Project 3

ECE 528: Application Software Design

Alan Palayil

Professor: Won-Jae Yi

Acknowledgement: I acknowledge all works including figures, codes and writings belong to me and/or persons who are referenced. I understand if any similarity in the code, comments, customized program behavior, report writings and/or figures are found, both the helper (original work) and the requestor (duplicated/modified work) will be called for academic disciplinary action.

Electronic Signature: Alan Palayil A20447935 (Due Date: 3/5/2023)

Table of Contents

Abstract:

This project focuses on utilizing the MQTT protocol to manage and monitor an IoT simulator. The Eclipse Mosquitto™ MQTT broker is pre-installed in the virtual machine to simplify the development process. The project requirements are described through user stories, and the design and implementation of classes are open-ended, allowing for flexibility in approach. A crucial part of the development process is following the red-green cycle by creating comprehensive unit tests to ensure that the code meets all requirements and functions as expected.

Introduction:

The MQTT protocol is widely used on the Internet of Things (IoT) industry to enable communication between devices and applications. In this project, we will be working with MQTT to control an IoT simulator and report its status. The Eclipse Mosquitto MQTT broker has been installed in the course virtual machine for our convenience. The project requirements are described through user stories, and we are free to choose our own class designs and implementations. Throughout the project, we should follow the red-green cycle to add unit tests to ensure the quality of our code. In this way, we can ensure that our implementation of the MQTT protocol meets the requirements of the project and performs as expected.

Background:

The project involves working with the MQTT protocol to control the IoT simulator and report its status. The Eclipse Mosquitto™ MQTT broker is installed in the course virtual machine for convenience. The project comprises several users stories:

- Plug On/Off Updates: This user story is about an end-user who wants to receive MQTT messages when a plug is turned on or off, so that they can monitor the plug on/off events using an MQTT client. The plug is identified by its name "plugName", and the topic for updates should be in the format of

"prefix/update/plugName/state". The message for this topic should be either "on" or "off". The configuration string "prefix" can have multiple "/" characters, such as "illinoistech/ece448/unit_test".

- Plug Power Updates: This user story requires the system to send MQTT messages containing the power consumption of a plug to an MQTT client. The topic for the message should be "prefix/update/plugName/power", where "prefix" is the configuration string and "plugName" is the name of the plug. The message should contain the power consumption in plain text.
- Toggle or Switch a Plug On/Off: This user story describes the requirement for the ability to send MQTT messages to control a plug's on/off state using an MQTT client. The topic should follow the format of prefix/action/plugName/actionString, where actionString can be one of three options: toggle, on, or off.

The topics and messages for each of these user stories are defined based on the plug name and configuration string. The project also includes testing procedures implemented in ece448.grading.GradeP3 to ensure that all user stories are covered.

Results and Discussion:

The project required us to create to files MQTTCommands and MQTTUpdates. During the implementation of the project, I had to modify two classes: Main and PlugSim. The MQTTCommands class has two methods:

- getTopic (): This method returns a properly formatted string containing the topic prefix.
- handleMessage (): This method processes MQTT messages based on the topic. If no plug is specified, it does nothing. Otherwise, it checks the action specified in the message. If the action is "on", it turns on the plug. If the action is "off", it turns off the plug. If the action is "toggle", it toggles the plug.

The MQTTUpdates class has two methods:

- getTopic (): This method returns a string with the topic prefix in the proper format.

- getMessage (): This method returns the message as a string that can be sent via MQTT and has the same content.

The Main class, I updated the constructor to include:

- A listener for incoming MQTT commands
- A listener for outgoing MQTT plug state messages.

PlugSim class I added the following:

- Modified to support the addition and handling Observers.

The following are the screenshots of the results I got during the project.

Screenshots of the Tests results:

The screenshot of the acceptance testing ‘Gradle’ results is added which was the base testing criteria for project 3.

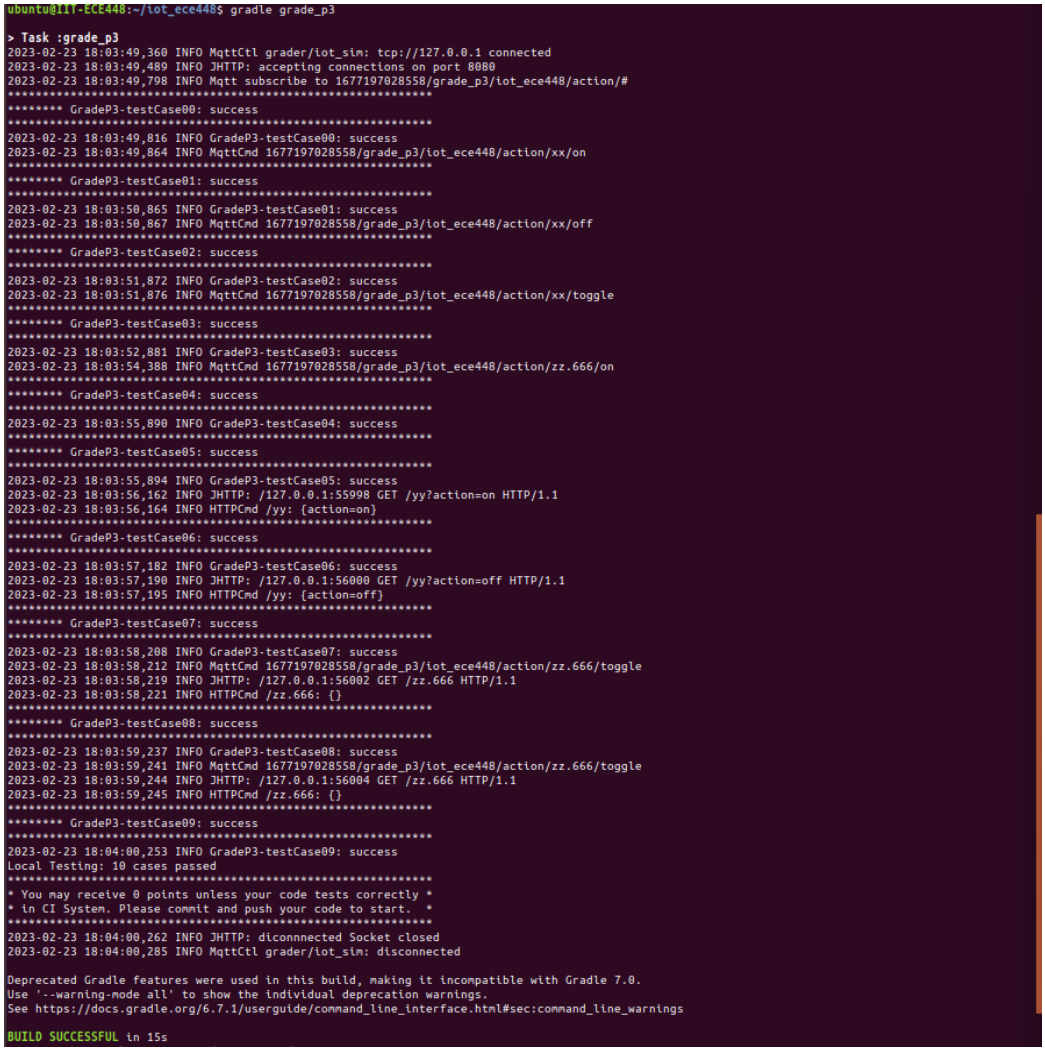


Figure 1: GradleP3- Test cases Results

Screenshots of the HTML results:

To completely utilize the MQTTCommands and MQTTUpdates, I referred to grade_p3 and checked each of the elements within the IoT_Sim program to work over the unit testing. The test cases were designed to test the utilization of the program.

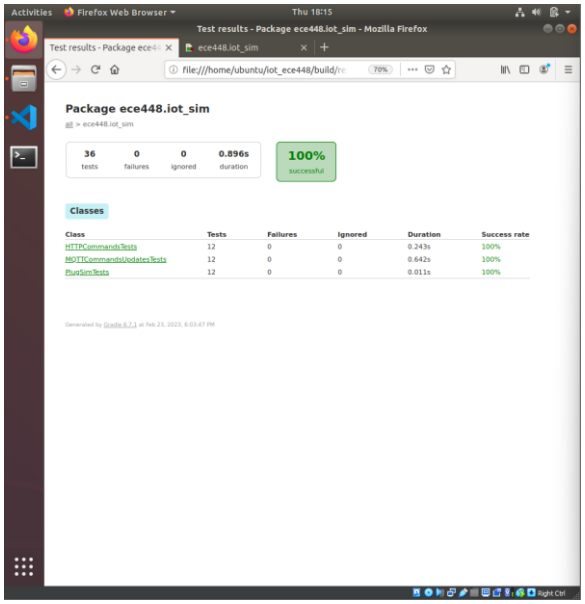


Figure 2: IoT-Sim Test Cases

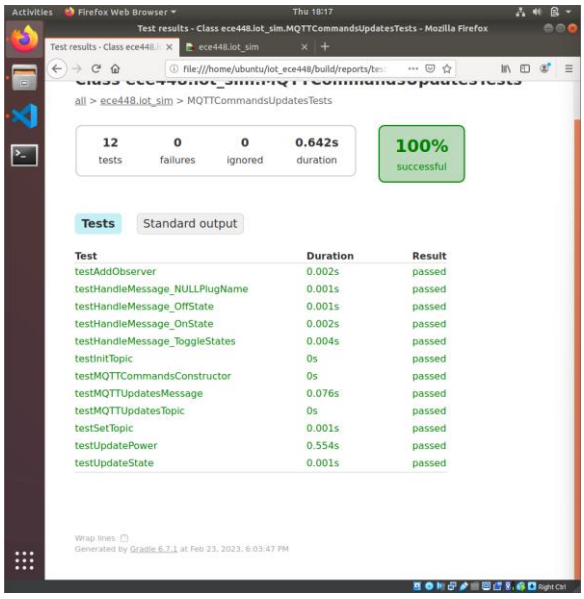
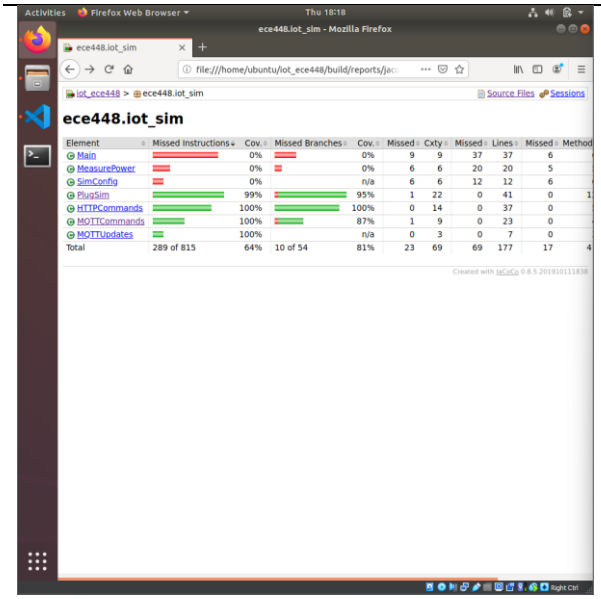


Figure 3: MQTTCommandsUpdates Test Cases



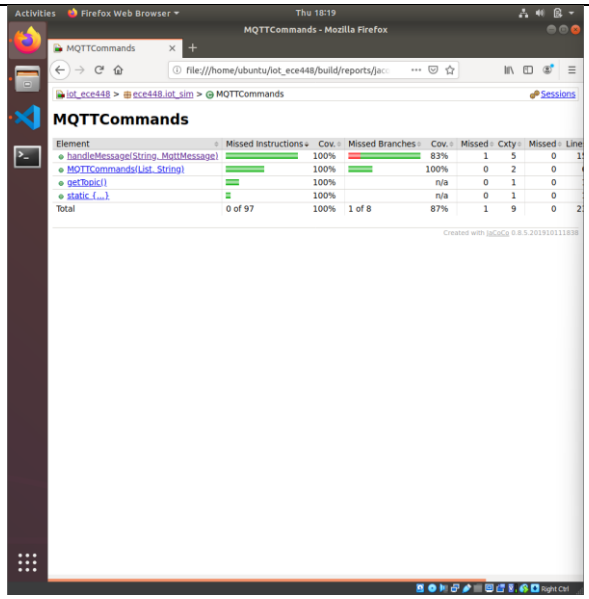


Figure 4: Complete MQTTCommands and MQTTUpdates Instructions

Figure 5: Utilization of Element MQTTCommands

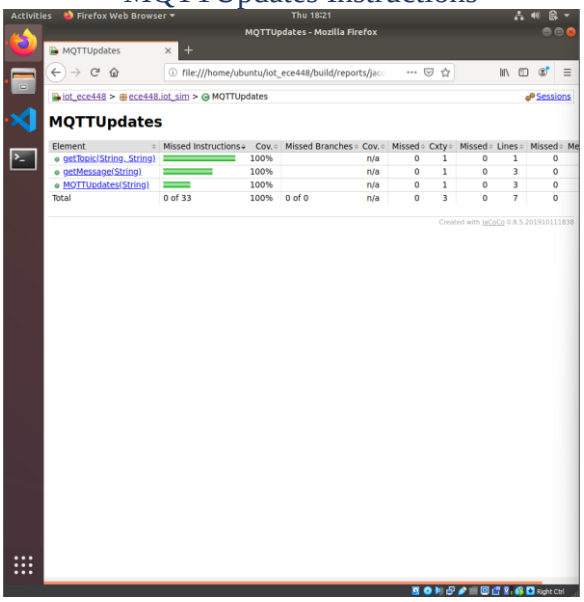


Figure 6: Utilization of Element MQTTUpdates

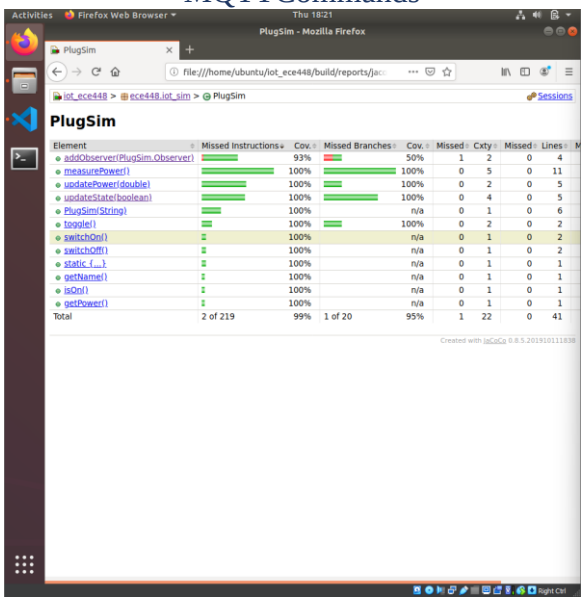


Figure 7: Utilization of Element PlugSim

Conclusion:

In conclusion, this project involves working with the MQTT protocol to control an IoT simulator and report its status. The Eclipse MosquittoTM MQTT broker is installed in the course virtual machine for convenience. The user stories for the project are described in detail, including receiving MQTT messages when a plug is turned on or off, receiving MQTT messages when the power consumption of a plug is measured, and sending MQTT messages to toggle or switch a plug on/off. Testing procedures are implemented in ece448.grading.GradeP3 and should be straightforward to verify that all user stories are covered. While class designs and implementations will be discussed in lectures, students are free to choose designs and implementations that they are comfortable with and should follow the red-green cycle to add unit tests.

Appendix:

Source Code of the edited programs within the project:

MQTTCommands.java

```
package ece448.iot_sim;

import java.util.List;
import java.util.TreeMap;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import org.eclipse.paho.client.mqttv3.MqttMessage;

public class MQTTCommands {
   private final TreeMap<String, PlugSim> plugs = new TreeMap<>();
   private final String topicPrefix;

   public MQTTCommands(List<PlugSim> plugs, String topicPrefix) {
      for (PlugSim plug: plugs)
         this.plugs.put(plug.getName(), plug);

      this.topicPrefix = topicPrefix;
   }

   public String getTopic() {
      return topicPrefix + "/action/#";
   }

   public void handleMessage(String topic, MqttMessage message) {
      logger.info("MqttCmd {}", topic);

      // switch on/off/toggle here
      int firstLast = topic.lastIndexOf("/");
      int secondLast = topic.substring(0, firstLast).lastIndexOf("/");

      String state = topic.substring(firstLast + 1);
      String plugName = topic.substring(secondLast + 1, firstLast);

      PlugSim plug = plugs.get(plugName);
      if (plug == null)
         return;

      switch (state) {
         case "on":
            plug.switchOn();
```

```
            break;

        case "off":
            plug.switchOff();
            break;

        case "toggle":
            plug.toggle();
            break;
      }
   }

   private static final Logger logger = LoggerFactory.getLogger(MQTTCommands.class);
}
```

MQTTUpdates.java

```
package ece448.iot_sim;

import org.eclipse.paho.client.mqttv3.MqttMessage;

public class MQTTUpdates {
   private final String topicPrefix;

   public MQTTUpdates(String topicPrefix) {
      this.topicPrefix = topicPrefix;
   }

   public String getTopic(String name, String key) {
      return topicPrefix + "/update/" + name + "/" + key;
   }

   public MqttMessage getMessage(String value) {
      MqttMessage message = new MqttMessage(value.getBytes());
      message.setRetained(true);
      return message;
   }
}
```

PlugSim.java

```
package ece448.iot_sim;

import java.util.ArrayList;

import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;

/**
 * Simulate a smart plug with power monitoring.
 */
public class PlugSim {

    private final String name;
    private boolean on = false;
    private double power = 0; // in watts

    public static interface Observer {
        void update(String name, String key, String value);
    }
    private final ArrayList<Observer> observers = new ArrayList<>();


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

    /**
     * No need to synchronize if read a final field.
     */
    public String getName() {
        return name;
    }

    synchronized public void addObserver(Observer observer) {
        observers.add(observer);
        //observer.update(name, "state", on ? "on" : "off");
        observer.update(name, "power", String.format("%.3f", power));
    }

    /**
     * Switch the plug on.
     */
    synchronized public void switchOn() {
        // P1: add your code here
        updateState(true);
    }

    /**
     * Switch the plug off.
     */
```

```
synchronized public void switchOff() {
    // P1: add your code here
    updateState(false);
}

/**
 * Toggle the plug.
 */
synchronized public void toggle() {
    // P1: add your code here
    updateState(!on);
}

/**
 * Measure power.
 */
synchronized public void measurePower() {
    if (!on) {
        updatePower(0d);
        return;
    }

    // a trick to help testing
    if (name.indexOf(".") != -1)
    {
        updatePower(Integer.parseInt(name.split("\\.")[1]));
    }
    // do some random walk
    else if (power < 100)
    {
        updatePower(power + Math.random() * 100);
    }
    else if (power > 300)
    {
        updatePower(power - Math.random() * 100);
    }
    else
    {
        updatePower(power + Math.random() * 40 - 20);
    }
}

protected void updateState(boolean o) {
    on = o;
    logger.debug("Plug {}: state {}", name, on? "on": "off");
```

```
        for (Observer observer: observers)
            observer.update(name, "state", on? "on": "off");
    }

    protected void updatePower(double p) {
        power = p;
        logger.debug("Plug {}: power {}", name, power);

        for (Observer observer: observers)
            observer.update(name, "power", String.format("%.3f", power));
    }

    /**
     * Getter: current state
     */
    synchronized public boolean isOn() {
        return on;
    }

    /**
     * Getter: last power reading
     */
    synchronized public double getPower() {
        return power;
    }

    private static final Logger logger = LoggerFactory.getLogger(PlugSim.class);
}
```

MQTTCommandsUpdatesTest.java

```
package ece448.iot_sim;
import ece448.iot_sim.PlugSim.Observer;

import static org.junit.Assert.*;

import java.lang.reflect.Field;
import java.util.*;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import jdk.jfr.Timestamp;

import static org.mockito.Mockito.*;

import org.junit.Test;
```

```
public class MQTTCommandsUpdatesTests {

    @Test
    public void testInitTopic() {
        PlugSim plugA = new PlugSim("a");
        PlugSim plugB = new PlugSim("b");
        PlugSim plugC = new PlugSim("c.1234");
        ArrayList<PlugSim> plugList = new ArrayList<PlugSim>();
        plugList.add(plugA);
        plugList.add(plugB);
        plugList.add(plugC);

        MQTTCommands cmd = new MQTTCommands(plugList, "testPrefix");

        String testTopic = cmd.getTopic();
        System.out.println(testTopic);

        assertTrue(testTopic != null);
    }

    @Test
    public void testSetTopic() {
        PlugSim plugA = new PlugSim("a");
        PlugSim plugB = new PlugSim("b");
        PlugSim plugC = new PlugSim("c.1234");
        ArrayList<PlugSim> plugList = new ArrayList<PlugSim>();
        plugList.add(plugA);
        plugList.add(plugB);
        plugList.add(plugC);

        MQTTCommands cmd = new MQTTCommands(plugList, "testPrefix");

        String testTopic = cmd.getTopic();
        System.out.println(testTopic);

        assertTrue(testTopic.equals("testPrefix/action/#"));
    }

    @Test
    @SuppressWarnings("unchecked")
    public void testMQTTCommandsConstructor() throws NoSuchFieldException, SecurityException,
IllegalArgumentException, IllegalAccessException {
        PlugSim plugA = new PlugSim("a");
        PlugSim plugB = new PlugSim("b");
```

```
        PlugSim plugC = new PlugSim("c");

        List<PlugSim> plugs1 = new ArrayList<>();
        List<PlugSim> plugs2 = new ArrayList<>();

        plugs1.add(plugA);
        plugs1.add(plugB);

        plugs2.add(plugA);
        plugs2.add(plugB);
        plugs2.add(plugC);

        MQTTCommands commands1 = new MQTTCommands(plugs1, "a1");
        MQTTCommands commands2 = new MQTTCommands(plugs2, "b2");

        Field prefixField = MQTTCommands.class.getDeclaredField("topicPrefix");
        prefixField.setAccessible(true);
        String a1=(String) prefixField.get(commands1);
        String b2=(String) prefixField.get(commands2);

        Field plugsField = MQTTCommands.class.getDeclaredField("plugs");
        plugsField.setAccessible(true);
        TreeMap<String, PlugSim> plugsTree1 = (TreeMap<String, PlugSim>) plugsField.get(commands1);
        TreeMap<String, PlugSim> plugsTree2 = (TreeMap<String, PlugSim>) plugsField.get(commands2);

        assertTrue((a1.equals("a1"))
            && (b2.equals("b2"))
            && (plugsTree1.containsKey("a") && plugsTree1.containsKey("b") && !plugsTree1.containsKey("c"))
            && (plugsTree2.containsKey("a") && plugsTree2.containsKey("b") && plugsTree2.containsKey("c"))
            && (plugsTree1.get("a").equals(plugA) && plugsTree1.get("b").equals(plugB))
            && (plugsTree2.get("a").equals(plugA) && plugsTree2.get("b").equals(plugB) &&
plugsTree2.get("b").equals(plugB)));
    }

    @Test
    public void testHandleMessage_NULLPlugName() {
        List<PlugSim> plugs = new ArrayList<>();
        PlugSim plug = new PlugSim("livingroom_lamp");
        plugs.add(plug);
        String topicPrefix = "home/livingroom/plugs";
        MQTTCommands mqttCommands = new MQTTCommands(plugs, topicPrefix);

        String topic = "home/livingroom/plugs/non_existent_plug/on";
        MqttMessage message = new MqttMessage("1".getBytes());
```

```
    mqttCommands.handleMessage(topic, message);

    assertFalse(plug.isOn()); // Plug should not be affected
}

@Test
public void testHandleMessage_OnState() {
    List<PlugSim> plugs = new ArrayList<>();
    PlugSim plug = new PlugSim("livingroom_lamp");
    plugs.add(plug);
    String topicPrefix = "home/livingroom/plugs";
    MQTTCommands mqttCommands = new MQTTCommands(plugs, topicPrefix);

    String topic = "home/livingroom/plugs/livingroom_lamp/on";
    MqttMessage message = new MqttMessage("1".getBytes());

    mqttCommands.handleMessage(topic, message);

    assertTrue(plug.isOn()); // Plug should be turned on
}

@Test
public void testHandleMessage_OffState() {
    List<PlugSim> plugs = new ArrayList<>();
    PlugSim plug = new PlugSim("livingroom_lamp");
    plugs.add(plug);
    String topicPrefix = "home/livingroom/plugs";
    MQTTCommands mqttCommands = new MQTTCommands(plugs, topicPrefix);

    String topic = "home/livingroom/plugs/livingroom_lamp/off";
    MqttMessage message = new MqttMessage("0".getBytes());

    mqttCommands.handleMessage(topic, message);

    assertFalse(plug.isOn()); // Plug should be turned off
}

@Test
public void testHandleMessage_ToggleStates() {
    PlugSim plugA = new PlugSim("a");
    PlugSim plugB = new PlugSim("b");

    List<PlugSim> plugs = new ArrayList<>();

    plugs.add(plugA);
```

```
        plugs.add(plugB);

        MQTTCommands commands = new MQTTCommands(plugs, "prefix");

        commands.handleMessage("bogus/a/on", null);
        assertTrue(plugA.isOn() && !plugB.isOn());

        commands.handleMessage("morebogus/a/toggle", null);
        assertTrue(!plugA.isOn() && !plugB.isOn());

        commands.handleMessage("1/2/3/b/toggle", null);
        assertTrue(!plugA.isOn() && plugB.isOn());

        commands.handleMessage("/b/off", null);
        assertTrue(!plugA.isOn() && !plugB.isOn());
    }

    @Test
    public void testMQTTUpdatesTopic() throws NoSuchFieldException, SecurityException,
IllegalArgumentException, IllegalAccessException {
        MQTTUpdates updates1 = new MQTTUpdates("a");
        MQTTUpdates updates2 = new MQTTUpdates("b");

        Field prefixField = MQTTUpdates.class.getDeclaredField("topicPrefix");
        prefixField.setAccessible(true);
        String a=(String) prefixField.get(updates1);
        String b=(String) prefixField.get(updates2);

        assertTrue((a.equals("a"))
            && (b.equals("b"))
            && (updates1.getTopic("name1", "key1").equals("a/update/name1/key1"))
            && (updates1.getTopic("name2", "key2").equals("a/update/name2/key2")));
    }

    @Test
    public void testMQTTUpdatesMessage() {
        MQTTUpdates updates = new MQTTUpdates("prefix");

        MqttMessage message1 = updates.getMessage("a");
        MqttMessage message2 = updates.getMessage("b");

        assertTrue((message1.isRetained() && message1.toString().equals("a"))
            && (message2.isRetained() && message2.toString().equals("b")));
    }
```

```
/*
Additional Test cases created for PlugSim updates to utilize 100% instructions
*/

@Test
public void testUpdateState() {
    // Set up
    PlugSim plug = new PlugSim("test_plug");
    TestObserver observer = new TestObserver();
    plug.addObserver(observer);
    // Call updateState
    plug.updateState(true);
    // Verify
    assertTrue(plug.isOn());
    // Call updateState
    plug.updateState(false);
    // Verify
    assertFalse(plug.isOn());
}

@Test
public void testAddObserver() {
    PlugSim plugSim = new PlugSim("TestPlug");
    TestObserver observer = new TestObserver();
    plugSim.addObserver(observer);
    assertEquals("TestPlug", observer.name);
    plugSim.switchOn();
    assertEquals("on", observer.getState());
    plugSim.switchOff();
    assertEquals("off", observer.getState());
    assertEquals("0.000", observer.power);
}

private static class TestObserver implements PlugSim.Observer {
    public String name;
    public String state;
    public String power;
    public String getState() {
        return state;
    }
    @Override
    public void update(String name, String key, String value) {
        this.name = name;
        if (key.equals("state")) {
```

```
            this.state = value;
        } else if (key.equals("power")) {
            this.power = value;
        }
    }
}

@Test
public void testUpdatePower() {
    // create mock observer
    Observer observer = mock(Observer.class);
    // create plug
    PlugSim plug = new PlugSim("TestPlug");
    // add observer to plug
    plug.addObserver(observer);
    // call updatePower method
    plug.updatePower(42.123);
    // verify that observer.update method was called with correct parameters
    assertEquals(42.123, plug.getPower(), 0.001);
}

}
```