Ashley Greenacre
Savannah Giese
Kenny Lu
CS362-Final Project

Manual testing: Here are some of the full URLs that we tested: http://www.amazon.com, http://www.google.com, http://www.water.org, http:wer.amazon.com.com, http://www.water.o.o., and http://www.wa.ter.org. When manual testing we found a bug where when we entered http://www.wa.ter.org, it should have reported that it wasn't a valid URL, but the code reported that it was true, that it was valid.

Good Input Partitioning: We tested some parts of a URL separately, like the protocol and port. So that we could test smaller units, to be able to verify where exactly the bug was.

Programming based testing: Then we wrote unit tests for testing parts of the URL. We tested protocol, port, domain name, and path. We tested what we knew were valid inputs and what we knew were invalid inputs. and checked to see what failed and passed our unit tests. From there we could tell where there was a bug, and we could narrow down what was the thing that was actually causing our test to fail.

Function Prototypes:
testValidProtocol() - This function tested for URL protocols that are valid. Upon running this test, all of the test values should return true. If they return false, then there is a bug.
testInvalidProtocol() - This function tested for URL protocols with invalid protocols. Upon running this test, all of the test should return false. If any of these values return true, then there is a bug.
testPorts() - This function tested for URL Ports that were valid. This is where we found the issue of the port number test failing if 1000 or greater, and passing if 999 or less. We tested using a for loop that incremented the port value each time through.
testInvalidPorts() - This function tested for invalid port numbers. Anything out of the suitable integer range (beyond an unsigned short): greater than 65535 is deemed invalid. In addition, all ports that were negative were deemed invalid. If any of these tests yield a true result, then there is a bug.
testValidDomain() - This function tested for valid domains. Using a for loop, we passed in port values from 0 to 65535 to see if any of these test cases failed.
testInvalidDomain() - This function tested for invalid domains. We checked several different sites we knew to be false. This is where we found the bug when there are too many periods.
testValidPath() - This function tested for URL paths that are valid. This is where we found the question mark bug. We tested this valid URL, https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=getty%20images", and it failed our unit test.
testInvalidPath() - This function tested for URL paths that are invalid. Upon running this test, all of the test values should return false, if there is a bug, then a test case will return true.

When testing the ports that were valid, we found a bug where any port value below 1000 passed. But if the port value was 1000 or greater the code failed.
-Line 158, it only accepts values that are 3 digits in length

Also if there is no protocol it should be true, but the URLValidator code reports it as false.

When testing the domains that were valid and invalid, we found a bug where any domain name that didn't have http:// in front of it failed, when it should be true.
Also a valid IP address reports false, when it should report true.

When testing the paths that were valid and invalid, we found a bug where it shouldn't allow periods, but periods in the path report true, that it is valid.

We worked together and met online, and then collaborated using Google docs and Google hangouts, using the screen sharing capabilities. We mainly just worked as a team to get the project done and collaborated on all the sections.

Bug Report 1

Name: The port values of 1000+ do not work.
Type: Bug
Priority: Medium
Description: When testing port values, we found a bug where valid port values caused our unit test, testPorts(), to fail.
-What is the failure?
Any port value below 1000 passed, but if the port value was 1000 or greater the code failed.

-How did you find it?
When testing the ports that were valid, we found a bug where any port value below 1000 passed, like 999 and less. But if the port value was 1000 or greater our unit tests failed.

We found this bug by looking closer into the code. When looking at the port regex, we noticed that that syntax matches at least 1, and at most 3 occurrences of that expression. Therefore it wouldn't match more than three digits. We tried changing that number to a 4 to see if that would work. Trying this caused all of the port values we tested, from 0 - 2000 to pass. Since it did pass, we were able to verify that is where the bug was.

-What is the cause of that failure? Explain what part of code is causing it.
On line 158 in the URLValidator.java code, it only accepts values that are 3 digits in length. Therefore because the number 1000 has 4 digits, it is not considered valid, when it should be.

Bug Report 2
Name: There are too many periods allowed in the domain.
Type: Bug
Priority: Medium

Description: Having two periods in a row, the code is claiming it is a valid url, when it should not be.

-What is the failure? When checking a url that contains something like “google.com/../”, our test case showed that it passes. This should have failed as “..” is not valid in a url.

-How did you find it? When checking our paths, we added a url that contained “/../”. By adding this, and knowing it should return false, we found that there was a bug when our tests showed it to be true. We then started to look in the URLValidator.java file to see if we could find it. We knew our problem was coming from the isValidPath function. Looking into this function further, we saw the dot2Count variable which was where our problem was. It was looking for all “..”, instead of “.”. Changing that line of code and testing again, we saw that our tests had now passed instead of failing like before.

-**What is the cause of that failure? Explain what part of code is causing it.**
On line 428 of URLValidator.java, we find that the line of code failing is int dot2Count = countToken("..", path);. The reason it is failing is because there are two periods in the code, when there should have been one. That means two periods are valid. If we remove one of the periods, the code works.

Bug Report 3
Name: The ? mark symbol that indicates the start of a query does not work.
Type: Bug
Priority: Medium

Description: The verifier doesn’t allow question marks to exist in the URL path. This makes it so that the query functionality in a URL doesn’t work.

-What is the failure? When in the territory of the path in a URL, a question mark symbol signifying the start of a query does not yield a valid URL. A question mark anywhere in the URL path will fail omitting all query functionality.

-How did you find it? We knew that the bug stemmed from one of the URL parsing regular expression validators. The query section of the URL has it’s own verifying functions so we tested to see if the query sections were even being called in the code. Upon seeing that the query section was being called correctly, we then decided to move onto the regular expressions. We first checked to see if the query were being isolated correctly by looking at the URL_REGEX. Everything looked correct here so we looked at where the fault would be most obvious in: the QUERY_REGEX. With some help on google about regular expressions and how they work in Java, we found that there was a wildcard character that was out of place in this part of the code. This wildcard character alone made it impossible for question marks to exist in the URL path leading to our bug.

-What is the cause of that failure? Explain what part of code is causing it. The cause of the failure was the result of an extra ("wild") wildcard character in the QUERY_REGEX regular expression checker on line 153. Instead of QUERY_REGEX = "^(.)$";, our buggy code looked like QUERY_REGEX = "^(.*)$"; with the red star character signifying the extra wildcard character.