   public static int[] findAllPositiveNumberDivisors(int number) {
       if (number <= 0) {
           throw new IllegalArgumentException("Number must be
positive");
       }

       return IntStream.range(1, number + 1)
               .filter(i -> number % i == 0)
               .toArray();
   }
}
```

4.2.2. DivisiorFinderDemo.java:

```
package part2.lab1.task4;

import java.util.Arrays;

/**
* The {@code DivisorFinderDemo} class is the main entry point for the
```

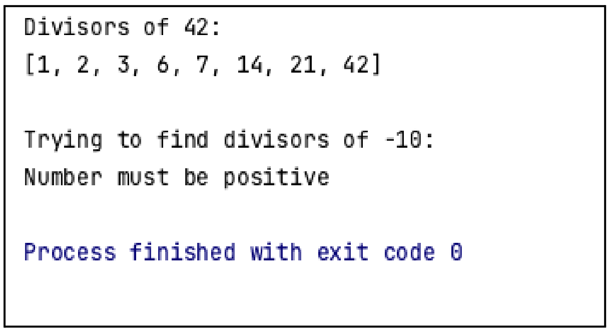
```
Divisor Finder application.
* <p> This class demonstrates the usage of the {@link
DivisorFinder#findAllPositiveNumberDivisors(int)} method
* by finding all positive divisors of a valid number and handling an
exception for an invalid number.
* <p> The {@link #main(String[])} method first prints all positive
divisors of a valid number.
* It then attempts to find divisors of an invalid number, catching and
printing the resulting IllegalArgumentException.
*/
public class DivisorFinderDemo {
   /** A valid number for which to find all positive divisors. */
   public static final int VALID_NUMBER = 42;

   /** An invalid number for which the DivisorFinder should throw an
IllegalArgumentException. */
   public static final int INVALID_NUMBER = -10;

   /**
    * Carries out showing of the functionality of the {@link
DivisorFinder}. The {@code args} are not used.
    * @param args the command-line arguments (not used).
    */
   public static void main(String[] args) {
       System.out.println("Divisors of " + VALID_NUMBER + ":");
       int[] result =
DivisorFinder.findAllPositiveNumberDivisors(VALID_NUMBER);
       System.out.println(Arrays.toString(result));

       try {
           System.out.println("\nTrying to find divisors of " +
INVALID_NUMBER + ":");
           DivisorFinder.findAllPositiveNumberDivisors(INVALID_NUMBER);
       } catch (IllegalArgumentException e) {
           System.out.println(e.getMessage());
       }
   }
}
```

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

4.3.1. DivisiorFinderTest.java:

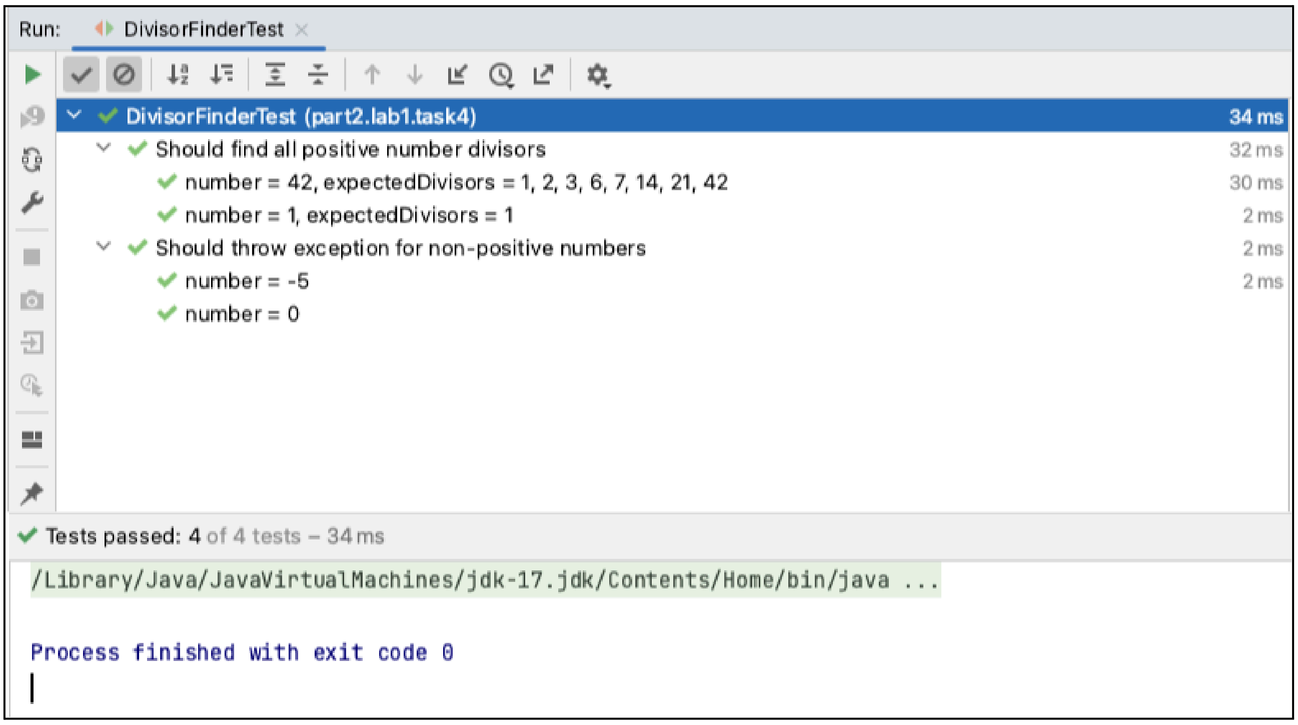
```
package part2.lab1.task4;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;

import java.util.Arrays;

import static org.junit.jupiter.api.Assertions.assertArrayEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

/** The {@code DivisorFinderTest} class provides unit tests for the
{@link DivisorFinder} class. */
class DivisorFinderTest {
   /**
    * Tests the {@link
DivisorFinder#findAllPositiveNumberDivisors(int)} method with positive
numbers.
    * <p> It uses a {@link CsvSource} to provide a set of numbers and
their expected divisors.
    * @param number the number to find divisors for;
    * @param expectedString a string representation of the expected
divisors of the number.
    */
   @ParameterizedTest(name = "number = {0}, expectedDivisors = {1}")
   @DisplayName("Should find all positive number divisors")
   @CsvSource({ "42, '1, 2, 3, 6, 7, 14, 21, 42'", "1, 1" })
   public void testFindAllPositiveNumberDivisors(int number, String
expectedString) {
       int[] expectedDivisors = Arrays.stream(
               expectedString.split(", "))
               .mapToInt(Integer::parseInt)
               .toArray();
       int[] actualDivisors =
DivisorFinder.findAllPositiveNumberDivisors(number);
       assertArrayEquals(expectedDivisors, actualDivisors);
   }

   /**
    * Tests the {@link
DivisorFinder#findAllPositiveNumberDivisors(int)} method with
non-positive numbers.
    * <p> This test checks if the method correctly throws an {@link
IllegalArgumentException}
    * when the input number is not positive. It uses a {@link
ValueSource} to provide a set of non-positive numbers.
    * @param number the non-positive number to find the divisors for.
    */
   @ParameterizedTest(name = "number = {0}")
   @DisplayName("Should throw exception for non-positive numbers")
   @ValueSource(ints = {-5, 0})
   public void testFindAllPositiveNumberDivisorsThrowsException(int
number) {
        assertThrows(IllegalArgumentException.class,
                () →
DivisorFinder.findAllPositiveNumberDivisors(number));
   }
}
```

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



```
Divisors of 42:
[1, 2, 3, 6, 7, 14, 21, 42]

Trying to find divisors of -10:
Number must be positive

Process finished with exit code 0
```

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



```
DivisorFinderTest (part2.lab1.task4)                                        34 ms
  Should find all positive number divisors                                  32 ms
    number = 42, expectedDivisors = 1, 2, 3, 6, 7, 14, 21, 42               30 ms
    number = 1, expectedDivisors = 1                                         2 ms
  Should throw exception for non-positive numbers                            2 ms
    number = -5                                                              2 ms
    number = 0
Tests passed: 4 of 4 tests – 34 ms
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Рисунок 4.4.2 – Результати модульного тестування з використанням Junit програмного коду класу DivisorFinderTest.java

5. Вправа для контролю до лабораторної роботи

5.1. Умова завдання

Реалізувати функцію отримання цілої частини квадратного кореня з числа типу BigInteger.

5.2. Програмний код реалізації вправи для контролю

5.2.1. BigIntegerSquareRoot.java:

```
package part2.lab1.task_control;

import java.math.BigInteger;
import java.util.Random;
import java.util.function.Function;
import java.util.stream.IntStream;

/**
* This class provides methods for working with BigInteger numbers,
* including generating a random BigInteger, finding the integer square
root of BigInteger,
* and comparing the results and durations of different methods of
finding the integer square root of BigInteger.
*/
public class BigIntegerSquareRoot {
   /** The number of digits of the random BigInteger number. */
   public static final int[] NUM_DIGITS = new int[] {10, 42, 90};

   /** The maximum number of digits to be displayed in the console. */
   public static final int MAX_DISPLAY_DIGITS = 100;

   /**
     * Generates a random BigInteger array with the number of digits
equal to the length of
     * the '{@code NUM_DIGITS}' array of digits.
     * @return an array of random BigInteger numbers.
     */
   public static BigInteger[] generateRandomBigIntegerArray() {
       Random random = new Random();

       return IntStream.range(0, NUM_DIGITS.length)
               .mapToObj(j -> {
                   StringBuilder sb = new StringBuilder(NUM_DIGITS[j]);
                   sb.append(random.nextInt(9) + 1);
                   IntStream.range(0, NUM_DIGITS[j])
                           .forEach(i -> {
                               sb.append(random.nextInt(10));
                           });
```

```
                    return new BigInteger(sb.toString());
                })
                .toArray(BigInteger[]::new);
    }

    /**
     * Finds the integer square root of BigInteger using the sqrt
function {@link BigInteger#sqrt()}.
     * @param number the BigInteger number;
     * @return the integer square root of BigInteger number.
     */
    public static BigInteger findSqrtUsingSqrtFunction(BigInteger
number) {
        return number.sqrt();
    }

    /**
     * Finds the integer square root of BigInteger using binary search
to efficiently find the root
     * using method {@link BigInteger#multiply(BigInteger)}.
     * @param number the BigInteger number;
     * @return the integer square root of BigInteger number;
     * @throws ArithmeticException {@code number} is negative
     * (this would cause the operation to yield a non-real value "i").
     */
    public static BigInteger findSqrtUsingMultiplication(BigInteger
number) {
        if (number.compareTo(BigInteger.ZERO) < 0) {
            throw new ArithmeticException("Negative number");
        }

        BigInteger left = BigInteger.ZERO;
        BigInteger right = number;

        while (left.compareTo(right) <= 0) {
            BigInteger mid = left.add(right).shiftRight(1);
            BigInteger midSquared = mid.multiply(mid);

            if (midSquared.compareTo(number) == 0) {
                return mid;
            } else if (midSquared.compareTo(number) < 0) {
                left = mid.add(BigInteger.ONE);
            } else {
                right = mid.subtract(BigInteger.ONE);
            }
        }

        return right;
    }
```

```
   /** A class to hold a result and the duration it took to compute
that result. */
   public static class ResultAndDurationPair {
       private final BigInteger result;
       private final long duration;

       public ResultAndDurationPair(BigInteger result, long duration) {
           this.result = result;
           this.duration = duration;
       }

       public BigInteger getResult() {
           return result;
       }

       public long getDuration() {
           return duration;
       }
   }

   /**
    * Calculates the result of a function and measures the time it took
to compute the result.
    * @param number the BigInteger number;
    * @param function {@code Function} the function to apply to find
the integer square root;
    * @return a ResultAndDurationPair containing the result and the
duration it took to compute the result.
    */
   public static ResultAndDurationPair
calculateAndMeasureTime(BigInteger number,

Function<BigInteger, BigInteger> function) {
       long startTime = System.nanoTime();
       BigInteger result = function.apply(number);
       long endTime = System.nanoTime();
       long duration = endTime - startTime;

       return new ResultAndDurationPair(result, duration);
   }

   /**
    * Formats a BigInteger for display, truncating it if it has more
than '{@value MAX_DISPLAY_DIGITS}' digits.
    * @param number the BigInteger to format;
    * @return a string representation of the BigInteger, truncated if
necessary.
    */
   public static String formatBigInteger(BigInteger number) {
       String str = number.toString();

       if (str.length() > MAX_DISPLAY_DIGITS) {
           str = str.substring(0, MAX_DISPLAY_DIGITS) + "...";
       }

       return str;
   }

   /**
    * Compares two BigIntegers and prints a message indicating whether
they are equal.
    * @param resultUsingSqrt the first BigInteger to compare;
    * @param resultUsingMultiplication the second BigInteger to
compare.
    */
   public static void printComparingResults(BigInteger resultUsingSqrt,
                                            BigInteger
resultUsingMultiplication) {
       if (resultUsingSqrt.equals(resultUsingMultiplication)) {
           System.out.println("The results are equal.");
       } else {
           System.out.println("The results are not equal.");
       }
   }

   /**
    * Compares two durations and prints a message indicating which is
shorter.
    * @param durationUsingSqrt the first duration to compare;
    * @param durationUsingMultiplication the second duration to
compare.
    */
   public static void printComparingDurations(long durationUsingSqrt,
                                              long
durationUsingMultiplication) {
       if (durationUsingSqrt < durationUsingMultiplication) {
           System.out.println("Sqrt function is faster.");
       } else if (durationUsingSqrt > durationUsingMultiplication) {
           System.out.println("Multiplication is faster.");
       } else {
           System.out.println("Both methods took the same amount of
time.");
       }
   }
}
```

5.2.2. BigIntegerSquareRootDemo.java:

```
package part2.lab1.task_control;

import java.math.BigInteger;
import java.util.stream.IntStream;

import static part2.lab1.task_control.BigIntegerSquareRoot.*;

/**
* The {@code BigIntegerSquareRootDemo} class represents showing of the
functionality of the {@link BigIntegerSquareRoot}.
* It generates array with random BigInteger numbers, finds the integer
square root of BigInteger
* using two different methods and compares the results and the time
taken by each method.
*/
public class BigIntegerSquareRootDemo {
   /**
    * Carries out showing of the functionality of the {@link
BigIntegerSquareRoot}. The {@code args} are not used.
    * @param args the command-line arguments (not used).
    */
   public static void main(String[] args) {
       BigInteger[] array = generateRandomBigIntegerArray();
       IntStream.range(0, array.length).forEach(i -> {
           System.out.println("\n***The number with randomly generated
digits to find the integer square roots:***");
           System.out.println("Number of digits = " + NUM_DIGITS[i]
                   + ",\tBigInteger number = " + array[i]);

           try {
               ResultAndDurationPair resultUsingSqrt =
                       calculateAndMeasureTime(array[i],
BigIntegerSquareRoot::findSqrtUsingSqrtFunction);
               System.out.println("Result using sqrt function: " +
formatBigInteger(resultUsingSqrt.getResult())
                       + "\nTime taken using sqrt function: " +
resultUsingSqrt.getDuration() + " nanoseconds");

               ResultAndDurationPair resultUsingMultiplication =
                       calculateAndMeasureTime(array[i],
BigIntegerSquareRoot::findSqrtUsingMultiplication);
               System.out.println("Result using multiplication
function: "
                       +
formatBigInteger(resultUsingMultiplication.getResult())
                       + "\nTime taken using multiplication function: "
                       + resultUsingMultiplication.getDuration() + "
nanoseconds");
```

```
                System.out.println("Conclusions:");
                printComparingResults(resultUsingSqrt.getResult(),
resultUsingMultiplication.getResult());
                printComparingDurations(resultUsingSqrt.getDuration(),
resultUsingMultiplication.getDuration());
            } catch (ArithmeticException e){
                System.err.println(e.getMessage());
            }
        });
    }
}
```

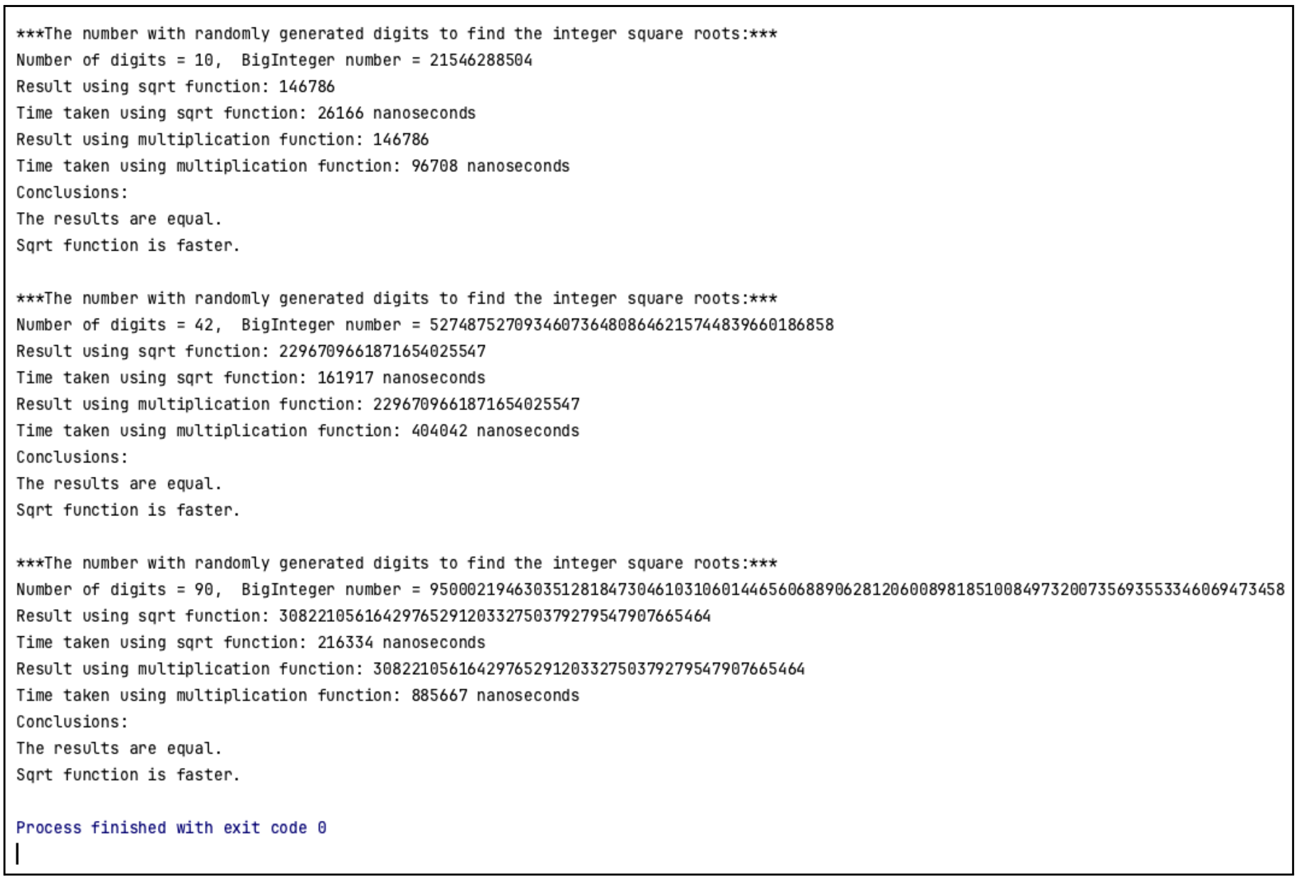
5.3. Програмний код модульного тестування з використанням Junit вправи для контролю

5.3.1. BigIntegerSquareRootTest.java:

```
package part2.lab1.task_control;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;

import java.math.BigInteger;
import java.util.Arrays;
import java.util.function.Function;
import java.util.stream.IntStream;
import java.util.stream.Stream;

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

/**
* The {@code BigIntegerSquareRootTest} class provides unit tests for
the {@link BigIntegerSquareRoot} class.
* It tests the generation of random BigIntegers, the calculation of
integer square root using two different methods,
* and the formatting of BigIntegers for display.
*/
class BigIntegerSquareRootTest {
   /** A constant array representing the BigInteger numbers with
different count of digits used in the tests. */
   private static final BigInteger[] BIG_INTEGER_NUMBERS = {
           new BigInteger("-1"),
           BigInteger.ZERO,
           BigInteger.ONE,
           new BigInteger("1361686103"),
           new
BigInteger("847039021550742412994880486347242064750832"),
           new
BigInteger("44693226483775587175566563590914066665235907772085336118184
55247415429729998617407903644 29")
   };

   /** A constant representing the array of the integer square root of
BigInteger numbers
    * for the {@code BIG_INTEGER_NUMBERS} array of the BigInteger
numbers. */
   private static final BigInteger[] BIG_INTEGER_SQRTS = {
           null,
           new BigInteger("0"),
           new BigInteger("1"),
           new BigInteger("36901"),
           new BigInteger("92034722879505776941 5"),
           new
BigInteger("668529928154122260725410609185195087552565391")
   };

   /**
    * Provides data for the integer square root calculation tests.
    * @return a stream of arguments for the integer square root
calculation tests.
    */
   private static Stream<Arguments>
provideDataForSqrtCalculationTests() {
       Function<BigInteger, BigInteger> sqrtFunction =
BigIntegerSquareRoot::findSqrtUsingSqrtFunction;
       Function<BigInteger, BigInteger> multiplicationFunction =
BigIntegerSquareRoot::findSqrtUsingMultiplication;

       return Stream.concat(
               Stream.of(BIG_INTEGER_NUMBERS).map(number ->
Arguments.of(sqrtFunction, number)),
               Stream.of(BIG_INTEGER_NUMBERS).map(number ->
Arguments.of(multiplicationFunction, number))
       );
   }

   /**
    * Tests the {@link
BigIntegerSquareRoot#generateRandomBigIntegerArray()} method.
    * Checks if the generated BigInteger has the expected number of
digits.
    */
   @Test
   @DisplayName("Should generate BigInteger with the expected number of
digits")
   public void generateRandomBigIntegerArray() {
       BigInteger[] result =
BigIntegerSquareRoot.generateRandomBigIntegerArray();
       int[] expectedNumDigits = BigIntegerSquareRoot.NUM_DIGITS;
       int[] actualNumDigits = Arrays.stream(result).mapToInt(number →
number.toString().length() - 1).toArray();

       assertArrayEquals(expectedNumDigits, actualNumDigits);
   }

   /** Tests the calculation of integer square roots using two
different methods. */
   @ParameterizedTest(name = "Test {index}")
   @DisplayName("Should calculate the integer square root. " +
           "First 7 tests using function findSqrtUsingSqrtFunction,
second 7 tests - findSqrtUsingMultiplication")
   @MethodSource("provideDataForSqrtCalculationTests")
   public void testSqrtCalculation(Function<BigInteger, BigInteger>
function) {
       IntStream.range(0, BIG_INTEGER_NUMBERS.length)
               .forEach(i → {
                   BigInteger expectedNumber = BIG_INTEGER_SQRTS[i];

                   if
(BIG_INTEGER_NUMBERS[i].compareTo(BigInteger.ZERO) < 0) {
                       assertThrows(ArithmeticException.class, () →
function.apply(BIG_INTEGER_NUMBERS[i]));
                   } else {
                       BigInteger actual =
function.apply(BIG_INTEGER_NUMBERS[i]);
                       assertEquals(BIG_INTEGER_SQRTS[i], actual);
                   }
               });
   }

   /** Tests the {@link
BigIntegerSquareRoot#formatBigInteger(BigInteger)} method. */
   @Test
   @DisplayName("Should format BigInteger correctly")
   public void formatBigInteger() {
       IntStream.range(4, BIG_INTEGER_NUMBERS.length)
               .forEach(i → {
                   String actualString =
BigIntegerSquareRoot.formatBigInteger(BIG_INTEGER_NUMBERS[i]);
                   int actualLength = actualString.length();
                   int expectedLength =
BIG_INTEGER_NUMBERS[i].toString().length();
```

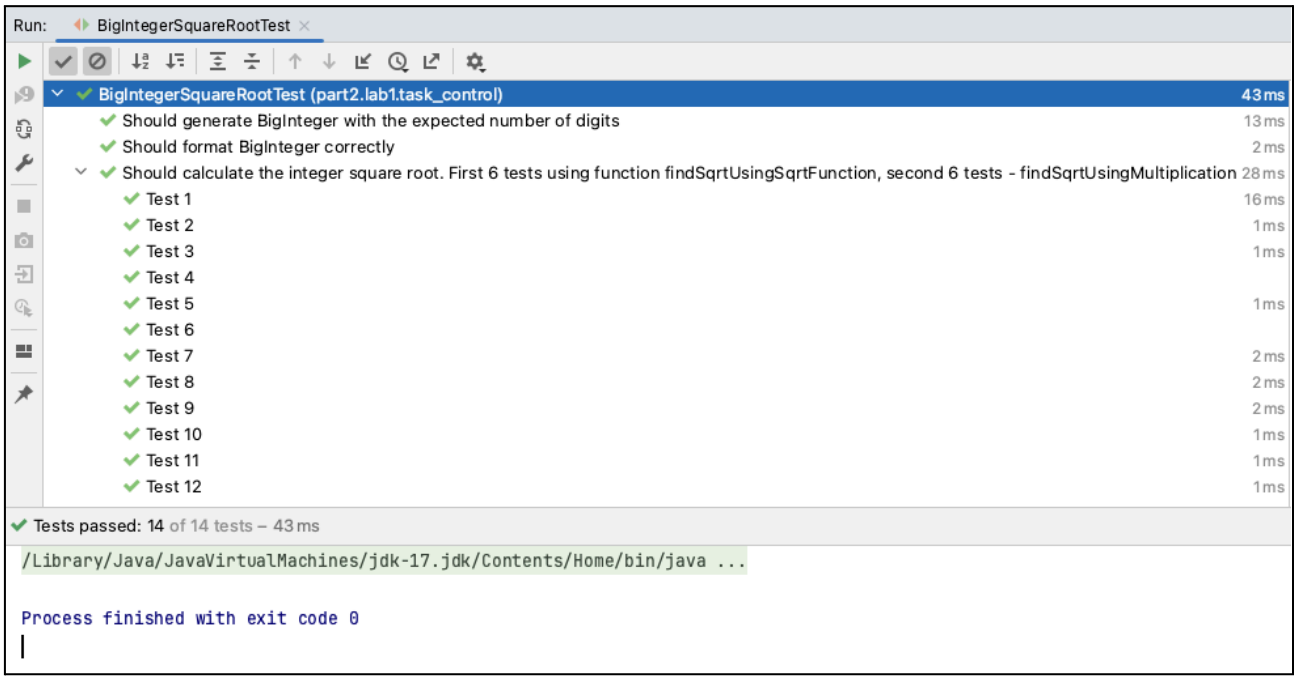
```
                        if (BIG_INTEGER_NUMBERS[i].toString().length() >
BigIntegerSquareRoot.MAX_DISPLAY_DIGITS) {
                            expectedLength =
BigIntegerSquareRoot.MAX_DISPLAY_DIGITS + 3;
                        }

                        assertEquals(expectedLength, actualLength);
                    });
    }
}
```

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



```
***The number with randomly generated digits to find the integer square roots:***
Number of digits = 10,  BigInteger number = 21546288504
Result using sqrt function: 146786
Time taken using sqrt function: 26166 nanoseconds
Result using multiplication function: 146786
Time taken using multiplication function: 96708 nanoseconds
Conclusions:
The results are equal.
Sqrt function is faster.

***The number with randomly generated digits to find the integer square roots:***
Number of digits = 42,  BigInteger number = 5274875270934607364808646215744839660186858
Result using sqrt function: 2296709661871654025547
Time taken using sqrt function: 161917 nanoseconds
Result using multiplication function: 2296709661871654025547
Time taken using multiplication function: 404042 nanoseconds
Conclusions:
The results are equal.
Sqrt function is faster.

***The number with randomly generated digits to find the integer square roots:***
Number of digits = 90,  BigInteger number = 9500021946303512818473046103106014465606889062812060089818510084973200735693553346069473458
Result using sqrt function: 3082210561642976529120332750379279547907665464
Time taken using sqrt function: 216334 nanoseconds
Result using multiplication function: 3082210561642976529120332750379279547907665464
Time taken using multiplication function: 885667 nanoseconds
Conclusions:
The results are equal.
Sqrt function is faster.

Process finished with exit code 0
|
```

Рисунок 5.4.1 – Результати №1 роботи програмного коду класу BigIntegerSquareRootDemo.java



```
Run:  BigIntegerSquareRootTest
✓ BigIntegerSquareRootTest (part2.lab1.task_control)                                                                              43 ms
    ✓ Should generate BigInteger with the expected number of digits                                                               13 ms
    ✓ Should format BigInteger correctly                                                                                           2 ms
    ✓ Should calculate the integer square root. First 6 tests using function findSqrtUsingSqrtFunction, second 6 tests - findSqrtUsingMultiplication 28 ms
        ✓ Test 1                                                                                                                  16 ms
        ✓ Test 2                                                                                                                   1 ms
        ✓ Test 3                                                                                                                   1 ms
        ✓ Test 4
        ✓ Test 5                                                                                                                   1 ms
        ✓ Test 6
        ✓ Test 7                                                                                                                   2 ms
        ✓ Test 8                                                                                                                   2 ms
        ✓ Test 9                                                                                                                   2 ms
        ✓ Test 10                                                                                                                  1 ms
        ✓ Test 11                                                                                                                  1 ms
        ✓ Test 12                                                                                                                  1 ms
✓ Tests passed: 14 of 14 tests – 43 ms
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

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

6. Висновки до лабораторної роботи

Під час виконання лабораторної роботи №1 з програмування, були здобуті практичні навички у роботі з Stream API та JUnit, що включало розширення ієрархії класів для ефективного представлення даних у вигляді списків та множин. Тестування реалізацій здійснювалося для перевірки правильності та консистентності отриманих результатів.

У процесі роботи було використано метод pow() класу BigInteger, що впроваджує ефективний алгоритм "бінарного піднесення до степеня". Цей алгоритм дозволяє зменшити кількість множень, що призводить до покращення швидкодії, особливо для великих степенів.

Також була виконана робота з великими числами та наборами даних, що включала генерацію випадкових значень, сортування, обчислення добутку додатних чисел та форматування значень для відображення. Всі ці завдання були виконані з використанням трьох різних підходів: з використанням циклів і умовних тверджень, без явних циклів і розгалужень, та з використанням засобів Stream API.

Загалом, ця лабораторна робота була надзвичайно корисною для поглиблення розуміння Stream API, JUnit, роботи з великими числами та колекціями у мові програмування Java, а також для розвитку навичок вирішення завдань в програмуванні.