CS 362 Final Project Part B
URLValidator

Project Group
Robyn Lin
Faye Yao
Shaun Stapleton

Due Monday, March 14

Manual Testing

Our manual testing methodology was to breakdown the components of what makes a URL, and test these components to see which components caused failures. We tested what should be valid URLs and invalid URLs. We can narrow down what components of the URLValidator are broken if we partition our input to test the different components of the URL and see if the URLValidator returned the correct result (false if it's false, true if it's true).

For example, URLs tested included:
https://www.amazon.com
http://amazon.com
ftp://www.amazon.com
h3t://www.amazon.com
www.amazon.com
amazon.com
http://www.amazon.ca
http://www.amazon.com:8080
http://www.amazon.com:80df
http:\\www.amazon.com
http://www.wmazon.com/test.html?action=test
“” (no URL)
https://foo.bar.com\t
https://www.amazon.com\t
https://amazon.com\t
https://amazon.com\b
https://amazon.com\f
https://amazon.com:22\t
https://amazon.com:22\t   /today

Input partitioning

The goal of input partitioning is to cover the domain by testing a substantial range of valid and invalid urls using the isValid function in URLValidator.java. This partitioning helps us better understand the behavior of the system under test, and helps expose any possible faults in the system. We decided to make the first partition block test a series of valid urls as input. Then the second block alters each part, such as scheme, authority, etc., of these valid urls so each component becomes invalid. The third block also alters the valid urls from the first block so each part of the url is omitted in turn. These blocks are disjoint and cover a large portion of the domain. Again, we used an Agan Principle of *'Divide and Conquer'* to divide the tests into separate blocks, and to test which side of the bug we are on.

For example:

```
public void testYourFirstPartition()
 {
		//test valid full url inputs
		System.out.println("Testing first partition: valid full url inputs");
		UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);
		System.out.println(urlVal.isValid("http://google.com"));
		System.out.println(urlVal.isValid("http://go.com:80"));
		System.out.println(urlVal.isValid("http://255.com"));
		System.out.println(urlVal.isValid("http://yahoo.com"));
}

  public void testYourSecondPartition(){
		//test invalid full url inputs
		System.out.println("Testing second partition: invalid full url inputs");
		UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);
		System.out.println(urlVal.isValid("http:/google.com"));
		System.out.println(urlVal.isValid("http://google/com"));
		System.out.println(urlVal.isValid("http://google.bad"));
		System.out.println(urlVal.isValid("://go.com:80"));
		System.out.println(urlVal.isValid("http://go..com:80"));
		System.out.println(urlVal.isValid("http://go.com:-1"));
		System.out.println(urlVal.isValid("htttp://255.com"));
		System.out.println(urlVal.isValid("http://255/com"));
		System.out.println(urlVal.isValid("http://255.ccc"));
		System.out.println(urlVal.isValid("yahoo..com"));
		System.out.println(urlVal.isValid("http:yahoo.abc"));
  }

  public void testYourThirdPartition(){
		System.out.println("Testing third partition: (partially) empty url inputs");
		UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);
		System.out.println(urlVal.isValid("http://..com"));
```

```
            System.out.println(urlVal.isValid("google.com"));-should have returned true
            System.out.println(urlVal.isValid("http://www.google"));
            System.out.println(urlVal.isValid("go.com:80"));-should have returned true
            System.out.println(urlVal.isValid("http://go"));
            System.out.println(urlVal.isValid("255.com"));-should have returned true
            System.out.println(urlVal.isValid("http://255"));
            System.out.println(urlVal.isValid("yahoo.com"));-should have returned true
            System.out.println(urlVal.isValid("http://www.yahoo"));
  }
```

Unit Testing

The goal of our unit testing was to cover several separate portions of the URL broken down for testing in conjunction with the isValid function from URLValidator.Java. We used an Agan Principle of *'Divide and Conquer'* here where we set up unit tests individually for the Query, Path, and Authority portions of the URL and put them under test with various known good and bad tests and then set up an oracle to provide us with the results.

Our unit tests were named after which components of the URL we were testing. We have the following unit tests, that can be found under URlValidatorTest.java in the class repository projects/yaof/UrlValidator/src/UrlValidatorTest.java file:

testUrlQuery()
This tests for whether or not UrlValidator successfully detects valid/invalid URLs based on the query component.
testUrlPath()
This tests for whether or not UrlValidator successfully detects valid/invalid URLs based on the path component.
testUrlAuthority()
This tests for whether or not UrlValidator successfully detects valid/invalid URLs based on the authority component.

Collaboration

We decided to split the three areas of testing (manual, input partitioning, unit tests) among the three of us, although we each did a bit of testing in each area. Shaun handled the majority of the manual testing as well as a portion of the unit testing, Robyn completed the input partitioning, and Faye did manual testing.

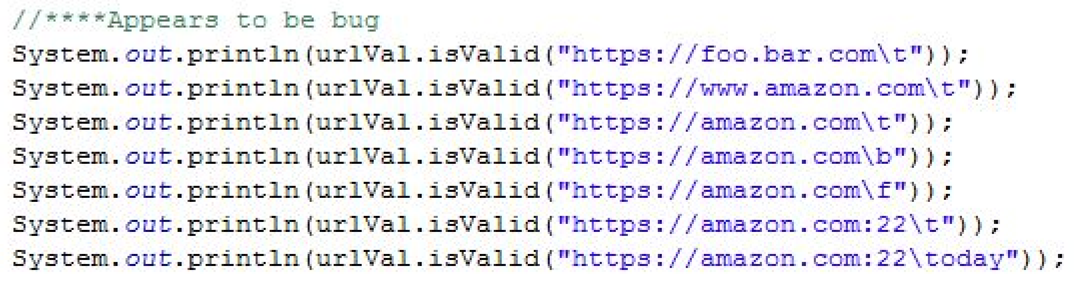
We collaborated mainly through email and Google docs, where each person had their individual section for bug reports/testing. That way, everyone was informed of overall progress on the project. If there were any issues or concerns, we promptly communicated with one another in email to further discuss how to divide the work. Since we split up the work in terms of testing, we were easily able to combine our efforts into a cohesive project report.

Bug Reports

Bug Report #1

What is the Failure?:
For the URLValidator isValid function, if provided with an escape character followed by several typical escape characters t,b,r, or f it is accepted. With the escape character followed by n or any other character string they are properly handled.

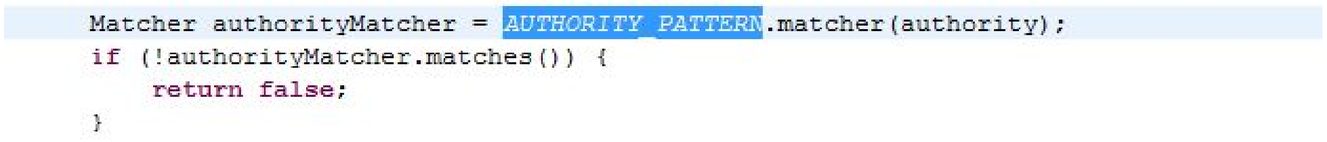
How did you find it?:
This was found through manual testing and with the test of edge cases after both the tld and port authority in the URL per the examples below. For this approach I used one of Agan's principles *'Change One thing at a time'* in order to figure out the subtle error here. Seeing as there were cases where the escape character was properly handled I went through the escape characters one by one and found what select ones were being accepted incorrectly.



```
//****Appears to be bug
System.out.println(urlVal.isValid("https://foo.bar.com\t"));
System.out.println(urlVal.isValid("https://www.amazon.com\t"));
System.out.println(urlVal.isValid("https://amazon.com\t"));
System.out.println(urlVal.isValid("https://amazon.com\b"));
System.out.println(urlVal.isValid("https://amazon.com\f"));
System.out.println(urlVal.isValid("https://amazon.com:22\t"));
System.out.println(urlVal.isValid("https://amazon.com:22\today"));
```

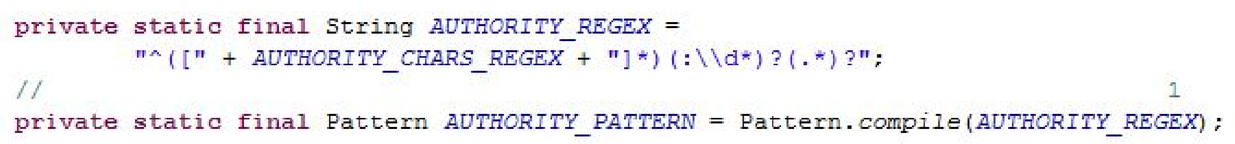
What is the cause of the failure? Explain what part of the code is causing it?:
In URLValidator.Java file, the following code snippets appear to mishandle a particular edge case of the URL Authority Pattern. It appears as though the AUTHORITY_PATTERN from line 372 is handling the input as valid. This appears to be rooted from the AUTHORITY_REGEX evaluation on line 137. I believe the regular expressions here should be corrected to handle this input.

Line 372 of URLValidator.Java



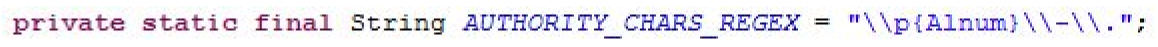
```
    Matcher authorityMatcher = AUTHORITY_PATTERN.matcher(authority);
    if (!authorityMatcher.matches()) {
        return false;
    }
```

Line 134 - 137 of URLValidator.Java



```
private static final String AUTHORITY_REGEX =
        "^([" + AUTHORITY_CHARS_REGEX + "]*)(:\\d*)?(.*)?";
//                                                                    1
private static final Pattern AUTHORITY_PATTERN = Pattern.compile(AUTHORITY_REGEX);
```

Line 102 of URLValidator.Java

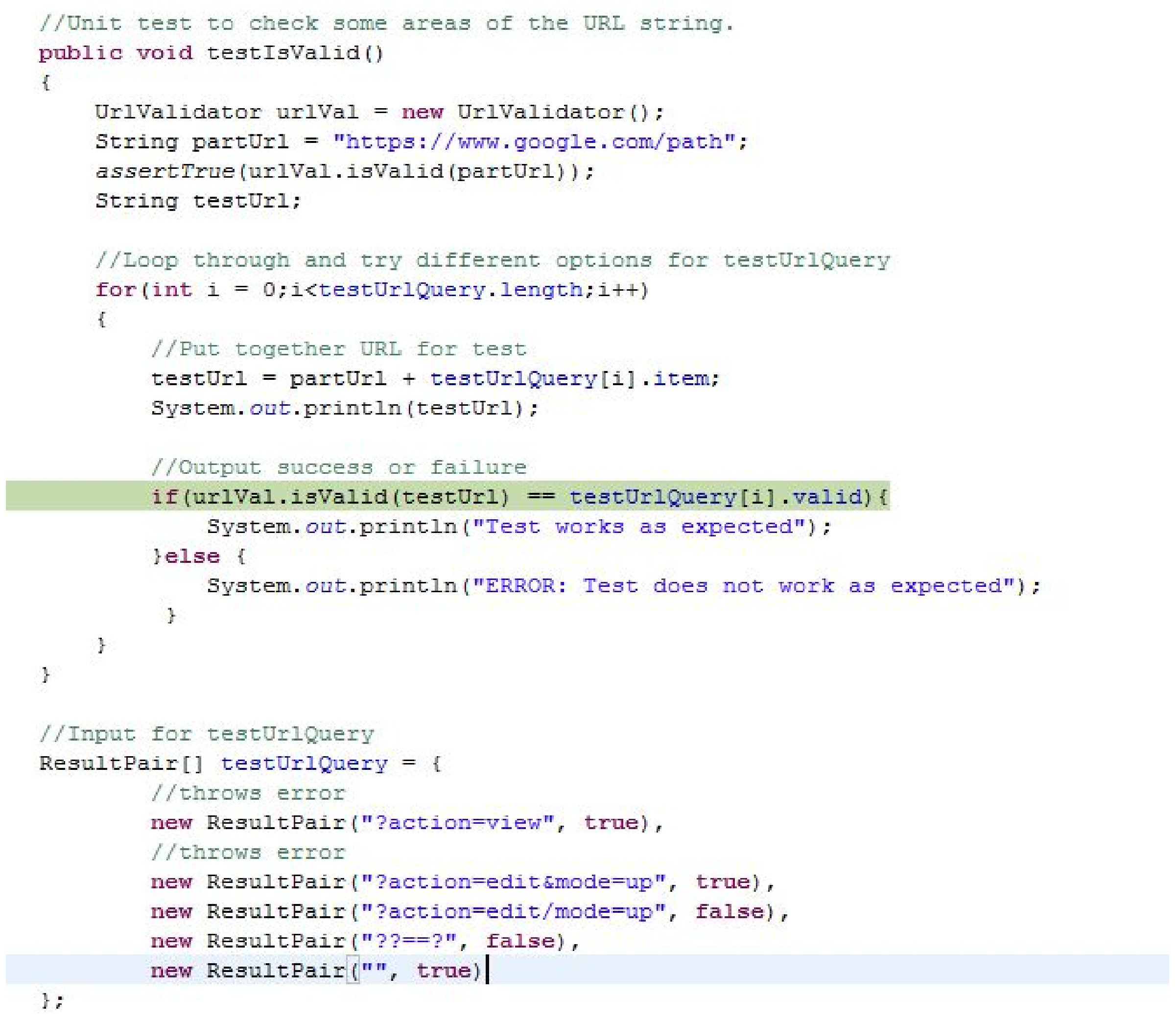


```
private static final String AUTHORITY_CHARS_REGEX = "\\p{Alnum}\\-\\.";
```

Bug Report #2

What is the Failure?:
Whenever valid Url input queries are run through the URL Validator system they do not evaluate to be proper Url's. An example valid input is https://www.google.com/?action=view, which tests out as invalid.

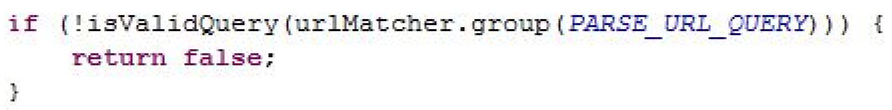
How did you find it?:
This was found through unit testing where I looped through a chosen array of known good and bad Url Query input. I ran these through the isValid function and checked the output against the known answer and found the errors.



```
//Unit test to check some areas of the URL string.
public void testIsValid()
{
    UrlValidator urlVal = new UrlValidator();
    String partUrl = "https://www.google.com/path";
    assertTrue(urlVal.isValid(partUrl));
    String testUrl;

    //Loop through and try different options for testUrlQuery
    for(int i = 0;i<testUrlQuery.length;i++)
    {
        //Put together URL for test
        testUrl = partUrl + testUrlQuery[i].item;
        System.out.println(testUrl);

        //Output success or failure
        if(urlVal.isValid(testUrl) == testUrlQuery[i].valid){
            System.out.println("Test works as expected");
        }else {
            System.out.println("ERROR: Test does not work as expected");
         }
    }
}

//Input for testUrlQuery
ResultPair[] testUrlQuery = {
        //throws error
        new ResultPair("?action=view", true),
        //throws error
        new ResultPair("?action=edit&mode=up", true),
        new ResultPair("?action=edit/mode=up", false),
        new ResultPair("??==?", false),
        new ResultPair("", true)
};
```

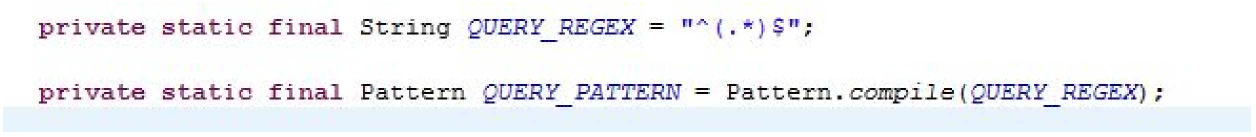
What is the cause of the failure? Explain what part of the code is causing it?:
In URLValidator.Java file, the following code on line 314 appears to mishandle valid query input in the isValidQuery function when called. This appears to call down to line 151-153 as shown below, where I believe the regular expressions here are not evaluating this input correctly.

Line 314 of URLValidator.Java



```
if (!isValidQuery(urlMatcher.group(PARSE_URL_QUERY))) {
    return false;
}
```

Line 151-153 of URLValidator.Java

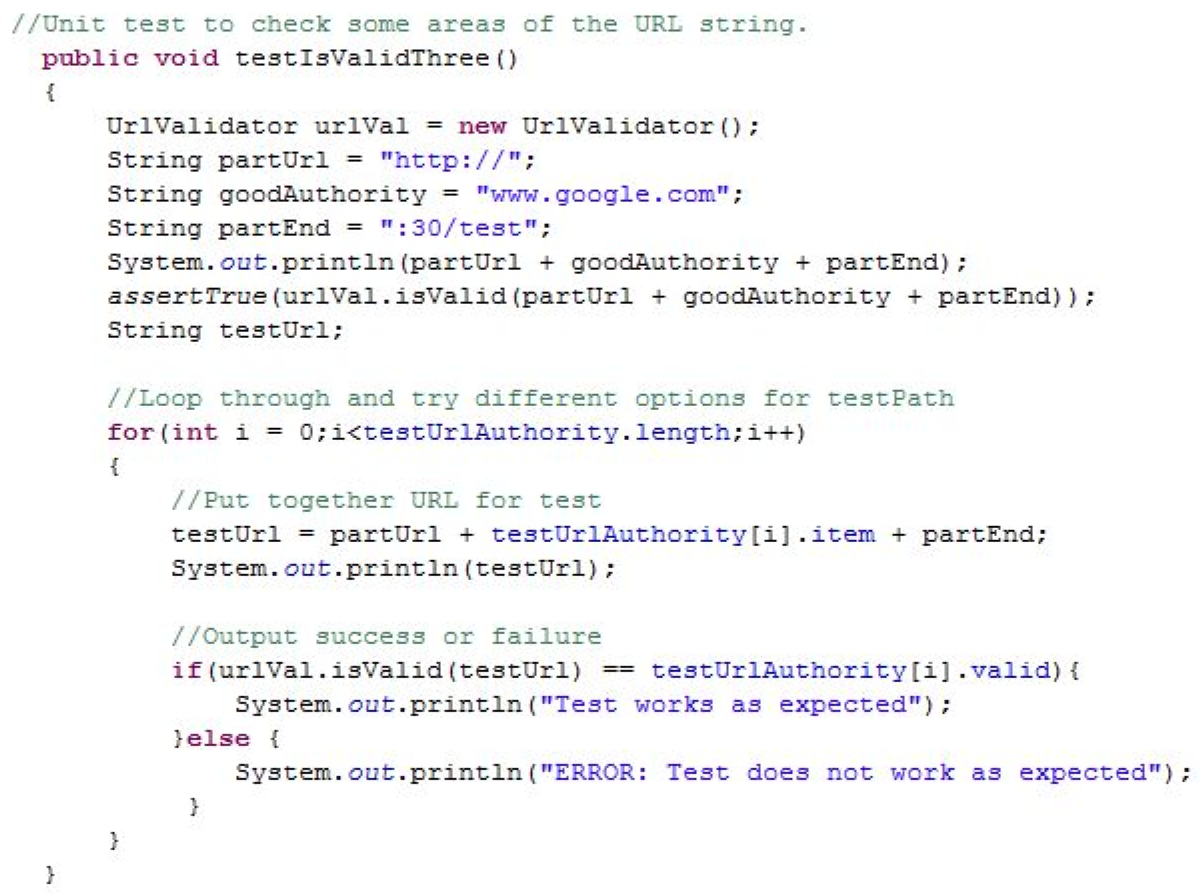


```java
private static final String QUERY_REGEX = "^(.*)$";

private static final Pattern QUERY_PATTERN = Pattern.compile(QUERY_REGEX);
```

Bug Report #3

What is the Failure?:
Whenever an invalid Url IP address is input, the URL Validator system evaluates it as valid. An example invalid input is http://256.256.256.256:30/test, which tests out as valid.

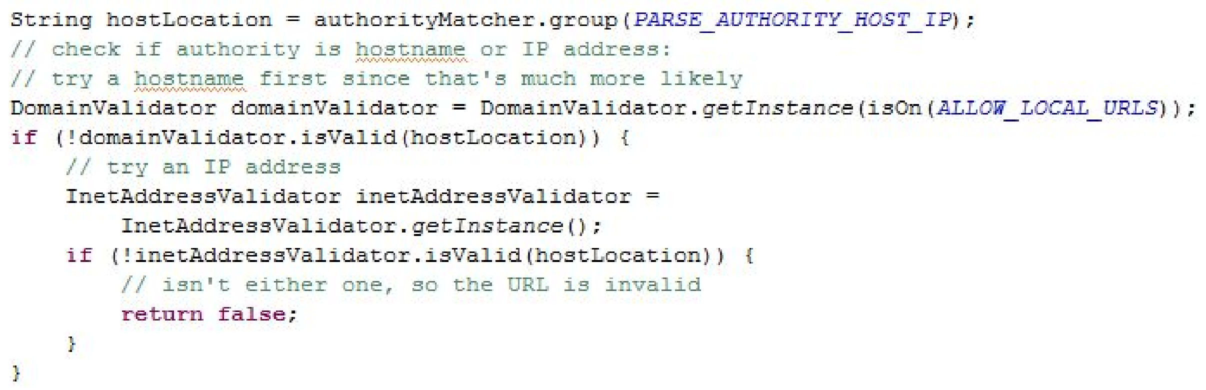
How did you find it?:
This was found through unit testing where I looped through a chosen array of known good and bad Url Authority input. I ran these through the isValid function and checked the output against the known answer and found the errors. For this approach I used one of Agan's principles *'Understand the System'*. My knowledge of the IP address system and byte boundaries allowed me to know that an error wasn't being thrown whenever it should have been. Without the knowledge that an IP address segment cannot be over 255, I would not have been able to understand the error that was taking place.



```java
//Unit test to check some areas of the URL string.
  public void testIsValidThree()
  {
      UrlValidator urlVal = new UrlValidator();
      String partUrl = "http://";
      String goodAuthority = "www.google.com";
      String partEnd = ":30/test";
      System.out.println(partUrl + goodAuthority + partEnd);
      assertTrue(urlVal.isValid(partUrl + goodAuthority + partEnd));
      String testUrl;

      //Loop through and try different options for testPath
      for(int i = 0;i<testUrlAuthority.length;i++)
      {
          //Put together URL for test
          testUrl = partUrl + testUrlAuthority[i].item + partEnd;
          System.out.println(testUrl);

          //Output success or failure
          if(urlVal.isValid(testUrl) == testUrlAuthority[i].valid){
              System.out.println("Test works as expected");
          }else {
              System.out.println("ERROR: Test does not work as expected");
           }
      }
  }
```

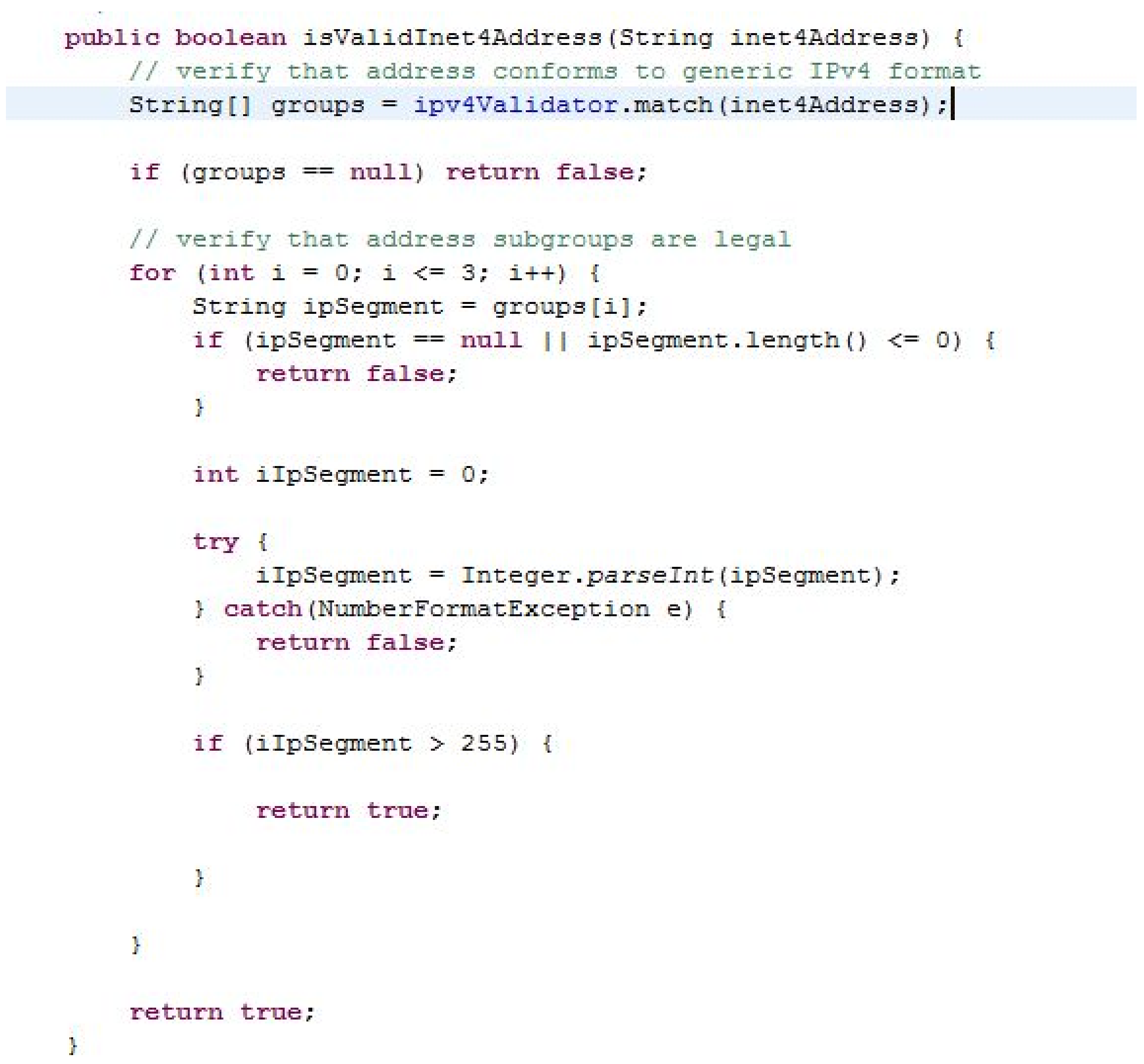
What is the cause of the failure? Explain what part of the code is causing it?:
In URLValidator.Java file, the following code between lines 377 and 390 will evaluate the authority if it is an IP adderss. This eventually calls on a function isValidInet4Address which appears to be the root of the issue. On line 94 of the InetAddressValidator.Java file the IP segments are evaluated to true if the value is higher than 255. This should in fact be evaluating any value above 255 to false. This causes incorrect IP addresses to not be caught.

Line 377-390 of URLValidator.Java



```
String hostLocation = authorityMatcher.group(PARSE_AUTHORITY_HOST_IP);
// check if authority is hostname or IP address:
// try a hostname first since that's much more likely
DomainValidator domainValidator = DomainValidator.getInstance(isOn(ALLOW_LOCAL_URLS));
if (!domainValidator.isValid(hostLocation)) {
    // try an IP address
    InetAddressValidator inetAddressValidator =
        InetAddressValidator.getInstance();
    if (!inetAddressValidator.isValid(hostLocation)) {
        // isn't either one, so the URL is invalid
        return false;
    }
}
```

Line 73-103 of InetAddressValidator.Java



```java
public boolean isValidInet4Address(String inet4Address) {
    // verify that address conforms to generic IPv4 format
    String[] groups = ipv4Validator.match(inet4Address);

    if (groups == null) return false;

    // verify that address subgroups are legal
    for (int i = 0; i <= 3; i++) {
        String ipSegment = groups[i];
        if (ipSegment == null || ipSegment.length() <= 0) {
            return false;
        }

        int iIpSegment = 0;

        try {
            iIpSegment = Integer.parseInt(ipSegment);
        } catch(NumberFormatException e) {
            return false;
        }

        if (iIpSegment > 255) {

            return true;

        }

    }

    return true;
}
```

Bug Report #4

What is the failure?
It appears that any ports of length 4 or greater (eg. 1000, 10000, 40000) are considered false even though this should be a valid URL. Passing ‘http://amazon.com:1000’, ‘http://amazon.com:8000’, or http://amazon.com:8080’, or any port of length 4 or greater is considered false when it should be true.

How did you find it?
I found this during manual testing when I tested various schemes, hosts, paths, and queries. On testing any ports greater than length 4, I found that the URLValidator would return false.

For example:
```
 public void testManualTest()
   {
```

```
            UrlValidator urlVal = new UrlValidator(null, null,
UrlValidator.ALLOW_ALL_SCHEMES);
            System.out.println(urlVal.isValid("http://amazon.com:10000"));
   }
```

I used Agan's principle of *'Make it fail'* in order to find this bug, as I had noticed that any valid URL appended with :8080 would fail. I started testing whether it was a port number of a specific range (this intermittently failed as I tested port numbers ranging from 1 to 10000), and then ports of a specific length. I deduced that the length was what was causing the failure since I could make the URLValidator deem an otherwise valid URL invalid by putting in any port number length greater than 4.

What is the cause of that failure? Explain what part of code is causing it?

On line 158 of UrlValidator.java:

```
private static final String PORT_REGEX = "^:(\\d{1,3})$";
```

This means that the REGEX pattern responsible for detecting correct port number specifications will only match 1-3 port numbers as acceptable. This REGEX pattern should at least accept up to 5 numbers as an acceptable port.

Bug Report #5

What is the failure?

Code in the URLValidator.Java file seems to be mishandling the case when there is no url scheme provided at the very beginning of the url. For example, instead of passing "http://google.com" to the isValid function, I pass "google.com" as a parameter. The isValid function should return true in this case, as users can just type "google.com" into their browser and get routed to the correct webpage. Also, according to the UrlValidatorTest.java file in the URLValidatorCorrect folder, it is not required to include any scheme in the url parameter, as indicated by this line of code: new ResultPair("", true)};.

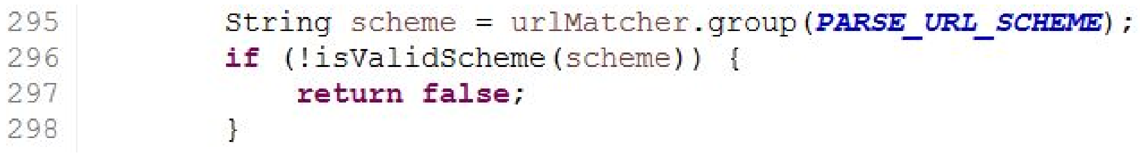
How did you find it?

This bug was caught by the third input partition test (code below); all of the tests in this partition returned false, which is not correct. The lines marked below should have returned true, as leaving out the url scheme should still result in a valid url.

```
 public void testYourThirdPartition(){
            System.out.println("Testing third partition: (partially) empty url inputs");
            UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);
            System.out.println(urlVal.isValid("http://..com"));
            System.out.println(urlVal.isValid("google.com"));-Bug: should have returned true
            System.out.println(urlVal.isValid("http://www.google"));
            System.out.println(urlVal.isValid("go.com:80"));-Bug: should have returned true
            System.out.println(urlVal.isValid("http://go"));
```

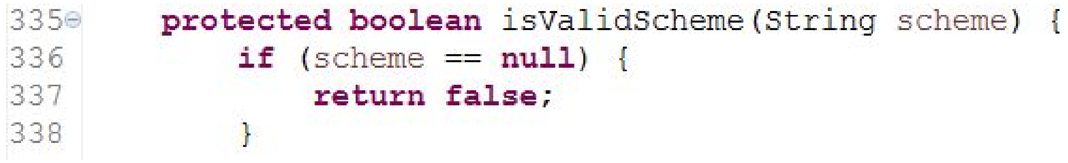
```
        System.out.println(urlVal.isValid("255.com"));-Bug: should have returned true
        System.out.println(urlVal.isValid("http://255"));
        System.out.println(urlVal.isValid("yahoo.com"));-Bug: should have returned true
        System.out.println(urlVal.isValid("http://www.yahoo"));
  }
```

What is the cause of that failure? Explain what part of code is causing it?
In the isValid function in the UrlValidator.java file, line 296 calls the isValidScheme function:



```
295         String scheme = urlMatcher.group(PARSE_URL_SCHEME);
296         if (!isValidScheme(scheme)) {
297             return false;
298         }
```

In the actual isValidScheme function, line 336 says if the scheme is null, the function should return false:



```
335     protected boolean isValidScheme(String scheme) {
336         if (scheme == null) {
337             return false;
338         }
```

Line 336 is incorrect, as the url should be allowed to have a null scheme and still be valid.