Software Architecture & Design Pattern List of Assignments

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

```
package observer;

public class WeatherStation {


	public static void main(String[] args) {

		WeatherData weatherData = new WeatherData();


		CurrentConditionsDisplay currentConditions = new
CurrentConditionsDisplay(weatherData);

		StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);

		ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);


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

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

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

	}

}


package observer;


import java.util.*;


public class HeatIndexDisplay implements Observer, DisplayElement {

	float heatIndex = 0.0f;
```

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

public void update(Observable observable, Object arg) {
        if (observable instanceof WeatherData) {
                WeatherData weatherData = (WeatherData)observable;
                float t = weatherData.getTemperature();
                float rh = weatherData.getHumidity();
                heatIndex = (float)
                        (
                        (16.923 + (0.185212 * t)) +
                        (5.37941 * rh) -
                        (0.100254 * t * rh) +
                        (0.00941695 * (t * t)) +
                        (0.00728898 * (rh * rh)) +
                        (0.000345372 * (t * t * rh)) -
                        (0.000814971 * (t * rh * rh)) +
                        (0.0000102102 * (t * t * rh * rh)) -
                        (0.000038646 * (t * t * t)) +
                        (0.0000291583 * (rh * rh * rh)) +
                        (0.00000142721 * (t * t * t * rh)) +
                        (0.000000197483 * (t * rh * rh * rh)) -
                        (0.0000000218429 * (t * t * t * rh * rh)) +
                        (0.000000000843296 * (t * t * rh * rh * rh)) -
                        (0.0000000000481975 * (t * t * t * rh * rh * rh)));
```

```
                display();
            }
        }

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

package observer;

import java.util.*;

@SuppressWarnings("deprecation")
public class ForecastDisplay implements Observer, DisplayElement {
        private float currentPressure = 29.92f;
        private float lastPressure;

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

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

```
                display();
            }
        }

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

package observer;

import java.util.*;

public class CurrentConditionsDisplay implements Observer, DisplayElement {
        Observable observable;
        private float temperature;
        private float humidity;

        public CurrentConditionsDisplay(Observable observable) {
            this.observable = observable;
```

```
            observable.addObserver(this);
        }

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

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

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

```
package decorator;

import java.io.*;

public class LowerCaseInputStream extends FilterInputStream {

        public LowerCaseInputStream(InputStream in) {
```

```
            super(in);
        }

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

        public int read(byte[] b, int offset, int len) throws IOException {
            int result = in.read(b, offset, len);
            for (int i = offset; i < offset+result; i++) {
                b[i] = (byte)Character.toLowerCase((char)b[i]);
            }
            return result;
        }
}

package decorator;

import java.io.*;

public class InputTest {
    public static void main(String[] args) throws IOException {
        int c;
        InputStream in = null;
        try {
            in =
```

```
                        new LowerCaseInputStream(
                                new BufferedInputStream(
                                        new FileInputStream("test.txt")));

                while((c = in.read()) >= 0) {
                        System.out.print((char)c);
                }
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                if (in != null) { in.close(); }
            }
            System.out.println();
            try (InputStream in2 =
                        new LowerCaseInputStream(
                                new BufferedInputStream(
                                        new FileInputStream("test.txt"))))
            {
                while((c = in2.read()) >= 0) {
                        System.out.print((char)c);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
}
```

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

```
package factory;


public class CheesePizza extends Pizza {
	public CheesePizza() {
		name = "Cheese Pizza";
		dough = "Regular Crust";
		sauce = "Marinara Pizza Sauce";
		toppings.add("Fresh Mozzarella");
		toppings.add("Parmesan");
	}
}


package factory;
public class ClamPizza extends Pizza {
	public ClamPizza() {
		name = "Clam Pizza";
		dough = "Thin crust";
		sauce = "White garlic sauce";
		toppings.add("Clams");
		toppings.add("Grated parmesan cheese");
	}
}


package factory;
public class PepperoniPizza extends Pizza {
```

```
	public PepperoniPizza() {
		name = "Pepperoni Pizza";
		dough = "Crust";
		sauce = "Marinara sauce";
		toppings.add("Sliced Pepperoni");
		toppings.add("Sliced Onion");
		toppings.add("Grated parmesan cheese");
	}
}

package factory;
import java.util.*;

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

	public String getName() {
		return name;
	}

	public void prepare() {
		System.out.println("Preparing " + name);
	}
```

```
	public void bake() {
		System.out.println("Baking " + name);
	}

	public void cut() {
		System.out.println("Cutting " + name);
	}

	public void box() {
		System.out.println("Boxing " + name);
	}

	public String toString() {
		// code to display pizza name and ingredients
		StringBuffer display = new StringBuffer();
		display.append("---- " + name + " ----\n");
		display.append(dough + "\n");
		display.append(sauce + "\n");
		for (String topping : toppings) {
			display.append(topping + "\n");
		}
		return display.toString();
	}
}

package factory;
public class PizzaStore {
```

```
	SimplePizzaFactory factory;

	public PizzaStore(SimplePizzaFactory factory) {
		this.factory = factory;
	}

	public Pizza orderPizza(String type) {
		Pizza pizza;

		pizza = factory.createPizza(type);

		pizza.prepare();
		pizza.bake();
		pizza.cut();
		pizza.box();

		return pizza;
	}

}

package factory;
public class PizzaTestDrive {

	public static void main(String[] args) {
		SimplePizzaFactory factory = new SimplePizzaFactory();
		PizzaStore store = new PizzaStore(factory);
```

```
		Pizza pizza = store.orderPizza("cheese");
		System.out.println("We ordered a " + pizza.getName() + "\n");
		System.out.println(pizza);

		pizza = store.orderPizza("veggie");
		System.out.println("We ordered a " + pizza.getName() + "\n");
		System.out.println(pizza);
	}
}

package factory;
public class SimplePizzaFactory {

	public Pizza createPizza(String type) {
		Pizza pizza = null;

		if (type.equals("cheese")) {
			pizza = new CheesePizza();
		} else if (type.equals("pepperoni")) {
			pizza = new PepperoniPizza();
		} else if (type.equals("clam")) {
			pizza = new ClamPizza();
		} else if (type.equals("veggie")) {
			pizza = new VeggiePizza();
		}
		return pizza;
```

```
	}
}
```

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

```
package singleton;

public class Singleton {
	private static Singleton uniqueInstance;

	// other useful instance variables here

	private Singleton() {}

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

	// other useful methods here
	public String getDescription() {
		return "I'm a thread safe Singleton!";
	}
}

package singleton;
public class SingletonClient {
```

```
	public static void main(String[] args) {
		Singleton singleton = Singleton.getInstance();
		System.out.println(singleton.getDescription());
	}
}
```

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

```
package command;
public class RemoteLoader {

	public static void main(String[] args) {
		RemoteControl remoteControl = new RemoteControl();

		Light livingRoomLight = new Light("Living Room");
		Light kitchenLight = new Light("Kitchen");
		CeilingFan ceilingFan= new CeilingFan("Living Room");
		GarageDoor garageDoor = new GarageDoor("Garage");
		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);
```

```
CeilingFanOnCommand ceilingFanOn =
                new CeilingFanOnCommand(ceilingFan);
CeilingFanOffCommand ceilingFanOff =
                new CeilingFanOffCommand(ceilingFan);

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, ceilingFanOn, ceilingFanOff);
remoteControl.setCommand(3, stereoOnWithCD, stereoOff);

System.out.println(remoteControl);

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

```
                remoteControl.offButtonWasPushed(2);
                remoteControl.onButtonWasPushed(3);
                remoteControl.offButtonWasPushed(3);
        }
}

package command;
//
// This is the invoker
//
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() {
		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");
		}
		return stringBuff.toString();
	}
}

package command;
public class LivingroomLightOnCommand implements Command {
	Light light;
```

```
	public LivingroomLightOnCommand(Light light) {
		this.light = light;
	}

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

package command;
public class LivingroomLightOffCommand implements Command {
	Light light;

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

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

package command;
public class LightOnCommand implements Command {
	Light light;

	public LightOnCommand(Light light) {
```

```
        this.light = light;
    }

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

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

```
package command;

public class CeilingFan {
    String location = "";
    int level;
    public static final int HIGH = 2;
    public static final int MEDIUM = 1;
    public static final int LOW = 0;

    public CeilingFan(String location) {
        this.location = location;
        System.out.println("run PROGRAM");
    }

    public void high() {
        // turns the ceiling fan on to high
        level = HIGH;
```

```
            System.out.println(location + " ceiling fan is on high");

        }

        public void medium() {
            // turns the ceiling fan on to medium
            level = MEDIUM;
            System.out.println(location + " ceiling fan is on medium");
        }

        public void low() {
            // turns the ceiling fan on to low
            level = LOW;
            System.out.println(location + " ceiling fan is on low");
        }

        public void off() {
            // turns the ceiling fan off
            level = 0;
            System.out.println(location + " ceiling fan is off");
        }

        public int getSpeed() {
            return level;
        }
}
package command;
```

```
public class RemoteLoader {

        public static void main(String[] args) {
                RemoteControl remoteControl = new RemoteControl();

                Light livingRoomLight = new Light("Living Room");
                Light kitchenLight = new Light("Kitchen");
                CeilingFan ceilingFan= new CeilingFan("Living Room");
                GarageDoor garageDoor = new GarageDoor("Garage");
                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);

                CeilingFanOnCommand ceilingFanOn =
                                new CeilingFanOnCommand(ceilingFan);
                CeilingFanOffCommand ceilingFanOff =
                                new CeilingFanOffCommand(ceilingFan);

                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, ceilingFanOn, ceilingFanOff);
        remoteControl.setCommand(3, stereoOnWithCD, stereoOff);

        System.out.println(remoteControl);

        remoteControl.onButtonWasPushed(0);
        remoteControl.offButtonWasPushed(0);
        remoteControl.onButtonWasPushed(1);
        remoteControl.offButtonWasPushed(1);
        remoteControl.onButtonWasPushed(2);
        remoteControl.offButtonWasPushed(2);
        remoteControl.onButtonWasPushed(3);
        remoteControl.offButtonWasPushed(3);
    }
}

package command;
```

```
//
// This is the invoker
//
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() {
		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");
		}
		return stringBuff.toString();
	}
}

package command;
public class LivingroomLightOnCommand implements Command {
	Light light;

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

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

```
}

package command;
public class LivingroomLightOffCommand implements Command {
	Light light;

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

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

package command;
public class LightOnCommand implements Command {
	Light light;

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

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

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

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

public class EnumerationIterator implements Iterator<Object> {
	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();
	}
}
package adapter;

import java.util.*;
```

```
public class EI {
	public static void main (String args[]) {
		Vector<String> v = new Vector<String>(Arrays.asList(args));
		Enumeration<String> enumeration = v.elements();
		while (enumeration.hasMoreElements()) {

			System.out.println(enumeration.nextElement());
		}
		Iterator<String> iterator = v.iterator();
		while (iterator.hasNext()) {
			System.out.println(iterator.next());
		}
	}
}
package adapter;
import java.util.*;

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

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

public class IteratorEnumerationTestDrive {
	public static void main (String args[]) {
		ArrayList<String> l = new ArrayList<String>(Arrays.asList(args));
		Enumeration<?> enumeration = new IteratorEnumeration(l.iterator());
		while (enumeration.hasMoreElements()) {
			System.out.println(enumeration.nextElement());
		}
	}
}
```

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

```
package iterator;

import java.util.ArrayList;

public class MenuTestDrive {
	public static void main(String args[]) {
		PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();
		DinerMenu dinerMenu = new DinerMenu();
		ArrayList<Menu> menus = new ArrayList<Menu>();
		menus.add(pancakeHouseMenu);
		menus.add(dinerMenu);
		Waitress waitress = new Waitress(menus);
		waitress.printMenu();
```

```
        }
}

package iterator;

import java.util.Iterator;

public class DinerMenu implements Menu {
        static final int MAX_ITEMS = 6;
        int numberOfItems = 0;
        MenuItem[] menuItems;

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

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

```
        addItem("Pasta",
                "Spaghetti with Marinara Sauce, and a slice of sourdough bread",
                true, 3.89);
}

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

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

public Iterator<MenuItem> createIterator() {
        return new DinerMenuIterator(menuItems);
        //return new AlternatingDinerMenuIterator(menuItems);
}

// other menu methods here
```

```
}

package iterator;
import java.util.Iterator;

public interface Menu {
	public Iterator<?> createIterator();
}

package iterator;
public class MenuItem {
	String name;
	String description;
	boolean vegetarian;
	double price;

	public MenuItem(String name,
		  String description,
		  boolean vegetarian,
		  double price)
	{
		this.name = name;
		this.description = description;
		this.vegetarian = vegetarian;
		this.price = price;
	}
```

```
	public String getName() {
		return name;
	}

	public String getDescription() {
		return description;
	}

	public double getPrice() {
		return price;
	}

	public boolean isVegetarian() {
		return vegetarian;
	}
}

package iterator;
import java.util.*;


public class Waitress {
	ArrayList<Menu> menus;


	public Waitress(ArrayList<Menu> menus) {
		this.menus = menus;
```

```
	}

	public void printMenu() {
		Iterator<?> menuIterator = menus.iterator();
		while(menuIterator.hasNext()) {
			Menu menu = (Menu)menuIterator.next();
			printMenu(menu.createIterator());
		}
	}

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

9. Write a Java Program to implement State Pattern for Gumball Machine. Create instance variable that holds current state from there, we just need to handle all actions, behaviors and state transition that can happen. For actions we need to implement methods to insert a quarter, remove a quarter, turning the crank and display gumball.

```
package state;

public class GumballMachine {

	State soldOutState;
```

```
State noQuarterState;
State hasQuarterState;
State soldState;
State winnerState;

State state = soldOutState;
int count = 0;

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

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

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

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

```
	}

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

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

	void releaseBall() {
		System.out.println("A gumball comes rolling out the slot...");
		if (count > 0) {
			count = count - 1;
		}
	}

	int getCount() {
		return count;
	}

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

```
        }

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 State getWinnerState() {
   return winnerState;
}

        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();
        }
}
package state;
public class GumballMachineTestDrive {

        public static void main(String[] args) {
                GumballMachine gumballMachine =
                        new GumballMachine(10);

                System.out.println(gumballMachine);

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

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

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

System.out.println(gumballMachine);

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

System.out.println(gumballMachine);

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

System.out.println(gumballMachine);

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

```
            System.out.println(gumballMachine);
        }
}

package state;
import java.util.Random;

public class HasQuarterState implements State {
        Random randomWinner = new Random(System.currentTimeMillis());
        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...");
            int winner = randomWinner.nextInt(10);
```

```
                if ((winner == 0) && (gumballMachine.getCount() > 1)) {
                        gumballMachine.setState(gumballMachine.getWinnerState());
                } else {
                        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";
        }
}
package state;
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";
        }
}

package state;
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";
        }
}
```

```
package state;
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";
        }
}

package state;

public interface State {

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

        public void refill();
}
package state;
public class WinnerState implements State {
   GumballMachine gumballMachine;

   public WinnerState(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("Please wait, we're already giving you a Gumball");
        }

        public void turnCrank() {
                System.out.println("Turning again doesn't get you another gumball!");
        }

        public void dispense() {
                gumballMachine.releaseBall();
                if (gumballMachine.getCount() == 0) {
                        gumballMachine.setState(gumballMachine.getSoldOutState());
                } else {
                        gumballMachine.releaseBall();
                        System.out.println("YOU'RE A WINNER! You got two gumballs for your
quarter");
                        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 "despensing two gumballs for your quarter, because YOU'RE A WINNER!";
        }
}
```

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

```
package combined;

public class HeartTestDrive {

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

public interface HeartModelInterface {
        int getHeartRate();
        void registerObserver(BeatObserver o);
        void removeObserver(BeatObserver o);
```

```
        void registerObserver(BPMObserver o);
        void removeObserver(BPMObserver o);
}
package combined;

import java.util.*;

public class HeartModel implements HeartModelInterface, Runnable {
        ArrayList beatObservers = new ArrayList();
        ArrayList bpmObservers = new ArrayList();
        int time = 1000;
   int bpm = 90;
        Random random = new Random(System.currentTimeMillis());
        Thread thread;

        public HeartModel() {
                thread = new Thread(this);
                thread.start();
        }

        public void run() {
                int lastrate = -1;

                for(;;) {
                        int change = random.nextInt(10);
                        if (random.nextInt(2) == 0) {
                                change = 0 - change;
```

```
                }
                int rate = 60000/(time + change);
                if (rate < 120 && rate > 50) {
                        time += change;
                        notifyBeatObservers();
                        if (rate != lastrate) {
                                lastrate = rate;
                                notifyBPMObservers();
                        }
                }
                try {
                        Thread.sleep(time);
                } catch (Exception e) {}
        }
}
public int getHeartRate() {
        return 60000/time;
}

public void registerObserver(BeatObserver o) {
        beatObservers.add(o);
}

public void removeObserver(BeatObserver o) {
        int i = beatObservers.indexOf(o);
        if (i >= 0) {
                beatObservers.remove(i);
```

```
        }
    }

    public void notifyBeatObservers() {
        for(int i = 0; i < beatObservers.size(); i++) {
            BeatObserver observer = (BeatObserver)beatObservers.get(i);
            observer.updateBeat();
        }
    }

    public void registerObserver(BPMObserver o) {
        bpmObservers.add(o);
    }

    public void removeObserver(BPMObserver o) {
        int i = bpmObservers.indexOf(o);
        if (i >= 0) {
            bpmObservers.remove(i);
        }
    }

    public void notifyBPMObservers() {
        for(int i = 0; i < bpmObservers.size(); i++) {
            BPMObserver observer = (BPMObserver)bpmObservers.get(i);
            observer.updateBPM();
        }
    }
```

```
}
package combined;

public class HeartController implements ControllerInterface {
	HeartModelInterface model;
	DJView view;

	public HeartController(HeartModelInterface model) {
		this.model = model;
		view = new DJView(this, new HeartAdapter(model));
      view.createView();
      view.createControls();
		view.disableStopMenuItem();
		view.disableStartMenuItem();
	}

	public void start() {}

	public void stop() {}

	public void increaseBPM() {}

	public void decreaseBPM() {}

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

```
package combined;

public class HeartAdapter implements BeatModelInterface {
	HeartModelInterface heart;

	public HeartAdapter(HeartModelInterface heart) {
		this.heart = heart;
	}

  public void initialize() {}

  public void on() {}

  public void off() {}

	public int getBPM() {
		return heart.getHeartRate();
	}

  public void setBPM(int bpm) {}

	public void registerObserver(BeatObserver o) {
		heart.registerObserver(o);
	}

	public void removeObserver(BeatObserver o) {
```

```
            heart.removeObserver(o);
        }

        public void registerObserver(BPMObserver o) {
            heart.registerObserver(o);
        }

        public void removeObserver(BPMObserver o) {
            heart.removeObserver(o);
        }
}
```

11. Design simple HR Application using Spring Framework

```
package springDemo;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class Testt {
public static void main(String[] args) {
   Resource resource=new ClassPathResource("Context.xml");
   BeanFactory factory=new XmlBeanFactory(resource);

   Student student=(Student)factory.getBean("studentbean");
   student.displayInfo();
}
}
package springDemo;


public class Student {
private String name;

public String getName() {
  return name;
}
```

```
public void setName(String name) {
  this.name = name;
}

public void displayInfo(){
  System.out.println("Hello: "+name);
}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
   xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="studentbean" class="springDemo.Student">
<property name="name" value="Name"></property>
</bean>
  </beans>
```