**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

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

**кафедра “Електронних обчислювальних машин”**

****

Звіт

з лабораторної роботи №3

дисципліни «Кросплатформні засоби програмування»

на тему: **«**Спадкування та інтерфейси**»**

Варіант 15

**Виконала:**

студентка групи КІ-303

Сухарник А.А.

**Прийняв:**

Іванов Ю.С.

Львів – 2024

ЛАБОРАТОРНА РОБОТА №3

**СПАДКУВАННЯ ТА ІНТЕРФЕЙСИ**

**Мета роботи**: ознайомитися з спадкуванням та інтерфейсами у мові Java.

**Теоретичний матеріал**

Спадкування в ООП призначене для розширення функціональності існуючих класів шляхом утворення нових класів на базі вже існуючих. У Java реалізована однокоренева архітектура класів згідно якої всі класи мають єдиного спільного предка (кореневий клас в ієрархії класів) – клас Object. Решта класів мови Java утворюються шляхом успадковування даного класу. Будь-яке спадкування у мові Java є відкритим, при цьому аналогів захищеному і приватному спадкуванню мови С++ не існує. На відміну від С++ у Java можливе спадкування лише одного базового класу (множинне спадкування відсутнє). Спадкування реалізується шляхом вказування ключового слова class піcля якого вказується назва підкласу, ключове слово extends та назва суперкласу, що розширюється у новому підкласі.

При наслідувані у Java дозволяється перевизначення (перевантаження) методів та полів. При цьому область видимості методу, що перевизначається, має бути не меншою, ніж область видимості цього методу у суперкласі, інакше компілятор видасть повідомлення, про обмеження привілеїв доступу до даних. Перевизначення методу полягає у визначені у підкласі методу з сигнатурою методу суперкласу. При виклику такого методу з-під об’єкта підкласу викличеться метод цього підкласу. Якщо ж у підкласі немає визначеного методу, що викликається, то викличеться метод суперкласу. Якщо ж у суперкласі даний метод також відсутній, то згенерується повідомлення про помилку.

Використання ключового слова super у конструкторах підкласів має дещо інший сенс, ніж у методах. Тут воно застосовується для виклику конструктора суперкласу. Виклик конструктора суперкласу має бути першим оператором конструктора підкласу. Конкретний конструктор, який необхідно викликати, вибирається по переданим параметрам. Явний виклик конструктора суперкласу часто є необхідним, оскільки підкласи не мають доступу до приватних полів суперкласів. Тож ініціалізація їх полів значеннями відмінними від значень за замовчуванням без явного виклику відповідного конструктора суперкласу є неможливою. Якщо виклик конструктора суперкласу не вказаний явно у підкласі або суперклас не має конструкторів, тоді автоматично викликається конструктор за замовчуванням суперкласу.

Механізм поліморфізму забезпечує можливість присвоєння об’єктним змінним суперкласу об’єктів похідних класів та звертання з-під цих змінних до перевизначених у підкласі членів суперкласу. У Java всі об’єктні змінні є поліморфними. Поліморфізм реалізується за допомогою механізму динамічного (пізнього) зв’язування, який полягає у тому, що вибір методу, який необхідно викликати, відбувається не на етапі компіляції, а під час виконання програми.

**Завдання (Варіант №15)**

1. Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі №2, для реалізації предметної області заданої варіантом. Суперклас, що реалізований у лабораторній роботі №2, зробити абстрактним. Розроблений підклас має забезпечувати механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab3 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленого пакету.
3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub
4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.
5. Дати відповідь на контрольні запитання.

**Виконання завдання**

**FrigateDriver.java**

package 303.Sukharnyk.Lab3;  
  
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 FrigateDriver {  
 */\*\*  
 \* Точка входу в програму. Демонструє роботу з телевізором шляхом виклику різних методів.  
 \*  
 \* @param args Аргументи командного рядка.  
 \*/* public static void main(String[] args) {  
 try {  
 Frigate frigate = new Frigate("Грізний");  
  
 frigate.startEngine();  
 frigate.changeCourse(45);  
 frigate.changeSpeed(30);  
 frigate.checkHullStatus();  
 frigate.updateNavigationSystem();  
 frigate.refuel(200);  
 frigate.performMaintenance();  
  
 frigate.activateWeaponSystems();  
 frigate.fireWeapon();  
 frigate.activateDefenseSystems();  
  
 frigate.performSpecificAction();  
  
 frigate.calculateArrivalTime(100);  
 frigate.simulateWeatherConditions("шторм");  
 frigate.stopEngine();  
  
 frigate.closeLogger();  
 } catch (IOException e) {  
 // Обробка помилок, що виникають під час запису в файл  
 throw new RuntimeException("Сталася помилка при записі в файл: " + e.getMessage());  
 }  
 }  
}

**Frigate.java**

package 303.Sukharnyk.Lab3;  
  
import java.io.IOException;  
  
*/\*\*  
 \* Клас Frigate представляє фрегат - тип військового корабля,  
 \* що може відкривати вогонь та активувати оборонні системи.  
 \*/*public class Frigate extends Ship implements WarShip {  
 private boolean weaponSystemActive;  
 private boolean defensiveSystemActive;  
  
 */\*\*  
 \* Конструктор для створення фрегата.  
 \* @param name Ім'я фрегата  
 \* @throws IOException якщо виникає помилка під час створення логера  
 \*/* public Frigate(String name) throws IOException {  
 super(name);  
 this.weaponSystemActive = false;  
 this.defensiveSystemActive = false;  
 }  
  
 */\*\*  
 \* Виконує постріл, якщо збройні системи активовані.  
 \*  
 \* @throws IOException якщо виникає помилка під час стрілянини  
 \*/* @Override  
 public void fireWeapon() throws IOException {  
 if (weaponSystemActive) {  
 logger.log(String.format("Фрегат %s відкрив вогонь.", name));  
 System.out.printf("Фрегат %s відкрив вогонь.\n", name);  
 } else {  
 logger.log(String.format("Не вдалося відкрити вогонь. Зброя фрегата %s не активована.", name));  
 System.out.printf("Не вдалося відкрити вогонь. Зброя фрегата %s не активована.\n", name);  
 }  
 }  
  
 */\*\*  
 \* Активує оборонні системи фрегата.  
 \*  
 \* @throws IOException якщо виникає помилка під час активації систем  
 \*/* @Override  
 public void activateDefenseSystems() throws IOException {  
 defensiveSystemActive = true;  
 logger.log(String.format("Активовано оборонні системи фрегата %s.", name));  
 System.out.printf("Активовано оборонні системи фрегата %s.\n", name);  
 }  
  
 */\*\*  
 \* Виконує специфічну дію для фрегата - патрулювання.  
 \*  
 \* @throws IOException якщо виникає помилка під час виконання дії  
 \*/* @Override  
 public void performSpecificAction() throws IOException {  
 logger.log(String.format("Фрегат %s виконує патрулювання.", name));  
 System.out.printf("Фрегат %s виконує патрулювання.\n", name);  
 }  
  
 */\*\*  
 \* Метод для активації збройних систем фрегата.  
 \*  
 \* @throws IOException якщо виникає помилка під час активації систем  
 \*/* public void activateWeaponSystems() throws IOException {  
 weaponSystemActive = true;  
 logger.log(String.format("Активовано збройні системи фрегата %s.", name));  
 System.out.printf("Активовано збройні системи фрегата %s.\n", name);  
 }  
}

**WarShip.java**

package 303.Sukharnyk.Lab3;  
  
import java.io.IOException;  
  
*/\*\*  
 \* Інтерфейс WarShip визначає базову функціональність військових кораблів,  
 \* включаючи стрілянину з озброєння та активацію систем захисту.  
 \*/*public interface WarShip {  
  
 */\*\*  
 \* Виконує постріл з озброєння корабля.  
 \*  
 \* @throws IOException якщо виникає помилка під час стрілянини  
 \*/* void fireWeapon() throws IOException;  
  
 */\*\*  
 \* Активує системи захисту корабля.  
 \*  
 \* @throws IOException якщо виникає помилка під час активації систем захисту  
 \*/* void activateDefenseSystems() throws IOException;  
}

**Ship.java**

package 303.Sukharnyk.Lab3;  
  
import java.io.IOException;  
import java.util.Random;  
*/\*\*  
 \* Клас, що представляє корабель з двигуном, корпусом та навігаційною системою.  
 \* Забезпечує управління основними функціями корабля.  
 \*/*public abstract class Ship {  
 protected Engine engine;  
 protected Hull hull;  
 protected NavigationSystem navigationSystem;  
 protected Logger logger;  
 protected 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()));  
 }  
  
 */\*\*  
 \* Абстрактний метод для виконання специфічних дій корабля.  
 \* @throws IOException якщо виникає помилка під час запису до логера  
 \*/* public abstract void performSpecificAction() throws IOException;  
  
 */\*\*  
 \* 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.Lab3;  
  
*/\*\*  
 \* 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.Lab3;  
  
*/\*\*  
 \* 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.Lab3;  
  
*/\*\*  
 \* 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.Lab3;  
  
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 реалізації спадкування. *class Subclass extends Superclass { }*
2. Що таке суперклас та підклас? *Суперклас — батьківський клас; підклас — клас, який успадковує від суперкласу.*
3. Як звернутися до членів суперкласу з підкласу? *Використовують ключове слово super*
4. Коли використовується статичне зв’язування при виклику методу? *Використовується для статичних методів або методів, які не переозначені в підкласі (компіляція на етапі компіляції).*
5. Як відбувається динамічне зв’язування при виклику методу? *Відбувається під час виконання програми (runtime), коли викликаються методи через поліморфізм.*
6. Що таке абстрактний клас та як його реалізувати? *Клас, який не можна створювати як об'єкт і містить абстрактні методи без реалізації:  
   abstract class ClassName { abstract void methodName(); }*
7. Для чого використовується ключове слово instanceof? *Використовується для перевірки, чи є об'єкт екземпляром певного класу або його підкласу:  
   object instanceof ClassName.*
8. Як перевірити чи клас є підкласом іншого класу? *Використовується метод isAssignableFrom Superclass.class.isAssignableFrom(Subclass.class)*
9. Що таке інтерфейс? *Абстрактний тип, що містить тільки оголошення методів, без їх реалізації.*
10. Як оголосити та застосувати інтерфейс?

*Оголошення:  
interface InterfaceName { void methodName(); }  
Застосування:  
class ClassName implements InterfaceName { void methodName() { // реалізація } }*

**Висновок:** Я ознайомилась з спадкуванням та інтерфейсами у мові Java.