Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра «Електронних обчислювальних машин»



Звіт

з лабораторної роботи № 4

з дисципліни: «Кросплатформні засоби програмування»

на тему: «Спадкування та інтерфейси»

**Виконав:**

студент групи КІ-35

Скіпар І.В.

**Прийняв:**

доцент кафедри ЕОМ

Іванов Ю. С.

Львів – 2022

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

**Завдання (варіант № 10)**

1. Написати та налагодити програму на мові Java, що розширює клас, що реалізований у лабораторній роботі №3, для реалізації предметної області заданої варіантом. Суперклас, що реалізований у лабораторній роботі №3, зробити абстрактним . Розроблений підклас має забезпечувати механізм свого конкретного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab4 та володіти коментарями, які дозволяють автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленого пакету.
3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації.
4. Дати відповідь на контрольні запитання:
5. Синтаксис реалізації спадкування.
6. Що таке суперклас та підклас?
7. Як звернутись до членів суперкласу з підкласу?
8. Коли використовується статичне зв’язування при виклику методу?
9. Як відбувається динамічне зв’язування при виклику методу?
10. Що таке абстрактний клас та як його реалізувати?
11. Для чого використовується ключове слово instanceof?
12. Як перевірити чи клас є підкласом іншого класу?
13. Що таке інтерфейс?
14. Як оголосити та застосувати інтерфейс?

**Текст програми**

House.java

package KI\_35.Skipar.Lab4;  
*/\*\*  
 \* Skipar package  
 \*/*import java.io.FileNotFoundException;  
import java.io.PrintWriter;  
  
*/\*\*  
 \* Abstract class <code>House</code> implement house  
 \*/*public abstract class House {  
  
 */\*\*  
 \* Default constructor  
 \** ***@throws*** *FileNotFoundException Not found log-file  
 \*/* public House() throws FileNotFoundException {  
 fout = new PrintWriter("Log.txt");  
 citizens = 0;  
 }  
  
 */\*\*  
 \* Constructor  
 \** ***@param*** *citizens count of citizens in the house  
 \** ***@param*** *l brick length  
 \** ***@param*** *w brick width  
 \** ***@param*** *h brick height  
 \** ***@param*** *br\_count count of bricks  
 \** ***@param*** *dr\_count count of doors  
 \** ***@param*** *wd\_count count of windows  
 \** ***@throws*** *FileNotFoundException Not found log-file  
 \*/* public House(int citizens, int l, int w, int h, int br\_count, int dr\_count, int wd\_count) throws FileNotFoundException {  
 this.citizens = citizens;  
 set\_brick(l, w, h, br\_count);  
 \_door.setCount(dr\_count);  
 \_window.setCount(wd\_count);  
 fout = new PrintWriter("Log.txt");  
 }  
  
 */\*\*  
 \* Method set or change bricks in the house  
 \** ***@param*** *c count of bricks  
 \** ***@param*** *h brick height  
 \** ***@param*** *l brick length  
 \** ***@param*** *w brick width  
 \*/* public void set\_brick(int l, int w, int h, int c) {  
 \_brick = new Brick(l,w,h,c);  
 }  
  
 */\*\*  
 \* Method which return count of brick in the house  
 \** ***@return*** *count of window  
 \*/* public int get\_brick\_count() {  
 return \_brick.getCount();  
 }  
  
 */\*\*  
 \* Method which return bricks height in the house  
 \** ***@return*** *count of window  
 \*/* public int get\_brick\_height() {  
 return \_brick.getHeight();  
 }  
  
 */\*\*  
 \* Method which return bricks width in the house  
 \** ***@return*** *count of window  
 \*/* public int get\_brick\_width() {  
 return \_brick.getWidth();  
 }  
  
 */\*\*  
 \* Method which return brick length in the house  
 \** ***@return*** *count of window  
 \*/* public int get\_brick\_length() {  
 return \_brick.getLength();  
 }  
  
 */\*\*  
 \* Method which return count of door  
 \** ***@return*** *count of doors  
 \*/* public int get\_door() {  
 return \_door.getCount();  
 }  
  
 */\*\*  
 \* Method set or change count of doors in the house  
 \** ***@param*** *count doors count  
 \*/* public void set\_door(int count) {  
 \_door = new Door(count);  
 }  
  
 */\*\*  
 \* Method which return count of window  
 \** ***@return*** *count of window  
 \*/* public int get\_window() {  
 return \_window.getCount();  
 }  
  
 */\*\*  
 \* Method set or change count of windows in the house  
 \** ***@param*** *count windows count  
 \*/* public void set\_window(int count) {  
 \_window = new Window(count);  
 }  
  
 */\*\*  
 \* Method return count of citizens in the house  
 \** ***@return*** *count of citizens in the house  
 \*/* public int getCitizens() {  
 return citizens;  
 }  
  
 */\*\*  
 \* Method set or change count of citizens in the house  
 \** ***@param*** *citizens count of citizens in the house  
 \** ***@throws*** *FileNotFoundException Not found log-file  
 \*/* public void setCitizens(int citizens) throws FileNotFoundException {  
 if (this.citizens<citizens)  
 fout.print(citizens-this.citizens + " new citizens in the house\n");  
 else fout.print(this.citizens-citizens + " citizens leave the house\n");  
 this.citizens = citizens;  
 fout.flush();  
 }  
  
 */\*\*  
 \* Method close log-file  
 \*/* public void closeFile () {  
 fout.close();  
 }  
  
 private Brick \_brick= new Brick();  
 private Door \_door = new Door();  
 private Window \_window = new Window();  
 protected int citizens;  
 protected PrintWriter fout;  
  
}

iOffice.java

package KI\_35.Skipar.Lab4;  
  
*/\*\*  
 \* Office interface  
 \*/*public interface iOffice {  
 */\*\*  
 \* Normal area for 1 person  
 \*/* static final long *AREA\_FOR\_PERSON* = 6000000;  
 */\*\*  
 \* Normal floor height  
 \*/* static final long *FLOOR\_HEIGHT* = 3000;  
  
 */\*\*  
 \* Method which calculate normal count of employees  
 \** ***@return*** *normal count of employees  
 \*/* long NormalEmployeesCount();  
}

OfficeCentre.java

package KI\_35.Skipar.Lab4;  
  
import java.io.FileNotFoundException;  
import java.io.PrintWriter;  
  
*/\*\*  
 \* Class <code>OfficeCentre</code> implement office centre  
 \** ***@author*** *Ілля  
 \** ***@version*** *4.0  
 \*/*public class OfficeCentre extends House implements iOffice {  
  
 */\*\*  
 \* Default constructor <code>OfficeCentre</code>  
 \** ***@throws*** *FileNotFoundException Not found log-file  
 \*/* public OfficeCentre() throws FileNotFoundException {  
 super();  
 }  
  
 */\*\*  
 \* Constructor <code>OfficeCentre</code>  
 \** ***@param*** *employees count of citizens in the house  
 \** ***@param*** *br\_length brick length  
 \** ***@param*** *br\_width brick width  
 \** ***@param*** *br\_height brick height  
 \** ***@param*** *br\_count count of bricks  
 \** ***@param*** *dr\_count count of doors  
 \** ***@param*** *wd\_count count of windows  
 \** ***@throws*** *FileNotFoundException Not found log-file  
 \*/* public OfficeCentre(int employees, int br\_length, int br\_width, int br\_height, int br\_count, int dr\_count, int wd\_count) throws FileNotFoundException {  
 super(employees, br\_length, br\_width, br\_height, br\_count, dr\_count, wd\_count);  
 }  
  
 */\*\*  
 \* Method which calculate normal count of employees  
 \** ***@return*** *normal count of employees  
 \*/* @Override  
 public long NormalEmployeesCount() {  
 return (long) ((Math.*pow*(get\_brick\_count()\*get\_brick\_height()\*get\_brick\_length()/*FLOOR\_HEIGHT*, 2)/16)/*AREA\_FOR\_PERSON*);  
 }  
  
 */\*\*  
 \* Override method from <code>House</code>  
 \** ***@param*** *citizens count of citizens in the house  
 \** ***@throws*** *FileNotFoundException Not found log-file  
 \*/* @Override  
 public void setCitizens(int citizens) throws FileNotFoundException {  
 if (super.getCitizens()<citizens)  
 super.fout.print(citizens-super.getCitizens() + " new employees in the office\n");  
 else fout.print(super.getCitizens()-citizens + " employees leave the office\n");  
 this.citizens = citizens;  
 fout.flush();  
 }  
}

**Результат виконання програми**

Консоль

Emp:0

BC: 0

BL: 250

BW: 120

BH: 65

D: 0

W: 0

Norm: 0

Emp:1500

BC: 15000

BL: 65

BW: 150

BH: 300

D: 4

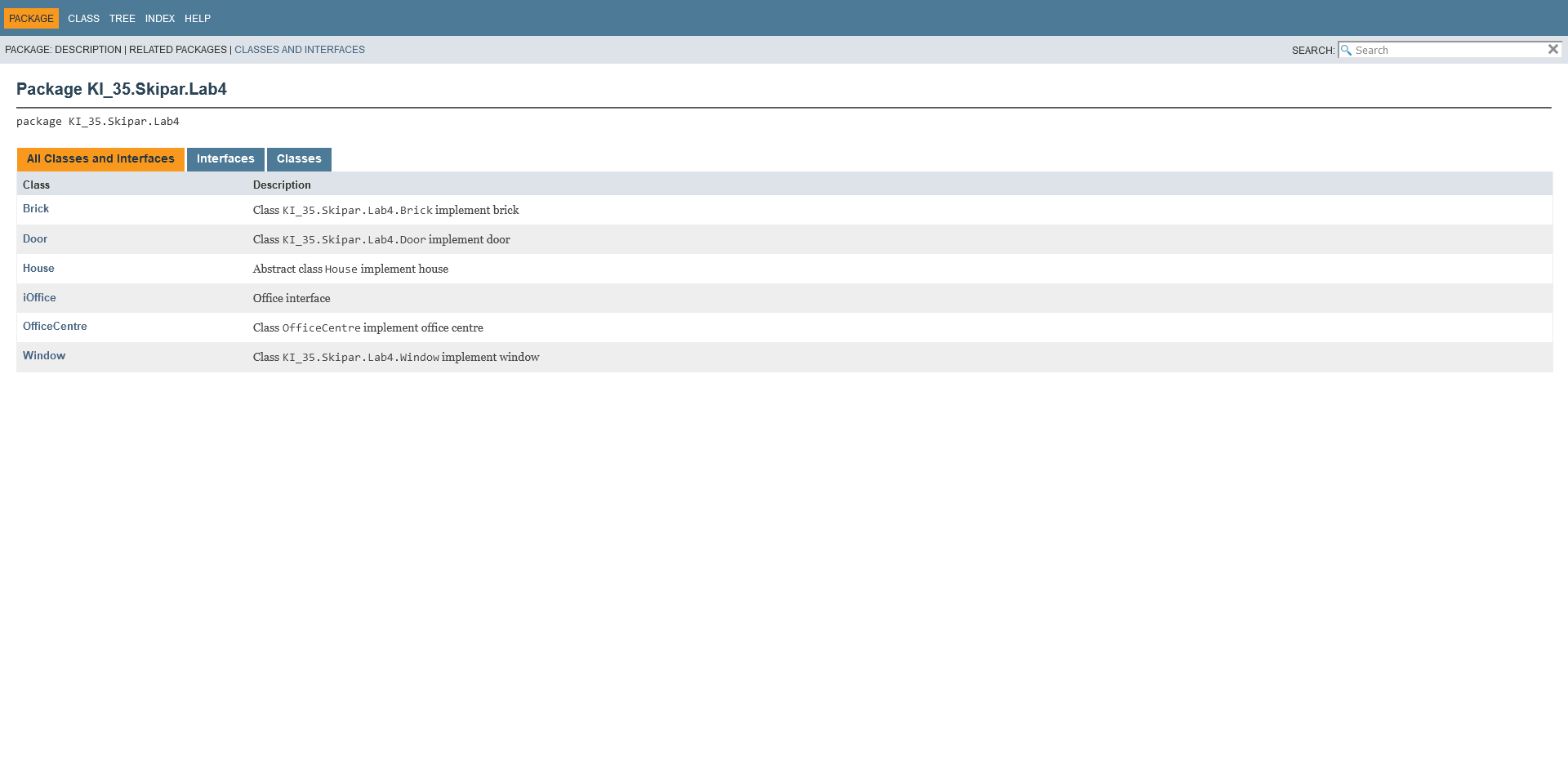
W: 20

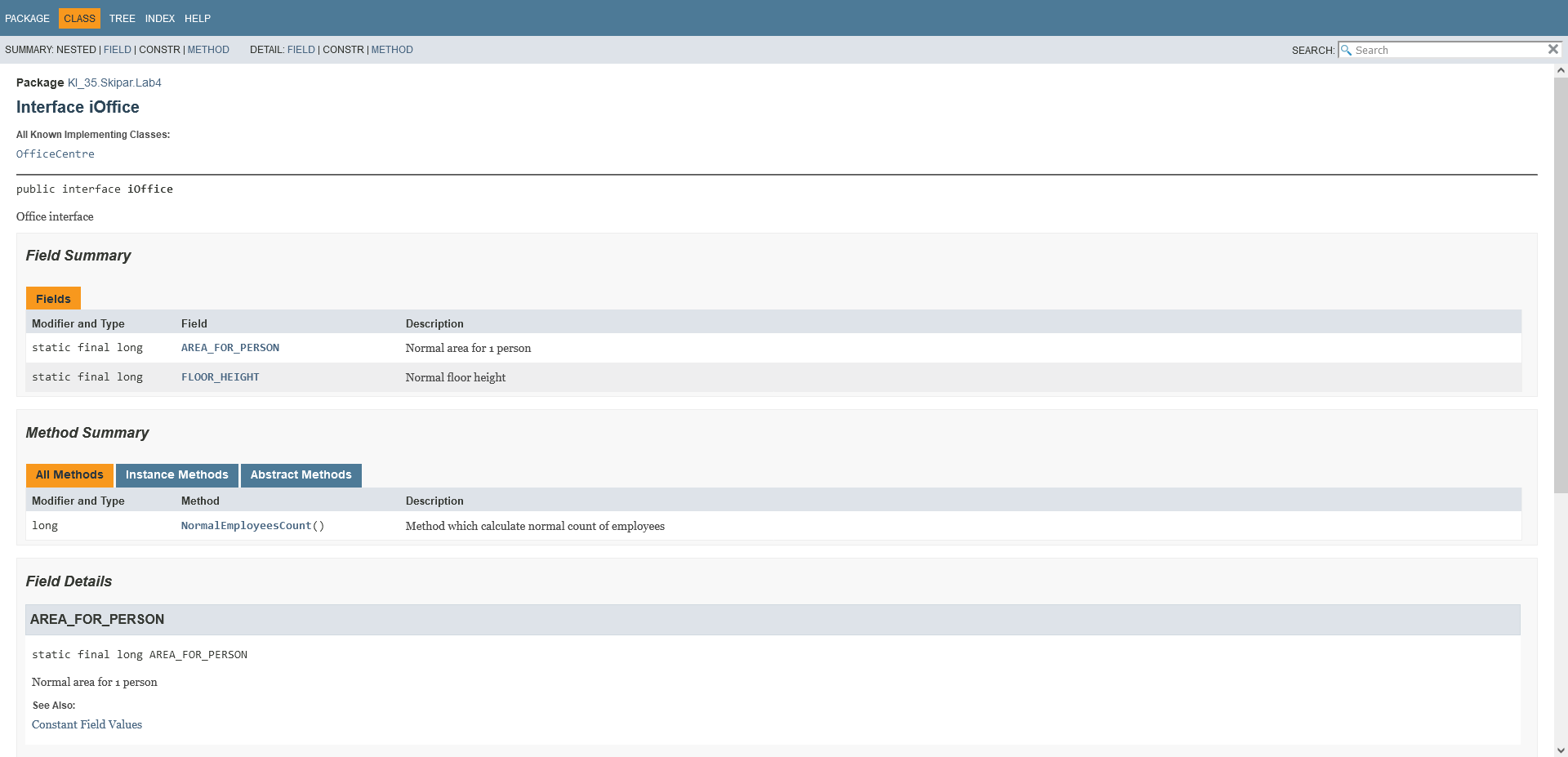
Norm: 99

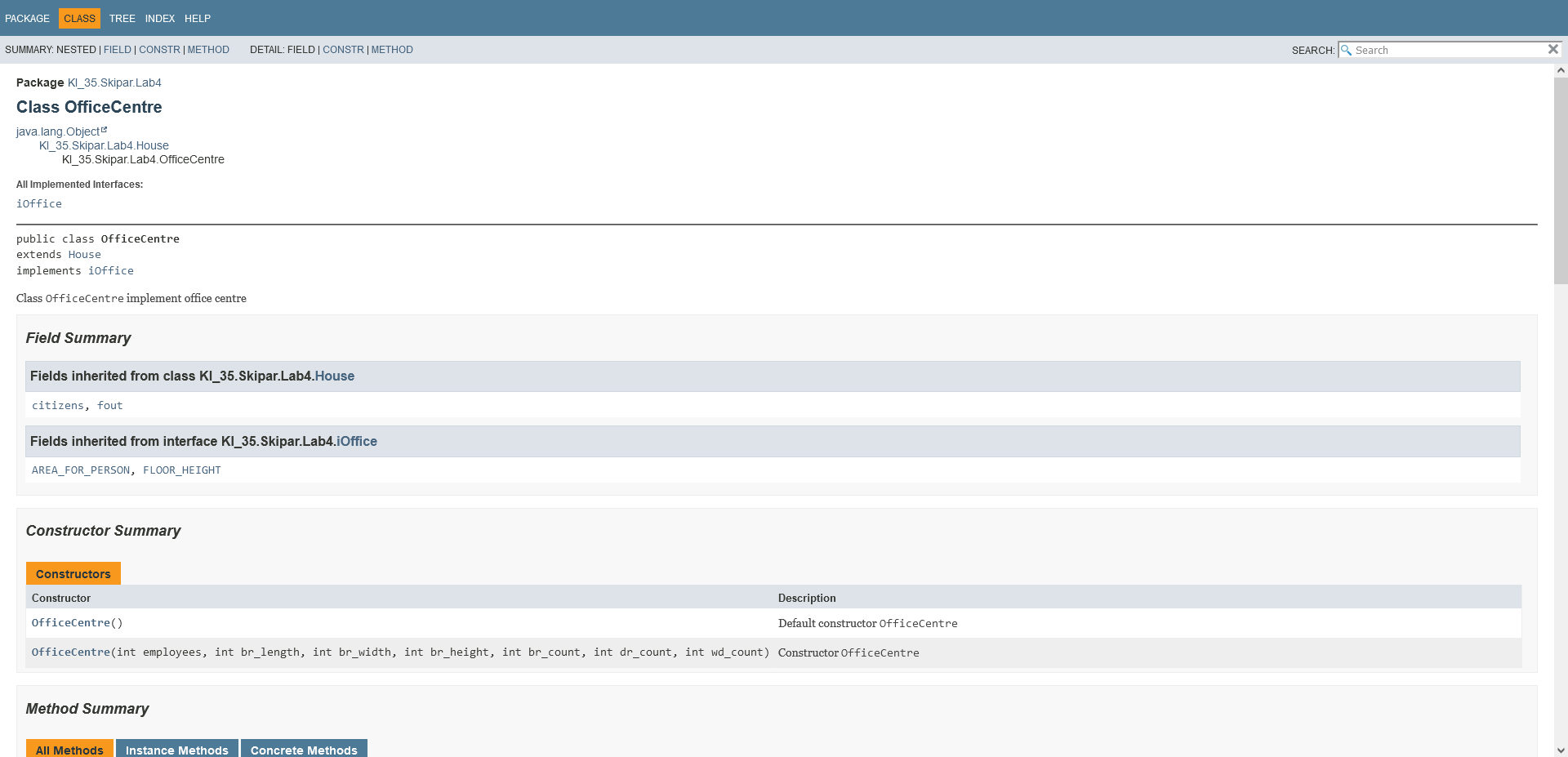
Log.txt

1500 new employees in the office

**Фрагмент згенерованої документації**







**Відповіді на контрольні запитання**

1. Суперклас - це батьківський клас. Підклас – клас, який успадковується від суперкласу

3. До членів суперкласу можна за допомогою ключового слова super.(назва методу/поля).

**Висновок**

В результаті виконання лабораторної роботи я навчився працювати з спадкуванням та інтерфейсами в Java. Частина коду автоматично згенерована середовищем IntelliJ IDEA. Пакет створений в лабораторній роботі може використовуватись в подальшому для обрахування кількості робочих місць в офісі