Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної і теплової енергетики
Кафедра цифрових технологій в енергетиці

ЗВІТ

з лабораторної роботи №4

з дисципліни «Програмування на мові Java»

Тема: «Розробка програм в середовищі INTELLIJ IDEA для дослідження концепції математичного моделювання та об'єктно-орієнтованого програмування»

Варіант №22

Виконав:
Студент групи ТР-12
Ковальов О. О.

Дата здачі: 27.11.2023

Київ — 2023

Мета роботи: набуття практичних навичок під час створення програмних проектів на мові Java.

Загальне завдання на лабораторну роботу:

1. Написати програму мовою Java. Варіант обрати за списком групи. Представити виконання в Intellij IDEA. Продемонструвати детальні скріншоти виконання коду програми з поясненням. Обов'язково – наявність висновків.
2. Написати програму мовою Java. Варіант обрати за списком групи. Представити виконання в Intellij IDEA. Продемонструвати детальні скріншоти виконання коду програми з поясненням. Обов'язково – наявність висновків.
3. Написати програму мовою Java. Варіант обрати за списком групи. Представити виконання в Intellij IDEA. Продемонструвати детальні скріншоти виконання коду програми з поясненням. Обов'язково – наявність висновків.
4. Написати програму мовою Java. Представити виконання в Intellij IDEA. Продемонструвати детальні скріншоти виконання коду програми з поясненням. Обов'язково – наявність висновків.
5. Зробити звіт з лабораторної роботи та вчасно надіслати викладачу на перевірку (дедлайн для надсилання звітів по **Лаб_4 – до 27.11.2023 року до 23:59**).

Завдання 1: Написати програму на Java, яка обчислює значення C за заданою формулою. **Варіант 2**, так як номер по списку групи **парний**.

$$C = x^3 \tan^3(x^2 + y)^3 + \frac{z}{x + y}$$

$$x = 0.2, y = 2.5, z = 11$$

Вимоги:

- Використати метод класу Math для обчислень ступенів, тригонометричних функцій, тощо.
- Перевірити результати, шляхом виконання тестування з різними значеннями x, y та z для перевірки коректності розрахунків та виведення результатів.

Для виконання завдання був написаний клас Calculator та інтерфейс EquationFunction – інтерфейс який помічений анотацією *@FunctionalInterface*. Це означає, що інтерфейс зобов'язує імплементувати всього один метод. Сенс цієї дії в тому, що реалізується передача методу у вигляді параметру в інший метод. Передається метод, який є математичною функцією у клас Calculator.

В класі Calculator наявні 3 приватних поля типу double: x, y, z. В них є гетери та сетери. Також, ще є приватне поле function типу EquationFunction, в якого теж є свій гетер та сетер. В конструктор передаються посилання на всі об'єкти відповідно приватним полям. У класі також наявні методи solve() та formattedResult(). У першому викликається метод apply інтерфейсу function. Повертає вже розв'язане значення змінної згідно переданої функції. У другому викликається перший метод, та повертається результат у текстовому вигляді. Для тестів використовуються відповідні змінні та лямбда-функції, в якій описані кроки для вирішення задачі з

умови. Потім створюється об’єкт калькулятор, у нього викликається метод для отримання відформатованого результату.

```
First Test:
x = 0.2000
y = 2.5000
z = 11.0000
Result = 4.0783

Second Test:
x = 0.6000
y = 4.1000
z = 9.0000
Result = 2.0916
```

Рис.1. Результат роботи програми

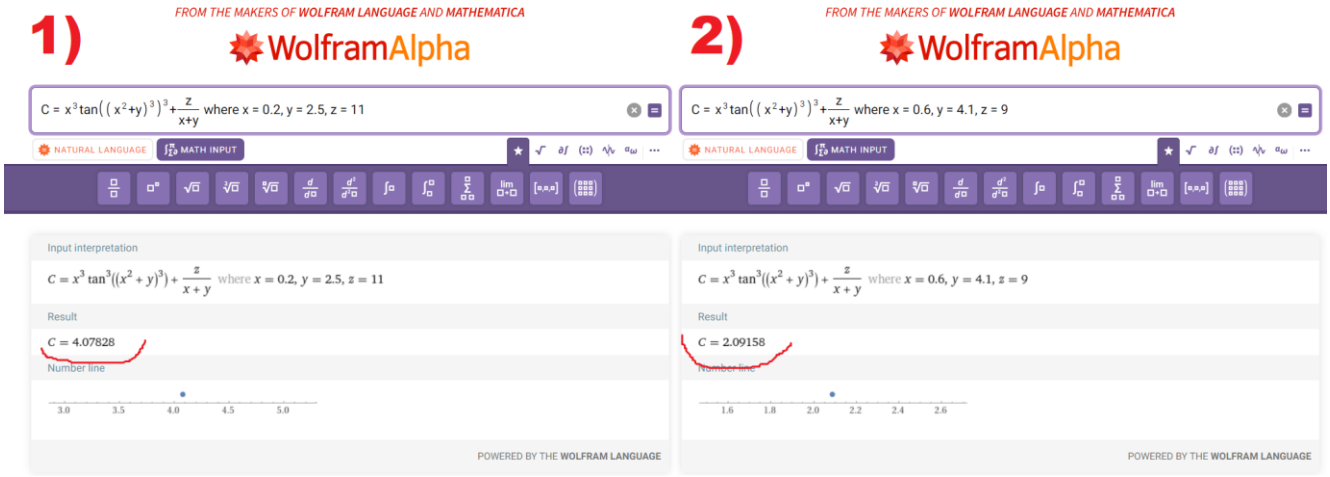

Рис.2. Перевірка результатів

Висновок: В задачі були застосовані основні принципи ООП та клас Math для роботи з математичними операціями. Також, були застосовані елементи парадигми функціонального програмування. Все працює.

Завдання 2: Написати програму на Java, яка обчислює значення L та N за заданими формулами. **Варіант 2**, так як номер по списку групи **парний**.

$$L = \frac{x^2(x+2)}{z - \sin^2(x+y)};\quad N = \sqrt{\frac{x}{y}} + \cos^2(x+z);$$

$$x = 0.1, y = 0.01, z = 0.7$$

Вимоги:

- Використати метод класу Math для обчислень ступенів, тригонометричних функцій тощо;
- Перевірити результати, шляхом виконання тестування з різними значеннями x, y та z для перевірки коректності розрахунків та виведення результатів.

Загалом, завдання ідентичне першому, змінились лише математичні функції. Для розв'язку задач використовувався клас Math і його статичні методи pow (піднесення до степеню), sin (синус), cos (косинус), tan (тангенс), sqrt (квадратний корінь). З нового – бачимо результат порушення області допустимих значень (ОДЗ) квадратного кореню, використовуємо від'ємне значення, отримуємо результат NaN (Not a Number).

```
First Test:          Second Test:
x = 0.1000           x = 1.1000
y = 0.0100           y = -1.2000
z = 0.7000           z = 4.4000
Result = 0.0305      Result = 0.8544

x = 0.1000           x = 1.1000
y = 0.0100           y = -1.2000
z = 0.7000           z = 4.4000
Result = 3.6477      Result = NaN
```

Рис.3. Результат роботи програми

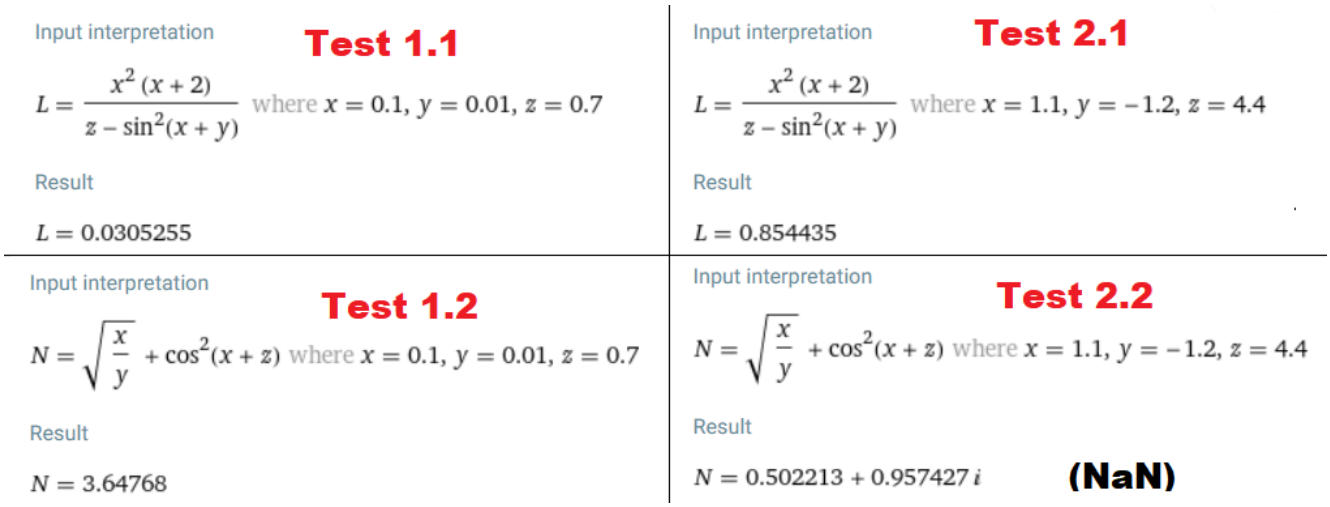

Рис.4. Перевірка результатів

Висновок: Java є чудовим засобом для обрахунку виразів такого вигляду. Але, треба зважати на те, що тип double є недостатньо точним, і якщо точність важлива, то треба обрати іншу мову програмування.

Завдання 3: Написати програму на Java, в якій буде створено ієрархію класів для різних типів транспортних засобів. Наприклад, базовий клас «Транспорт» і підкласи «Автомобіль», «Велосипед», «Мотоцикл». Кожен клас повинен включати методи для обчислення максимальної швидкості (або споживання палива). Використати наслідування для спрощення коду.

Рекомендації:

- Клас «Транспорт» може бути абстрактним, та може містити атрибути, такі як швидкість, вага, колір тощо.

- Може містити конструктор, гетери та сетери для атрибутів.
- Кожен підклас («Автомобіль», «Велосипед», «Мотоцикл») повинен реалізувати абстрактні методи та можуть додавати додаткові атрибути та методи, специфічні для кожного типу транспортного засобу.
- Для демонстрації створення об'єктів кожного з підкласів та виклику їхніх методів можна створити клас TransportTest.

Клас Vehicle – абстрактний. Має такі поля: швидкість, вага, колір, кількість коліс. Всі вони мають модифікатор приватності protected, для того щоб класи які будуть наслідувати цей клас, могли мати до них доступ. У кожного зі значень є гетери та сетери з відповідною перевіркою даних. В конструктор передаються відповідні значення для полів, і потім призначаються за допомогою сетерів. Є абстрактний метод calculateMaxSpeed, який повертає максимальну швидкість транспортного засобу, залежно від «бонусів», які у кожного виду транспорту свої. Також є ще метод printInfo, який виводить дані з перевантаженого методу toString в консоль. Дані – значення полів.

Клас Bicycle за допомогою ключового слова extends наслідує клас Vehicle. Має додатково поле «type» типу BicycleType (enum). Залежно від нього розраховується максимальна швидкість велосипеду. В конструктор передаються всі ті самі значення що й для батьківського класу, і разом з ними тип. В ньому викликається батьківський конструктор, і додатково встановлюється значення типу за допомогою сетеру. У останніх класів все відбувається так же як і тут.

Додаткові поля інших класів – FuelType для автомобіля та MotoType для мотоциклу. Вони також є enum.

Кожен клас реалізує абстрактний метод calculateMaxSpeed, додає атрибути специфічні для кожного типу транспортного засобу. Є перевантажені методи для виведення інформації про об’єкти.

```
------------------------------ ------------------------------ ------------------------------
Transport type: Car            Transport type: Bicycle        Transport type: Motorbike
Speed: 15.00 km/h              Speed: 5.00 km/h               Speed: 10.00 km/h
Weight: 1900.00 kg             Weight: 30.00 kg               Weight: 100.00 kg
Amount of wheels: 4 kg         Amount of wheels: 2 kg         Amount of wheels: 2 kg
Color: Grey                    Color: Red                     Color: Green
Fuel type: Petrol              Type: Road                     Type: Sports
------------------------------ ------------------------------ ------------------------------
```

Рис.5. Результат роботи програми

Висновок: За допомогою цього завдання було відпрацьовано на практиці використання основних принципів ООП: інкапсуляції, наслідування, абстракції, поліморфізму (поліморфізм підтипів, проявом якого є віртуальні методи).

Завдання 4: Написати програму на Java, в якій буде створено три класи: «Book», «Library», та «Librarian». Кожен клас представляє окремий аспект системи управління бібліотекою. Клас «Book» представляє книги з назвою та автором, клас

«Library» представляє бібліотеку зі списком книг, а клас «Librarian» представляє бібліотекаря, який відповідає за управління книгами.

Код повинен представити структуру для моделювання бібліотеки. Основне завдання коду – додавання книг до бібліотеки, зберігання їх та відображення списку книг на консолі. Задача полягає в створенні програми для керування книгами у бібліотеці та відображення інформації про них.

Рекомендації:

1) Клас «Book»:
- Повинен містити принаймні два атрибути: title (назва) та author (автор).
- Потрібно надати гетери та сетери для управління зазначеними атрибутами.
- Можна реалізувати метод toString() для виведення інформації про книги.

2) Клас «Library»:
- Повинен містити колекцію книг, наприклад, використовуючи ArrayList<Book>.
- Методи можуть включати: addBook, removeBook, findBookByTitle, listAllBooks, тощо.
- Можна реалізувати додатково пошук книг за автором або жанром.

3) Клас «Librarian»:
- Може включати методи для взаємодії з «Library», наприклад, manageBookAddition(), manageBookRemoval().
- Може містити методи для взаємодії з користувачами, такі як checkOutBook() / returnBook().

При виконанні завдання, студент повинен використати підготовлений список з 5-7 реальних книг з назвами та авторами для заповнення бібліотеки.

При демонстрації взаємодії класів, студент повинен підготувати приклад, який продемонструє, як об'єкти класів можуть взаємодіяти між собою, наприклад, як Librarian викликає методи Library для додавання або видалення книг.

Клас Book має 4 приватних поля – id, author, title та genres. У кожного поля окрім ідентифікатора є сетери. У всіх полів є гетери. В конструктор передаються всі значення для полів, та присвоюються за допомогою сетерів. Ідентифікатор назначається за допомогою методу UUID.randomUUID(). Також є ще два методи: addGenre та deleteGenre, за допомогою яких можна додавати або видаляти жанри зі списку жанрів. В усіх методах є перевірки параметрів. В цьому класі, як і в інших, перевантажений метод toString.

Клас Library має лише одне поле – список books. Є два конструктори – в першому список створюється, в інший передається. Є методи для додавання книг в список, видалення, знаходження книг за назвою, жанром, автором, тощо.

Клас Librarian має два приватних поля – бібліотека та список клієнтів. Бібліотека передається через конструктор, і цим реалізується один з другорядних принципів ООП – агрегація. Вона полягає в принципі взаємодії об’єктів: якщо один об’єкт є частиною іншого, але при цьому може самостійно існувати, то це агрегація. Ще одним прикладом з теми може слугувати полиця та книги. Книги можуть існувати й самі по собі, але є частиною полиці при їх взаємодії.

В класі також є методи для взаємодії клієнту та бібліотеки. Клієнт може брати книги в борг, та повертати їх. Бібліотекар веде облік всіх клієнтів, та оброблює всі дії.

Клас Client потрібен для відображення клієнтів. Всі позичені книги записуються в список borrowedBooks.

Також є допоміжні класи: Printer та CSVReader. Перший потрібен для того, щоб форматувати дані та потім виводити їх у консоль. Другий потрібен для того, щоб прочитати дані з CSV файлу з книгами.

```
ID: 15f7e901-6855-4500-beff-a7561eca83a9
Title: "Harry Potter and the Order of the Phoenix"
Author: J.K. Rowling, Mary GrandPré (Illustrator)
Genres:
    1. Fantasy
    2. Young Adult
    3. Fiction
    4. Magic
    5. Childrens
    6. Adventure
    7. Audiobook
    8. Middle Grade
    9. Classics
    10. Science Fiction Fantasy


ID: 9674eb64-ce96-4cd1-ad5d-573d5612c63b
Title: "To Kill a Mockingbird"
Author: Harper Lee
Genres:
    1. Classics
    2. Fiction
    3. Historical Fiction
    4. School
    5. Literature
    6. Young Adult
```

```
public static void main(String[] args) {
    Printer printer = new Printer();

    Library library = new Library(parseCSV());

    printer.allBooks(library);
```

Рис.6. Виведення інформації про всі книги з бібліотеки

```
Books by title: 1984
ID: 5d955786-61c1-4082-b548-008820748bfb
Title: "1984"
Author: George Orwell
Genres:
    1. Classics
    2. Fiction
    3. Science Fiction
    4. Dystopia
    5. Literature
    6. Novels
    7. Politics
    8. School
    9. Fantasy
    10. Adult
```

```
var title = "1984";
System.out.printf("Books by title: %s%n", title);
printer.bookList(library.findBookByTitle(title));
```

Рис.7. Пошук книг за назвою

Рис.8. Пошук книг за полем «Автор»

Рис.9. Пошук книг за жанром

Наступний приклад: Клієнт бере в борг книгу, потім повертає її. Демонстрація проміжних станів бібліотеки та полів клієнта:

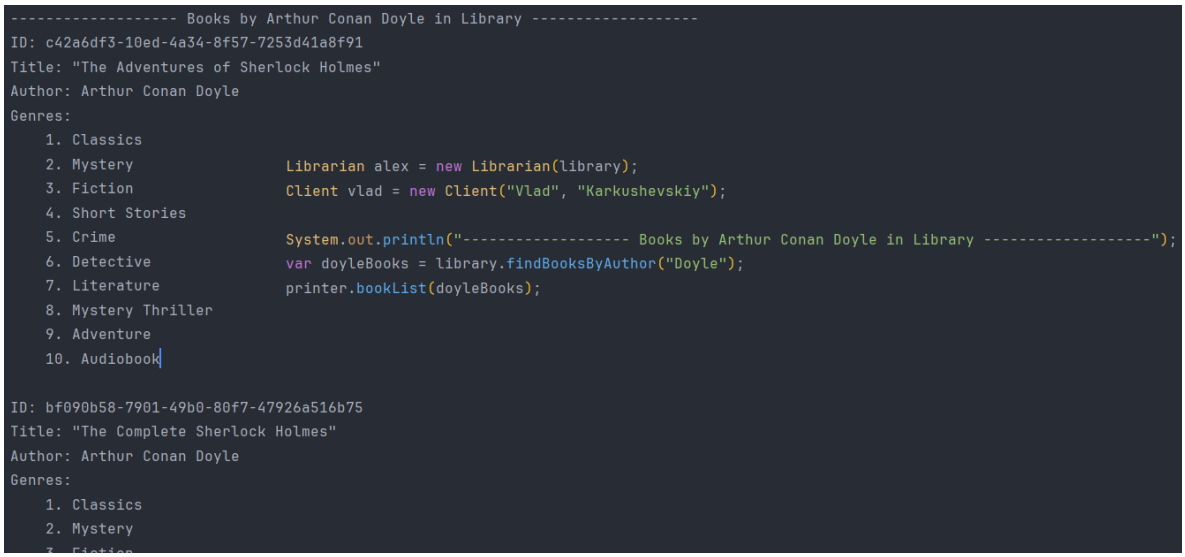

Рис.10. Книги Артура Конана Дойля у бібліотеці

```
------------------- Vlad before borrowing a book -------------------
Client ID: ea73b2ac-92a0-482f-9b80-717e0c606039
First name: Vlad
Last name: Karkushevskiy
Borrowed books: (0)
```

```
System.out.println("-------------------- Vlad before borrowing a book --------------------");
System.out.println(vlad);
```

Рис.11. Клієнт до того, як взяв книгу

```
------------------- Library after borrowing -------------------
ID: bf090b58-7901-49b0-80f7-47926a516b75
Title: "The Complete Sherlock Holmes"
Author: Arthur Conan Doyle
Genres:
    1. Classics
    2. Mystery
    3. Fiction
    4. Crime
    5. Short Stories
    6. Detective
    7. Literature
    8. Mystery Thriller
    9. Historical Fiction
    10. Thriller

------------------- Vlad after borrowing a book -------------------
Client ID: ea73b2ac-92a0-482f-9b80-717e0c606039
First name: Vlad
Last name: Karkushevskiy
Borrowed books: (1)
    1. "The Adventures of Sherlock Holmes" | Arthur Conan Doyle
```

```
var borrowedBook = alex.checkOutBook(vlad, doyleBooks.get(0));

System.out.println("-------------------- Library after borrowing --------------------");
doyleBooks = library.findBooksByAuthor("Doyle");
printer.bookList(doyleBooks);

System.out.println("-------------------- Vlad after borrowing a book --------------------");
System.out.println(vlad);
```

Рис.12. Бібліотека та клієнт, після того як позичив книгу

```
------------------- Vlad after returning a book -------------------
Client ID: ea73b2ac-92a0-482f-9b80-717e0c606039
First name: Vlad
Last name: Karkushevskiy
Borrowed books: (0)

------------------- Library after returning -------------------
ID: bf090b58-7901-49b0-80f7-47926a516b75
Title: "The Complete Sherlock Holmes"
Author: Arthur Conan Doyle
Genres:
    1. Classics
    2. Mystery
    3. Fiction
    4. Crime
    5. Short Stories
    6. Detective
    7. Literature
    8. Mystery Thriller
    9. Historical Fiction
    10. Thriller

ID: c42a6df3-10ed-4a34-8f57-7253d41a8f91
Title: "The Adventures of Sherlock Holmes"
Author: Arthur Conan Doyle
Genres:
```

```
alex.returnBook(vlad, borrowedBook);

System.out.println("-------------------- Vlad after returning a book --------------------");
System.out.println(vlad);

System.out.println("-------------------- Library after returning --------------------");
doyleBooks = library.findBooksByAuthor("Doyle");
printer.bookList(doyleBooks);
```

Рис.13. Бібліотека та клієнт, після того як повернув книгу

Висновок: В цьому завданні були опрацьовані основні принципи ООП, та взаємодія об’єктів між собою.

Додаток 1. Лістинги

Завдання 1.

```
public class Calculator {
    double x;
    double y;
    double z;

    EquationFunction function;

    public Calculator(double x, double y, double z, EquationFunction
function) {
        this.x = x;
        this.y = y;
        this.z = z;

        this.function = function;
    }

    public double solve() {
        return function.apply(x, y, z);
    }

    public String formattedResult() {
        var result = solve();

        var sb = new StringBuilder();
        sb.append(String.format("x = %.4f %n", x));
        sb.append(String.format("y = %.4f %n", y));
        sb.append(String.format("z = %.4f %n", z));
        sb.append(String.format("Result = %.4f", result));

        return sb.toString();
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        this.y = y;
    }

    public double getZ() {
        return z;
    }

    public void setZ(double z) {
        this.z = z;
    }

    public EquationFunction getFunction() {
        return function;
    }

    public void setFunction(EquationFunction function) {
        this.function = function;
    }
```

```
}

public static void main(String[] args) {
        double x, y, z;

        EquationFunction functionC = (a, b, c) -> {
            var firstFactor = Math.pow(a, 3);
            var underTgEquation = Math.pow(Math.pow(a, 2) + b, 3);
            var tgEquation = Math.pow(Math.tan(underTgEquation), 3);
            var firstMember = firstFactor * tgEquation;

            var secondMember = c / (a + b);

            return firstMember + secondMember;
        };

        // First test
        System.out.println("First Test:");
        x = 0.2;
        y = 2.5;
        z = 11;
        var calc1 = new Calculator(x, y, z, functionC);
        System.out.println(calc1.formattedResult());

        System.out.println();

        // Second Test
        System.out.println("Second Test:");
        x = 0.6;
        y = 4.1;
        z = 9;
        var calc2 = new Calculator(x, y, z, functionC);
        System.out.println(calc2.formattedResult());
    }

@FunctionalInterface
public interface EquationFunction {
    double apply(double x, double y, double z);
}
```

Завдання 2.

```
public static void main(String[] args) {
        double x, y, z;

        EquationFunction functionL = (a, b, c) -> {
            var numerator = Math.pow(a, 2) * (a + 2);
            var denominator = c - Math.pow(Math.sin(a + b), 2);

            return numerator / denominator;
        };

        EquationFunction functionN = (a, b, c) -> {
            var firstMember = Math.sqrt(a / b);
            var secondMember = Math.pow(Math.cos(a + c), 2);

            return firstMember + secondMember;
        };

        // First test
        System.out.println("First Test:");
        x = 0.1;
        y = 0.01;
        z = 0.7;
        var calc1 = new Calculator(x, y, z, functionL);
        System.out.println(calc1.formattedResult());
```

```
        System.out.println();
        calc1.setFunction(functionN);
        System.out.println(calc1.formattedResult());

        System.out.println("\n\n");

        // Second Test
        System.out.println("Second Test:");
        x = 1.1;
        y = -1.2;
        z = 4.4;
        var calc2 = new Calculator(x, y, z, functionL);
        System.out.println(calc2.formattedResult());
        System.out.println();
        calc2.setFunction(functionN);
        System.out.println(calc2.formattedResult());
    }
```

Завдання 3.

```
public abstract class Vehicle {
    protected double speed;
    protected double weight;
    protected String color;
    protected int amountOfWheels;

    public Vehicle(double speed, double weight, String color, int
amountOfWheels) {
        setSpeed(speed);
        setWeight(weight);
        setColor(color);
        setAmountOfWheels(amountOfWheels);
    }

    public abstract double calculateMaxSpeed();
    public void printInfo() {
        System.out.println("-".repeat(30));
        System.out.println(this.toString());
        System.out.println("-".repeat(30));
    }

    @Override
    public String toString() {
        var sb = new StringBuilder();

        sb.append(String.format("Transport           type:           %s%n",
this.getClass().getSimpleName()));
        sb.append(String.format("Speed: %.2f km/h%n", this.getSpeed()));
        sb.append(String.format("Weight: %.2f kg%n", this.getWeight()));
        sb.append(String.format("Amount     of     wheels:     %d     kg%n",
this.getAmountOfWheels()));
        sb.append(String.format("Color: %s%n", this.getColor()));

        return sb.toString();
    }

    public double getSpeed() {
        return speed;
    }

    public void setSpeed(double speed) {
        if (speed < 0) {
            throw new IllegalArgumentException("Speed value must be non-
negative");
        }
        this.speed = speed;
```

```
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double weight) {
        if (weight < 0) {
            throw new IllegalArgumentException("Weight value must be non-
negative");
        }
        this.weight = weight;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        if (color.trim().isEmpty()) {
            throw new IllegalArgumentException("Color value must be non-
empty");
        }
        this.color = color;
    }

    public int getAmountOfWheels() {
        return amountOfWheels;
    }

    public void setAmountOfWheels(int amountOfWheels) {
        if (amountOfWheels < 0) {
            throw new IllegalArgumentException("Wheels numbers value must
be non-negative");
        }
        this.amountOfWheels = amountOfWheels;
    }
}

public class Motorbike extends Vehicle {
    protected MotoType type;

    public Motorbike(double speed, double weight, String color, int
numberOfWheels, MotoType type) {
        super(speed, weight, color, numberOfWheels);

        setType(type);
    }

    @Override
    public double calculateMaxSpeed() {
        return speed + switch (type) {
            case Standard, DualPurpose -> 0;
            case Cruiser, Touring, OffRoad -> 5;
            case Sports -> 15;
        };
    }

    @Override
    public String toString() {
        var sb = new StringBuilder();

        sb.append(super.toString());
        sb.append(String.format("Type: %s", this.getType()));

        return sb.toString();
```

```
    }

    public MotoType getType() {
        return type;
    }

    public void setType(MotoType type) {
        this.type = type;
    }
}

public class Car extends Vehicle {
    protected FuelType fuelType;

    public Car(double speed, double weight, String color, int
numberOfWheels, FuelType fuelType) {
        super(speed, weight, color, numberOfWheels);

        setFuelType(fuelType);
    }

    public FuelType getFuelType() {
        return fuelType;
    }

    public void setFuelType(FuelType fuelType) {
        this.fuelType = fuelType;
    }

    @Override
    public double calculateMaxSpeed() {
        return speed + switch (fuelType) {
            case Electro -> 5;
            case Ethanol -> 10;
            case BioDiesel -> 15;
            case Diesel -> 20;
            case Petrol -> 25;
        };
    }

    @Override
    public String toString() {
        var sb = new StringBuilder();

        sb.append(super.toString());
        sb.append(String.format("Fuel type: %s", this.getFuelType()));

        return sb.toString();
    }
}

public class Bicycle extends Vehicle {
    protected BicycleType type;

    public Bicycle(double speed, double weight, String color, int
numberOfWheels, BicycleType type) {
        super(speed, weight, color, numberOfWheels);

        setType(type);
    }

    @Override
    public double calculateMaxSpeed() {
        return speed + switch (type) {
            case Road, Mountain, Gravel, Track -> 15;
            case Kids, Fat, Tricycle, Tandem -> 0;
```

```
            case EBike, Utility, Fitness -> 2;
            case BMX, Triathlon -> 25;
        };
    }

    @Override
    public String toString() {
        var sb = new StringBuilder();

        sb.append(super.toString());
        sb.append(String.format("Type: %s", this.getType()));

        return sb.toString();
    }

    public BicycleType getType() {
        return type;
    }

    public void setType(BicycleType type) {
        this.type = type;
    }
}

public enum BicycleType {
    Road,
    Mountain,
    Gravel,
    EBike,
    Utility,
    Fitness,
    Kids,
    Fat,
    Triathlon,
    Tandem,
    BMX,
    Tricycle,
    Track
}

public enum FuelType {
    Petrol,
    Diesel,
    BioDiesel,
    Electro,
    Ethanol
}

public enum MotoType {
    Standard,
    Cruiser,
    Touring,
    Sports,
    OffRoad,
    DualPurpose
}

public static void main(String[] args) {
        var car = new Car(15, 1900, "Grey", 4, FuelType.Petrol);
        var bicycle = new Bicycle(5, 30, "Red", 2, BicycleType.Road);
        var bike = new Motorbike(10, 100, "Green", 2, MotoType.Sports);

        car.printInfo();

        System.out.println();
```

```
        bicycle.printInfo();

        System.out.println();

        bike.printInfo();
    }
```

Завдання 4.

```
public class Book {
    private final UUID id;
    private String author;
    private String title;
    private List<String> genres;

    public Book(String author, String title, List<String> genres) {
        id = UUID.randomUUID();

        setAuthor(author);
        setTitle(title);
        setGenres(genres);
    }

    public UUID getId() {
        return id;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        if (author.trim().isEmpty()) throw new
IllegalArgumentException("Author must be non-empty");
        this.author = author;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        if (title.trim().isEmpty()) throw new
IllegalArgumentException("Title must be non-empty");
        this.title = title;
    }

    public List<String> getGenres() {
        return genres;
    }

    public void setGenres(List<String> genres) {
        if (genres != null) {
            this.genres = genres;
        } else {
            this.genres = new LinkedList<>();
        }
    }

    public void addGenre(String genre) {
        if (genres.contains(genre)) throw new
IllegalArgumentException("There's already genre like this");
        if (genre.trim().isEmpty()) throw new
IllegalArgumentException("Genre must be non-empty");

        genres.add(genre);
```

```
    }

    public void deleteGenre(String genre) {
        if (!genres.contains(genre)) throw new
IllegalArgumentException("There's no genre like this");
        if (genre.trim().isEmpty()) throw new
IllegalArgumentException("Genre must be non-empty");

        genres.remove(genre);
    }

    @Override
    public String toString() {
        var sb = new StringBuilder();

        sb.append(String.format("ID: %s%n", this.id));
        sb.append(String.format("Title: \"%s\"%n", this.title));
        sb.append(String.format("Author: %s%n", this.author));

        if (genres.isEmpty()) {
            sb.append("Genres: None");
            return sb.toString();
        }

        sb.append(String.format("Genres: %n"));
        for (int i = 0; i < this.genres.size(); i++) {
            var genre = this.genres.get(i);
            sb.append(String.format("\t%d. %s%n", i+1, genre));
        }

        return sb.toString();
    }
}

public class Client {
    private final UUID id;
    private final String firstName;
    private final String lastName;

    private final List<Book> borrowedBooks;

    public Client(String firstName, String lastName) {
        if (firstName.trim().isEmpty() || lastName.trim().isEmpty()) {
            throw new IllegalArgumentException("First or last name is
empty.");
        }

        id = UUID.randomUUID();
        this.firstName = firstName;
        this.lastName = lastName;

        this.borrowedBooks = new LinkedList<>();
    }

    public UUID getId() {
        return id;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
```

```
    public List<Book> getBorrowedBooks() {
        return borrowedBooks;
    }

    public Book borrowBook(Book book) {
        if (book == null) throw new IllegalArgumentException("Book reference
is null.");

        borrowedBooks.add(book);
        return book;
    }

    public Book returnBookById(UUID id) {
        Book desiredBook = null;

        for (var book : borrowedBooks) {
            if (book.getId() != id) continue;

            desiredBook = book;
        }

        if (desiredBook == null) throw new IllegalArgumentException("There's
no book with that ID");

        borrowedBooks.remove(desiredBook);

        return desiredBook;
    }

    public Book returnBook(Book book) {
        if (book == null) throw new IllegalArgumentException("Book reference
is null.");
        if        (!borrowedBooks.contains(book))        throw        new
IllegalArgumentException("Client did not borrow this book.");

        borrowedBooks.remove(book);
        return book;
    }

    @Override
    public String toString() {
        var sb = new StringBuilder();

        sb.append(String.format("Client ID: %s%n", id));
        sb.append(String.format("First name: %s%n", firstName));
        sb.append(String.format("Last name: %s%n", lastName));
        sb.append(String.format("Borrowed          books:          (%d)%n",
borrowedBooks.size()));

        for (int i = 0; i < borrowedBooks.size(); i++) {
            var book = borrowedBooks.get(i);
            sb.append(String.format("\t%d.    \"%s\"    |    %s%n",    i+1,
book.getTitle(), book.getAuthor()));
        }

        return sb.toString();
    }
}

public class Librarian {
    private Library library;
    private final List<Client> clients;

    public Librarian() {
        library = new Library();
        clients = new LinkedList<>();
```

```
    }

    public Librarian(Library library) {
        this();

        if (library != null) this.library = library;
    }

    public void manageBookAddition(Book book) {
        library.addBook(book);
    }

    public void manageBookRemoval(Book book) {
        library.removeBook(book);
    }

    public Book checkOutBook(Client client, Book book) {
        manageBookRemoval(book);
        var borrowed = client.borrowBook(book);

        if (!clients.contains(client)) clients.add(client);

        return borrowed;
    }

    public Book returnBook(Client client, Book book) {
        var returned = client.returnBook(book);
        manageBookAddition(returned);

        return returned;
    }
}

public class Library {
    private final List<Book> books;

    public Library() {
        this.books = new LinkedList<>();
    }

    public Library(List<Book> books) {
        if (books != null) this.books = books;
        else this.books = new LinkedList<>();
    }

    public void addBook(Book book) {
        if (book == null) throw new IllegalArgumentException("Book object
has null reference");

        books.add(book);
    }

    public void removeBookById(UUID id) {
        for (Book b : books) {
            if (b.getId() != id) continue;

            books.remove(b);
            return;
        }

        throw new IllegalArgumentException("There's no book with this id");
    }

    public void removeBook(Book book) {
        if (book == null) throw new IllegalArgumentException("Book object
has null reference");
```

```
        if (!books.contains(book)) throw new
IllegalArgumentException("There's no book with this id");

        books.remove(book);
    }

    public List<Book> findBookByTitle(String title) {
        var list = new LinkedList<Book>();

        for (var book : books) {
            if
(book.getTitle().toLowerCase().contains(title.toLowerCase()))
list.add(book);
        }

        return list;
    }

    public List<Book> findBooksByAuthor(String author) {
        var list = new LinkedList<Book>();

        for (var book : books) {
            if
(book.getAuthor().toLowerCase().contains(author.toLowerCase()))
list.add(book);
        }

        return list;
    }

    public List<Book> findBooksByGenre(String genre) {
        var list = new LinkedList<Book>();

        for (var book : books) {
            for (var bookGenre : book.getGenres()) {
                if  (bookGenre.toLowerCase().contains(genre.toLowerCase()))
list.add(book);
            }
        }

        return list;
    }

    public List<Book> listAllBooks() {
        return books;
    }
}

public class Printer {
    public void allBooks(Library library) {
        System.out.println("All books available: ");
        for (var book : library.listAllBooks()) {
            System.out.println(book + "\n");
        }
    }

    public void bookList(List<Book> books) {
        for (var book : books) {
            System.out.println(book);
        }
    }
}

public class CSVBookReader {
    public static List<Book> parse(String filename) throws IOException {
        Reader in = new FileReader(filename);
```

```
        CSVFormat csvFormat = CSVFormat.DEFAULT.builder()
                .setDelimiter(',')
                .setHeader(BookHeaders.class)
                .setSkipHeaderRecord(true)
                .build();

        Iterable<CSVRecord> records = csvFormat.parse(in);

        var books = new LinkedList<Book>();

        for (CSVRecord record : records) {
            String author = record.get(BookHeaders.author);
            String title = record.get(BookHeaders.title);

            var genres = parseStringArray(record.get(BookHeaders.genres));

            books.add(new Book(author, title, genres));
        }

        return books;
    }

    private static List<String> parseStringArray(String str) {
        var textArray = str.substring(1, str.length() - 1);
        var array = textArray.split(", ");

        for (int i = 0; i < array.length; i++) {
            array[i] = array[i].replaceAll("'", "");
        }

        return Arrays.asList(array);
    }
}

enum BookHeaders{
    title, author, genres
}

public class Main {
    public static void main(String[] args) {
        Printer printer = new Printer();

        Library library = new Library(parseCSV());

        printer.allBooks(library);

        var title = "1984";
        System.out.printf("Books by title: %s%n", title);
        printer.bookList(library.findBookByTitle(title));

        var author = "Tolkien";
        System.out.printf("Books by author: %s%n", author);
        printer.bookList(library.findBooksByAuthor(author));

        var genre = "Scotland";
        System.out.printf("Books by genre: %s%n", genre);
        printer.bookList(library.findBooksByGenre(genre));


        Librarian alex = new Librarian(library);
        Client vlad = new Client("Vlad", "Karkushevskiy");

        System.out.println("------------------- Books by Arthur Conan Doyle
in Library -------------------");
        var doyleBooks = library.findBooksByAuthor("Doyle");
```

```
        printer.bookList(doyleBooks);

        System.out.println("------------------- Vlad before borrowing a book
-------------------");
        System.out.println(vlad);

        var borrowedBook = alex.checkOutBook(vlad, doyleBooks.get(0));

        System.out.println("------------------- Library after borrowing ---
----------------");
        doyleBooks = library.findBooksByAuthor("Doyle");
        printer.bookList(doyleBooks);

        System.out.println("------------------- Vlad after borrowing a book
-------------------");
        System.out.println(vlad);

        alex.returnBook(vlad, borrowedBook);

        System.out.println("------------------- Vlad after returning a book
-------------------");
        System.out.println(vlad);

        System.out.println("------------------- Library after returning ---
----------------");
        doyleBooks = library.findBooksByAuthor("Doyle");
        printer.bookList(doyleBooks);
    }

    private static List<Book> parseCSV() {
        final String filename = "./src/Lab4/Task4/Data/books.csv";

        try {
            return CSVBookReader.parse(filename);
        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }

        return null;
    }
}
```