Assignment based on Software Architecture and Design Patterns (SADP)

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()

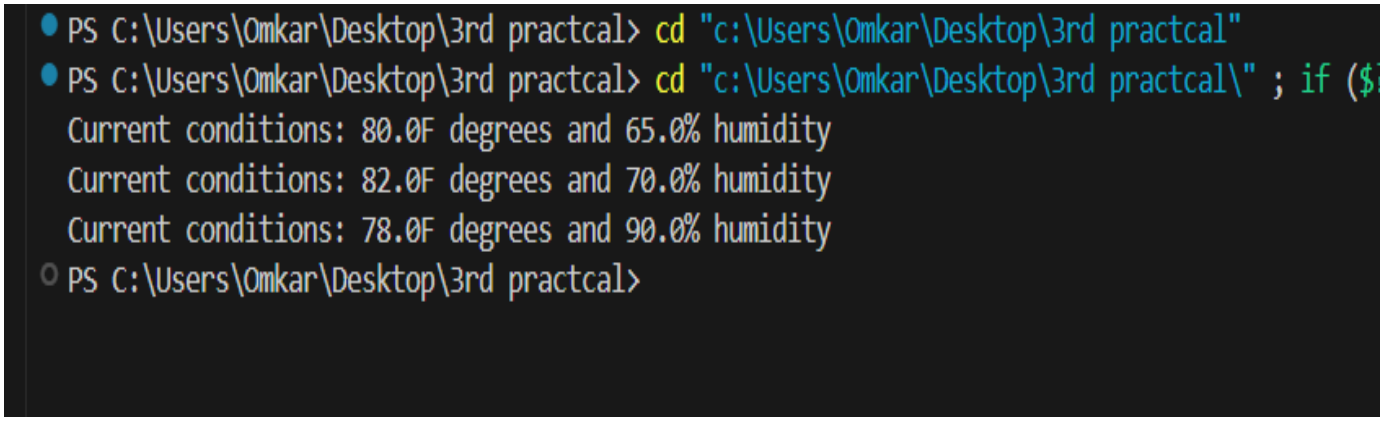
Code:-

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

@SuppressWarnings("deprecation")
class WeatherStation extends Observable {
    private float temperature;
    private float humidity;
 private float pressure;

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

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

```
    public float getTemperature() {
        return temperature;
    }

    public float getHumidity() {
        return humidity;
    }

    public float getPressure() {
        return pressure;
    }
}

@SuppressWarnings("deprecation")
class CurrentConditionsDisplay implements Observer, DisplayElement {
    private float temperature;
    private float humidity;
    public CurrentConditionsDisplay(Observable observable) {
        observable.addObserver(this);
    }

    public void update(Observable observable, Object arg) {
        if (observable instanceof WeatherStation) {
            WeatherStation weatherStation = (WeatherStation) observable;
            this.temperature = weatherStation.getTemperature();
            this.humidity = weatherStation.getHumidity();
            display();
        }
    }

    public void display() {
```

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

interface DisplayElement {
    void display();
}

public class WeatherStationTest {
    public static void main(String[] args) {
        WeatherStation weatherStation = new WeatherStation();
        new CurrentConditionsDisplay(weatherStation);

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

Output:



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

Code:

LowerCaseReader.java:

```
import java.io.FilterReader;
import java.io.IOException;
import java.io.Reader;

class LowerCaseReader extends FilterReader {
    protected LowerCaseReader(Reader in) {
        super(in);
    }

    @Override
    public int read() throws IOException {
        int c = super.read();
        if (c != -1) {
            c = Character.toLowerCase((char) c);
        }
        return c;
    }

    @Override
    public int read(char[] cbuf) throws IOException {
        int bytesRead = super.read(cbuf);
        for (int i = 0; i < bytesRead; i++) {
            cbuf[i] = Character.toLowerCase(cbuf[i]);
        }
        return bytesRead;
    }
```

```
    @Override
    public int read(char[] cbuf, int off, int len) throws IOException {
        int bytesRead = super.read(cbuf, off, len);
        for (int i = off; i < off + bytesRead; i++) {
            cbuf[i] = Character.toLowerCase(cbuf[i]);
        }
        return bytesRead;
    }
}
```

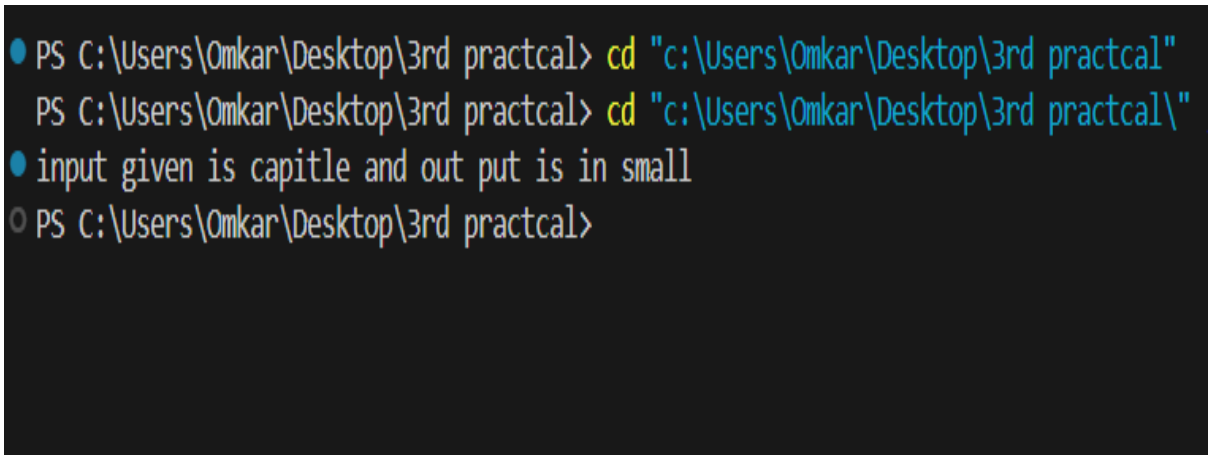
Main.java

Code:

```
import java.io.FilterReader;
import java.io.IOException;
import java.io.Reader;

class LowerCaseReader extends FilterReader {
    protected LowerCaseReader(Reader in) {
        super(in);
    }

    @Override
    public int read() throws IOException {
        int c = super.read();
        if (c != -1) {
            c = Character.toLowerCase((char) c);
        }
        return c;
    }
```

```
    @Override
    public int read(char[] cbuf) throws IOException {
        int bytesRead = super.read(cbuf);
        for (int i = 0; i < bytesRead; i++) {
            cbuf[i] = Character.toLowerCase(cbuf[i]);
        }
        return bytesRead;
    }

    @Override
    public int read(char[] cbuf, int off, int len) throws IOException {
        int bytesRead = super.read(cbuf, off, len);
        for (int i = off; i < off + bytesRead; i++) {
            cbuf[i] = Character.toLowerCase(cbuf[i]);
        }
        return bytesRead;
    }
}
```

Output:



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

Code:

```
// Abstract Product: Pizza
abstract class Pizza {
    abstract void prepare();
    abstract void bake();
    abstract void cut();
    abstract void box();
}

// Concrete Products: Different types of Pizzas
class CheesePizza extends Pizza {
    @Override
    void prepare() {
        System.out.println("Preparing Cheese Pizza...");
    }

    @Override
    void bake() {
        System.out.println("Baking Cheese Pizza...");
    }

    @Override
    void cut() {
        System.out.println("Cutting Cheese Pizza...");
    }
```

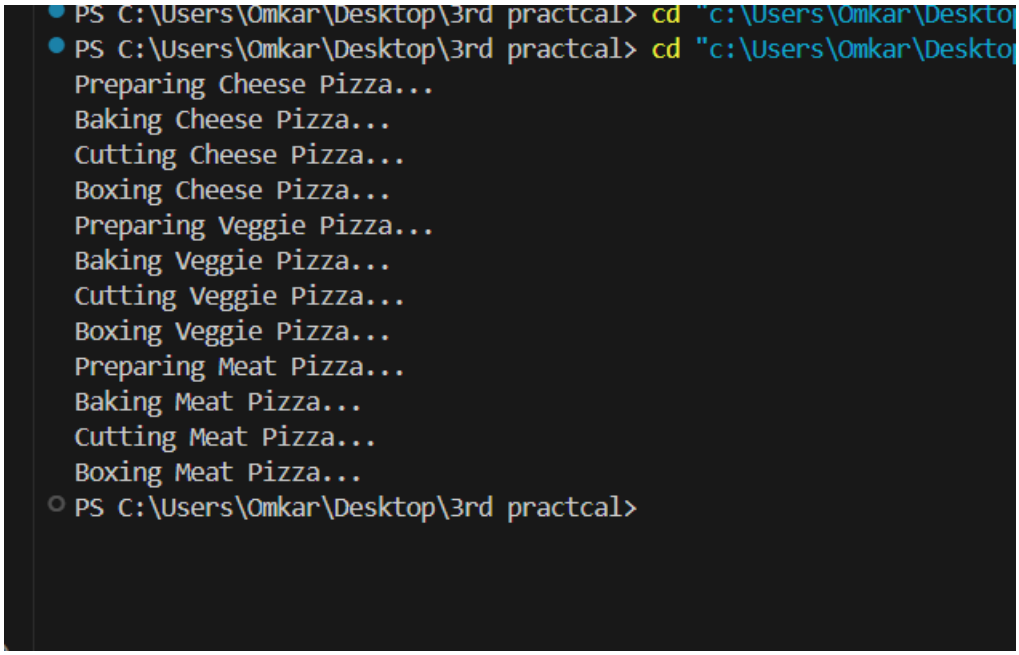
```
    @Override
    void box() {
        System.out.println("Boxing Cheese Pizza...");
    }
}

class VeggiePizza extends Pizza {
    @Override
    void prepare() {
        System.out.println("Preparing Veggie Pizza...");
    }

    @Override
    void bake() {
        System.out.println("Baking Veggie Pizza...");
    }

    @Override
    void cut() {
        System.out.println("Cutting Veggie Pizza...");
    }

    @Override
    void box() {
        System.out.println("Boxing Veggie Pizza...");
    }
}
```

```
class MeatPizza extends Pizza {
    @Override
    void prepare() {
        System.out.println("Preparing Meat Pizza...");
    }

    @Override
    void bake() {
        System.out.println("Baking Meat Pizza...");
    }

    @Override
    void cut() {
        System.out.println("Cutting Meat Pizza...");
    }

    @Override
    void box() {
        System.out.println("Boxing Meat Pizza...");
    }
}

// Abstract Factory: PizzaFactory
abstract class PizzaFactory {
    abstract Pizza createPizza();

    void orderPizza(Pizza pizza) {
        pizza.prepare();
        pizza.bake();
```

```
        pizza.cut();
        pizza.box();
    }
}

// Concrete Factories: Different types of Pizza Factories
class CheesePizzaFactory extends PizzaFactory {
    @Override
    Pizza createPizza() {
        return new CheesePizza();
    }
}

class VeggiePizzaFactory extends PizzaFactory {
    @Override
    Pizza createPizza() {
        return new VeggiePizza();
    }
}

class MeatPizzaFactory extends PizzaFactory {
    @Override
    Pizza createPizza() {
        return new MeatPizza();
    }
}

// Client code
public class PizzaStore {
```

```
    public static void main(String[] args) {
        PizzaFactory factory = new CheesePizzaFactory();
        Pizza pizza = factory.createPizza();
        factory.orderPizza(pizza);

        factory = new VeggiePizzaFactory();
        pizza = factory.createPizza();
        factory.orderPizza(pizza);

        factory = new MeatPizzaFactory();
        pizza = factory.createPizza();
        factory.orderPizza(pizza);
    }
}
```

Output:



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

Code:

Singleton.java

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

    private Singleton() {}

    public static Singleton getInstanceSync() {
        return getInstance();
    }

    public void doSomething() {
        System.out.println("Doing something...");
    }
}
```

MainThread.java

```
public class MainThread {
    public static void main(String[] args) {
        Thread t1 = new Thread(new Runnable() {
            public void run() {
```

```
            Singleton singleton = Singleton.getInstanceSync();
            singleton.doSomething();
        }
    });

    Thread t2 = new Thread(new Runnable() {
        public void run() {
            Singleton singleton = Singleton.getInstanceSync();
            singleton.doSomething();
        }
    });

    t1.start();
    t2.start();
  }
}
```

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

1. Command.java

java

EditCopy code

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

2. Light.java

java

EditCopy code

```
public class Light {
  public void on() {
```

```
        System.out.println("The light is ON");    }

    public void off() {
       System.out.println("The light is OFF");
    }
}
```

3. LightOnCommand.java

java

EditCopy code

```
public class LightOnCommand implements Command {
    private Light light;

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

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

4. LightOffCommand.java

java

EditCopy code

```
public class LightOffCommand implements Command {
    private Light light;

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

```
    }

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

5. RemoteControl.java

java

EditCopy code

```
public class RemoteControl {
    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }

    public void pressButton() {
        command.execute();
    }
-}
```

6. Main.java

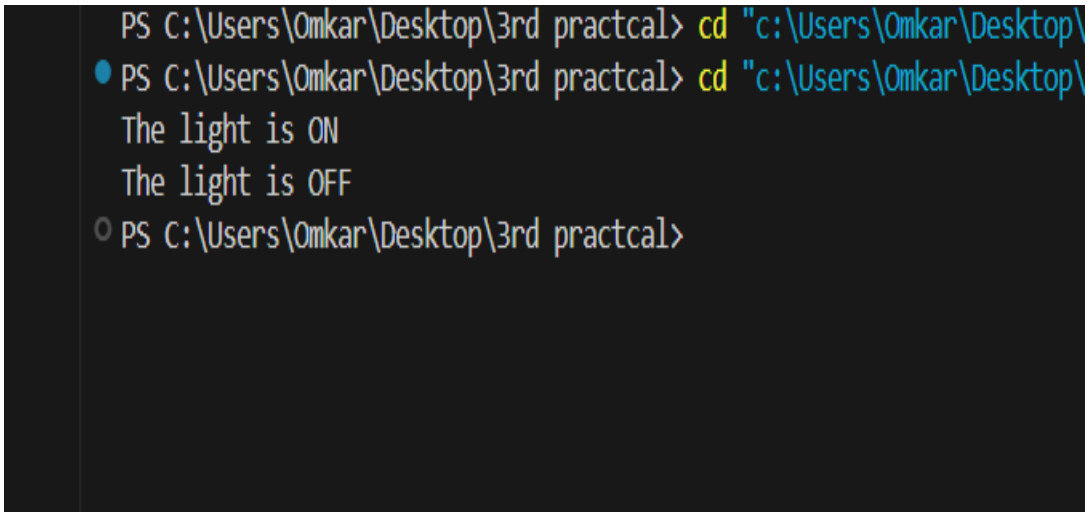
java

EditCopy code

```
public class Main {
    public static void main(String[] args) {
        Light light = new Light();
        Command lightOn = new LightOnCommand(light);
        Command lightOff = new LightOffCommand(light);
```

```
        RemoteControl remote = new RemoteControl();

        // Turn on the light
        remote.setCommand(lightOn);
        remote.pressButton();

        // Turn off the light
        remote.setCommand(lightOff);
        remote.pressButton();
    }
}
```

Output:



6. Write a Java Program to implement undo command to test Ceilingfan.

Code:

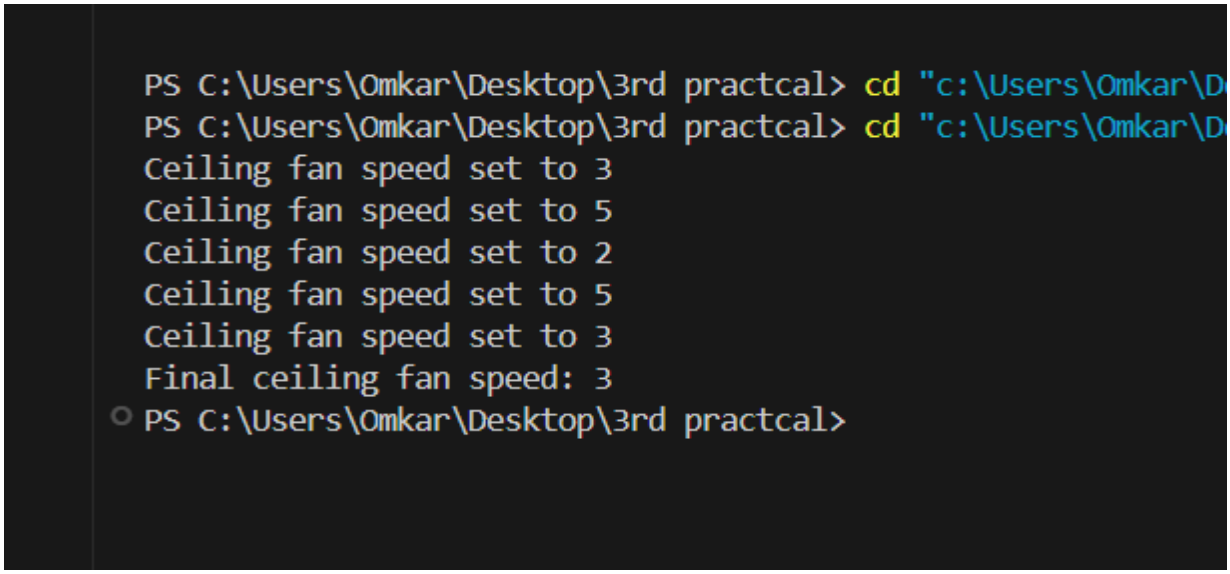
```
import java.util.Stack;

// Command interface
interface Command {
    void execute();
    void undo();
}

// CeilingFan class
class CeilingFan {
    private int speed;

    public void setSpeed(int speed) {
        this.speed = speed;
        System.out.println("Ceiling fan speed set to " + speed);
    }

    public int getSpeed() {
        return speed;
    }
}

// CeilingFanCommand class
class CeilingFanCommand implements Command {
    private CeilingFan ceilingFan;
    private int previousSpeed;
```

```
    private int newSpeed;

    public CeilingFanCommand(CeilingFan ceilingFan, int newSpeed) {
        this.ceilingFan = ceilingFan;
        this.newSpeed = newSpeed;
    }

    @Override
    public void execute() {
        previousSpeed = ceilingFan.getSpeed();
        ceilingFan.setSpeed(newSpeed);
    }

    @Override
    public void undo() {
        ceilingFan.setSpeed(previousSpeed);
    }
}

// UndoManager class
class UndoManager {
    private Stack<Command> commandStack = new Stack<>();

    public void executeCommand(Command command) {
        command.execute();
        commandStack.push(command);
    }

    public void undoCommand() {
```

```
        if (!commandStack.isEmpty()) {
            Command command = commandStack.pop();
            command.undo();
        }
    }
}

// Main class
public class CeilingFanUndoTest {
    public static void main(String[] args) {
        CeilingFan ceilingFan = new CeilingFan();
        UndoManager undoManager = new UndoManager();

        // Create and execute commands
        Command command1 = new CeilingFanCommand(ceilingFan, 3);
        undoManager.executeCommand(command1);

        Command command2 = new CeilingFanCommand(ceilingFan, 5);
        undoManager.executeCommand(command2);

        Command command3 = new CeilingFanCommand(ceilingFan, 2);
        undoManager.executeCommand(command3);

        // Undo commands
        undoManager.undoCommand();
        undoManager.undoCommand();

        // Print final ceiling fan speed
        System.out.println("Final ceiling fan speed: " + ceilingFan.getSpeed());
```

```
    }
}
```

Output



7. Write a Java Program to implement Iterator Pattern forDesigning Menu like

Breakfast, Lunch or DinnerMenu

Code:

MenuInterface.java

java

EditCopy code

```
import java.util.Iterator;
public interface MenuInterface {
    Iterator<MenuItem> createIterator();
}
```

MenuItem.java

java

EditCopy code

```
public class MenuItem {
    private String name;

    public MenuItem(String name) {
        this.name = name;
    }

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

Iterator.java

java

EditCopy code

```
import java.util.Iterator;

public interface Iterator {
    boolean hasNext();
    MenuItem next();
}
```

BreakfastMenu.java

java

EditCopy code

```
import java.util.ArrayList;

public class BreakfastMenu implements MenuInterface {
    private ArrayList<MenuItem> menuItems;

    public BreakfastMenu() {
```

```
        menuItems = new ArrayList<>();
       addItem("Eggs Benedict");
        addItem("Pancakes");
        addItem("Waffles");
    }

    public void addItem(String name) {
        menuItems.add(new MenuItem(name));
    }

    public Iterator<MenuItem> createIterator() {
        return new BreakfastMenuIterator(menuItems);
    }
}
```

LunchMenu.java

java

EditCopy code

```
import java.util.ArrayList;

public class LunchMenu implements MenuInterface {
    private ArrayList<MenuItem> menuItems;

    public LunchMenu() {
        menuItems = new ArrayList<>();
        addItem("Grilled Chicken");
        addItem("Veggie Burger");
        addItem("Soup of the Day");
    }
```

```
    public void addItem(String name) {
        menuItems.add(new MenuItem(name));
    }

    public Iterator<MenuItem> createIterator() {
        return new LunchMenuIterator(menuItems);
    }
}
```

DinnerMenu.java

java

EditCopy code

```
import java.util.ArrayList;

public class DinnerMenu implements MenuInterface {
    private ArrayList<MenuItem> menuItems;

    public DinnerMenu() {
        menuItems = new ArrayList<>();
        addItem("Steak");
        addItem("Shrimp Scampi");
        addItem("Vegetarian Lasagna");
    }

    public void addItem(String name) {
        menuItems.add(new MenuItem(name));
    }

    public Iterator<MenuItem> createIterator() {
        return new DinnerMenuIterator(menuItems);
```

```
    }
}
```

BreakfastMenuIterator.java

java

EditCopy code

```
import java.util.Iterator;

public class BreakfastMenuIterator implements Iterator {
    private ArrayList<MenuItem> menuItems;
    private int position = 0;

   public BreakfastMenuIterator(ArrayList<MenuItem> menuItems) {
       this.menuItems = menuItems;
   }

    public boolean hasNext() {
        return position < menuItems.size();
   }

    public MenuItem next() {
        MenuItem menuItem = menuItems.get(position);
        position++;
        return menuItem;
    }
}
```

Output:

```
+--------------------------------------+
|              MENU                    |
+--------------------------------------+

 BREAKFAST:
----------------------------------------
Pancakes              $5.99  Fluffy pancakes with syrup
Omelette              $7.99  Eggs, cheese, and veggies omelette
French Toast          $6.99  French toast with butter and honey
----------------------------------------

 LUNCH:
----------------------------------------
Burger                $8.99  Beef burger with cheese
Salad                 $6.99  Green salad with vinaigrette
Pasta                 $10.99 Pasta with Alfredo sauce
----------------------------------------

 DINNER:
----------------------------------------
Steak                 $19.99 Grilled steak with potatoes
Soup                  $7.99  Tomato soup with basil
Pizza                 $12.99 Thin-crust margherita pizza
```

IOT