Other (Previous Test cases written) – for your reference

test/java/ece448/iot_sim repository

MqttTests.java

```
package ece448.iot_sim;


import static org.junit.Assert.*;

import org.junit.Test;


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

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

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


import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;


public class MqttTests {


   // PlugSim Tests

   @Test

   public void testSwitchOn() {
```

```
    PlugSim plug = new PlugSim("testPlug");
    plug.switchOn();
    assertTrue(plug.isOn());
}

@Test
public void testSwitchOff() {
    PlugSim plug = new PlugSim("testPlug");
    plug.switchOn();
    plug.switchOff();
    assertFalse(plug.isOn());
}

@Test
public void testToggle() {
    PlugSim plug = new PlugSim("testPlug");
    plug.toggle();
    assertTrue(plug.isOn());
    plug.toggle();
    assertFalse(plug.isOn());
}

@Test
public void testMeasurePower() {
    PlugSim plug = new PlugSim("testPlug");
    plug.switchOn();
```

```
    plug.measurePower();
    assertNotEquals(0.0, plug.getPower(), 0.001);
}

@Test
public void testMeasurePowerWithDotInName() {
    PlugSim plug = new PlugSim("test.123");
    plug.switchOn();
    plug.measurePower();
    assertEquals(123.0, plug.getPower(), 0.001);
}

private static class TestObserver implements PlugSim.Observer {
    private String lastName;
    private String lastKey;
    private String lastValue;

    @Override
    public void update(String name, String key, String value) {
        this.lastName = name;
        this.lastKey = key;
        this.lastValue = value;
    }

    public boolean receivedStateUpdate(String state) {
        return "state".equals(lastKey) && state.equals(lastValue);
```

```
    }

    public boolean receivedPowerUpdate() {
        return "power".equals(lastKey);
    }
}

@Test
public void testObserverNotificationOnSwitchOn() {
    PlugSim plug = new PlugSim("testPlug");
    TestObserver observer = new TestObserver();
    plug.addObserver(observer);
    plug.switchOn();
    assertTrue(observer.receivedStateUpdate("on"));
}

@Test
public void testObserverNotificationOnPowerChange() {
    PlugSim plug = new PlugSim("testPlug");
    TestObserver observer = new TestObserver();
    plug.addObserver(observer);
    plug.switchOn();
    plug.measurePower();
    assertTrue(observer.receivedPowerUpdate());
}
```

```
// MqttCommands Tests
@Test
public void testHandleMessageOn() throws Exception {
   MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "testPrefix");
   PlugSim plug = new PlugSim("testPlug");
   mqttCmd.addPlug(plug);

   String topic = "testPrefix/action/testPlug/on";
   MqttMessage msg = new MqttMessage("".getBytes());

   mqttCmd.handleMessage(topic, msg);
   assertTrue(plug.isOn());
}


@Test
public void testHandleMessageOff() throws Exception {
   MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "testPrefix");
   PlugSim plug = new PlugSim("testPlug");
   plug.switchOn();
   mqttCmd.addPlug(plug);

   String topic = "testPrefix/action/testPlug/off";
   MqttMessage msg = new MqttMessage("".getBytes());

   mqttCmd.handleMessage(topic, msg);
```

```
    assertFalse(plug.isOn());
}

@Test
public void testHandleMessageToggle() throws Exception {
    MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "testPrefix");
    PlugSim plug = new PlugSim("testPlug");
    mqttCmd.addPlug(plug);

    String topic = "testPrefix/action/testPlug/toggle";
    MqttMessage msg = new MqttMessage("".getBytes());

    mqttCmd.handleMessage(topic, msg);
    assertTrue(plug.isOn());
}

@Test
public void testHandleMessageInvalidTopic() {
    MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "");
    String topic = "invalidTopic";
    MqttMessage msg = new MqttMessage();

    mqttCmd.handleMessage(topic, msg);
    // Ensure no exceptions and plug remains unchanged
}
```

```
@Test
public void testHandleMessageUnknownAction() {
    MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "");
    PlugSim plug = new PlugSim("testPlug");
    mqttCmd.addPlug(plug);

    String topic = "testPrefix/action/testPlug/invalid";
    MqttMessage msg = new MqttMessage();

    mqttCmd.handleMessage(topic, msg);
    assertFalse(plug.isOn()); // No change as action is unknown
}

@Test
public void testHandleMessageNonExistentPlug() {
    MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "");
    String topic = "testPrefix/action/nonExistentPlug/on";
    MqttMessage msg = new MqttMessage();

    mqttCmd.handleMessage(topic, msg);
    // No plug exists, ensure no exceptions
}

// MqttUpdates Tests
@Test
public void testGetTopic() throws Exception {
```

```
    MqttClient mqttClient = new MqttClient("tcp://localhost:1883", "testClient");
    MqttUpdates mqttUpd = new MqttUpdates("testPrefix", mqttClient);
    String name = "testPlug";
    String key = "state";
    String expectedTopic = "testPrefix/update/testPlug/state";
    assertEquals(expectedTopic, mqttUpd.getTopic(name, key));
}

@Test
public void testGetTopicWithMultiLevelPrefix() throws Exception {
    MqttClient mqttClient = new MqttClient("tcp://localhost:1883", "testClient");
    MqttUpdates mqttUpd = new MqttUpdates("a/b/c", mqttClient);
    String topic = mqttUpd.getTopic("plugName", "state");
    assertEquals("a/b/c/update/plugName/state", topic);
}

@Test
public void testGetMessage() throws Exception {
    MqttClient mqttClient = new MqttClient("tcp://localhost:1883", "testClient");
    MqttUpdates mqttUpd = new MqttUpdates("testPrefix", mqttClient);
    String value = "on";
    MqttMessage msg = mqttUpd.getMessage(value);
    assertEquals("on", new String(msg.getPayload()));
    assertTrue(msg.isRetained());
}
```

```
@Test
public void testPowerRandomWalk() {
   PlugSim plug = new PlugSim("testRandom");
   plug.switchOn();
   for (int i = 0; i < 10; i++) {
      plug.measurePower();
   }
   assertTrue(plug.getPower() >= 0);
}

@Test
public void testMessageRetentionFlag() throws Exception {
   try (MqttClient client = new MqttClient("tcp://localhost:1883", "testClient")) {
      MqttUpdates mqttUpd = new MqttUpdates("prefix", client);
      MqttMessage msg = mqttUpd.getMessage("on");
      assertTrue(msg.isRetained());
   }
}

@Test
public void testMultiLevelTopicPrefix() throws Exception {
   MqttClient client = null;
   try {
      client = new MqttClient("tcp://localhost:1883", "testClient", null);
      MqttUpdates mqttUpd = new MqttUpdates("a/b/c", client);
      String topic = mqttUpd.getTopic("plug", "state");
```

```
        assertEquals("a/b/c/update/plug/state", topic);
    } finally {
        if (client != null && client.isConnected()) {
            client.disconnect();
        }
    }
}

@Test
public void testPowerCalculationWithDottedName() {
    PlugSim plug = new PlugSim("test.250");
    plug.switchOn();
    plug.measurePower();
    assertEquals(250.0, plug.getPower(), 0.001);
}

@Test
public void testConcurrentToggle() throws InterruptedException {
    PlugSim plug = new PlugSim("concurrentPlug");
    int numThreads = 10;
    ExecutorService executor = Executors.newFixedThreadPool(numThreads);

    assertFalse(plug.isOn());

    for (int i = 0; i < numThreads; i++) {
        executor.submit(plug::toggle);
```

```
    }
    executor.shutdown();
    executor.awaitTermination(1, TimeUnit.SECONDS);

    assertFalse(plug.isOn());
}

@Test
public void testMqttCommandsConstructor() {
    List<PlugSim> plugList = new ArrayList<>();
    plugList.add(new PlugSim("plug1"));
    plugList.add(new PlugSim("plug2"));
    MqttCommands mqttCmd = new MqttCommands(plugList, "testPrefix");
    assertEquals(2, mqttCmd.plugs.size());
    assertTrue(mqttCmd.plugs.containsKey("plug1"));
    assertTrue(mqttCmd.plugs.containsKey("plug2"));
}

@Test
public void testGetTopic1() {
    MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "testPrefix");
    String expectedTopic = "testPrefix/action/#";
    assertEquals(expectedTopic, mqttCmd.getTopic());
}

@Test
```

```
public void testHandleMessageExceptionHandling() {
   MqttCommands mqttCmd = new MqttCommands(new ArrayList<>(), "testPrefix");
   String invalidTopic = null;
   MqttMessage msg = new MqttMessage();
   try {
      mqttCmd.handleMessage(invalidTopic, msg);
   } catch (Exception e) {
      fail("Exception should have been handled gracefully.");
   }
}

@Test
public void testPublishUpdateSuccess() throws Exception {
   MqttClient client = new MqttClient("tcp://localhost:1883", "testClient");
   client.connect();
   MqttUpdates mqttUpd = new MqttUpdates("testPrefix", client);
   String name = "testPlug";
   String key = "state";
   String value = "on";
   mqttUpd.publishUpdate(name, key, value);
   client.subscribe("testPrefix/update/testPlug/state", (topic, message) -> {
   assertEquals("on", new String(message.getPayload()));
   assertTrue(message.isRetained());
   client.disconnect();
   });
}
```

```
    @Test
    public void testPublishUpdateExceptionHandling() throws Exception {
        MqttClient client = new MqttClient("tcp://localhost:1883", "testClient");
        client.connect();
        client.disconnect();
        MqttUpdates mqttUpd = new MqttUpdates("testPrefix", client);
        try {
            mqttUpd.publishUpdate("testPlug", "state", "on");
        } catch (Exception e) {
            fail("Exception should have been handled gracefully.");
        }
    }

}
```

HTTPCommandsTests.java

```
package ece448.iot_sim;

import static org.junit.Assert.*;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.junit.Before;
```

```
import org.junit.Test;

public class HTTPCommandsTests {

    private HTTPCommands httpCommands;
    private PlugSim plug1;
    private PlugSim plug2;
    private PlugSim plugWithSpecialChar;

    @Before
    public void setUp() {
        plug1 = new PlugSim("plug1");
        plug2 = new PlugSim("plug2");
        plugWithSpecialChar = new PlugSim("zzzz.789");

        List<PlugSim> plugs = new ArrayList<>();
        plugs.add(plug1);
        plugs.add(plug2);
        plugs.add(plugWithSpecialChar);

        httpCommands = new HTTPCommands(plugs);
    }

    @Test
    public void testPlugReportDisplay() {
        String response = httpCommands.handleGet("/plug1", new HashMap<>());
```

```
    assertTrue(response.contains("plug1"));
    assertTrue(response.contains("plug1 is off"));
    assertTrue(response.contains("Power reading is 0.000"));
    assertTrue(response.contains("action=on"));
    assertTrue(response.contains("action=off"));
    assertTrue(response.contains("action=toggle"));
}

@Test
public void testSwitchOnAction() {
    Map<String, String> params = new HashMap<>();
    params.put("action", "on");
    String response = httpCommands.handleGet("/plug1", params);

    assertTrue(response.contains("plug1 is on"));
    assertTrue(plug1.isOn());

    String checkResponse = httpCommands.handleGet("/plug1", new HashMap<>());
    assertTrue(checkResponse.contains("plug1 is on"));
}

@Test
public void testSwitchOffAction() {

    plug1.switchOn();
```

```
    Map<String, String> params = new HashMap<>();
    params.put("action", "off");
    String response = httpCommands.handleGet("/plug1", params);

    assertTrue(response.contains("plug1 is off"));
    assertFalse(plug1.isOn());

    String checkResponse = httpCommands.handleGet("/plug1", new HashMap<>());
    assertTrue(checkResponse.contains("plug1 is off"));
}

@Test
public void testToggleActionOffToOn() {

    plug1.switchOff();

    Map<String, String> params = new HashMap<>();
    params.put("action", "toggle");
    String response = httpCommands.handleGet("/plug1", params);

    assertTrue(response.contains("plug1 is on"));
    assertTrue(plug1.isOn());
}

@Test
```

```
public void testToggleActionOnToOff() {

    plug1.switchOn();

    Map<String, String> params = new HashMap<>();
    params.put("action", "toggle");
    String response = httpCommands.handleGet("/plug1", params);

    assertTrue(response.contains("plug1 is off"));
    assertFalse(plug1.isOn());
}

@Test
public void testPowerReadingUpdate() {

    plugWithSpecialChar.switchOn();
    plugWithSpecialChar.updatePower(789.0);

    String response = httpCommands.handleGet("/zzzz.789", new HashMap<>());
    assertTrue(response.contains("Power reading is 789.000"));
}

@Test
public void testMultiplePlugsIndependence() {

    plug1.switchOn();
```

```
    plug2.switchOff();

    Map<String, String> params = new HashMap<>();
    params.put("action", "toggle");
    httpCommands.handleGet("/plug1", params);

    assertFalse(plug1.isOn());

    assertFalse(plug2.isOn());

    String plug2Response = httpCommands.handleGet("/plug2", new HashMap<>());
    assertTrue(plug2Response.contains("plug2 is off"));
}

@Test
public void testSpecialCharactersInPlugNames() {

    String initialResponse = httpCommands.handleGet("/zzzz.789", new HashMap<>());
    assertTrue(initialResponse.contains("zzzz.789"));

    Map<String, String> params = new HashMap<>();
    params.put("action", "on");
    String updatedResponse = httpCommands.handleGet("/zzzz.789", params);

    assertTrue(updatedResponse.contains("zzzz.789 is on"));
    assertTrue(plugWithSpecialChar.isOn());
```

```
}

@Test
public void testInvalidActionParameter() {

    Map<String, String> params = new HashMap<>();
    params.put("action", "invalid");
    String response = httpCommands.handleGet("/plug1", params);

    assertTrue(response.contains("plug1 is off"));
    assertFalse(plug1.isOn());
}

@Test
public void testConcurrentActions() {

    Map<String, String> onParams = new HashMap<>();
    onParams.put("action", "on");
    httpCommands.handleGet("/plug1", onParams);
    assertTrue(plug1.isOn());

    Map<String, String> offParams = new HashMap<>();
    offParams.put("action", "off");
    httpCommands.handleGet("/plug1", offParams);
    assertFalse(plug1.isOn());
```

```
        Map<String, String> onAgainParams = new HashMap<>();
        onAgainParams.put("action", "on");
        String finalResponse = httpCommands.handleGet("/plug1", onAgainParams);

        assertTrue(finalResponse.contains("plug1 is on"));
        assertTrue(plug1.isOn());
    }

    @Test
    public void testListPlugs() {

        String response = httpCommands.handleGet("/", new HashMap<>());

        assertTrue(response.contains("href='/plug1'"));
        assertTrue(response.contains("href='/plug2'"));
        assertTrue(response.contains("href='/zzzz.789'"));
    }

    @Test
    public void testNonExistentPlug() {

        String response = httpCommands.handleGet("/nonexistent", new HashMap<>());

        assertNull(response);
    }
}
```

PlugSimTests.java

```
package ece448.iot_sim;

import static org.junit.Assert.*;

import org.junit.Test;

public class PlugSimTests {

	@Test
	public void testInit() {
		PlugSim plug = new PlugSim("a");

		assertFalse(plug.isOn());
	}

	@Test
	public void testSwitchOn() {
		PlugSim plug = new PlugSim("a");

		plug.switchOn();

		assertTrue(plug.isOn());
	}

	@Test
```

```
public void testGetName() {
   PlugSim plug = new PlugSim("test.100");
   assertEquals("test.100", plug.getName());
}

        @Test
public void testSwitchOffFromOn() {
   PlugSim plug = new PlugSim("a");
   plug.switchOn();
   plug.switchOff();
   assertFalse(plug.isOn());
}

        @Test
public void testMultipleSwitching() {
   PlugSim plug = new PlugSim("a");
   plug.switchOn();
   plug.switchOff();
   plug.switchOn();
   assertTrue(plug.isOn());
}

        @Test
public void testToggleFromOn() {
   PlugSim plug = new PlugSim("a");
   plug.switchOn();
```

```
    plug.toggle();
    assertFalse(plug.isOn());
  }

      @Test
  public void testToggleFromOff() {
    PlugSim plug = new PlugSim("a");
    plug.toggle();
    assertTrue(plug.isOn());
  }

      @Test
public void testPowerMeasurementWhenOn() {
  PlugSim plug = new PlugSim("test.500");
  plug.switchOn();
  plug.measurePower();
  assertEquals(500.0, plug.getPower(), 0.001);
  }

      @Test
public void testPowerMeasurementWhenOff() {
  PlugSim plug = new PlugSim("test.500");
  plug.measurePower();
  assertEquals(0.0, plug.getPower(), 0.001);

      }
```

```
        @Test
public void testMultipleToggleAndPower() {
   PlugSim plug = new PlugSim("test.300");
   plug.toggle();
   plug.measurePower();
   assertEquals(300.0, plug.getPower(), 0.001);
   plug.toggle();
   plug.measurePower();
   assertEquals(0.0, plug.getPower(), 0.001);
   plug.toggle();
   plug.measurePower();
   assertEquals(300.0, plug.getPower(), 0.001);
        }

        @Test
public void testRandomWalkLowPower() {
   PlugSim plug = new PlugSim("a");
   plug.switchOn();
   plug.updatePower(50);
   double initialPower = plug.getPower();
   plug.measurePower();
   double newPower = plug.getPower();
   assertTrue(newPower > initialPower);
   assertTrue(newPower <= initialPower + 100);
}
```

```
@Test
public void testRandomWalkHighPower() {
   PlugSim plug = new PlugSim("a");
   plug.switchOn();
   plug.updatePower(350);
   double initialPower = plug.getPower();
   plug.measurePower();
   double newPower = plug.getPower();
   assertTrue(newPower < initialPower);
   assertTrue(newPower >= initialPower - 100);
}

@Test
public void testRandomWalkMediumPower() {
   PlugSim plug = new PlugSim("a");
   plug.switchOn();
   plug.updatePower(200);
   double initialPower = plug.getPower();
   plug.measurePower();
   double newPower = plug.getPower();
   assertTrue(Math.abs(newPower - initialPower) <= 20);
}

}
```