Where does all that spam come from?

A group exercise for Programming in Java

Birkbeck, University of London

1 Introduction

It is very common these days to receive tons of unsolicited email, also known as *spam*. These emails are usually sent to your mailbox by computer applications in an automated fashion. These applications are called *spambots* and they get your email address (and millions of others) by harvesting them from electronic material that is published on the web.

Spambots spawn agent-programs called web crawlers or web spiders that visit a web page, parse its source code (in HTML) and gather the email addresses on it; then they identify the links embedded on the page and use these links to move to new pages so that they can repeat the same process again. During this process the email addresses found in these pages are gathered in a central repository. The main application uses them to start sending unsolicited emails to unsuspecting people.

You will design and implement a simple spambot application in pairs.

Feel free to design this application using your own style. You can use any class from the Java Core Library, and in particular the Java Collections Library and the classes and interfaces in the `java.net` package.

Note 1. You should be able to finish the exercise without using anything else beyond your current knowledge (labouriously acquired during the last ten weeks) and the JavaDoc. Although it takes five seconds to find implementations of spambots on the internet, you will learn hardly anything[1] from such endeavour. It is recommended that you do the exercise on your own (in pairs) as a learning experience. Remember that your will not be graded by the result of this exercise: your only benefit will be what you learn in the process about programming and especially about programming in a small team.

Note 2. The interfaces depicted in the following sections are only suggestions. Feel free to modify them or to create your interfaces from scratch according to your own design.

[1] Hardly anything useful for the exam of Programming in Java, I mean. You will probably learn useful things for other purposes.

2 WebPage

One of the first interfaces that you will need to implement is the interface of a web page. From the point of view of the spambot, a webpage is just a set of email addresses and a set of links to other web pages. A web page is identified by a URL (its address, e.g. *www.bbk.ac.uk*).

```
/**
 * A possible interface representing a web page.
 * This is only a suggestion.
 */
public interface WebPage {
    /**
     * Returns the URL that identifies this web page.
     * @return the URL that identifies this web page.
     */
    String getUrl();

    /**
     * Returns all the links on this webpage.
     *
     * Implementing classes should return a read-only view of this
     * set, using Collections.unmodifiableSet().
     *
     * @return all the links on this webpage.
     */
    Set<String> getLinks();

    /**
     * Returns all the emails on this webpage.
     *
     * Implementing classes should return a read-only view of this
     * set, using Collections.unmodifiableSet().
     *
     * @return all the emails on this webpage.
     */
    Set<String> getEmails();

    // Also, implementing classes should override equals() to
    // ensure that p1.equals(p2) returns true if and only if
    // p1.getUrl().equals(p2.getUrl()) returns true
}
```

A class implementing this interface may receive the URL at construction time, communicate with the remote server, get the source code of the page, read and parse it, and store all the links and emails in two sets for future reference.

This process may fail if there are problems to get the information needed. Possible causes include (but are not limited to) invalid url formats, non-existing pages, and network difficulties.

3 Crawlers (spiders) everywhere

A crawler is typically an autonomous agent that runs in a separate thread. In terms of behaviour there are no specific requirements, hence the absence of any suggestion of an interface.

Crawlers have to interact with three types of data. First, they need to know / remember all the links to be visited; they also need to know / remember those links that have already been visited, so as to not visit and parse the same page twice. The third type of data they need to store are the emails they get from webpages.

Crawlers need a way to terminate their execution. It is up to you to decide when crawlers will end. Some possibilities for your considerations are time limits, number of email addresses found, number of unsuccessful attempts to read a link, or to find new links.

In a first stage of design, it may help to have only one active crawler (i.e. one thread). Once the functionality is ready, you should be able to span several crawlers in parallel, all of them sharing the information described above. You should at least launch a crawler per processor in your machine[2], and possibly more, because I/O waits over the network connection will make your crawlers waste a lot of time.

4 Spam, spam, spam, spambot

The main class of the application could implement an interface like the following:

```
/**
 * A spambot
 */
public interface SpamBot {
    /**
     * Sets the seed.
     *
     * The seed is the very first URL that has to be given to the
     * system. The associated web page is the starting point for
     * the whole process of fetching web pages, extracting their
     * links and email addresses, and fetching more web pages.
     *
     * @param seedUrl the first URL to fetch and analyse
     */
    void setSeed(String seedUrl) throws MalformedURLException;

    /**
     * Returns the seed URL.
     * @return the seed URL.
     */
    String getSeed();

    /**
```

[2] If you do not remember how many CPUs you have on your machine, go over the notes and exercises on concurrency again.

```
         * Sets the number of threads.
         *
         * The user should be able to set the number of threads to be
         * used for running the crawlers.
         *
         * @param count the number of threads (i.e. crawlers) to start in parallel
        void setThreads(int count);

        /**
         * Initiates the scanning process.
         */
        void scanSite();

        /**
         * Returns all the emails gathered.
         *
         * This method should be executed only after the last crawlers
         * have stopped. If it is called before that point, its
         * behaviour is not defined.
        Set<String> getEMails();
}
```

The main class and the crawlers need access to the same data (links analyised, pending links, and email addresses). It is up to you to decide how these data structures will be shared, and how the access to them will be synchronised.

A Links and email addresses in web pages

Web pages are written in HTML, a formatting language that some of you will learn more about next term. For the scope of this exercise, it should be enough to know the following facts:

- Web pages are text documents.
- To retrieve the contents of a web page, you can use method `URL.openStream()`. This will return an `InputStream`, which you can read with an `InputStreamReader`, from which you can read with a `BufferedReader`. You have used the latter to read from files.
- A web page is written in HTML, which means it is full of tags like `<a>`, `<p>`, and `<div>`, open and closed (the closed tags look like `</a>`, `</p>`, and `</div>`). These tags mark groups of characters with some semantics. For example, `<p>` marks the start of a paragraph and `</p>` marks the end.
- Web links can be found in anchors (`<a>`), in the href parameter. In other words, you must look for text similar to

  ```
  <a href="http://www.bbk.ac.uk">Go to BBK!</a>
  ```

from where to extract your links.

- Email addresses, as you well know, look like two words separated by an *at* sign (`@`).