Broken Link Checker v9.2 - 12/12/16

Developed by Josh Agda | Idea by Devaraj Bojan

# Summary

A Java program that crawls through a given URL and scans their response codes and take screenshots (if enabled), then outputs its results into an Excel file. Developed using standard Java API, threads, JSwing, JSoup, Selenium, and Apache POI.

# Purpose & Value

The purpose of this program is to quickly detect all broken/invalid links and their sources in a given site as well as any appearance or browser related defects (through screenshots) for later fixing.

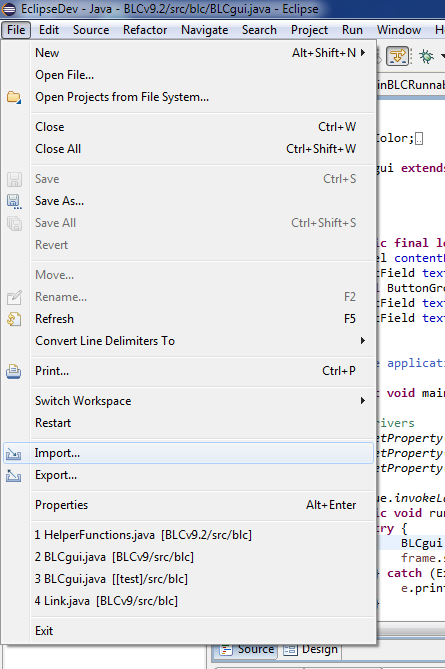
# The value of this program is that it greatly speeds up the QA process to find any defects related to links or appearance in a website. (This is especially useful when creating a new web environment.)

# How to Install

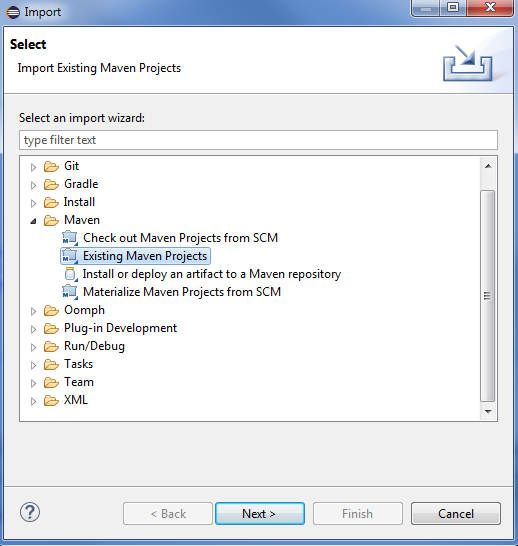
The installation process for this program is simply getting the project files and unzipping them on to your computer.

If you want to be able to use this on the Eclipse IDE there are several more steps to take:

* Run Eclipse
* Go to File->Import



* Click on Maven->Existing Maven Projects->Next



* Browse and select the unzipped project folder, then finish

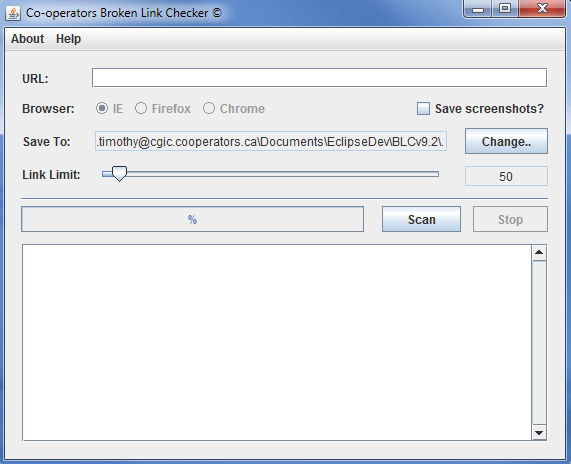
# How to Run

To start the program, you can either:

* Run the JAR file inside the project folder
* Run the project through Eclipse (F11)

(If the program doesn’t start or gives you an error in Eclipse, you might not have the right Java JRE version on your computer. That means you will have to upgrade your JRE to at least 1.8 to use this.)

After starting the program, a small GUI will pop up.



Here you can select what URL, which browser (for screenshots), where to save the results to, and how many unique links to scan for. Simply fill in the URL and change any other of the fields to your preference, then click the *Scan* button to start the scan.

It will go through several phases and will output the current status and results onto the textbox below and the progress bar just above that. If you wish to stop a scan, you will have to click on the *Stop* button and wait a short while before starting another scan.

There are some things to point out about this program:

* The scan starts from the URL given and *branches out*, therefore starting from *http://www.cooperators.ca/en/Investments/Understanding-Investments* will only branch out and look for links from */Understanding-Investments* out
* Please **remove** the suffix when entering the URL to allow for branching (i.e. remove .html, .aspx, etc.)
* To scan for secured websites, make sure that the URL starts with *https://*
* The program takes screenshots by opening the browser(s) to the web page and taking a screenshot. Please don’t touch the browsers while this is happening
* If result row is missing a *Link Text* value, then it may be hidden inside a picture or just secretly excluded in the HTML code
* If IE screenshots is initially **not working**, go to *Internet Options*->*Security* and *Enable* *Protected Mode* in all sections
* Make sure that your IE zoom is 100% before doing screenshots in IE

Notes for Developers

# Program Capabilities

* Runs on JRE version 1.8+ (**IMPORTANT**)
* Maven project made in Eclipse
* Operates using an internet connection
* Uses JSoup to parse document
* Uses threads (runnable)
* Creates new folders and lets user choose where to save results
* Outputs to an Excel file
* Uses a JSwing UI
* Can stop mid-scan
* Takes screenshots using Selenium
* Works well with **small – medium** sized websites. Please do not use on extremely large websites (the program will run out of memory)

# General Program Flow

1. Program starts
2. GUI opens
3. User fills in details
4. User starts scan (scan button disabled and stop button enabled)
5. Main thread (runnable) starts
6. Link scraping threads are created (threads to find links) by main thread
7. Link scraping threads start scanning webpages
8. Link scraping threads save results to temp file (URL, Origin URL, Link Text)
9. Main thread reads temp file
10. Main thread splits results into multiple smaller lists
11. Response code threads are created by main thread with the lists
12. Response code threads start scanning given list
13. Response code threads save results into second temp file (Title, Response Code)
14. Main Thread reads second temp file
15. Main Thread combines results from both temp files
16. Main Thread creates an Excel document using the results
17. Main Thread deletes temp files
18. Current scan is completed (scan button enabled and stop button disabled)

(**Note**: If the stop button was pressed anytime between any of these steps, it will wait until the step is completed and then stop the scan)

# Links to Code References

* <http://www.software-testing-tutorials-automation.com/2015/08/how-to-find-broken-linksimages-from.html>
* <http://stackoverflow.com/questions/1201048/allowing-java-to-use-an-untrusted-certificate-for-ssl-https-connection>
* <http://stackoverflow.com/questions/2895342/java-how-can-i-split-an-arraylist-in-multiple-small-arraylists>
* <https://developer.android.com/reference/java/util/concurrent/CountDownLatch.html>
* <http://www.java2s.com/Code/Java/Swing-JFC/SelectadirectorywithaJFileChooser.htm>
* <http://stackoverflow.com/questions/19540289/how-to-fix-the-java-security-cert-certificateexception-no-subject-alternative>
* <http://poi.apache.org/spreadsheet/quick-guide.html>
* <http://stackoverflow.com/questions/6273221/open-a-text-file-in-the-default-text-editor-via-java>