Software Architecture and Design Patterns

Class: SY.MSC.(Computer Science)

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

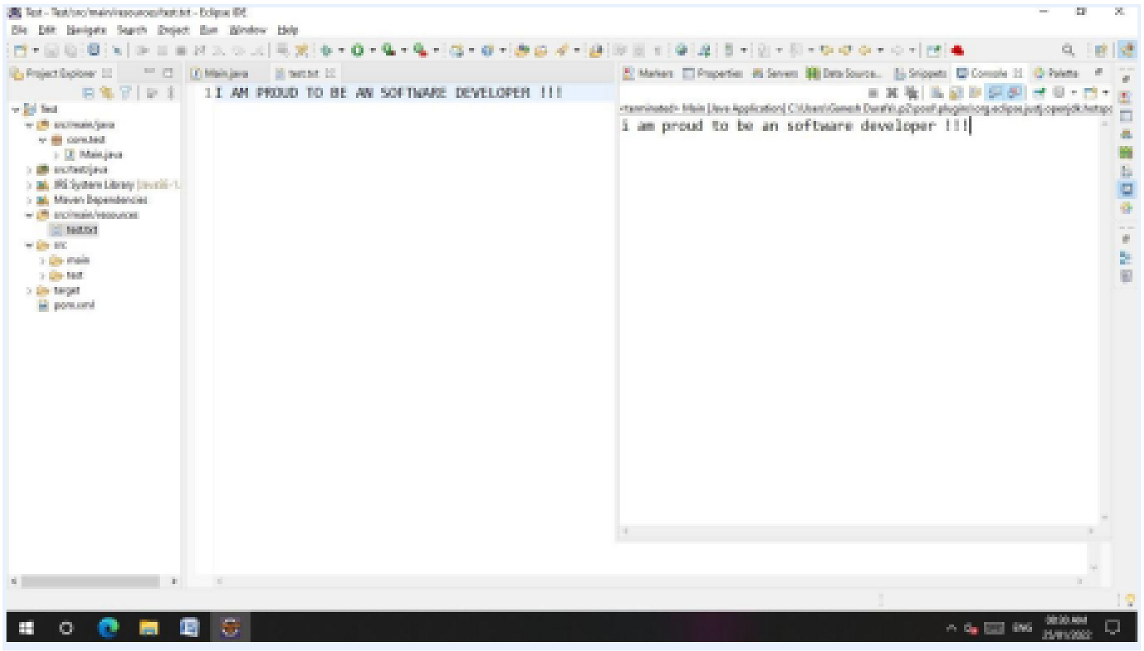
Main.java

```
package com.test;

import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.FilterInputStream;
import java.io.IOException;
import java.io.InputStream;

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) throwsIOException {
                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;
        }
}

public class Main {
        public static void main(String[] args) throws IOException {
                int c;
                try {
                        InputStream in = new LowerCaseInputStream(new BufferedInputStream(new
FileInputStream("src/main/resources/test.txt")));
                        while ((c = in.read()) >= 0) {
                                System.out.print((char) c);
                        }
                        in.close();

                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
  }
```

/src/main/resources/test.txt

```
I AM PROUD TO BE AN SOFTWARE DEVELOPER !!!
```

Output:



```
I AM PROUD TO BE AN SOFTWARE DEVELOPER !!!
```

```
i am proud to be an software developer !!!
```

2] Write a Java Program to implement Singleton pattern for multithreading.

VoiceRecognizer.java

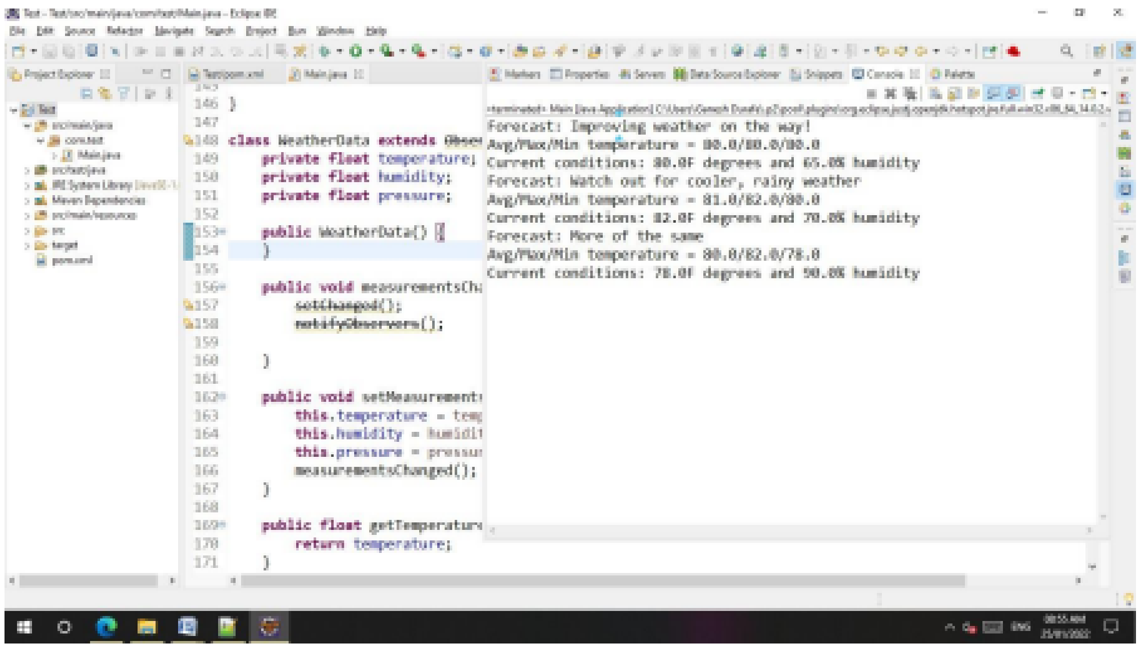
```
package com.sp.doublecheck.pattern;

public class VoiceRecognizer {
	private static VoiceRecognizer instance;

	private VoiceRecognizer() {
	}

	public static VoiceRecognizer getInstance() {
		if (instance == null) {
			System.out.println("lock");
			synchronized (VoiceRecognizer.class) {
				if (instance == null) {
					instance = new VoiceRecognizer();
				}
			}
			System.out.println("released lock");
		}
		return instance;
	}

	public void recognize() {
		System.out.println("double check voice recognizer : "
+ this.hashCode() + " is recognizing..");
	}
}
```

3] 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().

Main.java

```
package com.test;

import java.util.~~Observable~~;
import java.util.~~Observer~~;

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");

        }
}

interface DisplayElement {
        public void display();
}

 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");
                }
        }
}

class HeatIndexDisplay implements ~~Observer~~, DisplayElement {
        float heatIndex = 0.0f;

        public HeatIndexDisplay(~~Observable~~ observable) {
                observable.~~addObserver(this)~~;
        }

        public void update(~~Observable~~ observable, Object arg) {
                if (observable instanceof WeatherData) {
                        WeatherData weatherData = (WeatherData) observable;
                        float t = weatherData.getTemperature();
                        float rh = weatherData.getHumidity();
                        heatIndex = (float) (

                        (16.923 + (0.185212 * t)) + (5.37941 * rh) - (0.100254 * t * rh) +
(0.00941695 * (t * t))
                                        + (0.00728898 * (rh * rh)) + (0.000345372 * (t * t * rh)) -
(0.000814971 * (t * rh * rh)) +

                                        (0.0000102102 * (t * t * rh * rh)) - (0.000038646 * (t * t *
t)) + (0.0000291583 * (rh * rh * rh))
                                        + (0.00000142721 * (t * t * t * rh)) + (0.000000197483 *
(t * rh * rh * rh))
                                        - (0.0000000218429 * (t * t * t * rh * rh)) +
(0.000000000843296 * (t * t * rh * rh * rh))
                                        - (0.0000000000481975 * (t * t * t * rh * rh * rh)));
                        display();
                }

        }

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

        }

}

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);

        }

}

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;
        }

}

public class Main {

        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);

        }

}
```

Out Put:



```
146 }

class WeatherData extends Obser
    private float temperature;
    private float humidity;
    private float pressure;

    public WeatherData() {
    }

    public void measurementsCha
        setChanged();
        notifyObservers();

    }

    public void setMeasurement
        this.temperature = tem
        this.humidity = humidi
        this.pressure = pressu
        measurementsChanged();
    }

    public float getTemperatur
        return temperature;
    }
```

```
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
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
Forecast: More of the same
Avg/Max/Min temperature = 80.0/82.0/78.0
Current conditions: 78.0F degrees and 90.0% humidity
```

4] Write a Java Program to implement Factory method for Pizza Store with bake(), box(), prepareDough() . Use this to create variety of pizza's like ChickenPizza, VegPizza etc.

Pizza.java

```
package com.fm.beans;

public interface Pizza {
	void prepareDough();

	void bake();

	void box();
}
```

ChickenPizza.java

```
package com.fm.beans;

public class ChickenPizza implements Pizza {

	@Override
	public void prepareDough() {
		System.out.println("preparing hard and thick dough for
chicken pizza");
	}

	@Override
	public void bake() {
		System.out.println("baking chicken");
		System.out.println("applying chiken on the pizza base");
		System.out.println("baking to 900 degrees temparature");
	}

	@Override
	public void box() {
		System.out.println("cutting chicken pizza and placing in red
	box"); }

}
```

VegPizza.java

```
package com.fm.beans;
public class VegPizza implements Pizza {

	@Override
	public void prepareDough() {
		System.out.println("preparing soft and thin dough for veg
	pizza"); }
	@Override
	public void bake() {
		System.out.println("applying vegetables and baking");
	}
	@Override
	public void box() {
		System.out.println("cutting veg pizza and placed in green
	box"); }
}
```

PizzaFactory.java

```
package com.fm.pattern;

import com.fm.beans.Pizza;

public abstract class PizzaFactory {
	final public Pizza newPizza(String pizzaType) {
		Pizza pizza = null;

		pizza = createPizza(pizzaType);
```

```
            System.out.println("applying standard preparation process
in creating pizzas");
            pizza.prepareDough();
            pizza.bake();
            pizza.box();

            return pizza;
      }

      abstract protected Pizza createPizza(String pizzaType);
}
```

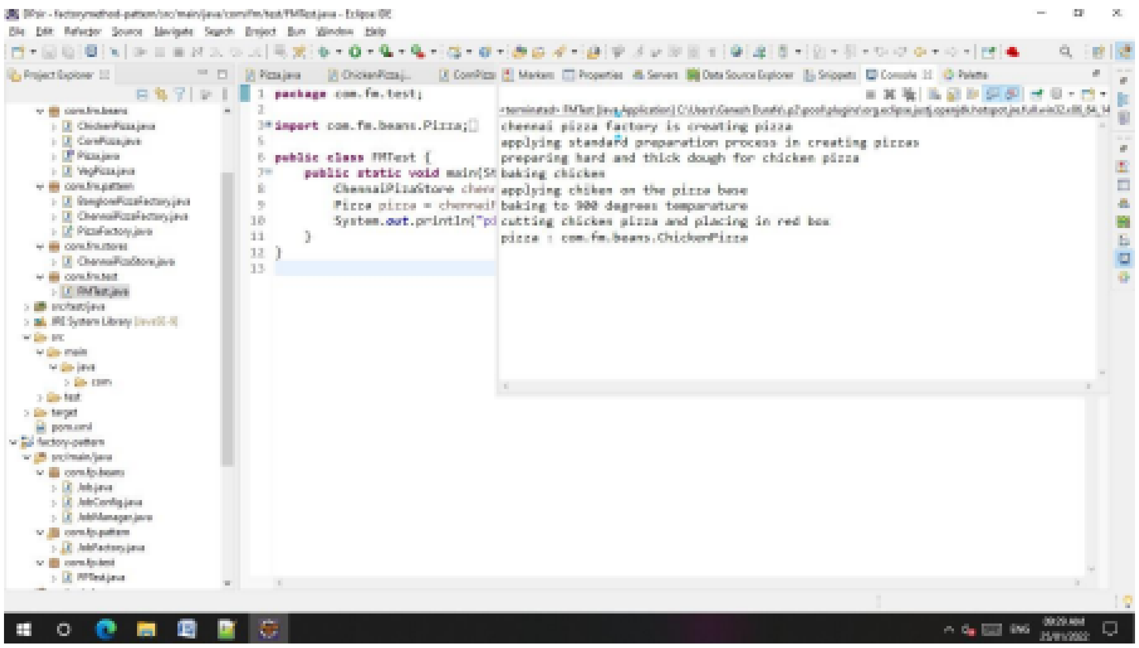
ChennaiPizzaFactory.java

```
package com.fm.pattern;

import com.fm.beans.ChickenPizza;
import com.fm.beans.Pizza;
import com.fm.beans.VegPizza;

public class ChennaiPizzaFactory extends PizzaFactory {
      @Override
      protected Pizza createPizza(String pizzaType) {
            Pizza pizza = null;

            System.out.println("chennai pizza factory is creating
            pizza"); if (pizzaType.equals("veg")) {
                  pizza = new VegPizza();
            } else if (pizzaType.equals("chicken")) {
                  pizza = new ChickenPizza();
            }

            return pizza;
      }

}
```

FMTest.java

```
package com.fm.test;

import com.fm.beans.Pizza;
import com.fm.stores.ChennaiPizaStore;

public class FMTest {
      public static void main(String[] args) {
            ChennaiPizaStore chennaiPizaStore = new ChennaiPizaStore();
            Pizza pizza = chennaiPizaStore.orderPizza("chicken");
            System.out.println("pizza : " + pizza.getClass().getName());
      }
}
```

Out Put:



```
package com.fm.test;

import com.fm.beans.Pizza;

public class FMTest {
    public static void main(St
        ChennaiPizzaStore chen
        Pizza pizza = chennai
        System.out.println("pi
    }
}
```

```
<terminated> FMTest [Java Application] C:\Users\Ganesh Bundhi\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_1
chennai pizza factory is creating pizza
applying standard preparation process in creating pizzas
preparing hard and thick dough for chicken pizza
baking chicken
applying chiken on the pizza base
baking to 900 degrees temparature
cutting chicken pizza and placing in red box
pizza : com.fm.beans.ChickenPizza
```

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

```java
package com.test;

import java.util.Arrays;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.Vector;

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();

        }

}

public class Main {
        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());
                }

        }

}
```

6] Write a Java Program to implement command pattern to test Remote

Control. package com.test;

```
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 l1;

        public LightOnCommand(Light a) {
                this.l1 = a;
        }

        public void execute() {
                l1.on();
        }
}

class LightOffCommand implements Command {
        Light l1;

        public LightOffCommand(Light a) {
                this.l1 = a;
        }

        public void execute() {
                l1.off();
        }
}

class SimpleRemoteControl {
        Command slot;

        public SimpleRemoteControl() {
        }
        public void setCommand(Command command) {
                slot = command;
        }
        public void buttonWasPressed() {
                slot.execute();
        }
}
public class Main {
        public static void main(String[] args) {
                SimpleRemoteControl r1 = new
                SimpleRemoteControl(); Light l1 = new Light();

                LightOnCommand lo = new LightOnCommand(l1);
                r1.setCommand(lo);
                r1.buttonWasPressed();

                LightOffCommand lO = new LightOffCommand(l1);
                r1.setCommand(lO);
                r1.buttonWasPressed();
        }
 }
```

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

```
package com.test;

interface Command {
        public void execute();
}

class CeilingFan {
        public void on() {
                System.out.println("Ceiling Fan is on");
        }

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

}

class CeilingFanOnCommand implements Command
        { CeilingFan c;

        public CeilingFanOnCommand(CeilingFan l) {
                this.c = l;
        }

        public void execute() {
                c.on();
        }

}

class CeilingFanOffCommand implements Command
        { CeilingFan c;

        public CeilingFanOffCommand(CeilingFan l) {
                this.c = l;
        }

        public void execute() {
```

```
			c.off();
		}

}

class SimpleRemoteControl {
		Command slot;

		public SimpleRemoteControl() {
		}

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

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

}

public class Main {

		public static void main(String[] args) {
			SimpleRemoteControl remote = new
			SimpleRemoteControl(); CeilingFan ceilingFan = new
			CeilingFan();
			CeilingFanOnCommand ceilingFanOn = new
CeilingFanOnCommand(ceilingFan);
			remote.setCommand(ceilingFanOn);

			remote.buttonWasPressed();

			CeilingFanOffCommand ceilingFanOff = new
CeilingFanOffCommand(ceilingFan);
			remote.setCommand(ceilingFanOff);
			remote.buttonWasPressed();
		}
  }
```

8] Write a Java Program to implement Command Design Pattern for Command Interface with execute() . Use this to create variety of commands for LightOnCommand, LightOffCommand, GarageDoorUpCommand,StereoOnWithCDCommand.

```
package com.test;
interface Command {
		public void execute();
}

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 Light {

        String location = "";

        public Light(String location) {
                this.location = location;
        }

        public void on() {

                System.out.println(location + " light is on");

        }

        public void off() {

                System.out.println(location + " light is 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 StereoOffCommand implements Command
        { Stereo stereo;

        public StereoOffCommand(Stereo stereo) {
                this.stereo = stereo;
        }

        public void execute() {

                stereo.off();

        }

}

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();

        }

}

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

        public void undo() {
        }

}

public class RemoteLoader {
        public static void main(String[] args) {
                RemoteControlWithUndo remoteControl = new RemoteControlWithUndo();

                Light livingRoomLight = new Light("Living Room");
                Light kitchenLight = new Light("Kitchen");
                GarageDoor garageDoor = new GarageDoor("");
                Stereo stereo = new Stereo("Living Room");

                LightOnCommand livingRoomLightOn =

                                new LightOnCommand(livingRoomLight);
                LightOffCommand livingRoomLightOff = new
LightOffCommand(livingRoomLight);
                LightOnCommand kitchenLightOn = new
                LightOnCommand(kitchenLight); LightOffCommand kitchenLightOff =
                new LightOffCommand(kitchenLight);

                GarageDoorUpCommand garageDoorUp =

                                new GarageDoorUpCommand(garageDoor);
                GarageDoorDownCommand garageDoorDown = new
GarageDoorDownCommand(garageDoor);

                StereoOnWithCDCommand stereoOnWithCD =

                                new StereoOnWithCDCommand(stereo);
                StereoOffCommand stereoOff = new StereoOffCommand(stereo);

                remoteControl.setCommand(0, livingRoomLightOn,
                livingRoomLightOff); remoteControl.setCommand(1, kitchenLightOn,
                kitchenLightOff);
                remoteControl.setCommand(2, stereoOnWithCD, stereoOff);

                System.out.println(remoteControl);
```

```
		remoteControl.onButtonWasPushed(0);

		remoteControl.offButtonWasPushed(0);
		remoteControl.onButtonWasPushed(1);
		remoteControl.offButtonWasPushed(1);
		remoteControl.onButtonWasPushed(2);
		remoteControl.offButtonWasPushed(2);
	}

}
```

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.

```
package com.test;

public interface State {

	public void insertQuarter();

	public void ejectQuarter();

	public void turnCrank();

	public void dispense();

	public void refill();

}

public class NoQuarterState implements State {
	GumballMachine gumballMachine;

	public NoQuarterState(GumballMachine gumballMachine) {
		this.gumballMachine = gumballMachine;
	}

	public void insertQuarter() {
		System.out.println("You inserted a quarter");
		gumballMachine.setState(gumballMachine.getHasQuarterState());

	}

	public void ejectQuarter() {

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

	}

	public void turnCrank() {

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

	}

	public void dispense() {

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

	public void refill() {
	}

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

	}

}

public class GumballMachine {

	State soldOutState;

	State noQuarterState;
	State hasQuarterState;
	State soldState;

	State state;
	int count = 0;

	public GumballMachine(int numberGumballs) {
		soldOutState = new SoldOutState(this);
		noQuarterState = new NoQuarterState(this);
		hasQuarterState = new HasQuarterState(this);
		soldState = new SoldState(this);

		this.count = numberGumballs;
		if (numberGumballs > 0) {
			state = noQuarterState;

		} else {

			state = soldOutState;

		}

	}

	public void insertQuarter() {
		state.insertQuarter();
	}

	public void ejectQuarter() {
		state.ejectQuarter();
	}

	public void turnCrank() {
		state.turnCrank();
		state.dispense();
	}

	void releaseBall() {

		System.out.println("A gumball comes rolling out the
			slot..."); if (count != 0) {

				count = count - 1;

			}

		}

		int getCount() {

			return count;

		}

		void refill(int count) {
			this.count += count;
			System.out.println("The gumball machine was just refilled; it's new count is:
	" + this.count);

			state.refill();

		}

		void setState(State state) {
			this.state = state;
		}

		public State getState() {
			return state;
		}

		public State getSoldOutState() {
			return soldOutState;
		}

		public State getNoQuarterState() {
			return noQuarterState;
		}

		public State getHasQuarterState() {
			return hasQuarterState;
		}

		public State getSoldState() {
			return soldState;

		}

		public String toString() {

			StringBuffer result = new StringBuffer();
			result.append("\nMighty Gumball, Inc.");
			result.append("\nJava-enabled Standing Gumball Model #2004");
			result.append("\nInventory: " + count + " gumball");
			if (count != 1) {

				result.append("s");
			}
```

```
                result.append("\n");
                result.append("Machine is " + state + "\n");
                return result.toString();
        }

}

public class HasQuarterState implements State {
        GumballMachine gumballMachine;

         public HasQuarterState(GumballMachine gumballMachine)
                   { this.gumballMachine = gumballMachine;
        }

        public void insertQuarter() {

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

        }

        public void ejectQuarter() {
                System.out.println("Quarter returned");
                gumballMachine.setState(gumballMachine.getNoQuarterState())

        ; }

        public void turnCrank() {
                System.out.println("You turned...");
                gumballMachine.setState(gumballMachine.getSoldState());

        }

        public void dispense() {

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

        }

        public void refill() {
        }

        public String toString() {

                return "waiting for turn of crank";

        }

}

public class SoldState implements State {

        GumballMachine gumballMachine;
        public SoldState(GumballMachine gumballMachine) {
                this.gumballMachine = gumballMachine;
        }

        public void insertQuarter() {
            System.out.println("Please wait, we're already giving you a

        gumball"); }

        public void ejectQuarter() {

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

        }

        public void turnCrank() {

            System.out.println("Turning twice doesn't get you another

        gumball!"); }

        public void dispense() {
            gumballMachine.releaseBall();
            if (gumballMachine.getCount() > 0) {
                gumballMachine.setState(gumballMachine.getNoQuarterState());
            } else {

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

                gumballMachine.setState(gumballMachine.getSoldOutState());

            }

        }

        public void refill() {
        }

        public String toString() {

            return "dispensing a gumball";

        }

}

public class SoldOutState implements State {
        GumballMachine gumballMachine;

        public SoldOutState(GumballMachine gumballMachine) {
            this.gumballMachine = gumballMachine;
        }
        public void insertQuarter() {

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

        out"); }

        public void ejectQuarter() {

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

        public void turnCrank() {

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

        }

        public void dispense() {

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

        }

        public void refill() {

                gumballMachine.setState(gumballMachine.getNoQuarterState())

        ; }

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

}

public class GumballMachineTestDrive {

        public static void main(String[] args) {

                GumballMachine gumballMachine = new GumballMachine(2);

                System.out.println(gumballMachine);

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

                System.out.println(gumballMachine);

                gumballMachine.insertQuarter();

                gumballMachine.turnCrank();
                gumballMachine.insertQuarter();
                gumballMachine.turnCrank();
                gumballMachine.refill(5);
                gumballMachine.insertQuarter();
                gumballMachine.turnCrank();

                System.out.println(gumballMachine);

        }

}
```

10] Write a Java Program to implement Strategy Pattern for Duck Behavior. Create instance variable that holds current state of Duck from there, we just need to handle all Flying Behaviors and Quack Behavior

```
package com.test;
```

```
public abstract class Duck {

            FlyBehaviour flyBehaviour;
            QuackBehaviour quackBehaviour;

            public Duck() {
            }

            public abstract void display();

            public void performFly() {
                        flyBehaviour.fly();
            }

            public void performQuack() {
                        quackBehaviour.quack();
            }

public void swim() {

System.out.println("All ducks float, even decoys!);

}

            public void setFlyBehaviour(FlyBehaviour fb) {
                        flyBehaviour = fb;
            }

            public void setQuackBehaviour(QuackBehaviour qb) {
                        quackBehaviour qb;
            }

}

public class MallardDuck extends Duck {

            public MallardDuck() {
                        quackBehaviour = new Quack();
                        flyBehaviour = new FlyWithWings();
            }

            public void display() {

                        System.out.println("I'm a real Mallard duck");

            }

}

public class ModolDuck extends Duck {
            public ModelDuck() {
flyBehaviour = new FlyNoWay(); quackBehaviour = new Quack();
}
            public void display() {
                        System.out.println("I'm a model duck");
            }

}

public interface FlyBehaviour {
            public void fly();
}

public class FlyWithWings implements FlyBehaviour
```

```
          { public void fly() {
                    System.out.println("I'm flying!!");

          }

}

public class FlyNoWay implements FlyBehaviour
          { public void fly() {
                    System.out.println("I can't fly");

          }

}

public class FlyRocketPowered implements FlyBehaviour
          { public void fly() {
                    System.out.println("I'm flying with a rocket!");

          }

}

public interface QuackBehaviour {
          public void quack() {
System.out.println("Quack"); }

}

public class Quack implements QuackBehaviour {

          public void quack() {
                    System.out.println("Quack");
          }

}

public class MuteQuack implements QuackBehaviour
          { public void quack() {
                    System.out.println("<< Silence >>");

          }

}

 public class Squeak implements QuackBehaviour
                  { public void quack() {
                    System.out.println("Squeak");
          }
}

public class MiniDuckSimulator {
          public static void main(String[] args) {
                    Duck mallard = new MallardDuck();
                    mallard.performQuack();

                    mallard.performFly();

          }

}

public class MiniDuckSimulator {
          public static void main(String[] args) {
                    Duck mallard = new MallardDuck();
```

```
                mallard.performQuack();
                mallard.performFly();

                Duck model = new ModelDuck();

                model.performFly();

                model.setFlyBehaviour(new

                FlyRocketPowered()); model.performFly();

        }

}
```

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

```
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);
}

class BeatModel implements BeatModelInterface, MetaEventListener {
        Sequencer sequencer;
        ArrayList beatObservers = new ArrayList();

        ArrayList bpmObservers = new ArrayList();
        int bpm = 90;
}

        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();
        }

// Code to register and notify observers

// Lots of MIDI code to handle the beat
}

class DJView implements ActionListener, BeatObserver, BPMObserver
        { BeatModelInterface model;
        ControllerInterface controller;
        JFrame viewFrame;
        JPanel viewPanel;
        BeatBar beatBar;
        JLabel bpmOutputLabel;

        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

        }

        public void updateBPM() {
                int bpm = model.getBPM();
                if (bpm == 0) {
                        bpmOutputLabel.setText("offline");

                } else {

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

                }

        }

        public void updateBeat() {

                beatBar.setValue(100);

        }

}

interface ControllerInterface {
        void start();

        void stop();

        void increaseBPM();

        void decreaseBPM();

        void setBPM(int bpm);
}

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);
        }

}

public class Main {

        public static void main(String[] args) {
                BeatModelInterface model = new BeatModel();
                ControllerInterface controller = new

        BeatController(model); }

}
```

12] Write a Java Program to implement Decorator Pattern for interface Car to define the assemble() method and then decorate it to Sports car and Luxury Car.

```
public interface Car {
        public void assemble();
}

public class BasicCar implements Car {
        @Override
        public void assemble() {
                System.out.print("Basic Car.");
```

```
          }
}

public class CarDecorator implements Car {
          protected Car car;
          public CarDecorator(Car c) {
                    this.car = c;
          }
          @Override
          public void assemble() {
                    this.car.assemble();
          }
}

public class SportsCar extends CarDecorator {
          public SportsCar(Car c) {
                    super(c);
          }

          @Override
          public void assemble() {
                    super.assemble();
                    System.out.print("Adding features of Sports Car.");
          }

}

public class LuxuryCar extends CarDecorator {
          public LuxuryCar(Car c) {
                    super(c);
          }

          @Override
          public void assemble() {
                    super.assemble();
                    System.out.print("Adding features of Luxury Car.");
          }

}

public class DecoratorPatternTest {
          public static void main(String[] args) {
                    Car sportsCar = new SportsCar(new BasicCar());
                    sportsCar.assemble();
                    System.out.println("\n ");
                    Car sportsLuxuryCar = new SportsCar(new LuxuryCar(new
                    BasicCar())); sportsLuxuryCar.assemble();
          }

}
```

13] Write a Java Program to implement an Adapter design pattern in mobile charger. Define two classes – Volt (to measure volts) and Socket (producing constant volts of 120V). Build an adapter that can produce 3 volts, 12 volts and default 120 volts. Implements Adapter pattern using Class Adapter.

```
public class Volt {
          private int volts;
```

```
        public Volt(int v) {
                this.volts = v;
        }

        public int getVolts() {
                return volts;
        }

        public void setVolts(int volts) {
                this.volts = volts;
        }

}

public class Socket {

        public Volt getVolt() {
                return new Volt(120);
        }

}

public interface SocketAdapter {
        public Volt get120Volt();

        public Volt get12Volt();

        public Volt get3Volt();

}

// using inheritance for adapter pattern

public class SocketClassAdapterImpl extends Socket implements SocketAdapter
        { @Override
        public Volt get120Volt() {
                return getVolt();
        }

        @Override

        public Volt get12Volt() {
                Volt v = getVolt();
                return convertVolt(v, 10);

        }

        @Override

        public Volt get3Volt() {
                Volt v = getVolt();
                return convertVolt(v, 40);
        }

        private Volt convertVolt(Volt v, int i) {
                return new Volt(v.getVolts() / i);
        }

}

public class SocketObjectAdapterImpl implements SocketAdapter { //

using composition for adapter pattern private Socket sock = new Socket();
```

```
@Override

        public Volt get120Volt() {
                return sock.getVolt();

        }

        @Override

        public Volt get12Volt() {
                Volt v = sock.getVolt();
                return convertVolt(v, 10);
        }

        @Override

        public Volt get3Volt() {
                Volt v = sock.getVolt();
                return convertVolt(v, 40);

        }

        private Volt convertVolt(Volt v, int i) {
                return new Volt(v.getVolts() / i);
        }

}

public class AdapterPatternTest {
        public static void main(String[] args) {
                testClassAdapter();
                testObjectAdapter();
        }

        private static void testObjectAdapter() {

                SocketAdapter sockAdapter = new SocketObjectAdapterImpl();
                Volt v3 = getVolt(sockAdapter, 3);
                Volt v12 = getVolt(sockAdapter, 12);
                Volt v120 = getVolt(sockAdapter, 120);
                System.out.println("v3 volts using Object Adapter=" + v3.getVolts());
        System.out.println("v12 volts using Object Adapter=" + v12.getVolts());
        System.out.println("v120 volts using Object Adapter=" + v120.getVolts()); }

        private static void testClassAdapter() {
                SocketAdapter sockAdapter = new SocketClassAdapterImpl();

                Volt v3 = getVolt(sockAdapter, 3);
                Volt v12 = getVolt(sockAdapter, 12);
                Volt v120 = getVolt(sockAdapter, 120);

                System.out.println("v3 volts using Class Adapter=" + v3.getVolts());
        System.out.println("v12 volts using Class Adapter=" + v12.getVolts());
        System.out.println("v120 volts using Class Adapter=" + v120.getVolts()); }

        private static Volt getVolt(SocketAdapter sockAdapter, int i) {
                switch (i) {
                case 3:
                        return sockAdapter.get3Volt();
                case 12:
                        return sockAdapter.get12Volt();
                case 120:
                        return sockAdapter.get120Volt();
                default:
                        return sockAdapter.get120Volt();
                }
            }
        }
```
14] Write a Java Program to implement Facade Design Pattern for Home Theater.

```
public class Amplifier {
            String description;
            Tuner tuner;
            DvdPlayer dvd;
            CdPlayer cd;

            public Amplifier(String description) {
                        this.description = description;
            }

            public void on() {

                        System.out.println(description + " on");

            }

            public void off() {
                        System.out.println(description + " off");
            }

            public void setStereoSound() {
                        System.out.println(description + " stereo mode on");
            }

            public void setSurroundSound() {

                        System.out.println(description + " surround sound on (5 speakers,
1 subwoofer)");

            }

            public void setVolume(int level) {
                        System.out.println(description + " setting volume to " + level);
            }

            public void setTuner(Tuner tuner) {
                        System.out.println(description + " setting tuner to " + dvd);
                        this.tuner = tuner;
            }

            public void setDvd(DvdPlayer dvd) {
                        System.out.println(description + " setting DVD player to " +
                        dvd); this.dvd = dvd;
            }

            public void setCd(CdPlayer cd) {
                        System.out.println(description + " setting CD player to " + cd);
                        this.cd = cd;
            }

            public String toString() {
```

```
			return description;

		}

	}

	public class CdPlayer {
		String description;
		int currentTrack;

		Amplifier amplifier;

		String title;

		public CdPlayer(String description, Amplifier amplifier) {
			this.description = description;
			this.amplifier = amplifier;

		}

		public void on() {
			System.out.println(description + " on");
		}

		public void off() {

			System.out.println(description + " off");

		}

		public void eject() {
			title = null;
			System.out.println(description + " eject");

		}

		public void play(String title) {
			this.title = title;
			currentTrack = 0;

			System.out.println(description + " playing \"" + title + "\"");

		}

		public void play(int track) {
			if (title == null) {
				System.out.println(description + " can't play track " + currentTrack +
	", no cd inserted");

			} else {

				currentTrack = track;

				System.out.println(description + " playing track " + currentTrack);
			}

		}

		public void stop() {
```

```
                currentTrack = 0;
                System.out.println(description + " stopped");

        }

        public void pause() {

                System.out.println(description + " paused \"" + title +

        "\""); }

        public String toString() {
                return description;
        }

}

public class DvdPlayer {
        String description;
        int currentTrack;

        Amplifier amplifier;

        String movie;

         public DvdPlayer(String description, Amplifier amplifier)
                        { this.description = description;
                this.amplifier = amplifier;

        }

        public void on() {
                System.out.println(description + " on");
        }

        public void off() {
                System.out.println(description + " off");

        }

        public void eject() {
                movie = null;
                System.out.println(description + " eject");

        }

        public void play(String movie) {
                this.movie = movie;
                currentTrack = 0;
               System.out.println(description + " playing \"" + movie + "\"");
        }

        public void play(int track) {
                if (movie == null) {
                        System.out.println(description + " can't play track " + track + " no
dvd inserted");

                } else {
                        currentTrack = track;

                        System.out.println(description + " playing track " + currentTrack + "
of \"" + movie + "\"");

                }

        }

        public void stop() {
                currentTrack = 0;
                System.out.println(description + " stopped \"" + movie + "\"");

        }

        public void pause() {

                System.out.println(description + " paused \"" + movie + "\"");

        }

        public void setTwoChannelAudio() {
                System.out.println(description + " set two channel audio");
        }

        public void setSurroundAudio() {

                System.out.println(description + " set surround audio");

        }

        public String toString() {
                return description;
        }

}

public class Projector {
        String description;
        DvdPlayer dvdPlayer;

        public Projector(String description, DvdPlayer dvdPlayer) {
                this.description = description;
                this.dvdPlayer = dvdPlayer;
        }

        public void on() {
                System.out.println(description + " on");

        }

        public void off() {
                System.out.println(description + " off");
        }

        public void wideScreenMode() {
```

```
			System.out.println(description + " in widescreen mode (16x9 aspect
	ratio)"); }

	public void tvMode() {

			System.out.println(description + " in tv mode (4x3 aspect ratio)");

	}

	public String toString() {
			return description;
	}

}

public class TheaterLights {
	String description;

	public TheaterLights(String description) {
			this.description = description;
	}

	public void on() {
			System.out.println(description + " on");
	}

	public void off() {
			System.out.println(description + " off");
	}

	public void dim(int level) {

			System.out.println(description + " dimming to " + level + "%");

	}

	public String toString() {
			return description;
	}

}
public class Screen {
	String description;

	public Screen(String description) {
			this.description = description;
	}

	public void up() {
			System.out.println(description + " going up");
	}

	public void down() {
			System.out.println(description + " going down");

	}
```

```
        public String toString() {
                return description;
        }

}

public class PopcornPopper {
        String description;

        public PopcornPopper(String description) {
                this.description = description;
        }

        public void on() {
                System.out.println(description + " on");
        }

        public void off() {
                System.out.println(description + " off");
        }

        public void pop() {

                System.out.println(description + " popping

        popcorn!"); }

        public String toString() {
                return description;
        }

}

public class HomeTheaterFacade {
        Amplifier amp;
        Tuner tuner;
        DvdPlayer dvd;
        CdPlayer cd;
        Projector projector;
        TheaterLights lights;
        Screen screen;
        PopcornPopper popper;

        public HomeTheaterFacade(Amplifier amp, Tuner tuner, DvdPlayer dvd,
CdPlayer cd, Projector projector, Screen screen,
                        TheaterLights lights, PopcornPopper popper) {

                this.amp = amp;
                this.tuner = tuner;
                this.dvd = dvd;
                this.cd = cd;

                this.projector = projector;
                this.screen = screen;
                this.lights = lights;
                this.popper = popper;
        }
```

```
public void watchMovie(String movie) {
        System.out.println("Get ready to watch a movie...");
        popper.on();
        popper.pop();
        lights.dim(10);
        screen.down();
        projector.on();
        projector.wideScreenMode();
        amp.on();
        amp.setDvd(dvd);
        amp.setSurroundSound();
        amp.setVolume(5);

        dvd.on();
        dvd.play(movie);
}

public void endMovie() {

        System.out.println("Shutting movie theater down...");
        popper.off();
        lights.on();

        screen.up();
        projector.off();
        amp.off();
        dvd.stop();

        dvd.eject();

        dvd.off();

}
public void listenToCd(String cdTitle) {

        System.out.println("Get ready for an audiopile experence...");
        lights.on();
        amp.on();
        amp.setVolume(5);
        amp.setCd(cd);
        amp.setStereoSound();
        cd.on();
        cd.play(cdTitle);
}

public void endCd() {
        System.out.println("Shutting down CD...");
        amp.off();
        amp.setCd(cd);
        cd.eject();
        cd.off();

}

public void listenToRadio(double frequency) {
        System.out.println("Tuning in the airwaves...");
        tuner.on();
```

```
            tuner.setFrequency(frequency);
            amp.on();
            amp.setVolume(5);
            amp.setTuner(tuner);
        }

        public void endRadio() {
            System.out.println("Shutting down the tuner...");
            tuner.off();
            amp.off();

        }

}

public class HomeTheaterTestDrive {
        public static void main(String[] args) {
            Amplifier amp = new Amplifier("Top-O-Line Amplifier");

            Tuner tuner = new Tuner("Top-O-Line AM/FM Tuner", amp);
            DvdPlayer dvd = new DvdPlayer("Top-O-Line DVD Player",
            amp); CdPlayer cd = new CdPlayer("Top-O-Line CD Player",
            amp);
            Projector projector = new Projector("Top-O-Line Projector", dvd);
            TheaterLights lights = new TheaterLights("Theater Ceiling
            Lights"); Screen screen = new Screen("Theater Screen");
            PopcornPopper popper = new PopcornPopper("Popcorn

            Popper"); HomeTheaterFacade homeTheater =

                            new HomeTheaterFacade(amp, tuner, dvd, cd, projector,
screen, lights, popper);
            homeTheater.watchMovie("Raiders of the Lost
            Ark"); homeTheater.endMovie();
        }

}
```

15] Write a Java Program to implement Observer Design Pattern for number conversion. Accept a number in Decimal form and represent it in Hexadecimal, Octal and Binary. Change the Number and it reflects in other forms also.

```
import java.util.ArrayList;
import java.util.List;

public class Subject {

        private List<Observer> observers = new ArrayList<Observer>();
        private int state;

        public int getState() {
            return state;
        }

        public void setState(int state) {
            this.state = state;
            notifyAllObservers();
        }
        public void attach(Observer observer) {
                observers.add(observer);
        }

        public void notifyAllObservers() {

                for (Observer observer : observers) {
                        observer.update();
                }

        }

}

public abstract class Observer {
        protected Subject subject;

        public abstract void update();

}

public class BinaryObserver extends Observer {

        public BinaryObserver(Subject subject) {
                this.subject = subject;
                this.subject.attach(this);
        }

        @Override
        public void update() {
                System.out.println("Binary String: " +
Integer.toBinaryString(subject.getState()));
        }

}

public class OctalObserver extends Observer {

        public OctalObserver(Subject subject) {
                this.subject = subject;

                this.subject.attach(this);

        }

        @Override

        public void update() {

                System.out.println("Octal String: " +

        Integer.toOctalString(subject.getState())); }

}

public class HexaObserver extends Observer {

        public HexaObserver(Subject subject) {
```

```
		this.subject = subject;
		this.subject.attach(this);
	}

	@Override
	public void update() {
		System.out.println("Hex String: " +
Integer.toHexString(subject.getState()).toUpperCase());

	}

}

public class ObserverPatternDemo {
	public static void main(String[] args) {
		Subject subject = new Subject();

		new HexaObserver(subject);
		new OctalObserver(subject);
		new BinaryObserver(subject);

		System.out.println("First state change: 15");
		subject.setState(15);
		System.out.println("Second state change: 10");
		subject.setState(10);
	}

}
```