Міністерство освіти і науки України Національний університет «Львівська політехніка» Кафедра «Електронних обчислювальних машин»

Звіт з лабораторної роботи № 2 з дисципліни: «Кросплатформенні засоби програмування» на тему: «Класи та пакети»

Виконав:

студент групи КІ-306

Глухенький Дмитро

Прийняв:

доцент кафедри

ЕОМ Іванов Ю. С.

Львів – 2023

Мета роботи: ознайомитися з процесом розробки класів та пакетів мовою Java.

Завдання (варіант № 29)

1. Написати та налагодити програму на мові Java, що реалізує у вигляді класу предметну область згідно варіанту(5. Машина). Програма має задовольняти наступним вимогам:
   - програма має розміщуватися в пакеті Група.Прізвище.Lab3;
   - клас має містити мінімум 3 поля, що є об’єктами класів, які описують складові частини предметної області;
   - клас має містити кілька конструкторів та мінімум 10 методів;
   - для тестування і демонстрації роботи розробленого класу розробити клас-драйвер;
   - методи класу мають вести протокол своєї діяльності, що записується у файл;
   - розробити механізм коректного завершення роботи з файлом (не надіятися на метод finalize());
   - програма має володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленого пакету.
3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації.
4. Дати відповідь на контрольні запитання

Вихідний код програми

Файл CarDriver.java

```
/**
 * The {@code CarDriver} class serves as a driver program to test and demonstrate
 * the functionality of the {@link Car} class, which represents a car in the automotive
domain.
 * It creates instances of the Car class, performs various actions on them, and displays
 * information about the cars and their engines.
 */
public class CarDriver {
```

```
    /**
     * The main method is the entry point for the CarDriver program.
     * It creates instances of the Car class, demonstrates their functionality,
     * and displays information about the cars.
     *
     * @param args The command-line arguments (not used in this program).
     */
    public static void main(String[] args) {
        // Create an engine for the first car
        Engine engine1 = new Engine("Gasoline", 2.0);

        // Create the first car with an engine
        Car car1 = new Car("Toyota", 2022, engine1);

        // Display information about the first car
        car1.displayInformation();

        // Start the first car
        car1.startCar();

        // Accelerate the first car
        car1.accelerate(20); // Accelerate by 20 km/h

        // Decelerate the first car
        car1.decelerate(10); // Decelerate by 10 km/h

        // Turn on the lights of the first car
        car1.turnLightsOn();

        // Display updated information about the first car
        car1.displayInformation();

        // Stop the first car
        car1.stopCar();

        // Create the second car without an engine
        Car car2 = new Car("BMW", 2023);

        // Display information about the second car (without an engine)
        car2.displayInformation();

        // Attempt to accelerate the second car (without an engine)
        car2.accelerate(30); // Should display an error message "Cannot accelerate the
car without an engine."
    }
}
```

Файл Carjava

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

/**
 * Class Car represents a car entity.
 *
 * This class models a car with a brand, manufacturing year, current speed,
 * and an optional engine. It also provides methods for car operations like starting,
 * stopping, changing the engine, accelerating, decelerating, turning on lights,
 * and displaying car information.
 *
 * @author Hlukehnkyi Dmytro
 * @version 1.0
 * @since 1.0
 */
public class Car {
    private final String brand;
    private final int manufacturingYear;
    private int currentSpeed;
    private Engine engine;
    private FileWriter logFileWriter;
    private final String logFileName = "CarDriver.txt";
```

```java
    /**
     * Constructs a car with the specified brand, manufacturing year, and engine.
     *
     * @param brand            The brand of the car.
     * @param manufacturingYear The manufacturing year of the car.
     * @param engine           The Engine object representing the car's engine.
     */
    public Car(String brand, int manufacturingYear, Engine engine) {
        this.brand = brand;
        this.manufacturingYear = manufacturingYear;
        this.engine = engine;
        initLogFileWriter();
    }

    /**
     * Constructs a car with the specified brand and manufacturing year.
     * The engine is set to null by default.
     *
     * @param brand            The brand of the car.
     * @param manufacturingYear The manufacturing year of the car.
     */
    public Car(String brand, int manufacturingYear) {
        this.brand = brand;
        this.manufacturingYear = manufacturingYear;
        this.engine = null;
        initLogFileWriter();
    }

    // Private method to initialize the log file writer
    private void initLogFileWriter() {
        try {
            File logFile = new File(logFileName);

            if (!logFile.exists()) {
                logFile.createNewFile();
            }

            logFileWriter = new FileWriter(logFile, true); // Append mode
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Logs a message to the car's log file.
     *
     * @param message The message to be logged.
     */
    public void log(String message) {
        try {
            logFileWriter.write(message + "\n");
            logFileWriter.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Disposes of the log file writer and closes the log file.
     *
     * @throws IOException Thrown in case of input/output errors.
     */
    public void dispose() throws IOException {
        if (logFileWriter != null) {
            logFileWriter.close();
        }
    }

    /**
     * Starts the car if an engine is available and logs the activity.
     */
    public void startCar() {
        if (engine != null) {
            System.out.println("Brand: " + brand + " Car started.");
```

```
            log("Brand: " + brand + " Car started.");
        } else {
            System.out.println("Cannot start the car without an engine.");
            log("Cannot start the car without an engine.");
        }
    }

    /**
     * Stops the car and logs the activity.
     */
    public void stopCar() {
        System.out.println("Brand: " + brand + " Car stopped.");
        log("Brand: " + brand + " Car stopped.");
    }

    /**
     * Changes the car's engine and logs the activity.
     *
     * @param newEngine The new Engine object to replace the current engine.
     */
    public void changeEngine(Engine newEngine) {
        engine = newEngine;
        System.out.println("Brand: " + brand + " Engine changed.");
        log("Brand: " + brand + " Engine changed.");
    }

    /**
     * Displays information about the car including brand, manufacturing year,
     * and engine details if available.
     */
    public void displayInformation() {
        System.out.println("Brand: " + brand);
        log("Brand: " + brand);
        System.out.println("Manufacturing Year: " + manufacturingYear);
        log("Manufacturing Year: " + manufacturingYear);
        if (engine != null) {
            engine.displayInformation();
        } else {
            System.out.println("Engine not available.");
            log("Engine not available.");
        }
    }

    /**
     * Accelerates the car by increasing its speed by a specified amount.
     *
     * @param accelerationAmount The amount by which to accelerate the car in km/h.
     */
    public void accelerate(int accelerationAmount) {
        if (engine != null) {
            int newSpeed = currentSpeed + accelerationAmount;
            System.out.println("Brand: " + brand + " Car accelerated to " + newSpeed + "
km/h.");
            log("Brand: " + brand + " Car accelerated to " + newSpeed + " km/h.");
            currentSpeed = newSpeed;
        } else {
            System.out.println("Cannot accelerate the car without an engine.");
            log("Cannot accelerate the car without an engine.");
        }
    }

    /**
     * Decelerates the car by reducing its speed by a specified amount.
     *
     * @param decelerationAmount The amount by which to decelerate the car in km/h.
     */
    public void decelerate(int decelerationAmount) {
        if (engine != null) {
            int newSpeed = currentSpeed - decelerationAmount;
            if (newSpeed < 0) {
                newSpeed = 0;
            }
            System.out.println("Brand: " + brand + " Car decelerated to " + newSpeed + "
km/h.");
```

```
            log("Brand: " + brand + " Car decelerated to " + newSpeed + " km/h.");
            currentSpeed = newSpeed;
        } else {
            System.out.println("Cannot decelerate the car without an engine.");
            log("Cannot decelerate the car without an engine.");
        }
    }

    /**
     * Turns on the lights of the car and logs the activity.
     */
    public void turnLightsOn() {
        if (engine != null) {
            Boolean lightsOn = true;
            System.out.println("Brand: " + brand + " Car lights turned on.");
            log("Brand: " + brand + " Car lights turned on.");
        } else {
            System.out.println("Cannot turn on the lights without an engine.");
            log("Cannot turn on the lights without an engine.");
        }
    }
}
```

Файл Engine.java

```
/**
 * The {@code Engine} class represents the engine component of a car.
 * It stores information about the engine type and displacement.
 */
public class Engine {
    private String type;         // The type of the engine (e.g., "Gasoline", "Diesel")
    private final double displacement; // The engine displacement in liters (e.g., 2.0,
2.5)

    /**
     * Constructs an Engine object with the specified type and displacement.
     *
     * @param type         The type of the engine (e.g., "Gasoline", "Diesel").
     * @param displacement The engine displacement in liters (e.g., 2.0, 2.5).
     */
    public Engine(String type, double displacement) {
        this.type = type;
        this.displacement = displacement;
    }

    /**
     * Displays information about the engine, including its type and displacement.
     * Example output:
     * Engine Type: Gasoline
     * Displacement: 2.0 L
     */
    public void displayInformation() {
        System.out.println("Engine Type: " + type);
        System.out.println("Displacement: " + displacement + " L");
    }
}
```

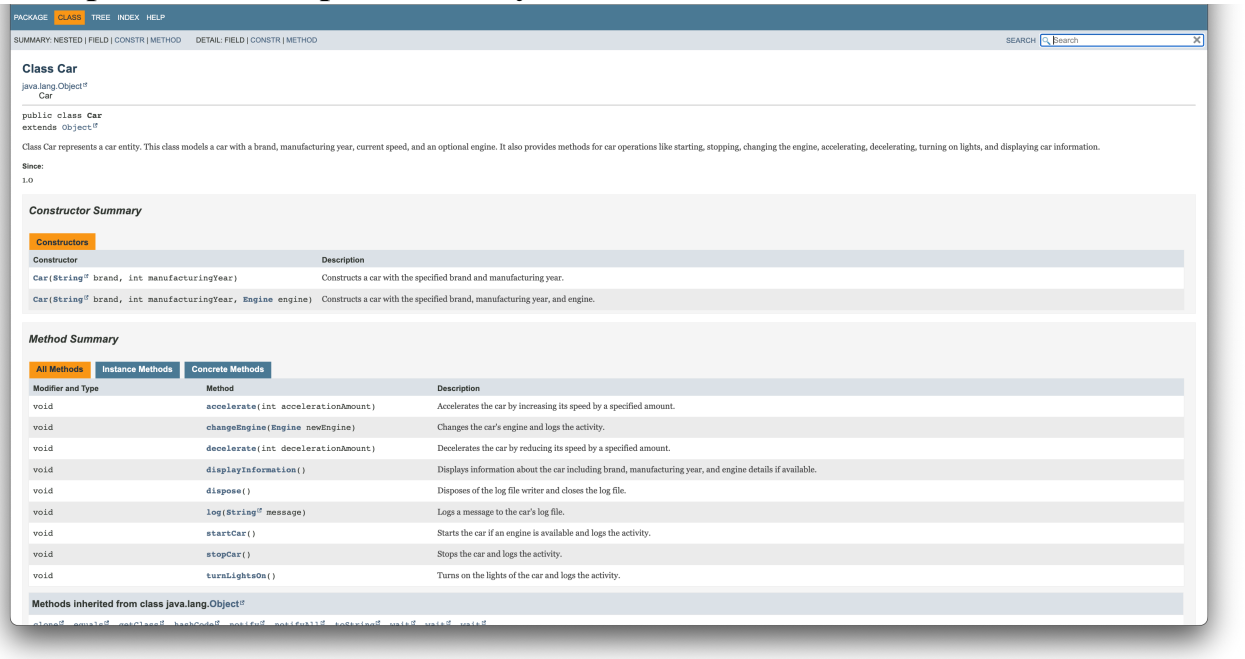
Результат виконання програми

CarDriver.txt:

Car.java | Engine.java | CarDriver.java | CarDriver.txt

```
Brand: Toyota
Manufacturing Year: 2022
Brand: Toyota Car started.
Brand: Toyota Car accelerated to 20 km/h.
Brand: Toyota Car decelerated to 10 km/h.
Brand: Toyota Car lights turned on.
Brand: Toyota
Manufacturing Year: 2022
Brand: Toyota Car stopped.
Brand: BMW
Manufacturing Year: 2023
Engine not available.
Cannot accelerate the car without an engine.
```

Фрагмент згенерованої документації



PACKAGE CLASS TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH

Class Car

java.lang.Object
Car

public class **Car**
extends Object

Class Car represents a car entity. This class models a car with a brand, manufacturing year, current speed, and an optional engine. It also provides methods for car operations like starting, stopping, changing the engine, accelerating, decelerating, turning on lights, and displaying car information.

Since:
1.0

Constructor Summary

Constructors

Constructor	Description
Car(String brand, int manufacturingYear)	Constructs a car with the specified brand and manufacturing year.
Car(String brand, int manufacturingYear, Engine engine)	Constructs a car with the specified brand, manufacturing year, and engine.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
void	accelerate(int accelerationAmount)	Accelerates the car by increasing its speed by a specified amount.
void	changeEngine(Engine newEngine)	Changes the car's engine and logs the activity.
void	decelerate(int decelerationAmount)	Decelerates the car by reducing its speed by a specified amount.
void	displayInformation()	Displays information about the car including brand, manufacturing year, and engine details if available.
void	dispose()	Disposes of the log file writer and closes the log file.
void	log(String message)	Logs a message to the car's log file.
void	startCar()	Starts the car if an engine is available and logs the activity.
void	stopCar()	Stops the car and logs the activity.
void	turnLightsOn()	Turns on the lights of the car and logs the activity.

Methods inherited from class java.lang.Object

Відповіді на контрольні запитання

1. Синтаксис визначення класу.
   - public class ClassName {
     // Class members (fields, methods, constructors)
     }

2. Синтаксис визначення методу.
   - public returnType methodName(parameters) {

```
    // Method body
}
```

3. Синтаксис оголошення поля.
   - accessModifier dataType fieldName;

4. Як оголосити та ініціалізувати константне поле?
   - public static final dataType CONSTANT_NAME = initial_value;

5. Які є способи ініціалізації полів?
   - Явна ініціалізація при оголошенні поля.
   - Ініціалізація у конструкторі класу.
   - Ініціалізація у блоку ініціалізації (конструкторі, статичному або звичайному).

6. Синтаксис визначення конструктора.
   - public ClassName(parameters) {
     // Constructor body
     }

7. Синтаксис оголошення пакету.
   - package packageName.subpackage;

8. Як підключити до програми класи, що визначені в зовнішніх пакетах?
   - Вказати повне ім'я класу перед використанням (наприклад, java.util.Date today = new java.util.Date();).
   - Використовувати оператор import для підключення класів з інших пакетів, щоб уникнути повторення повного імені класу.

9. В чому суть статичного імпорту пакетів?

- Статичний імпорт дозволяє підключити статичні методи і поля класів без повного імені класу.
- Завдяки статичному імпорту, можна використовувати статичні члени класу, не додаваючи перед ними ім'я класу.

10. Які вимоги ставляться до файлів і каталогів при використанні пакетів?
    - Назви пакетів повинні відповідати структурі каталогів.
    - Назви загальнодоступних класів повинні співпадати з назвами файлів, де вони розміщені.
    - Після компіляції ієрархія каталогів проекту повинна відповідати ієрархії пакетів.
    - Для компіляції та запуску програми слід використовувати шляхи до файлів та пакетів.

Висновок

У ході виконання даної лабораторної роботи, отримав цінні навички розробки класів та пакетів у мові програмування Java. Ця лабораторна робота надала мені можливість ознайомитися з базовими конструкціями Java, такими як оголошення класів, методів та полів. Я навчився правильно структурувати свій код, визначати доступ до класів та їх членів, а також використовувати модифікатори доступу для керування видимістю.