Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра автоматизованих систем управління



**Звіт**

до лабораторної роботи № 6

з дисципліни

Прикладне програмування​

на тему:

**“Комплексна лабораторна робота”**

Виконав: студент ОІ-24

**Тріщук Максим**

Прийняв: Пелех Ю.М.

Львів – 2024

***Лабораторна робота № 6***

**Завдання лабораторної роботи**

Створити консольну програму на мові програмування Java згідно до індивідуального завдання (ваш номер варіанту, згенерований випадковим чином, можна знайти у файлі «Варіанти завдань.pdf»). Саме завдання описане досить поверхово, якщо ви претендуєте на високу оцінку – його потрібно розширити. Наприклад, у завданні може бути сказано про «створити кілька квіток, з наявних квіток сформувати букет». Цілком логічно буде додати можливість видалити квітку, поміняти параметри існуючої квітки тощо. Оцінка за лабораторну роботу залежатиме і від глибини пропрацювання предметної області.

Окрім цього, програма повинна відповідати таким додатковим вимогам:

• Використовувати можливості ООП: класи, успадкування, поліморфізм, інкапсуляцію.

• Кожний клас повинен мати назву, яка повністю описує його суть, і інформативний склад.

• Успадкування потрібно використовувати тільки тоді, коли воно має сенс.

• При записі програми потрібно використовувати домовленості щодо оформлення коду java code convention.

• Класи потрібно грамотно розкласти по пакетах.

• Для збереження параметрів ініціалізації потрібно використовувати файли.

• Повинне бути консольне меню для вибору різних дій, наприклад: «завантажити з файлу», «довідка», «вихід» тощо.

Етапи виконання завдання (кожен етап вважається окремою лабораторною роботою):

1. Уважно пропрацюйте предметну область згідно вашого варіанту завдання. Опишіть технологічне завдання (який функціонал матиме ваша програма). Створіть Use Case та Class діаграми для вашого завдання.

2. Реалізуйте консольне меню до вашого завдання з використанням паттерну «Команда». На цьому етапі не потрібно реалізовувати функціонал конкретних пунктів меню – тільки створити «кістяк» вашої програми.

3. Реалізуйте функціонал програми. Обов’язково зверніть увагу на додаткові вимоги до програми (описані вище).

4. Покрийте вашу програму юніт тестами (coverage >=90%).

5. Додайте логгер з логуванням основних дій та виняткових ситуацій. При цьому далі логу мають зберігатися у файлі, а критичні помилки – додатково надсилатися на e-mail

**Варіант 15**

**Податки.** Визначити множину і суму податкових виплат фізичної особи за рік із врахуванням доходів з основного і додаткового місць роботи, авторських винагород, продажу майна, отримання в подарунок грошових сум і майна, переказів з-за кордону, пільг на дітей і матеріальну допомогу. Здійснити сортування податків по сумі. Знайти для особи податки, що відповідають заданому діапазону параметрів.

package com.taxapp;

import java.util.\*;

interface Command {

void execute();

}

class Tax {

private String type;

private double income;

private double rate;

public Tax(String type, double income, double rate) {

this.type = type;

this.income = income;

this.rate = rate;

}

public double calculateTax() {

return income \* rate / 100;

}

public String getType() {

return type;

}

public double getIncome() {

return income;

}

public double getRate() {

return rate;

}

@Override

public String toString() {

return String.format("Type: %s, Income: %.2f, Rate: %.2f%%, Tax: %.2f",

type, income, rate, calculateTax());

}

}

class Menu {

private Map<Integer, Command> commands = new HashMap<>();

private boolean isRunning = true;

public void addCommand(int option, Command command) {

commands.put(option, command);

}

public void displayMenu() {

System.out.println("\n=== Tax Management Menu ===");

System.out.println("1. Add a new tax");

System.out.println("2. Display all taxes");

System.out.println("3. Calculate total tax");

System.out.println("4. Sort taxes by amount");

System.out.println("5. Find taxes in a range");

System.out.println("6. Exit");

System.out.print("Choose an option: ");

}

public void executeCommand(int option) {

Command command = commands.get(option);

if (command != null) {

command.execute();

} else {

System.out.println("Invalid option. Please try again.");

}

}

public void stop() {

isRunning = false;

}

public boolean isRunning() {

return isRunning;

}

}

class AddTaxCommand implements Command {

private List<Tax> taxes;

private Scanner scanner;

public AddTaxCommand(List<Tax> taxes, Scanner scanner) {

this.taxes = taxes;

this.scanner = scanner;

}

@Override

public void execute() {

System.out.print("Enter tax type: ");

String type = scanner.nextLine();

System.out.print("Enter income: ");

double income = Double.parseDouble(scanner.nextLine());

System.out.print("Enter tax rate (%): ");

double rate = Double.parseDouble(scanner.nextLine());

taxes.add(new Tax(type, income, rate));

System.out.println("Tax added successfully!");

}

}

class DisplayTaxesCommand implements Command {

private List<Tax> taxes;

public DisplayTaxesCommand(List<Tax> taxes) {

this.taxes = taxes;

}

@Override

public void execute() {

if (taxes.isEmpty()) {

System.out.println("No taxes available.");

return;

}

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

for (Tax tax : taxes) {

System.out.println(tax);

}

}

}

class CalculateTotalTaxCommand implements Command {

private List<Tax> taxes;

public CalculateTotalTaxCommand(List<Tax> taxes) {

this.taxes = taxes;

}

@Override

public void execute() {

double totalTax = taxes.stream().mapToDouble(Tax::calculateTax).sum();

System.out.printf("Total tax amount: %.2f\n", totalTax);

}

}

class SortTaxesCommand implements Command {

private List<Tax> taxes;

public SortTaxesCommand(List<Tax> taxes) {

this.taxes = taxes;

}

@Override

public void execute() {

taxes.sort(Comparator.comparingDouble(Tax::calculateTax));

System.out.println("Taxes sorted by amount!");

}

}

class FindTaxesInRangeCommand implements Command {

private List<Tax> taxes;

private Scanner scanner;

public FindTaxesInRangeCommand(List<Tax> taxes, Scanner scanner) {

this.taxes = taxes;

this.scanner = scanner;

}

@Override

public void execute() {

System.out.print("Enter minimum tax amount: ");

double min = Double.parseDouble(scanner.nextLine());

System.out.print("Enter maximum tax amount: ");

double max = Double.parseDouble(scanner.nextLine());

System.out.println("=== Taxes in Range ===");

taxes.stream()

.filter(tax -> tax.calculateTax() >= min && tax.calculateTax() <= max)

.forEach(System.out::println);

}

}

class ExitCommand implements Command {

private Menu menu;

public ExitCommand(Menu menu) {

this.menu = menu;

}

@Override

public void execute() {

System.out.println("Exiting the program. Goodbye!");

menu.stop();

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

List<Tax> taxes = new ArrayList<>();

Menu menu = new Menu();

menu.addCommand(1, new AddTaxCommand(taxes, scanner));

menu.addCommand(2, new DisplayTaxesCommand(taxes));

menu.addCommand(3, new CalculateTotalTaxCommand(taxes));

menu.addCommand(4, new SortTaxesCommand(taxes));

menu.addCommand(5, new FindTaxesInRangeCommand(taxes, scanner));

menu.addCommand(6, new ExitCommand(menu));

while (menu.isRunning()) {

menu.displayMenu();

int option;

try {

option = Integer.parseInt(scanner.nextLine());

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter a number.");

continue;

}

menu.executeCommand(option);

}

scanner.close();

}

}

**Висновок**

На цій лабораторній роботі я закріпив свої знання про ООП: класи, успадкування, поліморфізм, інкапсуляцію.

https://github.com/itimodzi/Trishchuk\_oi-24\_2024.git