Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ



**Звіт**

З лабораторної роботи №3

Варіант – 24

З дисципліни: «Кросплатформні засоби програмування»

На тему: «Спадкування та інтерфейси»

Виконав: ст. гр. КІ-301

Чоланюк О.Ю

Прийняв:

Майдан М.В.

Львів 2023

**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 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. 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.

# 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.

# 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». Синтаксис оголошення абстрактного методу:

[СпецифікаторДоступу] abstract Тип назваМетоду([параметри]);

# 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. Синтаксис оголошення інтерфейсів:

[public] interface НазваІнтерфейсу

{

Прототипи методів та оголошення констант інтерфейсу

}

**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 24: Спорядження військового альпініста

package MyPr1;  
  
import java.io.\*;  
import java.io.IOException;  
import java.io.FileWriter;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
public abstract class Main {  
 public ClimbingEquipment climbingEquipment;  
 public Position currentPosition;  
 public Carabiner rightCarabiner;  
 public Carabiner leftCarabiner;  
 public PrintWriter fout;  
 PrintWriter actionLogWriter;  
 public static final String *LOG\_FILE* = "Log.txt";  
 static final String *ACTION\_LOG\_FILE* = "ActionLog.txt";  
  
 public Main() throws IOException {  
 climbingEquipment = new ClimbingEquipment();  
 currentPosition = new Position();  
 rightCarabiner = new Carabiner();  
 leftCarabiner = new Carabiner();  
 fout = new PrintWriter(new File(*LOG\_FILE*));  
 actionLogWriter = new PrintWriter(new FileWriter(*ACTION\_LOG\_FILE*, true));  
  
 }  
  
 void logToFile(String message) {  
 SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
 String timestamp = dateFormat.format(new Date());  
  
 String logMessage = "[" + timestamp + "] " + message;  
  
 try (FileWriter writer = new FileWriter(*ACTION\_LOG\_FILE*, true)) {  
 writer.write(logMessage + "\n");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
  
 public abstract void setPosition(int xPosition, int yPosition);  
  
 public abstract void clickLeftButton();  
  
 public abstract void clickRightButton();  
  
 public abstract int getRightButtonResource();  
  
 public abstract int getLeftButtonResource();  
  
 public abstract void scrollUp();  
  
 public abstract void scrollDown();  
  
 public abstract void scrollNeutral();  
  
  
 public abstract ClimbingEquipment.Directions getScrollingDirection();  
  
}  
  
  
class ClimbingEquipment {  
 public void setDownDirection() {  
 }  
  
 enum Directions {*NEUTRAL*, *UP*, *DOWN*};  
 private Directions direction;  
  
 public ClimbingEquipment() {  
 direction = Directions.*NEUTRAL*;  
 }  
  
 public void setUpDirection() {  
 direction = Directions.*UP*;  
 }  
  
 public void setNeutralDirection() {  
 direction = Directions.*NEUTRAL*;  
 }  
  
 public Directions getDirection() {  
 return direction;  
 }  
}  
  
class Position {  
 private int x, y;  
  
 public Position() {  
 x = 0;  
 y = 0;  
 }  
  
 public int getXPosition() {  
 return x;  
 }  
  
 public int getYPosition() {  
 return y;  
 }  
  
 public void getXPosition(Position obj) {  
 obj.x = x;  
 obj.y = y;  
 }  
  
}  
  
class Carabiner {  
 private int btnResource;  
  
 public Carabiner() {  
 btnResource = 10;  
 }  
  
 public void clickButton() {  
 btnResource = btnResource - 1;  
 }  
  
 public int getButtonResource() {  
 return btnResource;  
 }  
}

package MyPr1;  
  
import java.io.IOException;  
  
  
public class App {  
 public static void main(String[] args) throws IOException {  
 MilitaryClimber climber = null;  
  
 ClimbingEquipment.Directions dir = null;  
 climber = new MilitaryClimber(0, 0.0);  
 climber.clickLeftButton();  
 System.*out*.print("Вісь x: " + climber.getLeftButtonResource() + "\n");  
 climber.clickRightButton();  
 System.*out*.print("Вісь y: " + climber.getRightButtonResource() + "\n");  
 climber.setPosition(0, 0);  
 climber.scrollDown();  
 climber.scrollNeutral();  
 climber.scrollUp();  
 dir = climber.getScrollingDirection();  
 if (dir == ClimbingEquipment.Directions.*DOWN*)  
 System.*out*.print("Down" + "\n");  
 else if (dir == ClimbingEquipment.Directions.*UP*)  
 System.*out*.print("Up" + "\n");  
 else  
 System.*out*.print("Neutral" + "\n");  
 System.*out*.println("Залишок маскувальних сіток: " + climber.getCamouflageDurability() + " одиниць ");  
 System.*out*.println("Залишок мотузки: " + climber.getRopeLength() + " м ");  
  
 }  
}

package MyPr1;  
  
public interface IntEq {  
 int getCamouflageDurability();  
 double getRopeLength();  
}

package MyPr1;  
  
import java.io.File;  
import java.io.IOException;  
import java.io.FileWriter;  
import java.io.PrintWriter;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class MilitaryClimber extends Main implements IntEq {  
 private int getCamouflageDurability;  
 private double getRopeLength;  
 private PrintWriter fout;  
 PrintWriter actionLogWriter;  
 private static final String *LOG\_FILE* = "Log.txt";  
 static final String *ACTION\_LOG\_FILE* = "ActionLog.txt";  
  
  
 public MilitaryClimber(int getCamouflageDurability, double getRopeLength) throws IOException {  
 this.getCamouflageDurability = getCamouflageDurability;  
 this.getRopeLength = getRopeLength;  
  
 fout = new PrintWriter(new File(*LOG\_FILE*));  
 actionLogWriter = new PrintWriter(new FileWriter(*ACTION\_LOG\_FILE*, true));  
 }  
 void logToFile(String message) {  
 SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
 String timestamp = dateFormat.format(new Date());  
  
 String logMessage = "[" + timestamp + "] " + message;  
  
 try (FileWriter writer = new FileWriter(*ACTION\_LOG\_FILE*, true)) {  
 writer.write(logMessage + "\n");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 @Override  
 public void setPosition(int xPosition, int yPosition) {  
 logToFile("Set climber position. X: " + xPosition + ", Y: " + yPosition);  
 fout.print("Set climber position. X: " + xPosition + ", Y: " + yPosition + "\n");  
 fout.flush();  
 }  
 @Override  
 public void clickLeftButton() {  
 leftCarabiner.clickButton();  
 logToFile("Clicked left carabiner button. New resource: " + leftCarabiner.getButtonResource());  
 fout.print("Clicked left carabiner button. New resource: " + leftCarabiner.getButtonResource() + "\n");  
 fout.flush();  
 }  
 @Override  
 public void clickRightButton() {  
 rightCarabiner.clickButton();  
 logToFile("Clicked right carabiner button. New resource: " + rightCarabiner.getButtonResource());  
 fout.print("Clicked right carabiner button. New resource: " + rightCarabiner.getButtonResource() + "\n");  
 fout.flush();  
 }  
 @Override  
 public int getRightButtonResource() {  
 return -4;  
 }  
 @Override  
 public int getLeftButtonResource() {  
 return -4;  
 }  
 @Override  
 public void scrollUp() {  
 climbingEquipment.setUpDirection();  
 logToFile("Scrolled up. Climbing equipment direction: " + climbingEquipment.getDirection());  
 fout.print("Scrolled up. Climbing equipment direction: " + climbingEquipment.getDirection() + "\n");  
 fout.flush();  
 }  
 @Override  
 public void scrollDown() {  
 climbingEquipment.setDownDirection();  
 logToFile("Scrolled down. Climbing equipment direction: " + climbingEquipment.getDirection());  
 fout.print("Scrolled down. Climbing equipment direction: " + climbingEquipment.getDirection() + "\n");  
 fout.flush();  
 }  
 @Override  
 public void scrollNeutral() {  
  
 }  
 @Override  
 public ClimbingEquipment.Directions getScrollingDirection() {  
 int rightResource = getRightButtonResource();  
 int leftResource = getLeftButtonResource();  
  
 if (rightResource == 0 && leftResource == 0) {  
 return ClimbingEquipment.Directions.*NEUTRAL*;  
 } else if (rightResource >= 0 && leftResource > 0 || rightResource > 0 || rightResource < 0 && leftResource >= 0) {  
 return ClimbingEquipment.Directions.*UP*;  
 } else if (rightResource < 0) {  
 return ClimbingEquipment.Directions.*DOWN*;  
 } else {  
 return ClimbingEquipment.Directions.*NEUTRAL*;  
 }  
 }  
 public int getCamouflageDurability() {  
 return 3;  
 }  
 public double getRopeLength() {  
 return 41.4;  
 }  
  
}