Light Display Simulation

by

Hunter Holmes hholmes1

Report submitted on 25 October 2018

EE 333 – Engineering Programming Using Objects

Department of Electrical and Computer Engineering

The University of Alabama at Birmingham

Abstract

TBD**Table of Contents**

[Abstract i](#_Toc527993203)

[Introduction 1](#_Toc527993204)

[Project definition 1](#_Toc527993205)

[Constraints 1](#_Toc527993206)

[Goals 1](#_Toc527993207)

[Features (in MoSCoW list format) 2](#_Toc527993208)

[Applicable Standards 2](#_Toc527993209)

[Design 2](#_Toc527993210)

[Initial Design Process 2](#_Toc527993211)

[Appropriate for Object Oriented Approach 3](#_Toc527993212)

[Design Decisions 3](#_Toc527993213)

[Object Oriented Design 5](#_Toc527993214)

[Discovery and USE OF ONLINE INformation 8](#_Toc527993215)

[Debug 8](#_Toc527993216)

[Results 8](#_Toc527993217)

[Discussion 8](#_Toc527993218)

[Conclusions 8](#_Toc527993219)

[References 8](#_Toc527993220)

[Appendix 8](#_Toc527993221)

Introduction

The overall goal of this project is to improve the designer’s skills and understanding of object-oriented design and implementation. Some programming languages allow the designer to create objects in code that have definitions based on real-world objects. This type programming is called Object-Oriented. An example of an object-oriented programming language is Java which is used in this project. Although objects can be used to model a physical part of a problem they are also used for internal parts of the system, for example clocks which maintain synchronization or loggers which keep track of system activity.

Project definition

The program chosen for this project is a light control system. Hardware implementation is not included in this project, but lights which are a part of module will be connected to a controller which makes decisions regarding light behavior based on user input. Components of this system are to be designed and modeled using objects. A controller will send commands to an array of modules which will send commands the individual lights that are a part of that module. Each of these components shall be modeled as objects. The list that follow, detail the constraints, goals, and specific objectives of this project.

# Constraints

* CO-01: The developer shall program the solution in Java
* CO-02: The program shall be written using NetBeans
* CO-03: The modeling classes shall not contain I/O
* CO-04: The solution shall be built using Object-Oriented Design
* CO-05: Modules shall be connected in series

# Goals

* GO-01: The system should be scalable regarding number of lights connected
* GO-02: The system should work with light modules possessing a variety of features and architectures
* GO-03: The project should possess test code that demonstrates its capabilities

# Features (in MoSCoW list format)

The desired features include:

* Must-01: Have light modules synchronized with one another
* Must-02: Allow user to have some control of behavior
* Should-01: Support at least 10,000 lights total
* Should-02: Support 4 light modules per channel
* Should-03: Contain a variety of preset modes to send to light modules
* Should-04: Control lights’ brightness and color
* Should-05: Detect type and relative position of newly attached modules
* Should-06: Be user friendly
* Could-01: Be implemented with hardware
* Won’t-01: Include a GUI

# Applicable Standards

* STD-01: Documentation will follow Javadoc standards
* STD-02: Source code will follow standards set by the UAB Department of Electrical and Computer Engineering

Design

# Initial Design Process

Upon receiving project, the assignment document was reviewed carefully with notes being taken regarding which physical objects were being modeled, constraints, and clear goals. The system was considered as it would be expected to behave. A user may input a command to an individual light module or to the array of modules in its entirety. The system would then take this input and distribute commands to the proper modules which would distribute commands to its respective lights. This scenario sets up the basic structure for the overall operation of the system.

Observing that this system relies on commands being passed down from systems to subsystems, a few objects like controller, light module, and lights were readily apparent. Reviewing the document once more, all physical items which were being modeled were written down as potential objects to be designed in the program. The properties of each of these objects were studied and then written down as descriptions for each object. For example, a light object would have properties such as an ON/OFF setting, brightness value, and color value. These properties are important to the object-oriented design process as they allow for the creation of classes with the same properties that can be changed and observed.

With some potential objects defined, the classes necessary for writing the program began to take shape. Many classes were defined to be the same as the objects they modeled. For example, the class used for a module object would be a Module.java. Some other classes would also be needed to model some internal/non-physical components, such as a clock.

After defining the basic operation scenario as well as the prospective classes, interaction diagrams were created. These diagrams create visualization for queries and commands that are sent from one class to another in a system cycle for a particular scenario. There are many possible scenarios that could be used for interaction diagrams but only 2 diagrams that were most enlightening were used. A class diagram for the system was also created. This diagram demonstrates how classes are dependent on one another. This type of diagram is not dependent on a particular scenario so only one is needed.

# Appropriate for Object Oriented Approach

The light module controller project lends itself to an object-oriented approach. The system is comprised of interconnected components which relay information to one another. This fits very well with the concepts of commands and queries which are found in an object-oriented programming design. The problem also insists that some components should have standardized properties but should also be duplicated over and over, most notably the light modules and the lights themselves. A class called “module” could be created and then as many module objects as are necessary could be created out of the patterning class. This is just one example, but this technique will be used heavily in this project

# Design Decisions

AL-01 How should a message for a particular module identify itself?

AL-01A Each module asks the controller if the message belongs to it

AL-01B Sending the module’s ID along with the message

ALD-01 Adding information about the intended recipient would save a great deal of time if a message was for example for the last module in a series. The modules in line before it would simply ignore the message if it did not include their unique ID within the system. If a message is intended for all modules in a series, then the ID portion could be left out as all modules need to handle the message

AL-02 How should properties of newly connected module be established?

AL-02A User inputs module information

AL-02B Module identifies itself by its unique ID

ALD-02 For this problem option A is the most appropriate solution. A user can create a new module from a list of predetermined types. The user would then put this unique ID in the add method for the controller which would determine the properties of the module based on portions of the ID. If hardware implementation were used this feature may be withdrawn as the controller would simply poll the module to determine its ID. Since this implementation is completely virtual the user will have to input some identifying information about the module.

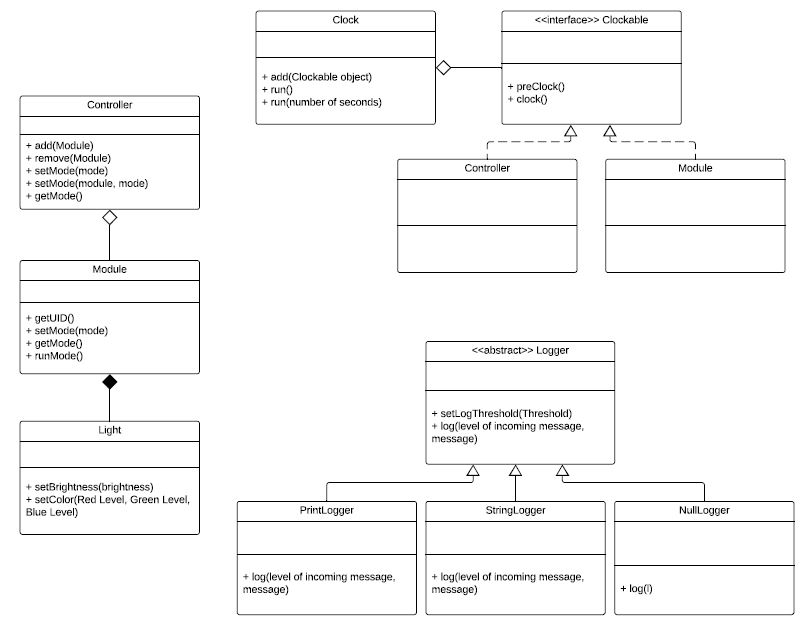
AL-03 How should timing synchronization be dealt with between modules?

AL-03A Use a clocking system

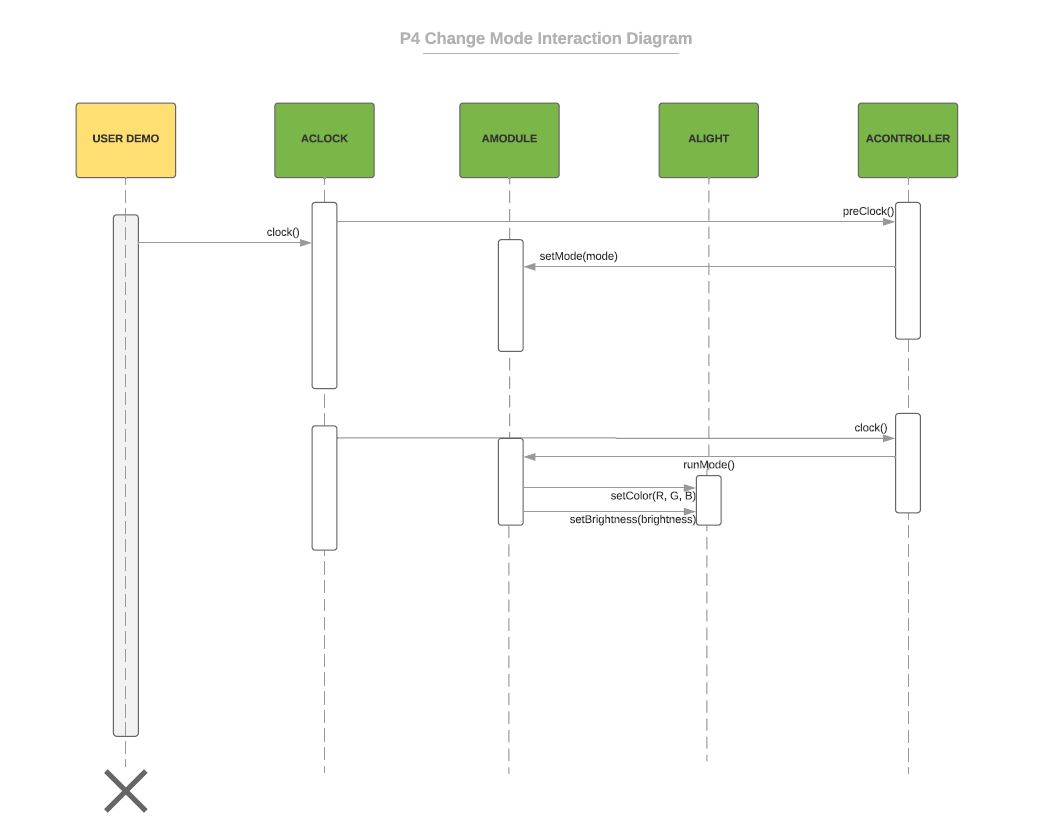
AL-03B Timing is ignored, simply send commands to modules when input is changed

ALD-03 Using a clocking system, option A, is the cleanest solution to this problem\. It adds extra work to the build and design of the system, but it deals with the issue of timing. Simply due to processing time some portions of the system may become out of time with the rest of the system. Having a shared clock between all modules would allow them all the chance to receive their next mode command in one cycle and then act on the next cycle. Since the clock command would be miniscule in size and processing time, the delay would be minimized.

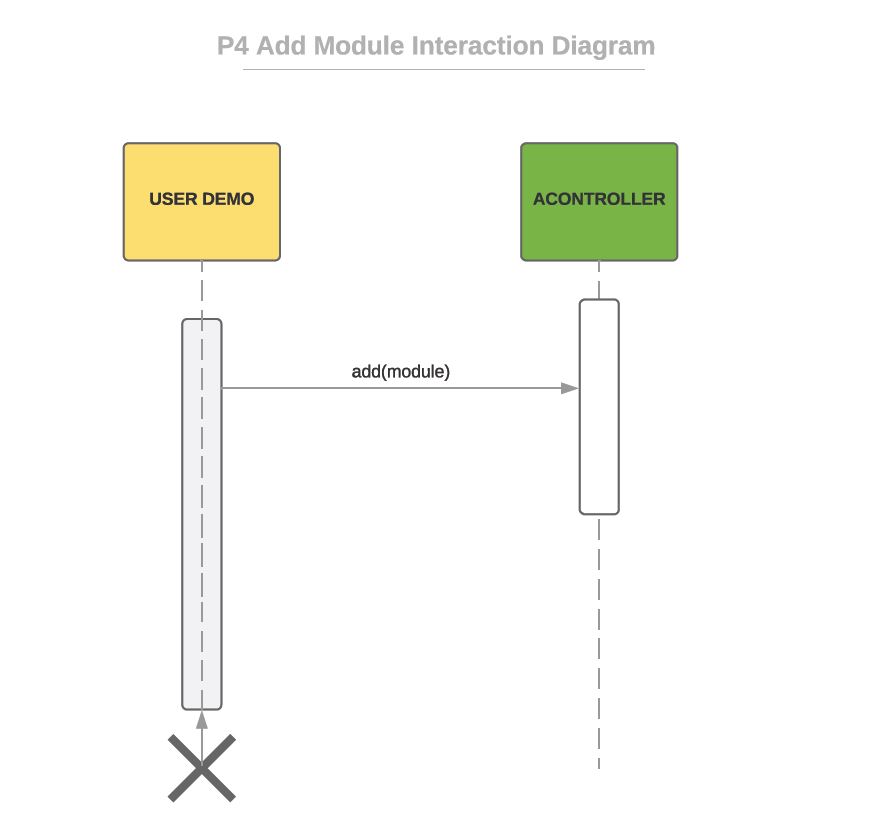
# Object Oriented Design



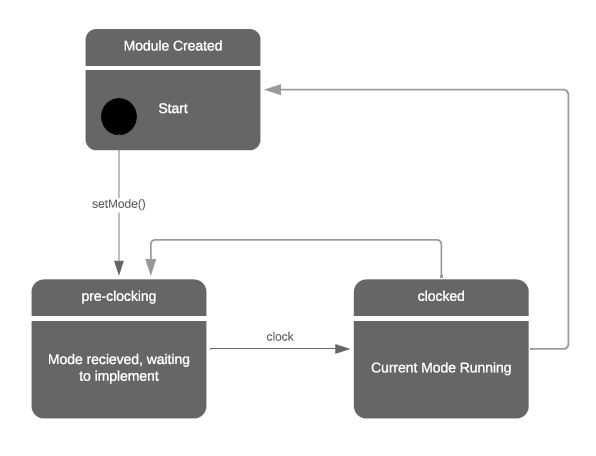
**Fig. 01 Class Diagram**



**Fig. 2 Mode Change Interaction Diagram**

****

**Fig. 3 Add Module Interaction Diagram**

****

**Fig. 4 Module State Diagram**

Discovery and USE OF ONLINE INformation

TBD

Debug

TBD

Results

TBD

Discussion

TBD

Conclusions

TBD

References

TBD

Appendix

TBD