```
import java.util.Map;
import java.util.concurrent.atomic.AtomicInteger;
import java.io.IOException;
import java.net.URL;
import java.net.MalformedURLException;
import java.util.concurrent.*;
import java.util.concurrent.ConcurrentHashMap;
import java.util.PriorityQueue;
import java.util.Comparator;

/**
 * This class is a multithreaded web crawler that takes in an initial URL and
 * prints out the top 10 most occurring URLs from that base URL.
 * This class does not go beyond the URLs with the base of the root URL.
 *
 * @param args[0] URL to open
 *
 * @throws IOException on network error
 * @throws MalformedURLException if given URL is not valid
 */
public class WebCrawler {

    private static final int MAX_THREADS = 50;

    public static void main(String[] args) throws MalformedURLException,
            IOException {

        // Create the thread pool
        ExecutorService service = Executors.newFixedThreadPool(MAX_THREADS);
        // The command line argument and root URL
        URL initialURL = null;
        // A concurrent map that holds the URL's that have been explored, and
        // the number of time they've been called.
        Map<URL, AtomicInteger> visitedURLs = new ConcurrentHashMap
                <URL, AtomicInteger>();
        // Keeps track of the number of threads in the thread pool/thread queue
        AtomicInteger threadCount = new AtomicInteger(0);
        // This is the queue used to print out the results in defending order
        PriorityQueue<Map.Entry<URL, AtomicInteger>> URL_PriorityQueue =
                new PriorityQueue<>(new CompareMapEntries());

        // Test for given arguments
        if (args.length < 1) {
            System.err.println("No URL given\nUsage: WebCrawler <URL>\n");
            System.exit(1);
        }

        // Test to see if valid URL
        try {
            initialURL = new URL(args[0]);
        }
        catch (MalformedURLException m) {
            System.err.println("Invalid URL: '"+args[0]+"'\n");
            System.exit(1);
        }

        // Create the initial thread
        service.submit(new Task(visitedURLs,
                initialURL, initialURL, threadCount, service));

        // Wait to shutdown
        try {
            service.awaitTermination(Long.MAX_VALUE, TimeUnit.DAYS);
        }
        catch (InterruptedException e) { }

        // Adds the map entries into a priority queue using a custom comparator.
        for (Map.Entry<URL, AtomicInteger> entry : visitedURLs.entrySet()) {
            URL_PriorityQueue.add(entry);
        }

        // Print out the top 10 entries
        for (int i = 0; i < 10; i++) {
            System.out.print(URL_PriorityQueue.peek().getKey() + " ");
            System.out.println(URL_PriorityQueue.poll().getValue());
        }
    }
}

/**
 * This class implements runnable and takes in the thread information.
 * This information is used to create more threads, edit the data structures,
 * and decrement/increment the number of threads.
 */
class Task implements Runnable {
[redacted]
}

@Override
public void run() {
    // Where count is the reference to the object in the map
    AtomicInteger currentValue = visitedURLs.putIfAbsent(currentURL,
            new AtomicInteger(1));

    // Not null means that the value exists already and increment
    if (currentValue != null) {
        currentValue.incrementAndGet();
    }
    // Else find the URL's in the path
    else {
        try {
            for (URL childURL : new HTMLLinks(currentURL)) {
[redacted]
                }
            }
            // This catches MalformedURLException error as well as any
            // other IO exception thrown
            catch (IOException e) {
                System.err.println("Invalid URL");
            }
        }
        if (threadCount.decrementAndGet() == 0) {
            service.shutdown();
        }
    }
}

/**
 * This is a custom Comparator to compare a Map entry data type to another's
 * Atomic Integer
 */
class CompareMapEntries implements Comparator<Map.Entry<URL, AtomicInteger>> {

    public int compare(Map.Entry<URL, AtomicInteger> s1, Map.Entry<URL,
            AtomicInteger> s2) {
        // If the entries are equal
        int returnValue = 0;

        int firstValue = s1.getValue().get();
        int secondValue = s2.getValue().get();

        if (firstValue < secondValue) {
            returnValue = 1;
        }
        else if (firstValue > secondValue) {
            returnValue = -1;
        }
        return returnValue;
    }
}
```