Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра “Електронних обчислювальних машин”



**з дисципліни**

**Кросплатформні засоби програмування**

**Звіт з Лабораторної роботи № 3**

**Спадкування та інтерфейси**

**Виконав:**

КІ-307

Герега Р. О.

**Перевірив:**

Іванов Ю.С

**2023**

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

**Теоретичний вступ**

**Спадкування**

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.

**Поліморфізм**

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. Для глибшого розуміння поліморфізму розглянемо покроково виклик методу класу:

1. Компілятор визначає об’явлений тип об’єкту і ім’я методу та нумерує всі методи з однаковою назвою у класі та всі загальнодоступні методи з такою ж назвою у його суперкласах.

2. Компілятор визначає типи параметрів, що вказані при виклику методу. Якщо серед усіх методів з вказаним іменем є лише один метод, типи параметрів якого співпадають з вказаним, то відбувається його виклик. Цей процес називається дозволом перевантаження. Якщо компілятор не знаходить жодного методу з підходящим набором параметрів або в результаті перетворення типів виявлено кілька методів, що відповідають даному виклику, то видається повідомлення про помилку.

3. Якщо метод є приватним, статичним, фінальним або конструктором, то для нього застосовується механізм статичного зв’язування. Механізм статичного зв’язування передбачає визначення методу, який необхідно викликати, на етапі компіляції. В протилежному випадку метод, що необхідно викликати, визначається по фактичному типу неявного параметру (динамічне зв’язування).

4. Якщо для виклику методу використовується динамічне зв’язування, то віртуальна машина повинна викликати версію методу, що відповідає фактичному типу об’єкту на який посилається об’єктна змінна.

Оскільки на пошук необхідного методу потрібно багато часу, то віртуальна машина заздалегідь створює для кожного класу таблицю методів, в якій перелічуються сигнатури всіх методів і фактичні методи, що підлягають виклику. При виклику методу віртуальна машина просто переглядає таблицю методів. Якщо відбувається виклик методу з суперкласу за допомогою ключового слова super, то при виклику методу переглядається таблиця методів суперкласу неявного параметру.

**Абстрактні класи**

Абстрактні класи призначені бути основою для розробки ієрархій класів та не дозволяють створювати об’єкти свого класу. Вони реалізуються за допомогою ключового слова abstract. На відміну від звичайних класів абстрактні класи можуть містити абстрактні методи (а можуть і не містити). Абстрактні методи – це методи, що оголошені з використанням ключового слова abstract і не місять тіла. Розширюючи абстрактний клас можна залишити деякі або всі методи невизначеними. При цьому підклас автоматично стане абстрактним. Перевизначення у підкласі усіх абстрактних методів призведе до того, що підклас не буде абстрактним, що дозволить створювати на його основі об’єкти класу. Синтаксис оголошення абстрактного класу наведено в пункті «Класи та об’єкти».

**Інтерфейси**

Інтерфейси вказують що повинен робити клас не вказуючи як саме він це повинен робити. Інтерфейси покликані компенсувати відсутність множинного спадкування у мові Java та гарантують визначення у класах оголошених у собі прототипів методів.

**Варіант 6**



**Завдання:**

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.

**Код**

/\*\*

\* lab 3 package **Plane.java**

\*/

package KI307.Hereha.Lab3;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Date;

/\*\*

\* Клас що реалізує предметну область Літаку у вигляді класу.

\*/

public abstract class Plane {

// Поля класу, що описують складові частини літака

private String model;

private Engine engine;

private Pilot pilot;

private AircraftFactory factory;

private int passengerCapacity;

// Конструктори

public Plane(String model, Engine engine, Pilot pilot, AircraftFactory factory, int passengerCapacity) {

this.model = model;

this.engine = engine;

this.pilot = pilot;

this.factory = factory;

this.passengerCapacity = passengerCapacity;

}

public Plane(String model, String engineType,

int enginePower, int passengerCapacity,

String firstName, String lastName, int flightCount,

String country, String state, String name

)

{

this.model = model;

this.engine = new Engine(engineType, enginePower);

this.pilot = new Pilot(firstName, lastName, flightCount);

this.factory = new AircraftFactory(country, state, name);

this.passengerCapacity = passengerCapacity;

}

// Методи для доступу до полів класу

public String getModel() {

return model;

}

// Метод класу, що відповідає за отримання типу двигуна

public Engine getEngine() {

return engine;

}

// Метод класу, що відповідає за отримання пілота

public Pilot getPilot() {

return pilot;

}

// Метод класу, що відповідає за отримання авіазаводу

public AircraftFactory getFactory() {

return factory;

}

// Метод класу, що відповідає за отримання кількості пасажирів

public int getPassengerCapacity() {

return passengerCapacity;

}

// Метод для запису протоколу в файл

public void logActivity(String message) {

try (FileWriter writer = new FileWriter("log.txt", true)) {

writer.write(new Date() + ": " + message + "\n");

} catch (IOException e) {

e.printStackTrace();

}

}

// Інші методи класу, що відповідають предметній області

public abstract void takeOff();

// Метод класу, що відповідає за логування посадки

public void land() {

logActivity("Літак " + model + " приземлився.");

}

// Метод класу, що відповідає за логування додавання кількості пасажирів

public void addPassengers(int count) {

passengerCapacity += count;

logActivity("До літака " + model + " додано " + count + " пасажирів.");

}

// Інші методи

// Методи, які демонструють роботу класу

public void displayInfo() {

System.out.println("Літак: " + model);

System.out.println("Двигун: " + engine.getType() + ", Потужність: " + engine.getPower());

System.out.println("Пасажиромісткість: " + passengerCapacity);

System.out.println("Пілот: " + pilot.getFullName());

System.out.println("Створено: " + factory.getDataAboutFactory());

}

// Внутрішній клас для представлення двигуна

class Engine {

private String type;

private int power;

public Engine(String type, int power) {

this.type = type;

this.power = power;

}

public String getType() {

return type;

}

public int getPower() {

return power;

}

}

// Внутрішній клас для представлення пілота

class Pilot {

private String firstName;

private String lastName;

private int flightCount;

public Pilot(String firstName, String lastName, int flightCount) {

this.firstName = firstName;

this.lastName = lastName;

this.flightCount = flightCount;

}

public String getFullName() {

return firstName + " " + lastName;

}

public int getExperience() {

return flightCount;

}

}

// Внутрішній клас для представлення авіазаводу

class AircraftFactory{

private String country;

private String state;

private String name;

public AircraftFactory(String country, String state, String name) {

this.country = country;

this.state = state;

this.name = name;

}

public String getDataAboutFactory() {

return country + ", " + state + ", " + name;

}

}

}

/\*\*

\* lab 3 package **PlaneDriver.java**

\*/

package KI307.Hereha.Lab3;

/\*\*

\* Клас драйвер для демонстрації роботи класу літака

\*/

public class PlaneDriver {

public static void main(String[] args) {

BomberPlane plane1 = new BomberPlane("F-16", "Jet", 10000, 2, "Rostyslav", "Hereha", 100, "UA", "Kharkiv", "\r\n"

+ "Kharkiv Aviation Plant", 4);

plane1.displayInfo();

plane1.takeOff();

plane1.dropBombs();

plane1.dropBombs();

plane1.dropBombs();

plane1.dropBombs();

plane1.dropBombs();

plane1.land();

}

}

/\*\*

\* lab 3 package **BomberPlane.java**

\*/

package KI307.Hereha.Lab3;

public class BomberPlane extends Plane implements Bomber {

private int bombCapacity;

private Plane.Engine engine;

private Plane.Pilot pilot;

private Plane.AircraftFactory factory;

public BomberPlane(String model, String engineType,

int enginePower, int passengerCapacity,

String firstName, String lastName, int flightCount,

String country, String state, String name, int bombCapacity) {

super(model, engineType, enginePower, passengerCapacity,

firstName, lastName, flightCount, country, state, name);

this.bombCapacity = bombCapacity;

this.engine = super.getEngine(); // Отримуємо екземпляр Engine

this.pilot = super.getPilot(); // Отримуємо екземпляр Pilot

this.factory = super.getFactory(); // Отримуємо екземпляр AircraftFactory

}

@Override

public void takeOff() {

logActivity("Бомбардувальний літак " + getModel() + " вилетів.");

}

@Override

public void land() {

logActivity("Бомбардувальний літак " + getModel() + " приземлився.");

}

@Override

public void displayInfo() {

System.out.println("Бомбардувальний літак: " + getModel());

System.out.println("Двигун: " + engine.getType() + ", Потужність: " + engine.getPower());

System.out.println("Пасажиромісткість: " + getPassengerCapacity());

System.out.println("Пілот: " + pilot.getFullName());

System.out.println("Створено: " + factory.getDataAboutFactory());

System.out.println("Кількість бомб: " + bombCapacity);

}

@Override

public void dropBombs() {

if(bombCapacity != 0) {

logActivity("Бомбардувальний літак " + getModel() + " скинув бомбу. Залишилось " + bombCapacity + " бомб." );

bombCapacity--;

} else {

logActivity("Бомбардувальний літак " + getModel() + " скинув всі бомби. Потрібна перезарядка." );

}

}

}

/\*\*

\* lab 3 package **Bomber.java**

\*/

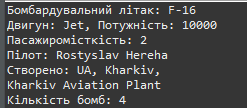
package KI307.Hereha.Lab3;

public interface Bomber {

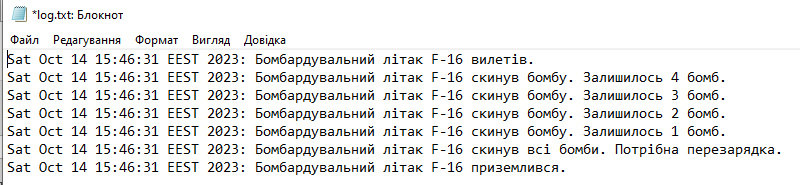
void dropBombs();

}

**Результат:**



**log.txt:**



**Короткий висновок**

Під час виконання даної лабораторної роботи я ознайомився з спадкуванням та інтерфейсами у мові Java.