**CMSC 132 -- Project 8**

**Threaded Web Crawler / Slideshow**

## Closed Policy

**This is a closed project.** You are expected to do all of the work on this project without consulting with anyone other than the CMSC 132 instructors and TAs.  Treat this project as though it were a take home exam.  If you have any questions about whether or not a certain topic is okay to discuss with classmates or friends, please ask your instructor.

## Goals

The goals of this project are:

* Practice Threads and concurrency

## Overview

Essentially, this project is a web-crawler, which automatically explores the web by following links from one web page to another to another to another, etc.  It performs something similar to a breadth-first-search.  As it explores, it maintains a list of any pictures that it encounters.  These pictures are slowly displayed in a "slideshow" window at regular intervals.

The project maintains two primary data structures, which will be accessed concurrently by numerous threads:

* linkQueue -- A queue of URL's that represent web-pages that have been discovered, but not yet visited.
* picQueue -- A queue of URL's that represent pictures that have been discovered, but not yet displayed in the slideshow.

There are two other data structures being maintained, both sets:

* beenThere -- a set containing every webpage URL that has ever been put into the linkQueue
* doneThat -- a set containing every picture URL that has ever been put into the picQueue

These two sets are maintained so that we can ensure that we never put duplicate copies of webpages or pictures into the queues.  (That would be boring!)

These data structures start off empty.

## Threads

**There will be four types of threads all operating at the same time:**

1. **CrawlerGUI** -- the CrawlerGUI appears below.  It serves two purposes.  First, it allows the program to get started by having the user put the very first website into the linkQueue.  (That's what happens when you press the "GO" button.)  Also, it allows you to monitor the progress of the whole system.  It looks like this:  
   

The fields in the GUI labelled "Links Discovered", "Pics Discovered", "Links in Queue", and "Pics in Queue" represent the sizes of the four data structures beenThere, doneThat, linkQueue, and picQueue, respectively.Pressing the "GO" button activates a method that waits for any existing ExtractorThreads to die off, empties the four data structures (linkQueue, picQueue, beenThere, and doneThat), and puts the URL that the user has typed into both linkQueue and beenThere.

1. **SlideshowGUI** -- a window that will constantly pull pictures out of the picQueue and display them.  Here is an image representing the SlideshowGUI at some point during a typical run of the program:
2. **ExtractorThread** -- Many of these will be running at the same time.  (There is a fixed size array of ExtractorThreads, with a default size of  5.  The array starts off with 5 null values, but the Crawler will soon begin creating ExtractorThreads and putting them into the array.)  The constructor of the ExtractorThread is passed a URL representing an html webpage that the extractor is supposed to explore.  By "exploring", we mean the extractor will go through the web page looking for pictures and links to other web pages.  Any URL's representing pictures that it finds it puts into the picQueue; any URL's representing web pages that it finds it puts into the linkQueue.  The ExtractorThread dies when it has scanned the entire page it was created to explore.
3. **Crawler** -- The crawler contains the main method, which will run in a thread that never ends. The crawler begins by starting up the SlideshowGUI (pictured above) and the CrawlerGUI (pictured above).  Then the crawler constantly cycles through the array of ExtractorThreads, one-by-one.  If the current array entry refers to an ExtractorThread that is still alive, the Crawler moves on to the next element of the array.  If the current array entry is null or refers to an ExtractorThread that has died, then the Crawler pulls a webpage out of the linkQueue, starts a new ExtractorThread that will explore this webpage, and puts this new ExtractorThread into the array in the current position.  Then the crawler moves on to the next position in the array.  Once it has gone through the whole array, it starts again from the beginning.

## What you will Implement

Much of the project is written for you, but you will write a good deal of it.  Details follow.

### MyQueue class

You must implement a queue designed to be accessed by numerous threads concurrently.  **You may wrap one of the existing Java collection classes, but you may not use Collections.synchronizedList() or something similar.** You must write code that ensures that there will be no data races or other problems when many threads each perform a single operation on the queue.  Below is a description of the methods which you must provide:

* public int size() -- returns the number of items in the queue
* public void clear() -- removes all items from the queue
* public void enqueue(T o) -- adds the object to one end of the queue
* public T dequeue() -- removes an object from the end opposite of that to which things are added.  If the queue is empty, the thread will wait here until an item becomes available

### MySet class

You must implement a set designed to be accessed by numerous threads concurrently.  **You may wrap one of the existing Java collection classes, but you may not use Collections.synchronizedSet() or something similar.** You must write code that ensures that there will be no data races or other problems when many threads each perform a single operation on the set.  Below is a description of the methods which you must provide:

* public int size() -- returns the number of items in the set
* public void clear() -- removes all items from the set
* public boolean remove(T o) -- removes the object from the set
* public boolean add(T o) -- adds the object to the set
* public boolean contains(T o) -- returns true if the set contains the object, false otherwise

### ExtractorThread class

Please begin by looking at the portion of this class that has already been provided for you. Note that the constructor for the ExtractorThread takes 5 parameters: The URL that this thread will explore, and references to the four data structures that are shared by all threads in the project.

You will find two static methods have been provided:

1. private static Set getLinks(String s, URL currentURL)  
   If you pass this method an html String (and the URL where that string was found), it will return to you a set of URL's that represent links to other webpages found in the String.
2. private static Set getPicURLs(String s, URL currentURL)  
   if you pass this method an html Sttring (and the URL where that string was found), it will return to you a set of URL's that represent links to images found in the String.

You will complete the *run* method for the Thread, as follows.  **Be careful that the code you write will not cause any data races!**

**1.  Extracting Links**

Open an InputStream to the url that is associated with the ExtractorThread. You may find it useful to wrap the InputStream so as to obtain a BufferedReader.

Go through the entire contents of the webpage you are now connected to, reading in one line at a time. For each line, call the method getLinks, passing it the current line along with the url of the webpage you are currently exploring. This method will parse the string, and will return to you a list of links to new webpages that were found. For each URL that is returned, do the following:

Check that your new URL is not null, and then get it's protocol (a String).  If the protocol is null or anything other than "http" or "file", then we don't want it, so skip it.  If the protocol looks okay, then check whether or not this URL is already in the set beenThere.  If it isn't in the set yet, then add this URL to the linkQueue and also to the beenThere set.

**2.  Extracting Pictures**

Start over again scanning the webpage from the top.  This time you will search for pictures by calling the method getPicURLs for each line in the web page. This method will parse the string, and will return to you a list of links to pictures that were found. For each URL that is returned, do the following:

Check if this URL is already contained in the doneThat set.  If not, then add it to the doneThat set and also to the picQueue.

### Crawler class

Please look at the .java file to see what is already provided.  You must fill in the rest of the main() method.  The method must continually cycle through all elements of the *extractors* array.  When you have processed the whole array, have the thread sleep for a short period of time to allow the CPU to process other threads.  [I used  *Thread.sleep(500).]* After the short wait, start over and process the entire array again!  This should go in indefinitely.  For each element in the array, you should perform the following steps:

* If the entry in the array is an ExtractorThread that is "alive" (yes, there is an instance method for threads named "isAlive"), then you should do nothing and just move on to the next element.
* Once you have found an entry that is *null* or contains a dead ExtractorThread, get a lock on the linkQueue and check if it is empty.  If it is empty, *wait* on the linkQueue.  (Your thread will be notified when an object is added to the linkQueue, assuming you have written the linkQueue correctly!)  If you are curious as to why we are doing this step, try removing it and running the program.  Can you figure out why it is behaving badly without this step?
* Release the lock on the linkQueue.
* Acquire a lock on the extractors array.
* Take a URL out of the linkQueue.  Open a URLConnection object on the URL.  Get the "content type" of the URLConnection (this is a String).  If it is *null* or doesn't begin with the sub-string "text/html", then this URL is not a good one, so take another URL out of the linkQueue and repeat this step.
* Once you've obtained a good URL, instantiate an ExtractorThread for it, assign this thread to the dead/unused entry in the array, and start the thread.
* Release your lock on the extractors array.

### CrawlerGUI class

This class has been provided for you.

### SlideShowGUI class

This class has been provided for you.

## Grading

Your grade on this project will be calculated as follows.  THIS IS SUBJECT TO CHANGE IF WE ENCOUNTER PROBLEMS WITH THE AUTOMATED TESTING SYSTEM.

* Release tests (MySet, MyQueue, ExtractorThread)  -- 80%
* Crawler tests (looked at by human eyes) -- 20%

[Web Accessibility](https://www.umd.edu/web-accessibility)