SE2832 Lab 7: A State Driven Security Light

Due: April 30, 2014 23:59

1 Introduction

Thusfar in lab, you have done adhoc testing in which you were given a series of use case scenarios and asked to test a software package based on those scenarios. You have also have used boundary value analysis and equivalence partitioning, as well as mock objects. Now it is time to move into a different realm of test design, namely test design from a formal specification.

A finite-state machine is a commonly used technique to design digital logic and computer software. It is a behavior model composed of a finite number of states, transitions between those states, and actions which occur upon entry and exit from states.

In this lab, you will be analyzing a formal state machine and developing test cases to ensure that the implementation of that state machine is correct.

2 Lab Objectives

- Analyze and construct test cases from a state diagram.
- Perform state testing on an existing class, noting any found defects.
- Use Mock objects and Mock Object tools to verify correct operation of an existing software package.

3 Domain background

The system you are to verify represents a security light. The security light has a light sensor which detects if it is night or day. During the day, the motion sensor is not active and the light will not turn on. However, at night, the system will look for motion, and if motion is found, the lamp will be turned on for 30 seconds. If at night the security alarm is tripped, the light will flash on and off until it is cleared by the user. The system also has a manual override switch which will turn the lamp on at any time.

A UI for this program has been developed for prototyping purposes, as is shown in Figure 1. It allows the user to simulate inputs and monitor the behavior.

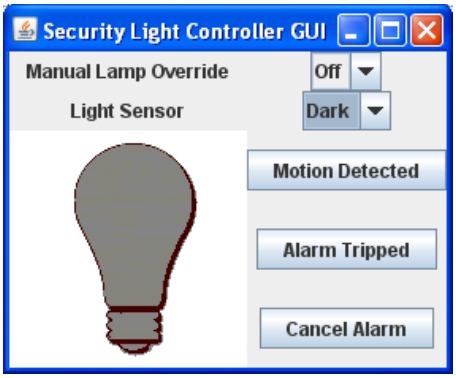

Figure 1 Demonstration UI.

Formally, the behavior for this system is specified using a UML state chart. This is shown in Figure 2. Note specifically the entry and exit actions for each of the states.

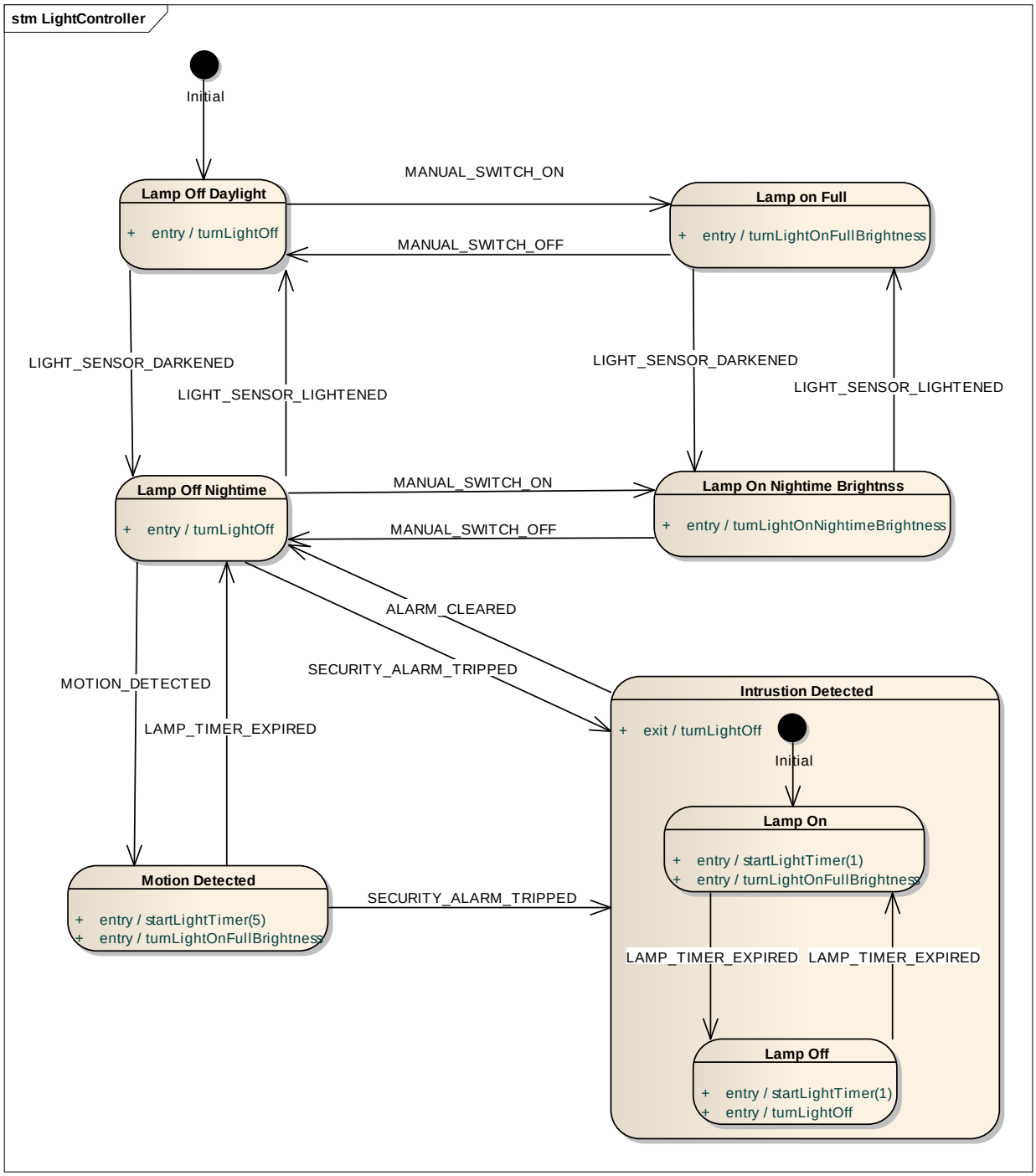

Figure 2 UML state diagram for program behavior.

The system itself has been implemented as is shown in the class diagram of Figure 3.

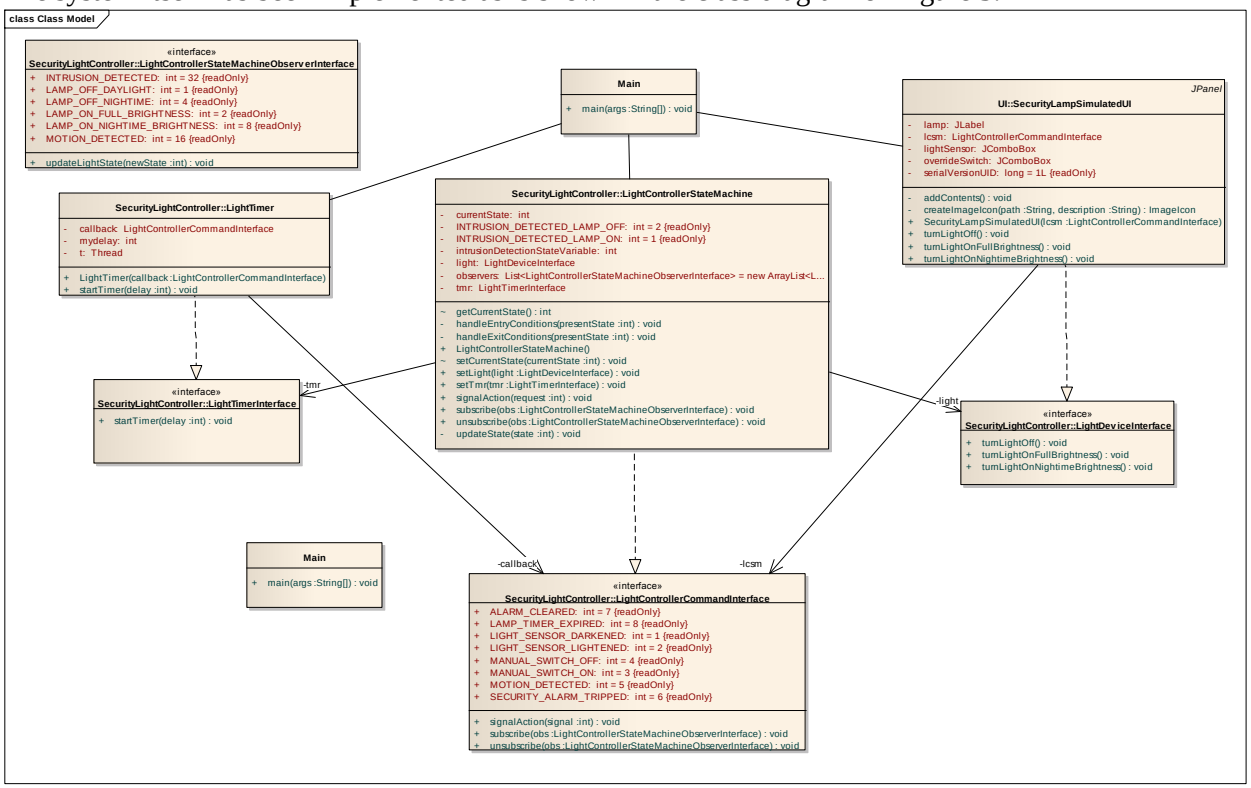

Figure 3 UML class diagram for the system.

Using JUnit / TestNG and the Mock object tool, and the behavioral specification from the state machine, you are to develop a set of test cases to verify that the LightControllerStateMachine functions properly. (Hint: You may want to draw sequence diagrams from the state chart to determine the correct interactions and message sequences between objects.)

Once you have created the test cases, you are to verify the behavior of the LightControllerStateMachine. This involves running your tests on the program and correcting any problems you encounter. If there are any bugs, fix them in the source code.

4 Deliverables / Submission

Now that you have completed your lab assignment, submit the following lab report detailing your experiences. The lab report should be submitted electronically through the course upload page.

1. Introduction
   a. What are you trying to accomplish with this lab? This section shall be written IN YOUR OWN WORDS. DO NOT copy directly from the assignment.
2. Strategy
   a. How did you go about determining your test cases?
   b. How did you organize your test cases?
   c. Why did you choose the test cases and how did you approach organizing them in a logical fashion.
3. Test Cases

   a. What are the test cases which you developed? Be certain to show the initial conditions (or initial states), the transitions, and the expected values.
4. Bugs
   a. List, in a tabular format, the bugs which you found during testing. Where were they found in the source code and what was wrong?
5. Things gone right / Things gone wrong
   a. This section shall discuss the things which went correctly with this experiment as well as the things which posed problems during this lab.
6. Conclusions
   a. What have you learned with this experience?

In addition to this, you should submit a zip file of your test cases and corrected source code through the website.

If you have any questions, consult your instructor.

Preliminary Grading Rubric

	Weight Factor	Rubric Score	
Debugged Source Code	1	4	- **Changes only were made to areas of the code which had bugs** - **Changes indicated with programmer initials and contained a comment indicating why the change was made.** - **Debugged source code passes "Golden" test suite.** - **All bugs effectively removed from submitted code**
Test Cases	1	5	**Test Cases Created** - **Test cases fully test all state transitions within the model.** - **Test cases verify complex sequences as well as simple sequences.** - **Test cases have appropriate coverage.**
	2	4	**JUnit / TestNG Documentation** - **Test cases contain JavaDoc header describing the tests and purposes** - **Test cases are appropriately commented.** - **Test cases developed in a logical manner** - **JUnit / TestNG code matches the designed test cases from the test case section.**
	0.5	5	**Test Case Compilation** - **Test cases compile without warning** - **Test cases properly formatted and using proper Java naming and test conventions** - **Test case definitions use source code constants for values when applicable instead of magic constants.** - **Test cases conform to JUnit / TestNG requirements**
Submission	1	4	**Introduction** - **Introduction fully describes what was intended for the lab** - **Introduction written in words separate from the lab assignment** - **Introduction written in multiple sentences**
	1	4	**Testing Strategy** - **Strategy for approaching test design described** - **Explanation of stock values used for testing provided** - **Multiple, complete sentences provided.**
	1	5	**Bug Log** - **Bug log details the bugs that were uncovered** - **Bug log details location of bugs in source code** - **Bug log details the fixes applied to the source code**
	1	4	**Reflection: Things gone right and things gone wrong** - **Things which went right with the lab fully described.** - **Things which went wrong with the lab fully described.**
	2	4	**Conclusions** - **What was learned from the lab described** - **Written in multiple, grammatically correct sentences**

5 JavaDoc for code

5.1 Interface LightControllerCommandInterface

All Known Implementing Classes:

LightControllerStateMachine

`public interface` **`LightControllerCommandInterface`**

This interface defines the light controller interface. Included are the signals that can be sent into the controller (based on the detection of changes in the system) as well as the methods which can be invoked.

Author:

schilling

Field Summary

`static int`	**`ALARM_CLEARED`** This signal indicates that someone has cleared the active alarm.
`static int`	**`LAMP_TIMER_EXPIRED`** This signal indicates that a timer has expired.
`static int`	**`LIGHT_SENSOR_DARKENED`** This signal indicates that the light sensor has been darkened, indicating diminished ambient light and the approach of nightfall.
`static int`	**`LIGHT_SENSOR_LIGHTENED`** This signal indicates that the light sensor has been lit up, indicating the sun is up and daylight is here.
`static int`	**`MANUAL_SWITCH_OFF`** This signal indicates that a manual override switch has been placed into the off position.
`static int`	**`MANUAL_SWITCH_ON`** This signal indicates that a manual override switch has been placed into the on position.
`static int`	**`MOTION_DETECTED`** This signal indicates that motion has been detected in the area.
`static int`	**`SECURITY_ALARM_TRIPPED`** This signal indicates that an intrusion has been detected.

Method Summary

void	**signalAction**(int signal) This method provides a mechanism for a signal to be received by the light controller.
void	**subscribe**(LightControllerStateMachineObserverInterface obs) This method will allow an external observer to subscribe to state machine, receiving updates when states change.
void	**unsubscribe**(LightControllerStateMachineObserverInterface obs) This method will allow an external observer to unsubscribe to state machine, receiving updates when states change.

Field Detail

5.1.1 LIGHT_SENSOR_DARKENED

`static final int` **LIGHT_SENSOR_DARKENED**

This signal indicates that the light sensor has been darkened, indicating diminished ambient light and the approach of nightfall.

5.1.2 LIGHT_SENSOR_LIGHTENED

`static final int` **LIGHT_SENSOR_LIGHTENED**

This signal indicates that the light sensor has been lit up, indicating the sun is up and daylight is here.

5.1.3 MANUAL_SWITCH_ON

`static final int` **MANUAL_SWITCH_ON**

This signal indicates that a manual override switch has been placed into the on position.

5.1.4 MANUAL_SWITCH_OFF

`static final int` **MANUAL_SWITCH_OFF**

This signal indicates that a manual override switch has been placed into the off position.

5.1.5 MOTION_DETECTED

`static final int` **MOTION_DETECTED**

This signal indicates that motion has been detected in the area. Typically, this indicates that a motion sensor has gone off.

5.1.6 SECURITY_ALARM_TRIPPED

`static final int` **SECURITY_ALARM_TRIPPED**

This signal indicates that an intrusion has been detected. Perhaps a window has been opened or some other detection of intrusion has occurred.

5.1.7 ALARM_CLEARED

`static final int` **`ALARM_CLEARED`**

This signal indicates that someone has cleared the active alarm. This is typically done by resetting the alarm.

5.1.8 LAMP_TIMER_EXPIRED

`static final int` **`LAMP_TIMER_EXPIRED`**

This signal indicates that a timer has expired.

Method Detail

5.1.9 signalAction

`void` **`signalAction`**`(int signal)`

This method provides a mechanism for a signal to be received by the light controller. The signal will be one of the defined values provided previously.

Parameters:

`signal` - This is the signal that is being received. It can be one of LIGHT_SENSOR_DARKENED, LIGHT_SENSOR_LIGHTENED, MANUAL_SWITCH_ON, MANUAL_SWITCH_OFF, MOTION_DETECTED, SECURITY_ALARM_TRIPPED, ALARM_CLEARED, or LAMP_TIMER_EXPIRED. Any other values are to be ignored.

5.1.10 subscribe

`void` **`subscribe`**`(LightControllerStateMachineObserverInterface obs)`

This method will allow an external observer to subscribe to state machine, receiving updates when states change.

Parameters:

`obs` - This is the observer interface that is to be subscribed.

5.1.11 unsubscribe

`void` **`unsubscribe`**`(LightControllerStateMachineObserverInterface obs)`

This method will allow an external observer to unsubscribe to state machine, receiving updates when states change.

Parameters:

`obs` - This is the observer interface that is to be unsubscribed.

5.2 Interface LightControllerStateMachineObserverInterface

`public interface` **`LightControllerStateMachineObserverInterface`**

This interface defines an interface which a class must implement if it desires to observe the lamp state.

Author:
schilling

Field Summary

static int	**INTRUSION_DETECTED**
static int	**LAMP_OFF_DAYLIGHT**
static int	**LAMP_OFF_NIGHTIME**
static int	**LAMP_ON_FULL_BRIGHTNESS**
static int	**LAMP_ON_NIGHTIME_BRIGHTNESS**
static int	**MOTION_DETECTED**

Method Summary

void	**updateLightState**(int newState) This method will update the state of the light, passing in one of the the parameters representing the current state.

Field Detail

5.2.1 LAMP_OFF_DAYLIGHT

static final int **LAMP_OFF_DAYLIGHT**

5.2.2 LAMP_ON_FULL_BRIGHTNESS

static final int **LAMP_ON_FULL_BRIGHTNESS**

5.2.3 LAMP_OFF_NIGHTIME

static final int **LAMP_OFF_NIGHTIME**

5.2.4 LAMP_ON_NIGHTIME_BRIGHTNESS

```
static final int LAMP_ON_NIGHTIME_BRIGHTNESS
```

5.2.5 MOTION_DETECTED

```
static final int MOTION_DETECTED
```

5.2.6 INTRUSION_DETECTED

```
static final int INTRUSION_DETECTED
```

Method Detail

5.2.7 updateLightState

```
void updateLightState(int newState)
```

This method will update the state of the light, passing in one of the the parameters representing the current state.

Parameters:

`newState` - The new state, one of LAMP_OFF_DAYLIGHT , LAMP_ON_FULL_BRIGHTNESS, LAMP_OFF_NIGHTIME, LAMP_ON_NIGHTIME_BRIGHTNESS, MOTION_DETECTED, INTRUSION_DETECTED. Any other values are to be ignored.

5.3 Interface LightDeviceInterface

All Known Implementing Classes:

SecurityLampSimulatedUI

```
public interface LightDeviceInterface
```

This interface defines those things which must be present for a light. This includes the methods invoked from the state machine to control the light.

Author:

schilling

Method Summary

void	**turnLightOff**()
void	**turnLightOnFullBrightness**()

void	**turnLightOnNightimeBrightness**()

Method Detail

5.3.1 turnLightOff

void **turnLightOff**()

5.3.2 turnLightOnFullBrightness

void **turnLightOnFullBrightness**()

5.3.3 turnLightOnNightimeBrightness

void **turnLightOnNightimeBrightness**()

5.4 Interface LightTimerInterface

All Known Implementing Classes:

LightTimer

---

public interface **LightTimerInterface**

This interface defines that which is used for a light timer.

Author:

schilling

Method Summary

void	**startTimer**(int delay) This method will start the timer.

Method Detail

5.4.1 startTimer

void **startTimer**(int delay)

This method will start the timer.

Parameters:

`delay` - This is the delay, in seconds, for the given timer.

5.5 Class LightControllerStateMachine

`java.lang.Object`
`└ SecurityLightController.LightControllerStateMachine`

All Implemented Interfaces:

LightControllerCommandInterface

```
public class LightControllerStateMachine
extends java.lang.Object
implements LightControllerCommandInterface
```

This class implements a state machine that can be used to control a security light.

Author:

schilling

Field Summary

private int	**currentState** This variable holds the current state for the state machine.
private static int	**INTRUSION_DETECTED_LAMP_OFF** This value is used for keeping track of the Intrusion detected substates.
private static int	**INTRUSION_DETECTED_LAMP_ON** This value is used for keeping track of the Intrusion detected substates.
private int	**intrusionDetectionStateVariable** This variable holds the substate for the intrusion detected state.
private LightDeviceInterface	**light** This variable holds the light which is to be controlled by this state machine.
private java.util.List<LightControllerStateMachineObserverInterface>	**obsservers** This variable holds a list of observers.
private LightTimerInterface	**tmr** This variable is the instance of the timer that is to be used for timed activities.

Fields inherited from interface SecurityLightController.LightControllerCommandInterface

ALARM_CLEARED, LAMP_TIMER_EXPIRED, LIGHT_SENSOR_DARKENED, LIGHT_SENSOR_LIGHTENED, MANUAL_SWITCH_OFF, MANUAL_SWITCH_ON, MOTION_DETECTED, SECURITY_ALARM_TRIPPED

Constructor Summary

LightControllerStateMachine()	This is the default constructor, which will instantiate a new instance of this class.

Method Summary

private void	**handleEntryConditions**(int presentState) This method will process entry conditions based upon entering a state.
private void	**handleExitConditions**(int presentState) This method will process exit conditions based upon leaving a state.
void	**setLight**(LightDeviceInterface light) This method will set the light that is to be controlled by this state machine.
void	**setTmr**(LightTimerInterface tmr) This method will set an instance of the timer that is to be used with this class.
void	**signalAction**(int request) This method provides a mechanism for a signal to be received by the light controller.
void	**subscribe**(LightControllerStateMachineObserverInterface obs) This method will allow an external observer to subscribe to state machine, receiving updates when states change.
void	**unsubscribe**(LightControllerStateMachineObserverInterface obs) This method will allow an external observer to unsubscribe to state machine, receiving updates when states change.
private void	**updateState**(int state) This method will be invoked to update the observers

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

5.5.1 observers

```
private java.util.List<LightControllerStateMachineObserverInterface>
observers
```

This variable holds a list of observers. Whenever a state changes, the observers are updated with the state change.

5.5.2 currentState

```
private int currentState
```

This variable holds the current state for the state machine.

5.5.3 light

```
private LightDeviceInterface light
```

This variable holds the light which is to be controlled by this state machine.

5.5.4 tmr

```
private LightTimerInterface tmr
```

This variable is the instance of the timer that is to be used for timed activities.

5.5.5 INTRUSION_DETECTED_LAMP_ON

```
private static final int INTRUSION_DETECTED_LAMP_ON
```

This value is used for keeping track of the Intrusion detected substates.

5.5.6 INTRUSION_DETECTED_LAMP_OFF

```
private static final int INTRUSION_DETECTED_LAMP_OFF
```

This value is used for keeping track of the Intrusion detected substates.

5.5.7 intrusionDetectionStateVariable

```
private int intrusionDetectionStateVariable
```

This variable holds the substate for the intrusion detected state.

Constructor Detail

5.5.8 LightControllerStateMachine

```
public LightControllerStateMachine()
```

This is the default constructor, which will instantiate a new instance of this class.

Method Detail

5.5.9 setLight

```
public void setLight(LightDeviceInterface light)
```

This method will set the light that is to be controlled by this state machine.

Parameters:

`light` - This is the instance of the light that is to be directly controlled.

5.5.10 setTmr

`public void` **`setTmr`**`(LightTimerInterface tmr)`

This method will set an instance of the timer that is to be used with this class.

Parameters:

`tmr` - This is the timer instance.

5.5.11 updateState

`private void` **`updateState`**`(int state)`

This method will be invoked to update the observers

Parameters:

`state` - This is the new state.

5.5.12 subscribe

`public void` **`subscribe`**`(LightControllerStateMachineObserverInterface obs)`

Description copied from interface: `LightControllerCommandInterface`

This method will allow an external observer to subscribe to state machine, receiving updates when states change.

Specified by:

`subscribe` in interface `LightControllerCommandInterface`

Parameters:

`obs` - This is the observer interface that is to be subscribed.

5.5.13 unsubscribe

`public void` **`unsubscribe`**`(LightControllerStateMachineObserverInterface obs)`

Description copied from interface: `LightControllerCommandInterface`

This method will allow an external observer to unsubscribe to state machine, receiving updates when states change.

Specified by:

`unsubscribe` in interface `LightControllerCommandInterface`

Parameters:

`obs` - This is the observer interface that is to be unsubscribed.

5.5.14 handleExitConditions

`private void` **`handleExitConditions`**`(int presentState)`

This method will process exit conditions based upon leaving a state.

Parameters:

`presentState` - This is the present state. It will be used to determine which exit actions to invoke.

5.5.15 handleEntryConditions

`private void` **`handleEntryConditions`**`(int presentState)`

This method will process entry conditions based upon entering a state.

Parameters:

`presentState` - This is the present state. It will be used to determine which exit actions to invoke.

5.5.16 signalAction

`public void` **`signalAction`**`(int request)`

Description copied from interface: **`LightControllerCommandInterface`**

This method provides a mechanism for a signal to be received by the light controller. The signal will be one of the defined values provided previously.

Specified by:

`signalAction` in interface `LightControllerCommandInterface`

Parameters:

`request` - This is the signal that is being received. It can be one of LIGHT_SENSOR_DARKENED, LIGHT_SENSOR_LIGHTENED, MANUAL_SWITCH_ON, MANUAL_SWITCH_OFF, MOTION_DETECTED, SECURITY_ALARM_TRIPPED, ALARM_CLEARED, or LAMP_TIMER_EXPIRED.

5.5.17 getCurrentState

`int getCurrentState()`

This method will return the current state of the Light controller. It is only intended to be used for testing. Thus, the protected attribute.

Returns:

This will return the current state of the light controller.

5.5.18 setCurrentState

`void` **`setCurrentState`**`(int currentState)`

This method will set the current state of the light controller. It is intended to only be used for testing, as it may or may not properly invoke any entry or exit conditions.

Parameters:

`currentState` - This is the desired state for the light controller.

5.6 Class LightTimer

`java.lang.Object`

└ **SecurityLightController.LightTimer**

All Implemented Interfaces:

LightTimerInterface

```
public class LightTimer
extends java.lang.Object
implements LightTimerInterface
```

This class implements a light timer. It is used to control the timeouts associated with a given light.

Author:

schilling

Field Summary

private LightControllerCommandInterface	**callback** This variable holds the callback which is invoked when the timer expires.
private int	**mydelay** This variable holds the delay in seconds until the timer expires.
private java.lang.Thread	**t** This variable holds the thread which is to tun for this timer.

Constructor Summary

LightTimer(LightControllerCommandInterface callback)

Method Summary

void	**startTimer**(int delay) This method will start the timer.

Field Detail

5.6.1 mydelay

`private int` **`mydelay`**

This variable holds the delay in seconds until the timer expires.

5.6.2 callback

`private` `LightControllerCommandInterface` **`callback`**

This variable holds the callback which is invoked when the timer expires.

5.6.3 t

`private java.lang.Thread` **`t`**

This variable holds the thread which is to run for this timer.

Constructor Detail

5.6.4 LightTimer

`public` **`LightTimer`**`(LightControllerCommandInterface callback)`

Parameters:

`callback` - This is the callback for when the timer expires.

Method Detail

5.6.5 startTimer

`public void` **`startTimer`**`(int delay)`

Description copied from interface: **`LightTimerInterface`**

This method will start the timer.

Specified by:

`startTimer` in interface `LightTimerInterface`

Parameters:

`delay` - This is the delay, in seconds, for the given timer.