|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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»  Розробка автоматизованої системи керування апаратами | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | | | | | | |
|  | Виконав: здобувач освіти 3 курсу, | | | | | |
|  | групи | 35 | |  | | |
|  | напрям підготовки: 121 «Інженерія | | | | | |
|  | програмного забезпечення» | | | | | |
|  | спеціалізації «Розробка програмного | | | | | |
|  | забезпечення» | | | | | |
|  | Дядечко О.С. | | | | | |
|  | (прізвище та ініціали) | | | | | |
|  | Керівник | |  | |  | В.В. Олійник |
|  |  | | (підпис) | |  | (прізвище та ініціали) |
|  | | | | | | |
| Полтава – 2024 | | | | | | |

**ЗМІСТ**

[ВСТУП 4](#_Toc161896637)

[1. ПОСТАНОВКА ЗАВДАННЯ 5](#_Toc161896638)

[1.1 Основні вимоги до програми 5](#_Toc161896639)

[1.2 Особливості програми 5](#_Toc161896640)

[2. ОПИС ВИКОРИСТАНИХ МЕТОДІВ ПРОГРАМУВАННЯ 7](#_Toc161896641)

[3. ТЕКСТ ПРОГРАМИ З КОМЕНТАРЯМИ 8](#_Toc161896642)

[3.1. СalculatorApp.java 8](#_Toc161896643)

[3.2. ParametersAndResults.java 8](#_Toc161896644)

[3.3. CalculationResult.java 8](#_Toc161896645)

[3.4. AbstractCalculator.java 9](#_Toc161896646)

[3.5. AdditionCalculator.java 9](#_Toc161896647)

[3.6. MultiplicationCalculator.java 9](#_Toc161896648)

[3.7. ResultFormatter.java 9](#_Toc161896649)

[3.8. TextResultFormatter.java 10](#_Toc161896650)

[3.9. HtmlResultFormatter.java 10](#_Toc161896651)

[3.10. TextTableFormatter.java 10](#_Toc161896652)

[3.11. CalculatorFactory.java 10](#_Toc161896653)

[3.12. Command.java 11](#_Toc161896654)

[3.13. UndoableCommand.java 11](#_Toc161896655)

[3.14. CommandHistory.java 11](#_Toc161896656)

[3.15. Calculator.java 12](#_Toc161896657)

[3.16. ConsoleUI.java 14](#_Toc161896658)

[3.17. CalculatorTest.java 15](#_Toc161896659)

[3.18. ParallelProcessing.java 15](#_Toc161896660)

[3.19. TaskQueue.java 16](#_Toc161896661)

[3.20. Main.java 17](#_Toc161896662)

[4. РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ 18](#_Toc161896663)

[ВИСНОВКИ 20](#_Toc161896664)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 21](#_Toc161896665)

[**ДОДАТОК А. ПОСИЛАННЯ НА GITHUB РЕПОЗИТОРІЙ** 22](#_Toc161896666)

# ВСТУП

Метою практики є закріплення теоретичних знань із спеціальних і орієнтованих на організацію професійної діяльності дисциплін, набуття практичних навичок і досвіду відповідно до профілю обраної спеціальності, виховання потреби систематично поновлювати свої знання та творчо застосовувати їх у практичній діяльності.

Завданнями практики з «Об’єктно-орієнтованого програмування» є:

* Поглиблення і закріплення теоретичних знань з Об’єктно орієнтовано програмування, Теорії алгоритмів і структури даних, Технологій розробки програмного забезпечення та інших спеціальних дисциплін;
* Засвоєння практичних навичок з об’єктно-орієнтованого програмування;
* Вивчення основних інструментів об’єктно-орієнтованого програмування та їх застосування в ході розробки реального програмного продукту;
* Здійснення аналізу об’єктно-орієнтованого програмного коду;
* Отримання практичних навичок створення об’єктно-орієнтованого програмного коду;
* Ознайомлення з життєвим циклом розробки програмного продукту;
* Підготовка звіту – комплекту проектної документації, в якому необхідно відобразити основні результати практики;
* Створення програмного продукту.

Практика з «Об’єктно-орієнтованого програмування» покликана сформувати професійні вміння, навички приймати самостійні рішення на певних ділянках роботи (або з конкретних питань) в реальних виробничих умовах шляхом виконання окремих функцій і завдань, властивих майбутній професії.

# 1. ПОСТАНОВКА ЗАВДАННЯ

## 1.1 Основні вимоги до програми

Розробка класу для зберігання параметрів і результатів обчислень. Створити клас, що серіалізується, який буде використовуватися для зберігання параметрів і результатів обчислень. Використовуючи агрегацію, необхідно реалізувати клас для знаходження рішення задачі.

Реалізація демонстраційного класу збереження та відновлення стану об'єкта. Розробити клас, який демонструватиме в діалоговому режимі збереження та відновлення стану об'єкта за допомогою серіалізації. В цьому класі слід продемонструвати особливості використання transient полів.

Оформити звіт з практики згідно методичним рекомендаціям.

## 1.2 Особливості програми

Реалізація шаблону проектування Factory Method. Використовуючи шаблон проектування Factory Method (Virtual Constructor), розробити ієрархію класів, яка передбачає розширення можливостей рахунку додавання нових відображуваних класів.

Розширення ієрархії інтерфейсом "фабрикованих" об'єктів. Розробити розширену ієрархію, що включає інтерфейс "фабрикованих" об'єктів, що містить набір методів для відображення результатів обчислень. Реалізувати методи виведення результатів у текстовому форматі.

Реалізація можливості скасування операцій. Забезпечити можливість скасування (undo) операцій (команд), використовуючи шаблон проектування Command та Singletone для керування історією команд.

Паралельна обробка елементів колекції. Продемонструвати можливість паралельної обробки елементів колекції, включаючи пошук мінімуму, максимуму, обчислення середнього значення тощо. Управління чергою завдань (команд) реалізувати за допомогою шаблону Worker Thread.

Реалізація класу для тестування функціональності програми. Розробити клас для тестування основної функціональності програми, включаючи перевірку коректності результатів обчислень та серіалізації/десеріалізації.

# 2. ОПИС ВИКОРИСТАНИХ МЕТОДІВ ПРОГРАМУВАННЯ

При виконанні даного завдання використовувалися такі методи та конструкції:

* Шаблон Factory Method (Virtual Constructor)
* Шаблон Command
* Шаблон Singleton
* Шаблон Worker Thread
* Агрегація
* Серіалізація та десеріалізація
* Паралельна обробка
* Анотації (Annotations)

# 3. ТЕКСТ ПРОГРАМИ З КОМЕНТАРЯМИ

## 3.1. СalculatorApp.java

import java.util.InputMismatchException;

import java.util.Scanner;

public class CalculatorApp {

private static final Calculator calculator = Calculator.getInstance();

private static final ConsoleUI ui = new ConsoleUI();

public static void main(String[] args) {

ui.run();

}

}

## 3.2. ParametersAndResults.java

import java.io.Serializable;

import java.util.ArrayList;

import java.util.List;

public class ParametersAndResults implements Serializable {

private double[] parameters;

private List<CalculationResult> results;

public ParametersAndResults(double[] parameters, double[] results) {

this.parameters = parameters;

this.results = new ArrayList<>();

for (double result : results) {

this.results.add(new CalculationResult("Default", result));

}

}

public void addResult(CalculationResult result) {

results.add(result);

}

public List<CalculationResult> getResults() {

return results;

}

}

## 3.3. CalculationResult.java

import java.io.Serializable;

public class CalculationResult implements Serializable {

private String type;

private double value;

public CalculationResult(String type, double value) {

this.type = type;

this.value = value;

}

public String getType() {

return type;

}

public double getValue() {

return value;

}

}

## 3.4. AbstractCalculator.java

public abstract class AbstractCalculator {

public abstract double calculate(double a, double b);

public String getType() {

return "Abstract Calculator";

}

}

## 3.5. AdditionCalculator.java

public class AdditionCalculator extends AbstractCalculator {

@Override

public double calculate(double a, double b) {

return a + b;

}

@Override

public String getType() {

return "Addition Calculator";

}

}

## 3.6. MultiplicationCalculator.java

public class MultiplicationCalculator extends AbstractCalculator {

@Override

public double calculate(double a, double b) {

return a \* b;

}

@Override

public String getType() {

return "Multiplication Calculator";

}

}

## 3.7. ResultFormatter.java

public abstract class ResultFormatter {

public abstract String formatResult(CalculationResult result);

}

## 3.8. TextResultFormatter.java

public class TextResultFormatter extends ResultFormatter {

@Override

public String formatResult(CalculationResult result) {

return "Type: " + result.getType() + ", Value: " + result.getValue();

}

}

## 3.9. HtmlResultFormatter.java

public class HtmlResultFormatter extends ResultFormatter {

@Override

public String formatResult(CalculationResult result) {

return "<html>" +

"<head><title>" + result.getType() + " Result</title></head>" +

"<body>" +

"<h1>" + result.getType() + "</h1>" +

"<p>Value: " + result.getValue() + "</p>" +

"</body>" +

"</html>";

}

}

## 3.10. TextTableFormatter.java

public class TextTableFormatter extends ResultFormatter {

private int columnWidth = 15; // Default column width

// Constructor for customization

public TextTableFormatter(int columnWidth) {

this.columnWidth = columnWidth;

}

@Override

public String formatResult(CalculationResult result) {

StringBuilder sb = new StringBuilder();

String headerFormat = "| %-" + columnWidth + "s | %-" + columnWidth + "s |\n";

String lineSeparator = String.format("+%s+%-s+\n", "-".repeat(columnWidth), "-".repeat(columnWidth));

sb.append(lineSeparator);

sb.append(String.format(headerFormat, "Type", "Value"));

sb.append(lineSeparator);

sb.append(String.format(headerFormat, result.getType(), result.getValue()));

sb.append(lineSeparator);

return sb.toString();

}

}

## 3.11. CalculatorFactory.java

public class CalculatorFactory {

public AbstractCalculator createCalculator(String type) {

return createCalculator(type);

}

public ResultFormatter createFormatter(String type) {

switch (type) {

case "Text":

return new TextResultFormatter();

case "Html":

return new HtmlResultFormatter();

case "Table":

return new TextTableFormatter(20);

default:

throw new IllegalArgumentException("Unknown formatter type: " + type);

}

}

}

## 3.12. Command.java

public abstract class Command {

public abstract void execute();

}

## 3.13. UndoableCommand.java

public abstract class UndoableCommand extends Command {

public abstract void undo();

}

## 3.14. CommandHistory.java

import java.util.ArrayList;

import java.util.List;

public class CommandHistory {

private List<Command> commands = new ArrayList<>();

public void add(Command command) {

commands.add(command);

}

public void undo() {

if (!commands.isEmpty()) {

Command command = commands.remove(commands.size() - 1);

if (command instanceof UndoableCommand) {

((UndoableCommand) command).undo();

}

}

}

public boolean canUndo() {

return !commands.isEmpty();

}

}

## 3.15. Calculator.java

public class Calculator {

private static final Calculator instance = new Calculator();

private CommandHistory history = new CommandHistory();

private Calculator() {

}

public static Calculator getInstance() {

return instance;

}

public void add(double a, double b) {

Command command = new AddCommand(a, b);

command.execute();

history.add(command);

}

public void subtract(double a, double b) {

Command command = new SubtractCommand(a, b);

command.execute();

history.add(command);

}

public void multiply(double a, double b) {

Command command = new MultiplyCommand(a, b);

command.execute();

history.add(command);

}

public void divide(double a, double b) {

Command command = new DivideCommand(a, b);

command.execute();

history.add(command);

}

public void undo() {

history.undo();

}

public boolean canUndo() {

return history.canUndo();

}

private static class AddCommand extends UndoableCommand {

private double a;

private double b;

public AddCommand(double a, double b) {

this.a = a;

this.b = b;

}

@Override

public void execute() {

double result = a + b;

System.out.println("Result: " + result);

}

@Override

public void undo() {

double result = a + b; // Reversing addition operation

System.out.println("Undo: " + result);

}

}

private static class SubtractCommand extends UndoableCommand {

private double a;

private double b;

public SubtractCommand(double a, double b) {

this.a = a;

this.b = b;

}

@Override

public void execute() {

double result = a - b;

System.out.println("Result: " + result);

}

@Override

public void undo() {

double result = a - b; // Reversing subtraction operation

System.out.println("Undo: " + result);

}

}

private static class MultiplyCommand extends UndoableCommand {

private double a;

private double b;

public MultiplyCommand(double a, double b) {

this.a = a;

this.b = b;

}

@Override

public void execute() {

double result = a \* b;

System.out.println("Result: " + result);

}

@Override

public void undo() {

double result = a \* b; // Reversing multiplication operation

System.out.println("Undo: " + result);

}

}

private static class DivideCommand extends UndoableCommand {

private double a;

private double b;

public DivideCommand(double a, double b) {

this.a = a;

this.b = b;

}

@Override

public void execute() {

if (b != 0) {

double result = a / b;

System.out.println("Result: " + result);

} else {

System.out.println("Cannot divide by zero.");

}

}

@Override

public void undo() {

if (b != 0) {

double result = a / b; // Reversing division operation

System.out.println("Undo: " + result);

} else {

System.out.println("Cannot divide by zero.");

}

}

}

}

## 3.16. ConsoleUI.java

import java.util.InputMismatchException;

import java.util.Scanner;

public class ConsoleUI {

private Calculator calculator = Calculator.getInstance();

public void run() {

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("Enter command [add/subtract/multiply/divide/undo/exit]:");

String command = scanner.nextLine();

try {

switch (command) {

case "add":

System.out.println("Enter first number:");

double a = scanner.nextDouble();

System.out.println("Enter second number:");

double b = scanner.nextDouble();

calculator.add(a, b);

break;

case "subtract":

System.out.println("Enter first number:");

a = scanner.nextDouble();

System.out.println("Enter second number:");

b = scanner.nextDouble();

calculator.subtract(a, b);

break;

case "multiply":

System.out.println("Enter first number:");

a = scanner.nextDouble();

System.out.println("Enter second number:");

b = scanner.nextDouble();

calculator.multiply(a, b);

break;

case "divide":

System.out.println("Enter first number:");

a = scanner.nextDouble();

System.out.println("Enter second number:");

b = scanner.nextDouble();

calculator.divide(a, b);

break;

case "undo":

calculator.undo();

break;

case "exit":

System.exit(0);

break;

default:

System.out.println("Unknown command.");

}

} catch (InputMismatchException e) {

System.out.println("Error: Expected numerical value.");

scanner.next(); // Clear input buffer

} catch (Exception e) {

System.out.println("Error: " + e.getMessage());

}

}

}

}

## 3.17. CalculatorTest.java

public class CalculatorTest {

public static void test() {

// Add tests here

}

}

## 3.18. ParallelProcessing.java

import java.util.Comparator;

import java.util.List;

import java.util.NoSuchElementException;

import java.util.concurrent.\*;

public class ParallelProcessing {

private ExecutorService executorService = Executors.newFixedThreadPool(4); // Thread pool

public Future<Double> calculateMeanAsync(List<Double> values) {

Callable<Double> task = () -> {

double sum = 0;

for (double value : values) {

sum += value;

}

return sum / values.size();

};

return executorService.submit(task);

}

public Future<Double> findMinAsync(List<Double> values) {

Callable<Double> task = () -> values.stream().min(Comparator.naturalOrder()).orElseThrow(NoSuchElementException::new);

return executorService.submit(task);

}

public Future<Double> findMaxAsync(List<Double> values) {

Callable<Double> task = () -> values.stream().max(Comparator.naturalOrder()).orElseThrow(NoSuchElementException::new);

return executorService.submit(task);

}

public void shutdownExecutorService() {

executorService.shutdown();

try {

executorService.awaitTermination(5, TimeUnit.SECONDS);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

## 3.19. TaskQueue.java

import java.util.concurrent.BlockingQueue;

import java.util.concurrent.LinkedBlockingQueue;

public class TaskQueue {

private static final TaskQueue instance = new TaskQueue();

private BlockingQueue<String> tasks = new LinkedBlockingQueue<>();

private WorkerThread workerThread;

private TaskQueue() {

workerThread = new WorkerThread();

workerThread.start();

}

public static TaskQueue getInstance() {

return instance;

}

public void addTask(String task) {

tasks.add(task);

}

private class WorkerThread extends Thread {

@Override

public void run() {

while (true) {

try {

String task = tasks.take(); // Blocking wait

processTask(task);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

private void processTask(String task) {

// Implement task processing logic

}

}

}

## 3.20. Main.java

// Main.java

public class Main {

public static void main(String[] args) {

CalculatorApp.main(args);

}

}

# 4. РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ

При відкритті програми, перше що ми бачимо це головне меню (рис. 4.1), де ми можемо обрати потрібну нам дію.



Рисунок 4.1 – Головне меню

У випадку, якщо команда не вірна, користувача повертає до головного меню (рис. 4.2).

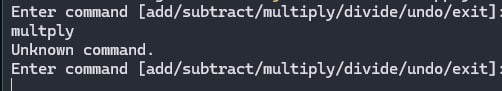


Рисунок 4.2 – Повернення до меню

Візьмемо до прикладу дію ділення, на (рис. 4.3), бачимо запит на введення першого та другого числа, при коректному вводі користувач отримує результат

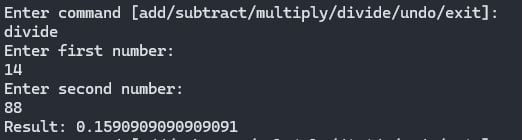


Рисунок 4.3 – Ділення

В випадку якщо, користувач вводить команду undo, програма повертає останній вивід (рис. 4.4).

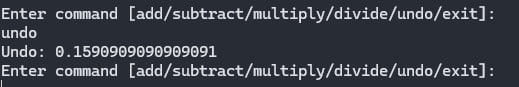


Рисунок 4.4 – Повернення останнього результату

# ВИСНОВКИ

Навички програмування з використанням ООП: Під час практики я вдосконалив навички програмування з використанням об’єктно-орієнтованого підходу (ООП) на мові Java. Це включало в себе роботу з класами, об’єктами, спадкуванням, поліморфізмом та іншими ключовими аспектами ООП.

Створення об’єктів та взаємодія між ними: Я навчився створювати класи, ініціалізувати об’єкти та взаємодіяти з ними за допомогою методів та полів.

Принципи ООП: Я оволодів принципами ООП, такими як інкапсуляція, абстракція, спадкування та поліморфізм. Це дозволило мені створювати більш структурований та гнучкий код.

Робота з наслідуванням та інтерфейсами: Я вивчив роботу з наслідуванням, реалізацією інтерфейсів та використанням абстрактних класів.

Створення програм з ООП: В результаті моєї практики були створені програми, які використовували принципи ООП. Це було досягнуто завдяки використанню набутих знань та навичок.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Руководство по языку программирования Java. [Електронний ресурс] – Режим доступу до ресурсу: <https://metanit.com/java/tutorial/> (дата звернення 03.03.2021). – Назва з екрану.
2. JavaRush. Помощь по задачам. [Електронний ресурс] – Режим доступу до ресурсу: <https://javarush.ru/help> (дата звернення 05.03.2021). – Назва з екрану.
3. Stack Overflow. [Електронний ресурс] – Режим доступу до ресурсу: <https://stackoverflow.com> (дата звернення 05.03.2021). – Назва з екрану.
4. GeeksforGeeks. Java Programming Language. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/java/> (дата звернення 05.03.2021). – Назва з екрану.
5. 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.

**ДОДАТОК А. ПОСИЛАННЯ НА GITHUB РЕПОЗИТОРІЙ**

Рисунок А.1 – Посилання на репозиторій