HTTP, Threading Assignment (web crawler)

Re-submit Assignment

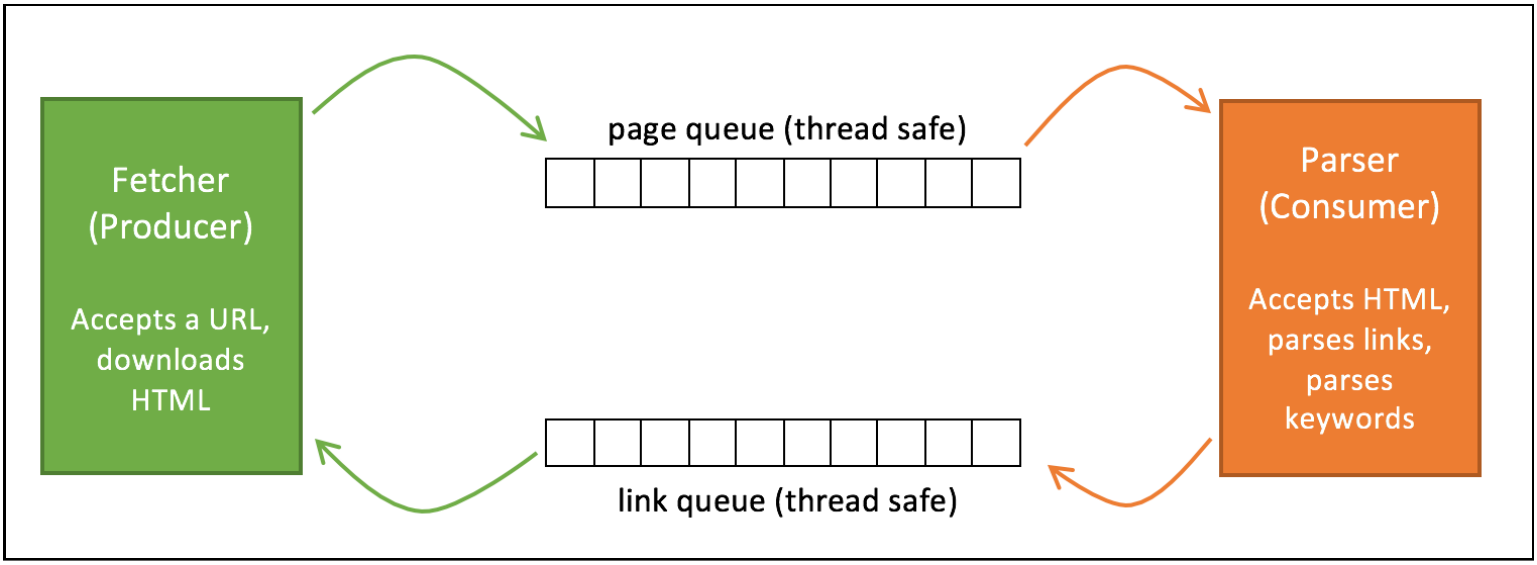
Due Mar 6, 2017 by 11:59pm **Points** 150 **Submitting** a file upload **File Types** zip

Overview

For this assignment we will focus on client/server programming, HTTP connections and concurrency. Our goal is to building a simple web crawler. The purpose of a web crawler is to traverse the links on a network (in this case the internet) and collect statistics on each page encountered. Google maintains web crawlers which look for relevant pages to index and show as search engine results. More about this process can be read here: https://en.wikipedia.org/wiki/Web_crawler (https://en.wikipedia.org/wiki/Web_crawler).

Producers and Consumers

We will be creating producer and consumer threads to solve this problem. Producer threads will be responsible for downloading web pages found on the world-wide-web, while consumers will analyze each page found. This interaction will follow the producer-consumer pattern as described in class. A diagram of thread interactions can be seen below:



Producer Threads

A producer thread (Fetcher) should do the following repeatedly (ie. in an infinite loop):

- pull a link from the link queue
- download the (HTML) page text at the given URL
- store the (HTML) page text on the page queue as a string

A BufferedReader object can be used to read from the remote page using the following code sample:

```
HttpURLConnection connection = (HttpURLConnection)url.openConnection();
BufferedReader download = new BufferedReader(new InputStreamReader(connection.getInputStream()));
```

Note: you will still need to write a loop that pulls each line from the remote file over HTTP. You should then build a single string value that contains the entire contents of the remote file.

Consumer Threads

A consumer thread (Parser) should do the following repeatedly (ie. in an infinite loop):

- pull a page from the page queue
- search the page for all links in anchor (<a href="">) elements
- add each link found to the link queue
- search the page for any keywords specified by the user of the web crawler (more on this later)
- keep track of how many keywords are encountered

Parsing a web document for links can be a difficult task. Regular expressions are a pattern recognition language that is used match complex patterns in text. A good tutorial on the subject can be found at: **http://www.regular-expressions.info/tutorial.html** (http://www.regular-expressions.info/tutorial.html). Below I have provided a short code segment that will match most URLs found in anchor tags on a page. The loop will execute for each link found:

```
Pattern pattern = Pattern.compile("href=\"(http:.*?)\"");
Matcher matcher = pattern.matcher(pageText);

while (matcher.find())
{
    String link = matcher.group(1);

    //do something with link...
}
```

Note: you will cover regular expressions in depth at a later point in your classes.

To determine how many times a keyword shows up in a document, you can use the String.split(delimiter) method on your page text. This method will return an array with elements located on each side of a delimiter given. For example, given the string "hello world how are you?" and the delimiter "o", the String.split() method will give the following output:

```
String testString = "hello world how are you?";
String[] parts = testString.split("o");

for (int i = 0; i < parts.length; i++)
{
    System.out.println(parts[i]);
}

*****************************************************

hell
 w
rld h
w are y
u?
```

Using the String.split(keyword) will give you the text on each side of a keyword found in a document. These are stored in an array that is returned from the split() method. This functionality should be enough to determine how many times the keyword was found in a file (more on this below).

Shared Page Queue

The shared page queue should store a linked list of strings, each of which has the text of a web document. Your queue should have at least the following methods:

- public void addPage(String pageText)
  - Adds a new page to the queue. The queue should have a maximum size of 50000. If there is no room on the queue, then the thread should call wait() on the queue.
  - After adding a new page to the queue, you should call notify() on the queue.
- public String getNextPage()
  - Returns a page from the queue. If the queue is empty, then the thread should call wait() on the queue.
  - After removing a page from the queue, you should call notify() on the queue before returning the page text.
- public int getPagesDownloaded()
  - This method returns the total number of pages that have been added to the queue through the addPage() method.

Note: All interactions with this class should be thread safe.

Shared Link Queue

The shared link queue should store a linked list of strings, each of which has a URL for a web document. Your queue shoudl have at least the following methods:

- public void addLink(String url)
  - Adds a link to the queue if it has not yet been seen. This class should store a HashSet<String> of string URLs that have been seen so far. You should use to HashSet object to ensure that a link has not been added more than once.
  - The queue should have a maximum size of 50000. If there is no room on the queue, then the thread should call wait() on the queue.
  - After adding a new link to the queue, you should call notify() on the queue.
- public String getNextLink()
  - Returns a link from the queue. If the queue is empty, then the thread should call wait() on the queue.
  - After removing a link from the queue, you should call notify() on the queue before returning the URL.
- public int getLinksFound()
  - This method returns the total number of unique links found so far through the use of the addLink() method.

Web Crawler Console Program

Write a console program that presents your users with the following menu options:

```
1. Add seed url
2. Add consumer
3. Add producer
4. Add keyword search
5. Print stats
```

Add seed url

For this option you should prompt the user for a url, which is then added to the link queue.

Add consumer

Creates and starts a new Parse (consumer) thread. If there are no pages on the page queue, this will result in the thread going to sleep with a called to wait().

Add producer

Creates and starts a new Fetch (producer) thread. If there are no links on the link queue, this will result in the thread going to sleep with a called to wait().

Add keyword search

This menu option creates a new keyword that consumer threads can look for when parsing documents. Your program should maintain a List of keywords that each consumer can access.

Print stats

Choosing this menu option will print out the following details of the web crawler:

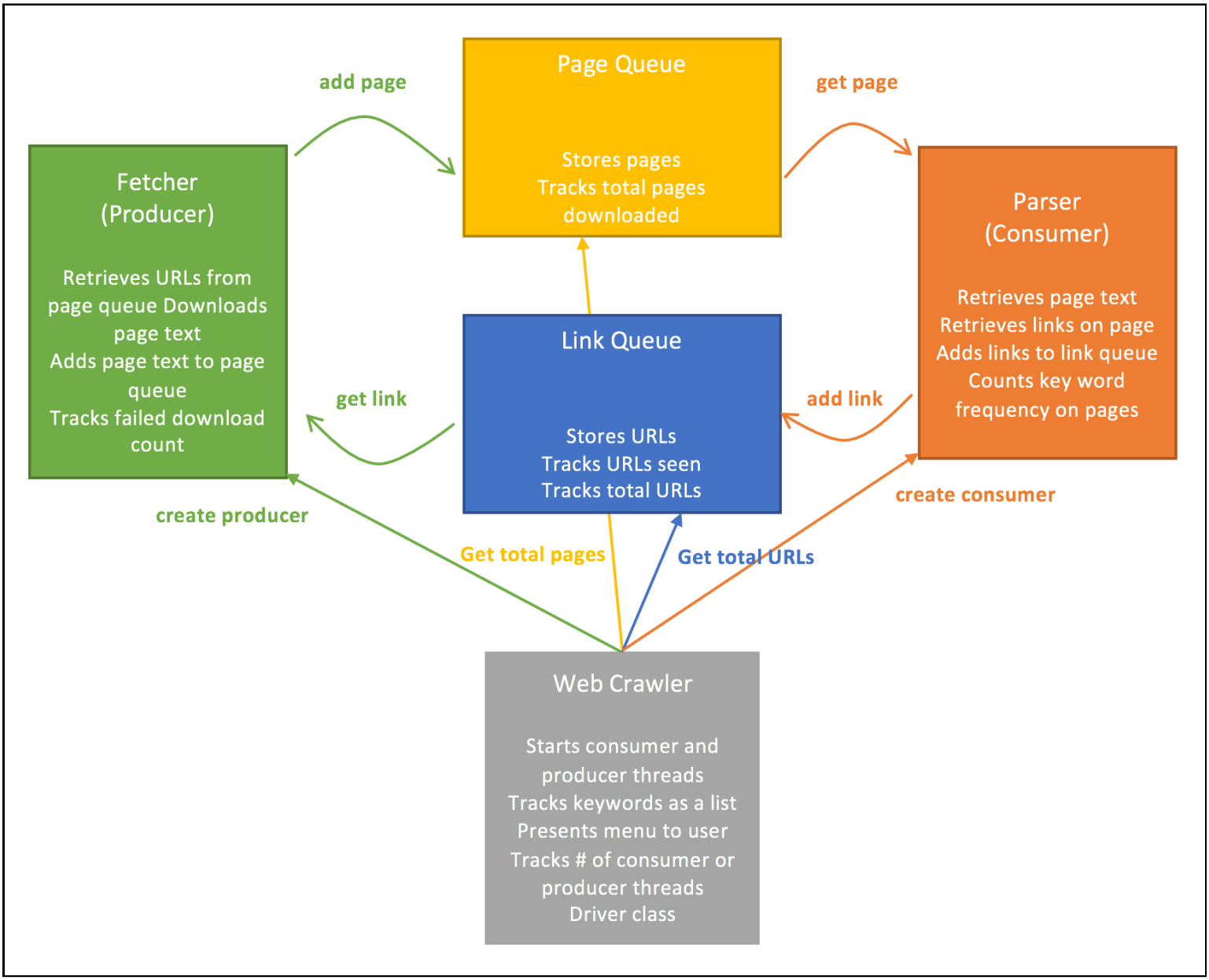
```
Keywords found: 557
Links found: 140
Pages found: 50
Failed downloads: 4
Producers: 100
Consumers: 10
```

Note:

- A list of keywords should be available to your consumer threads. You will need to keep track of the total number of keywords found among all consumer threads.
  - Keep in mind that different keywords can be found on any page.
- The number of links found should be accessible from your Link Queue class.
- The number of pages found should be accessible from your Page Queue class.
- The number of failed downloads should be recorded for all producer threads. A download fails when creation of an HttpURLConnection results in an exception (ie. you should be counting each time you enter the catch in your try-catch block in your producer class.
- The number of producer threads started should be tracked and available.
- The number of consumer threads started should be tracked and available.

Component Design

One possible class design can be viewed below, but this is just a suggestion:



Testing Your Program and Submission

- Try running your program with different seed urls, number of producers and number of consumers. Try to see how to maximize your CPU through your choices.

- Test your program thoroughly. Be ready to share your results with the class.
- You may work on this program in pairs using the pair programming style.