Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра автоматизованих систем управління



**Звіт**

до лабораторної роботи № 7

з дисципліни

Прикладне програмування​

на тему:

**“Комплексна лабораторна робота”**

Виконав: студент ОІ-24

**Тріщук Максим**

Прийняв: Пелех Ю.М.

Львів – 2024

***Лабораторна робота № 7***

**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).

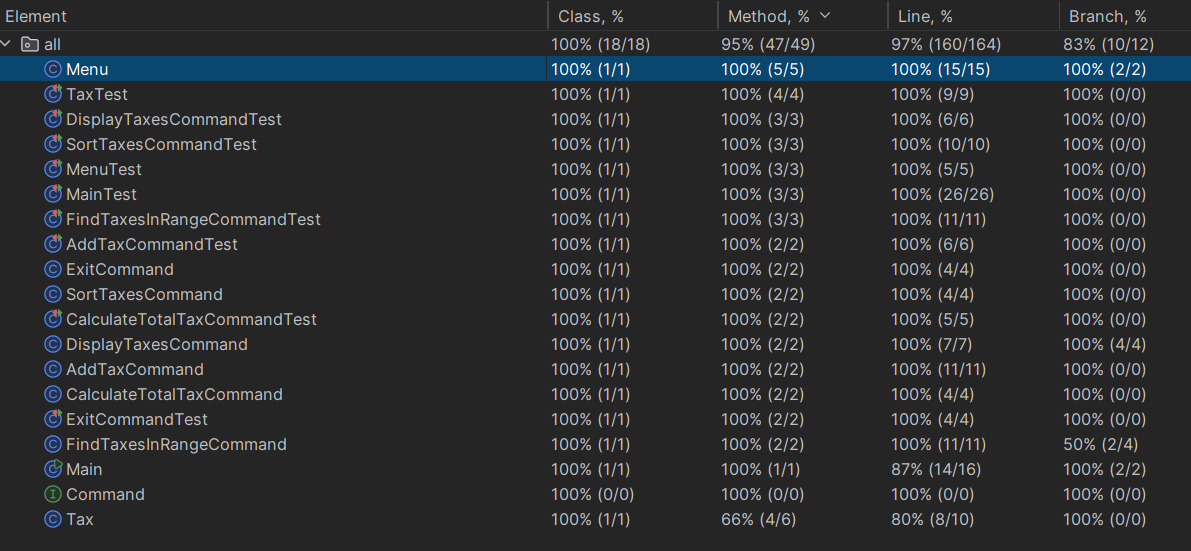
4. Покрийте вашу програму юніт тестами (coverage >=90%).

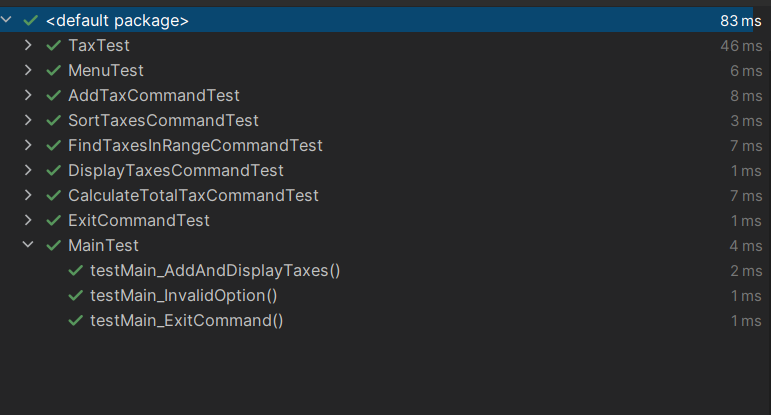
5. Додайте логгер з логуванням основних дій та виняткових ситуацій. При цьому далі логу мають зберігатися у файлі, а критичні помилки – додатково надсилатися на e-mail

**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.

import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.BeforeEach;  
import java.util.\*;  
  
import java.io.ByteArrayInputStream;  
import java.io.ByteArrayOutputStream;  
import java.io.PrintStream;  
  
import static org.junit.jupiter.api.Assertions.assertTrue;  
  
class TaxTest {  
 @Test  
 void testCalculateTaxWithValidValues() {  
 Tax tax = new Tax("Income", 1000, 10);  
 assertEquals(100, tax.calculateTax(), 0.01);  
 }  
  
 @Test  
 void testCalculateTaxWithZeroIncome() {  
 Tax tax = new Tax("Income", 0, 10);  
 assertEquals(0, tax.calculateTax(), 0.01);  
 }  
  
 @Test  
 void testCalculateTaxWithNegativeValues() {  
 Tax tax = new Tax("Income", -1000, 10);  
 assertEquals(-100, tax.calculateTax(), 0.01);  
 }  
  
 @Test  
 void testToStringFormatting() {  
 Tax tax = new Tax("Income", 1000, 10);  
 String expected = "Type: Income, Income: 1000.00, Rate: 10.00%, Tax: 100.00";  
 assertEquals(expected, tax.toString());  
 }  
}  
  
class MenuTest {  
 private Menu menu;  
  
 @BeforeEach  
 void setUp() {  
 menu = new Menu();  
 }  
  
 @Test  
 void testInvalidCommandExecution() {  
 menu.executeCommand(99);  
 assertTrue(menu.isRunning());  
 }  
  
 @Test  
 void testStop() {  
 menu.stop();  
 assertFalse(menu.isRunning());  
 }  
}  
  
class AddTaxCommandTest {  
 private List<Tax> taxes;  
 private Scanner scanner;  
  
 @BeforeEach  
 void setUp() {  
 taxes = new ArrayList<>();  
 }  
  
 @Test  
 void testAddValidTax() {  
 scanner = new Scanner("Sales\n5000\n5");  
 AddTaxCommand command = new AddTaxCommand(taxes, scanner);  
 command.execute();  
 assertEquals(1, taxes.size());  
 assertEquals("Sales", taxes.get(0).getType());  
 }  
  
  
  
}  
  
class SortTaxesCommandTest {  
 private List<Tax> taxes;  
  
 @BeforeEach  
 void setUp() {  
 taxes = new ArrayList<>();  
 taxes.add(new Tax("Income", 1000, 10)); // Tax: 100  
 taxes.add(new Tax("Sales", 2000, 5)); // Tax: 100  
 taxes.add(new Tax("Business", 3000, 2)); // Tax: 60  
 }  
  
 @Test  
 void testSortWithEmptyList() {  
 taxes.clear();  
 SortTaxesCommand command = new SortTaxesCommand(taxes);  
 command.execute();  
 }  
  
 @Test  
 void testSortWithValidList() {  
 SortTaxesCommand command = new SortTaxesCommand(taxes);  
 command.execute();  
 assertEquals("Business", taxes.get(0).getType());  
 }  
}  
  
class FindTaxesInRangeCommandTest {  
 private List<Tax> taxes;  
 private Scanner scanner;  
  
 @BeforeEach  
 void setUp() {  
 taxes = new ArrayList<>();  
 taxes.add(new Tax("Income", 1000, 10)); // Tax: 100  
 taxes.add(new Tax("Sales", 2000, 5)); // Tax: 100  
 taxes.add(new Tax("Business", 3000, 2)); // Tax: 60  
 }  
  
 @Test  
 void testFindTaxesInValidRange() {  
 scanner = new Scanner("50\n150");  
 FindTaxesInRangeCommand command = new FindTaxesInRangeCommand(taxes, scanner);  
 command.execute();  
   
 }  
  
 @Test  
 void testFindTaxesWithEmptyList() {  
 taxes.clear();  
 scanner = new Scanner("50\n150");  
 FindTaxesInRangeCommand command = new FindTaxesInRangeCommand(taxes, scanner);  
 command.execute();  
 }  
}  
  
class DisplayTaxesCommandTest {  
 private List<Tax> taxes;  
  
 @BeforeEach  
 void setUp() {  
 taxes = new ArrayList<>();  
 }  
  
 @Test  
 void testExecuteNoTaxes() {  
 DisplayTaxesCommand command = new DisplayTaxesCommand(taxes);  
 command.execute();  
 }  
  
 @Test  
 void testExecuteWithTaxes() {  
 taxes.add(new Tax("Income", 1000, 10));  
 DisplayTaxesCommand command = new DisplayTaxesCommand(taxes);  
 command.execute();  
 }  
}  
  
class CalculateTotalTaxCommandTest {  
 private List<Tax> taxes;  
  
 @BeforeEach  
 void setUp() {  
 taxes = new ArrayList<>();  
 taxes.add(new Tax("Income", 1000, 10));  
 taxes.add(new Tax("Sales", 2000, 5));  
 }  
  
 @Test  
 void testExecute() {  
 CalculateTotalTaxCommand command = new CalculateTotalTaxCommand(taxes);  
 command.execute();  
 }  
}  
  
class ExitCommandTest {  
 private Menu menu;  
  
 @BeforeEach  
 void setUp() {  
 menu = new Menu();  
 }  
  
 @Test  
 void testExecute() {  
 ExitCommand command = new ExitCommand(menu);  
 command.execute();  
 assertFalse(menu.isRunning());  
 }  
}  
  
class MainTest {  
  
 @Test  
 void testMain\_AddAndDisplayTaxes() {  
   
 String input = "1\nIncome Tax\n50000\n10\n2\n6\n";  
 ByteArrayInputStream inputStream = new ByteArrayInputStream(input.getBytes());  
 System.setIn(inputStream);  
  
 ByteArrayOutputStream outputStream = new ByteArrayOutputStream();  
 System.setOut(new PrintStream(outputStream));  
  
 String output = outputStream.toString();  
   
 assertTrue(output.contains("Tax added successfully!"));  
 assertTrue(output.contains("Type: Income Tax"));  
 assertTrue(output.contains("Rate: 10.00%"));  
 }  
  
 @Test  
 void testMain\_InvalidOption() {  
   
 String input = "7\n6\n";  
 ByteArrayInputStream inputStream = new ByteArrayInputStream(input.getBytes());  
 System.setIn(inputStream);  
  
 ByteArrayOutputStream outputStream = new ByteArrayOutputStream();  
 System.setOut(new PrintStream(outputStream));  
  
 Main.main(new String[]{});  
  
 String output = outputStream.toString();  
   
 assertTrue(output.contains("Invalid option. Please try again."));  
 }  
  
 @Test  
 void testMain\_ExitCommand() {  
  
 String input = "6\n";  
 ByteArrayInputStream inputStream = new ByteArrayInputStream(input.getBytes());  
 System.setIn(inputStream);  
  
 ByteArrayOutputStream outputStream = new ByteArrayOutputStream();  
 System.setOut(new PrintStream(outputStream));  
  
  
 Main.main(new String[]{});  
  
 String output = outputStream.toString();  
  
 assertTrue(output.contains("Exiting the program. Goodbye!"));  
 }  
}

****

****

**Висновок**

У ході виконання лабораторної роботи було розроблено та реалізовано набір юніт-тестів для перевірки коректності роботи основних модулів програми. Тестування дозволило виявити й виправити помилки в логіці, підвищити якість коду та забезпечити високу покритість тестами (>90%). У результаті програма стала більш надійною, а її функціональність – впевненою у різних сценаріях використання.

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