Worksheet 6: Dependency Injection - Refactoring

a) Refactored SecuritySystem class

```
public class SecuritySystem implements SensorObserver
{
    private MotionSensor motionSens;
    private HeatSensor heatSens;
    private Alarm alarm;
    EmailSystem emailSys;
    private boolean armed;

    public SecuritySystem(MotionSensor motionSens, HeatSensor heatSens,
                          Alarm alarm, EmailSystem emailSys)
    {
        this.motionSens = motionSens;
        this.heatSens = heatSens;
        this.alarm = alarm;
        this.emailSys = emailSys;
        armed = false;
    }

    public void setupObservers()
    {
        motionSens.addSensorObserver(this);
        heatSens.addSensorObserver(this);
    }

    public void setArmed(boolean newArmed)
    {
        arm = newArmed;
        emailSys.sendMessage("Armed: " + newArmed);
    }

    @Override
    public void sensorDetection(Sensor s)
    {
        if(armed)
        {
            alarm.ring();
            emailSys.sendMessage("Sensor detection for " + s.toString());
        }
    }
}
```

b) Injector code

```
SecuritySystem secSys;
Hardware hw;
```

```
SensorBundle sens;
MotionSensor motionSens;
HeatSensor heatSens;
Alarm alarm;
EmailSystem emailSys;

//Assuming objects are created with empty constructors
hw = new Hardware();
sens = hw.getSensors();
motionSens = sens.getMotionSensor();
heatSens = sens.getHeatSensor();
alarm = new Alarm();
emailSys = new EmailSystem(); //Refactored to use non-static methods

secSys = new SecuritySystem(motionSens, heatSens, alarm, emailSys);
```