­­­­­­­­Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра автоматизованих систем управління



**Звіт**

до лабораторної роботи № 7-8

з дисципліни

*Прикладне програмування*

на тему:

**“ Комплексна лабораторна робота ”**

Виконав: студент ОІ-24

**Живоронко Роман**

Прийняв: асистент каф. АСУ

Чорненький В.Я.

**Львів – 2024**

*Лабораторна робота №* *7-8*

**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 8**

Транспорт. Визначити ієрархію рухомого складу залізничного транспорту. Створити пасажирський потяг. Підрахувати загальну кількість пасажирів та багажу. Здійснити сортування вагонів поїзда на основі рівня комфортності. Знайти вагони в поїзді, що відповідають заданому діапазону параметрів кількості пасажирів.

**Код програми**

package MainTest;  
  
import Main.Main;  
import Menu.Menu;  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
class MainTest {  
 @Test  
 void testMainExecution() {  
 Menu mockMenu = *mock*(Menu.class);  
 Main.*main*(new String[]{});  
 *verify*(mockMenu, *times*(1)).menu();  
 }  
}

package MenuTest;  
  
import Train.Train;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import Menu.\*;  
  
import java.util.ArrayList;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class MenuTest {  
 private Menu menu;  
  
 @BeforeEach  
 public void setUp(){  
 menu = new Menu();  
 }  
  
 @Test  
 public void testCreateTrain() {  
 menu.menu();  
 ArrayList<Train> trains = menu.createTrain();  
 *assertEquals*(6,trains.size());  
 }  
  
 @Test  
 public void testInvalidOptionHandling() {  
 *assertEquals*(5,menu.getCommands().size());  
 }  
}

package TrainTest;  
  
import Train.\*;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import Coaches.\*;  
  
import static org.junit.jupiter.api.Assertions.\*;  
 class TrainTest {  
 private Train train;  
  
 @BeforeEach  
 void setUp() {  
 train = new Train("Test Train");  
 // Adding different types of coaches with predefined values  
 Coach businessCoach = new BussinesCoach();  
 Coach economyCoach = new EcoCoach();  
 train.addCoach(businessCoach);  
 train.addCoach(economyCoach);  
 }  
  
 @Test  
 void testAddCoach() {  
 *assertEquals*(2, train.getCoaches().size(), "The train should have 2 coaches.");  
 }  
  
 @Test  
 void testNumOfPassengers() {  
 // Assuming BussinesCoach has 20 seats and EconomyCoach has 30 seats.  
 int expectedPassengers = train.getCoaches().get(0).getNumberOfSeats() +  
 train.getCoaches().get(1).getNumberOfSeats();  
 *assertEquals*(expectedPassengers, train.numofpass(), "The number of passengers should match expected count.");  
 }  
  
 @Test  
 void testNumOfBaggage() {  
 int expectedBags = (train.getCoaches().get(0).getNumOfBagsPerPerson() \* train.getCoaches().get(0).getNumberOfSeats()) +  
 (train.getCoaches().get(1).getNumOfBagsPerPerson() \* train.getCoaches().get(1).getNumberOfSeats());  
 *assertEquals*(expectedBags, train.numofbagage(), "The number of baggage should match expected count.");  
 }  
  
 @Test  
 void testSortCoaches() {  
 train.sortCoaches();  
 *assertTrue*(train.getCoaches().get(0).getComfLevel() <= train.getCoaches().get(1).getComfLevel(),  
 "Coaches should be sorted by comfort level in ascending order.");  
 }  
  
 @Test  
 void testFindCoach() {  
 // Redirecting System.out for testing output  
 var outputStreamCaptor = new java.io.ByteArrayOutputStream();  
 System.*setOut*(new java.io.PrintStream(outputStreamCaptor));  
  
 // Example input: we look for a coach with at least 20 seats  
 train.findCoach(20);  
  
 // Check output contains details of the coach that meets the condition  
 *assertTrue*(outputStreamCaptor.toString().contains("BussinesCoach") || outputStreamCaptor.toString().contains("EconomyCoach"),  
 "Expected output should contain details of the found coach.");  
 }  
  
 @Test  
 void testToString() {  
 String expectedString = "\nTest Train, Вагони:[BussinesCoach, EconomyCoach]";  
 *assertTrue*(train.toString().contains("Test Train") && train.toString().contains("Вагони"),  
 "toString should include the train name and list of coaches.");  
 }  
 }

package commandTest;  
  
import Coaches.BussinesCoach;  
import Menu.\*;  
import Train.Train;  
  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import java.io.ByteArrayInputStream;  
import java.io.ByteArrayOutputStream;  
import java.io.PrintStream;  
import java.util.ArrayList;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class CommandTest {  
 private ArrayList<Train> trains;  
 private final ByteArrayOutputStream outputStreamCaptor = new ByteArrayOutputStream();  
  
 @BeforeEach  
 void setUp() {  
 trains = new ArrayList<>();  
 Train testTrain = new Train("Тестовий потяг");  
 testTrain.addCoach(new BussinesCoach());  
 trains.add(testTrain);  
 System.*setOut*(new PrintStream(outputStreamCaptor));  
 }  
  
 @Test  
 public void testCountNumOfBaggageCommand() {  
 // Імітація введення "1" для вибору першого потягу  
 System.*setIn*(new ByteArrayInputStream("1\n".getBytes()));  
  
 CountNumOfBaggageCommand command = new CountNumOfBaggageCommand(trains, new java.util.Scanner(System.*in*));  
 command.execute();  
  
 // Очікуване значення багажу для першого потягу  
 *assertTrue*(outputStreamCaptor.toString().contains("200"));  
 }  
  
 @Test  
 public void testCountNumOfPassengersCommand() {  
 System.*setIn*(new ByteArrayInputStream("1\n".getBytes()));  
  
 CountNumOfPassengersCommand command = new CountNumOfPassengersCommand(trains, new java.util.Scanner(System.*in*));  
 command.execute();  
  
 // Очікуване значення кількості пасажирів для першого потягу  
 *assertTrue*(outputStreamCaptor.toString().contains("20"));  
 }  
  
 @Test  
 public void testFindCoachByPassCommand() {  
 System.*setIn*(new ByteArrayInputStream("99\n".getBytes()));  
  
 FindCoachByPassCommand command = new FindCoachByPassCommand(trains, new java.util.Scanner(System.*in*));  
 command.execute();  
  
 // Перевірка, що знайдений потяг відображено у виводі  
 *assertTrue*(outputStreamCaptor.toString().contains("Тестовий потяг"));  
 }  
  
 @Test  
 public void testSortCoachByComfortCommand() {  
 System.*setIn*(new ByteArrayInputStream("1\n".getBytes()));  
  
 SortCoachByComfortCommand command = new SortCoachByComfortCommand(trains, new java.util.Scanner(System.*in*));  
 command.execute();  
  
 // Перевірка, що сортування відбулося успішно за рівнем комфорту  
 *assertEquals*(3, trains.get(0).getCoaches().get(0).getComfLevel());  
 }  
  
 @Test  
 public void testInvalidTrainSelection() {  
 System.*setIn*(new ByteArrayInputStream("99\n".getBytes()));  
  
 CountNumOfPassengersCommand command = new CountNumOfPassengersCommand(trains, new java.util.Scanner(System.*in*));  
 command.execute();  
  
 // Перевірка на відсутність результатів, оскільки вибрано неіснуючий потяг  
 *assertTrue*(outputStreamCaptor.toString().contains("Не знайдено"));  
 }  
}

package CoachTest;  
  
import Coaches.\*;  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class CoachTest {  
 @Test  
 public void testBussinesCoachProperties() {  
 BussinesCoach bussinesCoach = new BussinesCoach();  
 *assertEquals*("Бізнес клас", bussinesCoach.getName());  
 *assertEquals*(20, bussinesCoach.getNumberOfSeats());  
 *assertEquals*(10, bussinesCoach.getNumOfBagsPerPerson());  
 *assertEquals*(2000.0, bussinesCoach.getPrice());  
 *assertEquals*(3, bussinesCoach.getComfLevel());  
 }  
  
 @Test  
 public void testBussinesToString() {  
 BussinesCoach bussinesCoach = new BussinesCoach();  
 String expected = "Назва вагона - Бізнес клас, К-сть місць - 20, К-сть сумок на особу - 10, Ціна - 2000.0грн Рівень комфортності 3;";  
 *assertEquals*(expected, bussinesCoach.toString());  
 }  
  
 @Test  
 public void testEcoCoachProperties() {  
 EcoCoach ecoCoach = new EcoCoach();  
 *assertEquals*("Економ", ecoCoach.getName());  
 *assertEquals*(60, ecoCoach.getNumberOfSeats());  
 *assertEquals*(2, ecoCoach.getNumOfBagsPerPerson());  
 *assertEquals*(500.0, ecoCoach.getPrice());  
 *assertEquals*(1, ecoCoach.getComfLevel());  
 }  
  
 @Test  
 public void testEcoToString() {  
 EcoCoach ecoCoach = new EcoCoach();  
 String expected = "Назва вагона - Економ клас, К-сть місць - 60, К-сть сумок на особу - 2, Ціна - 500.0грн Рівень комфортності 1";  
 *assertEquals*(expected, ecoCoach.toString());  
 }  
  
 @Test  
 public void testMidleCoachProperties() {  
 MidleCoach midleCoach = new MidleCoach();  
 *assertEquals*("Середній клас", midleCoach.getName());  
 *assertEquals*(45, midleCoach.getNumberOfSeats());  
 *assertEquals*(4, midleCoach.getNumOfBagsPerPerson());  
 *assertEquals*(750.0, midleCoach.getPrice());  
 *assertEquals*(2, midleCoach.getComfLevel());  
 }  
  
 @Test  
 public void testMidleToString() {  
 MidleCoach midleCoach = new MidleCoach();  
 String expected = "Назва вагона - Середній клас, К-сть місць - 45, К-сть сумок на особу - 4, Ціна - 750.0грн Рівень комфортності 2";  
 *assertEquals*(expected, midleCoach.toString());  
 }  
  
 @Test  
 public void testDifferentComfortLevels() {  
 BussinesCoach bussinesCoach = new BussinesCoach();  
 EcoCoach ecoCoach = new EcoCoach();  
 MidleCoach midleCoach = new MidleCoach();  
  
 *assertTrue*(bussinesCoach.getComfLevel() > ecoCoach.getComfLevel());  
 *assertTrue*(midleCoach.getComfLevel() < bussinesCoach.getComfLevel());  
 }  
}

**Лабораторна робота 8**

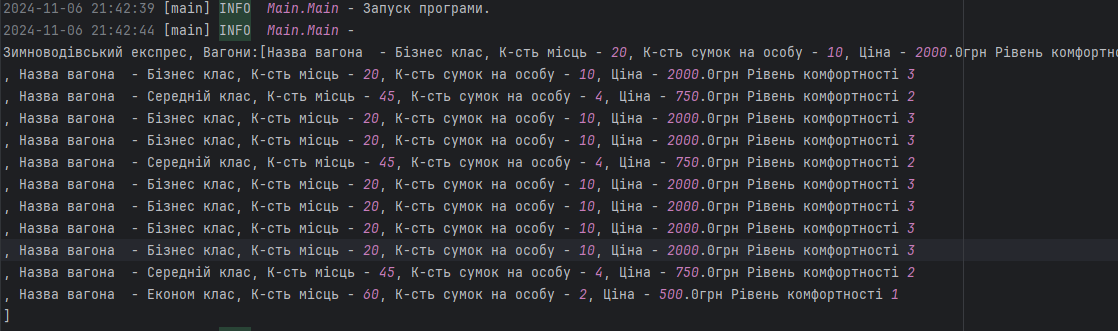
Додайте логгер з логуванням основних дій та виняткових ситуацій. При цьому далі логу мають зберігатися у файлі, а критичні помилки – додатково надсилатися на email.

**Файл конфігурації**<?xml version="1.0" encoding="UTF-8"?>  
<Configuration status="WARN">  
<Appenders>  
 <Console name="ConsoleAppender" target="SYSTEM\_OUT">  
 <PatternLayout pattern="%msg%n"/>  
 </Console>  
  
 <File name="FileAppender" fileName="logs/app.log" append="true">  
 <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} [%t] %-5level %logger{36} - %msg%n"/>  
 </File>  
  
 <SMTP name="EmailAppender" subject="Critical Error in Application"  
 to="xxx@gmail.com" from="xxx@gmail.com"  
 smtpHost="smtp.gmail.com" smtpPort="465"  
 smtpUsername="xxx@gmail.com" smtpPassword="PASWORD"  
 smtpProtocol="smtps" smtpDebug="false">  
 <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} [%t] %-5level %logger{36} - %msg%n"/>  
 <Filters>  
 <ThresholdFilter level="ERROR" onMatch="ACCEPT" onMismatch="DENY"/>  
 </Filters>  
 </SMTP>  
</Appenders>  
  
<Loggers>  
 <Root level="info">  
 <AppenderRef ref="ConsoleAppender"/>  
 <AppenderRef ref="FileAppender"/>  
 <AppenderRef ref="EmailAppender"/>  
 </Root>  
</Loggers>  
</Configuration>

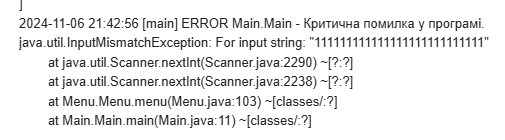
**Використання логера**

package Main;  
import Menu.\*;  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
public class Main {  
 public static final Logger *logger* = LogManager.*getLogger*(Main.class);  
 public static void main(String[] args) {  
 *logger*.info("Запуск програми.");  
 try {  
 Menu menu = new Menu();  
 menu.menu();  
 } catch (Exception e) {  
 *logger*.error("Критична помилка у програмі.", e);  
 }  
  
 *logger*.info("Програма завершена.");  
 }  
}

**Приклад нормальної роботи програми**

****

**Приклад критичної помилки**

****

**Git репозиторій:** https://github.com/zhyvoronkoRoman/LPNU\_PP\_Lab\_7-8

**Висновок**

На цій лабораторній роботі я закріпив можливості ООП: класи, успадкування, поліморфізм, інкапсуляцію