Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ



**Звіт**

до лабораторної роботи №3

З дисципліни: «Кросплатформні засоби програмування»

На тему: «Спадкування та інтерфейси»

**Варіант 5**

Виконала:

ст. гр. КІ-305

Гринь С.М.

Прийняв:

Іванов Ю.С.

Львів 2023

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

**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.

**Варіант 5:** *Вантажна машина*

**Код програми:**

ClassAutomobile:

package KI305.Hryn.Lab3;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

/\*\*

\* Клас, що представляє автомобіль та зберігає інформацію про нього.

\*/

public abstract class Automobile {

//Поля класу

private Company company;

private Model model;

private Price price;

//Поле для запису протоколу

private PrintWriter logWriter;

/\*\*

\* Пустий конструктор без аргументів.

\* Ініціалізує об'єкт Automobile та створює файл протоколу.

\*/

//Пустий конструктор без аргументів

public Automobile() {

try {

logWriter = new PrintWriter(new FileWriter("C:\\Users\\User\\IdeaProjects\\java\_lab\\src\\KI305\\Hryn\\Lab3\\Automobile.txt"));

} catch (IOException e) {

e.printStackTrace();

}

}

//Конструктор зі всіма аргументами

public Automobile(Company company, Model model, Price price) {

this.company = company;

this.model = model;

this.price = price;

try {

logWriter = new PrintWriter(new FileWriter("C:\\Users\\User\\IdeaProjects\\java\_lab\\src\\KI305\\Hryn\\Lab3\\Automobile.txt"));

} catch (IOException e) {

e.printStackTrace();

}

}

//Абстрактні методи класу

public abstract void moveObject();

public abstract void stopObject();

//Гетери і сетери для об'єктів класу

public Company getCompany() {

return company;

}

public void setCompany(Company company) {

this.company = company;

}

public Model getModel() {

return model;

}

public void setModel(Model model) {

this.model = model;

}

public Price getPrice() {

return price;

}

public void setPrice(Price price) {

this.price = price;

}

/\*\*

\* Метод для запису дій в файл протоколу.

\*

\* @param activity Дія, яку потрібно зареєструвати в протоколі.

\*/

private void logActivity(String activity) {

if (logWriter != null) {

logWriter.println(activity);

logWriter.flush();

}

}

//Методи для роботи з автомобілем

public void startEngine() {

System.out.println("Starting the car engine.");

logActivity("We started the car engine.");//Запис у протокол

}

public void brake() {

System.out.println("Turning on the brakes.");

logActivity("We turned on the brakes.");//Запис у протокол

}

public void turnLeft() {

System.out.println("Turning to the left.");

logActivity("We turned to the left.");//Запис у протокол

}

public void turnRight() {

System.out.println("Turning to the right.");

logActivity("We turned to the right.");//Запис у протокол

}

public void checkEngineStatus() {

System.out.println("Сhecking the condition of the engine.");

logActivity("We checked the condition of the engine.");//Запис у протокол

}

public void turnOnHeadlights() {

System.out.println("Turning on the headlights.");

logActivity("We turned on the headlights.");//Запис у протокол

}

public void checkFuelLevel() {

System.out.println("Сhecking the level of the fuel tank in the car.");

logActivity("We checked the level of the fuel tank in the car.");//Запис у протокол

}

public void lockDoors() {

System.out.println("Locking the doors.");

logActivity("We locked the doors.");//Запис у протокол

}

public void stopEngine() {

System.out.println("Stopping the car engine.");

logActivity("We stopped the car engine.");//Запис у протокол

closeLogFile();//Закриваємо файл при завершенні роботи

}

// Метод для закриття файлу протоколу

private void closeLogFile() {

if (logWriter != null) {

logWriter.close();

}

}

//Метод toString()

@Override

public String toString() {

return "Automobile: Company : " + company + ", Model : " + model + ", Price : " + price;

}

}

ClassCompany:

package KI305.Hryn.Lab3;

/\*\*

\* Клас, що представляє інформацію про компанію.

\*/

public class Company {

// Поля класу

private String nameCompany; // Назва компанії

private String infoCompany; // Інформація про компанію

/\*\*

\* Пустий конструктор без аргументів.

\* Ініціалізує об'єкт Company з пустими значеннями.

\*/

public Company(){

}

/\*\*

\* Конструктор з усіма аргументами.

\* Ініціалізує об'єкт Company з вказаними параметрами.

\*

\* @param nameCompany Назва компанії.

\* @param infoCompany Інформація про компанію.

\*/

public Company(String nameCompany, String infoCompany){

this.nameCompany = nameCompany;

this.infoCompany = infoCompany;

}

/\*\*

\* Перевизначений метод toString().

\*

\* @return Рядок, що представляє об'єкт Company.

\*/

@Override

public String toString() {

return "Company{" +

"name='" + nameCompany + '\'' +

", info='" + infoCompany + '\'' +

'}';

}

}

Class ***Model***:

package KI305.Hryn.Lab3;

/\*\*

\* Клас, що представляє модель автомобіля.

\*/

public class Model {

// Поле класу

private String nameModel; // Назва моделі автомобіля

/\*\*

\* Пустий конструктор без аргументів.

\* Ініціалізує об'єкт Model з пустою назвою моделі.

\*/

public Model(){

}

/\*\*

\* Конструктор з усіма аргументами.

\* Ініціалізує об'єкт Model з вказаною назвою моделі.

\*

\* @param nameModel Назва моделі автомобіля.

\*/

public Model(String nameModel){

this.nameModel = nameModel;

}

/\*\*

\* Перевизначений метод toString().

\*

\* @return Рядок, що представляє об'єкт Model.

\*/

@Override

public String toString() {

return "Model{" +

"name:'" + nameModel + '\'' +

'}';

}

}

ClassPrice:

package KI305.Hryn.Lab3;

/\*\*

\* Клас, що представляє ціну.

\*/

public class Price {

// Поле класу

private int sum; // Сума ціни

/\*\*

\* Пустий конструктор без аргументів.

\* Ініціалізує об'єкт Price з нульовою сумою.

\*/

public Price() {

}

/\*\*

\* Конструктор з усіма аргументами.

\* Ініціалізує об'єкт Price з вказаною сумою ціни.

\*

\* @param sum Сума ціни.

\*/

public Price(int sum) {

this.sum = sum;

}

/\*\*

\* Перевизначений метод toString().

\*

\* @return Рядок, що представляє об'єкт Price.

\*/

@Override

public String toString() {

return "Price{" + "sum=" + sum + "}";

}

}

ClassTruck:

package KI305.Hryn.Lab3;

/\*\*

\* Клас, який представляє вантажний автомобіль.

\* Вантажний автомобіль є розширенням класу Automobile та реалізує інтерфейс ILights для керування світлами.

\*/

public class Truck extends Automobile implements ILights {

private boolean lights;

/\*\*

\* Конструктор без параметрів для вантажного автомобіля.

\* Створює вантажний автомобіль з вимкнутими світлами за замовчуванням.

\*/

public Truck(){

super();

lights = false;

}

public Truck(Company company, Model model, Price price) {

super(company,model,price);

}

/\*\*

\* Перевизначений метод для переміщення вантажного автомобіля.

\*/

@Override

public void moveObject() {

System.out.println("Object is moving");

}

/\*\*

\* Перевизначений метод для зупинки вантажного автомобіля.

\*/

@Override

public void stopObject() {

System.out.println("Object stopped");

}

@Override

public void setLights(boolean data) {

}

/\*\*

\* Реалізація методу інтерфейсу ILights для виведення інформації про стан фар.

\*/

@Override

public void lightInfo() {

if (lights)

System.out.println("Lights is on");

else

System.out.println("Lights is off");

}

public void setLight(boolean data) {

this.lights = data;

}

}

Interface ILights:

package KI305.Hryn.Lab3;

/\*\*

\* Інтерфейс, що представляє функціональність керування світлами.

\* Реалізуючі класи повинні реалізовувати методи цього інтерфейсу для управління світлами і надавати інформацію про їх стан.

\*/

public interface ILights {

/\*\*

\* Встановлює стан світла (увімкнено або вимкнено).

\*

\* @param data true, якщо світло має бути увімкнено, false, якщо світло має бути вимкнено.

\*/

void setLights(boolean data);

/\*\*

\* Виводить інформацію про поточний стан світла.

\*/

void lightInfo();

}

ClassAutomobileDriver:

package KI305.Hryn.Lab3;

/\*\*

\* Головний клас, який представляє програму для водія автомобіля.

\*/

public class AutomobileDriver {

/\*\*

\* Головний метод програми.

\*

\* @param args Масив рядків аргументів командного рядка.

\*/

public static void main(String[] args) {

// Ініціалізація полів

Truck tr = new Truck(new Company("Volvo", "volvo is the best company"), new Model("MD-233"), new Price(2233));

System.out.println(tr);

tr.startEngine();

tr.checkEngineStatus();

tr.lightInfo();

tr.setLight(true);

tr.lightInfo();

tr.moveObject();

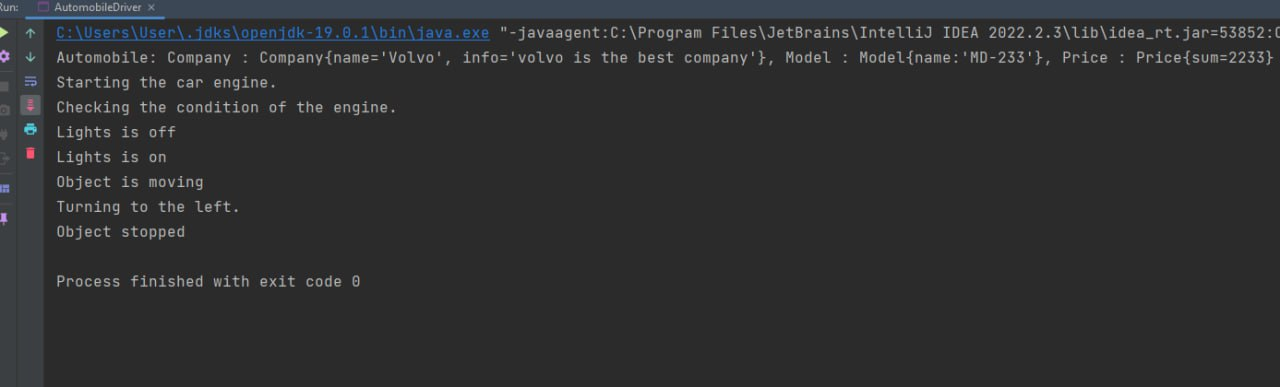
tr.turnLeft();

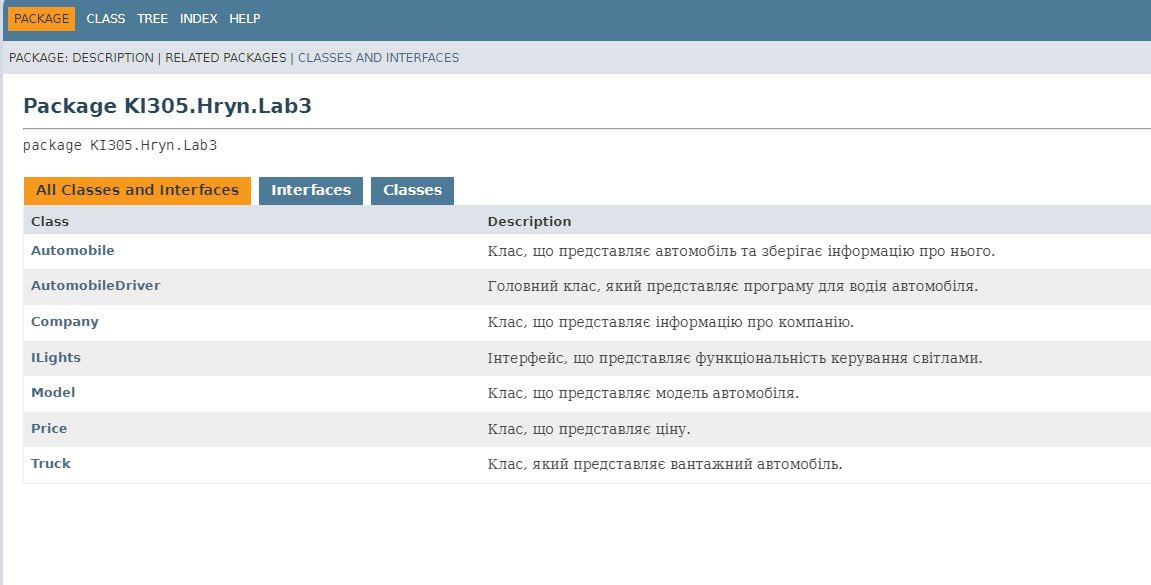
tr.stopObject();

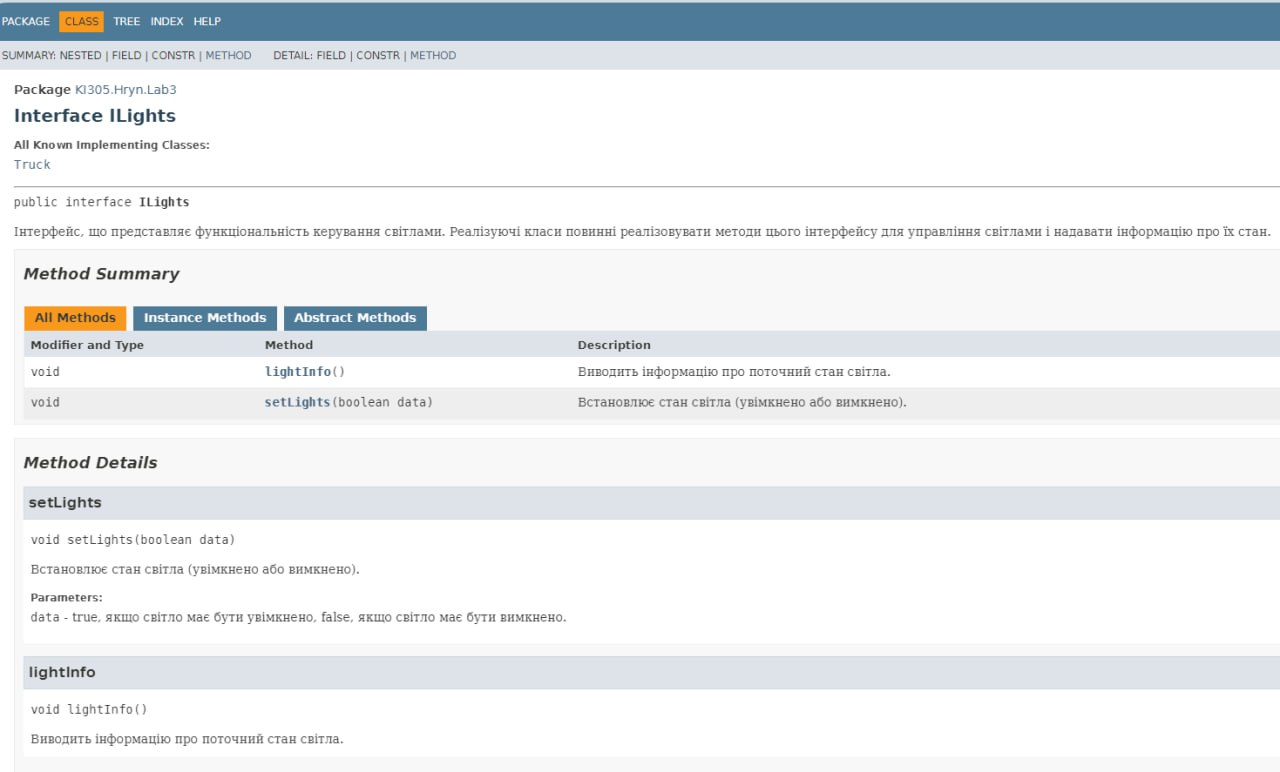
}

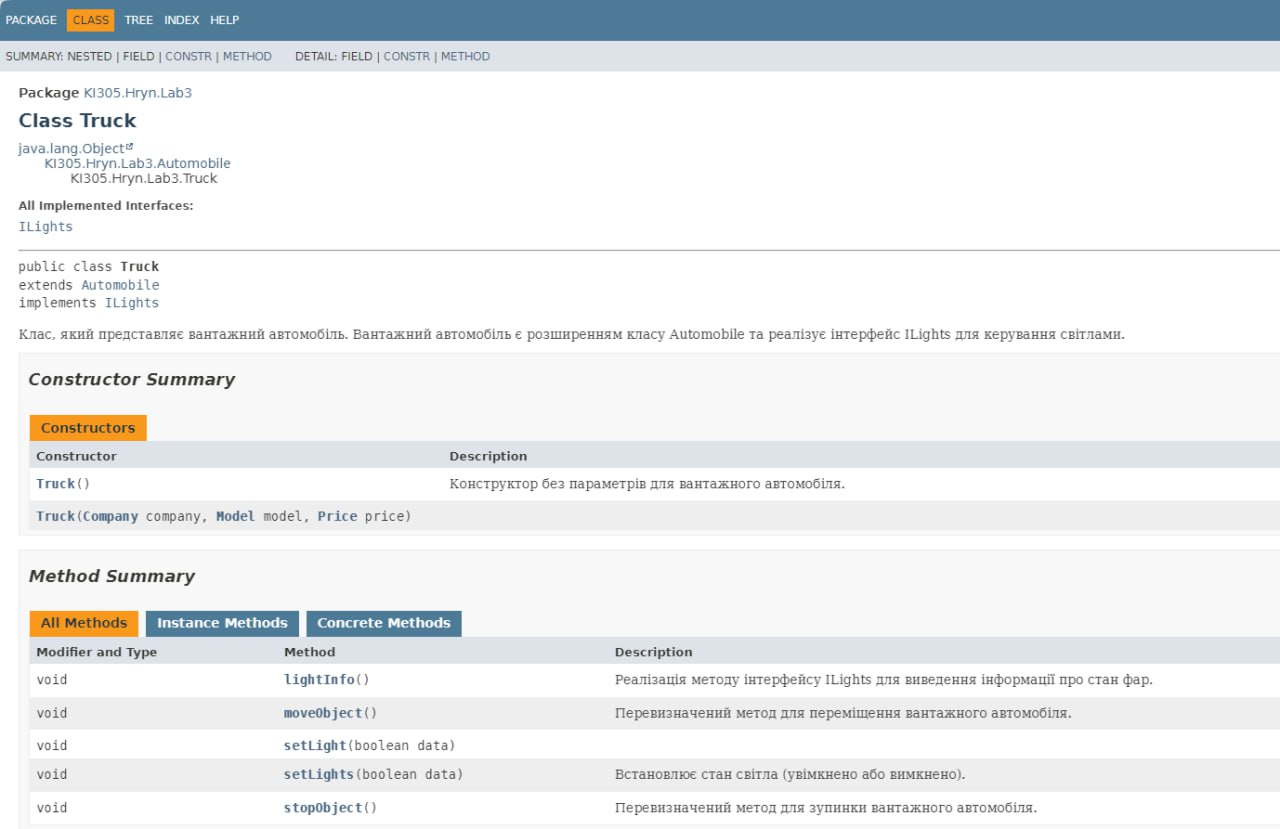
}

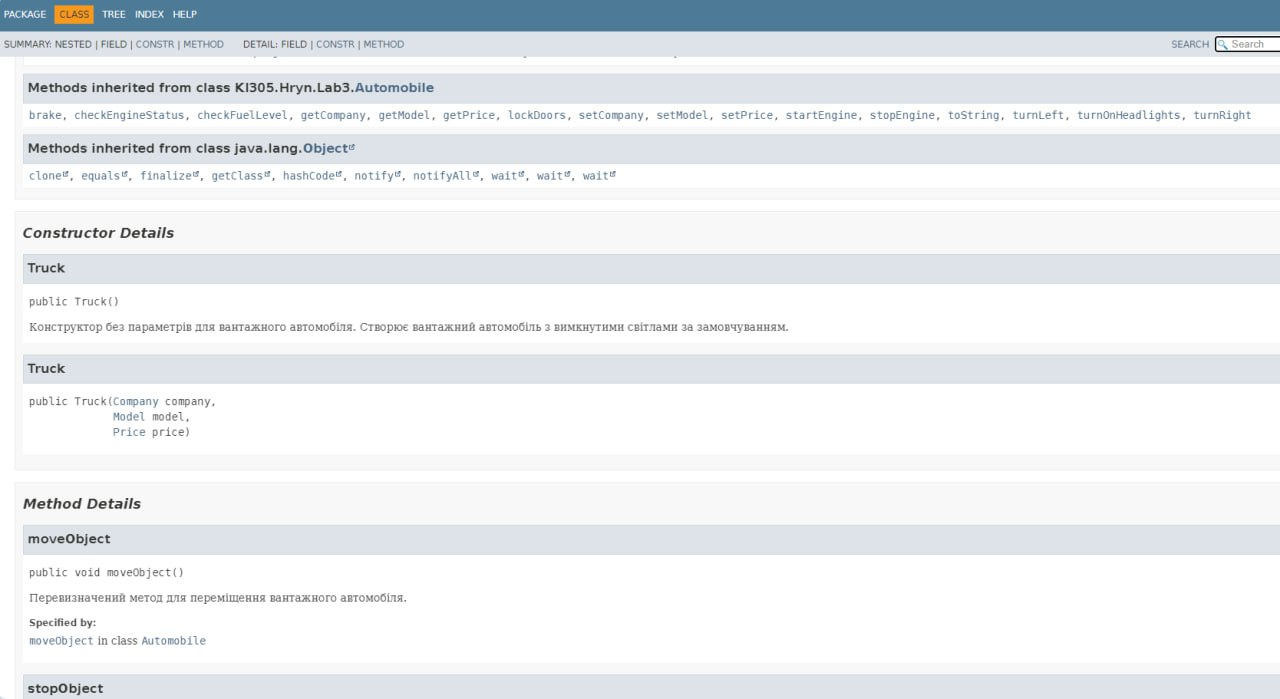
**Результат роботи програми:**

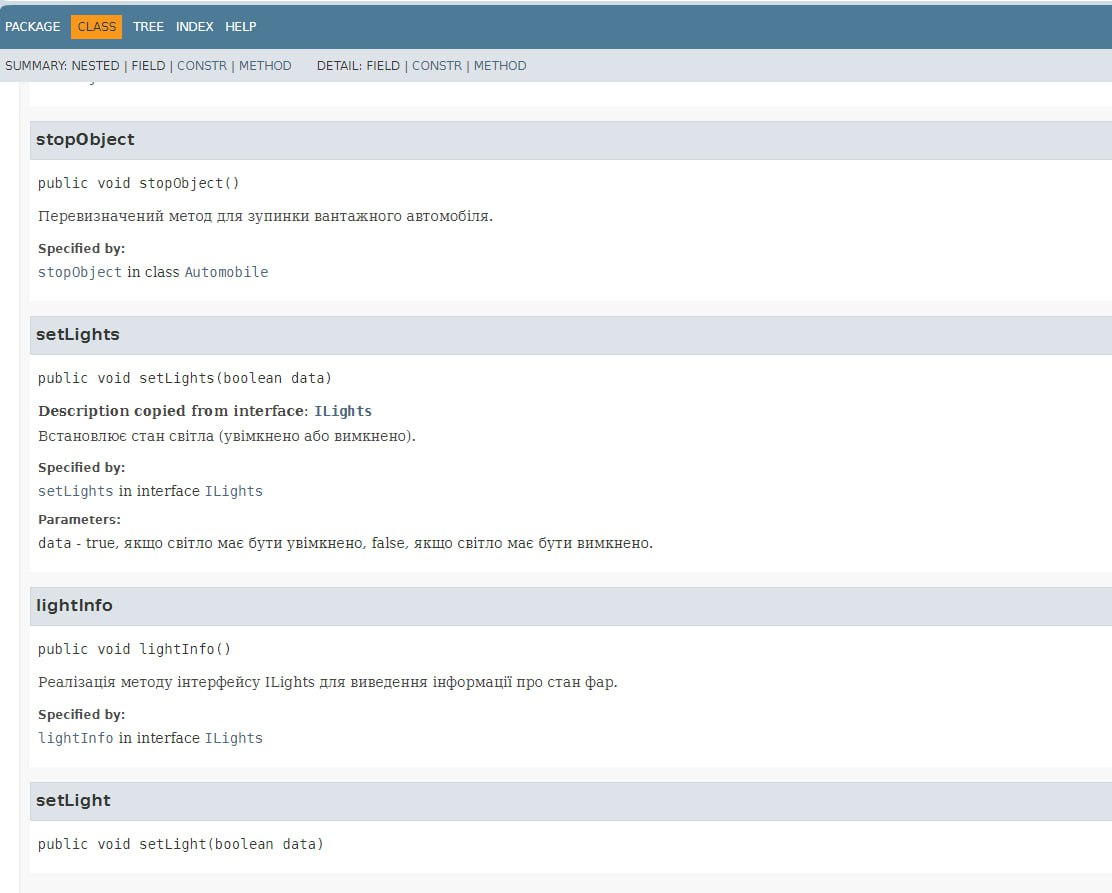


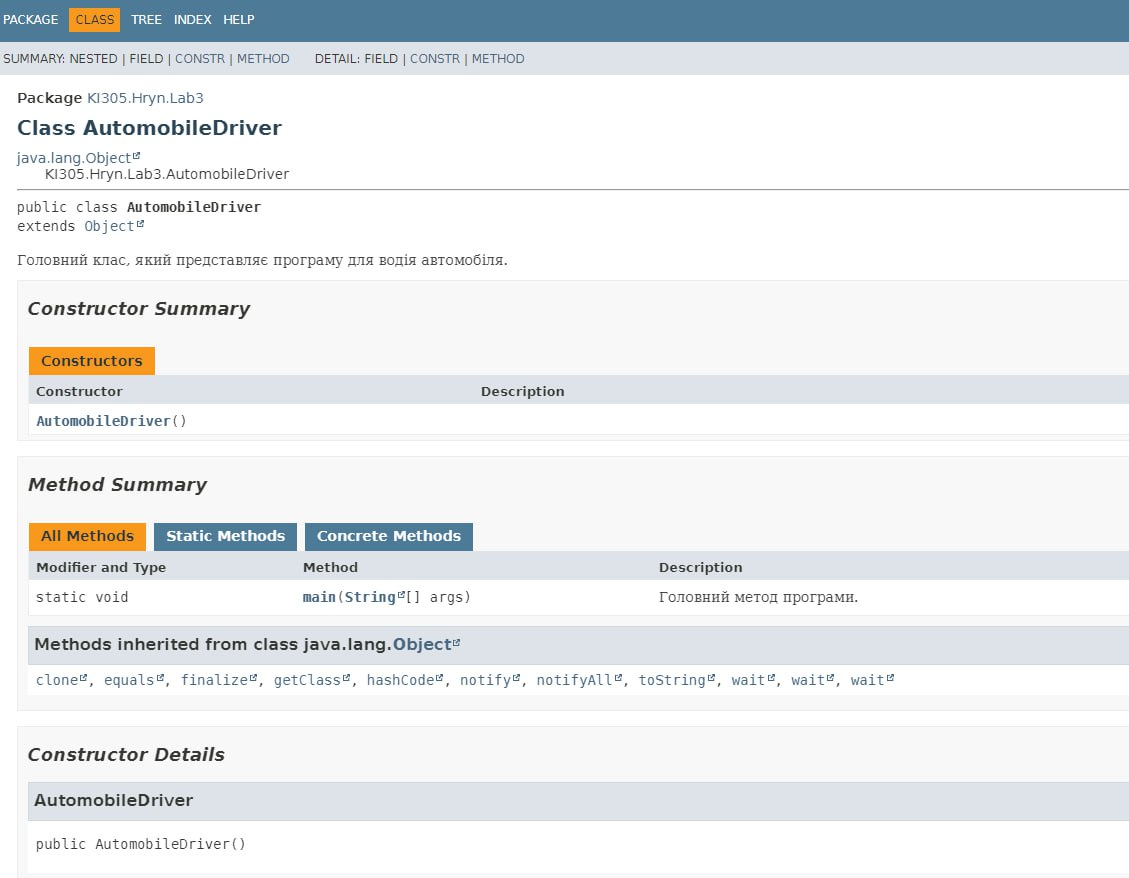


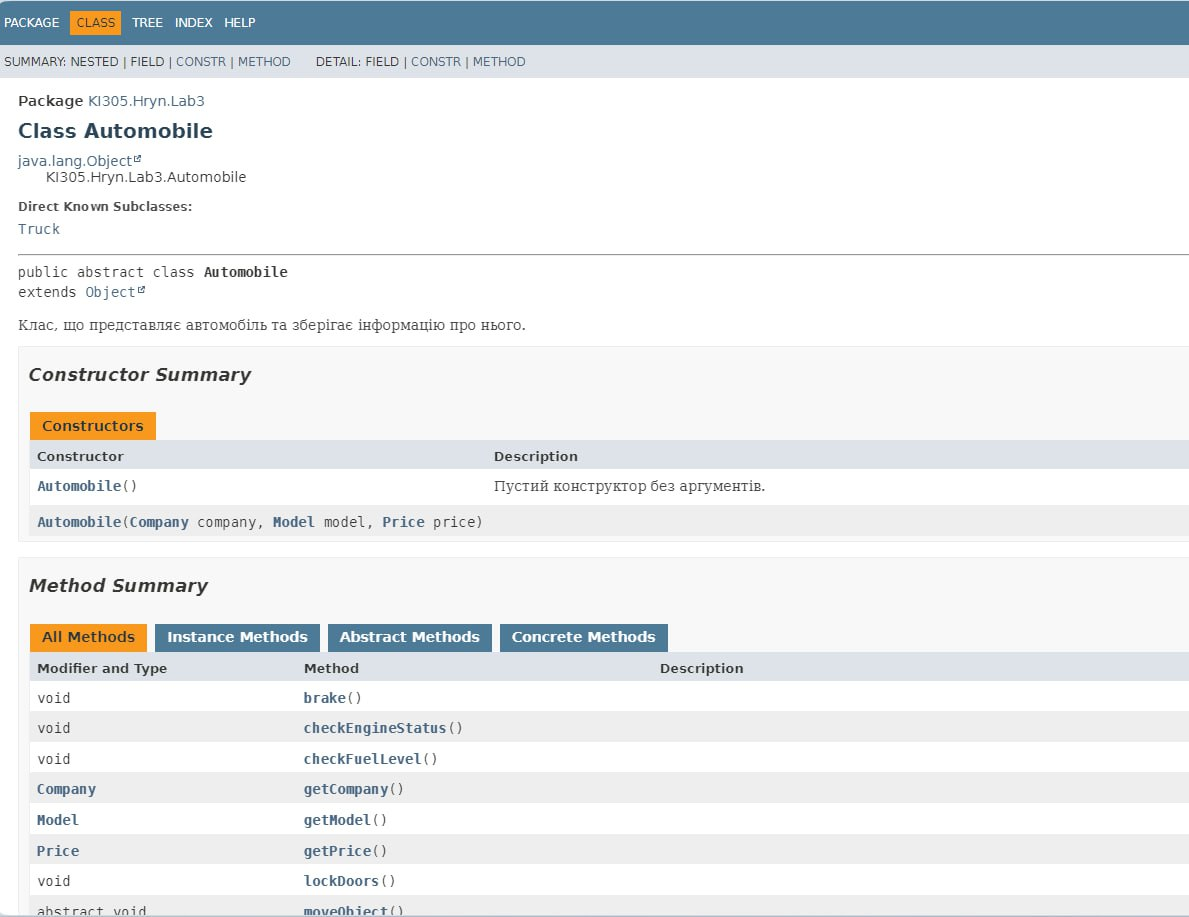


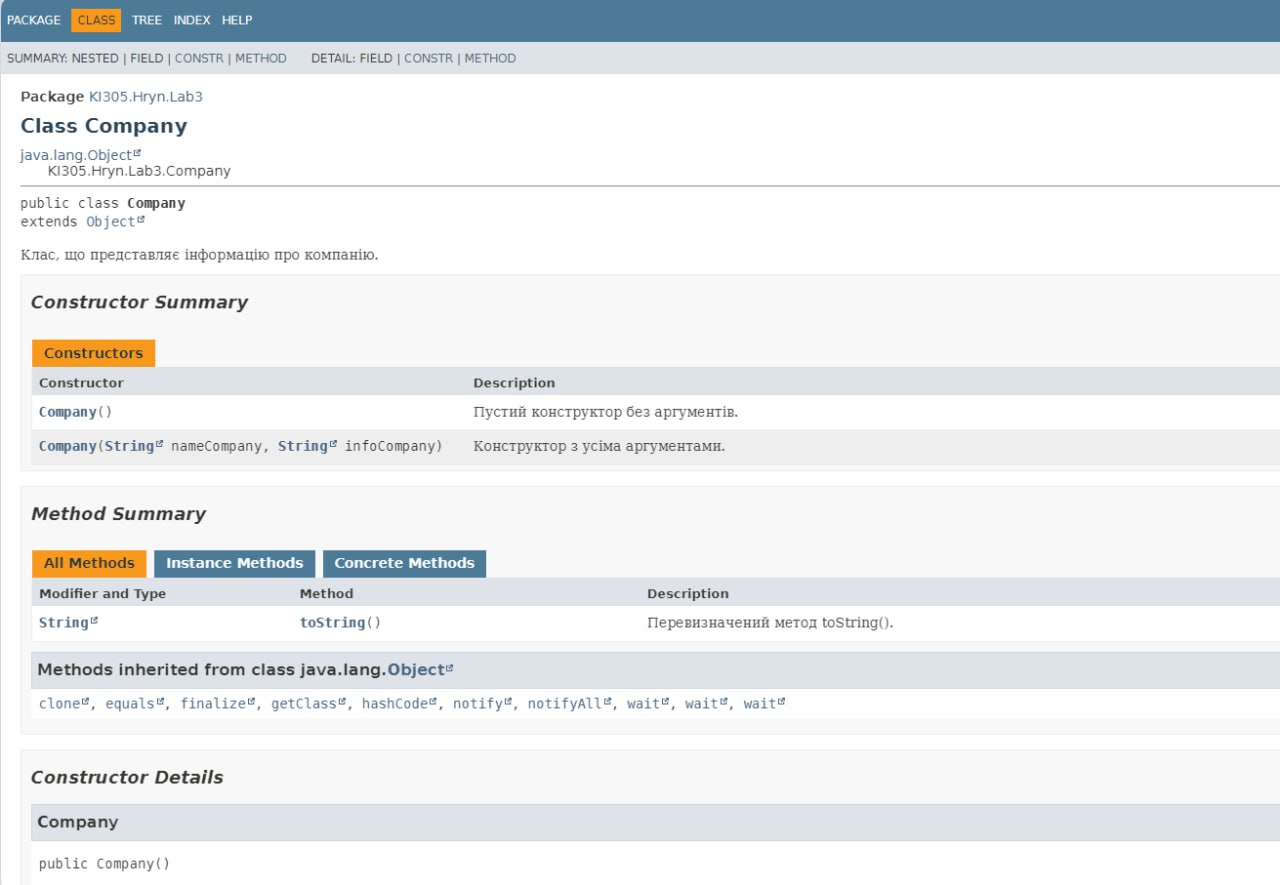


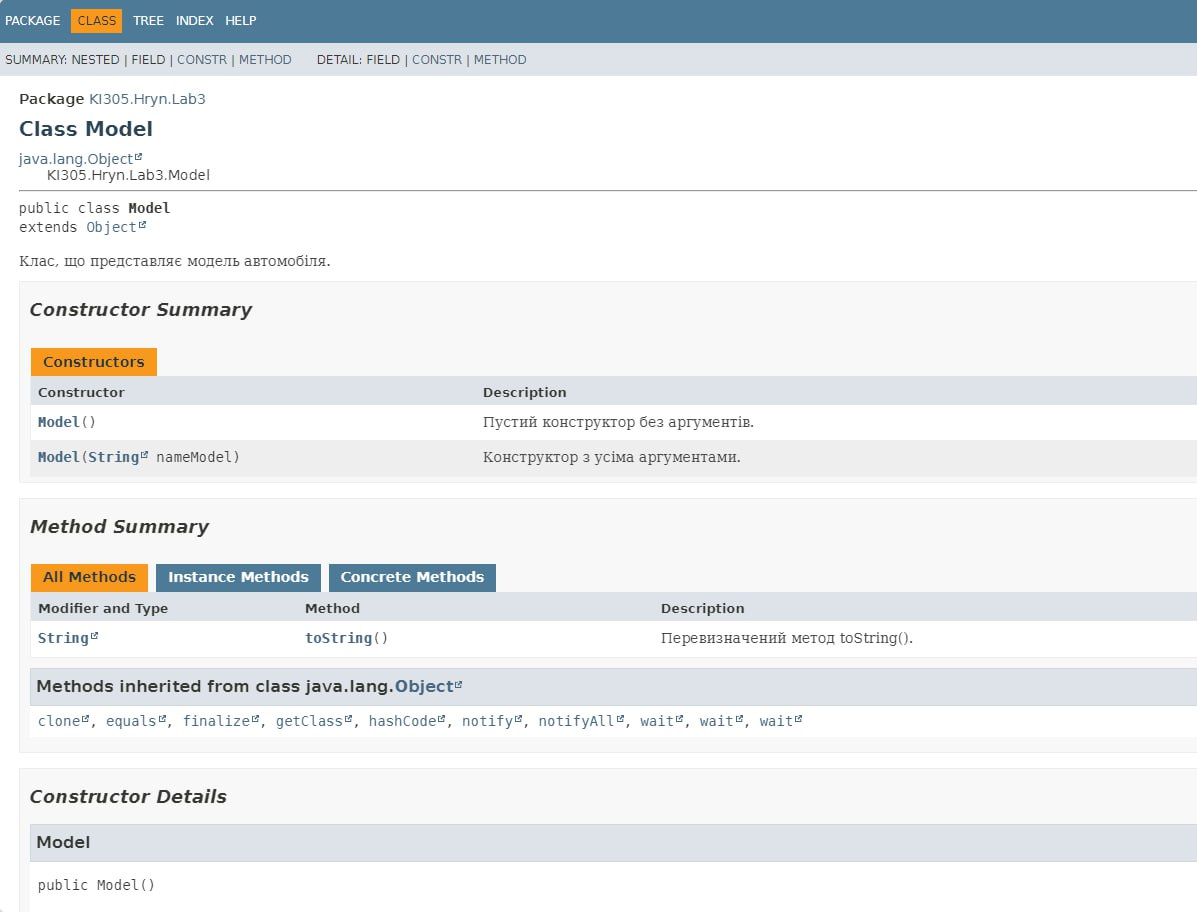


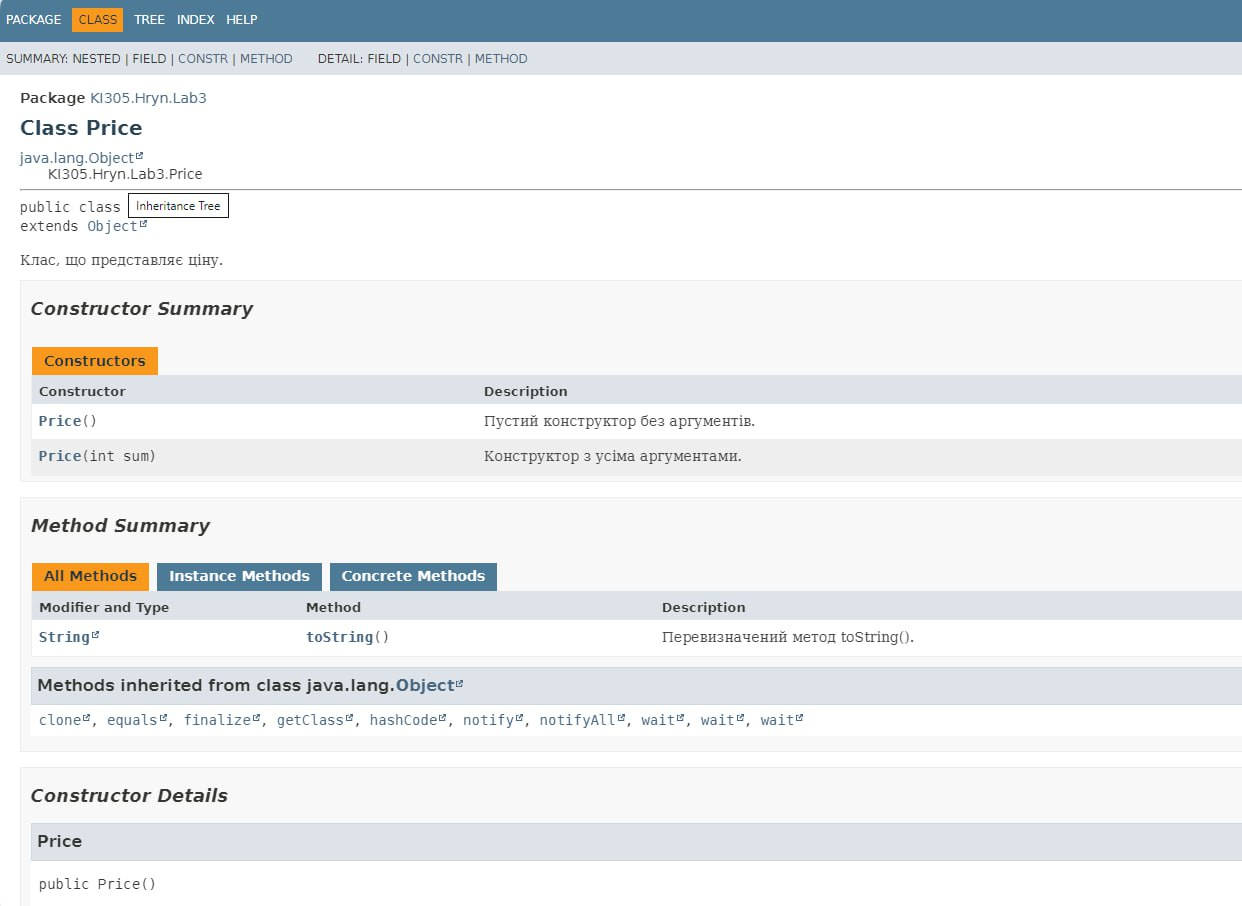






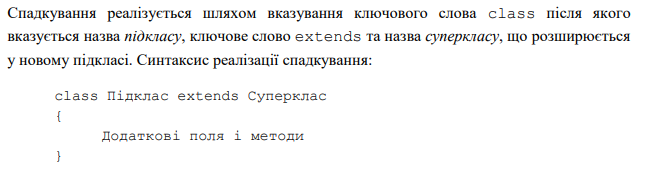




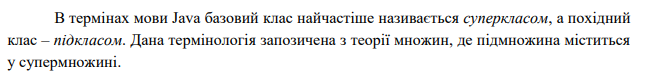


**Контрольні запитання**

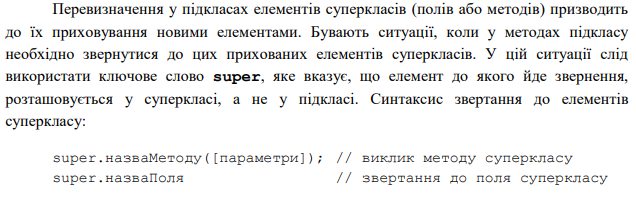
1. ***Синтаксис реалізації спадкування.***

******

1. ***Що таке суперклас та підклас?***

******

1. ***Як звернутися до членів суперкласу з підкласу?***

******

1. ***Коли використовується статичне зв’язування при виклику методу?***

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.

1. ***Як відбувається динамічне зв’язування при виклику методу?***

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 5 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.

1. ***Що таке абстрактний клас та як його реалізувати?***

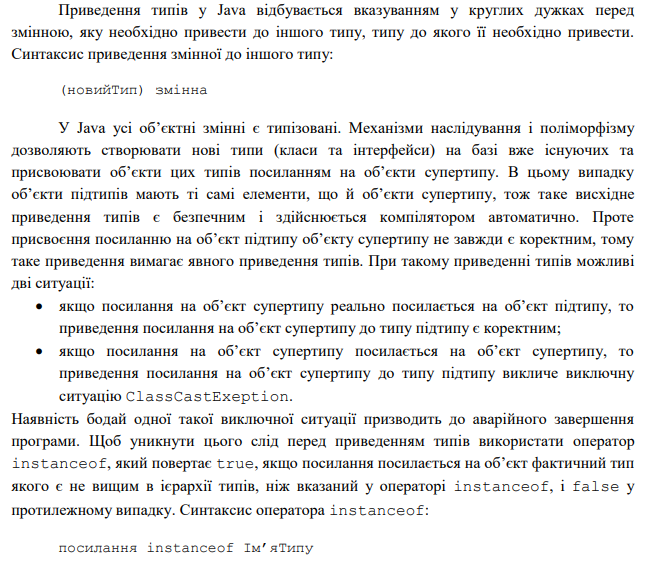
Абстрактні класи призначені бути основою для розробки ієрархій класів та не дозволяють створювати об’єкти свого класу. Вони реалізуються за допомогою ключового слова abstract. На відміну від звичайних класів абстрактні класи можуть містити абстрактні методи (а можуть і не містити). Абстрактні методи – це методи, що 6 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». Синтаксис оголошення абстрактного методу:

******

1. ***Для чого використовується ключове слово instanceof?***

Щоби перевірити коректність приведення супертипу до підтипу слід використовувати оператор instanceof.

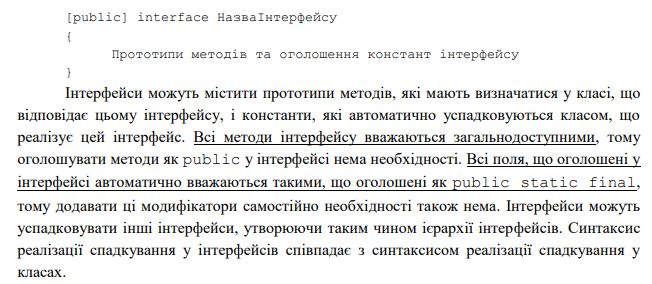
1. ***Як перевірити чи клас є підкласом іншого класу?***

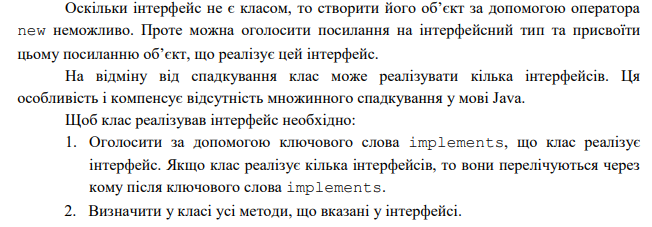
******

1. ***Що таке інтерфейс?***

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.

1. ***Як оголосити та застосувати інтерфейс?***

******

****

**Висновок:** я ознайомилася із спадкуванням та інтерфейсами у мові Java.