CERTIFICATE

This is to certify that student Mr. Pranav Kiran Bhavsar of M.Sc. (CS) Semester III having Seat No.67 at Suryadatta College of Management Information Research & Technology (SCMIRT), Pune, has successfully completed the assigned practical in Software Architecture and Design patterns prescribed by the University of Pune During the academic year 2021-2022.

Internal Examiner **External**

Principal

Place: Pune
Date: 25/02/2022

Practical 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()

DisplayElement.java:

```
public interface DisplayElement
{
        public void display();
}
```

Observer.java:

```
public interface Observer
{
        public void update(float temp, float humidity, float pressure);
}
```

Subject.java:

```
public interface Subject
{
        public void registerObserver(Observer o);
public void removeObserver(Observer o);     public void
notifyObservers();
}
```

WeatherData.java:

```
import java.util.*; public class WeatherData
implements Subject
{
        private ArrayList<Observer> observers;
        private float temperature;
private float humidity;
private float pressure;          public
WeatherData()
        {
                observers = new ArrayList<>();
        }
        public void registerObserver(Observer o)
```

```
	{
		observers.add(o);
	}
	public void removeObserver(Observer o)
	{
	int i = observers.indexOf(o); if (i >=
0)
		{
			observers.remove(i);
		}
	}
	public void notifyObservers()
	{
		for (int i = 0; i < observers.size(); i++)
		{
	Observer observer = (Observer)observers.get(i);	observer.update(temperature,
humidity, pressure);
		}
	}
	public void measurementsChanged()
	{
		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;
	}
```

```
}
```

ForecastDisplay.java:

```
class ForecastDisplay implements Observer, DisplayElement
{
        private float currentPressure =
29.92f;         private float lastPressure;
private WeatherData weatherData;

        public ForecastDisplay(WeatherData weatherData)
        {
        this.weatherData = weatherData;     weatherData.registerObserver(this);
        }
        public void update(float temp, float humidity, float pressure)
        {
                lastPressure = currentPressure;
                currentPressure = pressure;
                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");
                }
        }
}
```

StatisticsDisplay.java

```
class StatisticsDisplay implements Observer, DisplayElement
{
```

```
	private float maxTemp = 0.0f;	private float
minTemp = 200;	private float tempSum= 0.0f; private
int numReadings;	private WeatherData weatherData;
public StatisticsDisplay(WeatherData weatherData)
	{
		this.weatherData = weatherData;
weatherData.registerObserver(this);
	}
	public void update(float temp, float humidity, float pressure)
	{
	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);
	}
}
```

CurrentConditionsDisplay.java:

```
class CurrentConditionsDisplay implements Observer, DisplayElement
{
	private float temperature;
private float humidity;	private
Subject weatherData;

	public CurrentConditionsDisplay(Subject weatherData)
	{
	this.weatherData = weatherData;
weatherData.registerObserver(this);
```

```
	}

	public void update(float temperature, float humidity, float pressure)
	{
	this.temperature = temperature;	this.humidity =
humidity;
		display();
	}

	public void display()
	{
 System.out.println("Current conditions: " + temperature+ "F degrees and " + humidity + "%
humidity");
	}
}
```

WeatherStation.java:

```
class WeatherStation
{
	public static void main(String[] args)
	{
		WeatherData weatherData = new WeatherData();

		CurrentConditionsDisplay currentDisplay=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:
D:\MScSem3Practicals\SADP Pract Assignment\Pract1>javac WeatherData.java

D:\MScSem3Practicals\SADP Pract Assignment\Pract1>java WeatherStation
Current conditions: 80.0F degrees and 65.0% humidity
Avg/Max/Min temperature = 80.0/80.0/80.0 Forecast:
Improving weather on the way!
```

```
Current conditions: 82.0F degrees and 70.0% humidity
Avg/Max/Min temperature = 81.0/82.0/80.0
Forecast: Watch out for cooler, rainy weather
Current conditions: 78.0F degrees and 90.0% humidity
Avg/Max/Min temperature = 80.0/82.0/78.0
Forecast: More of the same

*/
```

Practical 2:

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

LowerCaseInputStream.java

```
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;
	}
}
```

InputTest.java:

```
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();
                }
        }
}

/*
Input: test.txt:
Hello
How are YOU!!?

Output:
D:\MScSem3Practicals\SADP Pract Assignment\Pract2>java InputTest
hello how are you!!?
D:\MScSem3Practicals\SADP Pract Assignment\Pract2>
*/
```

Practical 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.

PizzaFactoryDemo.java:

```
import java.util.ArrayList; abstract
class Pizza
{
	String  name;
	String dough;
	String sauce;
	ArrayList toppings = new ArrayList();
	void prepare()
	{
		System.out.println("Preparing " + name);
		System.out.println("Tossing dough...");
		System.out.println("Adding sauce...");
		System.out.println("Adding toppings: ");
		for (int i = 0; i < toppings.size(); i++)
		{
			System.out.println(" " + toppings.get(i));
		}
	}
	void bake()
	{
		System.out.println("Bake for 25 minutes at 350");
	}
	void cut()
	{
		System.out.println("Cutting the pizza into diagonal slices");
	}
	void box()
	{
		System.out.println("Place pizza in official PizzaStore box");
	}
	public String getName()
	{
		return name;
	}
	public String toString()
	{
		StringBuffer display = new
		StringBuffer(); display.append("---- " +
		name + "----\n"); display.append(dough
		+ "\n"); display.append(sauce + "\n"); for
		(int i = 0; i < toppings.size(); i++)
		{
			display.append((String )toppings.get(i) + "\n");
		}
```

```
                return display.toString();
        }
}

class ChicagoStyleCheesePizza extends Pizza
{
        public ChicagoStyleCheesePizza()
        {
                name = "Chicago Style Deep Dish Cheese
                Pizza"; dough = "Extra Thick Crust Dough"; sauce
                = "Plum Tomato Sauce";
                toppings.add("Shredded Mozzarella Cheese");
        }
        void cut()
        {
                System.out.println("Cutting the pizza into square slices");
        }
}

class ChicagoStyleClamPizza extends Pizza
{
        public ChicagoStyleClamPizza()
        {
                name = "Chicago Style Clam Pizza"; dough = "Extra
                Thick Crust Dough"; sauce = "Plum Tomato Sauce";
                toppings.add("Shredded Mozzarella Cheese");
                toppings.add("Frozen Clams from Chesapeake
                Bay");
        }
        void cut()
        {
                System.out.println("Cutting the pizza into square slices");
        }
}

class ChicagoStylePepperoniPizza extends Pizza
{
        public ChicagoStylePepperoniPizza()
        {
                name = "Chicago Style Pepperoni Pizza";
                dough = "Extra Thick Crust Dough"; sauce =
                "Plum Tomato Sauce";
                toppings.add("Shredded Mozzarella
                Cheese"); toppings.add("Black Olives");
```

```
                toppings.add("Spinach");
                toppings.add("Eggplant");
                toppings.add("Sliced Pepperoni");
        }
        void cut()
        {
                System.out.println("Cutting the pizza into square slices");
        }
}

class ChicagoStyleVeggiePizza extends Pizza
{
        public ChicagoStyleVeggiePizza()
        {
                name = "Chicago Deep Dish Veggie Pizza";
                dough = "Extra Thick Crust Dough"; sauce =
                "Plum Tomato Sauce";
                toppings.add("Shredded Mozzarella
                Cheese"); toppings.add("Black Olives");
                toppings.add("Spinach");
                toppings.add("Eggplant");
        }
        void cut()
        {
                System.out.println("Cutting the pizza into square slices");
        }
}

class NYStyleCheesePizza extends Pizza
{
        public NYStyleCheesePizza()
        {
                name = "NY Style Sauce and Cheese
                Pizza"; dough = "Thin Crust Dough"; sauce =
                "Marinara Sauce"; toppings.add("Grated
                Reggiano Cheese");
        }
}

class NYStyleClamPizza extends Pizza
{
        public NYStyleClamPizza()
        {
```

```
                name = "NY Style Clam Pizza"; dough = "Thin Crust
                Dough"; sauce = "Marinara Sauce";
                toppings.add("Grated Reggiano Cheese");
                toppings.add("Fresh Clams from Long Island
                Sound");
        }
}

class NYStylePepperoniPizza extends Pizza
{
        public NYStylePepperoniPizza()
        {
                name = "NY Style Pepperoni Pizza";
                dough = "Thin Crust Dough"; sauce =
                "Marinara Sauce"; toppings.add("Grated
                Reggiano Cheese"); toppings.add("Sliced
                Pepperoni"); toppings.add("Garlic");
                toppings.add("Onion");
                toppings.add("Mushrooms");
                toppings.add("Red Pepper");
        }
}

class NYStyleVeggiePizza extends Pizza
{
        public NYStyleVeggiePizza()
        {
                name = "NY Style Veggie Pizza"; dough =
                "Thin Crust Dough"; sauce = "Marinara
                Sauce"; toppings.add("Grated Reggiano
                Cheese"); toppings.add("Garlic");
                toppings.add("Onion");
                toppings.add("Mushrooms");
                toppings.add("Red Pepper");
        }
}


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;
        }
}

class ChicagoPizzaStore extends PizzaStore
{
        Pizza createPizza(String item)
        {
                if (item.equals("cheese"))
                {
                        return new ChicagoStyleCheesePizza();
                }
                else if (item.equals("veggie"))
                {
                        return new ChicagoStyleVeggiePizza();
                }
                else if (item.equals("clam"))
                {
                        return new ChicagoStyleClamPizza();
                }
                else if (item.equals("pepperoni"))
                {
                        return new ChicagoStylePepperoniPizza();
                }
                else return null;
        }
}

class NYPizzaStore extends PizzaStore
{
        Pizza createPizza(String item)
        {
                if (item.equals("cheese"))
                {
                        return new NYStyleCheesePizza();
                }
                else if (item.equals("veggie"))
                {
                        return new NYStyleVeggiePizza();
                }
                else if (item.equals("clam"))
                {
                        return new NYStyleClamPizza();
```

```
                }
                else if (item.equals("pepperoni"))
                {
                        return new NYStylePepperoniPizza();
                }
                else return null;
        }
}

public class PizzaFactoryDemo
{
        public static void main(String[] args)
        {
                PizzaStore nyStore = new NYPizzaStore();
                PizzaStore chicagoStore = new ChicagoPizzaStore();
                Pizza pizza = nyStore.orderPizza("cheese");
                System.out.println("Customer1 ordered a " + pizza.getName() +"\n");
                pizza = chicagoStore.orderPizza("cheese");
                System.out.println("Customer2 ordered a " + pizza.getName() +"\n");
                pizza = nyStore.orderPizza("clam");
                System.out.println("Customer1 ordered a " + pizza.getName() +"\n");
                pizza = chicagoStore.orderPizza("clam");
                System.out.println("Customer2 ordered a " + pizza.getName() +"\n");
                pizza = nyStore.orderPizza("pepperoni");
                System.out.println("Customer1 ordered a " + pizza.getName() +"\n");
                pizza = chicagoStore.orderPizza("pepperoni");
                System.out.println("Customer2 ordered a " + pizza.getName() +"\n");
                pizza = nyStore.orderPizza("veggie");
                System.out.println("Customer1 ordered a " + pizza.getName() +"\n");
                pizza = chicagoStore.orderPizza("veggie");
                System.out.println("Customer2 ordered a " + pizza.getName() +"\n");
        }
}

/*
Output:

Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.

D:\MScSem3Practicals\SADP Pract Assignment\Pract3>java PizzaFactoryDemo
--- Making a NY Style Sauce and Cheese Pizza---
Preparing NY Style Sauce and Cheese Pizza
Tossing dough... Adding sauce...
```

Adding toppings:
 Grated Reggiano Cheese
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Customer1 ordered a NY Style Sauce and Cheese Pizza

--- Making a Chicago Style Deep Dish Cheese Pizza---
Preparing Chicago Style Deep Dish Cheese Pizza
Tossing dough... Adding sauce...
Adding toppings:
 Shredded Mozzarella Cheese
Bake for 25 minutes at 350
Cutting the pizza into square slices
Place pizza in official PizzaStore box
Customer2 ordered a Chicago Style Deep Dish Cheese Pizza

--- Making a NY Style Clam Pizza---
Preparing NY Style Clam Pizza
Tossing dough... Adding sauce...
Adding toppings:
 Grated Reggiano Cheese
 Fresh Clams from Long Island Sound
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Customer1 ordered a NY Style Clam Pizza

--- Making a Chicago Style Clam Pizza---
Preparing Chicago Style Clam Pizza
Tossing dough... Adding sauce...
Adding toppings:
 Shredded Mozzarella Cheese
Frozen Clams from Chesapeake Bay
Bake for 25 minutes at 350
Cutting the pizza into square slices
Place pizza in official PizzaStore box
Customer2 ordered a Chicago Style Clam Pizza

--- Making a NY Style Pepperoni Pizza---
Preparing NY Style Pepperoni Pizza
Tossing dough... Adding sauce...
Adding toppings:
 Grated Reggiano Cheese

Sliced Pepperoni
Garlic
Onion
Mushrooms
Red Pepper
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Customer1 ordered a NY Style Pepperoni Pizza

--- Making a Chicago Style Pepperoni Pizza---
Preparing Chicago Style Pepperoni Pizza
Tossing dough... Adding sauce...
Adding toppings:
Shredded Mozzarella Cheese
Black Olives
Spinach
Eggplant
Sliced Pepperoni
Bake for 25 minutes at 350
Cutting the pizza into square slices
Place pizza in official PizzaStore box
Customer2 ordered a Chicago Style Pepperoni Pizza --- Making a NY Style Veggie Pizza---
Preparing NY Style Veggie Pizza Tossing dough... Adding sauce...
Adding toppings:
Grated Reggiano Cheese
Garlic
Onion
Mushrooms
Red Pepper
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Customer1 ordered a NY Style Veggie Pizza

--- Making a Chicago Deep Dish Veggie Pizza---
Preparing Chicago Deep Dish Veggie Pizza
Tossing dough... Adding sauce...
Adding toppings:
Shredded Mozzarella Cheese
Black Olives
Spinach
Eggplant
Bake for 25 minutes at 350

Cutting the pizza into square slices
Place pizza in official PizzaStore box
Customer2 ordered a Chicago Deep Dish Veggie Pizza

*/

Practical 4:

Write a Java Program to implement Singleton pattern for multithreading.

ChocolateControllerDemo.java:

```
class ChocolateBoiler
{
        private boolean empty; private boolean
        boiled; private static ChocolateBoiler
        uniqueInstance;

        private ChocolateBoiler()
        {
                empty = true;
                boiled = false;
        }

        public static synchronized ChocolateBoiler getInstance()
        {
                if (uniqueInstance == null)
                {
                        System.out.println("Creating unique instance of Chocolate Boiler");
                        uniqueInstance = new ChocolateBoiler();
                }
                System.out.println("Returning instance of Chocolate Boiler");
                return uniqueInstance;
        }
```

```
	public void fill(String threadName)
	{
		if (isEmpty())
		{
			empty = false;
			boiled = false;
			System.out.println(threadName+ ": The boiler is filled with a
milk/chocolate mixture");
		}
	}

	public void drain(String threadName)
	{
		if (!isEmpty() && isBoiled())
		{
			System.out.println(threadName +": Drain the boiled milk and chocolate");
			empty = true;
		}
	}

	public void boil(String threadName)
	{
		if (!isEmpty() && !isBoiled())
		{
			System.out.println(threadName +": Bring the contents to a boil");
			boiled = true;
		}
	}

	public boolean isEmpty()
	{
		return empty;
	}

	public boolean isBoiled()
	{
		return boiled;
	}
}

class ChocolateController implements Runnable
{
```

```
        private Thread t; private
        String threadName; private
        ChocolateBoiler boiler;

        ChocolateController( String name)
        {
                threadName = name;
                System.out.println("Creating " + threadName);
                boiler = ChocolateBoiler.getInstance();
        }

        public void run()
        {
                System.out.println("Running " + threadName +" with boiler instance "+boiler ); try
                {
                        boiler.fill(threadName);
                        boiler.boil(threadName);
                        boiler.drain(threadName);
                        // Let the thread sleep for a while.
                        Thread.sleep(50);
                }
                catch (InterruptedException e)
                {
                        System.out.println("Thread " + threadName + " interrupted.");
                }
                System.out.println("Thread " + threadName + " exiting.");
        }

        public void start ()
        {
                System.out.println("Starting " + threadName );
                if (t == null)
                {
                        t = new Thread (this, threadName);
                        t.start ();
                }
        }
}
```

ChocolateControllerDemo.java:

```
public class ChocolateControllerDemo
{
```

```
	public static void main(String args[])
	{
		ChocolateController t1 = new ChocolateController( "Thread-1");
		t1.start();
		ChocolateController t2 = new ChocolateController( "Thread-2");
		t2.start();
	}
}

/*
Output:
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.

D:\MScSem3Practicals\SADP Pract Assignment\Pract4>java ChocolateControllerDemo
Creating Thread-1
Creating unique instance of Chocolate Boiler
Returning instance of Chocolate Boiler
Starting Thread-1
Creating Thread-2
Returning instance of Chocolate Boiler
Running Thread-1 with boiler instance ChocolateBoiler@4c31f4d0
Starting Thread-2
Thread-1: The boiler is filled with a milk/chocolate mixture
Thread-1: Bring the contents to a boil
Thread-1: Drain the boiled milk and chocolate
Running Thread-2 with boiler instance ChocolateBoiler@4c31f4d0
Thread-2: The boiler is filled with a milk/chocolate mixture
Thread-2: Bring the contents to a boil Thread-2:
Drain the boiled milk and chocolate Thread
Thread-2 exiting.
Thread Thread-1 exiting.

*/
```

Practical 5:

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

RemoteControlTest.java

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

class Light
{
	public void on()
	{
		System.out.println("Light is on");
	}
	public void off()
	{
		System.out.println("Light is off");
	}
}
class LightOnCommand implements Command
{
	Light light;

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

```
        {
        light.on();
        }
}
class LightOffCommand implements Command
{
        Light light; public
        LightOffCommand(Light light)
        { this.light = light;
        }
        public void execute()
        { light.off();
        }
}

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)
        {
        System.out.println("Stereo volume set"+ " to " + volume);
        }
}

class StereoOffCommand implements Command
```

```
{
        Stereo stereo; public
        StereoOffCommand(Stereo stereo)
        {
                this.stereo = stereo;
        }
        public void execute()
        {
        stereo.off();
        }
}

class StereoOnWithCDCommand implements Command
{
        Stereo stereo; public
        StereoOnWithCDCommand(Stereo stereo)
        {
                this.stereo = stereo;
        }
        public void execute()
        {
                stereo.on();
                stereo.setCD();
                stereo.setVolume(11);
        }
}

class SimpleRemoteControl
{
        Command slot;

        public SimpleRemoteControl()
        {
        }

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

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

```
}

// Driver class class
RemoteControlTest
{
	public static void main(String[] args)
	{
		SimpleRemoteControl remote =new SimpleRemoteControl();
		Light light = new Light();
		Stereo stereo = new Stereo();

		remote.setCommand(new LightOnCommand(light));
		remote.buttonWasPressed(); remote.setCommand(new
		StereoOnWithCDCommand(stereo)); remote.buttonWasPressed();
		remote.setCommand(new StereoOffCommand(stereo));
		remote.buttonWasPressed();
	}
}

/*
Output:
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.

D:\MScSem3Practicals\SADP Pract Assignment\Pract5>java RemoteControlTest
Light is on
Stereo is on
Stereo is set for CD input
Stereo volume set to 11
Stereo is off
*/
```

Practical 6:

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

RemoteLoader.java:

```
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:

```
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.

D:\MScSem3Practicals\SADP Pract Assignment\Pract6>java RemoteLoader
Living Room ceiling fan is on medium
Living Room ceiling fan is off

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

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

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

Living Room ceiling fan is on medium

*/
```

RemoteControlWithUndo.java:

```
import java.util.*; public class
RemoteControlWithUndo {
        Command[] onCommands;
        Command[] offCommands;
        Command undoCommand;
```

```
	public RemoteControlWithUndo() {
		onCommands = new Command[7];
		offCommands = new Command[7];

		Command noCommand = new 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();
	}
}
```

CeilingFanOffCommand.java:

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

CeilingFanMediumCommand.java:

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

CeilingFanLowCommand.java:

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

CeilingFanHighCommand.java:

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

CeilingFan.java

```
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;
        }
}
```

Command.java:

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

NoCommand.java:

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

Practical 7:

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

EnumerationIterator.java

```
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();
        }
}
```

EnumerationIteratorTestDrive.java:

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

```
/*
Output:
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.

D:\MScSem3Practicals\SADP Pract Assignment\Pract7>java EnumerationIteratorTestDrive hi
hello how are you ?
hi
hello
how
are
you ?
*/
```

Practical 8:

Write a Java Program to implement Iterator Pattern for Designing Menu like Breakfast, Lunch or Dinner Menu. Book 6 (page no 326)

MenuTestDrive.java:

```
import java.util.*; public
class MenuTestDrive
{
	public static void main(String args[])
	{
		PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();
		DinerMenu dinerMenu = new DinerMenu();
		Waitress waitress = new Waitress(pancakeHouseMenu,
		dinerMenu); waitress.printMenu(); 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");
		}

	}
}

/*
Output:
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.
```

```
D:\MScSem3Practicals\SADP Pract Assignment\Pract8\>java MenuTestDrive
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 -- Steamed vegetables over brown rice Pasta,
3.89 -- Spaghetti with Marinara Sauce, and a slice of sourdough bread

VEGETARIAN MENU
----
BREAKFAST
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

LUNCH
Vegetarian BLT          2.99
      (Fakin') Bacon with lettuce & tomato on whole wheat
Steamed Veggies and Brown Rice          3.99
      Steamed vegetables over brown rice
Pasta           3.89
      Spaghetti with Marinara Sauce, and a slice of sourdough bread

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

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

Menu.java:

```
import java.util.Iterator; public
interface Menu
{
        public Iterator createIterator();
}
```

MenuItem.java:

```
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;
        }
}
```

DinnerMenu.java:

```
import java.util.Iterator; public class
DinerMenu implements Menu
{
	static final int MAX_ITEMS = 6;
	int numberOfItems = 0;
	MenuItem[] menuItems;

	public DinerMenu()
	{
		menuItems = new MenuItem[MAX_ITEMS];

		addItem("Vegetarian BLT","(Fakin') Bacon with lettuce & tomato on whole
wheat", true, 2.99);
		addItem("BLT","Bacon with lettuce & tomato on whole wheat", false, 2.99);
		addItem("Soup of the day","Soup of the day, with a side of potato salad", false,
3.29); addItem("Hotdog","A hot dog, with saurkraut, relish, onions, topped with
cheese",false, 3.05); addItem("Steamed Veggies and Brown Rice","Steamed vegetables over
		brown
rice", true, 3.99); addItem("Pasta","Spaghetti with Marinara Sauce, and a slice of
		sourdough
bread",true, 3.89);
	}

	public void addItem(String name, String description,boolean vegetarian, double price)
	{
		MenuItem menuItem = new MenuItem(name, description, vegetarian, price);
		if (numberOfItems >= MAX_ITEMS)
		{
			System.err.println("Sorry, menu is full! Can't add item to menu");
		}
		else
		{
			menuItems[numberOfItems] = menuItem;
			numberOfItems = numberOfItems + 1;
		}
	}

	public MenuItem[] getMenuItems()
	{
		return menuItems;
	}
```

```
        public Iterator createIterator()
        {
                return new DinerMenuIterator(menuItems);
        }
}
```

DinerMenuIterator.java:

```
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; }
        }
}
```

PancakeHouseMenu.java:

```
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();
        }
}
```

Waitress.java:

```
import java.util.Iterator; public
class Waitress
{
        Menu pancakeHouseMenu;
        Menu dinerMenu;

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

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

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

        private void printMenu(Iterator iterator) {
                while (iterator.hasNext()) {
                        MenuItem menuItem = (MenuItem)iterator.next();
                        System.out.print(menuItem.getName() + ", ");
                        System.out.print(menuItem.getPrice() + " -- ");
                        System.out.println(menuItem.getDescription());
                }
        }

        public void printVegetarianMenu() {
                System.out.println("\nVEGETARIAN MENU\n--- \nBREAKFAST");
                printVegetarianMenu(pancakeHouseMenu.createIterator());
                System.out.println("\nLUNCH");
                printVegetarianMenu(dinerMenu.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;
            }
            return false;
        }


        private void printVegetarianMenu(Iterator iterator) {
            while (iterator.hasNext()) {
                MenuItem menuItem = (MenuItem)iterator.next();
                if (menuItem.isVegetarian()) {
                    System.out.print(menuItem.getName());
                    System.out.println("\t\t" + menuItem.getPrice());
                    System.out.println("\t" + menuItem.getDescription());
                }
            }
        }

        private boolean isVegetarian(String name, Iterator iterator) {
            while (iterator.hasNext()) {
                MenuItem menuItem =
                (MenuItem)iterator.next(); if
                (menuItem.getName().equals(name)) { if
                (menuItem.isVegetarian()) { return true;
                    }
                }
            }
            return false;
        }
}
```

Practical 9

Write a Java Program to implement State Pattern for Gumball Machine. Create instance ariable 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:

```
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.ejectQuarter();
		gumballMachine.turnCrank();

		System.out.println(gumballMachine);

		gumballMachine.insertQuarter();
		gumballMachine.turnCrank();
		gumballMachine.insertQuarter();
		gumballMachine.turnCrank();
		gumballMachine.ejectQuarter();

		System.out.println(gumballMachine);

		gumballMachine.insertQuarter();
		gumballMachine.insertQuarter();
		gumballMachine.turnCrank();
		gumballMachine.insertQuarter();
		gumballMachine.turnCrank();
		gumballMachine.insertQuarter();
		gumballMachine.turnCrank();

		System.out.println(gumballMachine);
	}
}

/*
Output:
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.
```

```
D:\MScSem3Practicals\SADP Pract Assignment\Pract9>java GumballMachineTestDrive

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
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 #2004
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
Quarter returned
You turned but there's no quarter

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
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
You haven't inserted a quarter

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 2 gumballs
Machine is waiting for quarter

You inserted a quarter You
can't insert another quarter You
turned...
A gumball comes rolling out the slot
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

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 0 gumballs
Machine is sold out

*/

GumballMachine.java:

```
public class GumballMachine
{
	final static int SOLD_OUT = 0;
	final static int NO_QUARTER = 1;
	final static int HAS_QUARTER =
	2; final static int SOLD = 3;

	int state = SOLD_OUT;
	int count = 0;

	public GumballMachine(int count)
	{
		this.count = count;
		if (count > 0)
		{
			state = NO_QUARTER;
		}
	}

	public void insertQuarter()
	{
		if (state == HAS_QUARTER)
		{
			System.out.println("You can't insert another quarter");
		}
		else if (state == NO_QUARTER)
		{
			state = HAS_QUARTER;
			System.out.println("You inserted a quarter");
		}
```

```
        else if (state == SOLD_OUT)
        {
                System.out.println("You can't insert a quarter, the machine is sold out");
        }
        else if (state == SOLD)
        {
System.out.println("Please wait, we're already giving you a gumball");
        }
}

public void ejectQuarter()
{
        if (state == HAS_QUARTER)
        {
                System.out.println("Quarter returned");
                state = NO_QUARTER;
        }
        else if (state == NO_QUARTER)
        {
                System.out.println("You haven't inserted a quarter");
        }
        else if (state == SOLD)
        {
                System.out.println("Sorry, you already turned the crank");
        }
        else if (state == SOLD_OUT)
        {
System.out.println("You can't eject, you haven't inserted a quarter yet");
        }
}
public void turnCrank()
{
        if (state == SOLD)
        {
                System.out.println("Turning twice doesn't get you another gumball!");
        }
        else if (state == NO_QUARTER)
        {
                System.out.println("You turned but there's no quarter");
        }
        else if (state == SOLD_OUT)
        {
                System.out.println("You turned, but there are no gumballs");
        }
```

```
        else if (state == HAS_QUARTER)
        {
                System.out.println("You
                turned..."); state = SOLD;
                dispense();
        }
}

public void dispense()
{
        if (state == SOLD)
        {
                System.out.println("A gumball comes rolling out the
                slot"); count = count - 1; if (count == 0)
                {
                        System.out.println("Oops, out of gumballs!");
                        state = SOLD_OUT;
                }
                else
                {
                        state = NO_QUARTER;
                }
        }
        else if (state == NO_QUARTER)
        {
                System.out.println("You need to pay first");
        }
        else if (state == SOLD_OUT)
        {
                System.out.println("No gumball dispensed");
        }
        else if (state == HAS_QUARTER)
        {
                System.out.println("No gumball dispensed");
        }
}

public void refill(int numGumBalls)
{
        this.count = numGumBalls;
        state = NO_QUARTER;
}

public String toString()
```

```
	{
		StringBuffer result = new StringBuffer(); result.append("\nMighty
		Gumball, Inc."); result.append("\nJava-enabled Standing Gumball
		Model #2004\n"); result.append("Inventory: " + count + " gumball");
		if (count != 1)
		{
			result.append("s");
		}
		result.append("\nMachine is ");
		if (state == SOLD_OUT)
		{
			result.append("sold out");
		}
		else if (state == NO_QUARTER)
		{
			result.append("waiting for quarter");
		}
		else if (state == HAS_QUARTER)
		{
			result.append("waiting for turn of crank");
		}
		else if (state == SOLD)
		{
			result.append("delivering a gumball");
		}
		result.append("\n");
		return result.toString();
	}
}
```

Practical 10

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

HeartModel.java:

```
package djview; 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();
		}
	}
}
```

BeatModel.java:

```
package djview; 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;
        }
}
```

BeatController.java:

```
package djview; 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);
	}
}
```

BeatBar.java:

```
package djview; 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) {};
                }
        }
}
```

HeartModelInterface.java:

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

BeatModelInterface.java:

```
package djview; 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);
}
```

ControllerInterface.java:

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

BPMObserver.java

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

BeatObserver.java:

```
package djview; public
interface BeatObserver
{
	void updateBeat();
}
```

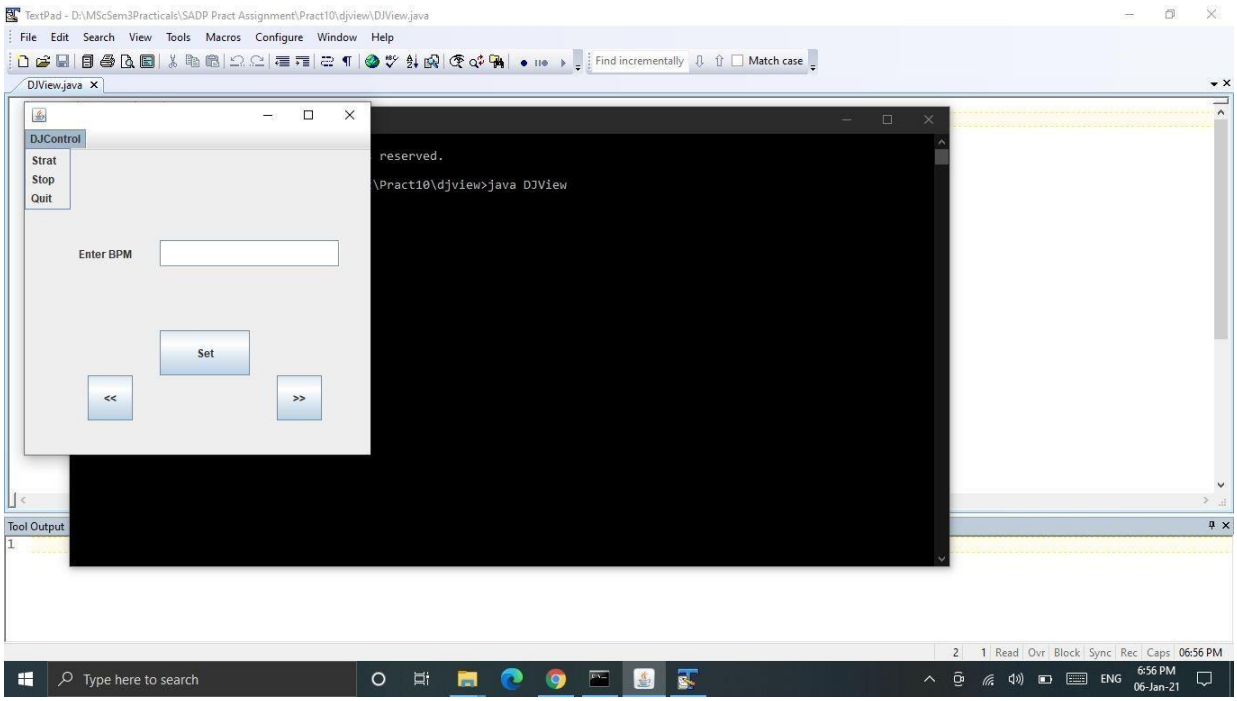
HeartTestDrive.java:

```
package djview;
public class HeartTestDrive
{
	public static void main (String[] args)
	{
		HeartModel heartModel = new HeartModel();
		ControllerInterface model = new HeartController(heartModel);
	}
}
```

DJTestDrive.java:

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

Output:


TextPad - D:\MScSem3Practicals\SADP Pract Assignment\Pract10\djview\DJView.java
File Edit Search View Tools Macros Configure Window Help
Find incrementally
Match case
DJView.java
DJControl
Strat
Stop
Quit
Enter BPM
Set
<<
>>
reserved.
\Pract10\djview>java DJView
Tool Output
1
Type here to search
ENG
6:56 PM
06-Jan-21