Software Architecture and Design Patterns

Practical Assignments

Assignment no : 1

Name : Sonukumar C. Thakur

Roll no : 7649 Class : S.Y.MSc(C.S)

Batch: Date:

Pracrical No:

**Q)** Write a JAVA Program to implement built-in support (java.util.Observable) Weather station with members temperature, humidity, pressure and methods mesurmentsChanged(), setMesurment(), getTemperature(), getHumidity(), getPressure()

**Solution :**

**WeatherStationHeatIndex.java** main file outside the package.

import weatherobservable.\*;

public class WeatherStationHeatIndex

{

public static void main(String[] args)

{

WeatherData weatherData = new WeatherData();

CurrentConditionsDisplay currentConditions = new CurrentConditionsDisplay(weatherData);

StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);

ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);

HeatIndexDisplay heatIndexDisplay = new HeatIndexDisplay(weatherData);

weatherData.setMeasurements(80, 65, 30.4f);

weatherData.setMeasurements(82, 70, 29.4f);

weatherData.setMeasurements(78, 90, 29.2f);

}

}

Create Package **weatherobservable** and inside that write below files :

**a) CurrentConditionsDisplay.java**

package weatherobservable;

import java.util.Observable;

import java.util.Observer;

public class CurrentConditionsDisplay implements Observer, DisplayElement

{

Observable observable;

private float temperature;

private float humidity;

public CurrentConditionsDisplay(Observable observable)

{

this.observable = observable;

observable.addObserver(this);

}

public void update(Observable obs, Object arg)

{

if (obs instanceof WeatherData)

{

WeatherData weatherData = (WeatherData)obs;

this.temperature = weatherData.getTemperature();

this.humidity = weatherData.getHumidity();

display();

}

}

public void display()

{

System.out.println("Current conditions: " + temperature

+ "F degrees and " + humidity + "% humidity");

}

}

**b) DisplayElement.java**

package weatherobservable;

public interface DisplayElement {

public void display();

}

**c) ForecastDisplay.java**

package weatherobservable;

import java.util.Observable;

import java.util.Observer;

public class ForecastDisplay implements Observer, DisplayElement

{

private float currentPressure = 29.92f;

private float lastPressure;

public ForecastDisplay(Observable observable)

{

observable.addObserver(this);

}

public void update(Observable observable, Object arg)

{

if (observable instanceof WeatherData)

{

WeatherData weatherData = (WeatherData)observable;

lastPressure = currentPressure;

currentPressure = weatherData.getPressure();

display();

}

}

public void display()

{

System.out.print("Forecast: ");

if (currentPressure > lastPressure)

{

System.out.println("Improving weather on the way!");

} else if (currentPressure == lastPressure)

{

System.out.println("More of the same");

} else if (currentPressure < lastPressure)

{

System.out.println("Watch out for cooler, rainy weather");

}

}

}

**d) HeatIndexDisplay.java**

package weatherobservable;

import java.util.Observable;

import java.util.Observer;

public class HeatIndexDisplay implements Observer, DisplayElement

{

float heatIndex = 0.0f;

public HeatIndexDisplay(Observable observable)

{

observable.addObserver(this);

}

public void update(Observable observable, Object arg)

{

if (observable instanceof WeatherData)

{

WeatherData weatherData = (WeatherData)observable;

float t = weatherData.getTemperature();

float rh = weatherData.getHumidity();

heatIndex = (float)

(

(16.923 + (0.185212 \* t)) +

(5.37941 \* rh) -

(0.100254 \* t \* rh) +

(0.00941695 \* (t \* t)) +

(0.00728898 \* (rh \* rh)) +

(0.000345372 \* (t \* t \* rh)) -

(0.000814971 \* (t \* rh \* rh)) +

(0.0000102102 \* (t \* t \* rh \* rh)) -

(0.000038646 \* (t \* t \* t)) +

(0.0000291583 \* (rh \* rh \* rh)) +

(0.00000142721 \* (t \* t \* t \* rh)) +

(0.000000197483 \* (t \* rh \* rh \* rh)) -

(0.0000000218429 \* (t \* t \* t \* rh \* rh)) +

(0.000000000843296 \* (t \* t \* rh \* rh \* rh)) -

(0.0000000000481975 \* (t \* t \* t \* rh \* rh \* rh)));

display();

}

}

public void display() {

System.out.println("Heat index is " + heatIndex);

}

}

**e) StatisticsDisplay.java**

package weatherobservable;

import java.util.Observable;

import java.util.Observer;

public class StatisticsDisplay implements Observer, DisplayElement

{

private float maxTemp = 0.0f;

private float minTemp = 200;

private float tempSum= 0.0f;

private int numReadings;

public StatisticsDisplay(Observable observable)

{

observable.addObserver(this);

}

public void update(Observable observable, Object arg)

{

if (observable instanceof WeatherData)

{

WeatherData weatherData = (WeatherData)observable;

float temp = weatherData.getTemperature();

tempSum += temp;

numReadings++;

if (temp > maxTemp)

{

maxTemp = temp;

}

if (temp < minTemp)

{

minTemp = temp;

}

display();

}

}

public void display()

{

System.out.println("Avg/Max/Min temperature = " + (tempSum / numReadings)

+ "/" + maxTemp + "/" + minTemp);

}

}

**f) WeatherData.java**

package weatherobservable;

import java.util.Observable;

import java.util.Observer;

public class WeatherData extends Observable

{

private float temperature;

private float humidity;

private float pressure;

public WeatherData() { }

public void measurementsChanged()

{

setChanged();

notifyObservers();

}

public void setMeasurements(float temperature, float humidity, float pressure)

{

this.temperature = temperature;

this.humidity = humidity;

this.pressure = pressure;

measurementsChanged();

}

public float getTemperature() {

return temperature;

}

public float getHumidity() {

return humidity;

}

public float getPressure() {

return pressure;

}

}

**Output:**

Heat index is2567.7097

Forecast :More of the same

Avg/MAx/Min temperature = 80.0/80.0/80.0

Current conditions: 80.0F Degrees and 65.0%humidity

Heat index is3114.5156

Forecast :More of the same

Avg/MAx/Min temperature = 81.0/82.0/80.0

Current conditions: 82.0F Degrees and 70.0%humidity

Heat index is4612.1143

Forecast :More of the same

Avg/MAx/Min temperature = 80.0/82.0/78.0

Current conditions: 78.0F Degrees and 90.0%humidity

Assignment no : 2

Name : Sonukumar C. Thakur

Roll no : 7649 Class : S.Y.MSc(C.S)

Batch: Date:

Pracrical No:

**Q)** Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.

**Solution :**

Create **test.txt** file which will contain String for coverting outside the package.

**InputTest.java** main file outside the package.

import DecoratorPackage.\*;

import java.io.\*;

public class InputTest {

public static void main(String[] args) throws IOException {

int c;

try {

InputStream in = new LowerCaseInputStream(new BufferedInputStream(new FileInputStream("test.txt")));

while((c = in.read()) >= 0) {

System.out.print((char)c);

}

in.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

Create Package **DecoratorPackage** and inside that write below file :

**a) LowerCaseInputStream.java**

package DecoratorPackage;

import java.io.\*;

public class LowerCaseInputStream extends FilterInputStream {

public LowerCaseInputStream(InputStream in) {

super(in);

}

public int read() throws IOException {

int c = super.read();

return (c == -1 ? c : Character.toLowerCase((char)c));

}

public int read(byte[] b, int offset, int len) throws IOException {

int result = super.read(b, offset, len);

for (int i = offset; i < offset+result; i++) {

b[i] = (byte)Character.toLowerCase((char)b[i]);

}

return result;

}

}

**Output :**

this is a file. // “THIS IS A FILE” Converted to Lower case.

Assignment no : 3

Name : Sonukumar C. Thakur

Roll no : 7649 Class : S.Y.MSc(C.S)

Batch: Date:

Pracrical No:

Q) Write a Java Program to implement Factory method for Pizza Store with createPizza(), orederPizza(), prepare(), Bake(), cut(), box(). Use this to create variety of pizza’s like NyStyleCheesePizza, ChicagoStyleCheesePizza etc.

**Solution :**

**PizzaTestDrive.java** main file outside the package.

import FactoryPackage.\*;

public class PizzaTestDrive {

public static void main(String[] args) {

PizzaStore nyStore = new NYPizzaStore();

PizzaStore chicagoStore = new ChicagoPizzaStore();

Pizza pizza = nyStore.orderPizza("cheese");

System.out.println("Ethan ordered a " + pizza.getName() + "\n");

pizza = chicagoStore.orderPizza("cheese");

System.out.println("Joel ordered a " + pizza.getName() + "\n");

}

}

Create Package **FactoryPackage** and inside that write below files :

**a) Pizza.java**

package FactoryPackage;

import java.util.ArrayList;

public abstract class Pizza {

String name;

String dough;

String sauce;

ArrayList toppings = new ArrayList();

void prepare() {

System.out.println("Preparing " + name);

System.out.println("Tossing dough...");

System.out.println("Adding sauce...");

System.out.println("Adding toppings: ");

for (int i = 0; i < toppings.size(); i++) {

System.out.println(" " + toppings.get(i));

}

}

void bake() {

System.out.println("Bake for 25 minutes at 350");

}

void cut() {

System.out.println("Cutting the pizza into diagonal slices");

}

void box() {

System.out.println("Place pizza in official PizzaStore box");

}

public String getName() {

return name;

}

public String toString() {

StringBuffer display = new StringBuffer();

display.append("---- " + name + " ----\n");

display.append(dough + "\n");

display.append(sauce + "\n");

for (int i = 0; i < toppings.size(); i++) {

display.append((String )toppings.get(i) + "\n");

}

return display.toString();

}

}

**b) PizzaStore.java**

package FactoryPackage;

public abstract class PizzaStore {

abstract Pizza createPizza(String item);

public Pizza orderPizza(String type) {

Pizza pizza = createPizza(type);

System.out.println("--- Making a " + pizza.getName() + " ---");

pizza.prepare();

pizza.bake();

pizza.cut();

pizza.box();

return pizza;

}

}

**c) ChicagoPizzaStore.java**

package FactoryPackage;

public class ChicagoPizzaStore extends PizzaStore {

Pizza createPizza(String item) {

if (item.equals("cheese")) {

return new ChicagoStyleCheesePizza();

} else if (item.equals("veggie")) {

return new ChicagoStyleVeggiePizza();

} else if (item.equals("clam")) {

return new ChicagoStyleClamPizza();

} else if (item.equals("pepperoni")) {

return new ChicagoStylePepperoniPizza();

} else return null;

}

}

**d) ChicagoStyleCheesePizza.java**

package FactoryPackage;

public class ChicagoStyleCheesePizza extends Pizza {

public ChicagoStyleCheesePizza() {

name = "Chicago Style Deep Dish Cheese Pizza";

dough = "Extra Thick Crust Dough";

sauce = "Plum Tomato Sauce";

toppings.add("Shredded Mozzarella Cheese");

}

void cut() {

System.out.println("Cutting the pizza into square slices");

}

}

e) NYPizzaStore.java

package FactoryPackage;

public class NYPizzaStore extends PizzaStore {

Pizza createPizza(String item) {

if (item.equals("cheese")) {

return new NYStyleCheesePizza();

} else if (item.equals("veggie")) {

return new NYStyleVeggiePizza();

} else if (item.equals("clam")) {

return new NYStyleClamPizza();

} else if (item.equals("pepperoni")) {

return new NYStylePepperoniPizza();

} else return null;

}

}

f) NYStyleCheesePizza

package FactoryPackage;

public class NYStyleCheesePizza extends Pizza {

public NYStyleCheesePizza() {

name = "NY Style Sauce and Cheese Pizza";

dough = "Thin Crust Dough";

sauce = "Marinara Sauce";

toppings.add("Grated Reggiano Cheese");

}

}

**Output :**

----Making aNY Style Sauce and Cheese Pizza----

Prearing NY Style Sauce and Cheese Pizza

Tossing dough....

Adding Sauce....

Adding Toppings ;

Grated Reggiano Cheese

Bake for 25 minutes at 350 degrees

Cutting the pizza into Diagonal slices

Place pizza into official Pizzastore Box

Ethan ordered aNY Style Sauce and Cheese Pizza

----Making aChicago Style Deep Dish Cheese Pizza----

Prearing Chicago Style Deep Dish Cheese Pizza

Tossing dough....

Adding Sauce....

Adding Toppings ;

Shredded Mozzarela Cheese

Bake for 25 minutes at 350 degrees

Cutting the pizza into square slices

Place pizza into official Pizzastore Box

Joel ordered aChicago Style Deep Dish Cheese Pizza

Assignment no : 4

Name : Sonukumar C. Thakur

Roll no : 7649 Class : S.Y.MSc(C.S)

Batch: Date:

Pracrical No:

Q) Write a Java Program to implement Singleton pattern for multithreading.

**Solution:**

**SingletonTestDrive.java** main file outside the package.

import SingletonPackage.\*;

public class SingletonTestDrive {

public static void main(String[] args) {

Singleton foo = CoolerSingleton.getInstance();

Singleton bar = HotterSingleton.getInstance();

System.out.println(foo);

System.out.println(bar);

}

}

Create Package **SingletonPackage** and inside that write below files :

a) CoolerSingleton.java

package SingletonPackage;

public class CoolerSingleton extends Singleton {

// useful instance variables here

protected static Singleton uniqueInstance;

private CoolerSingleton() {

super();

}

// useful methods here

}

b) HotterSingleton.java

package SingletonPackage;

public class HotterSingleton extends Singleton {

// useful instance variables here

private HotterSingleton() {

super();

}

// useful methods here

}

c) Singleton.java

package SingletonPackage;

public class Singleton {

protected static Singleton uniqueInstance;

// other useful instance variables here

protected Singleton() {}

public static synchronized Singleton getInstance() {

if (uniqueInstance == null) {

uniqueInstance = new Singleton();

}

return uniqueInstance;

}

// other useful methods here

}

**Output** :

SingletonPackage.Singleton@2a139a55

SingletonPackage.Singleton@2a139a55

Assignment no : 5

Name : Sonukumar C. Thakur

Roll no : 7649 Class : S.Y.MSc(C.S)

Batch: Date:

Pracrical No:

**Q)** Write a Java Program to implement command pattern to test Remote Control.

**Solution:**

**RemoteLoader.java** main file outside the package.

import undo.\*;

public class RemoteLoader

{

public static void main(String[] args)

{

RemoteControl remoteControl = new RemoteControl();

CeilingFan ceilingFan = new CeilingFan("Living Room");

CeilingFanMediumCommand ceilingFanMedium = new CeilingFanMediumCommand(ceilingFan);

CeilingFanHighCommand ceilingFanHigh = new CeilingFanHighCommand(ceilingFan);

CeilingFanOffCommand ceilingFanOff = new CeilingFanOffCommand(ceilingFan);

remoteControl.setCommand(0, ceilingFanMedium, ceilingFanOff);

remoteControl.setCommand(1, ceilingFanHigh, ceilingFanOff);

remoteControl.onButtonWasPushed(0);

remoteControl.offButtonWasPushed(0);

System.out.println(remoteControl);

//remoteControl.undoButtonWasPushed();

remoteControl.onButtonWasPushed(1);

System.out.println(remoteControl);

//remoteControl.undoButtonWasPushed();

}

}

Create Package **undo** and inside that write below files :

**a) CeilingFan.java**

package undo;

public class CeilingFan

{

public static final int HIGH = 3;

public static final int MEDIUM = 2;

public static final int LOW = 1;

public static final int OFF = 0;

String location;

int speed;

public CeilingFan(String location)

{

this.location = location;

speed = OFF;

}

public void high()

{

speed = HIGH;

System.out.println(location + " ceiling fan is on high");

}

public void medium()

{

speed = MEDIUM;

System.out.println(location + " ceiling fan is on medium");

}

public void low()

{

speed = LOW;

System.out.println(location + " ceiling fan is on low");

}

public void off()

{

speed = OFF;

System.out.println(location + " ceiling fan is off");

}

public int getSpeed()

{

return speed;

}

}

**b) CeilingFanHighCommand.java**

package undo;

public class CeilingFanHighCommand implements Command

{

CeilingFan ceilingFan;

int prevSpeed;

public CeilingFanHighCommand(CeilingFan ceilingFan)

{

this.ceilingFan = ceilingFan;

}

public void execute()

{

prevSpeed = ceilingFan.getSpeed();

ceilingFan.high();

}

public void undo()

{

if (prevSpeed == CeilingFan.HIGH)

{

ceilingFan.high();

}

else if (prevSpeed == CeilingFan.MEDIUM)

{

ceilingFan.medium();

}

else if (prevSpeed == CeilingFan.LOW)

{

ceilingFan.low();

}

else if (prevSpeed == CeilingFan.OFF)

{

ceilingFan.off();

}

}

}

**c) CeilingFanLowCommand.java**

package undo;

public class CeilingFanLowCommand implements Command

{

CeilingFan ceilingFan;

int prevSpeed;

public CeilingFanLowCommand(CeilingFan ceilingFan)

{

this.ceilingFan = ceilingFan;

}

public void execute()

{

prevSpeed = ceilingFan.getSpeed();

ceilingFan.low();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

} else if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

} else if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

} else if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

**d) CeilingFanMediumCommand.java**

package undo;

public class CeilingFanMediumCommand implements Command

{

CeilingFan ceilingFan;

int prevSpeed;

public CeilingFanMediumCommand(CeilingFan ceilingFan)

{

this.ceilingFan = ceilingFan;

}

public void execute()

{

prevSpeed = ceilingFan.getSpeed();

ceilingFan.medium();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

} else if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

} else if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

} else if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

**e) CeilingFanOffCommand.java**

package undo;

public class CeilingFanOffCommand implements Command

{

CeilingFan ceilingFan;

int prevSpeed;

public CeilingFanOffCommand(CeilingFan ceilingFan)

{

this.ceilingFan = ceilingFan;

}

public void execute()

{

prevSpeed = ceilingFan.getSpeed();

ceilingFan.off();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

} else if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

} else if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

} else if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

**f) Command.java**

package undo;

public interface Command

{

public void execute();

public void undo();

}

**e) RemoteControl.java**

package undo;

import java.util.\*;

public class RemoteControl

{

Command[] onCommands;

Command[] offCommands;

public RemoteControl()

{

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;

}\*/

}

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

onCommands[slot] = onCommand;

offCommands[slot] = offCommand;

}

public void onButtonWasPushed(int slot) {

onCommands[slot].execute();

}

public void offButtonWasPushed(int slot) {

offCommands[slot].execute();

}

public String toString() {

StringBuffer stringBuff = 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");

}

return stringBuff.toString();

}

}

Assignment no : 6

Name : Sonukumar C. Thakur

Roll no : 7649 Class : S.Y.MSc(C.S)

Batch: Date:

Pracrical No:

**Q)** Write a Java Program to implement undo command to test CeilingFan.

**Solution:**

**RemoteLoader.java** main file outside the package.

import undo.\*;

public class RemoteLoader

{

public static void main(String[] args)

{

RemoteControl remoteControl = new RemoteControl();

CeilingFan ceilingFan = new CeilingFan("Living Room");

CeilingFanMediumCommand ceilingFanMedium = new CeilingFanMediumCommand(ceilingFan);

CeilingFanHighCommand ceilingFanHigh = new CeilingFanHighCommand(ceilingFan);

CeilingFanOffCommand ceilingFanOff = new CeilingFanOffCommand(ceilingFan);

remoteControl.setCommand(0, ceilingFanMedium, ceilingFanOff);

remoteControl.setCommand(1, ceilingFanHigh, ceilingFanOff);

remoteControl.onButtonWasPushed(0);

remoteControl.offButtonWasPushed(0);

System.out.println(remoteControl);

//remoteControl.undoButtonWasPushed();

remoteControl.onButtonWasPushed(1);

System.out.println(remoteControl);

//remoteControl.undoButtonWasPushed();

}

}

Create Package **undo** and inside that write below files :

**a) CeilingFan.java**

package undo;

public class CeilingFan

{

public static final int HIGH = 3;

public static final int MEDIUM = 2;

public static final int LOW = 1;

public static final int OFF = 0;

String location;

int speed;

public CeilingFan(String location)

{

this.location = location;

speed = OFF;

}

public void high()

{

speed = HIGH;

System.out.println(location + " ceiling fan is on high");

}

public void medium()

{

speed = MEDIUM;

System.out.println(location + " ceiling fan is on medium");

}

public void low()

{

speed = LOW;

System.out.println(location + " ceiling fan is on low");

}

public void off()

{

speed = OFF;

System.out.println(location + " ceiling fan is off");

}

public int getSpeed()

{

return speed;

}

}

**b) CeilingFanHighCommand.java**

package undo;

public class CeilingFanHighCommand implements Command

{

CeilingFan ceilingFan;

int prevSpeed;

public CeilingFanHighCommand(CeilingFan ceilingFan)

{

this.ceilingFan = ceilingFan;

}

public void execute()

{

prevSpeed = ceilingFan.getSpeed();

ceilingFan.high();

}

public void undo()

{

if (prevSpeed == CeilingFan.HIGH)

{

ceilingFan.high();

}

else if (prevSpeed == CeilingFan.MEDIUM)

{

ceilingFan.medium();

}

else if (prevSpeed == CeilingFan.LOW)

{

ceilingFan.low();

}

else if (prevSpeed == CeilingFan.OFF)

{

ceilingFan.off();

}

}

}

**c) CeilingFanLowCommand.java**

package undo;

public class CeilingFanLowCommand implements Command

{

CeilingFan ceilingFan;

int prevSpeed;

public CeilingFanLowCommand(CeilingFan ceilingFan)

{

this.ceilingFan = ceilingFan;

}

public void execute()

{

prevSpeed = ceilingFan.getSpeed();

ceilingFan.low();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

} else if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

} else if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

} else if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

**d) CeilingFanMediumCommand.java**

package undo;

public class CeilingFanMediumCommand implements Command

{

CeilingFan ceilingFan;

int prevSpeed;

public CeilingFanMediumCommand(CeilingFan ceilingFan)

{

this.ceilingFan = ceilingFan;

}

public void execute()

{

prevSpeed = ceilingFan.getSpeed();

ceilingFan.medium();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

} else if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

} else if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

} else if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

**e) CeilingFanOffCommand.java**

package undo;

public class CeilingFanOffCommand implements Command

{

CeilingFan ceilingFan;

int prevSpeed;

public CeilingFanOffCommand(CeilingFan ceilingFan)

{

this.ceilingFan = ceilingFan;

}

public void execute()

{

prevSpeed = ceilingFan.getSpeed();

ceilingFan.off();

}

public void undo() {

if (prevSpeed == CeilingFan.HIGH) {

ceilingFan.high();

} else if (prevSpeed == CeilingFan.MEDIUM) {

ceilingFan.medium();

} else if (prevSpeed == CeilingFan.LOW) {

ceilingFan.low();

} else if (prevSpeed == CeilingFan.OFF) {

ceilingFan.off();

}

}

}

**f) Command.java**

package undo;

public interface Command

{

public void execute();

public void undo();

}

**e) RemoteControl.java**

package undo;

import java.util.\*;

public class RemoteControl

{

Command[] onCommands;

Command[] offCommands;

public RemoteControl()

{

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;

}\*/

}

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

onCommands[slot] = onCommand;

offCommands[slot] = offCommand;

}

public void onButtonWasPushed(int slot) {

onCommands[slot].execute();

}

public void offButtonWasPushed(int slot) {

offCommands[slot].execute();

}

public String toString() {

StringBuffer stringBuff = 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");

}

return stringBuff.toString();

}

}

**f) RemoteControlWithUndo.java**

package undo;

import java.util.\*;

public class RemoteControlWithUndo {

Command[] onCommands;

Command[] offCommands;

Command undoCommand;

public RemoteControlWithUndo() {

onCommands = new Command[7];

offCommands = new Command[7];

/\*Command noCommand = new Command();

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() {

StringBuffer stringBuff = 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");

}

stringBuff.append("[undo] " + undoCommand.getClass().getName() + "\n");

return stringBuff.toString();

}

}

Assignment no : 7

Name : Sonukumar C. Thakur

Roll no : 7649 Class : S.Y.MSc(C.S)

Batch: Date:

Pracrical No:

**Q)** Write a Java Program to implement Iterator Pattern for Designing Menu like Breakfast, Lunch or Dinner Menu.

**Solution :**

**MenuTestDrive.java** main file outside the package.

import dinermerger.\*;

import java.util.\*;

public class MenuTestDrive

{

public static void main(String args[])

{

PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();

DinerMenu dinerMenu = new DinerMenu();

Waitress waitress = new Waitress(pancakeHouseMenu, dinerMenu);

waitress.printMenu();

}

public static void printMenu()

{

PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();

DinerMenu dinerMenu = new DinerMenu();

ArrayList breakfastItems = pancakeHouseMenu.getMenuItems();

for (int i = 0; i < breakfastItems.size(); i++) {

MenuItem menuItem = (MenuItem)breakfastItems.get(i);

System.out.print(menuItem.getName());

System.out.println("\t\t" + menuItem.getPrice());

System.out.println("\t" + menuItem.getDescription());

}

MenuItem[] lunchItems = dinerMenu.getMenuItems();

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

MenuItem menuItem = lunchItems[i];

System.out.print(menuItem.getName());

System.out.println("\t\t" + menuItem.getPrice());

System.out.println("\t" + menuItem.getDescription());

}

}

}

Create Package **dinermerger** and inside that write below files :

**a) AlternatingDinerMenuIterator.java**

package dinermerger;

import java.util.Calendar;

public class AlternatingDinerMenuIterator implements Iterator {

MenuItem[] list;

int position;

public AlternatingDinerMenuIterator(MenuItem[] list) {

this.list = list;

Calendar rightNow = Calendar.getInstance();

position = rightNow.DAY\_OF\_WEEK % 2;

}

public Object next() {

MenuItem menuItem = list[position];

position = position + 2;

return menuItem;

}

public boolean hasNext() {

if (position >= list.length || list[position] == null) {

return false;

} else {

return true;

}

}

public String toString() {

return "Alternating Diner Menu Iterator";

}

}

**b) ArrayIterator.java**

package dinermerger;

public class ArrayIterator implements Iterator {

MenuItem[] items;

int position = 0;

public ArrayIterator(MenuItem[] items) {

this.items = items;

}

public Object next() {

MenuItem menuItem = items[position];

position = position + 1;

return menuItem;

}

public boolean hasNext() {

if (position >= items.length || items[position] == null) {

return false;

} else {

return true;

}

}

}

**c) ArrayListIterator.java**

package dinermerger;

import java.util.ArrayList;

public class ArrayListIterator implements Iterator {

ArrayList items;

int position = 0;

public ArrayListIterator(ArrayList items) {

this.items = items;

}

public Object next() {

Object object = items.get(position);

position = position + 1;

return object;

}

public boolean hasNext() {

if (position >= items.size()) {

return false;

} else {

return true;

}

}

}

**d) DinerMenu.java**

package dinermerger;

public class DinerMenu implements Menu {

static final int MAX\_ITEMS = 6;

int numberOfItems = 0;

MenuItem[] menuItems;

public DinerMenu() {

menuItems = new MenuItem[MAX\_ITEMS];

addItem("Vegetarian BLT",

"(Fakin') Bacon with lettuce & tomato on whole wheat", true, 2.99);

addItem("BLT",

"Bacon with lettuce & tomato on whole wheat", false, 2.99);

addItem("Soup of the day",

"Soup of the day, with a side of potato salad", false, 3.29);

addItem("Hotdog",

"A hot dog, with saurkraut, relish, onions, topped with cheese",

false, 3.05);

addItem("Steamed Veggies and Brown Rice",

"Steamed vegetables over brown rice", true, 3.99);

addItem("Pasta",

"Spaghetti with Marinara Sauce, and a slice of sourdough bread",

true, 3.89);

}

public void addItem(String name, String description,

boolean vegetarian, double price)

{

MenuItem menuItem = new MenuItem(name, description, vegetarian, price);

if (numberOfItems >= MAX\_ITEMS) {

System.err.println("Sorry, menu is full! Can't add item to menu");

} else {

menuItems[numberOfItems] = menuItem;

numberOfItems = numberOfItems + 1;

}

}

public MenuItem[] getMenuItems() {

return menuItems;

}

public Iterator createIterator() {

return new DinerMenuIterator(menuItems);

}

// other menu methods here

}

**e) DinerMenuIterator.java**

package dinermerger;

public class DinerMenuIterator implements Iterator {

MenuItem[] items;

int position = 0;

public DinerMenuIterator(MenuItem[] items) {

this.items = items;

}

public Object next() {

MenuItem menuItem = items[position];

position = position + 1;

return menuItem;

}

public boolean hasNext() {

if (position >= items.length || items[position] == null) {

return false;

} else {

return true;

}

}

}

**f) Iterator.java**

package dinermerger;

public interface Iterator {

boolean hasNext();

Object next();

}

**g) Menu.java**

package dinermerger;

public interface Menu {

public Iterator createIterator();

}

**h) MenuItem.java**

package dinermerger;

public class MenuItem {

String name;

String description;

boolean vegetarian;

double price;

public MenuItem(String name,

String description,

boolean vegetarian,

double price)

{

this.name = name;

this.description = description;

this.vegetarian = vegetarian;

this.price = price;

}

public String getName() {

return name;

}

public String getDescription() {

return description;

}

public double getPrice() {

return price;

}

public boolean isVegetarian() {

return vegetarian;

}

public String toString() {

return (name + ", $" + price + "\n " + description);

}

}

**i) PancakeHouseMenu.java**

package dinermerger;

import java.util.ArrayList;

public class PancakeHouseMenu implements Menu {

ArrayList menuItems;

public PancakeHouseMenu() {

menuItems = new ArrayList();

addItem("K&B's Pancake Breakfast",

"Pancakes with scrambled eggs, and toast",

true,

2.99);

addItem("Regular Pancake Breakfast",

"Pancakes with fried eggs, sausage",

false,

2.99);

addItem("Blueberry Pancakes",

"Pancakes made with fresh blueberries",

true,

3.49);

addItem("Waffles",

"Waffles, with your choice of blueberries or strawberries",

true,

3.59);

}

public void addItem(String name, String description,

boolean vegetarian, double price)

{

MenuItem menuItem = new MenuItem(name, description, vegetarian, price);

menuItems.add(menuItem);

}

public ArrayList getMenuItems() {

return menuItems;

}

public Iterator createIterator() {

return new PancakeHouseMenuIterator(menuItems);

}

public String toString() {

return "Objectville Pancake House Menu";

}

// other menu methods here

}

**j) PancakeHouseMenuIterator.java**

package dinermerger;

import java.util.ArrayList;

public class PancakeHouseMenuIterator implements Iterator {

ArrayList items;

int position = 0;

public PancakeHouseMenuIterator(ArrayList items) {

this.items = items;

}

public Object next() {

Object object = items.get(position);

position = position + 1;

return object;

}

public boolean hasNext() {

if (position >= items.size()) {

return false;

} else {

return true;

}

}

}

**k) Waitress.java**

package dinermerger;

public class Waitress {

PancakeHouseMenu pancakeHouseMenu;

DinerMenu dinerMenu;

public Waitress(PancakeHouseMenu pancakeHouseMenu, DinerMenu dinerMenu) {

this.pancakeHouseMenu = pancakeHouseMenu;

this.dinerMenu = dinerMenu;

}

public void printMenu() {

Iterator pancakeIterator = pancakeHouseMenu.createIterator();

Iterator dinerIterator = dinerMenu.createIterator();

System.out.println("MENU\n----\nBREAKFAST");

printMenu(pancakeIterator);

System.out.println("\nLUNCH");

printMenu(dinerIterator);

}

private void printMenu(Iterator iterator) {

while (iterator.hasNext()) {

MenuItem menuItem = (MenuItem)iterator.next();

System.out.print(menuItem.getName() + ", ");

System.out.print(menuItem.getPrice() + " -- ");

System.out.println(menuItem.getDescription());

}

}

public void printVegetarianMenu() {

printVegetarianMenu(pancakeHouseMenu.createIterator());

printVegetarianMenu(dinerMenu.createIterator());

}

public boolean isItemVegetarian(String name) {

Iterator breakfastIterator = pancakeHouseMenu.createIterator();

if (isVegetarian(name, breakfastIterator)) {

return true;

}

Iterator dinnerIterator = dinerMenu.createIterator();

if (isVegetarian(name, dinnerIterator)) {

return true;

}

return false;

}

private void printVegetarianMenu(Iterator iterator) {

while (iterator.hasNext()) {

MenuItem menuItem = (MenuItem)iterator.next();

if (menuItem.isVegetarian()) {

System.out.print(menuItem.getName());

System.out.println("\t\t" + menuItem.getPrice());

System.out.println("\t" + menuItem.getDescription());

}

}

}

private boolean isVegetarian(String name, Iterator iterator) {

while (iterator.hasNext()) {

MenuItem menuItem = (MenuItem)iterator.next();

if (menuItem.getName().equals(name)) {

if (menuItem.isVegetarian()) {

return true;

}

}

}

return false;

}

}

**Output:**

MENU

----­­­---------------------------------------------------------------------------

BREAKFAST

K&B's Pancake Breakfast, 2.99 -- Pancakes with scrambled eggs, and toast

Regular Pancake Breakfast, 2.99 -- Pancakes with fried eggs, sausage

Blueberry Pancakes, 3.49 -- Pancakes made with fresh blueberries

Waffles, 3.59 -- Waffles, with your choice of blueberries or strawberries

LUNCH

Vegetarian BLT, 2.99 -- (Fakin') Bacon with lettuce & tomato on whole wheat

BLT, 2.99 -- Bacon with lettuce & tomato on whole wheat

Soup of the day, 3.29 -- Soup of the day, with a side of potato salad

Hotdog, 3.05 -- A hot dog, with saurkraut, relish, onions, topped with cheese

Steamed Veggies and Brown Rice, 3.99 -- Steamed vegetables over brown rice

Pasta, 3.89 -- Spaghetti with Marinara Sauce, and a slice of sourdough bread