Task - 1

interface Command {

void execute();

void undo();

}

interface Light {

void on();

void off();

void dim(int level);

}

class KitchenRoomLight implements Light {

@Override

public void on() {

System.*out*.println("Kitchen Room Light is on and full brightness");

}

@Override

public void off() {

System.*out*.println("Kitchen Room Light is off");

}

@Override

public void dim(int level) {

System.*out*.println("Kitchen Room Light is dimmed to " + level + "%");

}

}

class LivingRoomLight implements Light {

@Override

public void on() {

System.*out*.println("Living Room Light is on and full brightness");

}

@Override

public void off() {

System.*out*.println("Living Room Light is off");

}

@Override

public void dim(int level) {

System.*out*.println("Living Room Light is dimmed to " + level + "%");

}

}

class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.on();

}

@Override

public void undo() {

light.off();

}

}

class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.off();

}

@Override

public void undo() {

light.on();

}

}

class LightDimCommand implements Command {

private Light light;

private int level;

public LightDimCommand(Light light, int level) {

this.light = light;

this.level = level;

}

@Override

public void execute() {

light.dim(level);

}

@Override

public void undo() {

light.dim(100);

}

}

class RemortCommand{

private Command onCommand;

private Command offCommand;

private int slot;

public RemortCommand(int slot, Command onCommand, Command offCommand) {

this.slot = slot;

this.onCommand = onCommand;

this.offCommand = offCommand;

}

public void onButtonPressed(){

onCommand.execute();

}

public void offButtonPressed(){

offCommand.execute();

}

public void undoButtonPressed(){

onCommand.undo();

}

}

class RemoteControl {

private final ArrayList<RemortCommand> onCommands;

public RemoteControl() {

onCommands = new ArrayList<>();

}

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

onCommands.add(new RemortCommand(slot, onCommand, offCommand));

}

public void onButtonPressed(int slot){

onCommands.get(slot).onButtonPressed();

}

public void offButtonPressed(int slot){

onCommands.get(slot).offButtonPressed();

}

public void undoButtonPressed(int slot) {

onCommands.get(slot).undoButtonPressed();

}

}

public class Example {

public static void main(String[] args) {

RemoteControl remoteControl = new RemoteControl();

Light kitchenRoomLight = new KitchenRoomLight();

Light livingRoomLight = new LivingRoomLight();

Command kitchenRoomLightOnCommand = new LightOnCommand(kitchenRoomLight);

Command kitchenRoomLightOffCommand = new LightOffCommand(kitchenRoomLight);

Command kitchenRoomLightDimCommand = new LightDimCommand(kitchenRoomLight, 50);

Command livingRoomLightOnCommand = new LightOnCommand(livingRoomLight);

Command livingRoomLightOffCommand = new LightOffCommand(livingRoomLight);

Command livingRoomLightDimCommand = new LightDimCommand(livingRoomLight, 50);

remoteControl.setCommand(0, kitchenRoomLightOnCommand, kitchenRoomLightOffCommand);

remoteControl.setCommand(1, livingRoomLightOnCommand, livingRoomLightOffCommand);

remoteControl.onButtonPressed(0);

remoteControl.offButtonPressed(0);

remoteControl.undoButtonPressed(0);

remoteControl.onButtonPressed(1);

remoteControl.offButtonPressed(1);

remoteControl.undoButtonPressed(1);

kitchenRoomLightDimCommand.execute();

kitchenRoomLightDimCommand.undo();

livingRoomLightDimCommand.execute();

livingRoomLightDimCommand.undo();

}

}

Task - 2

package lk.drivamate.backend.prf;

import java.util.Scanner;

abstract class Beverages {

boolean wantsExtras;

abstract void brew();

abstract void addCondiments();

public void setWantsExtras(boolean wantsExtras) {

this.wantsExtras = wantsExtras;

}

final void finalTemplateMethod() {

boilWater();

brew();

pourInCup();

addCondiments();

if (wantsExtras) {

addExtras();

}

}

void boilWater() {

System.*out*.println("Boiling water...");

}

void pourInCup() {

System.*out*.println("Pouring in cup...");

}

abstract void addExtras();

}

class Tea extends Beverages {

@Override

void brew() {

System.*out*.println("Steeping tea...");

}

@Override

void addCondiments() {

System.*out*.println("Adding lemon and sugar...");

}

@Override

void addExtras() {

System.*out*.println("Adding honey...");

}

}

class Coffee extends Beverages {

@Override

void brew() {

System.*out*.println("Grinding coffee beans and brewing...");

}

@Override

void addCondiments() {

System.*out*.println("Adding milk and sugar...");

}

@Override

void addExtras() {

System.*out*.println("Adding whipped cream...");

}

}

public class Example {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.*in*);

System.*out*.print("Do you want extras for tea? (yes/no): ");

boolean wantTeaExtras = scanner.nextLine().equalsIgnoreCase("yes");

System.*out*.print("Do you want extras for coffee? (yes/no): ");

boolean wantCoffeeExtras = scanner.nextLine().equalsIgnoreCase("yes");

Tea tea = new Tea();

tea.setWantsExtras(wantTeaExtras);

tea.finalTemplateMethod();

Coffee coffee = new Coffee();

coffee.setWantsExtras(wantCoffeeExtras);

coffee.finalTemplateMethod();

}

}

Task - 3

package lk.drivamate.backend.prf;

interface Shape {

void draw();

}

class Circle implements Shape {

private int radius = 10;

@Override

public void draw() {

int r = radius;

int diameter = 2 \* r + 1;

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

for (int j = 0; j < diameter; j++) {

int x = i - r;

int y = j - r;

if (x \* x + y \* y <= r \* r + r \* 0.8) {

System.*out*.print("\* ");

} else {

System.*out*.print(" ");

}

}

System.*out*.println();

}

}

}

class Rectangle implements Shape {

@Override

public void draw() {

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 10; j++) {

System.*out*.print("\* ");

}

System.*out*.println();

}

}

}

class Square implements Shape {

@Override

public void draw() {

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

System.*out*.print("\* ");

}

System.*out*.println();

}

}

}

class Triangle implements Shape {

@Override

public void draw() {

System.*out*.println("Triangle::draw()");

}

}

class ShapeFactory {

public Shape getShape(String shapeType) {

if (shapeType == null) {

return null;

}

if (shapeType.equalsIgnoreCase("CIRCLE")) {

return new Circle();

} else if (shapeType.equalsIgnoreCase("RECTANGLE")) {

return new Rectangle();

} else if (shapeType.equalsIgnoreCase("SQUARE")) {

return new Square();

} else if (shapeType.equalsIgnoreCase("TRIANGLE")) {

return new Triangle();

}

return null;

}

}

public class Example {

public static void main(String[] args) {

ShapeFactory shapeFactory = new ShapeFactory();

Shape aquare = shapeFactory.getShape("SQUARE");

if (aquare != null) {

System.*out*.println("SQURE");

aquare.draw();

System.*out*.println();

}

Shape rectangle = shapeFactory.getShape("RECTANGLE");

if (rectangle != null) {

System.*out*.println("RECTANGLE");

rectangle.draw();

System.*out*.println();

}

Shape circle = shapeFactory.getShape("CIRCLE");

if (circle != null) {

System.*out*.println("CIRCLE");

circle.draw();

System.*out*.println();

}

Shape invalidShape = shapeFactory.getShape("PENTAGON");

if (invalidShape != null) {

invalidShape.draw();

}

}

}