Software Architecture and Design Patterns

Practical Assignments

SLIP 1

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

Solution :

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

InputTest.java main file outside the package.

```
import DecoratorPackage.*;


import java.io.*;


public class InputTest {
        public static void main(String[] args) throws IOException {
                int c;


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


                        while((c = in.read()) >= 0) {
                                System.out.print((char)c);
                        }


                        in.close();
```

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

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

a) LowerCaseInputStream.java

```
package DecoratorPackage;

import java.io.*;

public class LowerCaseInputStream extends FilterInputStream {

        public LowerCaseInputStream(InputStream in) {
                super(in);
        }

        public int read() throws IOException {
                int c = super.read();
                return (c == -1 ? c : Character.toLowerCase((char)c));
        }

        public int read(byte[] b, int offset, int len) throws IOException {
                int result = super.read(b, offset, len);
                for (int i = offset; i<offset+result; i++) {
```

```
                b[i] = (byte)Character.toLowerCase((char)b[i]);
            }
            return result;
        }
}
```

Output :

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

SLIP 2

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

Solution:

SingletonTestDrive.java main file outside the package.

```
import SingletonPackage.*;

public class SingletonTestDrive {
    public static void main(String[] args) {
        Singleton foo = CoolerSingleton.getInstance();
        Singleton bar = HotterSingleton.getInstance();
        System.out.println(foo);
        System.out.println(bar);
    }
```

}

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

a) CoolerSingleton.java

```
package SingletonPackage;
public class CoolerSingleton extends Singleton {
	// useful instance variables here
	protected static Singleton uniqueInstance;
	private CoolerSingleton() {
		super();
	}

	// useful methods here
}
```

b)HotterSingleton.java

```
package SingletonPackage;

public class HotterSingleton extends Singleton {
	// useful instance variables here

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

```
	// useful methods here
}
```

c) Singleton.java

```
package SingletonPackage;

public class Singleton {
	protected static Singleton uniqueInstance;

	// other useful instance variables here

	protected Singleton() {}

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

	// other useful methods here
}
```

Output :

```
SingletonPackage.Singleton@2a139a55
```

SingletonPackage.Singleton@2a139a55

SLIP 3

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

Solution :

WeatherStationHeatIndex.java main file outside the package.

```
import weatherobservable.*;

public class WeatherStationHeatIndex

{
```

```
public static void main(String[] args)
{
WeatherDataweatherData = new WeatherData();
CurrentConditionsDisplaycurrentConditions = new CurrentConditionsDisplay(weatherData);
StatisticsDisplaystatisticsDisplay = new StatisticsDisplay(weatherData);
ForecastDisplayforecastDisplay = new ForecastDisplay(weatherData);
HeatIndexDisplayheatIndexDisplay = new HeatIndexDisplay(weatherData);

weatherData.setMeasurements(80, 65, 30.4f);
weatherData.setMeasurements(82, 70, 29.4f);
weatherData.setMeasurements(78, 90, 29.2f);
}
}
```

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

a) CurrentConditionsDisplay.java

```
package weatherobservable;
import java.util.Observable;
import java.util.Observer;

public class CurrentConditionsDisplay implements Observer, DisplayElement
{
```

```
    Observable observable;
    private float temperature;
    private float humidity;

    public CurrentConditionsDisplay(Observable observable)
  {
        this.observable = observable;
        observable.addObserver(this);
    }

    public void update(Observable obs, Object arg)
  {
    if (obsinstanceofWeatherData)
    {
        WeatherDataweatherData = (WeatherData)obs;
        this.temperature = weatherData.getTemperature();
        this.humidity = weatherData.getHumidity();
        display();
    }
  }

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

b)DisplayElement.java

```
package weatherobservable;

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

c) ForecastDisplay.java

```
package weatherobservable;


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

public class ForecastDisplay implements Observer, DisplayElement
{
	private float currentPressure = 29.92f;
	private float lastPressure;

	public ForecastDisplay(Observable observable)
    {
		observable.addObserver(this);
	}
```

```
    public void update(Observable observable, Object arg)
  {
     if (observable instanceofWeatherData)
   {
    WeatherDataweatherData = (WeatherData)observable;
    lastPressure = currentPressure;
    currentPressure = weatherData.getPressure();
    display();
      }
    }

    public void display()
  {
    System.out.print("Forecast: ");
     if (currentPressure>lastPressure)
   {
    System.out.println("Improving weather on the way!");
     } else if (currentPressure == lastPressure)
      {
            System.out.println("More of the same");
             } else if (currentPressure<lastPressure)
        {
            System.out.println("Watch out for cooler, rainy weather");
                }
    }
}
```

d) HeatIndexDisplay.java

```
package weatherobservable;

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

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

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

        public void update(Observable observable, Object arg)
    {
         if (observable instanceofWeatherData)
     {
        WeatherDataweatherData = (WeatherData)observable;
           float t = weatherData.getTemperature();
           float rh = weatherData.getHumidity();
        heatIndex = (float)
                        (
                        (16.923 + (0.185212 * t)) +
```

```
                    (5.37941 * rh) -
                    (0.100254 * t * rh) +
                    (0.00941695 * (t * t)) +
                    (0.00728898 * (rh * rh)) +
                    (0.000345372 * (t * t * rh)) -
                    (0.000814971 * (t * rh * rh)) +
                    (0.0000102102 * (t * t * rh * rh)) -
                    (0.000038646 * (t * t * t)) +
                    (0.0000291583 * (rh * rh * rh)) +
                    (0.00000142721 * (t * t * t * rh)) +
                    (0.000000197483 * (t * rh * rh * rh)) -
                    (0.0000000218429 * (t * t * t * rh * rh)) +
                    (0.000000000843296 * (t * t * rh * rh * rh)) -
                    (0.0000000000481975 * (t * t * t * rh * rh * rh)));
                    display();
            }
        }

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

e) StatisticsDisplay.java

```
package weatherobservable;
```

```
import java.util.Observable;
import java.util.Observer;

public class StatisticsDisplay implements Observer, DisplayElement
{
 private float maxTemp = 0.0f;
 private float minTemp = 200;
 private float tempSum= 0.0f;
 private int numReadings;

 public StatisticsDisplay(Observable observable)
 {
observable.addObserver(this);
 }

 public void update(Observable observable, Object arg)
 {
  if (observable instanceofWeatherData)
   {
          WeatherDataweatherData = (WeatherData)observable;
          float temp = weatherData.getTemperature();
          tempSum += temp;
          numReadings++;

          if (temp >maxTemp)
      {
          maxTemp = temp;
```

```
        }

        if (temp <minTemp)
    {
        minTemp = temp;
        }

        display();
  }
 }

 public void display()
 {
System.out.println("Avg/Max/Min temperature = " + (tempSum / numReadings)
                        + "/" + maxTemp + "/" + minTemp);
        }
}
```

f) WeatherData.java

```
package weatherobservable;

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

public class WeatherData extends Observable
{
```

```
    private float temperature;
    private float humidity;
    private float pressure;

    public WeatherData() { }

    public void measurementsChanged()
{
            setChanged();
            notifyObservers();
    }

    public void setMeasurements(float temperature, float humidity, float pressure)
{
            this.temperature = temperature;
            this.humidity = humidity;
            this.pressure = pressure;
            measurementsChanged();
    }

    public float getTemperature() {
            return temperature;
    }

    public float getHumidity() {
            return humidity;
    }
```

```
        public float getPressure() {
                return pressure;
        }
}
```

Output:

Heat index is2567.7097

Forecast :More of the same

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

Current conditions: 80.0F Degrees and 65.0%humidity

Heat index is3114.5156

Forecast :More of the same

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

Current conditions: 82.0F Degrees and 70.0%humidity

Heat index is4612.1143

Forecast :More of the same

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

Current conditions: 78.0F Degrees and 90.0%humidity

SLIP 4

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

Solution:

PizzaTestDrive.java main file outside the package.

```
import FactoryPackage.*;

public class PizzaTestDrive {

    public static void main(String[] args) {
        PizzaStorenyStore = new NYPizzaStore();
        PizzaStorechicagoStore = new ChicagoPizzaStore();

        Pizza pizza = nyStore.orderPizza("cheese");
        System.out.println("Ethan ordered a " + pizza.getName() + "\n");

        pizza = chicagoStore.orderPizza("cheese");
```

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

        }
}
```

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

a) Pizza.java

```
package FactoryPackage;

import java.util.ArrayList;

public abstract class Pizza {
        String name;
        String dough;
        String sauce;
        ArrayList toppings = new ArrayList();

        void prepare() {
            System.out.println("Preparing " + name);
            System.out.println("Tossing dough...");
            System.out.println("Adding sauce...");
            System.out.println("Adding toppings: ");
```

```
        for (int i = 0; i<toppings.size(); i++) {
                System.out.println("   " + toppings.get(i));
        }
}

void bake() {
        System.out.println("Bake for 25 minutes at 350");
}

void cut() {
        System.out.println("Cutting the pizza into diagonal slices");
}

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

public String getName() {
        return name;
}

public String toString() {
        StringBuffer display = new StringBuffer();
        display.append("---- " + name + " ----\n");
        display.append(dough + "\n");
        display.append(sauce + "\n");
        for (int i = 0; i<toppings.size(); i++) {
```

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

b) PizzaStore.java

```
package FactoryPackage;
public abstract class PizzaStore {

    abstract Pizza createPizza(String item);

    public Pizza orderPizza(String type) {
        Pizza pizza = createPizza(type);
        System.out.println("--- Making a " + pizza.getName() + " ---");
        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();
        return pizza;
    }
}
```

c)ChicagoPizzaStore.java

```
package FactoryPackage;

public class ChicagoPizzaStore extends PizzaStore {
```

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

d) ChicagoStyleCheesePizza.java

```
package FactoryPackage;

public class ChicagoStyleCheesePizza extends Pizza {

	public ChicagoStyleCheesePizza() {
		name = "Chicago Style Deep Dish Cheese Pizza";
		dough = "Extra Thick Crust Dough";
		sauce = "Plum Tomato Sauce";

		toppings.add("Shredded Mozzarella Cheese");
	}
```

```
	void cut() {
		System.out.println("Cutting the pizza into square slices");
	}
}
```

e)NYPizzaStore.java

```
package FactoryPackage;

public class NYPizzaStore extends PizzaStore {

	Pizza createPizza(String item) {
		if (item.equals("cheese")) {
			return new NYStyleCheesePizza();
		} else if (item.equals("veggie")) {
			return new NYStyleVeggiePizza();
		} else if (item.equals("clam")) {
			return new NYStyleClamPizza();
		} else if (item.equals("pepperoni")) {
			return new NYStylePepperoniPizza();
		} else return null;
	}
}
```

f)NYStyleCheesePizza

```
package FactoryPackage;

public class NYStyleCheesePizza extends Pizza {
```

```
        public NYStyleCheesePizza() {
                name = "NY Style Sauce and Cheese Pizza";
                dough = "Thin Crust Dough";
                sauce = "Marinara Sauce";

                toppings.add("Grated Reggiano Cheese");
        }
}
```

Output :

```
----Making aNY Style Sauce and Cheese Pizza----
Prearing NY Style Sauce and Cheese Pizza
Tossing dough....
Adding Sauce....
Adding Toppings ;
  Grated Reggiano Cheese
Bake for 25 minutes at 350 degrees
Cutting the pizza into Diagonal slices
Place pizza into official Pizzastore Box
Ethan ordered aNY Style Sauce and Cheese Pizza

----Making aChicago Style Deep Dish Cheese Pizza----
Prearing Chicago Style Deep Dish Cheese Pizza
Tossing dough....
Adding Sauce....
```

Adding Toppings ;

Shredded Mozzarela Cheese

Bake for 25 minutes at 350 degrees

Cutting the pizza into square slices

Place pizza into official Pizzastore Box

Joel ordered aChicago Style Deep Dish Cheese Pizza

SLIP 5

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

Solution:-

IteratorEnumerationTestDrive.java main file outside the package.

```
import iterenum.*;

import java.util.*;

public class IteratorEnumerationTestDrive {
        public static void main (String args[]) {
                ArrayList l = new ArrayList(Arrays.asList(args));
                Enumeration enumeration = new IteratorEnumeration(l.iterator());
```

```
            while (enumeration.hasMoreElements()) {
                System.out.println(enumeration.nextElement());
            }
        }
}
```

EnumerationIteratorTestDrive.java main file outside the package.

```
import iterenum.*;

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());
            }
        }
}
```

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

a) EnumerationIterator.java

```
package iterenum;

import java.util.*;
```

```
public class EnumerationIterator implements Iterator {
	Enumeration enumeration;

	public EnumerationIterator(Enumeration enumeration) {
		this.enumeration = enumeration;
	}

	public booleanhasNext() {
		return enumeration.hasMoreElements();
	}

	public Object next() {
		return enumeration.nextElement();
	}

	public void remove() {
		throw new UnsupportedOperationException();
	}
}
```

b)IteratorEnumeration.java

```
package iterenum;

import java.util.*;

public class IteratorEnumeration implements Enumeration {
```

```
    Iterator iterator;

    public IteratorEnumeration(Iterator iterator) {
      this.iterator = iterator;
    }

    public booleanhasMoreElements() {
      return iterator.hasNext();
    }

    public Object nextElement() {
      return iterator.next();
    }
}
```

SLIP 6

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

```
// A simple Java program to demonstrate
// implementation of Command Pattern using
// a remote control example.

// An interface for command
interface Command
{
      public void execute();
}
```

```
// Light class and its corresponding command
// classes
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;

      // The constructor is passed the light it
      // is going to control.
      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();
      }
}

// Stereo and its command classes
class Stereo
{
      public void on()
      {
            System.out.println("Stereo is on");
      }
      public void off()
      {
            System.out.println("Stereo is off");
      }
      public void setCD()
```

```
	{
		System.out.println("Stereo is set " +
					"for CD input");
	}
	public void setDVD()
	{
		System.out.println("Stereo is set"+
					" for DVD input");
	}
	public void setRadio()
	{
		System.out.println("Stereo is set" +
					" for Radio");
	}
	public void setVolume(int volume)
	{
	// code to set the volume
	System.out.println("Stereo volume set"
					+ " to " + volume);
	}
}
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);
	}
}

// A Simple remote control with one button
class SimpleRemoteControl
{
	Command slot; // only one button

	public SimpleRemoteControl()
```

```
	{
	}

	public void setCommand(Command command)
	{
		// set the command the remote will
		// execute
		slot = command;
	}

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

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

		// we can change command dynamically
		remote.setCommand(new
					LightOnCommand(light));
		remote.buttonWasPressed();
		remote.setCommand(new
				StereoOnWithCDCommand(stereo));
		remote.buttonWasPressed();
		remote.setCommand(new
				StereoOffCommand(stereo));
		remote.buttonWasPressed();
	}
}
```

SLIP 7

Write a Java Program to implement undo command to testCeilingFan.

Solution:

RemoteLoader.java main file outside the package.

```
import undo.*;


public class RemoteLoader
{
	public static void main(String[] args)
	{
		RemoteControlremoteControl = new RemoteControl();
		CeilingFanceilingFan = new CeilingFan("Living Room");
		CeilingFanMediumCommandceilingFanMedium = new
CeilingFanMediumCommand(ceilingFan);
		CeilingFanHighCommandceilingFanHigh = new
CeilingFanHighCommand(ceilingFan);
		CeilingFanOffCommandceilingFanOff = new CeilingFanOffCommand(ceilingFan);
```

```
        remoteControl.setCommand(0, ceilingFanMedium, ceilingFanOff);
        remoteControl.setCommand(1, ceilingFanHigh, ceilingFanOff);

        remoteControl.onButtonWasPushed(0);
        remoteControl.offButtonWasPushed(0);

        System.out.println(remoteControl);

        //remoteControl.undoButtonWasPushed();
        remoteControl.onButtonWasPushed(1);

        System.out.println(remoteControl);

        //remoteControl.undoButtonWasPushed();

    }

}
```

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

a) CeilingFan.java

```
package undo;
public class CeilingFan
{
```

```
        public static final int HIGH = 3;
        public static final int MEDIUM = 2;
        public static final int LOW = 1;
        public static final int OFF = 0;
        String location;
        int speed;

        public CeilingFan(String location)
        {

        this.location = location;
        speed = OFF;
        }

        public void high()
        {
        speed = HIGH;
        System.out.println(location + " ceiling fan is on high");
        }

public void medium()
        {
        speed = MEDIUM;
        System.out.println(location + " ceiling fan is on medium");
        }

        public void low()
```

```
        {
        speed = LOW;
        System.out.println(location + " ceiling fan is on low");
        }

        public void off()
        {
        speed = OFF;
        System.out.println(location + " ceiling fan is off");
        }

        public int getSpeed()
{
return speed;
}
}
```

b)CeilingFanHighCommand.java

```
package undo;

public class CeilingFanHighCommand implements Command
{
CeilingFanceilingFan;
int prevSpeed;

public CeilingFanHighCommand(CeilingFanceilingFan)
        {
```

```
this.ceilingFan = ceilingFan;
}

public void execute()
{
prevSpeed = ceilingFan.getSpeed();
ceilingFan.high();
}

public void undo()
{
if (prevSpeed == CeilingFan.HIGH)
        {
        ceilingFan.high();
}
else if (prevSpeed == CeilingFan.MEDIUM)
        {
        ceilingFan.medium();
        }
        else if (prevSpeed == CeilingFan.LOW)
        {
        ceilingFan.low();
        }
        else if (prevSpeed == CeilingFan.OFF)
        {
        ceilingFan.off();
        }
```

```
        }
}
```

c) CeilingFanLowCommand.java

```
package undo;
public class CeilingFanLowCommand implements Command
{
        CeilingFanceilingFan;
        int prevSpeed;

        public CeilingFanLowCommand(CeilingFanceilingFan)
        {
                this.ceilingFan = ceilingFan;
        }
        public void execute()
        {
        prevSpeed = ceilingFan.getSpeed();
                ceilingFan.low();
        }

        public void undo() {
                if (prevSpeed == CeilingFan.HIGH) {
                        ceilingFan.high();
                } else if (prevSpeed == CeilingFan.MEDIUM) {
                        ceilingFan.medium();
                } else if (prevSpeed == CeilingFan.LOW) {
                        ceilingFan.low();
```

```
            } else if (prevSpeed == CeilingFan.OFF) {
                ceilingFan.off();
            }
        }
}
```

d) CeilingFanMediumCommand.java

```
package undo;

public class CeilingFanMediumCommand implements Command
        {
        CeilingFanceilingFan;
        int prevSpeed;

        public CeilingFanMediumCommand(CeilingFanceilingFan)
        {
            this.ceilingFan = ceilingFan;
        }

        public void execute()
         {
            prevSpeed = ceilingFan.getSpeed();
            ceilingFan.medium();
        }

        public void undo() {
            if (prevSpeed == CeilingFan.HIGH) {
```

```
                ceilingFan.high();
            } else if (prevSpeed == CeilingFan.MEDIUM) {
                ceilingFan.medium();
            } else if (prevSpeed == CeilingFan.LOW) {
                ceilingFan.low();
            } else if (prevSpeed == CeilingFan.OFF) {
                ceilingFan.off();
            }
        }
}
```

e)CeilingFanOffCommand.java

```
package undo;

public class CeilingFanOffCommand implements Command
{
    CeilingFanceilingFan;
    int prevSpeed;

    public CeilingFanOffCommand(CeilingFanceilingFan)
    {
        this.ceilingFan = ceilingFan;
    }

    public void execute()
    {
        prevSpeed = ceilingFan.getSpeed();
```

```
            ceilingFan.off();
        }

        public void undo() {
            if (prevSpeed == CeilingFan.HIGH) {
                ceilingFan.high();
            } else if (prevSpeed == CeilingFan.MEDIUM) {
                ceilingFan.medium();
            } else if (prevSpeed == CeilingFan.LOW) {
                ceilingFan.low();
            } else if (prevSpeed == CeilingFan.OFF) {
                ceilingFan.off();
            }
        }
}
```

f)Command.java

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

e) RemoteControl.java

```
package undo;
```

```
import java.util.*;

public class RemoteControl
{
	Command[] onCommands;
	Command[] offCommands;

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

		/*Command noCommand = new NoCommand();
		for (int i = 0; i< 7; i++)
		{
			onCommands[i] = noCommand;
			offCommands[i] = noCommand;
		}*/
	}

	public void setCommand(int slot, Command onCommand, Command offCommand) {
		onCommands[slot] = onCommand;
		offCommands[slot] = offCommand;
	}

	public void onButtonWasPushed(int slot) {
		onCommands[slot].execute();
```

```
	}

	public void offButtonWasPushed(int slot) {
		offCommands[slot].execute();
	}

	public String toString() {
		StringBufferstringBuff = new StringBuffer();
		stringBuff.append("\n------ Remote Control -------\n");
		for (int i = 0; i<onCommands.length; i++) {
			stringBuff.append("[slot " + i + "] " +
onCommands[i].getClass().getName()
				+ "    " + offCommands[i].getClass().getName() + "\n");
		}
		return stringBuff.toString();
	}
}
```

f)RemoteControlWithUndo.java

```
package undo;
import java.util.*;

public class RemoteControlWithUndo {
	Command[] onCommands;
	Command[] offCommands;
	Command undoCommand;
```

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

        /*Command noCommand = new Command();
        for(int i=0;i<7;i++) {
                onCommands[i] = noCommand;
                offCommands[i] = noCommand;
        }
        undoCommand = noCommand;*/
}

public void setCommand(int slot, Command onCommand, Command offCommand) {
        onCommands[slot] = onCommand;
        offCommands[slot] = offCommand;
}

public void onButtonWasPushed(int slot) {
        onCommands[slot].execute();
        undoCommand = onCommands[slot];
}

public void offButtonWasPushed(int slot) {
        offCommands[slot].execute();
        undoCommand = offCommands[slot];
}
```

```
	public void undoButtonWasPushed() {
		undoCommand.undo();
	}

	public String toString() {
		StringBufferstringBuff = 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();
	}
}
```

Output :

Living Room ceiling fan is on medium

Living Room ceiling fan is off

SLIP 8

```
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
interface State {

 public void insertQuarter();
 public void ejectQuarter();
 public void turnCrank();
 public void dispense();

 public void refill();
}
 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";
 }
}
 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();
 }
}
 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";
 }
}
 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";
 }
}
 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 Main {
 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);
 }
}
```

SLIP 9

```
package hrapplication.controller;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
```

```java
import org.springframework.web.bind.annotation.RequestParam;
import hrapplication.model.HrApplicationBean;
import hrapplication.model.LoginBean;
import hrapplication.service.HrApplicationService;
import hrapplication.service.LoginService;


@Controller
@RequestMapping("/hrApplications")
public class HrApplicationController {

    @Autowired
    private HrApplicationService hrApplicationService;


    @RequestMapping("/secure")
    public String defaultPage(Model model) {
        List<HrApplicationBean> hrApplications = hrApplicationService.getAllHrApplication();
        model.addAttribute("hrApplications", hrApplications);
        return "hrApplicationSecure";
    }

    @RequestMapping("/all")
    public String allHrApplications(Model model) {

        List<HrApplicationBean> hrApplications = hrApplicationService.getAllHrApplication();
        model.addAttribute("hrApplications", hrApplications);
        return "hrApplicationList";
    }

    @RequestMapping("/hrApplication")
    public String getHrApplicationById(@RequestParam(name = "hrApplicationId") int hrApplicationId, Model model)
{
        HrApplicationBean hrApplication = hrApplicationService.getHrApplicationById(hrApplicationId);
        model.addAttribute("hrApplication", hrApplication);
        return "hrApplicationDetail";
    }

    @RequestMapping(value = "/add", method = RequestMethod.GET)
    public String addHrApplication(@ModelAttribute("newHrApplication") HrApplicationBean
hrApplicationBean,BindingResult bindingResult) {

        return "hrApplicationAdd";
    }

    @RequestMapping("/deleteHrApplication{hrApplicationId}")
    public String deleteHrApplication(@RequestParam("hrApplicationId") int hrApplicationId, Model model) {
        hrApplicationService.deleteHrApplication(hrApplicationId);
        return "hrApplicationList";
    }


    @RequestMapping(value = "/save", method = RequestMethod.POST)
    public String saveHrApplication(@ModelAttribute("newHrApplication") HrApplicationBean
hrApplicationBean,BindingResult bindingResult) {

        hrApplicationService.addHrApplication(hrApplicationBean);
        return "hrApplicationList";
    }

}
```

```java
package hrapplication.controller;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.SessionAttribute;
import org.springframework.web.servlet.ModelAndView;
import hrapplication.model.ApplicantBean;
import hrapplication.model.HrApplicationBean;
import hrapplication.service.ApplicantService;
import hrapplication.service.HrApplicationService;


public class ApplicantController {

    @Autowired
    private ApplicantService applicantService;

    @Autowired
    private HrApplicationService hrApplicationService;

    @RequestMapping(method = RequestMethod.GET)
    public ModelAndView addApplicant(@ModelAttribute("newApplicant") ApplicantBean applicantBean,BindingResult
bindingResult) {
        Map<String, Object> model = new HashMap<String,Object>();
        model.put("hrApplications",  hrApplicationService.getAllHrApplication());
        return new ModelAndView("applicantAdd",model);
    }

    @RequestMapping(value = "/save", method = RequestMethod.POST)
    public String saveApplicant(@ModelAttribute("newApplicant") ApplicantBean applicantBean,BindingResult
bindingResult, HrApplicationBean hrApplicationBean){

        applicantService.save(applicantBean);
        applicantService.update(applicantBean,hrApplicationService.getAllHrApplicationId(hrApplicationBean));

        return "redirect:/hrApplications";
    }




}


package hrapplication.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
```

```java
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import hrapplication.model.LoginBean;
import hrapplication.service.LoginService;

@Controller
public class LoginController {

    @Autowired
    private LoginService loginService;

    @RequestMapping(method = RequestMethod.GET)
    public String defaultPage(LoginBean loginBean,BindingResult bindingResult) {
        //start project
            return "index";
    }

    @RequestMapping(value="/login",method = RequestMethod.GET)
    public String addLogin(LoginBean loginBean,BindingResult bindingResult) {
        loginService.addLogin(loginBean);
        return "login";
    }

     @RequestMapping("/login")
       public ModelAndView login(HttpServletRequest request,HttpServletResponse response,LoginBean
loginBean,BindingResult bindingResult) {
          loginService.addLogin(loginBean);
          String loginEmail=request.getParameter("loginEmail");
          String loginPassword=request.getParameter("loginPassword");
          String message;
          if(loginEmail != null &&
                  !loginEmail.equals("")
                  && loginEmail.equals(loginBean.getLoginEmail()) &&
                  loginPassword != null &&
                  !loginPassword.equals("") &&
                  loginPassword.equals(loginBean.getLoginPassword())){
              message = "Welcome " +loginEmail + ".";
              return new ModelAndView("hrApplicationList");

          }else{
              message = "Wrong email or password.";
              return new ModelAndView("login",
                      "message", message);
          }
       }


}
```

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

```
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 q;
}
}
class MallardDuck extends Duck {
public MallardDuck() {
quackBehaviour = new Quack();
flyBehaviour = new FlyWithWings();
}
public void display() {
System.out.println("I'm a real Mallard duck");
}
}
interface FlyBehaviour {
public void fly();
}
interface QuackBehaviour {
public void quack() {
System.out.println("Quack");
}
}
class Quack implements QuackBehaviour {
```

```
 public void quack() {
 System.out.println("Quack");
 }
}
 class FlyWithWings implements FlyBehaviour {
 public void fly() {
 System.out.println("I'm flying!!");
 }
                                                                                                                        }
public class Main {
 public static void main(String[] args) {
 Duck mallard = new MallardDuck();
 mallard.performQuack();
 mallard.performFly();
 }
}
/*Simple Programme*/
interface DuckB
{
r
 public void oper();
}
class Fly implements DuckB
{
 public void oper()
 {
 System.out.println("Duck Flies");
 }
}
 class Quack implements DuckB
{
 public void oper()
 {
 System.out.println("Duck Sounds Quack Quack");
 }
}
class Context
{
 private DuckB s1;
 public Context(DuckB p)
 {
 this.s1=p;
 }
 public void est()
 {
 s1.oper();
 }
}
public class Main
{
public static void main(String[] args) {
Context c1=new Context(new Fly());
System.out.println("Duck Behaviour");
c1.est();
c1=new Context(new Quack());
System.out.println("Duck Behaviour ");
c1.est();
 }
}
```