Installation Instructors for Crawler4j

The following installation instructions should work for both Unix and Windows. As Crawler4j is a Java program it is useful to make sure you first have the latest version of Java Development Kit (JDK) and Java's run-time environment (JRE) installed. Following that you should download and install the Eclipse Software Development Environment.

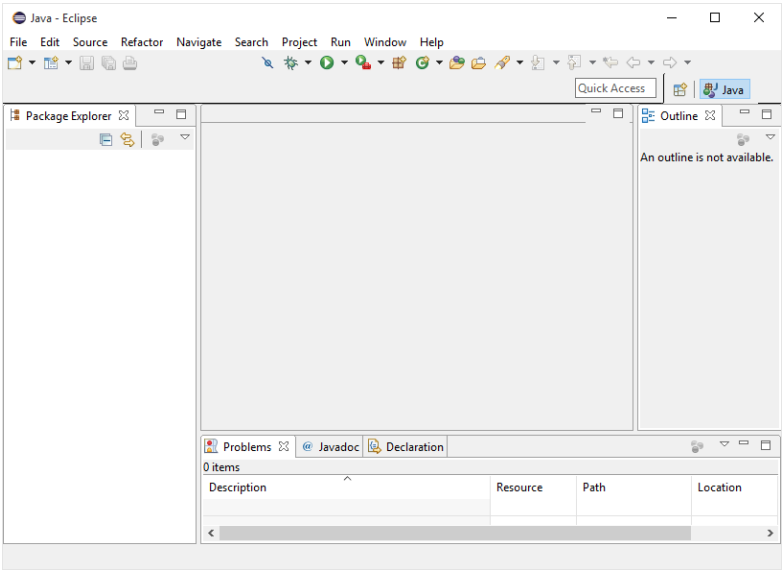
Eclipse Installation

Eclipse is an open source community, whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle.

The Eclipse Project was originally created by IBM in November 2001 and supported by a consortium of software vendors. The Eclipse Top-Level Project is an open source, robust, full-featured, commercial-quality, industry platform for the development of highly integrated tools and rich client applications. The home page of the Eclipse Foundation is located at:

http://www.eclipse.org/

Download and install Eclipse. These instructions were created for the Mars version of Eclipse. For windows users, first you need to install Java on your local machine. You can download JDK from http://www.oracle.com/technetwork/java/javase/downloads/index.html.



The initial screen should look like the one above. After installing the JDK, you need to add Windows environment variables for JDK.

- Properties -> Advanced -> Environment Variables -> System variables -> New Variable Name: JAVA_HOME, Variable Value: <Full path to the JDK>
- Typically, this full path looks something like C:\Program Files\Java\jdkx.x.x. Then modify the PATH variable as follows on Microsoft Windows: C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Java\jdkx.x.x\bin
  This path may vary depending on your installation.
- Note: The PATH environment variable is a series of directories separated by semicolons (;) and is not case-sensitive. Microsoft Windows looks for programs in the PATH directories in order, from left to right. You should only have one bin directory for a JDK in the path at a time. Those following the first instance are ignored. If you are not sure where to add the path, add it to the right of the value of the PATH variable. The new path takes effect in each new command window you open after setting the PATH variable.

- Reboot your computer and type “java –version” in the terminal to see whether your JDK has been installed correctly.

For Mac OS X users and Linux users, you already have Java and the JDK installed with the OS. Older version may need to install both the JDK and the JRE. After installing JDK, you can download Eclipse from http://www.eclipse.org/downloads/. Any one of the latest versions should be fine.

If you are using the Eclipse Installer, select Eclipse IDE for Java Developers.

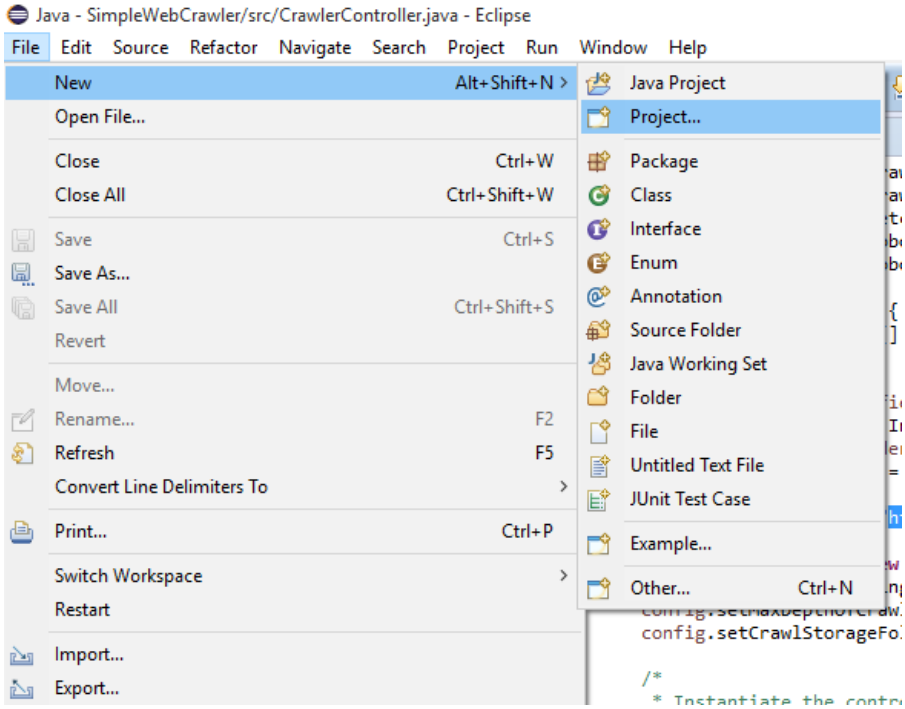
Crawler4j Installation

crawler4j is an open source web crawler for Java which provides a simple interface for crawling the Web. Using it, you can setup a multi-threaded web crawler in few minutes.

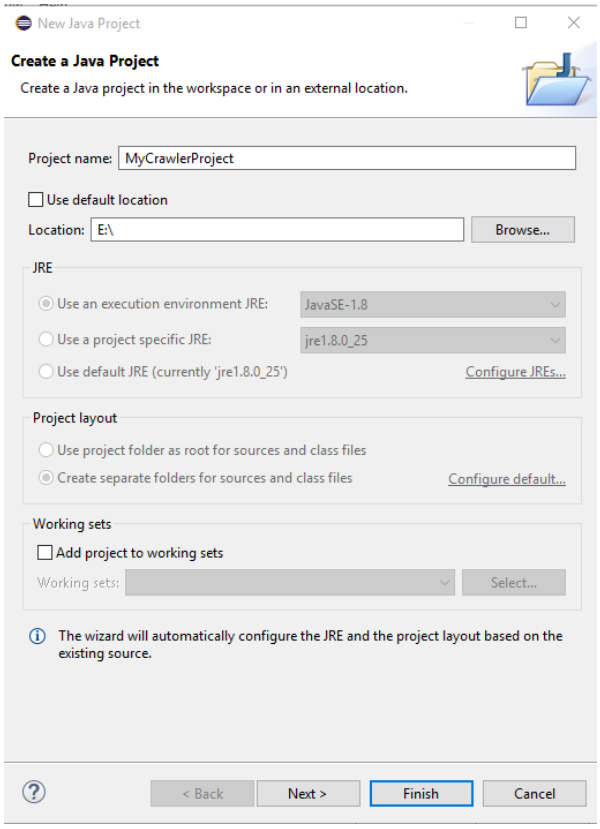
1. Download latest crawler4j-x.x-jar-with-dependencies.jar where x represents the current revision major and minor numbers from https://github.com/yasserg/crawler4j/releases

2. In Eclipse add the JAR file to the class path.

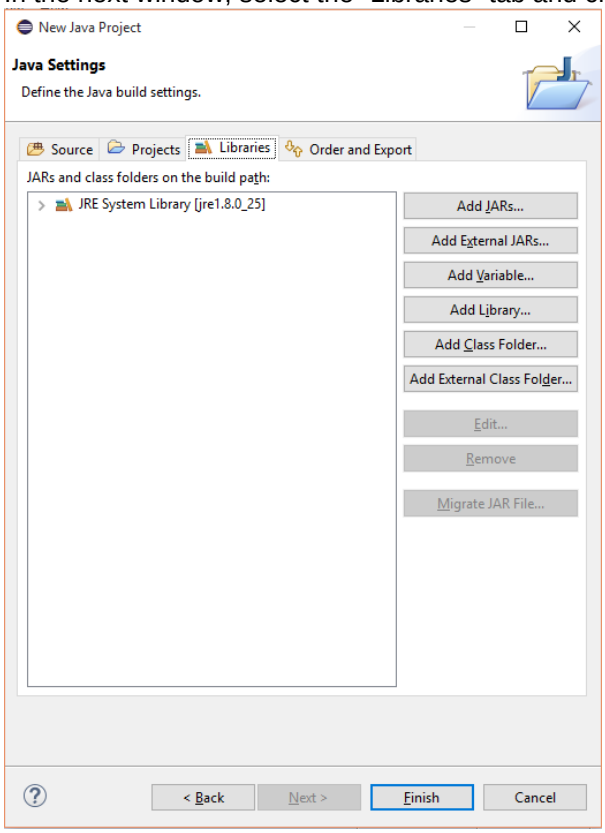
i. In Eclipse, create a new Java project. Goto File -> New ->Java Project.



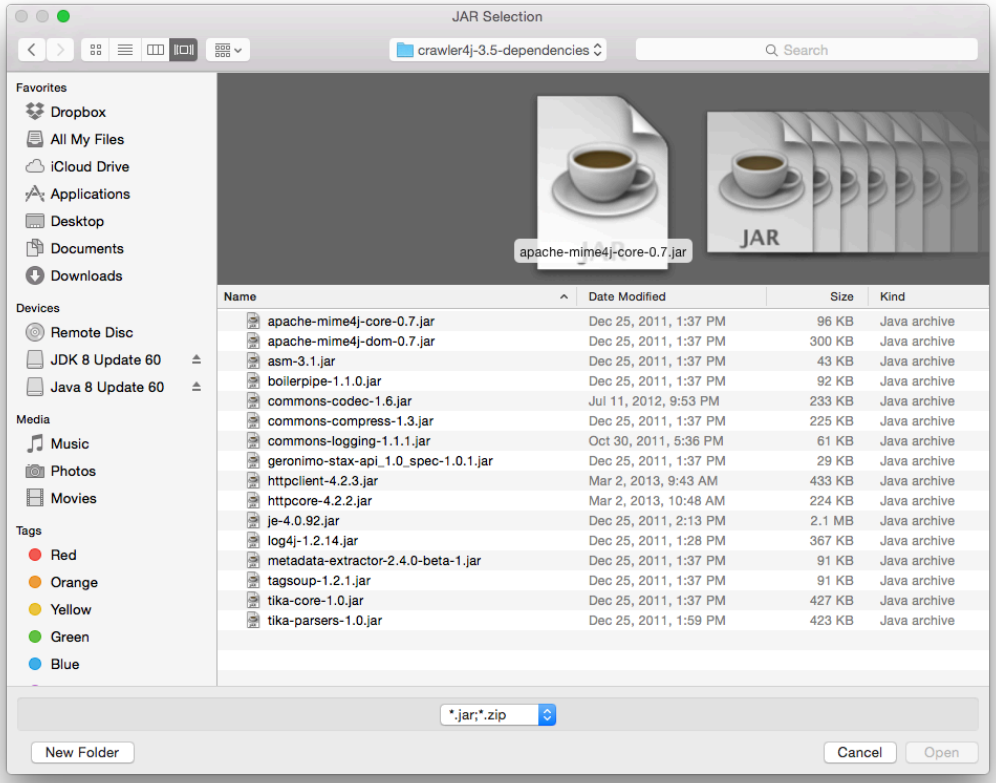
ii. Enter Project name, and click on next. Enter the project name you wish to use. You can probably use all of the default settings.

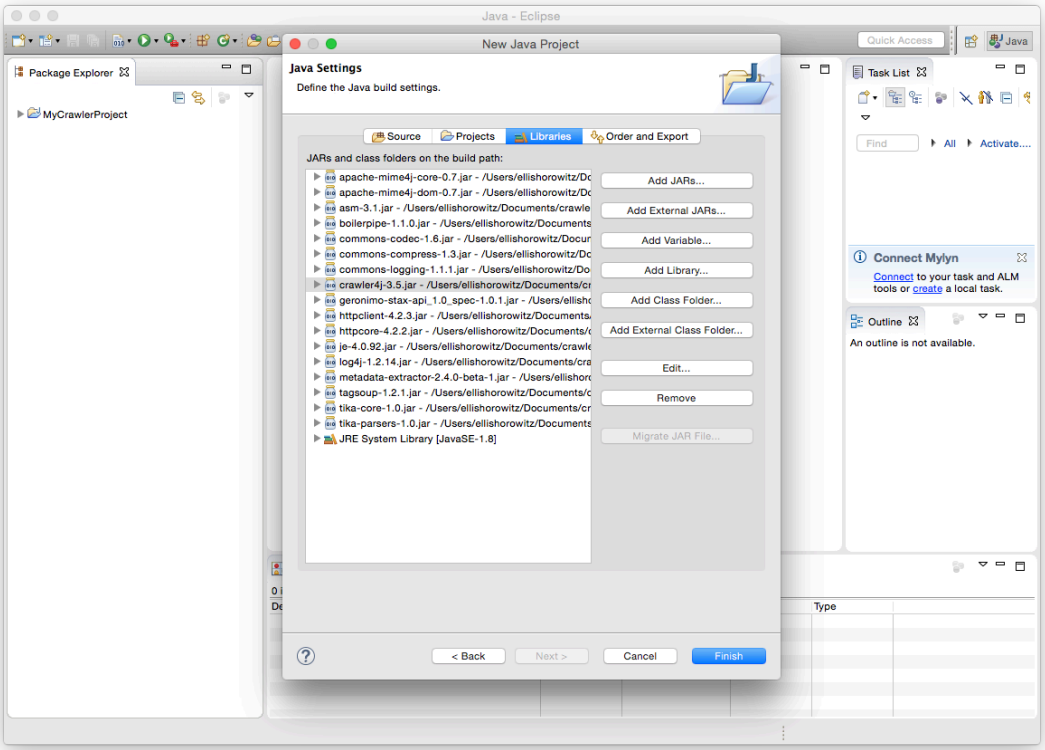


iii. In the next window, select the “Libraries” tab and click on Add External JARs… option.

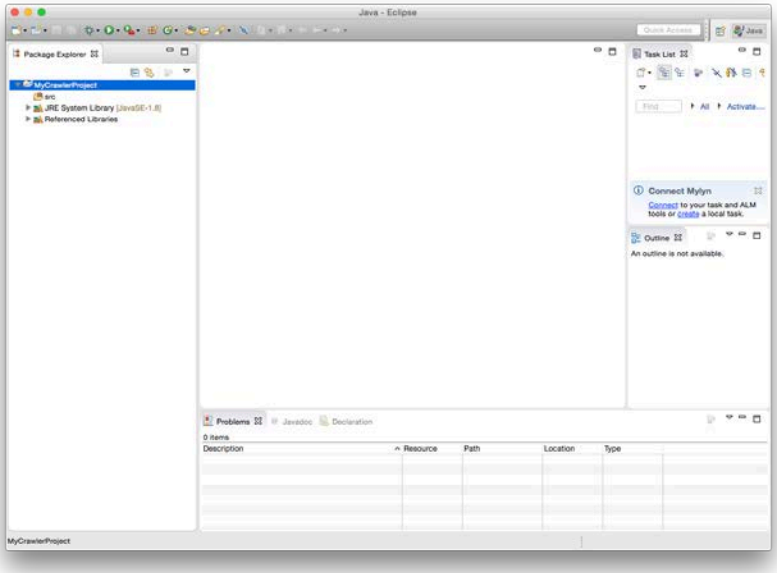


iv. Select the JAR that you had downloaded previously in the File Browser that pops up. Once that is done, you will see the JAR added in the Libraries tab mentioned in step iii.



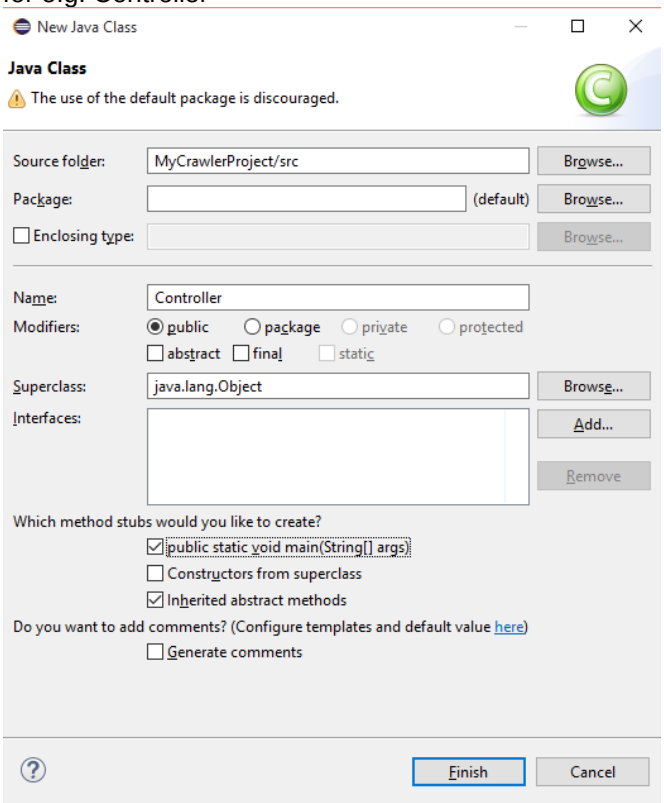


v. Click on Finish. In the Project Explorer window, you will see a new Project created, with Referenced Libraries containing the JAR you have added.



You can add more JARs/external libraries in the Project in the following way. Right click on your new web crawler project and select the Build Path option. Select the “Add External JARs…” option and select the jar file that you extracted from the zip files.

3. You should also implement a controller class which specifies the seeds (starting URLs) of the crawl, the folder in which intermediate crawl data should be stored and the number of concurrent threads, and other configurations dealing with politeness. Right click on the **src** folder and select New->Class... In the wizard, enter the class name, for e.g. Controller



The result is a `public class Controller` with a `public static void main()` and nothing else. Below is a sample code that you would have to write in order to configure the controller and define the seed URLs. The code below will save downloaded files in /data/crawl and there is only a single seed, http://viterbi.usc.edu/

```
public class Controller {
    public static void main(String[] args) throws Exception {
        String crawlStorageFolder = "/data/crawl";
        int numberOfCrawlers = 7;

        CrawlConfig config = new CrawlConfig();
        config.setCrawlStorageFolder(crawlStorageFolder);

        /*
         * Instantiate the controller for this crawl.
         */
        PageFetcher pageFetcher = new PageFetcher(config);
        RobotstxtConfig robotstxtConfig = new RobotstxtConfig();
        RobotstxtServer robotstxtServer = new RobotstxtServer(robotstxtConfig, pageFetcher);
        CrawlController controller = new CrawlController(config, pageFetcher, robotstxtServer);

        /*
         * For each crawl, you need to add some seed urls. These are the first
         * URLs that are fetched and then the crawler starts following links
         * which are found in these pages
         */
        controller.addSeed("http://www.viterbi.usc.edu/”);
        /*
         * Start the crawl. This is a blocking operation, meaning that your code
```

```
         * will reach the line after this only when crawling is finished.
         */
        controller.start(MyCrawler.class, numberOfCrawlers);
    }
}
```

The controller class has a mandatory parameter of type CrawlConfig. Instances of this class can be used for configuring crawler4j. These are some of the crawl configurations that you can customize.

i. Crawl depth – There is no depth for crawling by default. However, you can limit the crawl depth by specifying this parameter in the CrawlConfig object.
E.g. A is the seed URL, and if A->B->C->D is the link structure. Setting a crawl depth of 2, will result in crawling/visiting of pages A (depth 0), B (depth 1) and C (depth 2) and will avoid crawling page D.

```
crawlConfig.setMaxDepthOfCrawling(maxDepthOfCrawling);
```

ii. Maximum number of pages to crawl – To limit the number of pages that are fetched and stored. By default there is no such limit.

```
crawlConfig.setMaxPagesToFetch(maxPagesToFetch);
```

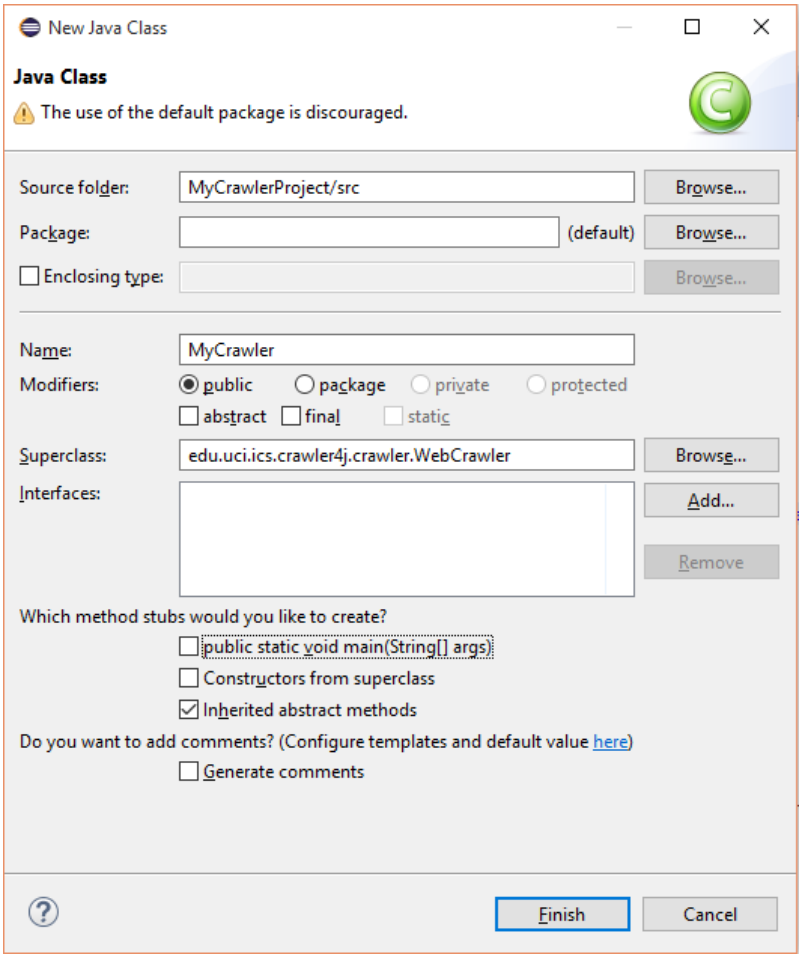
iii. Politeness – Crawling puts a huge load on servers that you are trying to crawl. If you bombard the server with multiple requests in a short duration of time, they will block you. It is really important to crawl politely, and not disrupt the services provided by the server. By default, crawler4j waits for at least 200ms between requests, but you might want to increase this duration. However you should not make it very large, as your crawl might take forever to finish.

```
crawlConfig.setPolitenessDelay(politenessDelay);
```

iv. User Agent – This is part of the politeness policy as well, identifying yourself to the server you are crawling.

```
crawlConfig.setUserAgentString(userAgentString);
```

4. A second class is needed, one that extends the WebCrawler class. This class decides which URLs should be crawled, any filtering you may want to implement and handles the downloaded pages. In the New Class wizard, select WebCrawler as the superclass by Clicking on Superclass-Browse and searching for WebCrawler. Click on OK. Next provide a name for the class and click on Finish.




```
public class MyCrawler extends WebCrawler {
```

5. Two methods should be overridden in this class
   i. shouldVisit: This function decides whether the given URL should be crawled or not. In the following example, this example is not allowing .css, .js and media files and only allows pages within 'www.viterbi.usc.edu' domain.

```
public class MyCrawler extends WebCrawler {

    private final static Pattern FILTERS = Pattern.compile(".*(\\.(css|js|gif|jpg"
                                                           + "|png|mp3|mp3|zip|gz))$");

    /**
     * This method receives two parameters. The first parameter is the page
     * in which we have discovered this new url and the second parameter is
     * the new url. You should implement this function to specify whether
     * the given url should be crawled or not (based on your crawling logic).
     * In this example, we are instructing the crawler to ignore urls that
     * have css, js, git, ... extensions and to only accept urls that start
     * with "http://www.viterbi.usc.edu/". In this case, we didn't need the
     * referringPage parameter to make the decision.
     */
    @Override
    public boolean shouldVisit(Page referringPage, WebURL url) {
        String href = url.getURL().toLowerCase();
```

```
            return !FILTERS.matcher(href).matches()
                    && href.startsWith("http://www.viterbi.usc.edu/");
    }
```

ii. visit: This function is called after the content of a URL is downloaded successfully. You can easily get the URL, text, links, html, and unique id of the downloaded page.

```
/**
     * This function is called when a page is fetched and ready
     * to be processed by your program.
     */
    @Override
    public void visit(Page page) {
        String url = page.getWebURL().getURL();
        System.out.println("URL: " + url);

        if (page.getParseData() instanceof HtmlParseData) {
            HtmlParseData htmlParseData = (HtmlParseData) page.getParseData();
            String text = htmlParseData.getText();
            String html = htmlParseData.getHtml();
            Set<WebURL> links = htmlParseData.getOutgoingUrls();

            System.out.println("Text length: " + text.length());
            System.out.println("Html length: " + html.length());
            System.out.println("Number of outgoing links: " + links.size());
        }
    }
```

6. Once this is done, you can run the Controller class and your basic crawler will be running. Once your crawler finishes crawling, you will see these messages.

```
INFO [Thread-1] It looks like no thread is working, waiting for 10 seconds to make sure...
INFO [Thread-1] No thread is working and no more URLs are in queue waiting for another 10 seconds to make sure...
INFO [Thread-1] All of the crawlers are stopped. Finishing the process...
INFO [Thread-1] Waiting for 10 seconds before final clean up...
```

For this exercise you will have to make more modifications to collect the output that is required.