# Crawler documentation

Contents

[Crawler documentation 1](#_Toc19089660)

[Unit Tests 2](#_Toc19089661)

[Class Diagram 3](#_Toc19089662)

[Linter 3](#_Toc19089663)

[Sitemap 3](#_Toc19089664)

[JavaDoc 4](#_Toc19089665)

[Log 4](#_Toc19089666)

[Pseudocode 4](#_Toc19089667)

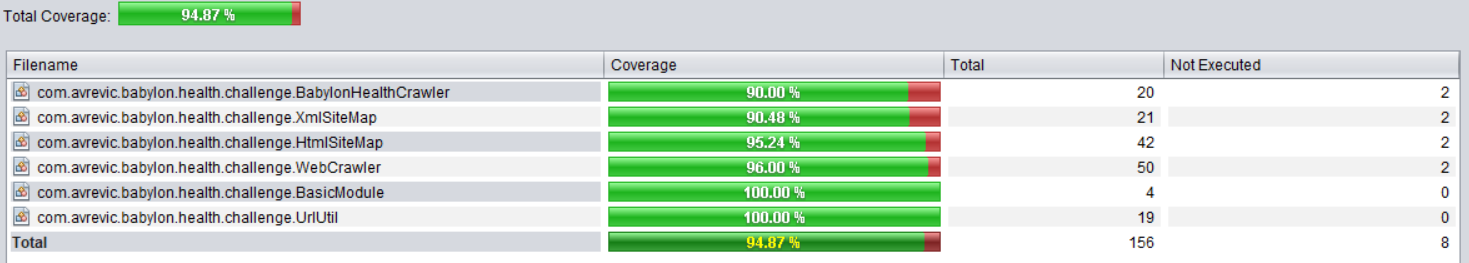
[Performance considerations 4](#_Toc19089668)

[Crawler traps 5](#_Toc19089669)

[Libraries and Software used for development/testing/doc 5](#_Toc19089670)

## Unit Tests

In total there are 18 unit tests that have 94.87% of code coverage. A couple of exceptions handlers are not unit tested, otherwise the coverage would be 100%. I have developed the code using TDD approach. Screenshot of code coverage is below. I have also generated test report file target/jacoco.exec which can be imported into Sonar Qube.

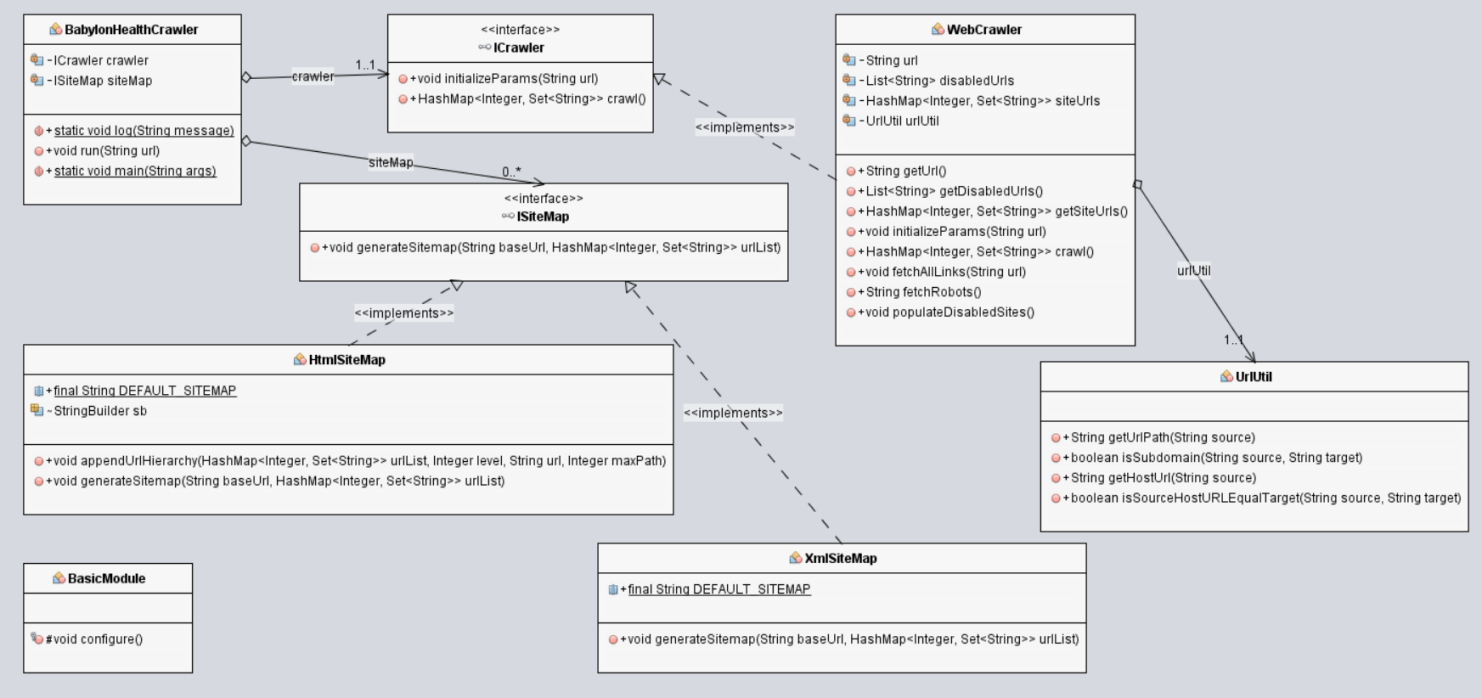


To test the crawler and sitemap generator logic, I have created the following local files:

* src/test/index.html – HTML page
* src/test/page2.html – HTML page
* vendor – HTML page
* robots.txt
* sitemap-test.html – HTML sitemap
* sitemap-test.xml – XML sitemap

HTML pages are served inside local Docker nginx container so that they can be accessed using localhost. Docker commands can be found in src/test/docker-build-commands.txt. Dockerfile is in the src/test/Dockerfile. These HTML pages are served by Docker and then used by tests to check if the crawler generated data is valid. Sitemap files are also produced from the local sites and checked if they match with the data generated from the sitemap and crawler classes.

## Class Diagram



There are 6 classes and 2 interfaces:

* BabylonHealthCrawler class – main class for the program that calls the crawler
* BasicModule class – DI support class
* UrlUtil class – helper class for URL parsing
* Icrawler interface – interface for crawler
* IsiteMap interface – interface for sitemap
* XmlSiteMap – implementation that generates XML sitemap
* HtmlSiteMap – implementation that generates HTML sitemap – this sitemap is easier to read for a person than XML sitemap and it contains URL hierarchy

## Linter

* No code linter was used

## Sitemap

* When the program is run, two types of sitemaps are generated:
  + XML Sitemap – machine readable format
  + HTML Sitemap – human readable format – also contains hierarchical representation of the data

Sitemaps are stored in the output folder.

## JavaDoc

JavaDoc is generated and can be found in target/site/apidocs folder. Open the index.html file first.

## Log

Only sysout logs were used, and there aren't plenty of them. For the future iterations, a more persistent type of log could be used, with VERBOSE/DEBUG/ERROR options.

## Pseudocode

1. Try to fetch the robots file
2. If robots file is found, then create a list of urls that shouldn't be crawled
3. Store the current link as root link
4. Call the fetch links method with the current link as param
5. Check if the link is already crawled, external or is disabled by robots
   * + If any of the above, then return the execution to the parent method
     + If none of the above, then:
     + Fetch the hierarchy level of the link by parsing the url path
     + Store the url in the url list with the correct hierarchy
     + Fetch all the links from the current url
     + Iterate through all the fetched links and
       - Store the current link as root link
       - Call the step 5

## Performance considerations

This program is limited by two factors regarding performance:

1. Fetching the site content to parse the href links – network issue
2. Parsing the html files for href content

A larger limitation is on 1) as it is affected by the host and client network speed, however it can potentially yield a better performance if the url fetching and parsing is multithreaded and if the parsing starts while the URL data is still fetched:

http://tutorials.jenkov.com/java-howto/java-web-crawler.html)

## Crawler traps

They can mostly be avoided by correctly following the robots file guidance, however AOPIC algorithm can be used to ensure further crawler robustness:

<https://stackoverflow.com/questions/5834808/designing-a-web-crawler>

## Libraries and Software used for development/testing/doc

* NetBeans
* JRE, JDK 8
* easyUML NetBeans plugin – for Class diagram, although initial diagram before implementation was created in LucidChart
* Windows 10 (including Bash for windows for git CLI)
* TikiOne JaCoCoverage NetBeans plugin – used to generate and preview code coverage reports
* Junit – testing
* Com.google.inject – Dependency injection annotations
* Guice – dependency injection support classes
* Commons-lang – String utils helper functions (trim etc.)
* Jsoup – url data fetching
* JaCoCo – code coverage
* Maven – dependency resolution and build support
* Git – Version control
* GitHub – private repository management