**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**Інститут комп’ютерних технологій, автоматики та метрології**

**кафедра “Електронних обчислювальних машин”**

****

Звіт

з лабораторної роботи №2

дисципліни «Кросплатформні засоби програмування»

на тему: **«**КЛАСИ ТА ПАКЕТИ**»**

Варіант 15

**Виконала:**

студентка групи КІ-303

Сухарник А.А.

**Прийняв:**

Іванов Ю.С.

Львів – 2024

ЛАБОРАТОРНА РОБОТА №2

**КЛАСИ ТА ПАКЕТИ**

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

**Теоретичний матеріал**

Мова Java є повністю об’єктно-орієнтованою мовою програмування, тому вона дозволяє писати програми лише з використанням об’єктно-орієнтованих парадигм програмування, що базуються на понятті класів.

Необов’язковий специфікатор доступу public робить клас загальнодоступним. У кожному файлі з кодом програми може бути лише один загальнодоступний клас, ім’я якого співпадає з назвою файлу, та безліч класів без специфікатора public.

Ініціалізація посилання на об’єкт класу здійснюється за допомогою оператора new і вказування конструктора, який має збудувати об’єкт. Одержаний в результаті цих операцій об’єкт розташується у області оперативної пам’яті що зветься ”куча”.

При створенні об’єктів дозволяється суміщати оголошення та ініціалізацію об’єктів, а також створювати анонімні об’єкти. Якщо посилання на об’єкт не посилається на жоден об’єкт, то йому слід присвоїти значення null. На відміну від полів-посилань на об’єкти, локальні змінні-посилання на об’єкти не ініціалізуються значенням null при оголошенні. Для них ініціалізацію посилання слід проводити явно.

Метод – функція-член класу, яка призначена маніпулювати станом об’єкту класу. Методи можуть бути перевантаженими. Перевантаження методів відбувається шляхом вказування різної кількості параметрів та їх типів методам з однаковими назвами.

Конструктори, методи, та поля класу можуть бути відкритими (public), закритими (private) та захищеними (protected), що визначається специфікатором доступу. Специфікатор доступу public робить елемент класу загальнодоступним в межах пакету (набору класів, з яких складається програма). Специфікатор доступу private робить елемент класу закритим (недоступним) для всіх зовнішніх відносно даного класу елементів програми (включаючи похідні класи). Специфікатор доступу protected робить елемент класу закритим (недоступним) для всіх зовнішніх відносно даного класу елементів програми, проте цей елемент буде загальнодоступним для похідних класів. Якщо будь-який елемент класу не має специфікатора доступу, то цей елемент автоматично стає відкритим та видимим у межах пакету (не плутати з public). Всі елементи класу, що оголошені без використання ключового слова static, належать об’єкту класу. Тобто, кожен об’єкт класу містить власну копію цих елементів класу. Ключове слово static робить поле або метод членом класу, а не об’єкту, тобто вони є спільними для всіх об’єктів класу. Оскільки клас існує завжди, на відміну від об’єктів, які створюються в процесі роботи програми, то статичні елементи класу доступні.

Пакет – це механізм мови Java, що дозволяє об’єднувати класи в простори імен. Об’єднання класів в пакети дозволяє відділяти класи, що розроблені одними розробниками, від класів, що розроблені іншими розробниками, забезпечуючи тим самим унікальність імен класів в межах програми та усуваючи можливі конфлікти імен класів. Пакети можуть бути вкладеними одні в одних, утворюючи цим самим ієрархії пакетів. Будь-який зв’язок між вкладеними пакетами відсутній. Всі стандартні пакети належать ієрархіям java і javax, наприклад, java.lang, java.util, java.net тощо.

**Завдання (Варіант №15)**

1. Написати та налагодити програму на мові Java згідно варіанту. Програма має задовольняти наступним вимогам

* програма має розміщуватися в пакеті 303.Sukharnyk.Lab2;
* клас має містити мінімум 3 поля, що є об’єктами класів, які описують складові частини предметної області; **ВАРІАНТ ЗАВДАННЯ: Корабель**
* 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.

1. Автоматично згенерувати документацію до розробленої програми.
2. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub
3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.
4. Дати відповідь на контрольні запитання.

**Виконання завдання**

**ShipDriver.java**

Package 303.Sukharnyk.Lab2;  
  
import java.io.IOException;  
  
  
*/\*\*  
 \* Клас ShipDriver використовується для демонстрації роботи класу Ship.  
 \* У цьому класі створюється об'єкт корабля і викликаються різні методи для керування ним.  
 \*/*public class ShipDriver {  
 */\*\*  
 \* Точка входу в програму. Демонструє роботу з телевізором шляхом виклику різних методів.  
 \*  
 \* @param args Аргументи командного рядка.  
 \*/* public static void main(String[] args) {  
 try {  
 Ship ship = new Ship(  
 new Engine(false, 0, 1000),  
 new Hull(70),  
 new NavigationSystem(20, 0, "2.0"),  
 "Темний Клик"  
 );  
  
 ship.startEngine();  
 ship.changeCourse(10);  
 ship.changeSpeed(50);  
 ship.checkHullStatus();  
 ship.updateNavigationSystem();  
 ship.refuel(100);  
 ship.performMaintenance();  
 ship.calculateArrivalTime(10);  
 ship.simulateWeatherConditions("шторм");  
 ship.stopEngine();  
  
 ship.closeLogger();  
 } catch (IOException e) {  
 // Обробка помилок, що виникають під час запису в файл  
 throw new RuntimeException("Сталася помилка при записі в файл: " + e.getMessage());  
 }  
 }  
}

**Ship.java**

Package 303.Sukharnyk.Lab2;  
  
import java.io.IOException;  
import java.util.Random;  
*/\*\*  
 \* Клас, що представляє корабель з двигуном, корпусом та навігаційною системою.  
 \* Забезпечує управління основними функціями корабля.  
 \*/*public class Ship {  
 private Engine engine;  
 private Hull hull;  
 private NavigationSystem navigationSystem;  
 private Logger logger;  
 private String name;  
  
 */\*\*  
 \* Створює новий об'єкт корабля з заданим ім'ям.  
 \* @param name Ім'я корабля  
 \* @throws IOException якщо виникає помилка під час створення логера  
 \*/* public Ship(String name) throws IOException {  
 this.name = name;  
 this.engine = new Engine();  
 this.hull = new Hull();  
 this.navigationSystem = new NavigationSystem();  
  
 this.logger = new Logger("ship\_log.txt");  
 logger.log(String.*format*("Корабель %s створено.", this.toString()));  
 }  
  
 */\*\*  
 \* Створює новий об'єкт корабля з параметрами двигуна, корпусу та навігаційної системи.  
 \* @param engine Двигун корабля  
 \* @param hull Корпус корабля  
 \* @param navigationSystem Навігаційна система  
 \* @param name Ім'я корабля  
 \* @throws IOException якщо виникає помилка під час створення логера  
 \*/* public Ship(Engine engine, Hull hull, NavigationSystem navigationSystem, String name) throws IOException {  
 this.engine = engine;  
 this.hull = hull;  
 this.navigationSystem = navigationSystem;  
 this.name = name;  
  
 this.logger = new Logger("ship\_log.txt");  
 logger.log(String.*format*("Корабель %s створено.", this.toString()));  
 }  
  
 */\*\*  
 \* Метод для запуску двигуна корабля.  
 \* @throws IOException якщо виникає помилка під час запису до логера  
 \*/* public void startEngine() throws IOException {  
 engine.start();  
 logger.log(String.*format*("Двигун корабля %s запущено", name));  
 System.*out*.printf("Двигун корабля %s запущено\n", name);  
 }  
  
 */\*\*  
 \* Метод для зупинки двигуна корабля.  
 \* @throws IOException якщо виникає помилка під час запису до логера  
 \*/* public void stopEngine() throws IOException {  
 engine.stop();  
 logger.log(String.*format*("Двигун корабля %s зупинено", name));  
 System.*out*.printf("Двигун корабля %s зупинено\n", name);  
 }  
  
 */\*\*  
 \* Метод для встановлення курсу корабля.  
 \* @param course Курс корабля  
 \* @throws IOException якщо виникає помилка під час запису до логера  
 \*/* public void changeCourse(int course) throws IOException {  
 navigationSystem.setCourse(course);  
 logger.log(String.*format*("Встановлено курс %s для корабля %s", course, name));  
 System.*out*.printf("Встановлено курс %s для корабля %s\n", course, name);  
 }  
  
 */\*\*  
 \* Метод для зміни швидкості корабля.  
 \* @param speed Нова швидкість  
 \* @throws IOException якщо виникає помилка під час запису до логера  
 \*/* public void changeSpeed(int speed) throws IOException {  
 engine.setSpeed(speed);  
 logger.log(String.*format*("Змінено швидкість корабля %s на %s", name, speed));  
 System.*out*.printf("Змінено швидкість корабля %s на %s\n", name, speed);  
 }  
  
 */\*\*  
 \* Метод для перевірки стану корпусу.  
 \* @return Стан корпусу  
 \* @throws IOException якщо виникає помилка під час запису до логера  
 \*/* public String checkHullStatus() throws IOException {  
 String status = hull.checkStatus();  
 logger.log(String.*format*("Перевірено стан корпусу корабля %s : %s", name, status));  
 System.*out*.printf("Перевірено стан корпусу корабля %s : %s\n", name, status);  
 return status;  
 }  
  
 */\*\*  
 \* Метод для оновлення навігаційної системи.  
 \* @throws IOException якщо виникає помилка під час запису до логера  
 \*/* public void updateNavigationSystem() throws IOException {  
 navigationSystem.update();  
 logger.log(String.*format*("Оновлено навігаційну систему корабля %s", name));  
 System.*out*.printf("Оновлено навігаційну систему корабля %s\n", name);  
 }  
  
 */\*\*  
 \* Метод для дозаправки корабля.  
 \* @param amount Кількість палива  
 \* @throws IOException якщо виникає помилка під час запису до логера  
 \*/* public void refuel(int amount) throws IOException {  
 engine.refuel(amount);  
 logger.log(String.*format*("Дозаправлено корабель %s на %s літрів", name, amount));  
 System.*out*.printf("Дозаправлено корабель %s на %s літрів\n", name, amount);  
 }  
  
 */\*\*  
 \* Метод для проведення технічного обслуговування корабля.  
 \* @return Результат технічного обслуговування  
 \* @throws IOException якщо виникає помилка під час запису до логера  
 \*/* public String performMaintenance() throws IOException {  
 Random random = new Random();  
 int maintenanceQuality = random.nextInt(100) + 1;  
 String result;  
  
 if (maintenanceQuality > 80) {  
 result = "Відмінне технічне обслуговування";  
 hull.improveCondition(10);  
 } else if (maintenanceQuality > 60) {  
 result = "Добре технічне обслуговування";  
 hull.improveCondition(5);  
 } else if (maintenanceQuality > 40) {  
 result = "Задовільне технічне обслуговування";  
 hull.improveCondition(2);  
 } else {  
 result = "Незадовільне технічне обслуговування";  
 }  
  
 logger.log(String.*format*("Проведено технічне обслуговування корабля %s: %s", name, result));  
 System.*out*.printf("Проведено технічне обслуговування корабля %s: %s\n", name, result);  
 return result;  
 }  
  
 */\*\*  
 \* Метод для розрахунку часу прибуття до пункту призначення.  
 \* @param distance Відстань до пункту призначення в кілометрах  
 \* @return Час прибуття в годинах  
 \* @throws IOException якщо виникає помилка під час запису до логера  
 \*/* public double calculateArrivalTime(double distance) throws IOException {  
 int currentSpeed = engine.getSpeed();  
 if (currentSpeed == 0) {  
 throw new IllegalStateException("Корабель не рухається. Запустіть двигун та встановіть швидкість.");  
 }  
  
 double time = distance / currentSpeed;  
 logger.log(String.*format*("Розраховано час прибуття корабля %s: %.2f годин", name, time));  
 System.*out*.printf("Розраховано час прибуття корабля %s: %.2f годин\n", name, time);  
 return time;  
 }  
  
 */\*\*  
 \* Метод для симуляції погодних умов та їх впливу на корабель.  
 \* @param weatherCondition Погодні умови (напр., "шторм", "ясно", "туман")  
 \* @throws IOException якщо виникає помилка під час запису до логера  
 \*/* public void simulateWeatherConditions(String weatherCondition) throws IOException {  
 switch (weatherCondition.toLowerCase()) {  
 case "шторм":  
 hull.deteriorateCondition(15);  
 engine.setSpeed(engine.getSpeed() / 2);  
 logger.log(String.*format*("Корабель %s потрапив у шторм. Швидкість знижено, стан корпусу погіршився.", name));  
 break;  
 case "ясно":  
 navigationSystem.improveAccuracy(10);  
 logger.log(String.*format*("Ясна погода покращила точність навігації корабля %s.", name));  
 break;  
 case "туман":  
 navigationSystem.reduceAccuracy(20);  
 engine.setSpeed(engine.getSpeed() / 3);  
 logger.log(String.*format*("Корабель %s у тумані. Швидкість знижено, точність навігації погіршилась.", name));  
 break;  
 default:  
 logger.log(String.*format*("Невідомі погодні умови для корабля %s: %s", name, weatherCondition));  
 }  
  
 System.*out*.printf("Симульовано погодні умови для корабля %s: %s\n", name, weatherCondition);  
 }  
  
 */\*\*  
 \* Закриває логер для збереження даних у файл.  
 \* @throws IOException якщо виникає помилка під час закриття логера  
 \*/* public void closeLogger() throws IOException {  
 logger.close();  
 }  
}

**Engine.java**

Package 303.Sukharnyk.Lab2;  
  
*/\*\*  
 \* Клас Engine представляє двигун корабля.  
 \* Дозволяє керувати станом двигуна, швидкістю та рівнем палива.  
 \*/*public class Engine {  
 private boolean isRunning;  
 private int speed;  
 private int fuelLevel;  
  
 */\*\*  
 \* Конструктор за замовчуванням, який створює двигун із вимкненим станом,  
 \* початковою швидкістю 0 та рівнем палива 100.  
 \*/* public Engine() {  
 isRunning = false;  
 speed = 0;  
 fuelLevel = 100;  
 }  
  
 */\*\*  
 \* Конструктор для створення двигуна з вказаним початковим станом.  
 \* @param isRunning Чи працює двигун  
 \* @param speed Початкова швидкість  
 \* @param fuelLevel Початковий рівень палива  
 \*/* public Engine(boolean isRunning, int speed, int fuelLevel) {  
 this.isRunning = isRunning;  
 this.speed = speed;  
 this.fuelLevel = fuelLevel;  
 }  
  
 */\*\*  
 \* Запускає двигун.  
 \*/* public void start() {  
 isRunning = true;  
 }  
  
 */\*\*  
 \* Зупиняє двигун і скидає швидкість до 0.  
 \*/* public void stop() {  
 isRunning = false;  
 speed = 0;  
 }  
  
 */\*\*  
 \* Повертає поточну швидкість двигуна.  
 \* @return Швидкість  
 \*/* public int getSpeed() {  
 return speed;  
 }  
  
 */\*\*  
 \* Встановлює нову швидкість для двигуна.  
 \* @param speed Нова швидкість  
 \*/* public void setSpeed(int speed) {  
 this.speed = speed;  
 }  
  
 */\*\*  
 \* Дозаправляє двигун на вказану кількість одиниць палива.  
 \* @param amount Кількість палива  
 \*/* public void refuel(int amount) {  
 fuelLevel += amount;  
 }  
  
 */\*\*  
 \* Повертає інформацію про поточний стан двигуна, швидкість та рівень палива.  
 \* @return Інформація про двигун у вигляді рядка  
 \*/* public String getInfo() {  
 return "Стан: " + (isRunning ? "працює" : "зупинено") +  
 ", Швидкість: " + speed +  
 ", Рівень палива: " + fuelLevel;  
 }  
}

**Hull.java**

Package 303.Sukharnyk.Lab2;  
  
*/\*\*  
 \* Клас Hull представляє корпус корабля та його стан.  
 \* Дозволяє перевіряти стан корпусу, погіршувати чи покращувати його,  
 \* а також отримувати інформацію про поточний стан.  
 \*/*public class Hull {  
 private int condition;  
  
 */\*\*  
 \* Створює корпус корабля з початковим станом 100%.  
 \*/* public Hull() {  
 condition = 100;  
 }  
  
 */\*\*  
 \* Створює корпус корабля з заданим початковим станом.  
 \* @param condition Початковий стан корпусу (у відсотках)  
 \*/* public Hull(int condition) {  
 this.condition = condition;  
 }  
  
 */\*\*  
 \* Перевіряє поточний стан корпусу та повертає описовий статус.  
 \* @return Статус корпусу у вигляді рядка  
 \*/* public String checkStatus() {  
 if (condition > 90) return "Відмінний";  
 if (condition > 70) return "Добрий";  
 if (condition > 50) return "Задовільний";  
 if (condition > 30) return "Поганий";  
 return "Критичний";  
 }  
  
 */\*\*  
 \* Погіршує стан корпусу на вказану кількість одиниць, але не менше 0.  
 \* @param amount Кількість одиниць для погіршення стану  
 \*/* public void deteriorateCondition(int amount) {  
 this.condition = Math.*max*(0, this.condition - amount);  
 }  
  
 */\*\*  
 \* Покращує стан корпусу на вказану кількість одиниць, але не більше 100.  
 \* @param amount Кількість одиниць для покращення стану  
 \*/* public void improveCondition(int amount) {  
 this.condition = Math.*min*(100, this.condition + amount);  
 }  
  
 */\*\*  
 \* Повертає інформацію про поточний стан корпусу у відсотках.  
 \* @return Інформація про стан корпусу у вигляді рядка  
 \*/* public String getInfo() {  
 return "Стан корпусу: " + condition + "%";  
 }  
}

**NavigationSystem.java**

Package 303.Sukharnyk.Lab2;  
  
*/\*\*  
 \* Клас NavigationSystem представляє навігаційну систему корабля.  
 \* Він дозволяє встановлювати курс, підвищувати або знижувати точність навігації,  
 \* оновлювати програмне забезпечення та отримувати інформацію про стан системи.  
 \*/*public class NavigationSystem {  
 private int course;  
 private int accuracy;  
 private String softwareVersion;  
  
 */\*\*  
 \* Створює нову навігаційну систему з початковими значеннями.  
 \* Курс встановлено на 0, точність на 0, версія ПЗ — "1.0".  
 \*/* public NavigationSystem() {  
 this.course = 0;  
 this.accuracy = 0;  
 this.softwareVersion = "1.0";  
 }  
  
 */\*\*  
 \* Створює нову навігаційну систему з заданими параметрами.  
 \* @param course Курс корабля  
 \* @param accuracy Точність навігації  
 \* @param softwareVersion Версія програмного забезпечення  
 \*/* public NavigationSystem(int course, int accuracy, String softwareVersion) {  
 this.course = course;  
 this.accuracy = accuracy;  
 this.softwareVersion = softwareVersion;  
 }  
  
 */\*\*  
 \* Підвищує точність навігації на задану кількість одиниць, не перевищуючи 100.  
 \* @param amount Кількість одиниць для підвищення точності  
 \*/* public void improveAccuracy(int amount) {  
 this.accuracy = Math.*min*(100, this.accuracy + amount);  
 }  
  
 */\*\*  
 \* Знижує точність навігації на задану кількість одиниць, але не менше 0.  
 \* @param amount Кількість одиниць для зниження точності  
 \*/* public void reduceAccuracy(int amount) {  
 this.accuracy = Math.*max*(0, this.accuracy - amount);  
 }  
  
 */\*\*  
 \* Встановлює новий курс корабля.  
 \* @param course Новий курс  
 \*/* public void setCourse(int course) {  
 this.course = course;  
 }  
  
 */\*\*  
 \* Оновлює програмне забезпечення навігаційної системи до версії "1.1".  
 \*/* public void update() {  
 softwareVersion = "1.1";  
 }  
  
 */\*\*  
 \* Повертає інформацію про поточний курс і версію програмного забезпечення.  
 \* @return Інформація у вигляді рядка  
 \*/* public String getInfo() {  
 return "Курс: " + course + ", Версія ПЗ: " + softwareVersion;  
 }  
}

**Logger.java**

Package 303.Sukharnyk.Lab2;  
  
import java.io.FileWriter;  
import java.io.IOException;  
  
*/\*\*  
 \* Клас Logger забезпечує логування повідомлень у файл.  
 \* Використовується для запису дій та подій, що відбуваються в програмі.  
 \*/*public class Logger {  
 private FileWriter fileWriter;  
  
 */\*\*  
 \* Конструктор створює об'єкт Logger для запису повідомлень у вказаний файл.  
 \*  
 \* @param fileName ім'я файлу для запису логів.  
 \* @throws IOException якщо виникає помилка при створенні або відкритті файлу.  
 \*/* public Logger(String fileName) throws IOException {  
 fileWriter = new FileWriter(fileName, true);  
 }  
  
 */\*\*  
 \* Метод записує повідомлення у файл логу.  
 \*  
 \* @param message повідомлення, яке потрібно записати у файл.  
 \* @throws IOException якщо виникає помилка при записі у файл.  
 \*/* public void log(String message) throws IOException {  
 if (fileWriter != null) {  
 fileWriter.write(message + "\n");  
 fileWriter.flush();  
 }  
 }  
  
 */\*\*  
 \* Метод закриває файл логу, звільняючи всі ресурси, пов'язані з ним.  
 \* У разі виникнення помилки при закритті, повідомлення про помилку буде виведено в консоль.  
 \*/* public void close() {  
 if (fileWriter != null) {  
 try {  
 fileWriter.close();  
 } catch (IOException e) {  
 System.*err*.println("Виникла помилка при закриванні файла: " + e.getMessage());  
 }  
 }  
 }  
}

**Відповіді на контрольні питання**

1. Синтаксис визначення класу. [public] class Name {}
2. Синтаксис визначення методу. [public] returnType nameMethod(param){}
3. Синтаксис оголошення поля. [public] тип fieldname;
4. Як оголосити та ініціалізувати константне поле? [public] static final [int ]CONSTANT\_NAME = 100;
5. Які є способи ініціалізації полів? Пряма ініціалізація при оголошенні, Ініціалізація в конструкторі, Ініціалізація через блок ініціалізації або статичний блок.
6. Синтаксис визначення конструктора. public ClassName (param){}
7. Синтаксис оголошення пакету. package com.example.myapp;
8. Як підключити до програми класи, що визначені в зовнішніх пакетах? import com.example.otherpackage.ClassName;
9. В чому суть статичного імпорту пакетів? Дозволяє імпортувати статичні поля або методи класу, щоб використовувати їх без вказування імені класу.
10. Які вимоги ставляться до файлів і каталогів при використанні пакетів? Ім'я пакету має відповідати структурі каталогів. Файли класів мають бути розміщені в каталогах, що відповідають іменам пакетів.

**Висновок:** Я ознайомилась з процесом розробки класів та пакетів мовою Java.