Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра экономической информатики

**Лабораторная работа № 8**

«Паттерны проектирования. VCS»

по курсу

«Сетевые информационные технологии»

Выполнил: Проверила:

студент группы 172304

Петрович Н.О Волошко Е.А.

Минск 2013 г.

**1 Паттерн Комманда**

Команда (англ. Command) — поведенческий шаблон проектирования, используемый при объектно-ориентированном программировании, представляющий действие. Объект команды заключает в себе само действие и его параметры.

Структура данного паттерна представлена на рис.1



Рис. 1 – Структура паттерна Комманда

**2 Реализацияпаттерна**

Пакетcom.patterns.command.interfaces

public interface Command {

public void execute();

public void undo();

}

importcom.patterns.command.interfaces.Command;

public class MacroCommand implements Command {

Command[] commands;

publicMacroCommand(Command[] commands) {

this.commands = commands;

}

public void execute() {

for (int i = 0; i <commands.length; i++) {

commands[i].execute();

}

}

public void undo() {

for (int i = 0; i <commands.length; i++) {

commands[i].undo();

}

}

}

public class NoCommand implements Command {

public void execute() {}

public void undo() {}

}

Пакетcom.patterns.command.objects.CeilingFan

packagecom.patterns.command.objects.CeilingFan;

public class CeilingFan {

String name;

int speed;

public static final int HIGH = 3;

public static final int MEDIUM = 2;

public static final int LOW = 1;

public static final int OFF = 0;

CeilingFan(String name) {

this.name = name;

}

public void high() {

System.out.println(name + " CeilingFan is HIGH");

speed = HIGH;

}

public void medium() {

System.out.println(name + " CeilingFan is MEDIUM");

speed = MEDIUM;

}

public void low() {

System.out.println(name + " CeilingFan is LOW");

speed = LOW;

}

public void on() {

System.out.println(name + " CeilingFan on");

}

public void off() {

System.out.println(name + " CeilingFan is OFF");

speed = OFF;

}

publicintgetSpeed() {

return speed;

}

}

packagecom.patterns.command.objects.CeilingFan;

importcom.patterns.command.interfaces.Command;

public class CeilingFanHighCommand implements Command {

CeilingFanceilingFan;

intprevSpeed;

publicCeilingFanHighCommand(CeilingFanceilingFan) {

this.ceilingFan = ceilingFan;

}

public void execute() {

prevSpeed = ceilingFan.getSpeed();

ceilingFan.high();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

}

if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

}

if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

}

if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

packagecom.patterns.command.objects.CeilingFan;

importcom.patterns.command.interfaces.Command;

public class CeilingFanLowCommand implements Command {

CeilingFanceilingFan;

intprevSpeed;

publicCeilingFanLowCommand(CeilingFanceilingFan) {

this.ceilingFan = ceilingFan;

}

public void execute() {

prevSpeed = ceilingFan.getSpeed();

ceilingFan.low();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

}

if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

}

if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

}

if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

packagecom.patterns.command.objects.CeilingFan;

importcom.patterns.command.interfaces.Command;

public class CeilingFanMediumCommand implements Command {

CeilingFanceilingFan;

intprevSpeed;

publicCeilingFanMediumCommand(CeilingFanceilingFan) {

this.ceilingFan = ceilingFan;

}

public void execute() {

prevSpeed = ceilingFan.getSpeed();

ceilingFan.medium();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

}

if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

}

if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

}

if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

packagecom.patterns.command.objects.CeilingFan;

importcom.patterns.command.interfaces.Command;

public class CeilingFanOffCommand implements Command {

CeilingFanceilingFan;

intprevSpeed;

publicCeilingFanOffCommand(CeilingFanceilingFan) {

this.ceilingFan = ceilingFan;

}

public void execute() {

prevSpeed = ceilingFan.getSpeed();

ceilingFan.off();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

}

if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

}

if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

}

if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

packagecom.patterns.command.objects.CeilingFan;

importcom.patterns.command.interfaces.Command;

public class CeilingFanOnCommand implements Command {

CeilingFanceilingFan;

publicCeilingFanOnCommand(CeilingFanceilingFan) {

this.ceilingFan = ceilingFan;

}

public void execute() {

ceilingFan.on();

}

public void undo() {

ceilingFan.off();

}

}

Пакетcom.patterns.command.objects.GarageDoor

packagecom.patterns.command.objects.GarageDoor;

public class GarageDoor {

String name;

GarageDoor(String name) {

this.name = name;

}

public void up() {

System.out.println(name + " Door is up!");

}

public void down() {

System.out.println(name + " Door is down!");

}

public void stop() {

System.out.println(name + " Door is stopped!");

}

public void lightOn() {

System.out.println(name + " Light is on!");

}

public void lightOff() {

System.out.println(name + " Light is off!");

}

}

packagecom.patterns.command.objects.GarageDoor;

importcom.patterns.command.interfaces.Command;

public class GarageDoorDownCommand implements Command {

GarageDoorgarageDoor;

publicGarageDoorDownCommand(GarageDoorgarageDoor) {

this.garageDoor = garageDoor;

}

public void execute() {

garageDoor.down();

}

public void undo() {

garageDoor.up();

}

}

packagecom.patterns.command.objects.GarageDoor;

importcom.patterns.command.interfaces.Command;

public class GarageDoorUpCommand implements Command {

GarageDoorgarageDoor;

publicGarageDoorUpCommand(GarageDoorgarageDoor) {

this.garageDoor = garageDoor;

}

public void execute() {

garageDoor.up();

}

public void undo() {

garageDoor.down();

}

}

Пакетcom.patterns.command.objects.Hottub

packagecom.patterns.command.objects.Hottub;

public class Hottub {

publicHottub() {}

public void on() {

System.out.println("Hottub on");

}

public void off() {

System.out.println("Hottub off");

}

}

packagecom.patterns.command.objects.Hottub;

importcom.patterns.command.interfaces.Command;

public class HottubOffCommand implements Command {

Hottubhottub;

publicHottubOffCommand(Hottubhottub) {

this.hottub = hottub;

}

public void execute() {

hottub.off();

}

public void undo() {

hottub.on();

}

}

packagecom.patterns.command.objects.Hottub;

importcom.patterns.command.interfaces.Command;

public class HottubOnCommand implements Command {

Hottubhottub;

publicHottubOnCommand(Hottubhottub) {

this.hottub = hottub;

}

public void execute() {

hottub.on();

}

public void undo() {

hottub.off();

}

}

Пакетcom.patterns.command.objects.Light

packagecom.patterns.command.objects.Light;

public class Light {

String name;

public Light(String name) {

this.name = name;

}

public void on() {

System.out.println(name + " Light on");

}

public void off() {

System.out.println(name + " Light off");

}

}

packagecom.patterns.command.objects.Light;

importcom.patterns.command.interfaces.Command;

public class LightOffCommand implements Command {

Light light;

publicLightOffCommand(Light light) {

this.light = light;

}

public void execute() {

light.off();

}

public void undo() {

light.on();

}

}

packagecom.patterns.command.objects.Light;

importcom.patterns.command.interfaces.Command;

public class LightOnCommand implements Command {

Light light;

publicLightOnCommand(Light light) {

this.light = light;

}

public void execute() {

light.on();

}

public void undo() {

light.off();

}

}

Пакетcom.patterns.command.objects.Remotes

packagecom.patterns.command.objects.Remotes;

importcom.patterns.command.interfaces.Command;

importcom.patterns.command.interfaces.NoCommand;

public class RemoteControlWithUndo {

Command[] onCommands;

Command[] offCommands;

Command undoCommand;

publicRemoteControlWithUndo() {

onCommands = new Command[7];

offCommands = new Command[7];

Command noCommand = new NoCommand();

for (int i = 0; i < 7; i++) {

onCommands[i] = noCommand;

offCommands[i] = noCommand;

}

undoCommand = noCommand;

}

public void setCommand (int slot, Command onCommand, Command offCommand) {

onCommands[slot] = onCommand;

offCommands[slot] = offCommand;

}

public void onButtonWasPushed (int slot) {

onCommands[slot].execute();

undoCommand = onCommands[slot];

}

public void offButtonWasPushed (int slot) {

offCommands[slot].execute();

undoCommand = offCommands[slot];

}

public void undoButtonWasPushed() {

undoCommand.undo();

}

public String toString() {

StringBufferstringBuff = new StringBuffer();

stringBuff.append("\n------ Remote Control ------\n");

for (int i = 0; i <onCommands.length; i++) {

stringBuff.append("[slot " + i + "] " + onCommands[i].getClass().getName() + " " + offCommands[i].getClass().getName() + "\n");

}

returnstringBuff.toString();

}

}

packagecom.patterns.command.objects.Remotes;

importcom.patterns.command.interfaces.MacroCommand;

importcom.patterns.command.objects.TV.TVOffCommand;

import com.patterns.command.objects.TV.TV;

importcom.patterns.command.objects.TV.TVOnCommand;

importcom.patterns.command.objects.Stereo.StereoOnCommand;

importcom.patterns.command.objects.Stereo.StereoOffCommand;

importcom.patterns.command.objects.Stereo.Stereo;

importcom.patterns.command.objects.Light.Light;

importcom.patterns.command.objects.Light.LightOnCommand;

importcom.patterns.command.objects.Light.LightOffCommand;

importcom.patterns.command.objects.Hottub.Hottub;

importcom.patterns.command.objects.Hottub.HottubOnCommand;

importcom.patterns.command.objects.Hottub.HottubOffCommand;

importcom.patterns.command.interfaces.Command;

public class RemoteLoader {

public static void main(String[] args) {

RemoteControlWithUndoremoteControl = new RemoteControlWithUndo();

Light light = new Light("Living room");

TV tv = new TV("Living room");

Stereo stereo = new Stereo("Living room");

Hottubhottub = new Hottub();

LightOnCommandlightOn = new LightOnCommand(light);

StereoOnCommandstereoOn = new StereoOnCommand(stereo);

TVOnCommandtvOn = new TVOnCommand(tv);

HottubOnCommandhottubOn = new HottubOnCommand(hottub);

LightOffCommandlightOff = new LightOffCommand(light);

StereoOffCommandstereoOff = new StereoOffCommand(stereo);

TVOffCommandtvOff = new TVOffCommand(tv);

HottubOffCommandhottubOff = new HottubOffCommand(hottub);

Command[] partyOn = { lightOn, stereoOn, tvOn, hottubOn };

Command[] partyOff = { lightOff, stereoOff, tvOff, hottubOff };

MacroCommandpartyOnMacro = new MacroCommand(partyOn);

MacroCommandpartyOffMacro = new MacroCommand(partyOff);

remoteControl.setCommand(0, partyOnMacro, partyOffMacro);

System.out.println(remoteControl);

System.out.println("--- Pushing Macro ON ---");

remoteControl.onButtonWasPushed(0);

System.out.println("--- Pushing Macro OFF ---");

remoteControl.offButtonWasPushed(0);

}

}

Пакетcom.patterns.command.objects.Stereo

packagecom.patterns.command.objects.Stereo;

public class Stereo {

String name;

public Stereo(String name) {

this.name = name;

}

public void on() {

System.out.println(name + " Stereo on!");

}

public void off() {

System.out.println(name + " Stereo off!");

}

public void setCd() {

System.out.println(name + " Stereo setCd!");

}

public void setDvd() {

System.out.println(name + " Stereo setDvd!");

}

public void setRadio() {

System.out.println(name + " Stereo setRadio!");

}

public void setVolume(int volume) {

System.out.println(name + " Stereo setVolume is set to " + volume);

}

}

packagecom.patterns.command.objects.Stereo;

importcom.patterns.command.interfaces.Command;

public class StereoOffCommand implements Command{

Stereo stereo;

publicStereoOffCommand(Stereo stereo) {

this.stereo = stereo;

}

public void execute() {

stereo.off();

}

public void undo() {

stereo.on();

}

}

packagecom.patterns.command.objects.Stereo;

importcom.patterns.command.interfaces.Command;

public class StereoOnCommand implements Command{

Stereo stereo;

publicStereoOnCommand(Stereo stereo) {

this.stereo = stereo;

}

public void execute() {

stereo.on();

}

public void undo() {

stereo.off();

}

}

packagecom.patterns.command.objects.Stereo;

importcom.patterns.command.interfaces.Command;

public class StereoOnWithCDCommand implements Command {

Stereo stereo;

publicStereoOnWithCDCommand(Stereo stereo) {

this.stereo = stereo;

}

public void execute() {

stereo.on();

stereo.setCd();

stereo.setVolume(11);

}

public void undo() {

stereo.off();

}

}

Пакет com.patterns.command.objects.TV

package com.patterns.command.objects.TV;

public class TV {

String name;

public TV(String name) {

this.name = name;

}

public void on() {

System.out.println(name + " TV on");

}

public void off() {

System.out.println(name + " TV off");

}

}

package com.patterns.command.objects.TV;

importcom.patterns.command.interfaces.Command;

public class TVOffCommand implements Command {

TV tv;

publicTVOffCommand(TV tv) {

this.tv = tv;

}

public void execute() {

tv.off();

}

public void undo() {

tv.on();

}

}

package com.patterns.command.objects.TV;

importcom.patterns.command.interfaces.Command;

public class TVOnCommand implements Command {

TV tv;

publicTVOnCommand(TV tv) {

this.tv = tv;

}

public void execute() {

tv.on();

}

public void undo() {

tv.off();

}

}

**3 Система управления версиями**

Система управления версиями (от англ. VersionControlSystem, VCS или RevisionControlSystem) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

В данной лабораторной работе был использован GitHub.

GitHub — самый крупный веб-сервис для хостинга IT-проектов и их совместной разработки.Основан на системе контроля версий Git и разработан на RubyonRails и Erlang компанией GitHub, Inc (ранее LogicalAwesome).