НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра «ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ ТА ПРОГРАМУВАННЯ»

Інженерія програмного забезпечування

Лабораторна робота №15

**Тема**: Анотації. Рефлексія. Шаблон Observer

Виконав:

студент групи КІТ-26В

Лаврєнов Д.В.

Перевірив:

Шевердін І.В.

Харків 2018

ЗМІСТ

Анотації. Рефлексія. Шаблон Observer 2

1 Індивідуальне завдання 2

2 Розробка програми 2

2.1 Структура проекту та опис програми 2

2.2 Код програми 3

3 Результат 12

Висновки 12

АНОТАЦІЇ. РЕФЛЕКСІЯ. ШАБЛОН OBSERVER

**Мета:**

* придбання навичок використання засобів анотування;
* ознайомлення з механізмом рефлексії;
* реалізація обслуговування колекції об’єктів на основі шаблону проектування Observer;
* використання модульного тестування.

**1 Індивідуальне завдання:**

Розробити ієрархію класів згідно шаблону Observer та продемонструвати можливість обробки розробленої раніше колекції. При реалізації використовувати анотації.

**2 Розробка програми**

**2.1 Опис програми та структура проекту:**

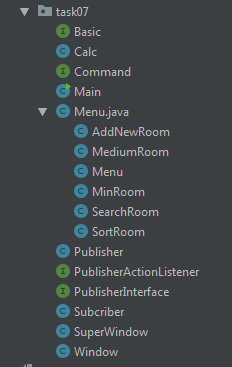


Рисунок 1 – структура проекту

Опис класів:

Клас Main створює потрібні об`єкти та демонструє роботу програми.

Клас Calc виконує обчислення площі, периметру вікон, SuperWindow – суперклас, який містить в собі потрібні дані, Window – наслідник, який спадкує всі дані від класу Window.

Клас Menu реалізує шаблон проектування Command

Класи Publisher та Subscriber реалізують «письменника та читача», PublisherInterface та PublisherActionListener реалізують методи для правильного виконання роботи шаблону проектування Observer.

**2.2 Програма**

файл Basic.java

package task7;  
  
public interface Basic {  
 String Perimeter(int a , int b);  
 String Volume(int a , int b , int c);  
 String Square(int a , int b);  
}

файл Calc.java

package task7;  
  
  
import java.io.Serializable;  
import java.util.Formatter;  
  
public class Calc implements Serializable, Basic {  
  
 private String perimeter;  
 private String square;  
 private String volume;  
  
 @Override  
 public String toString() {  
 Formatter fmt = new Formatter();  
 fmt.format("%-10.9s|%-15.10s|%-15.10s|", perimeter, square, volume);  
 return fmt.toString();  
 }  
  
 public String Perimeter(int a, int b) {  
 perimeter = (2 \* (a + b) + "");  
 return perimeter;  
 }  
  
 public String Volume(int a, int b, int c) {  
 volume = (a \* b \* c + "");  
 return volume;  
 }  
  
 public String Square(int a, int b) {  
 square = (a \* b + "");  
 return square;  
 }  
  
 public String Square(int a) {  
 square = (Math.*round*(Math.*PI* \* Math.*pow*(a, 2)) + "");  
 return square;  
 }  
  
 public String getSquare() {  
 return square;  
 }  
  
  
}

файл Window.java

package task7;  
  
  
import java.io.Serializable;  
  
public class Window extends SuperWindow implements Serializable {  
 private Calc calc;  
  
 public Window(int width, int length, int height) {  
 super(width, length, height);  
 }  
  
 public Window() {  
  
 }  
  
 public Calc getCalc() {  
 return calc;  
 }  
  
 public void setCalc(Calc calc) {  
 this.calc = calc;  
 }  
  
 @Override  
 public String PolimorphMethod(){  
 return "Window";  
 }  
  
 @Override  
 public String toString() {  
 return "Window{" +  
 "calc=" + calc +  
 '}';  
 }  
}

файл SuperWindow.java

package task7;  
  
  
import java.io.Serializable;  
  
  
public class SuperWindow implements Serializable {  
  
 private Calc calc;  
 private int width;  
 private int length;  
 private int height;  
  
 public SuperWindow(int width, int length, int height) {  
  
 this.width = width;  
 this.length = length;  
 this.height = height;  
 }  
  
 public SuperWindow() {  
 }  
  
 public int getHeight() {  
  
 return height;  
 }  
  
 public int getLength() {  
  
 return length;  
 }  
  
 public int getWidth() {  
  
 return width;  
 }  
  
 public Calc getCalc() {  
  
 return calc;  
 }  
  
 public void setCalc(Calc calc) {  
  
 this.calc = calc;  
 }  
  
 public String PolimorphMethod(){  
 return "SuperWindow";  
 }  
  
}

файл Main.java

package task07;  
  
import java.lang.reflect.Field;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Main {  
 private final static Main *main* = new Main();  
 public static Main getInstance(){  
 return *main*;  
 }  
  
 static List<Window> *windowList* = new ArrayList<>();  
  
 public static void init() {  
 for (int i = 0; i < 5; i++) {  
  
 Window window = new Window((int) (Math.*random*() \* 30), (int) (Math.*random*() \* 30), (int) (Math.*random*() \* 3 + 1));  
 Calc calc = new Calc();  
 calc.Perimeter(window.getWidth(), window.getLength());  
 calc.Square(window.getWidth(), window.getLength());  
 calc.Volume(window.getWidth(), window.getLength(), window.getHeight());  
 window.setCalc(calc);  
 *windowList*.add(window);  
 }  
 }  
 public static void main(String[] args) throws NoSuchFieldException, IllegalAccessException {  
  
 System.*out*.println(new SuperWindow().PolimorphMethod());  
 System.*out*.println(new Window().PolimorphMethod());  
  
 Menu menu = new Menu();  
 *main*.*init*();  
  
 for (Window circleRoom : *windowList*)  
 System.*out*.println(circleRoom);  
 System.*out*.println("//////////////////////////");  
  
 AddNewRoom addNewRoom = new AddNewRoom();  
 addNewRoom.setWindowList(*windowList*);  
  
 MediumRoom mediumRoom = new MediumRoom();  
 mediumRoom.setWindowList(*windowList*);  
  
 SortRoom sortRoom = new SortRoom();  
 sortRoom.setWindowList(*windowList*);  
  
 SearchRoom searchRoom = new SearchRoom();  
 searchRoom.setWindowList(*windowList*);  
 searchRoom.setWindow(*windowList*.get(*windowList*.size() - 1));  
  
 MinRoom minRoom = new MinRoom();  
 minRoom.setWindowList(*windowList*);  
  
 menu.addCommand(addNewRoom);  
 menu.addCommand(mediumRoom);  
 menu.addCommand(searchRoom);  
 menu.addCommand(minRoom);  
 menu.addCommand(sortRoom);  
  
 new Thread(() -> {  
 for (int i = 0; i < 5; i++) {  
 menu.execute(menu.getCommandList().get(i));  
 }  
 }).start();  
 System.*out*.println("\n\n\n////////////////////////////////////////////////////////////////////////");  
  
 Publisher publisher = new Publisher();  
 publisher.addListener(new Subcriber());  
 publisher.addListener(new Subcriber());  
  
 publisher.createNewMessage("Message!!!!!");  
  
 Class classPublisher = Publisher.class;  
 Field field = classPublisher.getDeclaredField("str");  
 field.setAccessible(true);  
 System.*out*.println(field.get(publisher));  
 }  
}

файл Menu.java

package task7;  
  
  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.List;  
  
public class Menu implements Queue{  
  
 List<Command> commandList = new ArrayList<>();  
 private boolean waiting;  
 private boolean shutdown;  
  
 Menu(){  
 waiting = false;  
 }  
  
 void addCommand(Command command) {  
 commandList.add(command);  
 }  
  
 public List<Command> getCommandList() {  
 return commandList;  
 }  
  
 void execute() {  
 for (Command command : commandList) {  
 command.execute();  
 System.*out*.println("\n\n/////////////////////////");  
 }  
 }  
 void execute(Command command){  
 command.execute();  
 System.*out*.println("\n\n/////////////////////////");  
 }  
  
  
 @Override  
 public void put(Command cmd) {  
 commandList.add(cmd);  
 if(waiting){  
 synchronized (this){  
 notifyAll();  
 }  
 }  
 }  
  
 @Override  
 public Command take() {  
 if(commandList.isEmpty()){  
 synchronized (this){  
 waiting = true;  
 try{  
 wait();  
 } catch (InterruptedException e) {  
 waiting = false;  
 }  
 }  
 }  
 return (Command)commandList.remove(0);  
 }  
  
 private class Worker implements Runnable{  
  
 @Override  
 public void run() {  
 while(!shutdown){  
 Command c = take();  
 c.execute();  
 }  
 }  
 }  
}  
class AddNewRoom implements Command {  
  
 List<Window> windowList = new ArrayList<>();  
  
 public List<Window> getWindowList() {  
 return windowList;  
 }  
  
 public void setWindowList(List<Window> windowList) {  
 this.windowList = windowList;  
 }  
  
 @Override  
 public void execute() {  
 windowList.add(windowList.get(0));  
 System.*out*.println("added");  
 }  
  
}  
  
class MinRoom implements Command {  
  
 List<Window> windowList = new ArrayList<>();  
  
 public List<Window> getWindowList() {  
 return windowList;  
 }  
  
  
  
 public void setWindowList(List<Window> windowList) {  
 this.windowList = windowList;  
 }  
  
 @Override  
 public void execute() {  
 Window minCircleRoom = windowList.get(0);  
 for(Window circleRoom:windowList){  
 if(minCircleRoom.getHeight() > circleRoom.getHeight())  
 minCircleRoom = circleRoom;  
 }  
 System.*out*.println("min room - " + minCircleRoom);  
  
 }  
  
}  
  
class MediumRoom implements Command {  
  
 List<Window> windowList = new ArrayList<>();  
  
 public List<Window> getWindowList() {  
 return windowList;  
 }  
  
 public void setWindowList(List<Window> windowList) {  
 this.windowList = windowList;  
 }  
  
 @Override  
 public void execute() {  
  
 int mediumSquare = 0;  
 for (Window circleRoom : windowList) {  
 mediumSquare +=  
 Integer.*parseInt*(  
 circleRoom  
 .getCalc()  
 .getSquare());  
 }  
 mediumSquare /= windowList.size();  
 System.*out*.println("medium square = " + mediumSquare);  
 }  
  
}  
  
class SortRoom implements Command {  
  
 Window window;  
 List<Window> windowList = new ArrayList<>();  
  
 public List<Window> getWindowListList() {  
 return windowList;  
 }  
  
 public void setWindowList(List<Window> windowList) {  
 this.windowList = windowList;  
 }  
  
 public void setWindow(Window room) {  
 this.window = window;  
 }  
  
 @Override  
 public void execute() {  
 Collections.*sort*(windowList,  
 Comparator.*comparingInt*(  
 (first) ->  
 Integer.*parseInt*(first  
 .getCalc()  
 .getSquare()  
 )  
 )  
 );  
 System.*out*.println("Sort");  
 for (Window window : windowList)  
 System.*out*.println(window);  
 // (first, second)-> Integer.parseInt(first.getCalc().getSquare()) - Integer.parseInt(second.getCalc().getSquare()));  
 }  
  
}  
  
class SearchRoom implements Command {  
 Window room;  
 List<Window> windowList = new ArrayList<>();  
  
 public void setWindow(Window room) {  
 this.room = room;  
 }  
  
 public List<Window> getWindowList() {  
 return windowList;  
 }  
  
 public void setWindowList(List<Window> windowList) {  
 this.windowList = windowList;  
 }  
  
 @Override  
 public void execute() {  
 System.*out*.println("search - " + windowList.get(windowList.indexOf(room)));  
 }  
}

файл Queue.java

package task7;  
  
public interface Queue {  
 void put(Command cmd);  
  
 Command take();  
}

файл Command.java

package task7;  
  
public interface Command {  
 void execute();  
}

файл Subscriber.java

package task07;  
  
public class Subcriber implements PublisherActionListener {  
 static int *i* = 1;  
 @Override  
 public void doAction(String message) {  
 System.*out*.println(message + " from " + this.getClass().getName() + *i*++);  
 }  
}

файл PublisherActionListener.java

package task07;  
  
public interface PublisherActionListener {  
 void doAction(String message);  
  
}

файл PublisherInterface.java

package task07;  
  
public interface PublisherInterface {  
 void addListener(PublisherActionListener listener);  
 void removeListener(PublisherActionListener listener);  
 void removeAllListener();  
 void notifySubscribers(String message);  
  
}

файл Publisher.java

package task07;  
  
import java.util.ArrayList;  
  
public class Publisher implements PublisherInterface{  
  
 private String str = "text";  
 private ArrayList<PublisherActionListener> listeners = new ArrayList<>();  
  
 @Override  
 public void addListener(PublisherActionListener listener) {  
 listeners.add(listener);  
 }  
  
 @Override  
 public void removeListener(PublisherActionListener listener) {  
 listeners.remove(listener);  
 }  
  
 @Override  
 public void removeAllListener() {  
 listeners.clear();  
 }  
  
 @Override  
 public void notifySubscribers(String message) {  
 for(PublisherActionListener actionListener: listeners){  
 actionListener.doAction(message);  
 }  
 }  
  
 public void createNewMessage(String message){  
 System.*out*.println("Publisher printed message:\n" + message);  
 notifySubscribers(message);  
 }  
}

**3 Результат**

Результат роботи програми у вигляді результату проходження тестів наведено на рисунку 1.

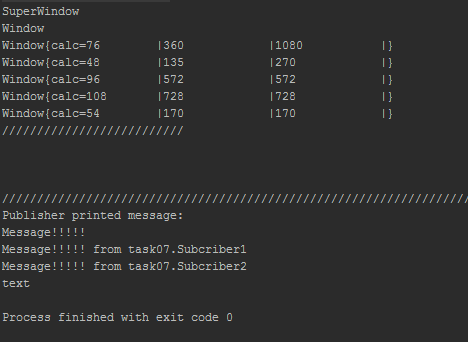


Рисунок 2 – Результат роботи програми

**Висновки:**

У цій лабораторній роботі ознайомились з анотаціями, рефлексією, та з використанням паттерна Observer.