МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ПОЛТАВСЬКИЙ ПОЛІТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ
НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Циклова комісія дисциплін програмної інженерії

ЗВІТ

з навчальної практики
«Практика з об’єктно-орієнтованого програмування»

Виконав: здобувач освіти 3 курсу,
групи 35
напрям підготовки: 121 «Інженерія програмного забезпечення»
спеціалізації «Розробка програмного забезпечення»
Грисенко А.О.
(прізвище та ініціали)

Керівник _____ Олійник В.В.
(підпис) (прізвище та ініціали)

Полтава – 2024

ЗМІСТ

ВСТУП

Практика з об'єктно-орієнтованого програмування є ключовим етапом у формуванні та розвитку професійних навичок студентів, які обрали спеціальність у сфері інформаційних технологій. Метою даної практики є не лише закріплення теоретичних знань, але й набуття практичних навичок та досвіду, що відповідають вимогам і особливостям сучасної організації професійної діяльності в галузі програмування.

Ця практика спрямована на вироблення у студентів ряду ключових компетенцій, які стануть основою для успішної реалізації їхньої майбутньої професійної діяльності. Серед цих компетенцій основними є:

- вивчення об'єктно-орієнтованого програмування, теорії алгоритмів і структур даних, технологій розробки програмного забезпечення та інших спеціальних дисциплін;
- отримання можливості відпрацьовувати свої навички під час розробки реального програмного продукту, використовуючи принципи об'єктно-орієнтованого програмування;
- засвоєння методів аналізу об'єктно-орієнтованого програмного коду, виявлення та вирішення можливих проблем та покращення ефективності;
- ознайомлення з усіма етапами життєвого циклу розробки програмного продукту, включаючи планування, реалізацію та тестування;
- розробка комплекту проектної документації, яка відображатиме основні результати практики та вміння студента;
- створення функціонального та ефективного об'єктно-орієнтованого програмного коду.

Ці завдання практики допоможуть студентам сформувати в собі високий рівень професійної підготовки, готовності до вирішення завдань у реальних умовах роботи та розвитку вмінь самостійно поновлювати свої знання та творчо застосовувати їх у практичній діяльності.

1. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №1

1.1. Постановка завдання

Створити програму на мові програмування Java в якій потрібно:

— розробити клас, що серіалізується, для зберігання параметрів і результатів обчислень. Використовуючи агрегування, розробити клас для знаходження рішення задачі;

— розробити клас для демонстрації в діалоговому режимі збереження та відновлення стану об'єкта, використовуючи серіалізацію. Показати особливості використання transient полів;

— розробити клас для тестування коректності результатів обчислень та серіалізації/десеріалізації. Використовувати докладні коментарі для автоматичної генерації документації засобами javadoc;

— індивідуальне завдання: визначити суму площ рівностороннього трикутника та рівностороннього прямокутника за заданою довжиною сторони у двійковій системі числення.

1.3. Текст програми з коментарями

```java
package org.example;
import java.io.Serializable;

public class Calc implements Serializable, CalcDisplay {
    private static final long serialVersionUID = 1L;
    // Параметри
    private double a;
    private double b;
    // Результат
    private double output;
    // Конструктор
    public Calc(double a, double b) {
        this.a = a;
        this.b = b;
    }
    // Гетери і сетери
    public double getA() {
        return a;
    }

    public void setA(double a) {
        this.a = a;
    }

    public double getB() {
        return b;
    }

    public void setB(double b) {
        this.b = b;
    }

    public double getOutput() {
        return output;
    }

    public void setResult(double output) {
        this.output = output;
    }
}
```

Рисунок 1.1 – Реалізація серіалізації.

```java
package org.example;

public class Solv {
    public Calc data;
    public double x;
    public double y;
    public double z;

    public Solv(double a, double b) {
        x = a;
        y = b;
        data = new Calc(a, b);
    }
    public void solve() {
        z = x - y;
        data.setResult(z);
    }
    public Calc getData() {
        return data;
    }
    public void setData(Calc data) {
        this.data = data;
    }
}
```

Рисунок 1.2 – Реалізація обчислення.

```java
package org.example;
import java.io.*;

public class Main {
    public static void main(String[] args) {
        // Створення Solv
        Solv solver = new Solv(60, 23);

        // Виклик методу solve
        solver.solve();

        // Серіалізація
        saveObject(solver.getData(), "data.ser");

        // Десеріалізація
        Calc restoredData = loadObject("data.ser");

        // Вивід результатів
        System.out.println("Parameter 1: " + restoredData.getA());
        System.out.println("Parameter 2: " + restoredData.getB());
        System.out.println("Output: " + restoredData.getOutput());
    }

    private static void saveObject(Calc data, String fileName) {
        try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(fileName))) {
            outputStream.writeObject(data);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static Calc loadObject(String fileName) {
        Calc data = null;
        try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(fileName))) {
            data = (Calc) inputStream.readObject();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        return data;
    }
}
```

Рисунок 1.3 – Реалізація серіалізації та десеріалізації.

```java
package org.example;
import java.util.Scanner;

public class IndividualTask {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Введіть довжину сторони (двійкове число): ");
        String binarySideLength = scanner.nextLine();


        int sideLength = Integer.parseInt(binarySideLength, 2);

        double triangleArea = (Math.sqrt(3) / 4) * Math.pow(sideLength, 2);

        double squareArea = Math.pow(sideLength, 2);

        double totalArea = triangleArea + squareArea;

        System.out.println("Сума площ: " + totalArea);
    }
}
```

Рисунок 1.4 – Індивідуальне завдання.

1.4. Результати роботи програми

```
Parameter 1: 60.0
Parameter 2: 23.0
Output: 37.0

Process finished with exit code 0
```

Рисунок 1.5 – Відображення працездатності програми.

```
Введіть довжину сторони (двійкове число): 10101
Сума площ: 631.9586015344687

Process finished with exit code 0
```

Рисунок 1.6 – Відображення працездатності програми.

2. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №2

2.1. Постановка завдання

— як основа використовувати вихідний текст проекту попередньої лабораторної роботи. Забезпечити розміщення результатів обчислень у колекції з можливістю збереження/відновлення;

— використовуючи шаблон проектування Factory Method (Virtual Constructor), розробити ієрархію, що передбачає розширення рахунок додавання нових відображуваних класів;

— розширити ієрархію інтерфейсом "фабрикованих" об'єктів, що представляє набір методів для відображення результатів обчислень;

— реалізувати ці методи виведення результатів у текстовому виді;

— розробити тареалізувати інтерфейс для "фабрикуючого" методу.

2.2. Текст програми з коментарями.

```java
package org.example;
import java.io.*;
import java.util.List;
import java.util.LinkedList;

public class Main {
    public static void main(String[] args) {
        // Створення списку для зберігання об'єктів
        List<Calc> calculations = new LinkedList<>();
        // Створення об'єктів та передача їм параметрів
        Solv solver1 = new Solv(60, 3);
        Solv solver2 = new Solv(23, 34);

        solver1.solve();
        solver2.solve();

        calculations.add(solver1.getData());
        calculations.add(solver2.getData());
        // Серіалізація списку
        saveObject(calculations, "data_collection.ser");

        // Десеріалізація об'єктів
        List<Calc> restoredCalculations = loadObject("data_collection.ser");
        // Виведення параметрів та результатів
        for (Calc restoredData : restoredCalculations) {
            restoredData.displayParameters(restoredData);
            restoredData.displayResult(restoredData);
            System.out.println("--------");
        }
    }
    // Метод для серіалізації списку
    private static void saveObject(List<Calc> data, String fileName) {
        try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(fileName))) {
            outputStream.writeObject(data);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    // Метод для десеріалізації
    private static List<Calc> loadObject(String fileName) {
        List<Calc> data = null;
        try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(fileName))) {
            data = (List<Calc>) inputStream.readObject();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        return data;
    }
}
```

Рисунок 2.1 – Реалізація серіалізації та десеріалізації.

2.3. Результати роботи програми

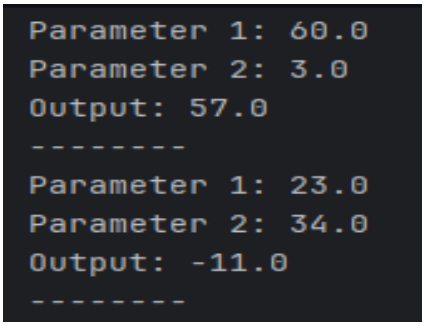

Рисунок 2.2 – Відображення працездатності програми.

3. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №3

3.1. Постановка завдання

— за основу використовувати вихідний текст проекту попередньої лабораторної роботи. Використовуючи шаблон проектування Factory Method (Virtual Constructor), розширити ієрархію похідними класами, реалізують методи для подання результатів у вигляді текстової таблиці. Параметри відображення таблиці мають визначатися користувачем;

— продемонструвати заміщення (перевизначення, overriding), поєднання (перевантаження, overloading), динамічне призначення методів (Пізнє зв'язування, поліморфізм, dynamic method dispatch);

— забезпечити діалоговий інтерфейс із користувачем;

— розробити клас для тестування основної функціональності;

— використати коментарі для автоматичної генерації документації засобами javadoc

3.2. Текст програми з коментарями

```
package org.example;


public class MainTest {
    public static void main(String[] args) {
        testSimpleTableDisplay();
        testFancyTableDisplay();
    }

    private static void testSimpleTableDisplay() {
        System.out.println("Testing Simple:");
        ExtCalcFactory factory = new ExtCalcFactoryImpl();

        Solv solver = new Solv(10, 23);
        solver.solve();
        Calc data = solver.getData();

        TableDisplay tableDisplay = new SimpleTableDisplay();
        tableDisplay.displayTable(data, new String[]{"Param1", "Param2", "Result"});
    }

    private static void testFancyTableDisplay() {
        System.out.println("Testing Fancy:");
        ExtCalcFactory factory = new ExtCalcFactoryImpl();

        Solv solver = new Solv(20, 4);
        solver.solve();
        Calc data = solver.getData();

        TableDisplay tableDisplay = new FancyTableDisplay();
        tableDisplay.displayTable(data, new String[]{"Param1", "Param2", "Result"});
    }
}
```

Рисунок 3.1 – Перевірка функціональності.

3.3. Результати роботи програми

```
Testing Simple:
Simple Table Display:
Param1  Param2  Result
---------------------------------
10.0    23.0    -13.0
---------------------------------
Testing Fancy:
Fancy Table Display:
Param1              Param2              Result
-----------------------------------------------------
20.0                4.0             16.0
-----------------------------------------------------
```

Рисунок 3.2 – Відображення працездатності програми.

4. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №4

4.1. Постановка завдання

— реалізувати можливість скасування (undo) операцій (команд);

— продемонструвати поняття "макрокоманда";

— при розробці програми використовувати шаблон Singletone;

— забезпечити діалоговий інтерфейс із користувачем;

— розробити клас для тестування функціональності програми.

4.2. Текст програми з коментарями

```java
package org.example;

public interface Command {
    void execute();
    void undo();
}

// Реалізація конкретної команди для обчислення
class SolveCommand implements Command {
    private Solv solver;

    public SolveCommand(Solv solver) {
        this.solver = solver;
    }

    @Override
    public void execute() {
        solver.solve();
    }

    @Override
    public void undo() {
        // Скасування операції solve необхідно для скасування результату
        solver.getData().setResult(Double.NaN);
    }
}

// Реалізація конкретної команди для зміни параметрів обчислення
class ChangeParamsCommand implements Command {
    private Solv solver;
    private double oldX;
    private double oldY;

    public ChangeParamsCommand(Solv solver, double newX, double newY) {
        this.solver = solver;
        this.oldX = solver.x;
        this.oldY = solver.y;
        solver.x = newX;
        solver.y = newY;
    }

    @Override
    public void execute() {
        solver.solve();
    }

    @Override
    public void undo() {
        solver.x = oldX;
        solver.y = oldY;
        solver.getData().setResult(Double.NaN);
    }
}
```

Рисунок 4.1 – Реалізації команд для виконання та скасування операцій обчислення.

```java
package org.example;

import java.util.Stack;

public class CommandManager {
    private static CommandManager instance;
    private Stack<Command> commandHistory;

    private CommandManager() {
        this.commandHistory = new Stack<>();
    }

    public static CommandManager getInstance() {
        if (instance == null) {
            instance = new CommandManager();
        }
        return instance;
    }

    public void executeCommand(Command command) {
        command.execute();
        commandHistory.push(command);
    }

    public void undoLastCommand() {
        if (!commandHistory.isEmpty()) {
            Command lastCommand = commandHistory.pop();
            lastCommand.undo();
        } else {
            System.out.println("Немає доступних операцій для скасування.");
        }
    }
}
```

Рисунок 4.2 – Управління виконанням та скасуванням команд.

```java
package org.example;

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

public class MacroCommand implements Command {
    private List<Command> commands;

    public MacroCommand() {
        this.commands = new ArrayList<>();
    }

    public void addCommand(Command command) {
        commands.add(command);
    }

    @Override
    public void execute() {
        for (Command command : commands) {
            command.execute();
        }
    }

    @Override
    public void undo() {
        // Скасування макрокоманди викликає скасування всіх вкладених команд
        for (int i = commands.size() - 1; i >= 0; i--) {
            commands.get(i).undo();
        }
    }
}
```

Рисунок 4.3 – Реалізація макрокоманди.

4.3. Результати роботи програми

```
Виберіть тип таблиці:
1. Simple Table Display
2. Fancy Table Display
2
Fancy Table Display:
Param1              Param2              Result
-----------------------------------------------------
60.0                3.0             57.0
-----------------------------------------------------
Fancy Table Display:
Param1              Param2              Result
-----------------------------------------------------
23.0                34.0                -11.0
-----------------------------------------------------
```

Рисунок 4.4 – Відображення працездатності програми.

5. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №5

5.1. Постановка завдання

— продемонструвати можливість паралельної обробки елементів колекції (пошук мінімуму, максимуму, обчислення середнього значення, відбір за критерієм, статистична обробка тощо);

— управління чергою завдань (команд) реалізувати за допомогою шаблону Worker Thread.

5.2. Текст програми з коментарями

```
public class MainTest {
    public static void main(String[] args) {
        testMainFunctionality();
    }

    private static void testMainFunctionality() {
        // Виведення повідомлення про тестування основного функціоналу
        System.out.println("Тестування основного функціоналу:");
        // Створення об'єкта solver класу Solv
        Solv solver = new Solv();
        // Виклик методу для розрахунку площ
        solver.calculateAreas();

        // Створення об'єкта tableDisplay класу SimpleTableDisplay
        TableDisplay tableDisplay = new SimpleTableDisplay();
        // Виведення початкових результатів
        System.out.println("\nПочаткові результати:");
        // Відображення даних у вигляді таблиці
        tableDisplay.displayTable(solver.getData(), new String[]{"Трикутник", "Прямокутник", "Сума площ"});

        // Виклик методу для зміни параметрів
        executeChangeParamsCommand(solver, 15.0);

        // Виклик методу для скасування останньої команди
        undoLastCommand(solver);

        // Виведення результатів після зміни параметрів
        System.out.println("\nРезультати після зміни параметрів:");
        // Відображення оновлених даних у вигляді таблиці
        tableDisplay.displayTable(solver.getData(), new String[]{"Трикутник", "Прямокутник", "Сума площ"});
    }

    private static void executeChangeParamsCommand(Solv solver, double newSideLength) {
        // Отримання екземпляра CommandManager
        CommandManager commandManager = CommandManager.getInstance();

        // Створення команди для зміни параметрів
        ChangeParamsCommand changeParamsCommand = new ChangeParamsCommand(solver, newSideLength);
        // Виконання команди
        commandManager.executeCommand(changeParamsCommand);

        // Виведення результатів після зміни параметрів
        System.out.println("\nРезультати після скасування останньої команди:");
    }

    private static void undoLastCommand(Solv solver) {
        // Отримання екземпляра CommandManager
        CommandManager commandManager = CommandManager.getInstance();

        // Скасування останньої команди
        commandManager.undoLastCommand();

        // Виведення результатів після скасування останньої команди
        System.out.println("\nResults After Undoing Last Command:");
    }
}
```

Рисунок 5.1 – Тестування розроблених класів.

5.3. Результати роботи програми

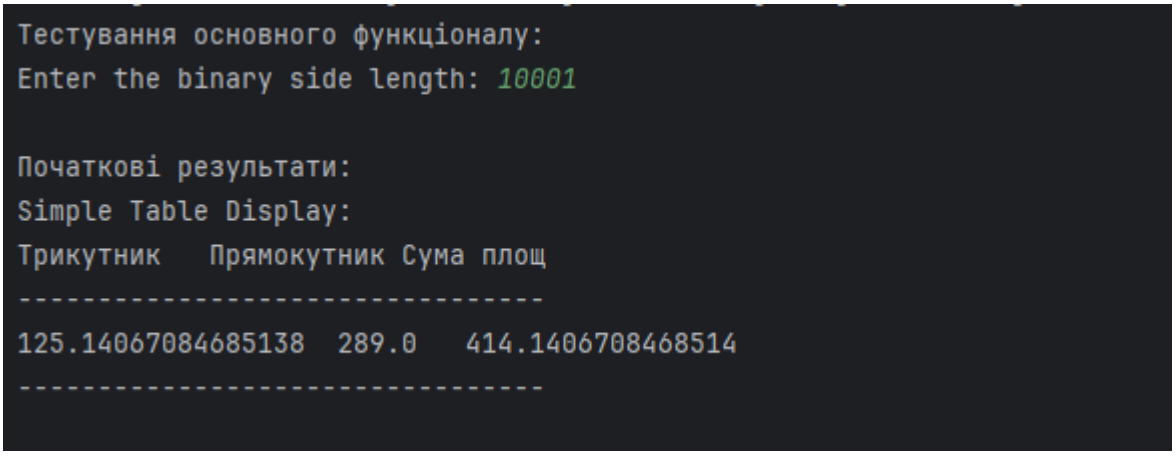

Рисунок 5.2 – Виконання роботи програми.

6. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ №6

6.1. Постановка завдання

— розробити ієрархію класів відповідно до шаблону Observer (java) та продемонструвати можливість обслуговування розробленої раніше колекції (об'єкт, що спостерігається, Observable) різними (не менше двох) спостерігачами (Observers) – відстеження змін, упорядкування, висновок, відображення і т.д.;

— при реалізації ієрархії класів використати інструкції (Annotation). Відзначити особливості різних політик утримання анотацій (annotation retention policies). Продемонструвати підтримку класів концепції рефлексії (Reflection);

— використовуючи раніше створені класи, розробити додаток, що відображає результати обробки колекції об'єктів у графічному вигляді;

— забезпечити діалоговий інтерфейс з користувачем та перемальовування графіка під час зміни значень елементів колекції.

6.2. Текст програми з коментарями

```java
package com.rx.javajxpr;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("hello-view.fxml"));
        Parent root = fxmlLoader.load();

        // Отримуємо контролер після завантаження FXML
        HelloController controller = fxmlLoader.getController();
        controller.setStage(stage); // Передаємо stage контролеру

        Scene scene = new Scene(root);
        stage.setTitle("Калькулятор суми площ");
        stage.setScene(scene);
        stage.show();
    }

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

Рисунок 6.1 – Реалізація графічного інтерфейсу.

```java
package com.rx.javajxpr;
import java.util.ArrayList;
import java.util.List;

// Інтерфейс спостерігача (Observer)
interface Observer {
    void update(List<Integer> data);
}

// Клас спостережуваного об'єкта (Observable)
public class Observable {
    private List<Observer> observers = new ArrayList<>();
    private List<Integer> collection = new ArrayList<>();

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void removeObserver(Observer observer) {
        observers.remove(observer);
    }

    public void addData(int data) {
        collection.add(data);
        notifyObservers();
    }

    private void notifyObservers() {
        for (Observer observer : observers) {
            observer.update(collection);
        }
    }
}
```

Рисунок 6.2 – Реалізація Observer.

6.3. Результати роботи програми

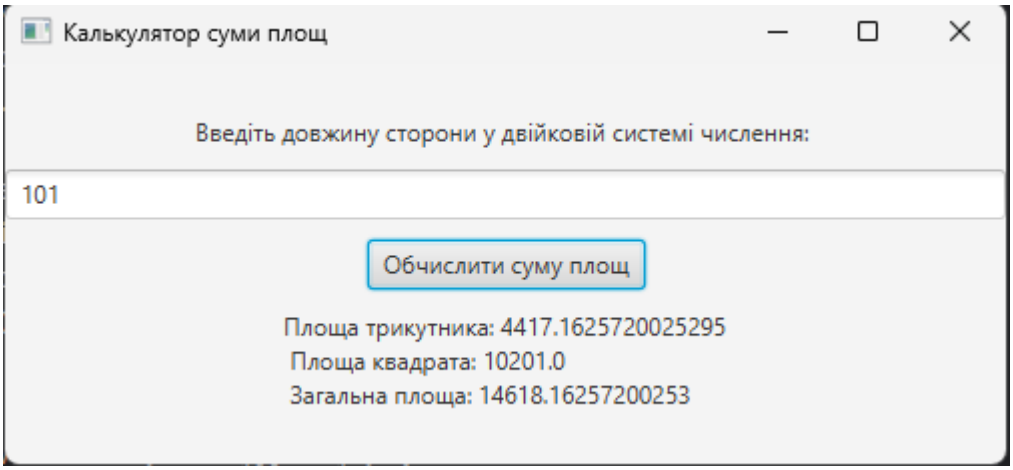

Рисунок 6.3 – Виконання роботи програми.

ВИСНОВКИ

Протягом практики було здійснено глибоке вивчення об’єктно-орієнтованого програмування, теорії алгоритмів, структур даних та сучасних технологій розробки програмного забезпечення. Я отримав цінний досвід, відпрацювавши навички на реальному програмному продукті, що дозволило застосувати теоретичні знання на практиці. Було освоєно методи аналізу програмного коду, ідентифікації та вирішення проблем, що сприяло підвищенню ефективності розробки. Знання про життєвий цикл програмного продукту було поглиблено, включаючи всі етапи від планування до тестування. Розроблено комплект проектної документації, який відображає ключові досягнення практики та демонструє професійні вміння. В результаті, я набув здатності створювати функціональний та ефективний об’єктно-орієнтований програмний код.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. МЕТОДИЧНІ ВКАЗІВКИ ДО ПРОХОДЖЕННЯ ПРАКТИКИ З «ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ» ДЛЯ ЗДОБУВАЧІВ ОСВІТИ НАПРЯМУ ПІДГОТОВКИ 121 «ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ» СПЕЦІАЛІЗАЦІЇ «РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ» [Текст]: МЕТОДИЧНІ ВКАЗІВКИ / Відокремлений структурний підрозділ «Полтавський політехнічний фаховий коледж Національного технічного університету «Харківський політехнічний інститут»; [уклад.: В.В.Олійник, А.О.Зелінська]. – Полтава: ВСП «ППФК НТУ «ХПІ», 2022. – 27 с.
2. Effective Java / за ред. Joshua Bloch. United States, 2017. 416 с.
3. Java: A Beginner's Guide / за ред. Herbert Schildt. United States, 2018. 648 с.
4. Java Concurrency in Practice / за ред. Brian Goetz . United States, 2006. 432 с.
5. Clean Code: A Handbook of Agile Software Craftsmanship / за ред. Robert C. Martin . United States, 2008. 464 с.
6. Oracle Java Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/en/java/ – Назва з екрану.
7. Java Code Geeks [Електронний ресурс] – Режим доступу до ресурсу: https://www.javacodegeeks.com/ – Назва з екрану.
8. Stack Overflow [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/ – Назва з екрану.

ДОДАТОК А. РЕПОЗИТОРІЙ ПРОГРАМИ

Рисунок А.1 – QR код на повний лістинг програми