EECS 4313 Assignment 2
Black-box and White-box Testing with JUnit

Student Name — Student Number — EECS Account
Edward Vaisman — 212849857 — eddyv
Robin Bandzar — 212200531 — cse23028
Kirusanth Thiruchelvam — 212918298 — kirusant
Sadman Sakib Hasan — 212497509 — cse23152

March 2, 2018

Contents

1 Black Box Testing

- **Technique**: *Boundary Value Testing*
- **Class**: *net.sf.borg.common.SocketClient.java*
- **Method**: *sendMsg(String host, int port, String msg)*
- **Method Description**: This method sends a given message to a given host, port and returns the response from the socket.
  - the first argument *host* is the host that the socket client should be connected to.
  - the second argument *port* is the port on the host that the socket client should be connected to
  - the third argument *msg* is the message that should be sent over the host and port given.

  IOException: If an I/O error occurs when sending the message.
- **Justification**: Boundary value testing is best suited for methods that have inputs that could be seperated into partitions. For this method the port could be partitioned. We have our valid partition which is between 0 and 65535 (inclusive) and our invalid partitions which is any port<0 or any port>65535. The msg could be anything and the host could be partitioned into valid/invalid hostnames.
- **Evaluation**: The test below applies weak robust testing on the method *sendMsg* which reveals a bug within the method that causes the method to throw an IllegalArgumentException If the port parameter is outside the specified range of valid port values, which is between 0 and 65535, inclusive. Refer to Figure. 1, Figure. 2 and Listing 1

  The test is designed to not fail on any IOExceptions that may occur such as UnknownHostException or ConnectionException since the method is supposed to *throw* an IOException.

  However, applying weak normal testing will cause all the tests to pass. Refer to Figure, 3 and Listing. 2

Listing 1: Weak Robust Testing Variables

```
String validHost = "localhost";
port_norm = 2929; // x_norm
port_min = 0; // x_min
port_min_plus = 1; // x_min+
port_max = 65535; // x_max
port_max_minus = 65534; // x_max-
// robustness
String invalidHost = "asdfasdf"; //unknownhostexception
int port_min_minus = -1; // x_min-
int port_max_plus = 65536; // x_max_+
```

Listing 2: Weak Normal Testing Variables

```
String validHost = "localhost";
port_norm = 2929; // x_norm
port_min = 0; // x_min
port_min_plus = 1; // x_min+
port_max = 65535; // x_max
port_max_minus = 65534; // x_max-
\label{list:weak_normal}
```

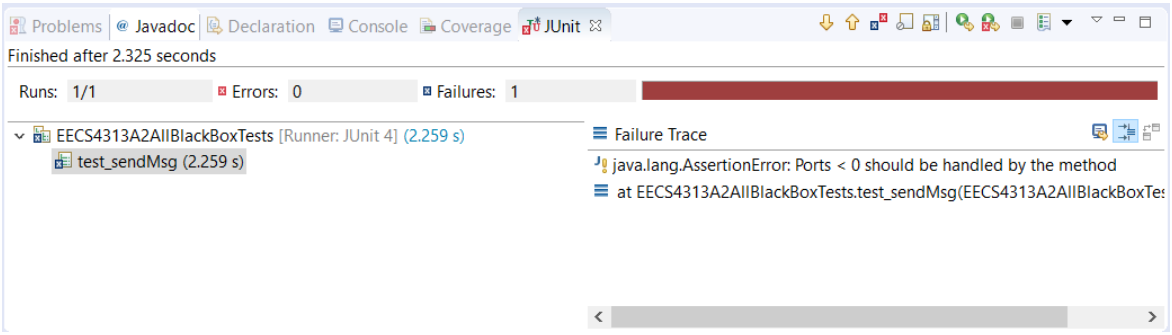

Figure 1: Test results using Weak Robust Boundary Value Testing

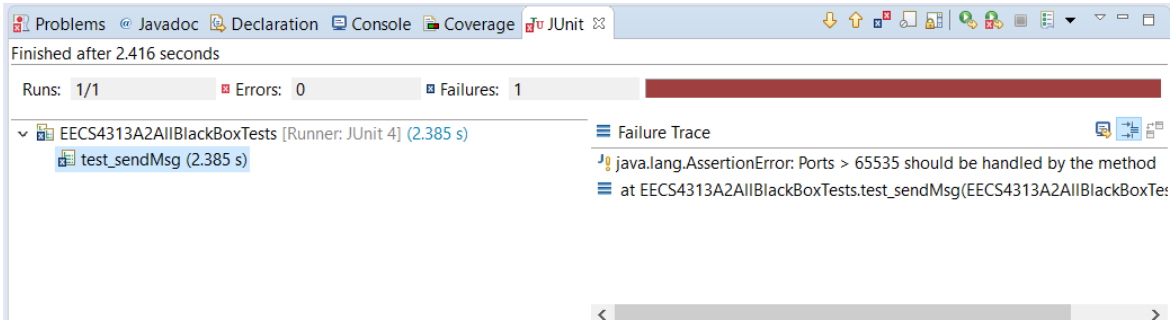

Figure 2: Test results using Weak Robust Boundary Value Testing

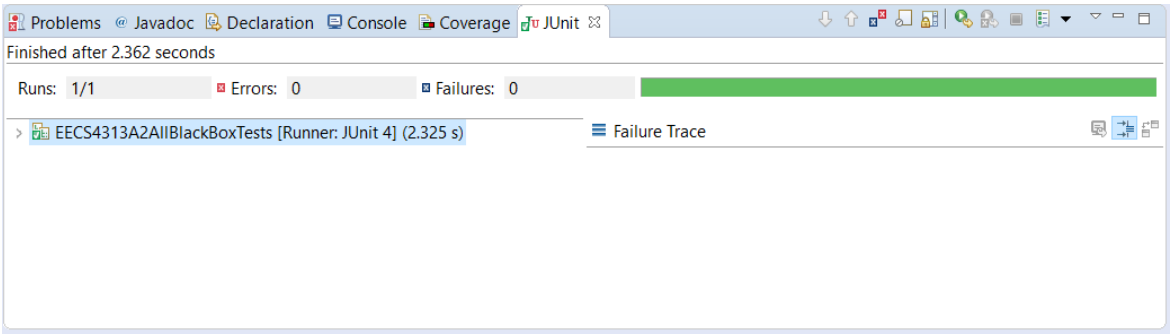

Figure 3: Test results using Weak Normal Boundary Value Testing

```
public class EECS4313A2AllBlackBoxTests implements
    SocketHandler {

  /**
   * process a socket message
   */
  @Override
  public synchronized String processMessage(String msg) {
    return msg;
  }

  @Test
  public void test_sendMsg() {
    /** Method used: Boundary Value Testing **/
    String validHost = "localhost";

    int port_norm = 2929; // x_norm
    int port_min = 0; // x_min
    int port_min_plus = 1; // x_min+
    int port_max = 65535; // x_max
    int port_max_minus = 65534; // x_max-

    // robustness
    String invalidHost = "asdfasdf";
    int port_min_minus = -1; // x_min-
    int port_max_plus = 65536; // x_max_+

    String response = "";
    // port_norm
    String msg = "Port 2929";
    SocketServer ss = new SocketServer(port_norm, this);
    try {
      response = SocketClient.sendMsg(validHost, port_norm,
          msg);
      assertTrue("Testing if a localhost on port_norm sends a
          message", response.equals(msg));
    } catch (IOException e) {
      e.printStackTrace();
    }
    /* Unknown host exception extends IOException */
    try {
```

```
    response = SocketClient.sendMsg(invalidHost, port_norm,
       msg);
    assertTrue("Testing if an invalid host on port_norm
       sends a message", response.equals(msg));
  } catch (IOException e) {
    e.printStackTrace();
  }

  // port_min
  /*
   * Throws connection problem. port 0 isn't available on
     my computer Connect
   * Exception extends Socket Exception which extends
     IOException
   */
  msg = "Port 0";
  try {
    ss = new SocketServer(port_min, this);
    response = SocketClient.sendMsg(validHost, port_min,
       msg);
    assertTrue("Testing if a localhost on port_min sends a
       message", response.equals(msg));
  } catch (IOException e) {
    e.printStackTrace();
  }
  // port_min+
  msg = "Port 1";
  try {
    ss = new SocketServer(port_min_plus, this);
    response = SocketClient.sendMsg(validHost,
       port_min_plus, msg);
    assertTrue("Testing if a localhost on port port_min+
       sends a message", response.equals(msg));
  } catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }

  // port_max
  msg = "Port 65535";
  try {
```

```
    ss = new SocketServer(port_max, this);
    response = SocketClient.sendMsg(validHost, port_max,
       msg);
    assertTrue("Testing if a localhost on port port_max
       sends a message", response.equals(msg));
  } catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }

  // port_max-
  msg = "Port 65534";
  try {
    ss = new SocketServer(port_max_minus, this);
    response = SocketClient.sendMsg(validHost,
       port_max_minus, msg);
    assertTrue("Testing if a localhost on port_max- sends a
       message", response.equals(msg));

  } catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }

  // port_min-
  /*
   * Illegal argument Exception
   */

  msg = "Port -1";
  try {
    ss = new SocketServer(port_min_minus, this);
    response = SocketClient.sendMsg(validHost,
       port_min_minus, msg);
    assertTrue("Testing if a localhost on port_min- sends a
       message", response.equals(msg));
  } catch (IOException e) {
    e.printStackTrace();
  } catch (IllegalArgumentException iae) {
    fail("Ports < 0 should be handled by the method");
  }
```

```
    // port_max+
    /*
     * Illegal argument Exception
     */
    msg = "Port 65536";
    try {
      ss = new SocketServer(port_max_plus, this);
      response = SocketClient.sendMsg(validHost,
         port_max_plus, msg);
      assertTrue("Testing if a localhost on port_max+ sends a
         message", response.equals(msg));
    } catch (IOException e) {
      e.printStackTrace();
    } catch (IllegalArgumentException iae) {
      fail("Ports > 65535 should be handled by the method");
    }
  }

}
```

2 White Box Testing

- The statement coverage measurements for your Assignment 2 test suite.
- A description of the test cases that you added in this assignment to improve statement coverage. The marker will not read your code in order to see what you tested. You have to describe it.
- The statement coverage measurements for your final submission. Include the screenshots of the test running results and the screenshots of the coverage measurement. If your coverage is not 100
- The Control Flow Graph you created. Indicate the segments clearly (you will probably need to include the code for this).
- The path coverage discussion described in section 2 above.
- Attaching bug reports if bugs are discovered using your testing methods. You should use the same bug report format as in Assignment 1. Do not file these bug reports to the projects bug report system.
- An appendix with the specification of the methods you are testing (if there are new ones).