

## SUBJECT : SADP SLIP SOLUTION.

---

### ASSIGNMENT 1

**1)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()**

#### **WeatherStationHeatIndex.java**

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

#### **CurrentConditionDisplay.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");
    }
}

```

## DisplayElement.java

```

package weatherobservable;

public interface DisplayElement {
    public void display();
}

```

## 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");
        }
    }
}

```

## 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)) -
                        (0.00000000218429 * (t * t * t * rh * rh))
+
                        (0.0000000000843296 * (t * t * rh * rh *
rh)) -
                        (0.00000000000481975 * (t * t * t * rh * rh
* rh)));
                display();
            }
        }

        public void display() {
            System.out.println("Heat index is " + heatIndex);
        }
    }
}

```

## 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);
}
}

```

## 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;
    }
}

```

## WeatherStation.java

```

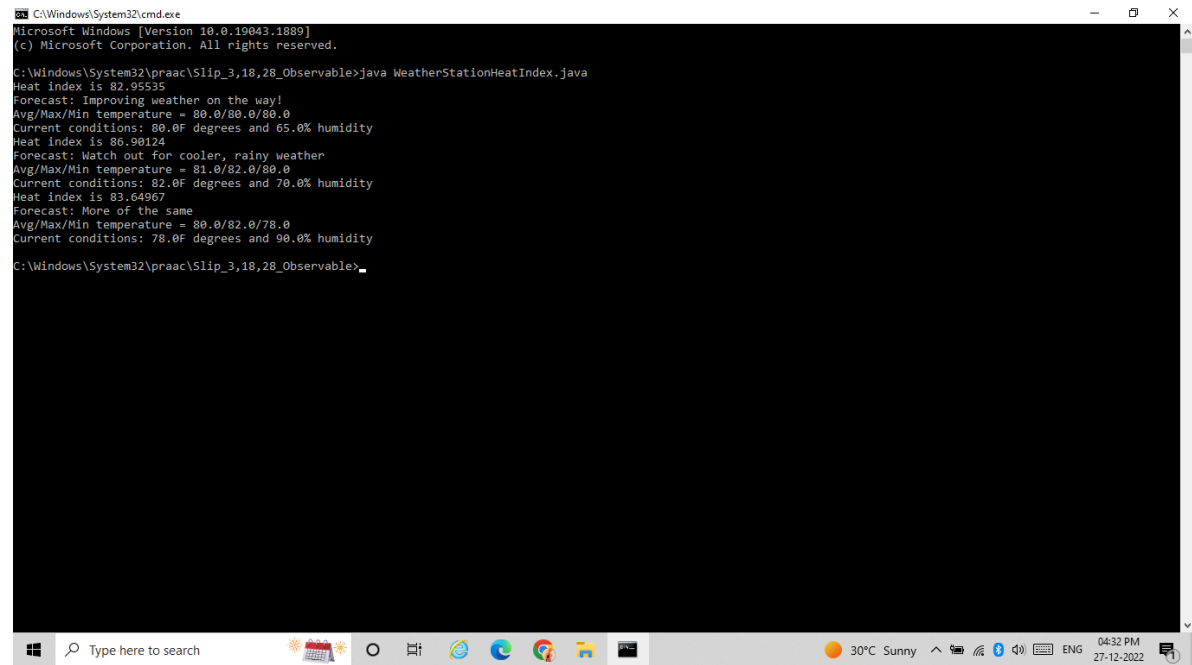
package headfirst.observer.weatherobservable;
public class WeatherStation {
    public static void main(String[] args) {
        WeatherData weatherData = new WeatherData();
        CurrentConditionsDisplay currentConditions = new
CurrentConditionsDisplay(weatherData);
        StatisticsDisplay statisticsDisplay = 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);  
    }  
}
```

## OUTPUT



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32\praaac\Slip_3,18,28_Observable>java WeatherStationHeatIndex.java
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
C:\Windows\System32\praaac\Slip_3,18,28_Observable>
```

The screenshot shows a Windows command prompt window with a black background and white text. The window title is "C:\Windows\System32\cmd.exe". The output of the Java program is as follows:

```
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
```

The Windows taskbar is visible at the bottom, showing the search bar, task view button, and several application icons. The system tray on the right shows the date and time as "04:32 PM 27-12-2022" and the language as "ENG".

## ASSIGNMENT 2

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

#### InputTest.java

```
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();
        }
    }
}
```

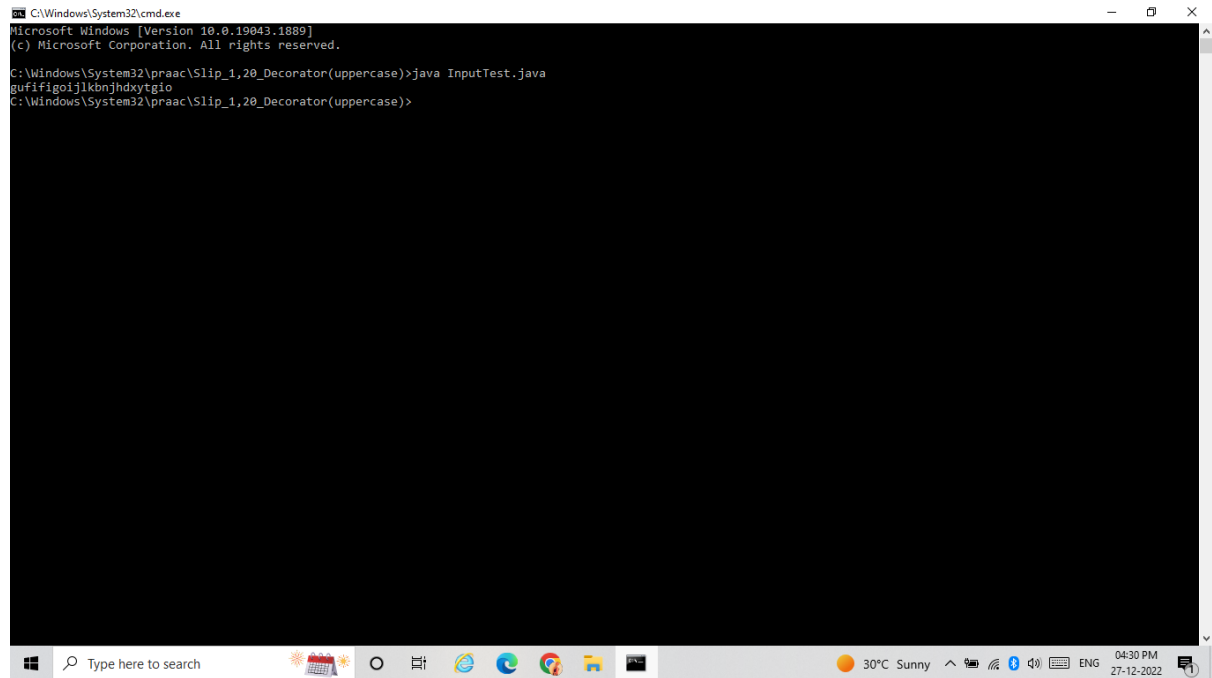
#### LowerCaseInputStream.java

```
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



The screenshot shows a Windows Command Prompt window titled "C:\Windows\System32\cmd.exe". The window displays the following text:

```
Microsoft Windows [Version 10.0.19043.1889]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Windows\System32\praa\Slip_1,20_Decorator(uppercase)>java InputTest.java  
gufifigoiJlkbjhdxytgio  
C:\Windows\System32\praa\Slip_1,20_Decorator(uppercase)>
```

The command prompt is running in a directory path that appears to be "C:\Windows\System32\praa\Slip\_1,20\_Decorator(uppercase)". The user has entered the command "java InputTest.java" and the output "gufifigoiJlkbjhdxytgio" is displayed. The Windows taskbar is visible at the bottom, showing the search bar, task view button, and several application icons. The system tray on the right indicates the date and time as "04:30 PM 27-12-2022" and the language as "ENG".

## ASSIGNMENT 3

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

### Pizza.java

```
package factorypackage;

import java.util.ArrayList;

public abstract class pizza
{
    String name;
    String dough;
    String sauce;
    ArrayList topping=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<topping.size();i++)
        {
            System.out.println(".." +topping.get(i));
        }
    }

    void bake()
    {
        System.out.println("bake for 25 mins at 350");
    }
    void cut()
    {
        System.out.println("cutting the pizza into diagonal slice");
    }
    void box()
    {
        System.out.println("place pizza in offical pizza store 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<topping.size();i++)
```



```

        {
            display.append((String) topping.get(i) + "\n");
        }
        return display.toString();
    }
}

```

## **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;
    }
}

```

## **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 return null;
    }
}

```

## **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";

        topping.add("shredded mozzarella cheese");
    }
    void cut()
    {
        System.out.println("cutting the pizza into diagonal slice");
    }
}

```

## **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 return null;
    }
}

```

## **Nystylecheesepizza.java**

```

package factorypackage;

public class nystylecheesepizza extends pizza
{
    public nystylecheesepizza()
    {
        name="ny style deep dish cheese pizza";
        dough="extra thick crust dough";
        sauce="plum tomato sauce";

        topping.add("grated regginao cheese");
    }
    void cut()
    {
        System.out.println("cutting the pizza into diagonal slice");
    }
}

```

## **Nystyleveggiepizza.java**

```

package factorypackage;

public class nystyleveggiepizza extends pizza
{
    public nystyleveggiepizza()
    {
        name="ny style deep dish cheese pizza";
        dough="extra thick crust dough";
        sauce="plum tomato sauce";

        topping.add("shredded mozzarella cheese");
        topping.add("black olives");
        topping.add("spinech");
        topping.add("eggplant");
        topping.add("tomato");
    }
    void cut()
    {
        System.out.println("cutting the pizza into diagonal slice");
    }
}

```

## OUTPUT

```

C:\Windows\System32\cmd.exe
C:\Windows\System32\prAAC\Slip_4,19,30_Pizza>java pizzatestdrive
making any style deep dish cheese pizza...
preparingny style deep dish cheese pizza
Tossing dough..
Adding Sauce..
Adding toppings
..grated reggiano cheese
bake for 25 mins at 350
cutting the pizza into diagonal slice
place pizza in official pizza store box
enter orderny style deep dish cheese pizza

making achicago style deep dish cheese pizza...
preparingchicago style deep dish cheese pizza
Tossing dough..
Adding Sauce..
Adding toppings
..shredded mozzarella cheese
bake for 25 mins at 350
cutting the pizza into diagonal slice
place pizza in official pizza store box
joel orderchicago style deep dish cheese pizza

making any style deep dish cheese pizza...
preparingny style deep dish cheese pizza
Tossing dough..
Adding Sauce..
Adding toppings
..shredded mozzarella cheese
..black olives
..spinech
..eggplant
..tomato
bake for 25 mins at 350
cutting the pizza into diagonal slice
place pizza in official pizza store box
enter orderny style deep dish cheese pizza

making achicago style deep dish cheese pizza...
preparingchicago style deep dish cheese pizza
Tossing dough..
Adding Sauce..
Adding toppings
..shredded mozzarella cheese

```

```

C:\Windows\System32\prAAC\Slip_4,19,30_Pizza>
enter orderny style deep dish cheese pizza
making achicago style deep dish cheese pizza...
preparingchicago style deep dish cheese pizza
Tossing dough..
Adding Sauce..
Adding toppings
..shredded mozzarella cheese
bake for 25 mins at 350
cutting the pizza into diagonal slice
place pizza in official pizza store box
joel orderchicago style deep dish cheese pizza

making any style deep dish cheese pizza...
preparingny style deep dish cheese pizza
Tossing dough..
Adding Sauce..
Adding toppings
..shredded mozzarella cheese
..black olives
..spinech
..eggplant
..tomato
bake for 25 mins at 350
cutting the pizza into diagonal slice
place pizza in official pizza store box
enter orderny style deep dish cheese pizza

making achicago style deep dish cheese pizza...
preparingchicago style deep dish cheese pizza
Tossing dough..
Adding Sauce..
Adding toppings
..shredded mozzarella cheese
..black olives
..spinech
..eggplant
bake for 25 mins at 350
cutting the pizza into diagonal slice
place pizza in official pizza store box
joel orderchicago style deep dish cheese pizza

C:\Windows\System32\prAAC\Slip_4,19,30_Pizza>

```

## ASSIGNMENT 4

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

#### SingletonTestDrive.java

```
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);
    }
}
```

#### Singleton.java

```
package singletonpackage;

public class Singleton
{
    protected static Singleton uniqueInstance;

    protected Singleton(){}

    public static synchronized Singleton getInstance()
    {
        if(uniqueInstance==null)
        {
            uniqueInstance=new Singleton();
        }
        return uniqueInstance;
    }
}
```

#### Hottersingleton.java

```
package singletonpackage;

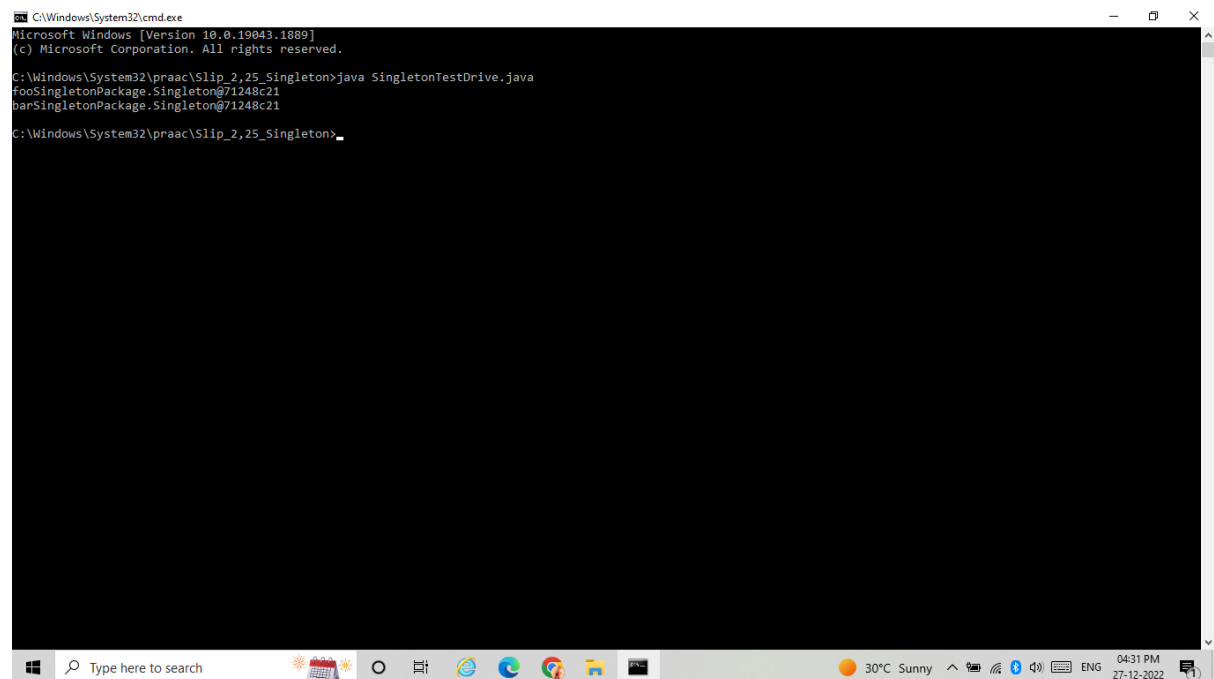
public class HotterSingleton extends Singleton
{
    private HotterSingleton()
    {
        super();
    }
}
```

#### Coolersingleton.java

```
package singletonpackage;
public class CoolerSingleton extends Singleton
{
    protected static Singleton uniqueInstance;

    private CoolerSingleton()
    {
        super();
    }
}
```

## OUTPUT



A screenshot of a Windows command prompt window. The title bar reads "C:\Windows\System32\cmd.exe". The window content shows the following text:

```
Microsoft Windows [Version 10.0.19043.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32\pracc\Slip_2,25_Singleton>java SingletonTestDrive.java
fooSingletonPackage.Singleton@71248c21
barSingletonPackage.Singleton@71248c21

C:\Windows\System32\pracc\Slip_2,25_Singleton>
```

The command prompt is running on a Windows 10 desktop. The taskbar at the bottom shows the Start button, a search bar with the text "Type here to search", and several application icons including Calendar, Photos, File Explorer, and Microsoft Edge. The system tray on the right shows the date and time as "04:31 PM 27-12-2022" and the language as "ENG".

## ASSIGNMENT 5

### 5) Write a Java Program to implement command pattern to test Remote Control.

#### RemoteControlTestDrive.java

```
import simpleremote.*;

public class RemoteControlTest
{
    public static void main(String[] args)
    {
        SimpleRemoteControl remote = new SimpleRemoteControl();

        Light light = new Light();

        GarageDoor garageDoor = new GarageDoor();

        LightOnCommand lightOn = new LightOnCommand(light);

        GarageDoorOpenCommand garageOpen = new
GarageDoorOpenCommand(garageDoor);

        remote.setCommand(lightOn);

        remote.buttonWasPressed();

        remote.setCommand(garageOpen);

        remote.buttonWasPressed();

    }
}
```

#### Command.java

```
package simpleremote;

public interface Command
{
    public void execute();
}
```

#### GarageDoor.java

```
package simpleremote;

public class GarageDoor
{

    public GarageDoor() {

    }

    public void up()
    {

        System.out.println("Garage Door is Open");

    }

    public void down()
    {

        System.out.println("Garage Door is Closed");

    }

    public void stop()
    {

        System.out.println("Garage Door is Stopped");

    }

    public void lightOn()
    {

        System.out.println("Garage light is on");

    }

    public void lightOff()
    {

        System.out.println("Garage light is off");

    }

}
```

### **GarageDooropencommand.java**

```
package simpleremote;
```

```

public class GarageDoorOpenCommand implements Command
{
    GarageDoor garageDoor;

    public GarageDoorOpenCommand(GarageDoor garageDoor)
    {
        this.garageDoor = garageDoor;
    }

    public void execute()
    {
        garageDoor.up();
    }
}

```

## Light.java

```

package simpleremote;

public class Light
{
    public Light() {

    }

    public void on()
    {
        System.out.println("Light is on");
    }

    public void off()
    {
        System.out.println("Light is off");
    }
}

```

## Lightoffcommand.java



```
package simplereMOTE;

public class LightOffCommand implements Command
{
    Light light;

    public LightOffCommand(Light light)
    {
        this.light = light;
    }

    public void execute()
    {
        light.off();
    }
}
```

### **Lightoncommand.java**

```
package simplereMOTE;

public class LightOnCommand implements Command
{
    Light light;

    public LightOnCommand(Light light)
    {
        this.light = light;
    }

    public void execute()
    {
        light.on();
    }
}
```

### **SimplereMOTEcontrol.java**

```
package simplereMOTE;
```

```
import java.util.*;

//
// This is the invoker
//

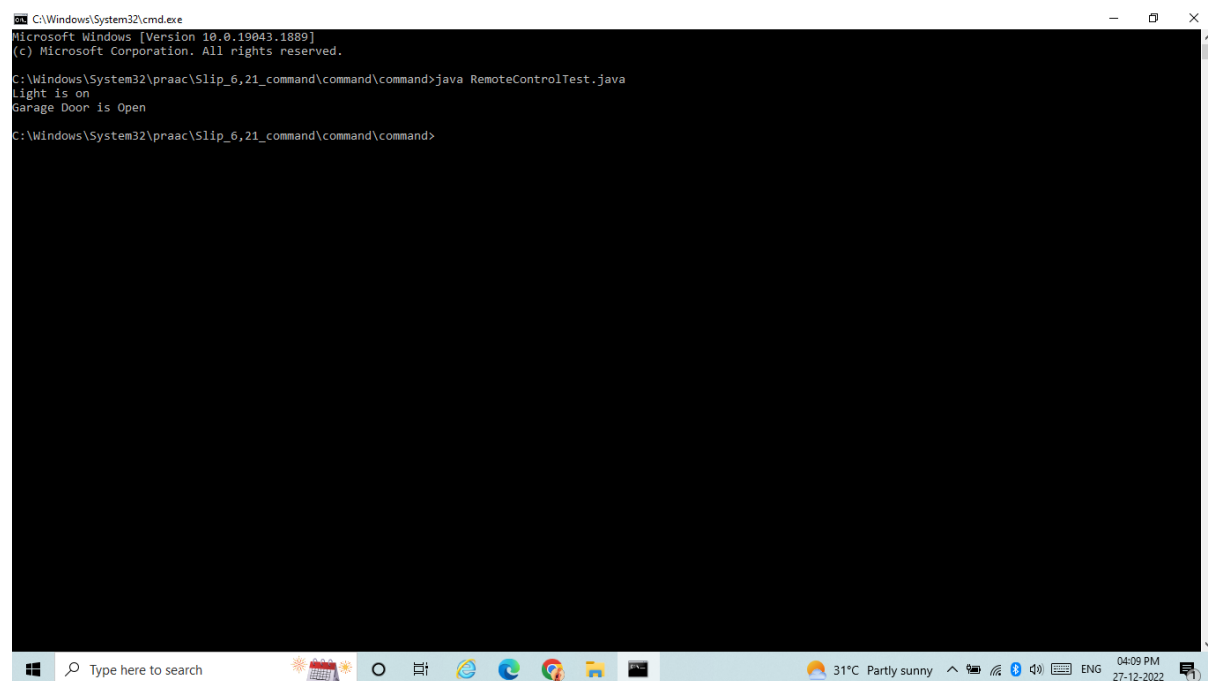
public class SimpleRemoteControl
{
    Command slot;

    public SimpleRemoteControl() {}

    public void setCommand(Command command)
    {
        slot = command;
    }

    public void buttonWasPressed()
    {
        slot.execute();
    }
}
```

## OUTPUT



A screenshot of a Windows command prompt window. The title bar reads "C:\Windows\System32\cmd.exe". The window content shows the following text:

```
Microsoft Windows [Version 10.0.19043.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32\praac\Slip_6,21_command\command>java RemoteControlTest.java
Light is on
Garage Door is Open

C:\Windows\System32\praac\Slip_6,21_command\command>
```

The Windows taskbar is visible at the bottom, showing the search bar, task view, and several application icons. The system tray on the right indicates a temperature of 31°C, "Partly sunny" weather, and the date/time "04:09 PM 27-12-2022".

## ASSIGNMENT 6

### 6) Write a Java Program to implement undo command to test Ceiling fan.

#### Remoteloader.java

```
import undo.*;

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();

    }
}
```

```
}
```

## **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;
    }
}

```

## **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();
        }
    }
}

```

## 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();

        }

    }

}

```

### **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();

        }

    }

}

```

### **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();

        }

    }

}

```

## **Command.java**

```

package undo;

public interface Command
{

    public void execute();
}

```

```
        public void undo();
    }
}
```

## **Nocommand.java**

```
package undo;

public class NoCommand implements Command
{
    public void execute() { }

    public void undo() { }
}
}
```

## **Remotecontrolwithundo.java**

```
package undo;

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 stringBuffer = new StringBuffer();

        stringBuffer.append("\n----- Remote Control ----- \n");
    }

```

```

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

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

        }

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

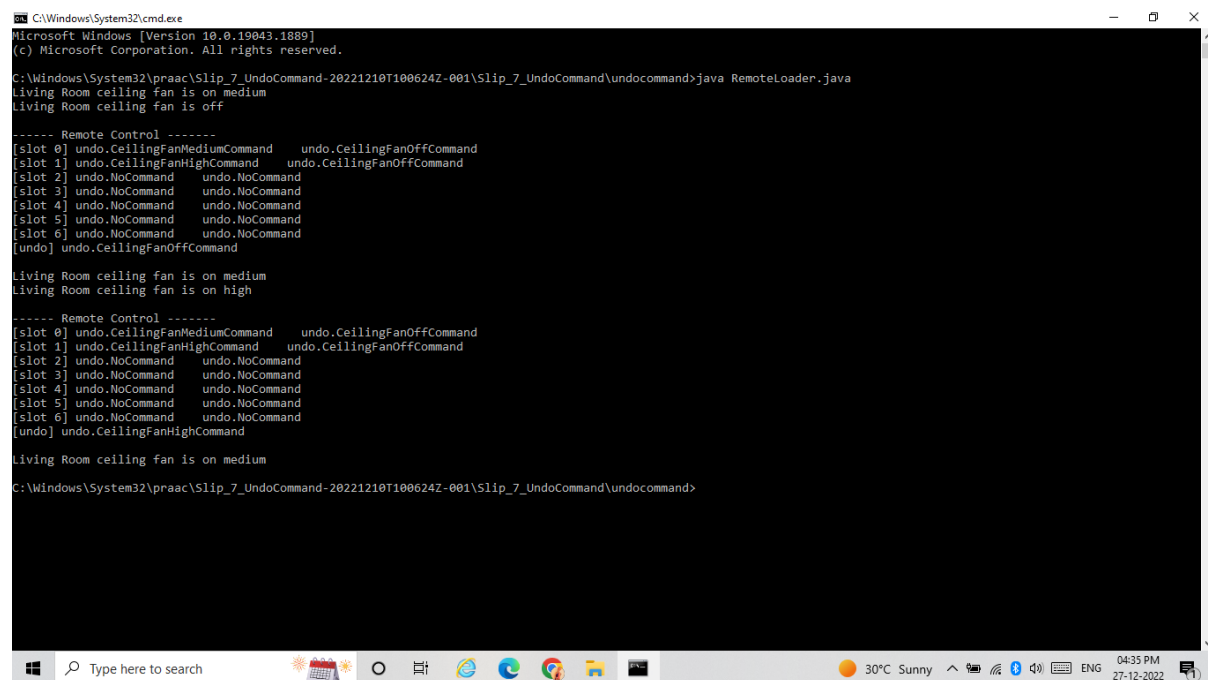
        return stringBuffer.toString();

    }

}

```

## OUTPUT



```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32\pracc\Slip_7_UndoCommand-20221210T100624Z-001\Slip_7_UndoCommand\undocommand>java RemoteLoader.java
Living Room ceiling fan is on medium
Living Room ceiling fan is off

----- Remote Control -----
[slot 0] undo.CeilingFanMediumCommand    undo.CeilingFanOffCommand
[slot 1] undo.CeilingFanHighCommand      undo.CeilingFanOffCommand
[slot 2] undo.NoCommand                  undo.NoCommand
[slot 3] undo.NoCommand                  undo.NoCommand
[slot 4] undo.NoCommand                  undo.NoCommand
[slot 5] undo.NoCommand                  undo.NoCommand
[slot 6] undo.NoCommand                  undo.NoCommand
[undo] undo.CeilingFanOffCommand

Living Room ceiling fan is on medium
Living Room ceiling fan is on high

----- Remote Control -----
[slot 0] undo.CeilingFanMediumCommand    undo.CeilingFanOffCommand
[slot 1] undo.CeilingFanHighCommand      undo.CeilingFanOffCommand
[slot 2] undo.NoCommand                  undo.NoCommand
[slot 3] undo.NoCommand                  undo.NoCommand
[slot 4] undo.NoCommand                  undo.NoCommand
[slot 5] undo.NoCommand                  undo.NoCommand
[slot 6] undo.NoCommand                  undo.NoCommand
[undo] undo.CeilingFanHighCommand

Living Room ceiling fan is on medium

C:\Windows\System32\pracc\Slip_7_UndoCommand-20221210T100624Z-001\Slip_7_UndoCommand\undocommand>

```

## ASSSIGNMENT 7

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

**EI.java**

```
import iterenum.*;
import java.util.*;
public class EI {
    public static void main (String args[]) {
        Vector v = new Vector(Arrays.asList(args));
        Enumeration enumeration = v.elements();
        while (enumeration.hasMoreElements()) {
            System.out.println(enumeration.nextElement());
        }
        Iterator iterator = v.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

**EnumerationIterator.java**

```
package iterenum;
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();
    }
}
```

**OUTPUT**

```
C:\Users\Administrator\Desktop\Iterator_Slip5>javac EI.java
Note: EI.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\Users\Administrator\Desktop\Iterator_Slip5>java EI 5 8 9 7 4 2
5
8
9
7
4
2
5
8
9
7
4
2

C:\Users\Administrator\Desktop\Iterator_Slip5>
```

## ASSSIGNMENT 8

8)write a java program to implement iterator pattern for designing menu like breakfast, lunch, dinner menu.

### MenuTestDrive.java

```
import dinermenu.*;
import java.util.*;
public class MenuTestDrive {
    public static void main(String args[]) {
        PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();
        DinerMenu = new DinerMenu();
        CafeMenu cafeMenu = new CafeMenu();
        Waitress waitress = new Waitress(pancakeHouseMenu,
dinerMenu, cafeMenu);
        waitress.printMenu();
        waitress.printVegetarianMenu();
        System.out.println("\nCustomer asks, is the Hotdog
vegetarian?");
        System.out.print("Waitress says: ");
        if (waitress.isItemVegetarian("Hotdog")) {
            System.out.println("Yes");
        } else {
            System.out.println("No");
        }
        System.out.println("\nCustomer asks, are the Waffles
vegetarian?");
        System.out.print("Waitress says: ");
        if (waitress.isItemVegetarian("Waffles")) {
            System.out.println("Yes");
        } else {
            System.out.println("No");
        }
    }
}
```

### AlternatingDinnerMenuIterator.java

```
package dinermenu;
import java.util.Iterator;
import java.util.Calendar;
public class AlternatingDinerMenuIterator implements Iterator {
    MenuItem[] items;
    int position;
    public AlternatingDinerMenuIterator(MenuItem[] items) {
        this.items = items;
        Calendar rightNow = Calendar.getInstance();
        position = rightNow.DAY_OF_WEEK % 2;
    }
    public Object next() {
        MenuItem menuItem = items[position];
        position = position + 2;
        return menuItem;
    }
    public boolean hasNext() {
        if (position >= items.length || items[position] == null) {
            return false;
        } else {
            return true;
        }
    }
}
```

```

        public void remove() {
            throw new UnsupportedOperationException(
                "Alternating Diner Menu Iterator does not support
remove()");
        }
    }

```

### **CafeMenu.java**

```

package dinermenu;
import java.util.*;
public class CafeMenu implements Menu {
    Hashtable menuItems = new Hashtable();
    public CafeMenu() {
        addItem("Veggie Burger and Air Fries",
            "Veggie burger on a whole wheat bun, lettuce,
tomato, and fries",
            true, 3.99);
        addItem("Soup of the day",
            "A cup of the soup of the day, with a side salad",
            false, 3.69);
        addItem("Burrito",
            "A large burrito, with whole pinto beans, salsa,
guacamole",
            true, 4.29);
    }
    public void addItem(String name, String description,
        boolean vegetarian, double price)
    {
        MenuItem menuItem = new MenuItem(name, description,
vegetarian, price);
        menuItems.put(menuItem.getName(), menuItem);
    }
    public Hashtable getItems() {
        return menuItems;
    }
    public Iterator createIterator() {
        return menuItems.values().iterator();
    }
}

```

### **DinnerMenu.java**

```

package dinermenu;
import java.util.Iterator;
public class DinnerMenu implements Menu {
    static final int MAX_ITEMS = 6;
    int numberOfItems = 0;
    MenuItem[] menuItems;
    public DinnerMenu() {
        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",
            "A medly of 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);
        //return new AlternatingDinerMenuIterator(menuItems);
    }

    // other menu methods here
}

```

### **DinnerMenuIterator.java**

```

package dinermenu;

import java.util.Iterator;

public class DinerMenuIterator implements Iterator {
    MenuItem[] list;
    int position = 0;

    public DinerMenuIterator(MenuItem[] list) {
        this.list = list;
    }

    public Object next() {
        MenuItem menuItem = list[position];
        position = position + 1;
        return menuItem;
    }

    public boolean hasNext() {
        if (position >= list.length || list[position] == null) {
            return false;
        } else {
            return true;
        }
    }

    public void remove() {
        if (position <= 0) {

```



```

        throw new IllegalStateException
            ("You can't remove an item until you've done
at least one next()");
    }
    if (list[position-1] != null) {
        for (int i = position-1; i < (list.length-1); i++) {
            list[i] = list[i+1];
        }
        list[list.length-1] = null;
    }
}

```

### **Menu.java**

```

package dinermenu;

import java.util.Iterator;

public interface Menu {
    public Iterator createIterator();
}

```

### **MenuItem.java**

```

package dinermenu;

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;
    }
}

```

### **PanCakeHouseMenu.java**

```

package dinermenu;

import java.util.ArrayList;
import java.util.Iterator;

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, and blueberry
syrup",
            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 menuItems.iterator();
    }

    // other menu methods here
}

```

## **Waitress.java**

```

package dinermenu;

import java.util.Iterator;

public class Waitress {
    Menu pancakeHouseMenu;
}

```

```

Menu dinerMenu;
Menu cafeMenu;

public Waitress(Menu pancakeHouseMenu, Menu dinerMenu, Menu
cafeMenu) {
    this.pancakeHouseMenu = pancakeHouseMenu;
    this.dinerMenu = dinerMenu;
    this.cafeMenu = cafeMenu;
}

public void printMenu() {
    Iterator pancakeIterator =
pancakeHouseMenu.createIterator();
    Iterator dinerIterator = dinerMenu.createIterator();
    Iterator cafeIterator = cafeMenu.createIterator();

    System.out.println("MENU\n---\nBREAKFAST");
    printMenu(pancakeIterator);
    System.out.println("\nLUNCH");
    printMenu(dinerIterator);
    System.out.println("\nDINNER");
    printMenu(cafeIterator);
}

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() {
    System.out.println("\nVEGETARIAN MENU\n-----");
    printVegetarianMenu(pancakeHouseMenu.createIterator());
    printVegetarianMenu(dinerMenu.createIterator());
    printVegetarianMenu(cafeMenu.createIterator());
}

public boolean isItemVegetarian(String name) {
    Iterator pancakeIterator =
pancakeHouseMenu.createIterator();
    if (isVegetarian(name, pancakeIterator)) {
        return true;
    }
    Iterator dinerIterator = dinerMenu.createIterator();
    if (isVegetarian(name, dinerIterator)) {
        return true;
    }
    Iterator cafeIterator = cafeMenu.createIterator();
    if (isVegetarian(name, cafeIterator)) {
        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.print(menuItem.getPrice() + " --
");

        System.out.println(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

```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32\prAAC\Slip_24_Iterator\Iterator>java MenuTestDrive.java
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, and blueberry syrup
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 -- A medly of steamed vegetables over brown rice
Pasta, 3.89 -- Spaghetti with Marinara Sauce, and a slice of sourdough bread

DINNER
Soup of the day, 3.69 -- A cup of the soup of the day, with a side salad
Burrito, 4.29 -- A large burrito, with whole pinto beans, salsa, guacamole
Veggie Burger and Air Fries, 3.99 -- Veggie burger on a whole wheat bun, lettuce, tomato, and fries

VEGETARIAN MENU
-----
K&B's Pancake Breakfast, 2.99 -- Pancakes with scrambled eggs, and toast
Blueberry Pancakes, 3.49 -- Pancakes made with fresh blueberries, and blueberry syrup
Waffles, 3.59 -- Waffles, with your choice of blueberries or strawberries
Vegetarian BLT, 2.99 -- (Fakin') Bacon with lettuce & tomato on whole wheat
Steamed Veggies and Brown Rice, 3.99 -- A medly of steamed vegetables over brown rice
Pasta, 3.89 -- Spaghetti with Marinara Sauce, and a slice of sourdough bread
Burrito, 4.29 -- A large burrito, with whole pinto beans, salsa, guacamole
Veggie Burger and Air Fries, 3.99 -- Veggie burger on a whole wheat bun, lettuce, tomato, and fries

Customer asks, is the Hotdog vegetarian?
Waitress says: No

Customer asks, are the Waffles vegetarian?
Waitress says: Yes

C:\Windows\System32\prAAC\Slip_24_Iterator\Iterator>

```



Type here to search



30°C Sunny 04:28 PM  
27-12-2022

## ASSIGNMENT 9

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.

### Gumballmachinetestdrive.java

```
import gumballstate.*;
public class GumballMachineTestDrive {
    public static void main(String[] args) {
        GumballMachine gumballMachine = new GumballMachine(5);
        System.out.println(gumballMachine);
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        System.out.println(gumballMachine);
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        System.out.println(gumballMachine);
    }
}
```

### GumballMachine.java

```
package gumballstate;
public class GumballMachine {
    State soldOutState;
    State noQuarterState;
    State hasQuarterState;
    State soldState;
    State state = soldOutState;
    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;
        }
    }
    public void insertQuarter() {
        state.insertQuarter();
    }
}
```

```

    public void ejectQuarter() {
        state.ejectQuarter();
    }
    public void turnCrank() {
        state.turnCrank();
        state.dispense();
    }
    void setState(State state) {
        this.state = state;
    }

    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;
        state = noQuarterState;
    }

    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();
    }
}

```

### **Hasquarterstate.java**

```

package gumballstate;

import java.util.Random;

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 String toString() {
        return "waiting for turn of crank";
    }
}

```

### **Noquarterstate.java**

```

package gumballstate;

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 String toString() {
        return "waiting for quarter";
    }
}

```

### **Soldoutstate.java**

```

package gumballstate;

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 String toString() {
        return "sold out";
    }
}

```

### **Soldstate.java**

```

package gumballstate;
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 String toString() {
            return "dispensing a gumball";
        }
    }
}

```

## State.java

```

package gumballstate;
public interface State {
    public void insertQuarter();
    public void ejectQuarter();
    public void turnCrank();
    public void dispense();
}

```

## OUTPUT

```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32\praac\Slip_8,23,29_State(GumBall)\Slip8_23_29_State>java GumballMachineTestDrive.java

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2804
Inventory: 5 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 #2804
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...
You inserted a quarter
You turned...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2804
Inventory: 2 gumballs
Machine is waiting for quarter

C:\Windows\System32\praac\Slip_8,23,29_State(GumBall)\Slip8_23_29_State>

```

## ASSIGNMENT 10

**10)Write a java program to implement adapter pattern to design heart model to beat model.**

### DJTestDrive.java

```
import Hearttobeat.*;
public class DJTestDrive {
    public static void main (String[] args) {
        BeatModelInterface model = new BeatModel();
        ControllerInterface controller = new
BeatController(model);
    }
}
```

### HeartTestDrive.java

```
import Hearttobeat.*;
public class HeartTestDrive {
    public static void main (String[] args) {
        HeartModel heartModel = new HeartModel();
        ControllerInterface model = new HeartController(heartModel);
    }
}
```

### BeatBar.java

```
package Hearttobeat;
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 Hearttobeat;
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 Hearttobeat;
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;
    }
}

```

### **BeatModelInterface.java**

```

package Hearttobeat;
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 Hearttobeat;
public interface BeatObserver {
    void updateBeat();
}

```

## **BPMObserver.java**

```
package Hearttobeat;
public interface BPMObserver {
    void updateBPM();
}
```

## **ControllerInterface.java**

```
package Hearttobeat;
public interface ControllerInterface {
    void start();
    void stop();
    void increaseBPM();
    void decreaseBPM();
    void setBPM(int bpm);
}
```

## **DJView.java**

```
package Hearttobeat;
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);
        setBPMBButton = new JButton("Set");
        setBPMBButton.setSize(new Dimension(10, 40));
        increaseBPMBButton = new JButton(">>");
        decreaseBPMBButton = new JButton("<<");
        setBPMBButton.addActionListener(this);
        increaseBPMBButton.addActionListener(this);
        decreaseBPMBButton.addActionListener(this);

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

```

```

        buttonPanel.add(decreaseBPMBButton);
        buttonPanel.add(increaseBPMBButton);

        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(setBPMBButton);
        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(setBPMBButton);
        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() == setBPMBButton) {
            int bpm =
Integer.parseInt(bpmTextField.getText());
            controller.setBPM(bpm);
        } else if (event.getSource() == increaseBPMBButton) {
            controller.increaseBPM();
        } else if (event.getSource() == decreaseBPMBButton) {
            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);
        }
    }
}

```

## HeartAdapter.java

```

package Hearttobeat;

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 Hearttobeat;

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 Hearttobeat;

```

```

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();
    }
}
}

```

## HeartModelInterface.java

```

package Hearttobeat;
public interface HeartModelInterface {
    int getHeartRate();
    void registerObserver(BeatObserver o);
    void removeObserver(BeatObserver o);
    void registerObserver(BPMObserver o);
    void removeObserver(BPMObserver o);
}

```

## OUTPUT

