Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Звіт

З лабораторної роботи №4

З дисципліни «Кросплатформенні засоби програмування»

На тему: «Спадкування та інтерфейси»

Виконав:
ст.гр. КІ-36
Литовко С.Г
Прийняв:
Іванов Ю.С

Львів-2022

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

ТЕОРЕТИЧНІ ВІДОМОСТІ

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 C++ не існує. На відміну від C++ у Java можливе спадкування лише одного базового класу (множинне спадкування відсутнє). Спадкування реалізується шляхом вказування ключового слова class після якого вказується назва *підкласу*, ключове слово extends та назва *суперкласу*, що розширюється у новому підкласі. Синтаксис реалізації спадкування:

```
class Підклас extends Суперклас
{
Додаткові поля і методи
}
```

В термінах мови Java базовий клас найчастіше називається *суперкласом*, а похідний клас – *підкласом*. Дана термінологія запозичена з теорії множин, де підмножина міститься у супермножині.

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.

Перевизначення у підкласах елементів суперкласів (полів або методів) призводить до їх приховування новими елементами. Бувають ситуації, коли у методах підкласу необхідно звернутися до цих прихованих елементів суперкласів. У цій ситуації слід використати ключове слово **super**, яке вказує,

що елемент до якого йде звернення, розташовується у суперкласі, а не у підкласі. Синтаксис звертання до елементів суперкласу:

```
super.назваМетоду([параметри]); // виклик методу суперкласу
super.назваПоля // звертання до поля суперкласу
```

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. Синтаксис виклику конструктора суперкласу з конструктора підкласу:

```
public НазваПідкласу([параметри])
{
super([параметри]);
оператори конструктора підкласу
}
```

ЗАВДАННЯ

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 механізми свого коректного функціонування та реалізовувати мінімум один інтерфейс. Програма має розміщуватися в пакеті Група.Прізвище.Lab4 та володіти коментарями, які дозволять 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.

Варіант

6. Бомбардувальник

Код програми:

App.java

```
package ki36.Lytovko.lab_04;
/**
 * Class App
 * @author Sofia
 *
 */
public class App {
    /**
     * @param argc
     */
    public static void main(String[] argc)
    {
        try
        {
            Bomber bomber = new Bomber();
            bomber.closeChassis();
            bomber.refuel();
            bomber.startFly();
            bomber.startEngine(4);
            bomber.throwBomb();
            bomber.dispose();
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Boomber.java

```
package ki36.Lytovko.lab_04;
/**
```

```
 * Class Bomber
 * @author Sofia
 *
 */
public class Bomber extends Plane implements IBomber{

      private int bombs = 50;
    /**
     * Constructor
     * @throws Exception
     */
    public Bomber() throws Exception {}
    /**
     * Method throws a bomb
     * @throws Exception
     */
    public void throwBomb() throws Exception {
        if(bombs == 0)
        {
            throw new Exception("No bombs");
        }
        bombs -= 1;
        super.message("Remain bombs: " + bombs);
    }
}
```

Chassis.java

```
package ki36.Lytovko.lab_04;
/**
 * Class Chassis
 * @author Sofia
 * @version 1.0
 */
public class Chassis {
    private boolean isClosed = false;
    /**
```

```
     * Method returns isOpen value
     */
    public boolean isOpen() {
        return !isClosed;
    }
    /**
     * Method opens the chassis
     */
    public void open()
    {
        isClosed = false;
    }
    /**
     * Method closes the chassis
     */
    public void close()
    {
        isClosed = true;
    }
}
```

Engine.java

```
package ki36.Lytovko.lab_04;
import java.io.PrintWriter;
/**
 * Class Engine
 * @author Sofia
 * @version 1.0
 */
public class Engine {

    private int fuel = 0;
    private final int FUEl_BANK_CAPACITY = 1000;
    private PrintWriter fout;
    /**
     * Constructor
```

```
 * @param fout
 */
public Engine(PrintWriter fout)
{
    this.fout = fout;
}
/**
 * Method starts the engine
 * @param sec
 * @throws Exception
 */
public void run(int sec) throws Exception {
    for(int i = 0; i < sec; i++)
    {
        if(fuel < 1)
        {
            throw new Exception("No fuel!");
        }
        fuel -= 10;
        message("Remain fuel: " + fuel);
        Thread.sleep(1000);
    }
}
/**
 * Method refuels the engine
 */
public void refuel()
{
    fuel = FUEl_BANK_CAPACITY;
}
/**
 * Method closes the file
 */
public void dispose()
{
    fout.flush();
```

```
        fout.close();
    }
    /**
     * Method returns fuel level
     */
    public int getFuelLevel()
    {
        return fuel;
    }

    protected void message(String message)
    {
        System.out.println(message);
        fout.println(message);
    }
}
```

IBoomber.java

```
package ki36.Lytovko.lab_04;
public interface IBomber {
    void throwBomb() throws Exception;
}
```

Plane.java

```
package ki36.Lytovko.lab_04;
import java.io.File;
import java.io.PrintWriter;
/**
 * Class Plane
 * @author Sofia
 *
 */
public abstract class Plane {

    protected PrintWriter fout;
    protected Chassis chassis = new Chassis();
```

```
private boolean isInTheAir = false;
private Engine engine;
/**
 * Constructor
 * @throws Exception
 */
public Plane() throws Exception
{
    fout = new PrintWriter(new File("lab4.txt"));
    engine = new Engine(fout);
}
/**
 * Method starts the engine
 * @param sec
 * @throws Exception
 */
public void startEngine(int sec) throws Exception
{
    engine.run(sec);
}
/**
 * Method refuels the engine
 */
public void refuel()
{
    message("Engine refuel...");
    engine.refuel();
}
/**
 * Method returns isOpen value
 */
public boolean isChassisOpen(){
    return chassis.isOpen();
}
/**
 * Method closes the chassis
```

```
 */
public void closeChassis() {
    chassis.close();
    message("Chassis closed");
}
/**
 * Method opens the chassis
 */
public void openChassis() {
    chassis.open();
    message("Chassis open");
}
/**
 * Method for landing
 */
public void land() throws Exception
{
    if(!chassis.isOpen())
    {
        throw new Exception("Chassis isn't open");
    }

    isInTheAir = false;

}
/**
 * Method starts to fly
 */
public void startFly() {
    isInTheAir = true;
}
/**
 * Method returns fuel level
 */
public int getFuelLevel()
{
```

```
        return engine.getFuelLevel();
    }
    /**
     * Method closes the file
     */
    public void dispose()
    {
        fout.flush();
        fout.flush();
        engine.dispose();
    }

    protected void message(String message)
    {
        System.out.println(message);
        fout.println(message);
    }
}
```

Результат програми:

```
Chassis closed
Engine refuel...
Remain fuel: 990
Remain fuel: 980
Remain fuel: 970
Remain fuel: 960
Remain bombs: 49
```

Висновок: Зробивши цю лабораторну роботу, я ознайомилася з спадкуванням та інтерфейсами у мові Java.