Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Звіт

з лабораторної роботи № 4 з дисципліни
«Мова програмування Java»

«Generics»

Варіант 1

Виконала ЗПІ-зп41 Федоренко О. Л.

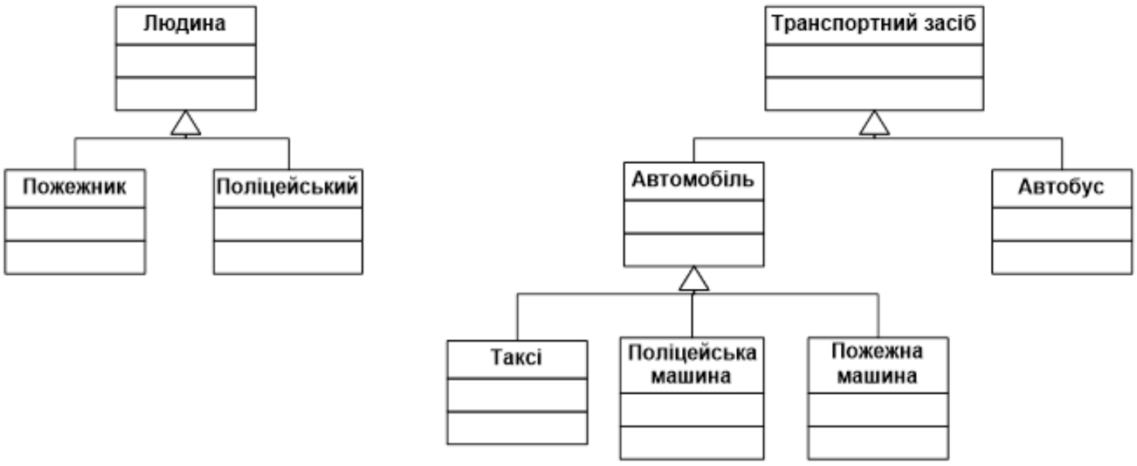
Перевірила Орленко С. П.

Київ 2025

1. Завдання (1)

Реалізувати із застосуванням узагальненого програмування ієрархію Java-класів для транспортних засобів, які можуть перевозити різні типи пасажирів.

Є наступні транспортні засоби: автобус, таксі, пожежна машина, поліцейська машина. У цих транспортних засобах можуть їздити наступні види пасажирів: звичайний пасажир, пожежник, поліцейський. Ієрархія вказаних об’єктів подана на рис.1. Додайте в меню команду для збереження пасажирів в файл та передбачте можливість читання набору пасажирів з файлу (використовуючи ObjectOutputStream та ObjectInputStream).



Автобус та таксі можуть перевозити будь-яких пасажирів, пожежна машина – тільки пожежників, міліцейська машина – тільки міліціонерів. Реалізувати на основі узагальненого програмування (generics) вказані обмеження щодо перевозки пасажирів.

Для класів транспортних засобів реалізувати наступні функції:

- Кожний транспортний засіб має обмежену кількість місць. Реалізувати функцію для отримання максимальної кількість місць та функцію для отримання кількості зайнятих місць.

- Посадка пасажира у транспортний засіб. Якщо всі місця вже зайнято, функція повинна ініціювати виключну ситуацію.

- Висадка пасажира із транспортного засобу. Функція повинна ініціювати виключну ситуацію, якщо вказаний пасажир «не сидить» у транспортному засобі.

Додатково реалізувати функцію підрахунку кількості людей, які перевозяться на автомобілями на дорозі. Варіант коду наданий

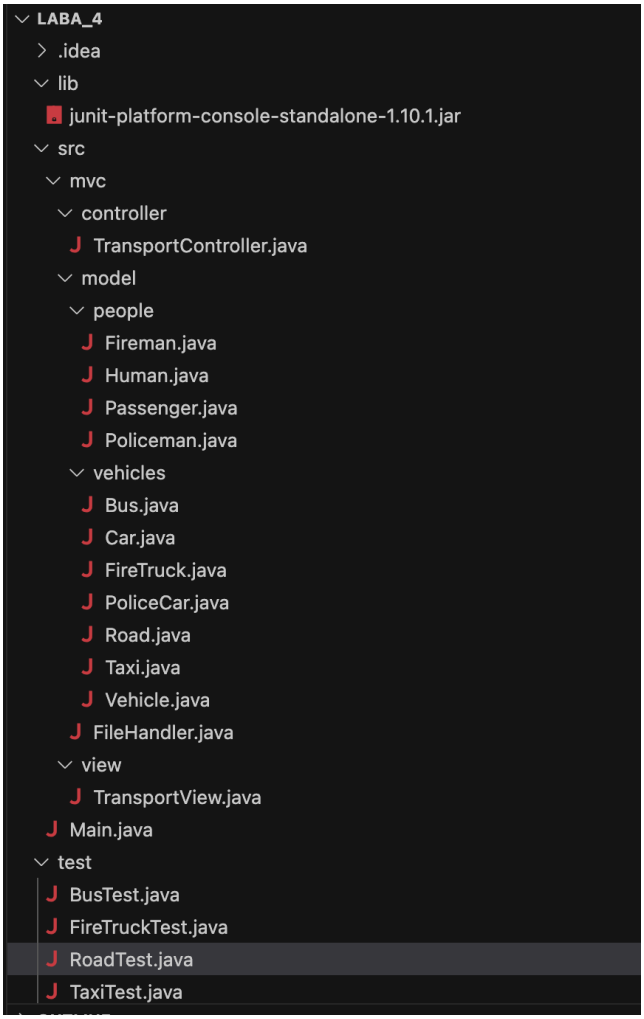
нижче. Дописати код до працездатного. Обов'язково використовувати generics та wildcard (де це потрібно).

```
public class Road {
public List<Vehicle> carsInRoad = new ArrayList<>();
public int getCountOfHumans(){….}
public void addCarToRoad( …. ){ … }}
```

Реалізувати **модульні тести** з наповнення транспортних засобів різними типами пасажирів (не забувайте і про тестування виключних ситуацій).

2. Лістинг програмного коду

2.1. Структура проєкту



2.2. Реалізація Generics

- Клас Vehicle

```
package mvc.model.vehicles;

import mvc.model.people.Human;
import java.util.ArrayList;
import java.util.List;

public abstract class Vehicle<T extends Human> {
    private String modelName;
    private int maxSeats;
    private List<T> passengers = new ArrayList<>();

    public Vehicle(String modelName, int maxSeats) {
        this.modelName = modelName;
        this.maxSeats = maxSeats;
    }

    public void board(T human) throws Exception {
        if (passengers.size() >= maxSeats) {
            throw new Exception("У транспортному засобі '" +
modelName + "' немає вільних місць!");
        }
        passengers.add(human);
    }

    public void unboard(T human) throws Exception {
        if (!passengers.remove(human)) {
            throw new Exception("Пасажира " + human.getName()
+ " немає в салоні '" + modelName + "'!");
        }
    }

    public int getMaxSeats() {
        return maxSeats;
    }

    public List<T> getPassengers() {
        return passengers;
    }

    public int getOccupiedCount() {
        return passengers.size();
    }
}
```

- Клас FireTruck

```
package mvc.model.vehicles;

import mvc.model.people.Fireman;

public class FireTruck extends Car<Fireman> {
    public FireTruck(String modelName, int maxSeats) {
        super(modelName, maxSeats);
    }
}
```

2.3. Реалізація Wildcards та логіки Дороги

- Клас Road

```
package mvc.model.vehicles;

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

public class Road {
    public List<Car<?>> carsInRoad = new ArrayList<>();

    /**
     * Підраховує загальну кількість людей у всіх автомобілях
на дорозі
     */
    public int getCountOfHumans() {
        int count = 0;
        for (Car<?> car : carsInRoad) {
            count += car.getOccupiedCount();
        }
        return count;
    }

    /**
     * Додає автомобіль на дорогу
     * @param car будь-який об'єкт, що успадковує Car
     */
    public void addCarToRoad(Car<?> car) {
        carsInRoad.add(car);
    }
}
```

2.3. Серіалізація

- Клас TransportController

```
package mvc.controller;

import mvc.model.vehicles.*;
import mvc.model.people.*;
import mvc.view.TransportView;
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class TransportController {
    private Road road;
    private TransportView view;
    private final String FILE_NAME = "passengers.dat";

    public TransportController() {
        this.road = new Road();
        this.view = new TransportView();
        setupInitialData();
    }

    /**
     * Початкове наповнення даними згідно з умовами завдання
     */
    private void setupInitialData() {
        try {
            FireTruck ft = new FireTruck("Scania", 2);
            ft.board(new Fireman("Василь"));
            road.addCarToRoad(ft);

            Taxi taxi = new Taxi("Toyota", 4);
            taxi.board(new Policeman("Петро"));
            taxi.board(new Passenger("Марія"));
            road.addCarToRoad(taxi);

            Bus bus = new Bus("Mercedes", 20);
            bus.board(new Passenger("Олексій"));
        } catch (Exception e) {
            view.displayMessage("Помилка при ініціалізації: "
+ e.getMessage());
        }
    }

    /**
     * Збереження пасажирів у файл (ObjectOutputStream)
     */
    public void savePassengers(List<? extends Human> list,
String file) {
        try (ObjectOutputStream out = new
ObjectOutputStream(new FileOutputStream(file))) {
            out.writeObject(list);
```

```
            view.displayMessage("Дані успішно збережено у
файл: " + file);
        } catch (IOException e) {
            view.displayMessage("Помилка запису: " +
e.getMessage());
        }
    }

    /**
     * Читання пасажирів з файлу (ObjectInputStream)
     */
    @SuppressWarnings("unchecked")
    public void loadPassengers(String file) {
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(file))) {
            List<Human> loaded = (List<Human>)
in.readObject();
            view.displayMessage("--- Завантажені пасажири з
файлу ---");
            if (loaded.isEmpty()) {
                view.displayMessage("Файл порожній.");
            } else {
                for (Human h : loaded) {
                    view.displayMessage("- " + h.toString());
                }
            }
        } catch (FileNotFoundException e) {
            view.displayMessage("Файл не знайдено. Спочатку
збережіть дані (пункт 2).");
        } catch (IOException | ClassNotFoundException e) {
            view.displayMessage("Помилка читання: " +
e.getMessage());
        }
    }

    /**
     * Основний цикл програми (Меню)
     */
    public void run() {
        try (Scanner scanner = new Scanner(System.in)) {
            while (true) {
                view.printMenu();
                if (!scanner.hasNextLine()) break;

                String choice = scanner.nextLine();

                switch (choice) {
                    case "1":
                        view.displayMessage("Загальна
кількість людей у машинах на дорозі: " +
road.getCountOfHumans());
```

```
                        break;
                    case "2":
                        List<Human> allOnRoad = new
ArrayList<>();
                        for (Vehicle<?> car : road.carsInRoad)
{

allOnRoad.addAll(car.getPassengers());
                        }
                        savePassengers(allOnRoad, FILE_NAME);
                        break;
                    case "3":
                        loadPassengers(FILE_NAME);
                        break;
                    case "0":
                        view.displayMessage("Завершення
роботи.");
                        return;
                    default:
                        view.displayMessage("Невірний вибір.
Спробуйте ще раз (0-3).");
                        break;
                }
            }
        }
    }
}
```

2.4. Модульні тести

- *BusTest*

```
import mvc.model.people.Passenger;
import mvc.model.vehicles.Bus;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;
import static org.junit.jupiter.api.Assertions.*;

public class BusTest {
    @Test
    @DisplayName("Виключна ситуація: Перевищення максимальної
місткості")
    void testCapacityExceeded() throws Exception {
        Bus bus = new Bus("Sprinter", 2);
        bus.board(new Passenger("Ivan"));
        bus.board(new Passenger("Oleg"));

        Exception ex = assertThrows(Exception.class, () ->
bus.board(new Passenger("Anna")));
        assertEquals("У транспортному засобі 'Sprinter' немає
вільних місць!", ex.getMessage());
```

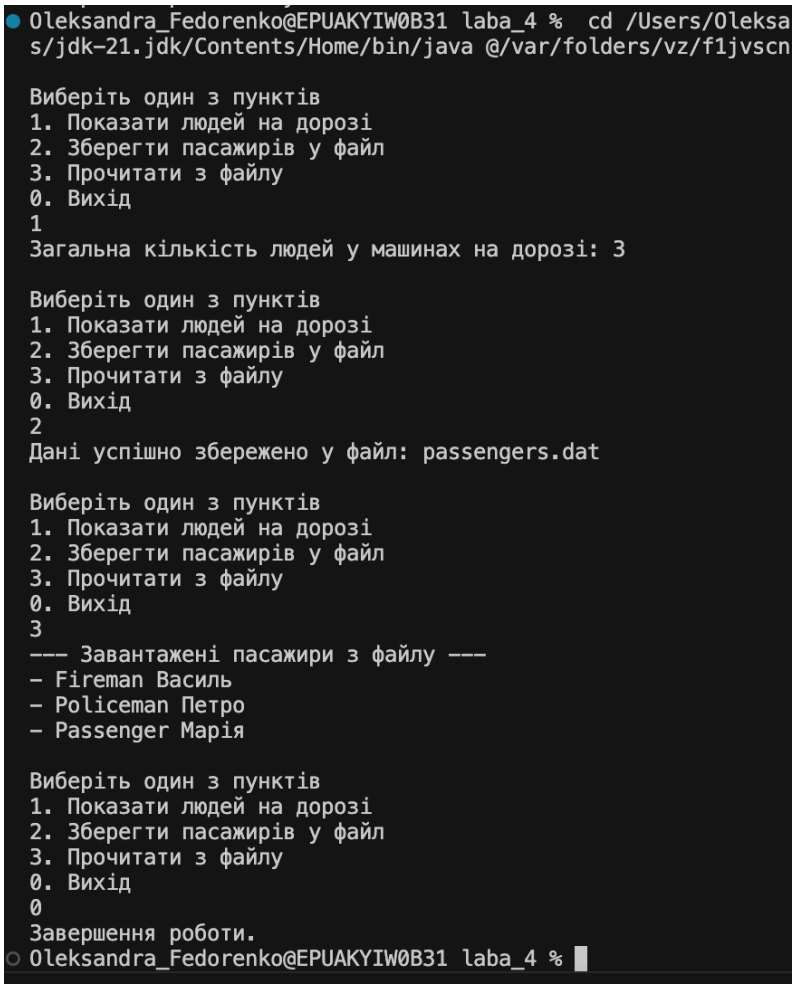
```
    }

    @Test
    @DisplayName("Виключна ситуація: Висадка пасажира з
порожнього автобуса")
    void testUnboardFromEmptyBus() {
        Bus bus = new Bus("Solaris", 10);
        Passenger p = new Passenger("Stepan");

        assertThrows(Exception.class, () -> bus.unboard(p),
"Має кинути помилку, бо автобус порожній");
    }

    @Test
    @DisplayName("Виключна ситуація: Висадка пасажира, який
сидить в іншому авто")
    void testUnboardWrongPassenger() throws Exception {
        Bus bus1 = new Bus("Bus 1", 5);
        Bus bus2 = new Bus("Bus 2", 5);
        Passenger p = new Passenger("Andriy");

        bus1.board(p);

        assertThrows(Exception.class, () -> bus2.unboard(p),
"Bus 2 не може висадити пасажира з Bus 1");
    }
}
```

- RoadTest

```
import mvc.model.people.Passenger;
import mvc.model.people.Policeman;
import mvc.model.vehicles.*;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;
import static org.junit.jupiter.api.Assertions.*;

public class RoadTest {
    @Test
    @DisplayName("Підрахунок людей на порожній дорозі")
    void testEmptyRoad() {
        Road road = new Road();
        assertEquals(0, road.getCountOfHumans(), "На порожній
дорозі має бути 0 людей");
    }

    @Test
    @DisplayName("Складний підрахунок: різні авто з різною
кількістю пасажирів")
    void testComplexRoadCounting() throws Exception {
        Road road = new Road();
```

```
        Taxi t1 = new Taxi("Taxi 1", 4);
        t1.board(new Passenger("A"));
        t1.board(new Passenger("B"));

        PoliceCar pc = new PoliceCar("Police", 2);
        pc.board(new Policeman("C"));

        FireTruck ft = new FireTruck("Fire", 2);

        road.addCarToRoad(t1);
        road.addCarToRoad(pc);
        road.addCarToRoad(ft);

        assertEquals(3, road.getCountOfHumans(), "Загалом має
бути 3 людини");
    }

    @Test
    @DisplayName("Додавання одного і того ж авто двічі")
    void testAddSameCarTwice() {
        Road road = new Road();
        Taxi t = new Taxi("Lanos", 4);

        road.addCarToRoad(t);
        road.addCarToRoad(t);

        assertEquals(2, road.carsInRoad.size());
    }
}
```

3. Скріншоти виконання програми



4. Висновки

В ході виконання даної лабораторної роботи було вивчено та практично застосовано принципи узагальненого програмування (Generics) у мові Java. На прикладі ієрархії транспортних засобів та пасажирів реалізовано механізм суворої типізації, що дозволив обмежити перевезення певних категорій людей у відповідних типах авто на етапі компіляції програми.

Опановано роботу з метасимволами (Wildcards) для створення гнучких колекцій, здатних обробляти різні типи параметризованих об'єктів (клас Road). Також реалізовано механізм збереження та відновлення об'єктів через бінарну серіалізацію (/). Модульне тестування за допомогою JUnit підтвердило надійність розробленої логіки та коректність обробки виключних ситуацій.
ObjectOutputStreamObjectInputStream