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 3**

Виконав: ст. гр. КІ-305

Гнідець В.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 3:** піддослідний пес

**Код програми**

**Class Dog**

package KI305\_Gnidec\_Lab3;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.io.PrintWriter;  
  
*/\*\*  
 \* Клас, що представляє собаку та зберігає інформацію про неї.  
 \*/*public abstract class Dog {  
 //Поля класу  
 private Breed breed;  
 private Color color;  
 private Age age;  
  
 //Поле для запису протоколу  
 private PrintWriter logWriter;  
  
 */\*\*  
 \* Пустий конструктор без аргументів.  
 \* Ініціалізує об'єкт Dog та створює файл протоколу.  
 \*/* //Пустий конструктор без аргументів  
 public Dog() {  
 try {  
 logWriter = new PrintWriter(new FileWriter("D:\\NULP\\KZP\\Lab3\\KI305\_Gnidec\_Lab3\\Dog.txt"));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 //Конструктор зі всіма аргументами  
 public Dog(Breed breed, Color color, Age age) {  
 this.breed = breed;  
 this.color = color;  
 this.age = age;  
 try {  
 logWriter = new PrintWriter(new FileWriter("D:\\NULP\\KZP\\Lab3\\KI305\_Gnidec\_Lab3\\Dog.txt"));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public abstract void petDog();  
 public abstract void feedDog();  
  
  
  
 */\*\*  
 \* 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 spinAroundThroughLeft() {  
 System.*out*.println("Dog is being said to spin around through the left side.");  
 logActivity("You told the dog to spin around through the left side.");//Запис у протокол  
 }  
  
 public void spinAroundThroughRight() {  
 System.*out*.println("Dog is being said to spin around through the right side.");  
 logActivity("You told the dog to spin around through the right side.");//Запис у протокол  
 }  
  
 public void goForAWalk() {  
 System.*out*.println("Go for a walk with dog.");  
 logActivity("You went for a walk with dog\"");//Запис у протокол  
 }  
  
 public void call() {  
 System.*out*.println("You called the dog.");  
 logActivity("You called the dog.");//Запис у протокол  
 }  
  
  
 // Метод для закриття файлу протоколу  
 private void closeLogFile() {  
 if (logWriter != null) {  
 logWriter.close();  
 }  
 }  
 //Метод toString()  
 @Override  
 public String toString() {  
 return "Dog:\nBreed - " + breed + "\nColor - " + color + "\nAge - " + age;  
 }  
  
}

**Class Breed**

package KI305\_Gnidec\_Lab3;  
  
*/\*\*  
 \* Клас, що представляє інформацію про породу собаки  
 \*/*public class Breed {  
 // Поля класу  
 private String nameBreed; // Назва компанії  
 private String infoBreed; // Інформація про компанію  
  
 */\*\*  
 \* Пустий конструктор без аргументів.  
 \* Ініціалізує об'єкт Breed з пустими значеннями.  
 \*/* public Breed(){  
  
 }  
  
 */\*\*  
 \* Конструктор з усіма аргументами.  
 \* Ініціалізує об'єкт Breed з вказаними параметрами.  
 \*  
 \* @param nameBreed Назва породи.  
 \* @param infoBreed Інформація про породу.  
 \*/* public Breed(String nameBreed, String infoBreed){  
 this.nameBreed = nameBreed;  
 this.infoBreed = infoBreed;  
 }  
  
 */\*\*  
 \* Перевизначений метод toString().  
 \*  
 \* @return Рядок, що представляє об'єкт Breed.  
 \*/* @Override  
 public String toString() {  
 return "Breed " +  
 "name: " + nameBreed +  
 ", info: " + infoBreed;  
 }  
}

**Class Color**

package KI305\_Gnidec\_Lab3;  
  
*/\*\*  
 \* Клас, що представляє колір собаки.  
 \*/*public class Color {  
 // Поле класу  
 private String nameColor; // Забарвлення собаки  
  
 */\*\*  
 \* Пустий конструктор без аргументів.  
 \* Ініціалізує об'єкт Color з пустою назвою моделі.  
 \*/* public Color(){  
  
 }  
  
 */\*\*  
 \* Конструктор з усіма аргументами.  
 \* Ініціалізує об'єкт Color з вказаною назвою моделі.  
 \*  
 \* @param nameColor Назва забарвлення собаки.  
 \*/* public Color(String nameColor){  
 this.nameColor = nameColor;  
 }  
  
 */\*\*  
 \* Перевизначений метод toString().  
 \*  
 \* @return Рядок, що представляє об'єкт Color.  
 \*/* @Override  
 public String toString() {  
 return "Color " +  
 "name: " + nameColor;  
 }  
}

**Class Age**

package KI305\_Gnidec\_Lab3;  
  
*/\*\*  
 \* Клас, що представляє вік собаки  
 \*/*public class Age {  
 // Поле класу  
 private int age; // Вік собаки  
  
 */\*\*  
 \* Пустий конструктор без аргументів.  
 \* Ініціалізує об'єкт Age з нульовою сумою.  
 \*/* public Age() {  
  
 }  
  
 */\*\*  
 \* Конструктор з усіма аргументами.  
 \* Ініціалізує об'єкт Age з вказаним віком собаки.  
 \*  
 \* @param age вік собаки.  
 \*/* public Age(int age) {  
 this.age = age;  
 }  
  
 */\*\*  
 \* Перевизначений метод toString().  
 \*  
 \* @return Рядок, що представляє об'єкт Age.  
 \*/* @Override  
 public String toString() {  
 return "Age " + "age: " + age;  
 }  
}

**Class DogOwner**

package KI305\_Gnidec\_Lab3;  
  
*/\*\*  
 \* Головний клас, який представляє програму для власника собаки.  
 \*/*public class DogOwner {  
 */\*\*  
 \* Головний метод програми.  
 \*  
 \* @param args Масив рядків аргументів командного рядка.  
 \*/* public static void main(String[] args) {  
  
 // Ініціалізація полів  
  
 ArcticDog haski = new ArcticDog(new Breed("Haski","this is artic dog"), new Color("white"), new Age(2));  
 System.*out*.println(haski);  
  
 haski.call();  
  
 haski.sitOrStayInfo();  
  
 haski.setSitOrStay(true);  
  
 haski.sitOrStayInfo();  
  
 haski.setSitOrStay(false);  
  
 haski.sitOrStayInfo();  
  
 haski.spinAroundThroughLeft();  
  
 haski.petDog();  
  
 haski.feedDog();  
 haski.countryInfo();  
 }  
}

**Class ArcticDog**

package KI305\_Gnidec\_Lab3;  
  
  
*/\*\*  
 \* Клас, який представляє область проживанння собаки.  
 \* Область проживанння собаки є розширенням класу Dog та реалізує інтерфейс Movements для команд собакою.  
 \*/*public class ArcticDog extends Dog implements Movements, NameOfCountruWhereLives {  
 private boolean state;  
 */\*\*  
 \* Конструктор без параметрів для певної породи собаки.  
 \* Створює собаку певної породи яка при виклику стоїть.  
 \*/* public ArcticDog(Breed breed, Color color, Age age) {  
 super(breed,color,age);  
 state = false;  
 }  
 */\*\*  
 \* Перевизначений метод для того щоб погладити собаку.  
 \*/* @Override  
 public void petDog() {  
 System.*out*.println("You petted the dog");  
 }  
 */\*\*  
 \* Перевизначений метод для того щоб погодувати собаку.  
 \*/* @Override  
 public void feedDog() {  
  
 System.*out*.println("You fed the dog");  
 }  
  
 @Override  
 public void setSitOrStay(boolean data) {  
 this.state = data;  
 }  
 */\*\*  
 \* Реалізація методу інтерфейсу Movements для виведення інформації про стан собаки.  
 \*/* @Override  
 public void sitOrStayInfo() {  
 if (state)  
 System.*out*.println("Dog is sitting");  
 else  
 System.*out*.println("Dog is staying");  
 }  
  
 @Override  
 public void countryInfo() {  
 System.*out*.println("Dog is leaving in Arctic");  
 }  
}

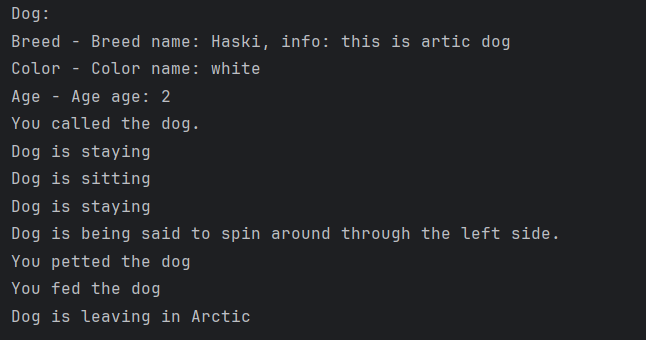
**Interface NameOfCountruWhereLives**

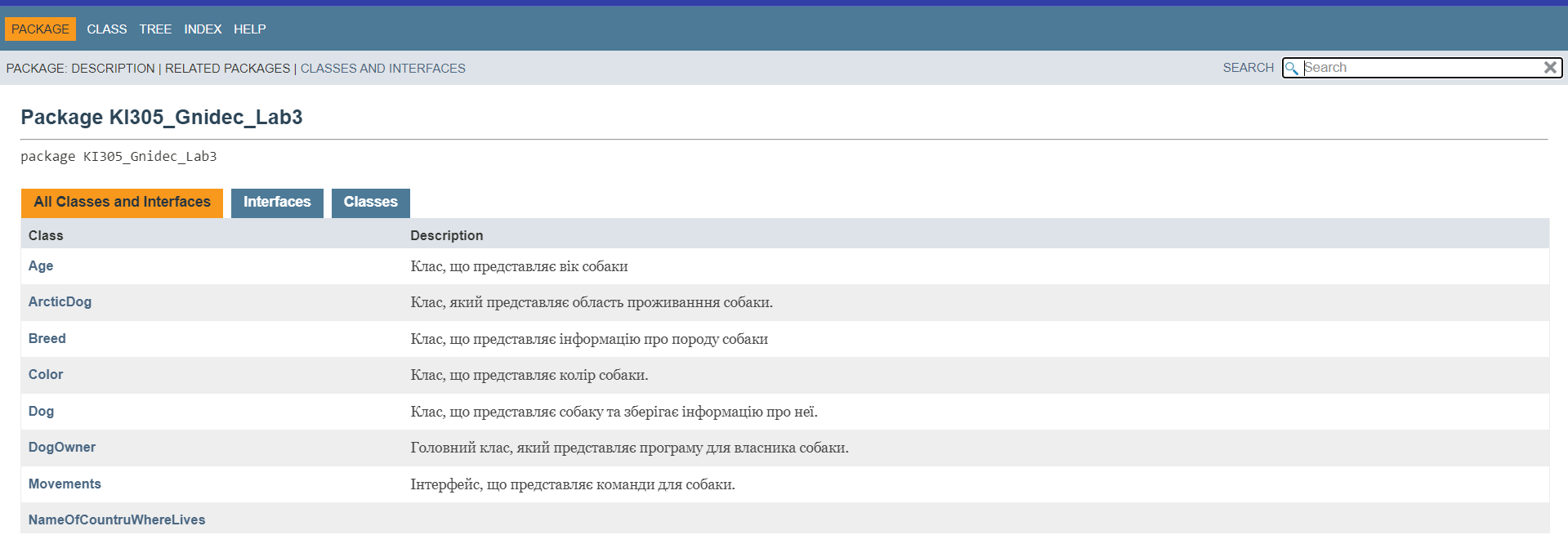
package KI305\_Gnidec\_Lab3;  
  
public interface NameOfCountruWhereLives {  
 void countryInfo();  
}

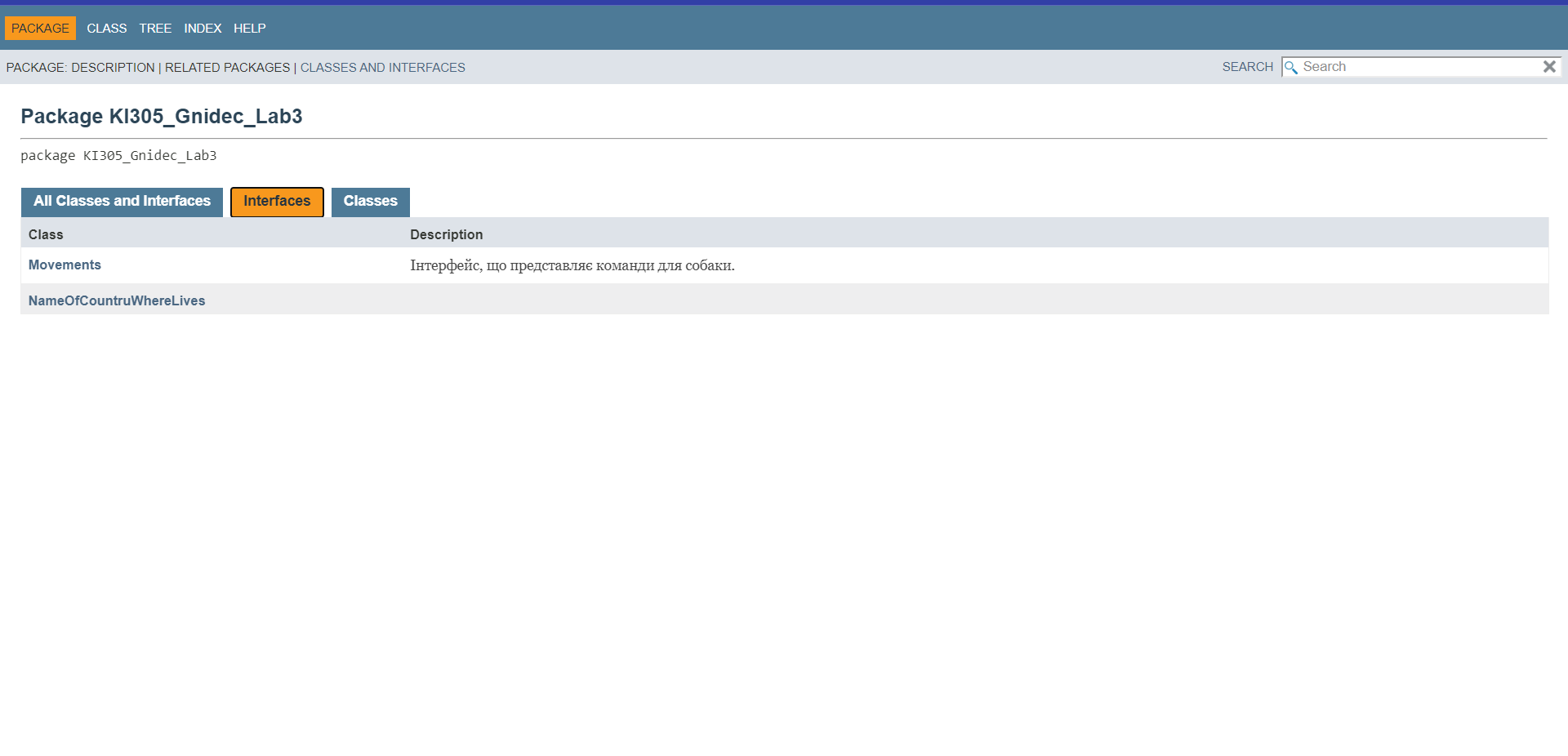
**Interface Movements**

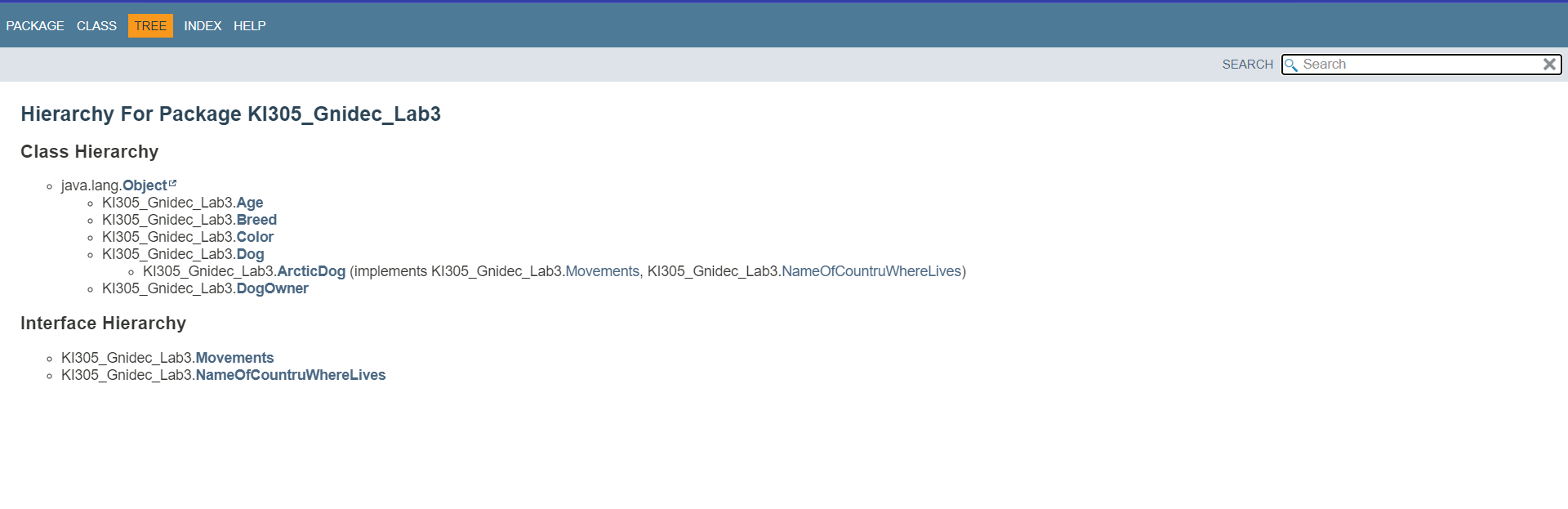
package KI305\_Gnidec\_Lab3;  
  
  
*/\*\*  
 \* Інтерфейс, що представляє команди для собаки.  
 \* Реалізуючі класи повинні реалізовувати методи цього інтерфейсу для вказування команд і надавати інформацію про їх стан.  
 \*/*public interface Movements {  
  
 */\*\*  
 \* Встановлює команду для собаки(сидіти або стояти).  
 \*  
 \* @param data true, якщо собака повинна сидіти, false, якщо собака повинна стояти.  
 \*/* void setSitOrStay(boolean data);  
  
 */\*\*  
 \* Виводить інформацію про поточний стан собачки.  
 \*/* void sitOrStayInfo();  
}

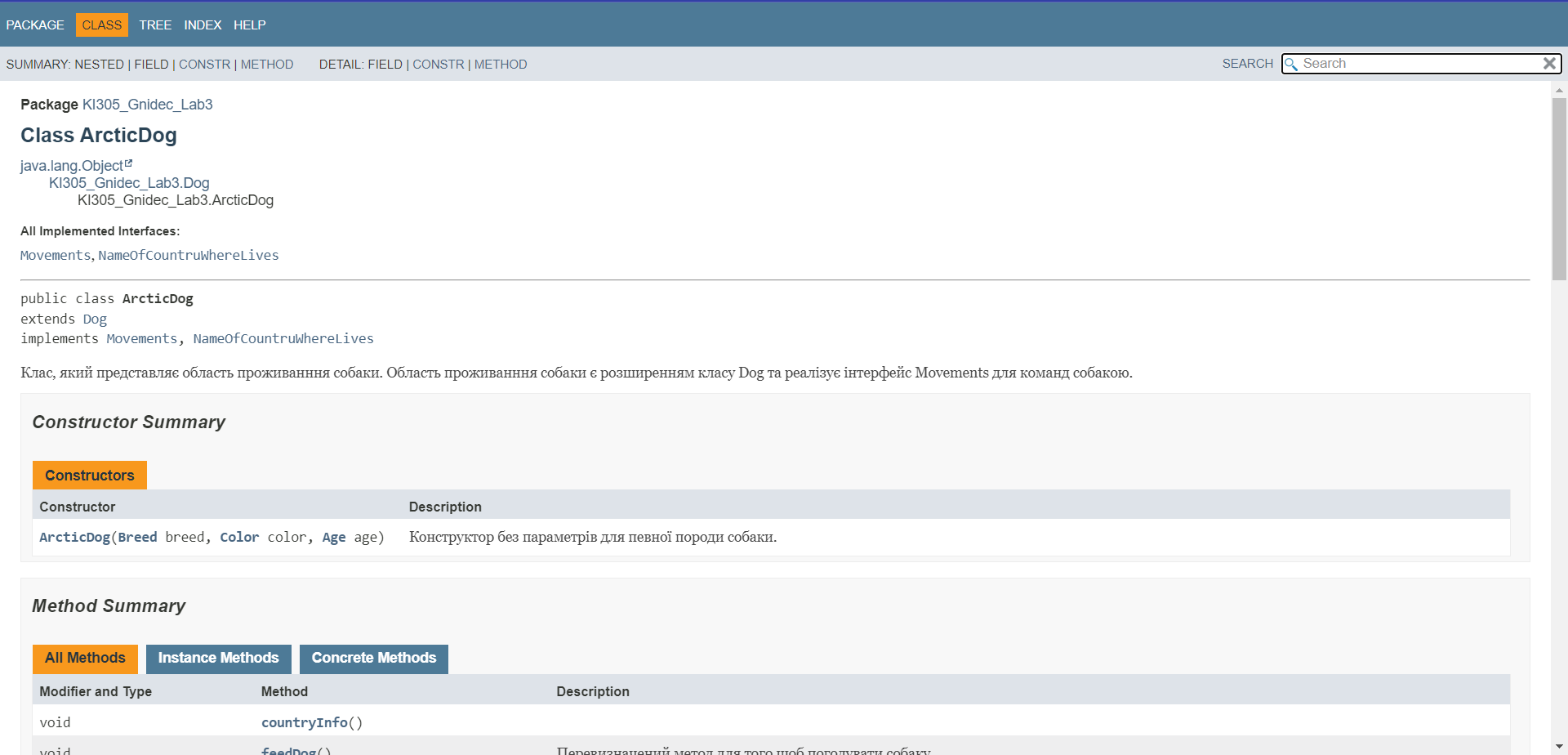
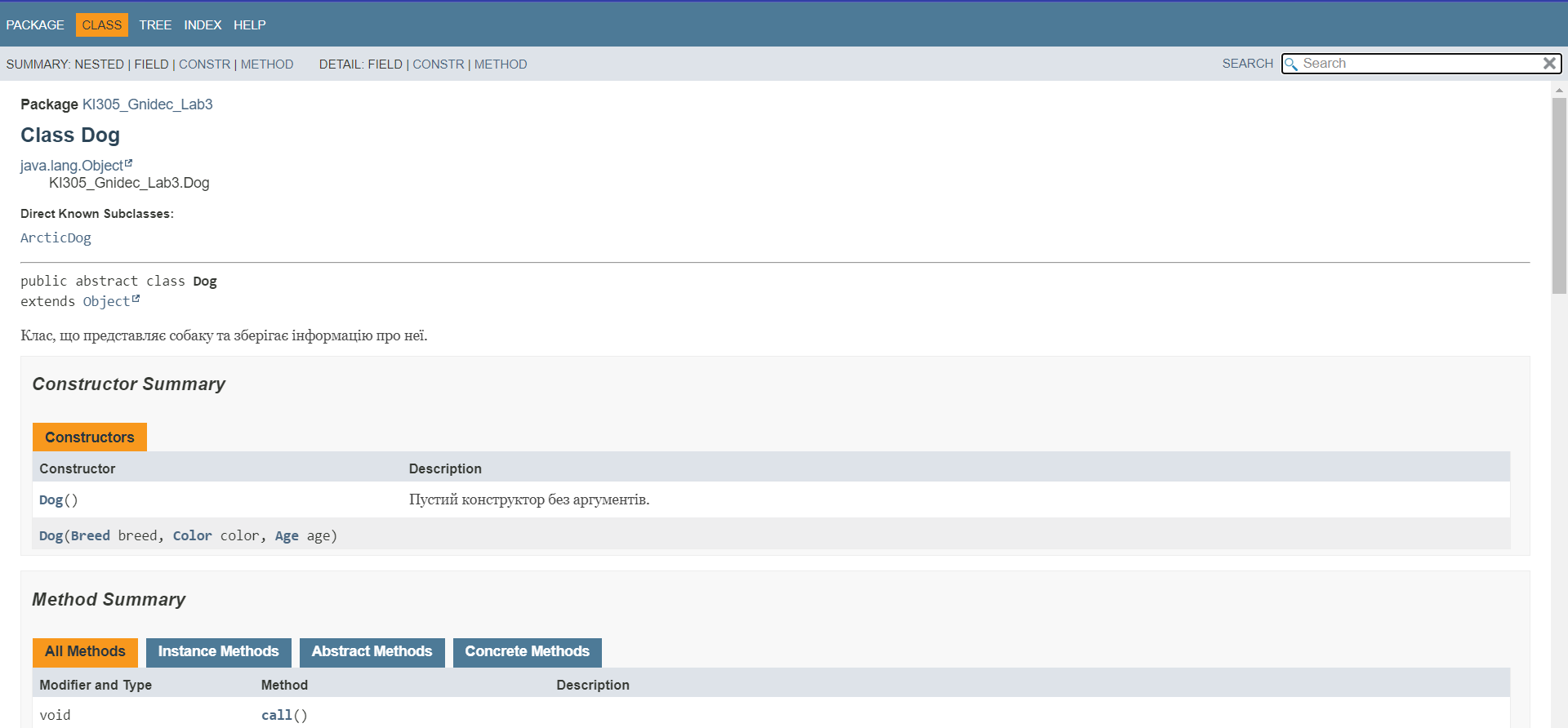
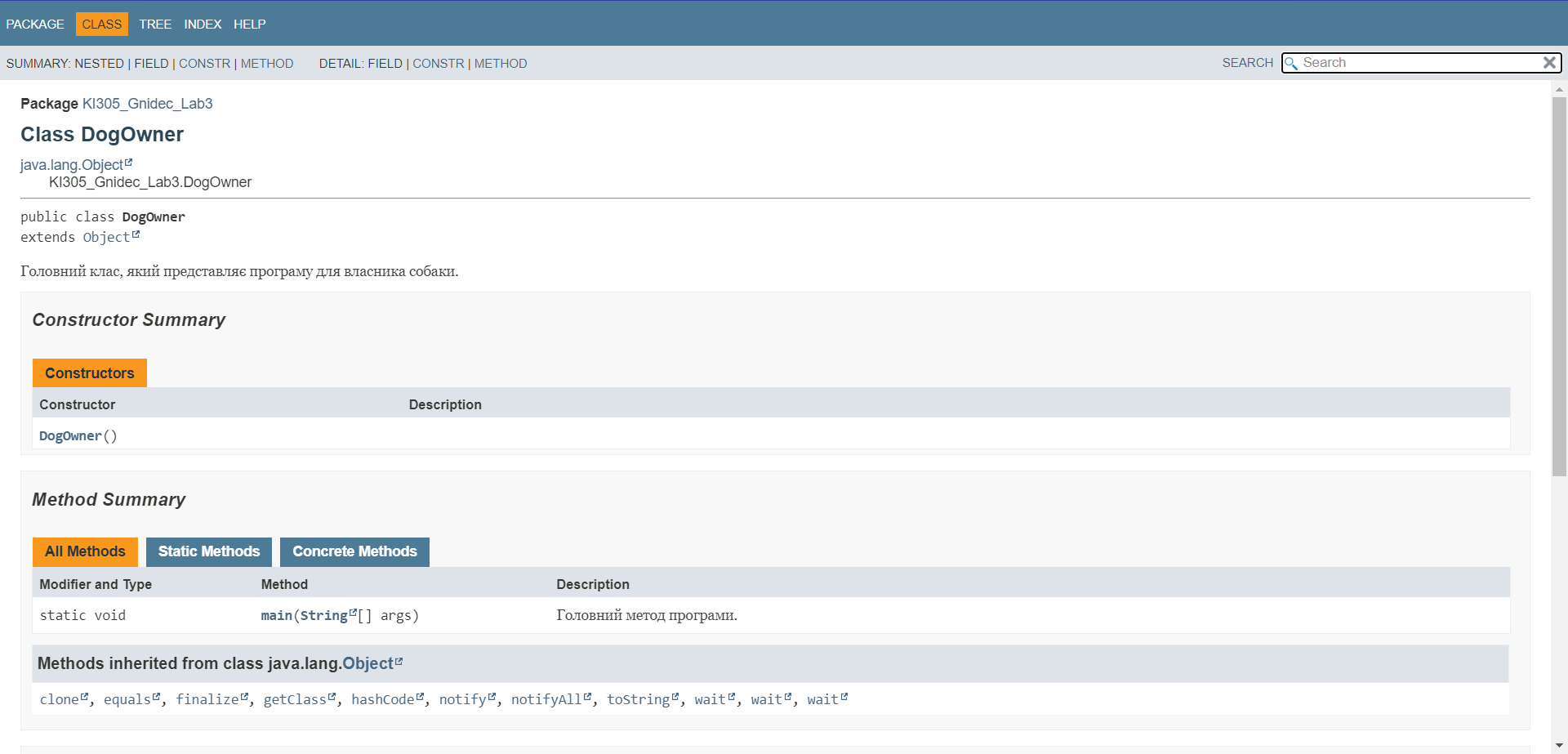
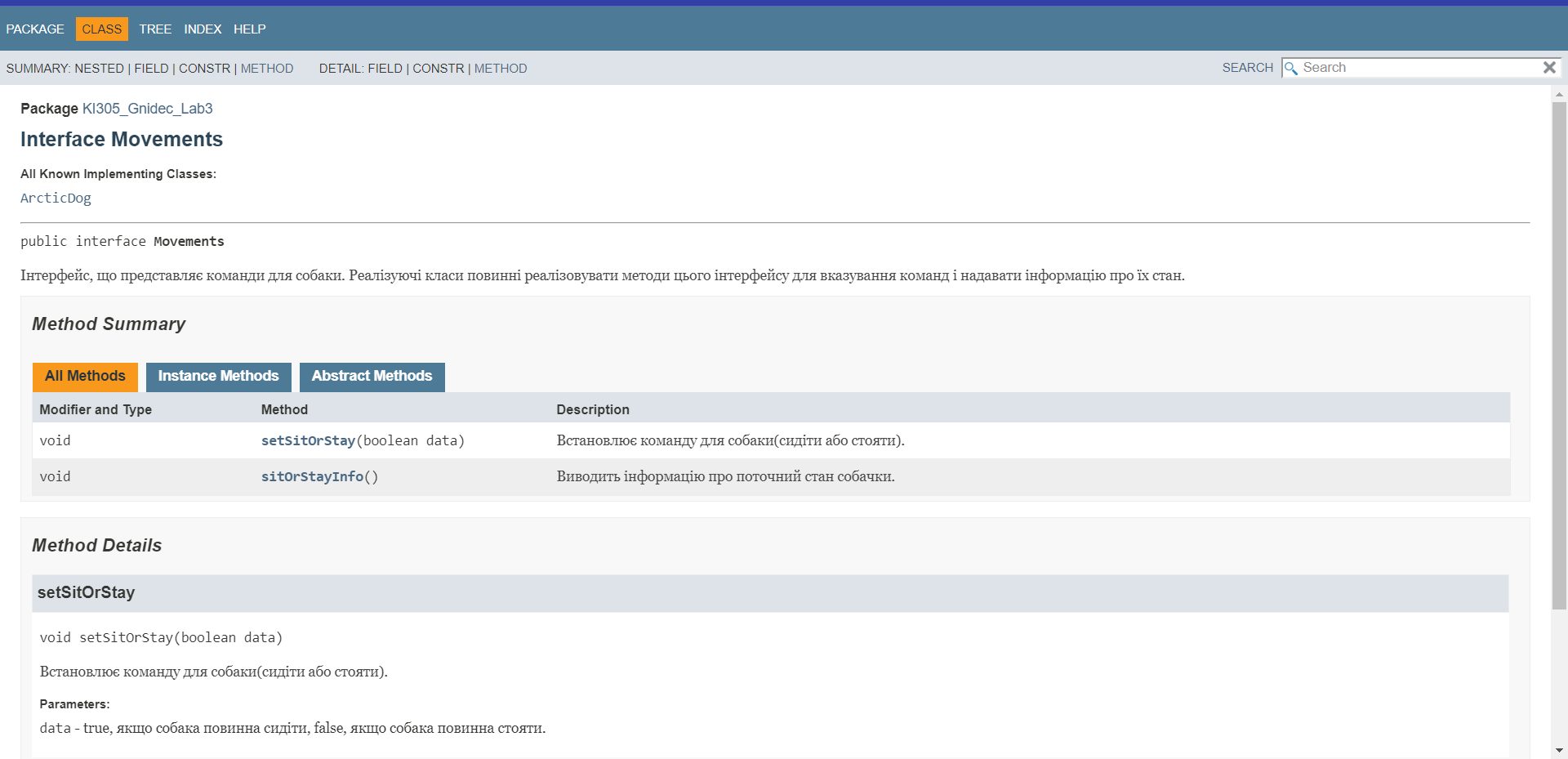
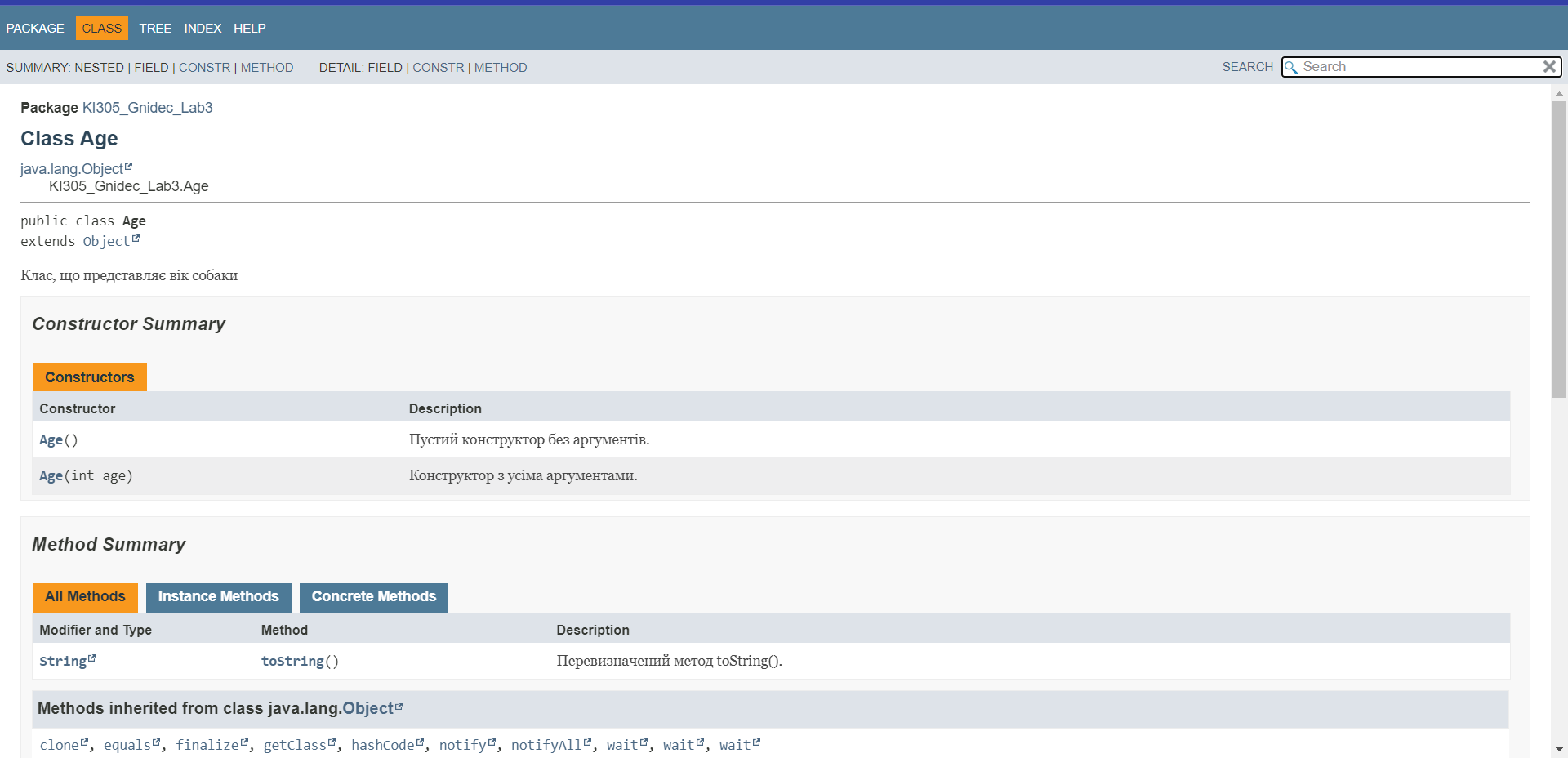
**Результат роботи програми:**

****

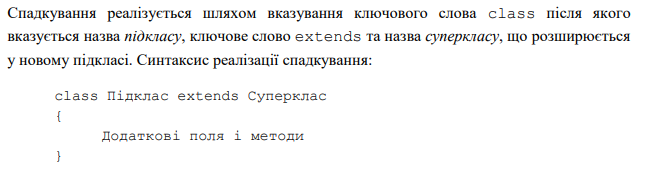




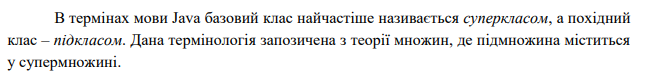
****

**** ****   **Контрольні запитання**

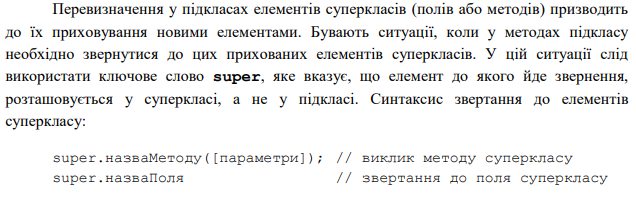
1. ***Синтаксис реалізації спадкування.***

******

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?***

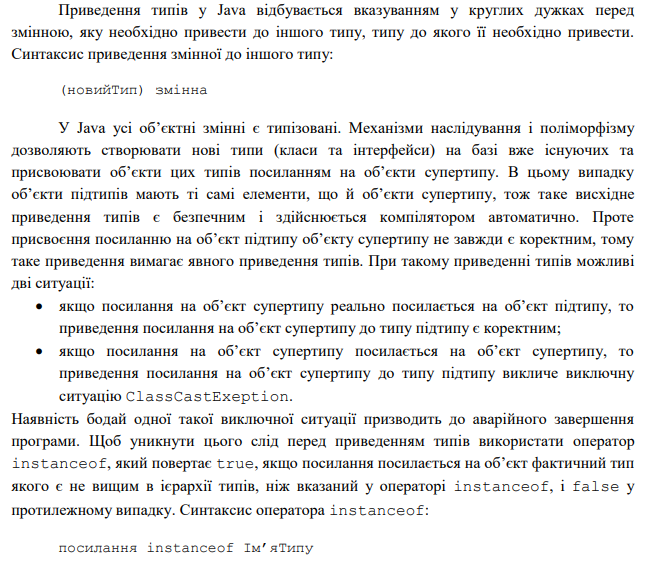
Абстрактні класи призначені бути основою для розробки ієрархій класів та не дозволяють створювати об’єкти свого класу. Вони реалізуються за допомогою ключового слова abstract. На відміну від звичайних класів абстрактні класи можуть містити абстрактні методи (а можуть і не містити). Абстрактні методи – 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.

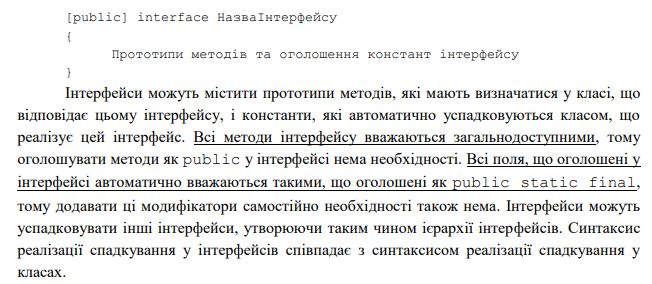
1. ***Як перевірити чи клас є підкласом іншого класу?***

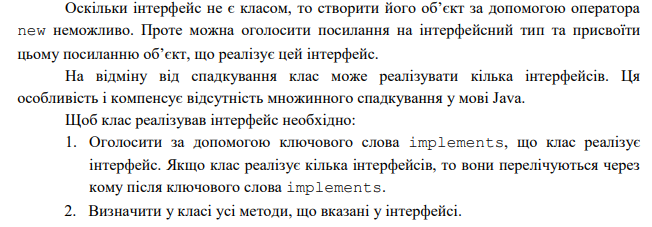
******

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 ознайомився із спадкуванням та інтерфейсами у мові Java.