Objective
- Being able to choose one (or a few suitable design) pattern(s) based on a given problem description.

PROBLEM DESCRIPTION
We are writing the software for a new electronic hand dryer. The dryer has a light sensor which can determine whenever someone’s hands are placed under it. This triggers the dryer to start blowing hot air. The dryer can also be triggered via a push button on the unit. Once the unit starts blowing, we do not want additional pushes of the button or triggers of the light sensor to extend the air blowing cycle.

TASKS
Select the most appropriate design pattern to use to address the problem and show how it is applied. In particular, shown an appropriate class diagram and enough code fragments to illustrate your use of the pattern to solve the problem.

ACTIVITIES
A. Do a brief analysis to identify the domain concepts (objects) and their relationships of the above problem.
B. Map the domain model into design model by adding attributes and methods to each classes.
C. Identify special conditions and requirements of the system under design and discuss which of the known design patterns can be applied. A few features that you will find helpful in making the design decision are:
   1. The dryer has on/off states
   2. Sensor and push button also have two states
   3. light sensor and push button have different interfaces
SOLUTION:

Code Fragment:

```java
import java.util.Observer;
import java.util.Observable;
import java.swing.Timer;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

class Dryer implements Observer, ActionListener{
    boolean blowing = false; // initial state is set to not blowing
    public void update(){
        if (!blowing){
            blowing = true;
            Timer tm = new Timer(6000, this);
            tm.start();
        } else {
            System.out.println("I am blowing.");
        }
    }
    public void ActionPerformed(){
        blowing = false; // has been blowing 1 minute
        System.out.println("I stops.");
    }
}

class Sensor extends Observable{
    public void handsUnder(){
        notify();
    }
}

class Button extends Observable{
}
```

Diagram:

[Diagram showing class relationships and interactions]

public void push()
    {
    notify();
    }

public class DryerTest{
    Button bt = new Button();
    Sensor sr = new Sensor();

    Dryer dryer = new Dryer();
    bt.addObserver(dryer);
    sr.addObserver(dryer);
    sr.handsUnder();
    bt.push();
}