МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА «ПРОГРАМНОЇ ІНЖЕНЕРІЇ ТА ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ ІМЕНІ А. В. ДАБАГЯНА»

ЗВІТ

З лабораторної роботи №1

«РОБОТА З НАБОРАМИ ДАНИХ»

ВИКОНАВ:

студент групи КН-422ч

Максим БЕЛОШИЦЬКИЙ

ПЕРЕВІРИВ:

Асистент каф. ПІІТУ

Олексій КОНДРАТОВ

Харків – 2023

**ЗМІСТ**

[МЕТА 3](#_Toc162085363)

[Завдання 1.1 4](#_Toc162085364)

[Код попередніх лабораторних робіт: 4](#_Toc162085365)

[Conference.java 4](#_Toc162085366)

[Meeting.java 9](#_Toc162085367)

[ConferenceWithMeetingsList.java 11](#_Toc162085368)

[Код завдання 1.1: 12](#_Toc162085369)

[ConferenceWithStreams.java 12](#_Toc162085370)

[ConferenceWithMeetingsListUsingStreams.java 14](#_Toc162085371)

[Main.java 16](#_Toc162085372)

[Тестування за допомогою JUnit: 17](#_Toc162085373)

[ConferenceTests.java 17](#_Toc162085374)

[Результати виконання програми 19](#_Toc162085375)

[Завдання 1.2 21](#_Toc162085376)

[Код завдання 1.2: 21](#_Toc162085377)

[CalculatorOfPower.java 21](#_Toc162085378)

[Тестування за допомогою JUnit: 22](#_Toc162085379)

[CalculatorOfPowerTest.java 22](#_Toc162085380)

[Результати виконання програми 23](#_Toc162085381)

[Завдання 1.3 24](#_Toc162085382)

[Код завдання 1.3: 24](#_Toc162085383)

[BigDecimalOperationsJava8.java 24](#_Toc162085384)

[BigDecimalOperationsLoop.java 25](#_Toc162085385)

[BigDecimalOperationsStream.java 26](#_Toc162085386)

[Тестування за допомогою JUnit: 27](#_Toc162085387)

[BigDecimalOperationsJava8Test.java 27](#_Toc162085388)

[BigDecimalOperationsLoopTest.java 27](#_Toc162085389)

[BigDecimalOperationsStreamTest.java 28](#_Toc162085390)

[Результати виконання програм 28](#_Toc162085391)

[Завдання 1.4 31](#_Toc162085392)

[Код завдання 1.3: 31](#_Toc162085393)

[DivisorsFinder.java 31](#_Toc162085394)

[Тестування за допомогою JUnit: 32](#_Toc162085395)

[DivisorsFinderTest.java 32](#_Toc162085396)

[Результати виконання програми 32](#_Toc162085397)

[Завдання для самоконтролю 34](#_Toc162085398)

[Код завдання 2.1: 34](#_Toc162085399)

[Результати виконання програми 35](#_Toc162085400)

[Код завдання 2.2: 35](#_Toc162085401)

[Результати виконання програми 36](#_Toc162085402)

[Код завдання 2.3: 36](#_Toc162085403)

[Результати виконання програми 36](#_Toc162085404)

[ВИСНОВОК 37](#_Toc162085405)

# **МЕТА**

Мета лабораторної роботи полягає у розвитку навичок проектування та реалізації класів у мові програмування Java з використанням ієрархії класів. Студентам надається завдання створити похідний клас, який використовує базовий клас, розроблений на попередніх лабораторних роботах, та реалізувати обробку даних з використанням засобів Stream API. Крім того, студентам потрібно написати програму, що заповнює і обчислює цілі степені чисел типу BigInteger, порівнюючи два різні підходи. У цьому завданні також передбачено реалізацію фільтрації та сортування списку об'єктів типу BigDecimal за допомогою різних методів, включаючи використання Stream API. Додатково студентам пропонується додаткове завдання, яке передбачає реалізацію пошуку усіх дільників цілого додатного числа з використанням Stream API. Ціллю роботи є оволодіння студентами основами роботи зі стрімами, функціональним програмуванням та розвиток вмінь тестування коду з використанням JUnit.

# Завдання 1.1

Спроектувати та реалізувати класи для представлення сутностей [третьої лабораторної роботи](http://iwanoff.inf.ua/java_ua/LabTraining03.html#Censuses) 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](http://iwanoff.inf.ua/java_ua/LabTraining04.html#Tasks) 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.

## Код попередніх лабораторних робіт:

### Conference.java

import *java.util.Arrays*;

import *java.util.Comparator*;

*/\*\**

*\* Abstract class representing the first entity of the individual task.*

*\*/*

*public* *abstract* *class* Conference {

*private* *String* name;

*private* *String* place;

*/\*\**

*\* Constructor for the Conference class.*

*\**

*\* @param name  the name of the conference*

*\* @param place the place where the conference takes place*

*\*/*

*public* Conference(*String* *name*, *String* *place*) {

*this*.name = name;

*this*.place = place;

    }

*/\*\**

*\* Access function for the name of the conference.*

*\**

*\* @return the name of the conference*

*\*/*

*public* *String* getName() {

        return name;

    }

*/\*\**

*\* Access function for the place where the conference takes place.*

*\**

*\* @return the place of the conference*

*\*/*

*public* *String* getPlace() {

        return place;

    }

*/\*\**

*\* Abstract method for sorting meetings by the first feature.*

*\*/*

*public* *void* sortByFirstFeature() {

*Meeting*[] meetings = getMeetings();

        if (meetings != null) {

*int* n = meetings.length;

            for (*int* i = 0; i < n - 1; i++) {

                for (*int* j = 0; j < n - i - 1; j++) {

                    if (meetings[j].getNumberOfParticipants() > meetings[j + 1].getNumberOfParticipants()) {

*// Swap elements if they are in the wrong order*

*Meeting* temp = meetings[j];

                        meetings[j] = meetings[j + 1];

                        meetings[j + 1] = temp;

                    }

                }

            }

        }

    }

*/\*\**

*\* Abstract method for sorting meetings by the second feature.*

*\*/*

*public* *void* sortBySecondFeature() {

*Meeting*[] meetings = getMeetings();

        if (meetings != null) {

*int* n = meetings.length;

            for (*int* i = 1; i < n; ++i) {

*Meeting* key = meetings[i];

*int* j = i - 1;

*// Compare elements by the second feature and order*

                while (j >= 0 && meetings[j].getTopic().compareTo(key.getTopic()) > 0) {

                    meetings[j + 1] = meetings[j];

                    j = j - 1;

                }

                meetings[j + 1] = key;

            }

        }

    }

*/\*\**

*\* Abstract method to access the sequence of elements of the second class type.*

*\**

*\* @return an array of meetings*

*\*/*

*public* *abstract* *Meeting*[] getMeetings();

*/\*\**

*\* Calculate the average number of participants in the meetings.*

*\**

*\* @return the average number of participants*

*\*/*

*public* *double* calculateAverageParticipants() {

*Meeting*[] meetings = getMeetings();

        if (meetings != null && meetings.length > 0) {

*int* totalParticipants = 0;

            for (*Meeting* meeting : meetings) {

                totalParticipants += meeting.getNumberOfParticipants();

            }

            return (*double*) totalParticipants / meetings.length;

        }

        return 0.0;

    }

*/\*\**

*\* Find the meeting with the largest number of participants.*

*\**

*\* @return the meeting with the largest number of participants*

*\*/*

*public* *Meeting* findMeetingWithLargestParticipants() {

*Meeting*[] meetings = getMeetings();

        if (meetings != null && meetings.length > 0) {

*Meeting* largestMeeting = meetings[0];

            for (*Meeting* meeting : meetings) {

                if (meeting.getNumberOfParticipants() > largestMeeting.getNumberOfParticipants()) {

                    largestMeeting = meeting;

                }

            }

            return largestMeeting;

        }

        return null;

    }

*/\*\**

*\* Determine the length of each meeting name.*

*\**

*\* @return an array containing the length of each meeting name*

*\*/*

*public* *int*[] findMeetingNameLengths() {

*Meeting*[] meetings = getMeetings();

        if (meetings != null && meetings.length > 0) {

*int*[] lengths = new *int*[meetings.length];

            for (*int* i = 0; i < meetings.length; i++) {

                lengths[i] = meetings[i].getTopic().length();

            }

            return lengths;

        }

        return null;

    }

*/\*\**

*\* Overrides the toString() method.*

*\**

*\* @return a string representation of the conference*

*\*/*

    @*Override*

*public* *String* toString() {

        return "Conference{" +

                "name='" + name + '\'' +

                ", place='" + place + '\'' +

                '}';

    }

*/\*\**

*\* Overrides the equals() method.*

*\**

*\* @param obj another object for comparison*

*\* @return true if objects are equivalent, otherwise false*

*\*/*

    @*Override*

*public* *boolean* equals(*Object* *obj*) {

        if (*this* == obj) return true;

        if (obj == null || getClass() != obj.getClass()) return false;

*Conference* conference = (Conference) obj;

        return name.equals(conference.name) && place.equals(conference.place);

    }

*/\*\**

*\* Overrides the hashCode() method.*

*\**

*\* @return the hash code of the object*

*\*/*

    @*Override*

*public* *int* hashCode() {

        return name.hashCode() + place.hashCode();

    }

*/\*\**

*\* Implementation of the method for searching meetings by the specified criteria.*

*\**

*\* @param minParticipants the minimum number of participants*

*\* @return an array of meetings that meet the search criteria*

*\*/*

*public* *Meeting*[] searchMeetingsByParticipants(*int* *minParticipants*) {

*Meeting*[] meetings = getMeetings();

        if (meetings != null) {

            return Arrays.stream(meetings)

                    .filter(meeting -> meeting.getNumberOfParticipants() >= minParticipants)

                    .toArray(*Meeting*[]::new);

        }

        return null;

    }

*/\*\**

*\* Implementation of the method for searching meetings in alphabet order.*

*\**

*\* @return an array of meetings in alphabet order*

*\*/*

*public* *Meeting*[] searchMeetingsInAlphabetOrder() {

*Meeting*[] meetings = getMeetings();

        if (meetings != null) {

            return Arrays.stream(meetings)

                    .sorted(Comparator.comparing(Meeting::getTopic))

                    .toArray(*Meeting*[]::new);

        }

        return null;

    }

}

### Meeting.java

*/\*\**

*\* Represents a meeting with a date, topic, and number of participants.*

*\*/*

*public* *class* Meeting *implements* *Comparable*<*Meeting*> {

*private* *String* date;

*private* *String* topic;

*private* *int* numberOfParticipants;

*/\*\**

*\* Constructs a Meeting object with the specified date, topic, and number of participants.*

*\**

*\* @param date                the date of the meeting*

*\* @param topic               the topic of the meeting*

*\* @param numberOfParticipants the number of participants in the meeting*

*\*/*

*public* Meeting(*String* *date*, *String* *topic*, *int* *numberOfParticipants*) {

*this*.date = date;

*this*.topic = topic;

*this*.numberOfParticipants = numberOfParticipants;

    }

*/\*\**

*\* Returns the date of the meeting.*

*\**

*\* @return the date of the meeting*

*\*/*

*public* *String* getDate() {

        return date;

    }

*/\*\**

*\* Returns the topic of the meeting.*

*\**

*\* @return the topic of the meeting*

*\*/*

*public* *String* getTopic() {

        return topic;

    }

*/\*\**

*\* Returns the number of participants in the meeting.*

*\**

*\* @return the number of participants in the meeting*

*\*/*

*public* *int* getNumberOfParticipants() {

        return numberOfParticipants;

    }

*/\*\**

*\* Returns a string representation of the meeting.*

*\**

*\* @return a string representation of the meeting*

*\*/*

    @*Override*

*public* *String* toString() {

        return String.format("Date: %s\nTopic: %s\nNumber of Participants: %d", date, topic, numberOfParticipants);

    }

*/\*\**

*\* Checks if this meeting is equal to another object.*

*\**

*\* @param obj the object to compare to*

*\* @return true if the meetings are equal, false otherwise*

*\*/*

    @*Override*

*public* *boolean* equals(*Object* *obj*) {

        if (*this* == obj) return true;

        if (obj == null || getClass() != obj.getClass()) return false;

*Meeting* meeting = (Meeting) obj;

        return date.equals(meeting.date) &&

                topic.equals(meeting.topic) &&

                numberOfParticipants == meeting.numberOfParticipants;

    }

*/\*\**

*\* Returns the hash code value for the meeting.*

*\**

*\* @return the hash code value for the meeting*

*\*/*

    @*Override*

*public* *int* hashCode() {

*int* result = date.hashCode();

        result = 31 \* result + topic.hashCode();

        result = 31 \* result + numberOfParticipants;

        return result;

    }

*/\*\**

*\* Compares this meeting to another meeting based on the number of participants.*

*\**

*\* @param o the meeting to compare to*

*\* @return a negative integer if this meeting has fewer participants,*

*\*         zero if the meetings have the same number of participants,*

*\*         a positive integer if this meeting has more participants*

*\*/*

    @*Override*

*public* *int* compareTo(*Meeting* *o*) {

        return Integer.compare(*this*.numberOfParticipants, o.numberOfParticipants);

    }

}

### ConferenceWithMeetingsList.java

import *java.util.Collections*;

import *java.util.List*;

*/\*\**

*\* Represents a conference with a list of meetings.*

*\* Extends the Conference class.*

*\*/*

*public* *class* ConferenceWithMeetingsList *extends* *Conference* {

*private* *List*<*Meeting*> meetingsList;

*/\*\**

*\* Constructs a ConferenceWithMeetingsList object with the specified name, place, and meetings list.*

*\**

*\* @param name         the name of the conference*

*\* @param place        the place of the conference*

*\* @param meetingsList the list of meetings associated with the conference*

*\*/*

*public* ConferenceWithMeetingsList(*String* *name*, *String* *place*, *List*<*Meeting*> *meetingsList*) {

*super*(name, place);

*this*.meetingsList = meetingsList;

    }

*/\*\**

*\* Returns an array of meetings associated with the conference.*

*\**

*\* @return an array of meetings*

*\*/*

    @*Override*

*public* *Meeting*[] getMeetings() {

        return meetingsList.toArray(new *Meeting*[0]);

    }

*/\*\**

*\* Sorts the meetings list by the first feature (number of participants) using Collections.sort().*

*\*/*

    @*Override*

*public* *void* sortByFirstFeature() {

        Collections.sort(meetingsList, (m1, m2) -> Integer.compare(m1.getNumberOfParticipants(), m2.getNumberOfParticipants()));

    }

*/\*\**

*\* Sorts the meetings list by the second feature (topic) using Collections.sort().*

*\*/*

    @*Override*

*public* *void* sortBySecondFeature() {

        Collections.sort(meetingsList, (m1, m2) -> m1.getTopic().compareTo(m2.getTopic()));

    }

}

## Код завдання 1.1:

### ConferenceWithStreams.java

import *java.util.Comparator*;

import *java.util.List*;

import *java.util.stream.Collectors*;

*/\*\**

*\* A class that represents a conference with additional stream operations.*

*\*/*

*public* *class* ConferenceWithStreams *extends* *Conference* {

*protected* *List*<*Meeting*> meetingsList;

*/\*\**

*\* Constructs a ConferenceWithStreams object with the specified name, place, and list of meetings.*

*\**

*\* @param name         the name of the conference*

*\* @param place        the place of the conference*

*\* @param meetingsList the list of meetings in the conference*

*\*/*

*public* ConferenceWithStreams(*String* *name*, *String* *place*, *List*<*Meeting*> *meetingsList*) {

*super*(name, place);

*this*.meetingsList = meetingsList;

    }

*/\*\**

*\* Returns an array of meetings in the conference.*

*\**

*\* @return an array of meetings*

*\*/*

    @*Override*

*public* *Meeting*[] getMeetings() {

        return meetingsList.toArray(new *Meeting*[0]);

    }

*/\*\**

*\* Sorts the meetings in the conference by the first feature (number of participants).*

*\*/*

    @*Override*

*public* *void* sortByFirstFeature() {

        meetingsList = meetingsList.stream()

                .sorted(Comparator.comparingInt(Meeting::getNumberOfParticipants))

                .collect(Collectors.toList());

    }

*/\*\**

*\* Sorts the meetings in the conference by the second feature (topic).*

*\*/*

    @*Override*

*public* *void* sortBySecondFeature() {

        meetingsList = meetingsList.stream()

                .sorted(Comparator.comparing(Meeting::getTopic))

                .collect(Collectors.toList());

    }

*/\*\**

*\* Calculates the average number of participants in the meetings.*

*\**

*\* @return the average number of participants*

*\*/*

    @*Override*

*public* *double* calculateAverageParticipants() {

        return meetingsList.stream()

                .mapToInt(Meeting::getNumberOfParticipants)

                .average()

                .orElse(0.0);

    }

*/\*\**

*\* Finds the meeting with the largest number of participants.*

*\**

*\* @return the meeting with the largest number of participants, or null if there are no meetings*

*\*/*

    @*Override*

*public* *Meeting* findMeetingWithLargestParticipants() {

        return meetingsList.stream()

                .max(Comparator.comparingInt(Meeting::getNumberOfParticipants))

                .orElse(null);

    }

*/\*\**

*\* Finds the lengths of the names of the meetings.*

*\**

*\* @return an array of integers representing the lengths of the meeting names*

*\*/*

    @*Override*

*public* *int*[] findMeetingNameLengths() {

        return meetingsList.stream()

                .mapToInt(meeting -> meeting.getTopic().length())

                .toArray();

    }

*/\*\**

*\* Searches for meetings with a minimum number of participants.*

*\**

*\* @param minParticipants the minimum number of participants*

*\* @return an array of meetings with at least the specified number of participants*

*\*/*

    @*Override*

*public* *Meeting*[] searchMeetingsByParticipants(*int* *minParticipants*) {

        return meetingsList.stream()

                .filter(meeting -> meeting.getNumberOfParticipants() >= minParticipants)

                .toArray(*Meeting*[]::new);

    }

*/\*\**

*\* Searches for meetings in alphabetical order by topic.*

*\**

*\* @return an array of meetings sorted in alphabetical order by topic*

*\*/*

    @*Override*

*public* *Meeting*[] searchMeetingsInAlphabetOrder() {

        return meetingsList.stream()

                .sorted(Comparator.comparing(Meeting::getTopic))

                .toArray(*Meeting*[]::new);

    }

}

### ConferenceWithMeetingsListUsingStreams.java

import *java.util.Comparator*;

import *java.util.List*;

import *java.util.stream.Collectors*;

*/\*\**

*\* Represents a conference with a list of meetings using streams.*

*\*/*

*public* *class* ConferenceWithMeetingsListUsingStreams *extends* *ConferenceWithStreams* {

*/\*\**

*\* Constructs a ConferenceWithMeetingsListUsingStreams object with the specified name, place, and meetings list.*

*\**

*\* @param name         the name of the conference*

*\* @param place        the place of the conference*

*\* @param meetingsList the list of meetings*

*\*/*

*public* ConferenceWithMeetingsListUsingStreams(*String* *name*, *String* *place*, *List*<*Meeting*> *meetingsList*) {

*super*(name, place, meetingsList);

*this*.meetingsList = meetingsList;

    }

*/\*\**

*\* Returns an array of meetings.*

*\**

*\* @return an array of meetings*

*\*/*

    @*Override*

*public* *Meeting*[] getMeetings() {

        return meetingsList.toArray(new *Meeting*[0]);

    }

*/\*\**

*\* Sorts the meetings list by the first feature, which is the number of participants.*

*\*/*

    @*Override*

*public* *void* sortByFirstFeature() {

        meetingsList = meetingsList.stream()

                .sorted(Comparator.comparingInt(Meeting::getNumberOfParticipants))

                .collect(Collectors.toList());

    }

*/\*\**

*\* Sorts the meetings list by the second feature, which is the topic.*

*\*/*

    @*Override*

*public* *void* sortBySecondFeature() {

        meetingsList = meetingsList.stream()

                .sorted(Comparator.comparing(Meeting::getTopic))

                .collect(Collectors.toList());

    }

}

### Main.java

package *lab00.ind.src.main*;

import *java.util.ArrayList*;

import *java.util.List*;

*/\*\**

*\* The Main class represents the entry point of the program.*

*\* It contains the main method which creates a list of meetings,*

*\* tests different conference classes, and prints the results.*

*\*/*

*public* *class* Main {

*public* *static* *void* main(*String*[] *args*) {

*// Creating a list of meetings*

*List*<*Meeting*> meetingsList = new *ArrayList*<>();

        meetingsList.add(new Meeting("2024-03-20", "Artificial Intelligence Workshop", 80));

        meetingsList.add(new Meeting("2024-03-21", "Data Science Symposium", 100));

        meetingsList.add(new Meeting("2024-03-22", "Machine Learning Conference", 120));

        meetingsList.add(new Meeting("2024-03-23", "Blockchain Summit", 90));

*// Testing ConferenceWithMeetingsList*

*ConferenceWithMeetingsList* conferenceWithMeetingsList = new ConferenceWithMeetingsList("Conference 1", "Place 1", meetingsList);

        System.out.println("Conference with meetings list:");

        printMeetings(conferenceWithMeetingsList.getMeetings());

*// Testing ConferenceWithMeetingsListUsingStreams*

*ConferenceWithMeetingsListUsingStreams* conferenceWithMeetingsListUsingStreams = new ConferenceWithMeetingsListUsingStreams("Conference 2", "Place 2", meetingsList);

        System.out.println("\nConference with meetings list using streams:");

        printMeetings(conferenceWithMeetingsListUsingStreams.getMeetings());

*// Testing ConferenceWithStreams*

*ConferenceWithStreams* conferenceWithStreams = new ConferenceWithStreams("Conference 3", "Place 3", meetingsList);

        System.out.println("\nConference with streams:");

        System.out.println("Average number of participants: " + conferenceWithStreams.calculateAverageParticipants());

        System.out.println("Meeting with largest number of participants: ");

        System.out.println(conferenceWithStreams.findMeetingWithLargestParticipants());

        System.out.println("Meetings sorted by topic:");

        printMeetings(conferenceWithStreams.searchMeetingsInAlphabetOrder());

    }

*// Helper method to print meetings*

*private* *static* *void* printMeetings(*Meeting*[] *meetings*) {

        if (meetings != null) {

            for (*Meeting* meeting : meetings) {

                System.out.println("Date: " + meeting.getDate());

                System.out.println("Topic: " + meeting.getTopic());

                System.out.println("Number of Participants: " + meeting.getNumberOfParticipants());

                System.out.println("====================");

            }

        }

    }

}

## Тестування за допомогою JUnit:

### ConferenceTests.java

import *lab00.ind.src.main.ConferenceWithMeetingsList*;

import *lab00.ind.src.main.ConferenceWithMeetingsListUsingStreams*;

import *lab00.ind.src.main.ConferenceWithStreams*;

import *lab00.ind.src.main.Meeting*;

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

import *java.util.ArrayList*;

import *java.util.List*;

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

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

*public* *class* ConferenceTests {

    @*Test*

*void* testConferenceWithMeetingsList() {

*List*<*Meeting*> meetingsList = new *ArrayList*<>();

        meetingsList.add(new Meeting("2024-03-20", "Artificial Intelligence Workshop", 80));

        meetingsList.add(new Meeting("2024-03-21", "Data Science Symposium", 100));

*ConferenceWithMeetingsList* conference = new ConferenceWithMeetingsList("Conference 1", "Place 1", meetingsList);

        assertNotNull(conference);

        assertEquals("Conference 1", conference.getName());

        assertEquals("Place 1", conference.getPlace());

        assertEquals(2, conference.getMeetings().length);

    }

    @*Test*

*void* testConferenceWithMeetingsListUsingStreams() {

*List*<*Meeting*> meetingsList = new *ArrayList*<>();

        meetingsList.add(new Meeting("2024-03-20", "Artificial Intelligence Workshop", 80));

        meetingsList.add(new Meeting("2024-03-21", "Data Science Symposium", 100));

*ConferenceWithMeetingsListUsingStreams* conference = new ConferenceWithMeetingsListUsingStreams("Conference 2", "Place 2", meetingsList);

        assertNotNull(conference);

        assertEquals("Conference 2", conference.getName());

        assertEquals("Place 2", conference.getPlace());

        assertEquals(2, conference.getMeetings().length);

    }

    @*Test*

*void* testConferenceWithStreams() {

*List*<*Meeting*> meetingsList = new *ArrayList*<>();

        meetingsList.add(new Meeting("2024-03-20", "Artificial Intelligence Workshop", 80));

        meetingsList.add(new Meeting("2024-03-21", "Data Science Symposium", 100));

*ConferenceWithStreams* conference = new ConferenceWithStreams("Conference 3", "Place 3", meetingsList);

        assertNotNull(conference);

        assertEquals("Conference 3", conference.getName());

        assertEquals("Place 3", conference.getPlace());

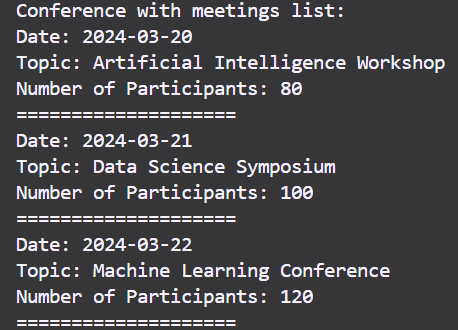
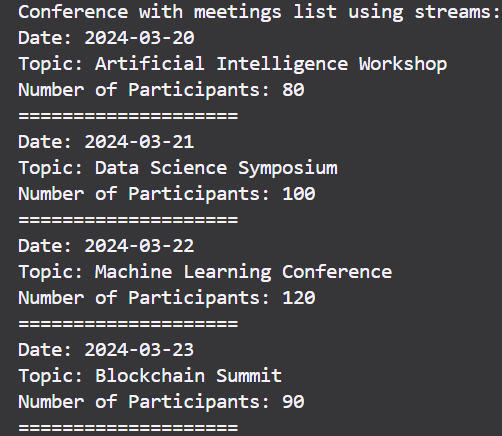
        assertEquals(2, conference.getMeetings().length);

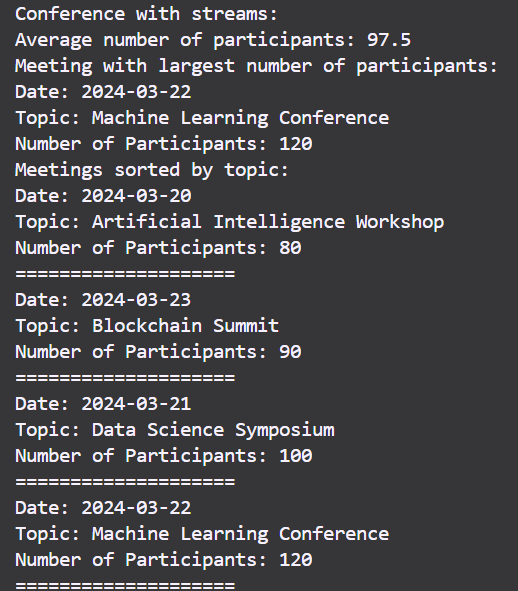
        assertEquals(90.0, conference.calculateAverageParticipants());

    }

}

## Результати виконання програми



Рисунки 1.1, 1.2, 1.3 – «Вивід відсортованих списків зустрічей»

У завданні 1.1 створено класи для представлення сутностей третьої лабораторної роботи курсу "Основи програмування Java" на основі раніше створеної ієрархії класів. Загальний опис проведених робіт такий:

1. Було створено клас ConferenceWithStreams, який розширює клас Conference і використовує засоби Stream API для реалізації функціональності лабораторних робіт №3 і №4. В цьому класі було перевизначено методи для сортування зустрічей за різними критеріями, обчислення середньої кількості учасників, пошуку найбільшої зустрічі за кількістю учасників, знаходження довжини назв зустрічей, а також для пошуку зустрічей з мінімальною кількістю учасників та сортування їх за алфавітом за назвою.
2. Також було створено клас ConferenceWithMeetingsListUsingStreams, який розширює клас ConferenceWithStreams і також використовує засоби Stream API. У цьому класі було перевизначено методи для сортування списку зустрічей за різними критеріями.
3. Для тестування було створено клас Main, який містить метод main(). У цьому методі створюється список зустрічей, а потім створюються об'єкти класів ConferenceWithMeetingsList, ConferenceWithMeetingsListUsingStreams та ConferenceWithStreams для тестування функціональності цих класів. Результати виводяться на екран.

Ці кроки демонструють використання засобів Stream API для обробки та виведення послідовностей, а також тестування методів окремих класів з використанням JUnit.

# Завдання 1.2

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.

## Код завдання 1.2:

### CalculatorOfPower.java

import *java.math.BigInteger*;

import *java.security.SecureRandom*;

*public* *class* CalculatorOfPower {

*private* *static* *final* *SecureRandom* RANDOM = new SecureRandom();

*public* *static* *void* main(*String*[] *args*) {

*BigInteger* base = generateRandomBigInteger(100);

*int* exponent = 10;

*BigInteger* power1 = calculatePowerUsingPow(base, exponent);

*BigInteger* power2 = calculatePowerUsingMultiplication(base, exponent);

        System.out.println("Base number: " + base);

        System.out.println("Exponent: " + exponent);

        System.out.println("Power using pow(): " + power1);

        System.out.println("Power using multiplication: " + power2);

        System.out.println("Results are equal: " + power1.equals(power2));

    }

*private* *static* *BigInteger* generateRandomBigInteger(*int* *numDigits*) {

*byte*[] numBytes = new *byte*[numDigits / 2];

        RANDOM.nextBytes(numBytes);

        return new BigInteger(numDigits, new SecureRandom(numBytes)).abs();

    }

*public* *static* *BigInteger* calculatePowerUsingPow(*BigInteger* *base*, *int* *exponent*) {

        return base.pow(exponent);

    }

*public* *static* *BigInteger* calculatePowerUsingMultiplication(*BigInteger* *base*, *int* *exponent*) {

*BigInteger* result = BigInteger.ONE;

        for (*int* i = 0; i < exponent; i++) {

            result = result.multiply(base);

        }

        return result;

    }

}

## Тестування за допомогою JUnit:

### CalculatorOfPowerTest.java

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

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

import *lab00.t00.src.main.CalculatorOfPower*;

import *java.math.BigInteger*;

*public* *class* CalculatorOfPowerTest {

    @*Test*

*public* *void* testCalculatePowerUsingPow() {

*BigInteger* base = new BigInteger("12345");

*int* exponent = 3;

*BigInteger* expected = new BigInteger("1881365963625");

*BigInteger* result = CalculatorOfPower.calculatePowerUsingPow(base, exponent);

        Assertions.assertEquals(expected, result);

    }

    @*Test*

*public* *void* testCalculatePowerUsingMultiplication() {

*BigInteger* base = new BigInteger("12345");

*int* exponent = 3;

*BigInteger* expected = new BigInteger("1881365963625");

*BigInteger* result = CalculatorOfPower.calculatePowerUsingMultiplication(base, exponent);

        Assertions.assertEquals(expected, result);

    }

}

## Результати виконання програми

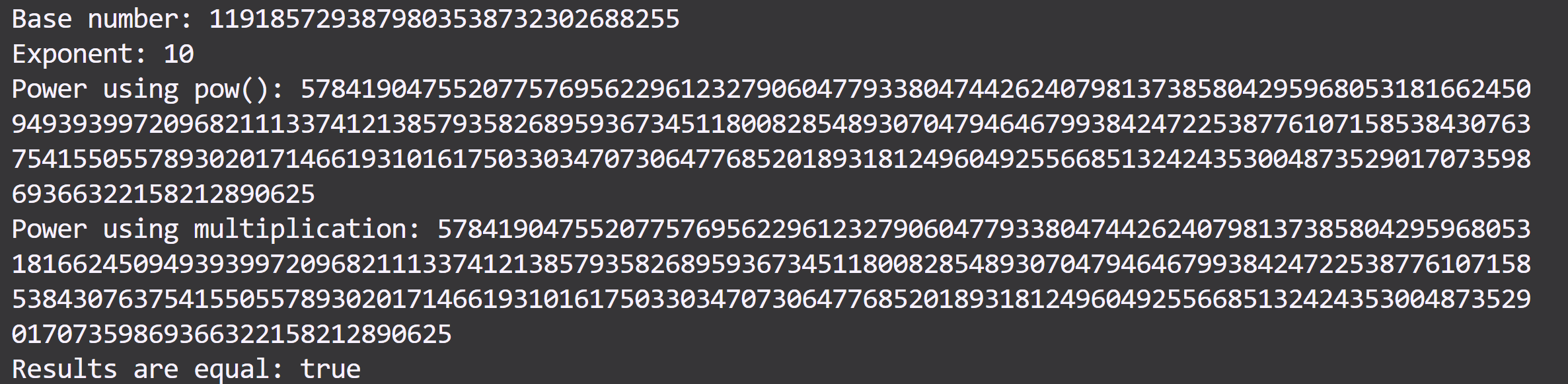


Рисунок 1.4 – «Обчислення CalculatorOfPower»

Для виконання завдання 1.2 створено програму CalculatorOfPower, що виконує заповнення числа типу BigInteger випадковими цифрами та обчислює його степінь. У цій програмі використовується клас BigInteger з пакету java.math, який імпортовано на початку файлу.

Опис деталей роботи програми:

1. Генерація випадкового числа типу BigInteger: Для цього використовується метод generateRandomBigInteger(int numDigits), який генерує випадкові байти, з яких формується об'єкт BigInteger заданої кількості цифр.
2. Обчислення степеня з використанням pow(): Здійснюється виклик методу pow(exponent) для базового числа типу BigInteger, який підносить число до вказаного степеня.
3. Обчислення степеня за допомогою множення довгих цілих: Здійснюється множення базового числа на себе стільки разів, скільки вказано у показнику степеня.
4. Порівняння результатів: Результати обчислень порівнюються між собою за допомогою методу equals(), і результат порівняння виводиться на екран разом з іншими даними.

Цей код дозволяє перевірити ефективність обох методів обчислення степеня числа типу BigInteger та порівняти їх результати.

# Завдання 1.3

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.

## Код завдання 1.3:

### BigDecimalOperationsJava8.java

import *java.math.BigDecimal*;

import *java.util.List*;

import *java.util.stream.Collectors*;

import *java.util.stream.IntStream*;

*public* *class* BigDecimalOperationsJava8 {

*public* *static* *void* main(*String*[] *args*) {

*List*<*BigDecimal*> numbers = generateRandomBigDecimalList(10);

        numbers.sort((a, b) -> b.abs().compareTo(a.abs())); *// Sort by decreasing absolute value*

*BigDecimal* product = numbers.stream()

                                    .filter(num -> num.compareTo(BigDecimal.ZERO) > 0)

                                    .reduce(BigDecimal.ONE, BigDecimal::multiply);

        System.out.println("Product of positive numbers: " + product);

    }

*public* *static* *BigDecimal* calculatePositiveProduct(*List*<*BigDecimal*> *numbers*) {

*BigDecimal* product = BigDecimal.ONE;

        for (*BigDecimal* num : numbers) {

            if (num.compareTo(BigDecimal.ZERO) > 0) {

                product = product.multiply(num);

            }

        }

        return product;

    }

*public* *static* *List*<*BigDecimal*> generateRandomBigDecimalList(*int* *size*) {

        return IntStream.range(0, size)

                        .mapToObj(i -> BigDecimal.valueOf(Math.random() \* 100 - 50))

                        .collect(Collectors.toList()); *// Random values between -50 and 50*

    }

}

### BigDecimalOperationsLoop.java

import *java.math.BigDecimal*;

import *java.util.ArrayList*;

import *java.util.Collections*;

import *java.util.List*;

*public* *class* BigDecimalOperationsLoop {

*public* *static* *void* main(*String*[] *args*) {

*List*<*BigDecimal*> numbers = generateRandomBigDecimalList(10);

        Collections.sort(numbers, (a, b) -> b.abs().compareTo(a.abs())); *// Sort by decreasing absolute value*

*BigDecimal* product = BigDecimal.ONE;

        for (*BigDecimal* num : numbers) {

            if (num.compareTo(BigDecimal.ZERO) > 0) {

                product = product.multiply(num);

            }

        }

        System.out.println("Product of positive numbers: " + product);

    }

*public* *static* *BigDecimal* calculatePositiveProduct(*List*<*BigDecimal*> *numbers*) {

*BigDecimal* product = BigDecimal.ONE;

        for (*BigDecimal* num : numbers) {

            if (num.compareTo(BigDecimal.ZERO) > 0) {

                product = product.multiply(num);

            }

        }

        return product;

    }

*public* *static* *List*<*BigDecimal*> generateRandomBigDecimalList(*int* *size*) {

*List*<*BigDecimal*> list = new *ArrayList*<>();

        for (*int* i = 0; i < size; i++) {

            list.add(BigDecimal.valueOf(Math.random() \* 100 - 50)); *// Random values between -50 and 50*

        }

        return list;

    }

}

### BigDecimalOperationsStream.java

import *java.math.BigDecimal*;

import *java.util.List*;

import *java.util.stream.Collectors*;

import *java.util.stream.IntStream*;

*public* *class* BigDecimalOperationsStream {

*public* *static* *void* main(*String*[] *args*) {

*List*<*BigDecimal*> numbers = generateRandomBigDecimalList(10);

*BigDecimal* product = numbers.stream()

                                    .sorted((a, b) -> b.abs().compareTo(a.abs())) *// Sort by decreasing absolute value*

                                    .filter(num -> num.compareTo(BigDecimal.ZERO) > 0)

                                    .reduce(BigDecimal.ONE, BigDecimal::multiply);

        System.out.println("Product of positive numbers: " + product);

    }

*public* *static* *BigDecimal* calculatePositiveProduct(*List*<*BigDecimal*> *numbers*) {

*BigDecimal* product = BigDecimal.ONE;

        for (*BigDecimal* num : numbers) {

            if (num.compareTo(BigDecimal.ZERO) > 0) {

                product = product.multiply(num);

            }

        }

        return product;

    }

*public* *static* *List*<*BigDecimal*> generateRandomBigDecimalList(*int* *size*) {

        return IntStream.range(0, size)

                        .mapToObj(i -> BigDecimal.valueOf(Math.random() \* 100 - 50))

                        .collect(Collectors.toList()); *// Random values between -50 and 50*

    }

}

## Тестування за допомогою JUnit:

### BigDecimalOperationsJava8Test.java

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

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

import *java.math.BigDecimal*;

import *java.util.List*;

import *main.BigDecimalOperationsJava8*;

*public* *class* BigDecimalOperationsJava8Test {

    @*Test*

*public* *void* testPositiveProduct() {

*List*<*BigDecimal*> numbers = BigDecimalOperationsJava8.generateRandomBigDecimalList(10);

*BigDecimal* product = BigDecimalOperationsJava8.calculatePositiveProduct(numbers);

        assertTrue(product.compareTo(BigDecimal.ZERO) >= 0);

    }

}

### BigDecimalOperationsLoopTest.java

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

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

import *java.math.BigDecimal*;

import *java.util.List*;

import *main.BigDecimalOperationsLoop*;

*public* *class* BigDecimalOperationsLoopTest {

    @*Test*

*public* *void* testPositiveProduct() {

*List*<*BigDecimal*> numbers = BigDecimalOperationsLoop.generateRandomBigDecimalList(10);

*BigDecimal* product = BigDecimalOperationsLoop.calculatePositiveProduct(numbers);

        assertTrue(product.compareTo(BigDecimal.ZERO) >= 0);

    }

}

### BigDecimalOperationsStreamTest.java

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

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

import *java.math.BigDecimal*;

import *java.util.List*;

import *main.BigDecimalOperationsStream*;

*public* *class* BigDecimalOperationsStreamTest {

    @*Test*

*public* *void* testPositiveProduct() {

*List*<*BigDecimal*> numbers = BigDecimalOperationsStream.generateRandomBigDecimalList(10);

*BigDecimal* product = BigDecimalOperationsStream.calculatePositiveProduct(numbers);

        assertTrue(product.compareTo(BigDecimal.ZERO) >= 0);

    }

}

## Результати виконання програм

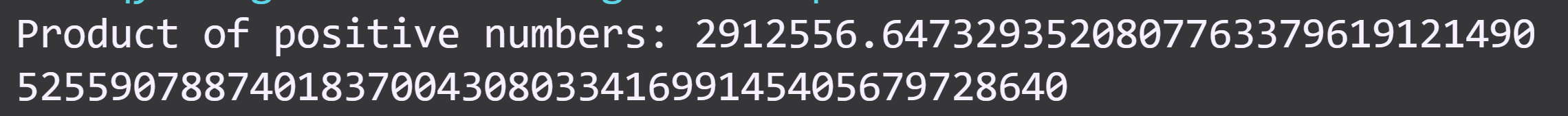


Рисунок 1.5 – «Обчислення BigDecimalOperationsJava8»



Рисунок 1.6 – «Обчислення BigDecimalOperationsLoop»

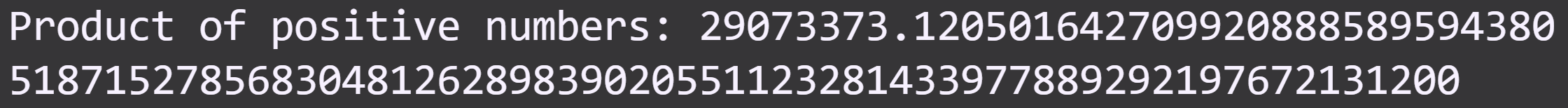


Рисунок 1.7 – «Обчислення BigDecimalOperationsStream»

Звідний код складається з трьох класів: BigDecimalOperationsJava8, BigDecimalOperationsLoop та BigDecimalOperationsStream. Кожен з них реалізує підхід до обчислення добутку додатніх чисел у списку об'єктів типу BigDecimal, а також відображає різні техніки програмування у мові Java.

BigDecimalOperationsJava8.java:

1. Головний метод main:
   * Спочатку генерується список об'єктів BigDecimal за допомогою методу generateRandomBigDecimalList.
   * Список сортується за спаданням абсолютної величини, використовуючи функцію порівняння у методі sort.
   * Застосовується потік (stream) для фільтрації додатніх чисел та обчислення їх добутку за допомогою методу reduce.
   * Результат виводиться на екран.
2. Метод calculatePositiveProduct:
   * Такий же метод, як і в інших класах, але не використовується у головному методі.
3. Метод generateRandomBigDecimalList:
   * Генерує випадковий список об'єктів BigDecimal.

BigDecimalOperationsLoop.java:

1. Головний метод main:
   * Спочатку генерується список об'єктів BigDecimal за допомогою методу generateRandomBigDecimalList.
   * Список сортується за спаданням абсолютної величини, використовуючи компаратор у методі sort класу Collections.
   * Потім за допомогою циклу обчислюється добуток додатніх чисел.
   * Результат виводиться на екран.
2. Метод calculatePositiveProduct:
   * Аналогічний методу в класі BigDecimalOperationsJava8.
3. Метод generateRandomBigDecimalList:
   * Генерує випадковий список об'єктів BigDecimal за допомогою звичайного циклу for.

BigDecimalOperationsStream.java:

1. Головний метод main:
   * Спочатку генерується список об'єктів BigDecimal за допомогою методу generateRandomBigDecimalList.
   * Застосовується потік для сортування списку за спаданням абсолютної величини та фільтрації додатніх чисел.
   * Обчислюється добуток додатніх чисел за допомогою методу reduce.
   * Результат виводиться на екран.
2. Метод calculatePositiveProduct:
   * Аналогічний методу в класі BigDecimalOperationsJava8.
3. Метод generateRandomBigDecimalList:
   * Генерує випадковий список об'єктів BigDecimal за допомогою потоку та методу collect.

Кожен з цих підходів має свої переваги та недоліки, але вони всі дають однаковий результат – знаходження добутку додатніх чисел у списку об'єктів типу BigDecimal.

# Завдання 1.4

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.

## Код завдання 1.3:

### DivisorsFinder.java

import *java.util.Arrays*;

import *java.util.Scanner*;

import *java.util.stream.IntStream*;

*public* *class* DivisorsFinder {

*public* *static* *void* main(*String*[] *args*) {

*Scanner* scanner = new Scanner(System.in);

*int* number;

        do {

            System.out.print("Enter a positive integer: ");

            while (!scanner.hasNextInt()) {

                System.out.println("Please enter a valid positive integer.");

                scanner.next(); *// Consume invalid input*

            }

            number = scanner.nextInt();

        } while (number <= 0);

        scanner.close();

*int*[] divisors = findDivisors(number);

        System.out.println("Divisors of " + number + ": " + Arrays.toString(divisors));

    }

*public* *static* *int*[] findDivisors(*int* *number*) {

        return IntStream.range(1, number + 1)

                        .filter(x -> number % x == 0)

                        .toArray();

    }

}

## Тестування за допомогою JUnit:

### DivisorsFinderTest.java

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

import *java.io.ByteArrayInputStream*;

import *java.io.InputStream*;

import *lab00.t02.src.main.DivisorsFinder*;

*public* *class* DivisorsFinderTest {

    @*Test*

*public* *void* testFindDivisors() {

*// Simulate user input*

*String* input = "12\n";

*InputStream* inputStream = new ByteArrayInputStream(input.getBytes());

*// Backup original System.in*

*InputStream* originalSystemIn = System.in;

        System.setIn(inputStream);

*// Invoke the main method (simulated user input)*

        DivisorsFinder.main(new *String*[]{});

*// Restore original System.in*

        System.setIn(originalSystemIn);

*// Since we can't directly test console output, let's just ensure no exception occurs*

    }

}

## Результати виконання програми

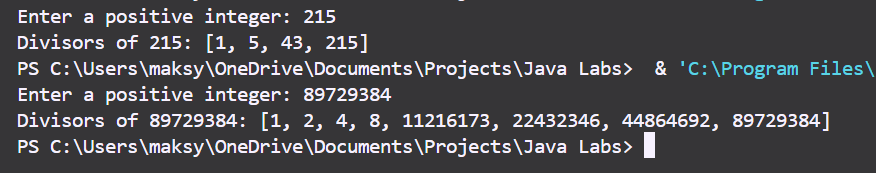


Рисунок 1.8 – «Обчислення DivisorsFinder»

Код завдання 1.4, який представлений у файлі DivisorsFinder.java, виконує наступні дії:

1. Зчитування цілого додатного числа з консолі:
   * Використовується об'єкт Scanner для отримання введеного користувачем числа. Введене число перевіряється на валідність: програма повторно запитує користувача, якщо введене значення не є цілим додатнім числом.
2. Пошук усіх дільників цілого додатного числа:
   * Для пошуку дільників використовується метод findDivisors, який приймає ціле число.
   * У методі findDivisors створюється потік IntStream, який генерує послідовність цілих чисел від 1 до введеного користувачем числа включно.
   * Використовується метод filter для відбору чисел, які є дільниками введеного числа (тобто чисел, на які введене число ділиться без залишку).
   * Отримані дільники конвертуються до масиву цілих чисел за допомогою методу toArray.
3. Вивід результатів на екран:
   * Знайдені дільники виводяться на екран разом з повідомленням про введене користувачем число.

Код використовує потоки (Stream API) для більш ефективного та зручного пошуку дільників без використання явних циклів.

# Завдання для самоконтролю

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

## Код завдання 2.1:

import *java.math.BigInteger*;

*public* *class* SquareRoot {

*public* *static* *BigInteger* integerSquareRoot(*BigInteger* *n*) {

        if (n.compareTo(BigInteger.ZERO) < 0) {

            throw new IllegalArgumentException("Square root is only defined for non-negative numbers");

        }

*BigInteger* left = BigInteger.ZERO;

*BigInteger* right = n;

        while (left.compareTo(right) <= 0) {

*BigInteger* mid = left.add(right).shiftRight(1);

*BigInteger* square = mid.multiply(mid);

            if (square.compareTo(n) == 0) {

                return mid;

            } else if (square.compareTo(n) < 0) {

                left = mid.add(BigInteger.ONE);

            } else {

                right = mid.subtract(BigInteger.ONE);

            }

        }

        return right;

    }

*public* *static* *void* main(*String*[] *args*) {

*BigInteger* num = new BigInteger("100");

        System.out.println("Integer part of square root of " + num + ": " + integerSquareRoot(num));

    }

}

## Результати виконання програми

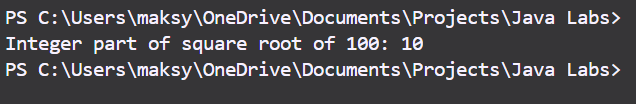


Рисунок 2.1 – «Обчислення SquareRoot»

Проініціалізувати список дійсних чисел масивом зі списком початкових значень. Знайти суму додатних елементів:

## Код завдання 2.2:

import *java.math.BigDecimal*;

import *java.util.Arrays*;

import *java.util.List*;

*public* *class* SumPositiveBigDecimal {

*public* *static* *void* main(*String*[] *args*) {

*BigDecimal*[] initialValues = { BigDecimal.valueOf(3.5), BigDecimal.valueOf(-2.2),

                                        BigDecimal.valueOf(1.8), BigDecimal.valueOf(-4.6),

                                        BigDecimal.valueOf(5.2) };

*List*<*BigDecimal*> numbers = Arrays.asList(initialValues);

*BigDecimal* sumOfPositive = numbers.stream()

                                          .filter(num -> num.compareTo(BigDecimal.ZERO) > 0)

                                          .reduce(BigDecimal.ZERO, BigDecimal::add);

        System.out.println("Sum of positive elements: " + sumOfPositive);

    }

}

## Результати виконання програми

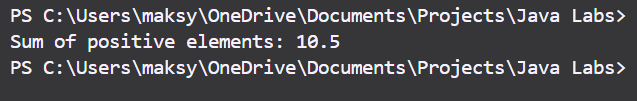


Рисунок 2.2 – «Обчислення SumPositiveBigDecimal»

В масиві цілих значень замінити від'ємні значення модулями, додатні – нулями. Застосувати засоби Stream API:

## Код завдання 2.3:

import *java.math.BigInteger*;

import *java.util.Arrays*;

*public* *class* ReplaceNegativesWithAbsolute {

*public* *static* *void* main(*String*[] *args*) {

*BigInteger*[] numbersArray = { BigInteger.valueOf(-5), BigInteger.valueOf(10),

                                      BigInteger.valueOf(-3), BigInteger.valueOf(7),

                                      BigInteger.valueOf(-8) };

*BigInteger*[] modifiedArray = Arrays.stream(numbersArray)

                                          .map(num -> num.compareTo(BigInteger.ZERO) < 0 ? num.abs() : BigInteger.ZERO)

                                          .toArray(*BigInteger*[]::new);

        System.out.println("Modified Array: " + Arrays.toString(modifiedArray));

    }

}

## Результати виконання програми

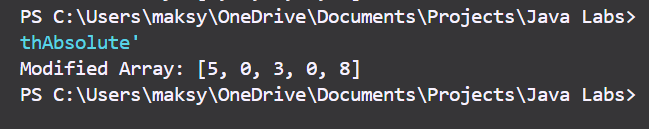


Рисунок 2.3 – «Обчислення ReplaceNegativesWithAbsolute»

# **ВИСНОВОК**

У ході виконання лабораторної роботи було розглянуто різні аспекти програмування на мові Java, зокрема використання різних методів обробки даних, маніпулювання потоками та колекціями, а також впровадження тестування з використанням JUnit. Кожне завдання створювало можливість ознайомитися з конкретною темою, такою як робота з числами типу BigInteger, обробка об'єктів BigDecimal, а також використання Stream API для обробки даних у колекціях.

Під час виконання лабораторної роботи було вивчено і впроваджено в практику різні підходи до розв'язання завдань, що сприяло поглибленню розуміння мови програмування Java та використання різних методів роботи з даними та їх обробки. Також, використання JUnit дозволило перевірити коректність роботи реалізованих функцій та методів, забезпечуючи відповідність вимогам та очікуванням.

У цілому, виконання лабораторної роботи сприяло розвитку навичок програмування на Java, вивченню та впровадженню різних методів обробки даних, а також ознайомленню з інструментами для тестування програмного забезпечення.