Final Project Report

Gregory White
Benjamin Tanner
Zackary Konopa

General

Communication about work for the final project was completed via email. Work was divided up informally between team members and we collaborated on a final project using a shared google document. Overall the team worked together fine as the project requirements were not super specific and people could work on whatever part of the project they wanted. It may have helped that we only had three people in our group.

Manual Testing:

First we generated a list of known valid and invalid URLs. Then we simply iterated over the list calling assert on each url and checking the return value appropriately. We were able to identify two relatively simple errors. The first is where it appears any provided port # that is longer than 3 digits appears to not work. The second is that IPs that include values greater than 255 are incorrectly accepted by the validator.

Some manual tests:

```
System.out.println(urlVal.isValid("http://https://www.amazon.com"));
System.out.println(urlVal.isValid("http://www.ama/zon.com"));
System.out.println(urlVal.isValid("http://wwww.amazon.com:523"));
System.out.println(urlVal.isValid("http://wwww.amazon.com:999"));
System.out.println(urlVal.isValid("http://wwww.amazon.com:1000"));
System.out.println(urlVal.isValid("http://wwww.amazon.com:8695"));
System.out.println(urlVal.isValid("http://amazon.org"));
System.out.println(urlVal.isValid("http://www.a?mazon.com"));
System.out.println(urlVal.isValid("http://www.amazon.com"));
```

Failures:

```
System.out.println(urlVal.isValid("http://wwww.amazon.com:8695"));
System.out.println(urlVal.isValid("http://100.256.100.100"));
```

Partition Testing:

For partition testing we considered each component of the URL (such as the port number and the query string) a partition and tested each individually. For port number partition, we used a consistent url except the port number, which we varied in length and composition. For the port number both valid and invalid values were considered including: empty port , one through six digit port numbers, and port #s consisting of both digits and letters, port #s consisting of only letters. Valid ports are null, or any value between 1 and 65535.

```
 public void testYourFirstPartition(){
        //partitions based on port length
        System.out.println("Testing second Partition!");
        UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);
        System.out.println(urlVal.isValid("http://www.amazon.com:1"));
        System.out.println(urlVal.isValid("http://www.amazon.com:12"));
        System.out.println(urlVal.isValid("http://www.amazon.com:123"));
        System.out.println(urlVal.isValid("http://www.amazon.com:1234"));
        System.out.println(urlVal.isValid("http://www.amazon.com:12345"));
        System.out.println(urlVal.isValid("http://www.amazon.com:123456"));
        System.out.println(urlVal.isValid("http://www.amazon.com:123456"));
        System.out.println(urlVal.isValid("http://www.amazon.com:65535"));
        System.out.println(urlVal.isValid("http://www.amazon.com:65536"));
        System.out.println(urlVal.isValid("http://www.amazon.com"));
        System.out.println(urlVal.isValid("http://www.amazon.com:12a"));
        System.out.println(urlVal.isValid("http://www.amazon.com:b34"));
        System.out.println(urlVal.isValid("http://www.amazon.com:abc"));
        System.out.println("End Testing second Partition!");

  }
```

We also partitioned a bug identified by a group member in the query string portion of the validator code. The bug seemed to result in almost all query strings being validated as false.

```
public void testYourSecondPartition()
 {
        //partition based on query string in URL
        System.out.println("Testing first Partition!");
        UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);
        System.out.println(urlVal.isValid("http://www.amazon.com"));
        System.out.println(urlVal.isValid("http://www.amazon.com?x=1"));
        System.out.println(urlVal.isValid("http://www.amazon.com?xxx"));
        System.out.println(urlVal.isValid("http://www.amazon.com?x"));
        System.out.println(urlVal.isValid("http://www.amazon.com?x\n"));
        System.out.println(urlVal.isValid("http://www.amazon.com?x\r"));
        System.out.println(urlVal.isValid("http://www.amazon.com?x\0"));
        System.out.println(urlVal.isValid("http://www.amazon.com?x\0" + " " + Character.toString ((char) 1)));
        System.out.println("Testing end Partition!");
 }
```

Programming Based Testing:

After identifying bugs in the code with the above two testing methodologies we further explored these areas of concern by programmatically identifying the extent of the bugs. As shown below, using random characters we found that the query string ONLY returned true when the value included a carriage return character (“\n”). A second bug identified above was programmatically diagnosed with a small unit test that applied random port values to the URL string.

```
 public void testIsValid()
   {

            //Programmatically checking for a functioning query string
            System.out.println("Testing Iteratively!");
            Random ran = new Random();
            UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);
            String RandomString = "";
            String RandomString2;
            char letter = ' ';
            int RandomNumber;
            for(int i = 0;i<1000;i++)
            {
                        RandomNumber = ran.nextInt(10) + 1;
                        for (int b=0; b<=RandomNumber; b++) {
                                    letter = (char)(ran.nextInt(127));
                                    RandomString= letter + RandomString;
                         }

                        // RandomString = new BigInteger(RandomNumber, random).toString(32);
              // RandomString = new BigInteger(RandomNumber, random).toString(32);
                        //RandomString2 = new BigInteger(RandomNumber, random).toString(32);
                        System.out.println(urlVal.isValid("http://www.google.com?" + RandomString + "=" + RandomString)) ;
                        RandomString = "";
            }
            System.out.println("End Testing Iteratively!");
   }

   public void testAnyOtherUnitTest()
   {
     //Programmatically checking for a functioning URL port
            System.out.println("Testing Iteratively!");
            SecureRandom random = new SecureRandom();
            UrlValidator urlVal = new UrlValidator(null, null, UrlValidator.ALLOW_ALL_SCHEMES);
            int RandomNumber;
            for(int i = 0;i<10000;i++)
            {
                        RandomNumber = random.nextInt(10000) + 1;
                        System.out.println(urlVal.isValid("http://www.google.com:"+ RandomNumber )) ;
            }
            System.out.println("End Testing Iteratively!");


}
```

Agan’s Principle:

Agan’s principles were employed in the debugging process: In general, simple manual test were conducted first to identify any obvious bugs, then a divide and conquer approach was used to iteratively identify the extent of the found bugs. During debugging, care was taken to separate different bugs into partitions so we didn't run into an issue where a unit test was "tripping" over multiple bugs at once causing a situation where a programmer might be resolving more than one problem at once.

Failures:

Bug report 1

Description: URLs containing valid port numbers longer than 3 digits are reported as invalid URLs
File: URLValidator.java
Code:

```
158     private static final String PORT_REGEX = "^:(\\d{1,3})$";
159     private static final Pattern PORT_PATTERN = Pattern.compile(PORT_REGEX);

391     String port = authorityMatcher.group(PARSE_AUTHORITY_PORT);
392     if (port != null) {
393             if (!PORT_PATTERN.matcher(port).matches()) {
394                     return false;
395             }
```

Making a call to the isValid method and passing in a url with a port that is longer than 3 digits results in the URL being incorrectly reported as invalid.

Debugging Efforts: After tracing down where the function was returning false, I noticed the regular expression on line 158 looked incorrect as it would only accept numbers of up to 3 digits. To test, I changed the regular expression to ^:(\\d{1,4})$ and found that the function then accepted urls with port lengths up to 4 digits. The regular expression should be changed such that it accepts numbers up to 5 digits in length and additional logic should be added to ensure the port is below the maximum port number of 65535.

Example to generate bug:

```
System.out.println(urlVal.isValid("http://wwww.amazon.com:523"));
System.out.println(urlVal.isValid("http://wwww.amazon.com:999"));
System.out.println(urlVal.isValid("http://wwww.amazon.com:1000"));
System.out.println(urlVal.isValid("http://wwww.amazon.com:8695"));
```

Function returns:

```
true
true
false
false
```

Expected results:

```
true
true
true
```

```
true
```

Bug report 2

Description: Authority IP addresses with octets greater than 255 are incorrectly validated as true.
File: InetAddressValidator.java
Code:
In function *public boolean isValidInet4Address(String inet4Address)*

```
94      if (iIpSegment > 255) {
95
96              return true;
97
98      }
```

Called from function *public boolean isValid(String inetAddress)*

```
62      public boolean isValid(String inetAddress) {
63
64              return isValidInet4Address(inetAddress);
65
66      }
```

Called from function *protected boolean isValidAuthority(String authority)* in file *URLValidator.java*

```
383     InetAddressValidator inetAddressValidator =
384             InetAddressValidator.getInstance();
385      if (!inetAddressValidator.isValid(hostLocation)) {
386             // isn't either one, so the URL is invalid
387              return false;
388      }
```

Debugging efforts: I traced the bug down to line 96 in file InetAddressValidator.java which returns true when the the ip segment is greater than 255. This should be changed to return false because octets greater than 255 are not valid ip addresses.

Example to generate bug:

```
System.out.println(urlVal.isValid("http://2.255.2.3"));
System.out.println(urlVal.isValid("http://2.256.2.3"));
System.out.println(urlVal.isValid("http://2.999.2.3"));
```

Function returns:

```
true
true
true
```

Expected results:

```
true
false
false
```

Bug Report 3

Description: URLs with invalid queries are incorrectly validated as true, and URLs with a valid query (and valid otherwise), are incorrectly validated as false.
File: URLValidator.java
Code:
In function *protected boolean isValidQuery(String query)*

```
441    protected boolean isValidQuery(String query) {
442            if (query == null) {
443            return true;
444            }
445
446            return !QUERY_PATTERN.matcher(query).matches();
447    }
```

Called from function *public boolean isValid(String value)*

```
314    if (!isValidQuery(urlMatcher.group(PARSE_URL_QUERY))) {
315            return false;
316    }
```

Debugging efforts: When the QUERY_PATTERN matcher returns the correct value of true or false in the isValidQuery function on line 446, it is negated and returned to the isValid function. This function also negates the returned value and if the result is true, returns false. What we want is for these to match, so the value returned by the matcher only needs to be negated once. This could be corrected in many locations, but the easiest would be to remove the negation in the return statement on line 446.

Example to generate bug:

```
  System.out.println(urlVal.isValid("http://www.amazon.com?x"));
  System.out.println(urlVal.isValid("http://www.amazon.com?x\n"));
System.out.println(urlVal.isValid("http://www.amazon.com?x\r"));
```

Function returns:

```
false
true
true
```

Expected results:

```
true
false
false
```