Software Architecture & Design Pattern List of Assignments

Q1) 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()

DisplayElement.java

**package** program\_1;

**public** **interface** DisplayElement {

**public** **void** display();

}

CurrentConditionsDisplay.java

package program\_1;

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");

}

}

ForecastDisplay.java

package program\_1;

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");

}

}

}

HeatIndexDisplay.java

package program\_1;

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);

}

}

StatisticsDisplay.java

package program\_1;

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);

}

}

WeatherData.java

package program\_1;

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;

}

}

WeatherStation.java

**package** program\_1;

**public** **class** WeatherStation {

**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);

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

weatherData.setMeasurements(82, 70, 29.2f);

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

}

}

WeatherStationHeatIndex.java

**package** program\_1;

**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, 29.2f);

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

}

}

Output :-

Heat index is 82.95535

Forecast: Improving weather on the way!

Avg/Max/Min temperature = 80.0/80.0/80.0

Current conditions: 80.0F degrees and 65.0% humidity

Heat index is 86.90124

Forecast: Watch out for cooler, rainy weather

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

Current conditions: 82.0F degrees and 70.0% humidity

Heat index is 83.64967

Forecast: More of the same

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

Current conditions: 78.0F degrees and 90.0% humidity

Q2) Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.

InputTest.java

**package** program\_2;

**import** java.io.\*;

**public** **class** InputTest {

**public** **static** **void** main(String[] args) {

String str1="Great Power";

StringBuffer newStr=**new** StringBuffer(str1);

**for**(**int** i = 0; i < str1.length(); i++) {

//Checks for lower case character

**if**(Character.*isLowerCase*(str1.charAt(i))) {

//Convert it into upper case using toUpperCase() function

newStr.setCharAt(i, Character.*toUpperCase*(str1.charAt(i)));

}

//Checks for upper case character

**else** **if**(Character.*isUpperCase*(str1.charAt(i))) {

//Convert it into upper case using toLowerCase() function

newStr.setCharAt(i, Character.*toLowerCase*(str1.charAt(i)));

}

}

System.***out***.println("String after case conversion : " + newStr);

}

}

Output :-String after case conversion : gREAT pOWER

Q.3: 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.

Pizza.java

**package** program\_3;

**import** java.util.ArrayList;

**public** **abstract** **class** Pizza {

String name;

String dough;

String sauce;

ArrayList toppings = **new** ArrayList();

**public** String getName() {

**return** name;

}

**public** **void** prepare() {

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

}

**public** **void** bake() {

System.***out***.println("Baking " + name);

}

**public** **void** cut() {

System.***out***.println("Cutting " + name);

}

**public** **void** box() {

System.***out***.println("Boxing " + name);

}

**public** String toString() {

// code to display pizza name and ingredients

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();

}

}

ChicagoStyleCheesePizza.java

**package** program\_3;

**public** **class** ChicagoStyleCheesePizza **extends** Pizza {

**public** **void** ChicagoStyleCheesePizza() {

name = "Clam Pizza";

dough = "Thin crust";

sauce = "White garlic sauce";

toppings.add("Clams");

toppings.add("Grated parmesan cheese");

}

}

NyStyleCheesePizza.java

**package** program\_3;

**public** **class** NyStyleCheesePizza **extends** Pizza {

**public** **void** NyStyleCheesePizza() {

name = "Cheese Pizza";

dough = "Regular Crust";

sauce = "Marinara Pizza Sauce";

toppings.add("Fresh Mozzarella");

toppings.add("Parmesan");

}

}

PepperoniPizza.java

**package** program\_3;

**public** **class** PepperoniPizza **extends** Pizza {

**public** **void** PepperoniPizza() {

name = "Pepperoni Pizza";

dough = "Crust";

sauce = "Marinara sauce";

toppings.add("Sliced Pepperoni");

toppings.add("Sliced Onion");

toppings.add("Grated parmesan cheese");

}

}

VeggiePizza.java

**package** program\_3;

**public** **class** VeggiePizza **extends** Pizza {

**public** VeggiePizza() {

name = "Veggie Pizza";

dough = "Crust";

sauce = "Marinara sauce";

toppings.add("Shredded mozzarella");

toppings.add("Grated parmesan");

toppings.add("Diced onion");

toppings.add("Sliced mushrooms");

toppings.add("Sliced red pepper");

toppings.add("Sliced black olives");

}

}

PizzaStore.java

**package** program\_3;

**public** **class** PizzaStore {

SimplePizzaFactory factory;

**public** PizzaStore(SimplePizzaFactory factory) {

**this**.factory = factory;

}

**public** Pizza orderPizza(String type) {

Pizza pizza;

pizza = factory.createPizza(type);

pizza.prepare();

pizza.bake();

pizza.cut();

pizza.box();

**return** pizza;

}

}

SimplePizzaFactory.java

**package** program\_3;

**public** **class** SimplePizzaFactory {

**public** Pizza createPizza(String type) {

Pizza pizza = **null**;

**if** (type.equals("cheese")) {

pizza = **new** ChicagoStyleCheesePizza();

} **else** **if** (type.equals("pepperoni")) {

pizza = **new** PepperoniPizza();

} **else** **if** (type.equals("clam")) {

pizza = **new** NyStyleCheesePizza();

} **else** **if** (type.equals("veggie")) {

pizza = **new** VeggiePizza();

}

**return** pizza;

}

}

PizzaTestDrive.java

**package** program\_3;

**public** **class** PizzaTestDrive {

**public** **static** **void** main(String[] args) {

SimplePizzaFactory factory = **new** SimplePizzaFactory();

PizzaStore store = **new** PizzaStore(factory);

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

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

pizza = store.orderPizza("veggie");

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

}

}

Q.4: Write a Java Program to implement Singleton pattern for multithreading.

Singleton.java

**package** program\_4;

**public** **class** Singleton {

**private** **static** Singleton *uniqueInstance*;

// other useful instance variables here

**private** Singleton() {}

**public** **static** **synchronized** Singleton getInstance() {

**if** (*uniqueInstance* == **null**) {

*uniqueInstance* = **new** Singleton();

}

**return** *uniqueInstance*;

}

// other useful methods here

}

Q.5: Write a Java Program to implement command pattern to test Remote Control.

Command.java

**package** program\_5;

**public** **interface** Command {

**public** **void** execute();

}

Light.java

**package** program\_5;

**public** **class** Light {

**public** **void** on()

{

System.***out***.println("Light is on");

}

**public** **void** off()

{

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

}

}

LightOffCommand.java

**package** program\_5;

**public** **class** LightOffCommand **implements** Command

{

Light light;

**public** LightOffCommand(Light light)

{

**this**.light = light;

}

**public** **void** execute()

{

light.off();

}

}

LightOnCommand.java

**package** program\_5;

**public** **class** LightOnCommand **implements** Command

{

Light light;

// The constructor is passed the light it

// is going to control.

**public** LightOnCommand(Light light)

{

**this**.light = light;

}

**public** **void** execute()

{

light.on();

}

}

SimpleRemoteControl.java

**package** program\_5;

**public** **class** SimpleRemoteControl {

Command slot; // only one button

**public** SimpleRemoteControl()

{

}

**public** **void** setCommand(Command command)

{

// set the command the remote will

// execute

slot = command;

}

**public** **void** buttonWasPressed()

{

slot.execute();

}

}

Stereo.java

**package** program\_5;

**public** **class** Stereo {

**public** **void** on()

{

System.***out***.println("Stereo is on");

}

**public** **void** off()

{

System.***out***.println("Stereo is off");

}

**public** **void** setCD()

{

System.***out***.println("Stereo is set " +

"for CD input");

}

**public** **void** setDVD()

{

System.***out***.println("Stereo is set"+

" for DVD input");

}

**public** **void** setRadio()

{

System.***out***.println("Stereo is set" +

" for Radio");

}

**public** **void** setVolume(**int** volume)

{

// code to set the volume

System.***out***.println("Stereo volume set"

+ " to " + volume);

}

}

StereoOffCommand.java

**package** program\_5;

**public** **class** StereoOffCommand **implements** Command

{

Stereo stereo;

**public** StereoOffCommand(Stereo stereo)

{

**this**.stereo = stereo;

}

**public** **void** execute()

{

stereo.off();

}

}

StereoOnWithCDCommand.java

**package** program\_5;

**public** **class** StereoOnWithCDCommand **implements** Command

{

Stereo stereo;

**public** StereoOnWithCDCommand(Stereo stereo)

{

**this**.stereo = stereo;

}

**public** **void** execute()

{

stereo.on();

stereo.setCD();

stereo.setVolume(11);

}

}

RemoteControlTest.java

**package** program\_5;

**public** **class** RemoteControlTest {

**public** **static** **void** main(String[] args)

{

SimpleRemoteControl remote =

**new** SimpleRemoteControl();

Light light = **new** Light();

Stereo stereo = **new** Stereo();

// we can change command dynamically

remote.setCommand(**new**

LightOnCommand(light));

remote.buttonWasPressed();

remote.setCommand(**new**

StereoOnWithCDCommand(stereo));

remote.buttonWasPressed();

remote.setCommand(**new**

StereoOffCommand(stereo));

remote.buttonWasPressed();

}

}

Output :-

Light is on

Stereo is on

Stereo is set for CD input

Stereo volume set to 11

Stereo is off

Q.6: Write a Java Program to implement undo command to test Ceiling fan.

Command.java

**package** program\_6;

**public** **interface** Command {

**public** **void** execute();

**public** **void** undo();

}

NoCommand.java

**package** program\_6;

**public** **class** NoCommand **implements** Command {

**public** **void** execute() { }

**public** **void** undo() { }

}

CeilingFan.java

**package** program\_6;

**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;

}

}

CeilingFanHighCommand.java

**package** program\_6;

**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();

}

}

}

CeilingFanLowCommand.java

**package** program\_6;

**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();

}

}

}

CeilingFanMediumCommand.java

**package** program\_6;

**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();

}

}

}

CeilingFanOffCommand.java

**package** program\_6;

**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();

}

}

}

RemoteControlWithUndo.java

**package** program\_6;

**import** java.util.\*;

//

// This is the invoker

//

**public** **class** RemoteControlWithUndo {

Command[] onCommands;

Command[] offCommands;

Command undoCommand;

**public** RemoteControlWithUndo() {

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() {

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();

}

}

RemoteLoader.java

**package** program\_6;

**public** **class** RemoteLoader {

**public** **static** **void** main(String[] args) {

RemoteControlWithUndo remoteControl = **new** RemoteControlWithUndo();

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();

}

}

Output :-

Living Room ceiling fan is on medium

Living Room ceiling fan is off

------ Remote Control -------

[slot 0] program\_6.CeilingFanMediumCommand program\_6.CeilingFanOffCommand

[slot 1] program\_6.CeilingFanHighCommand program\_6.CeilingFanOffCommand

[slot 2] program\_6.NoCommand program\_6.NoCommand

[slot 3] program\_6.NoCommand program\_6.NoCommand

[slot 4] program\_6.NoCommand program\_6.NoCommand

[slot 5] program\_6.NoCommand program\_6.NoCommand

[slot 6] program\_6.NoCommand program\_6.NoCommand

[undo] program\_6.CeilingFanOffCommand

Living Room ceiling fan is on medium

Living Room ceiling fan is on high

------ Remote Control -------

[slot 0] program\_6.CeilingFanMediumCommand program\_6.CeilingFanOffCommand

[slot 1] program\_6.CeilingFanHighCommand program\_6.CeilingFanOffCommand

[slot 2] program\_6.NoCommand program\_6.NoCommand

[slot 3] program\_6.NoCommand program\_6.NoCommand

[slot 4] program\_6.NoCommand program\_6.NoCommand

[slot 5] program\_6.NoCommand program\_6.NoCommand

[slot 6] program\_6.NoCommand program\_6.NoCommand

[undo] program\_6.CeilingFanHighCommand

Living Room ceiling fan is on medium

Q.7: Write a Java Program to implement Adapter pattern for Enumeration iterator.

EnumerationIterator.java

**package** program\_7;

**import** java.util.\*;

**public** **class** EnumerationIterator **implements** Iterator {

Enumeration enumeration;

**public** EnumerationIterator(Enumeration enumeration) {

**this**.enumeration = enumeration;

}

**public** **boolean** hasNext() {

**return** enumeration.hasMoreElements();

}

**public** Object next() {

**return** enumeration.nextElement();

}

**public** **void** remove() {

**throw** **new** UnsupportedOperationException();

}

}

EnumerationIteratorTestDriver.java

**package** program\_7;

**import** java.util.\*;

**public** **class** EnumerationIteratorTestDriver {

**public** **static** **void** main (String args[]) {

Vector<String> vector = **new** Vector<>();

vector.add("a");

vector.add("b");

vector.add("c");

Iterator iterator = **new** EnumerationIterator(vector.elements());

**while** (iterator.hasNext()) {

System.***out***.println(iterator.next());

}

}

}

Output :-

a

b

c

Q.8: Write a Java Program to implement Iterator Pattern for Designing Menu like Breakfast, Lunch or Dinner Menu.

Container.java

**package** program\_8;

**public** **interface** Container {

**public** Iterator getIterator();

}

Iterator.java

**package** program\_8;

**public** **interface** Iterator {

**public** **boolean** hasNext();

**public** Object next();

}

BreakfastRepository.java

**package** program\_8;

**public** **class** BreakfastRepository **implements** Container

{

**public** String bkmenu[] = {"pohe","shira","uttpa"};

**public** Iterator getIterator()

{

**return** **new** BreakfastIterator();

}

**private** **class** BreakfastIterator **implements** Iterator

{

**int** index;

**public** **boolean** hasNext()

{

**if**( index <bkmenu.length)

**return** **true**;

**else**

**return** **false**;

}

**public** Object next()

{

**if**(**this**.hasNext())

**return** bkmenu[index++];

**else**

**return** **null**;

}

}

}

DinnerRepository.java

**package** program\_8;

**public** **class** DinnerRepository **implements** Container

{

**public** String dmenu[] = {"dum aalo","kadhai paneer","matar pulao"};

**public** Iterator getIterator()

{

**return** **new** DinnerIterator();

}

**private** **class** DinnerIterator **implements** Iterator

{

**int** index;

**public** **boolean** hasNext()

{

**if**( index <dmenu.length)

**return** **true**;

**else**

**return** **false**;

}

**public** Object next()

{

**if**(**this**.hasNext())

**return** dmenu[index++];

**else**

**return** **null**;

}

}

}

LunchRepository.java

**package** program\_8;

**public** **class** LunchRepository **implements** Container

{

**public** String lmenu[] = {"curd rice","tawa pulao","punjabi thali"};

**public** Iterator getIterator()

{

**return** **new** LunchIterator();

}

**private** **class** LunchIterator **implements** Iterator

{

**int** index;

**public** **boolean** hasNext()

{

**if**( index <lmenu.length)

**return** **true**;

**else**

**return** **false**;

}

**public** Object next()

{

**if**(**this**.hasNext())

**return** lmenu[index++];

**else**

**return** **null**;

}

}

}

IteratorPatternDemo.java

**package** program\_8;

**public** **class** IteratorPatternDemo {

**public** **static** **void** main(String[] args)

{

BreakfastRepository n1=**new** BreakfastRepository();

System.***out***.println("breakfast menu !!!!");

**for**(Iterator i1=n1.getIterator(); i1.hasNext(); )

{

String s1=(String)i1.next();

System.***out***.println(s1);

}

LunchRepository l1=**new** LunchRepository();

System.***out***.println("Lunch menu !!!!");

**for**(Iterator i1=l1.getIterator(); i1.hasNext(); )

{

String s1=(String)i1.next();

System.***out***.println(s1);

}

DinnerRepository d1=**new** DinnerRepository();

System.***out***.println("dinner menu !!!!");

**for**(Iterator i1=d1.getIterator(); i1.hasNext(); )

{

String s1=(String)i1.next();

System.***out***.println(s1);

}

}

}

Output :-

breakfast menu !!!!

pohe

shira

uttpa

Lunch menu !!!!

curd rice

tawa pulao

punjabi thali

dinner menu !!!!

dum aalo

kadhai paneer

matar pulao

Q.9: Write a Java Program to implement State Pattern for Gumball Machine. Create instance variable that holds current state from there, we just need to handle all actions, behaviors and

state transition that can happen. For actions we need to implement methods to insert a quarter, remove a quarter, turning the crank and display gumball.

State.java

**package** program\_9;

**public** **interface** State {

**public** **void** insertQuarter();

**public** **void** ejectQuarter();

**public** **void** turnCrank();

**public** **void** dispense();

**public** **void** refill();

}

SoldState.java

**package** program\_9;

**public** **class** SoldState **implements** State {

GumballMachine gumballMachine;

**public** SoldState(GumballMachine gumballMachine) {

**this**.gumballMachine = gumballMachine;

}

**public** **void** insertQuarter() {

System.***out***.println("Please wait, we're already giving you a gumball");

}

**public** **void** ejectQuarter() {

System.***out***.println("Sorry, you already turned the crank");

}

**public** **void** turnCrank() {

System.***out***.println("Turning twice doesn't get you another gumball!");

}

**public** **void** dispense() {

gumballMachine.releaseBall();

**if** (gumballMachine.getCount() > 0) {

gumballMachine.setState(gumballMachine.getNoQuarterState());

} **else** {

System.***out***.println("Oops, out of gumballs!");

gumballMachine.setState(gumballMachine.getSoldOutState());

}

}

**public** **void** refill() { }

**public** String toString() {

**return** "dispensing a gumball";

}

}

SoldOutState.java

**package** program\_9;

**public** **class** SoldOutState **implements** State {

GumballMachine gumballMachine;

**public** SoldOutState(GumballMachine gumballMachine) {

**this**.gumballMachine = gumballMachine;

}

**public** **void** insertQuarter() {

System.***out***.println("You can't insert a quarter, the machine is sold out");

}

**public** **void** ejectQuarter() {

System.***out***.println("You can't eject, you haven't inserted a quarter yet");

}

**public** **void** turnCrank() {

System.***out***.println("You turned, but there are no gumballs");

}

**public** **void** dispense() {

System.***out***.println("No gumball dispensed");

}

**public** **void** refill() {

gumballMachine.setState(gumballMachine.getNoQuarterState());

}

**public** String toString() {

**return** "sold out";

}

}

NoQuarterState.java

**package** program\_9;

**public** **class** NoQuarterState **implements** State {

GumballMachine gumballMachine;

**public** NoQuarterState(GumballMachine gumballMachine) {

**this**.gumballMachine = gumballMachine;

}

**public** **void** insertQuarter() {

System.***out***.println("You inserted a quarter");

gumballMachine.setState(gumballMachine.getHasQuarterState());

}

**public** **void** ejectQuarter() {

System.***out***.println("You haven't inserted a quarter");

}

**public** **void** turnCrank() {

System.***out***.println("You turned, but there's no quarter");

}

**public** **void** dispense() {

System.***out***.println("You need to pay first");

}

**public** **void** refill() { }

**public** String toString() {

**return** "waiting for quarter";

}

}

HasQuarterState.java

**package** program\_9;

**public** **class** HasQuarterState **implements** State {

GumballMachine gumballMachine;

**public** HasQuarterState(GumballMachine gumballMachine) {

**this**.gumballMachine = gumballMachine;

}

**public** **void** insertQuarter() {

System.***out***.println("You can't insert another quarter");

}

**public** **void** ejectQuarter() {

System.***out***.println("Quarter returned");

gumballMachine.setState(gumballMachine.getNoQuarterState());

}

**public** **void** turnCrank() {

System.***out***.println("You turned...");

gumballMachine.setState(gumballMachine.getSoldState());

}

**public** **void** dispense() {

System.***out***.println("No gumball dispensed");

}

**public** **void** refill() { }

**public** String toString() {

**return** "waiting for turn of crank";

}

}

GumballMachine.java

**package** program\_9;

**public** **class** GumballMachine {

State soldOutState;

State noQuarterState;

State hasQuarterState;

State soldState;

State state;

**int** count = 0;

**public** GumballMachine(**int** numberGumballs) {

soldOutState = **new** SoldOutState(**this**);

noQuarterState = **new** NoQuarterState(**this**);

hasQuarterState = **new** HasQuarterState(**this**);

soldState = **new** SoldState(**this**);

**this**.count = numberGumballs;

**if** (numberGumballs > 0) {

state = noQuarterState;

} **else** {

state = soldOutState;

}

}

**public** **void** insertQuarter() {

state.insertQuarter();

}

**public** **void** ejectQuarter() {

state.ejectQuarter();

}

**public** **void** turnCrank() {

state.turnCrank();

state.dispense();

}

**void** releaseBall() {

System.***out***.println("A gumball comes rolling out the slot...");

**if** (count != 0) {

count = count - 1;

}

}

**int** getCount() {

**return** count;

}

**void** refill(**int** count) {

**this**.count += count;

System.***out***.println("The gumball machine was just refilled; it's new count is: " + **this**.count);

state.refill();

}

**void** setState(State state) {

**this**.state = state;

}

**public** State getState() {

**return** state;

}

**public** State getSoldOutState() {

**return** soldOutState;

}

**public** State getNoQuarterState() {

**return** noQuarterState;

}

**public** State getHasQuarterState() {

**return** hasQuarterState;

}

**public** State getSoldState() {

**return** soldState;

}

**public** String toString() {

StringBuffer result = **new** StringBuffer();

result.append("\nMighty Gumball, Inc.");

result.append("\nJava-enabled Standing Gumball Model #2004");

result.append("\nInventory: " + count + " gumball");

**if** (count != 1) {

result.append("s");

}

result.append("\n");

result.append("Machine is " + state + "\n");

**return** result.toString();

}

}

GumballMachineTestDrive.java

**package** program\_9;

**public** **class** GumballMachineTestDrive {

**public** **static** **void** main(String[] args) {

GumballMachine gumballMachine = **new** GumballMachine(2);

System.***out***.println(gumballMachine);

gumballMachine.insertQuarter();

gumballMachine.turnCrank();

System.***out***.println(gumballMachine);

gumballMachine.insertQuarter();

gumballMachine.turnCrank();

gumballMachine.insertQuarter();

gumballMachine.turnCrank();

gumballMachine.refill(5);

gumballMachine.insertQuarter();

gumballMachine.turnCrank();

System.***out***.println(gumballMachine);

}

}

Output :-

Mighty Gumball, Inc.

Java-enabled Standing Gumball Model #2004

Inventory: 2 gumballs

Machine is waiting for quarter

You inserted a quarter

You turned...

A gumball comes rolling out the slot...

Mighty Gumball, Inc.

Java-enabled Standing Gumball Model #2004

Inventory: 1 gumball

Machine is waiting for quarter

You inserted a quarter

You turned...

A gumball comes rolling out the slot...

Oops, out of gumballs!

You can't insert a quarter, the machine is sold out

You turned, but there are no gumballs

No gumball dispensed

The gumball machine was just refilled; it's new count is: 5

You inserted a quarter

You turned...

A gumball comes rolling out the slot...

Mighty Gumball, Inc.

Java-enabled Standing Gumball Model #2004

Inventory: 4 gumballs

Machine is waiting for quarter

Q.10: Write a java program to implement Adapter pattern to design Heart Model to Beat Model.

BeatModelInterface.java

**package** program\_10;

**public** **interface** BeatModelInterface {

**void** initialize();

**void** on();

**void** off();

**void** setBPM(**int** bpm);

**int** getBPM();

**void** registerObserver(BeatObserver o);

**void** removeObserver(BeatObserver o);

**void** registerObserver(BPMObserver o);

**void** removeObserver(BPMObserver o);

}

BeatObserver.java

**package** program\_10;

**public** **interface** BeatObserver {

**void** updateBeat();

}

BPMObserver.java

**package** program\_10;

**public** **interface** BPMObserver {

**void** updateBPM();

}

ControllerInterface.java

**package** program\_10;

**public** **interface** ControllerInterface {

**void** start();

**void** stop();

**void** increaseBPM();

**void** decreaseBPM();

**void** setBPM(**int** bpm);

}

HeartModelInterface.java

**package** program\_10;

**public** **interface** HeartModelInterface {

**int** getHeartRate();

**void** registerObserver(BeatObserver o);

**void** removeObserver(BeatObserver o);

**void** registerObserver(BPMObserver o);

**void** removeObserver(BPMObserver o);

}

BeatBar.java

package program\_10;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class BeatBar extends JProgressBar implements Runnable {

JProgressBar progressBar;

Thread thread;

public BeatBar() {

thread = new Thread(this);

setMaximum(100);

thread.start();

}

public void run() {

for(;;) {

int value = getValue();

value = (int)(value \* .75);

setValue(value);

repaint();

try {

Thread.sleep(50);

} catch (Exception e) {};

}

}

}

BeatController.java

**package** program\_10;

**public** **class** BeatController **implements** ControllerInterface {

BeatModelInterface model;

DJView view;

**public** BeatController(BeatModelInterface model) {

**this**.model = model;

view = **new** DJView(**this**, model);

view.createView();

view.createControls();

view.disableStopMenuItem();

view.enableStartMenuItem();

model.initialize();

}

**public** **void** start() {

model.on();

view.disableStartMenuItem();

view.enableStopMenuItem();

}

**public** **void** stop() {

model.off();

view.disableStopMenuItem();

view.enableStartMenuItem();

}

**public** **void** increaseBPM() {

**int** bpm = model.getBPM();

model.setBPM(bpm + 1);

}

**public** **void** decreaseBPM() {

**int** bpm = model.getBPM();

model.setBPM(bpm - 1);

}

**public** **void** setBPM(**int** bpm) {

model.setBPM(bpm);

}

}

BeatModel.java

package program\_10;

import javax.sound.midi.\*;

import java.util.\*;

public class BeatModel implements BeatModelInterface, MetaEventListener {

Sequencer sequencer;

ArrayList beatObservers = new ArrayList();

ArrayList bpmObservers = new ArrayList();

int bpm = 90;

Sequence sequence;

Track track;

public void initialize() {

setUpMidi();

buildTrackAndStart();

}

public void on() {

sequencer.start();

setBPM(90);

}

public void off() {

setBPM(0);

sequencer.stop();

}

public void setBPM(int bpm) {

this.bpm = bpm;

sequencer.setTempoInBPM(getBPM());

notifyBPMObservers();

}

public int getBPM() {

return bpm;

}

void beatEvent() {

notifyBeatObservers();

}

public void registerObserver(BeatObserver o) {

beatObservers.add(o);

}

public void notifyBeatObservers() {

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

BeatObserver observer = (BeatObserver)beatObservers.get(i);

observer.updateBeat();

}

}

public void registerObserver(BPMObserver o) {

bpmObservers.add(o);

}

public void notifyBPMObservers() {

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

BPMObserver observer = (BPMObserver)bpmObservers.get(i);

observer.updateBPM();

}

}

public void removeObserver(BeatObserver o) {

int i = beatObservers.indexOf(o);

if (i >= 0) {

beatObservers.remove(i);

}

}

public void removeObserver(BPMObserver o) {

int i = bpmObservers.indexOf(o);

if (i >= 0) {

bpmObservers.remove(i);

}

}

public void meta(MetaMessage message) {

if (message.getType() == 47) {

beatEvent();

sequencer.start();

setBPM(getBPM());

}

}

public void setUpMidi() {

try {

sequencer = MidiSystem.getSequencer();

sequencer.open();

sequencer.addMetaEventListener(this);

sequence = new Sequence(Sequence.PPQ,4);

track = sequence.createTrack();

sequencer.setTempoInBPM(getBPM());

} catch(Exception e) {

e.printStackTrace();

}

}

public void buildTrackAndStart() {

int[] trackList = {35, 0, 46, 0};

sequence.deleteTrack(null);

track = sequence.createTrack();

makeTracks(trackList);

track.add(makeEvent(192,9,1,0,4));

try {

sequencer.setSequence(sequence);

} catch(Exception e) {

e.printStackTrace();

}

}

public void makeTracks(int[] list) {

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

int key = list[i];

if (key != 0) {

track.add(makeEvent(144,9,key, 100, i));

track.add(makeEvent(128,9,key, 100, i+1));

}

}

}

public MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {

MidiEvent event = null;

try {

ShortMessage a = new ShortMessage();

a.setMessage(comd, chan, one, two);

event = new MidiEvent(a, tick);

} catch(Exception e) {

e.printStackTrace();

}

return event;

}

}

DJView.java

package program\_10;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class DJView implements ActionListener, BeatObserver, BPMObserver {

BeatModelInterface model;

ControllerInterface controller;

JFrame viewFrame;

JPanel viewPanel;

BeatBar beatBar;

JLabel bpmOutputLabel;

JFrame controlFrame;

JPanel controlPanel;

JLabel bpmLabel;

JTextField bpmTextField;

JButton setBPMButton;

JButton increaseBPMButton;

JButton decreaseBPMButton;

JMenuBar menuBar;

JMenu menu;

JMenuItem startMenuItem;

JMenuItem stopMenuItem;

public DJView(ControllerInterface controller, BeatModelInterface model) {

this.controller = controller;

this.model = model;

model.registerObserver((BeatObserver)this);

model.registerObserver((BPMObserver)this);

}

public void createView() {

// Create all Swing components here

viewPanel = new JPanel(new GridLayout(1, 2));

viewFrame = new JFrame("View");

viewFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

viewFrame.setSize(new Dimension(100, 80));

bpmOutputLabel = new JLabel("offline", SwingConstants.CENTER);

beatBar = new BeatBar();

beatBar.setValue(0);

JPanel bpmPanel = new JPanel(new GridLayout(2, 1));

bpmPanel.add(beatBar);

bpmPanel.add(bpmOutputLabel);

viewPanel.add(bpmPanel);

viewFrame.getContentPane().add(viewPanel, BorderLayout.CENTER);

viewFrame.pack();

viewFrame.setVisible(true);

}

public void createControls() {

// Create all Swing components here

JFrame.setDefaultLookAndFeelDecorated(true);

controlFrame = new JFrame("Control");

controlFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

controlFrame.setSize(new Dimension(100, 80));

controlPanel = new JPanel(new GridLayout(1, 2));

menuBar = new JMenuBar();

menu = new JMenu("DJ Control");

startMenuItem = new JMenuItem("Start");

menu.add(startMenuItem);

startMenuItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

controller.start();

}

});

stopMenuItem = new JMenuItem("Stop");

menu.add(stopMenuItem);

stopMenuItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

controller.stop();

}

});

JMenuItem exit = new JMenuItem("Quit");

exit.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

System.exit(0);

}

});

menu.add(exit);

menuBar.add(menu);

controlFrame.setJMenuBar(menuBar);

bpmTextField = new JTextField(2);

bpmLabel = new JLabel("Enter BPM:", SwingConstants.RIGHT);

setBPMButton = new JButton("Set");

setBPMButton.setSize(new Dimension(10,40));

increaseBPMButton = new JButton(">>");

decreaseBPMButton = new JButton("<<");

setBPMButton.addActionListener(this);

increaseBPMButton.addActionListener(this);

decreaseBPMButton.addActionListener(this);

JPanel buttonPanel = new JPanel(new GridLayout(1, 2));

buttonPanel.add(decreaseBPMButton);

buttonPanel.add(increaseBPMButton);

JPanel enterPanel = new JPanel(new GridLayout(1, 2));

enterPanel.add(bpmLabel);

enterPanel.add(bpmTextField);

JPanel insideControlPanel = new JPanel(new GridLayout(3, 1));

insideControlPanel.add(enterPanel);

insideControlPanel.add(setBPMButton);

insideControlPanel.add(buttonPanel);

controlPanel.add(insideControlPanel);

bpmLabel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));

bpmOutputLabel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));

controlFrame.getRootPane().setDefaultButton(setBPMButton);

controlFrame.getContentPane().add(controlPanel, BorderLayout.CENTER);

controlFrame.pack();

controlFrame.setVisible(true);

}

public void enableStopMenuItem() {

stopMenuItem.setEnabled(true);

}

public void disableStopMenuItem() {

stopMenuItem.setEnabled(false);

}

public void enableStartMenuItem() {

startMenuItem.setEnabled(true);

}

public void disableStartMenuItem() {

startMenuItem.setEnabled(false);

}

public void actionPerformed(ActionEvent event) {

if (event.getSource() == setBPMButton) {

int bpm = Integer.parseInt(bpmTextField.getText());

controller.setBPM(bpm);

} else if (event.getSource() == increaseBPMButton) {

controller.increaseBPM();

} else if (event.getSource() == decreaseBPMButton) {

controller.decreaseBPM();

}

}

public void updateBPM() {

if (model != null) {

int bpm = model.getBPM();

if (bpm == 0) {

if (bpmOutputLabel != null) {

bpmOutputLabel.setText("offline");

}

} else {

if (bpmOutputLabel != null) {

bpmOutputLabel.setText("Current BPM: " + model.getBPM());

}

}

}

}

public void updateBeat() {

if (beatBar != null) {

beatBar.setValue(100);

}

}

}

DJTestDrive.java

**package** program\_10;

**public** **class** DJTestDrive {

**public** **static** **void** main (String[] args) {

BeatModelInterface model = **new** BeatModel();

ControllerInterface controller = **new** BeatController(model);

}

}

HeartAdapter.java

**package** program\_10;

**public** **class** HeartAdapter **implements** BeatModelInterface {

HeartModelInterface heart;

**public** HeartAdapter(HeartModelInterface heart) {

**this**.heart = heart;

}

**public** **void** initialize() {}

**public** **void** on() {}

**public** **void** off() {}

**public** **int** getBPM() {

**return** heart.getHeartRate();

}

**public** **void** setBPM(**int** bpm) {}

**public** **void** registerObserver(BeatObserver o) {

heart.registerObserver(o);

}

**public** **void** removeObserver(BeatObserver o) {

heart.removeObserver(o);

}

**public** **void** registerObserver(BPMObserver o) {

heart.registerObserver(o);

}

**public** **void** removeObserver(BPMObserver o) {

heart.removeObserver(o);

}

}

HeartController.java

**package** program\_10;

**public** **class** HeartController **implements** ControllerInterface {

HeartModelInterface model;

DJView view;

**public** HeartController(HeartModelInterface model) {

**this**.model = model;

view = **new** DJView(**this**, **new** HeartAdapter(model));

view.createView();

view.createControls();

view.disableStopMenuItem();

view.disableStartMenuItem();

}

**public** **void** start() {}

**public** **void** stop() {}

**public** **void** increaseBPM() {}

**public** **void** decreaseBPM() {}

**public** **void** setBPM(**int** bpm) {}

}

HeartModel.java

**package** program\_10;

**import** java.util.\*;

**public** **class** HeartModel **implements** HeartModelInterface, Runnable {

ArrayList beatObservers = **new** ArrayList();

ArrayList bpmObservers = **new** ArrayList();

**int** time = 1000;

**int** bpm = 90;

Random random = **new** Random(System.*currentTimeMillis*());

Thread thread;

**public** HeartModel() {

thread = **new** Thread(**this**);

thread.start();

}

**public** **void** run() {

**int** lastrate = -1;

**for**(;;) {

**int** change = random.nextInt(10);

**if** (random.nextInt(2) == 0) {

change = 0 - change;

}

**int** rate = 60000/(time + change);

**if** (rate < 120 && rate > 50) {

time += change;

notifyBeatObservers();

**if** (rate != lastrate) {

lastrate = rate;

notifyBPMObservers();

}

}

**try** {

Thread.*sleep*(time);

} **catch** (Exception e) {}

}

}

**public** **int** getHeartRate() {

**return** 60000/time;

}

**public** **void** registerObserver(BeatObserver o) {

beatObservers.add(o);

}

**public** **void** removeObserver(BeatObserver o) {

**int** i = beatObservers.indexOf(o);

**if** (i >= 0) {

beatObservers.remove(i);

}

}

**public** **void** notifyBeatObservers() {

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

BeatObserver observer = (BeatObserver)beatObservers.get(i);

observer.updateBeat();

}

}

**public** **void** registerObserver(BPMObserver o) {

bpmObservers.add(o);

}

**public** **void** removeObserver(BPMObserver o) {

**int** i = bpmObservers.indexOf(o);

**if** (i >= 0) {

bpmObservers.remove(i);

}

}

**public** **void** notifyBPMObservers() {

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

BPMObserver observer = (BPMObserver)bpmObservers.get(i);

observer.updateBPM();

}

}

}

HeartTestDrive.java

**package** program\_10;

**public** **class** HeartTestDrive {

**public** **static** **void** main (String[] args) {

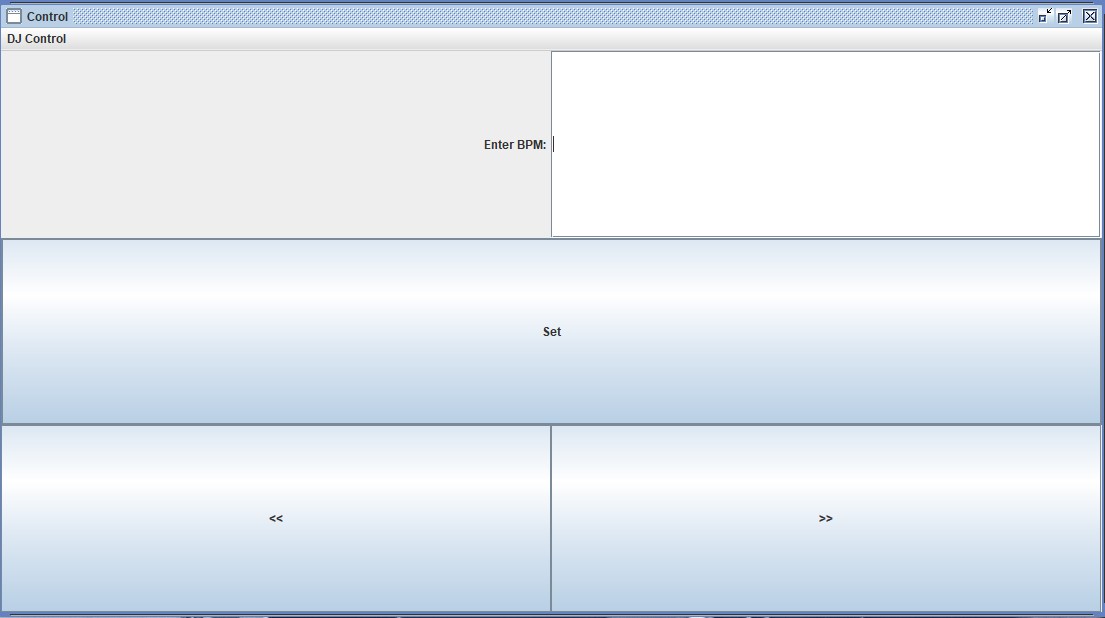
HeartModel heartModel = **new** HeartModel();

ControllerInterface model = **new** HeartController(heartModel);

}

}

Output :-



Q.11: Design simple HR Application using Spring Framework