Curtin University — Department of Computing

Software Engineering Concepts (COMP3003)

Semester 2, 2017

Assignment

Date: Thursday 26 October, 23:59 AWST / 21:29 IST

Weight: 25% of the unit mark.

Your task is to develop a multithreaded news feed application, with plugins and a GUI.

1 Functionality

Your application must do the following:

(a) Display the current date and time (for the local timezone), to the minute, and keep it updated.

(b) Load a list of plugins on startup, where the plugin names are specified on the command-line.

(Don't hard-code any plugin names in the core application! That would defeat the purpose.)

(c) Invoke each plugin periodically in order to download a subset of news headlines.

Each plugin will download a particular URL and parse the HTML to obtain a subset of headlines. At a minimum, you must have plugins for the following news sources: (i) news.bbc.co.uk, (ii) www.nytimes.com, and (iii) arstechnica.com. You only need to grab the headline text itself, such as "*Shareholders force Zuckerberg to give up plan for non-voting shares*" (not the link, article, etc.).

Each plugin must specify its own particular update frequency (e.g., 30 minutes). The core application must invoke the plugin exactly this often to obtain an updated subset of headlines. Any positive integer number of minutes must be supported (but you may support more fine-grained values too if you like). In your own plugins you may specify any frequency.

(See advice below on downloading and parsing web pages.)

(d) Display all news headlines together in one overall on-screen list, sorted in reverse chronological order; i.e., with the most recent headlines first. Each headline should have its origin and download time attached; e.g., "*arstechnica.com: Shareholders force Zuckerberg to give up plan for non-voting shares (2017-09-30 12:45pm)*".

To do this, you must keep track of the time that each headline was downloaded. Initially, all headlines will be downloaded at the same time (more or less), when the application loads. However, over time, additional headlines will appear that weren't in the first set to be downloaded, and others will disappear.

Once a headline is no longer shown by the relevant news source, it must also disappear from your application.

(e) Provide an "Update" button that will immediately trigger a download of all news sources (the effect being the same as if they were all scheduled to be updated right now). You must prevent simultaneous downloads of the same source. Apart from that, a manual update should not affect the timing of the periodic updates.

(f) Display all current downloads in a list; i.e., which news sources the application is currently obtaining updates from. This list should itself automatically update.

(g) Provide a "Cancel" button that, when pressed, stops all active downloads. This should not hinder any future updates, manual or periodic. (See advice below.)

(h) Advise the user of any errors without aborting the application (unless it's actually an unrecoverable error).

2 Constraints and Non-Functional Requirements

Your application must conform to several other requirements:

(a) It must use Java's Swing GUI. All input and output must occur via the GUI (other than the names of the plugins, and log messages if implemented – see below).

(b) It must use multithreading in a manner that maximises performance and GUI responsiveness, and eliminates any potential for deadlocks and race conditions. The exact design is up to you.

(c) It must be scalable. That is, while you yourself only need implement three plugins, your core application must be prepared to handle *hundreds* simultaneously. Design your threading arrangements accordingly.

(d) It must use Gradle as the build tool, and each plugin must be a separate subproject.

(e) It *may* (but does not necessarily need to) use any of the Apache Commons and/or Google Guava libraries. This will of course have implications for your build setup. You may not use any other third-party libraries.

(Note: proper NFRs would have exact testing criteria, but I wanted to allow some open-endedness in how you approach this problem.)

3 Code and Design Quality

You must adhere to good SE practices, including (but not necessarily limited to):

(a) High cohesion and low coupling (separation of concerns). Don't bundle together diverse responsibilities in a single file, or share individual responsibilities across multiple files. (You should treat any anonymous/local/nested classes as *part* of their respective containing classes for this purpose.)

(b) Thorough commenting. Every top-level class and top-level method should have an attached comment. (You do not have to explicitly comment anonymous classes, but any relevant remarks should appear in the enclosing method's comment.)

(c) Proper error handling. Handle errors in the right place (which is not necessarily where the program first discovers them), and handle them sensibly.

(d) In-code referencing, if relevant. If you use external sources in preparing your code, you must cite them *in the code* at the relevant point(s). However, be aware that if you use external sources to short-circuit the design work involved in this assignment, *you may compromise your mark*.

Don't worry about citing material from the SEC lectures or worksheets, or from the standard Java API.

4 Report

Prepare a report (about 2 pages, more if you feel it is necessary) that explains your (a) class structure, (b) threading arrangements, and (c) build setup, and relates these back to the requirements given in sections 1–3.

5 Implementation Advice

The advice below does not add any extra requirements, but it will save you time!

5.1 Exceptions, Logging, and Profiling

Multithreaded programs can be very difficult to debug, but you can make it easier:

- Use visualvm/jvisualvm, or any other reasonable profiler tool, to peek inside your application at runtime. You will be able to see what each of your threads is doing.
- Whenever your application performs some significant task, log it. You're almost certainly familiar with the practice of putting `println`s throughout your code to track down bugs. Logging is like this, but more systematic and pre-emptive.

  You can simply use `println`, but you may like to try out Java's real logging API. Remember to make your messages meaningful, so you can understand what's happening when you see them later on.

  Log messages are not part of the UI, so you can print them out to the console (and/or to a file).
- Whenever you catch an exception, whatever else you do, log it. However, be mindful that logging is *not* the same as telling the user that something went wrong (because it is not part of the UI).

5.2 User Interface

You can reuse (or refer to) the sample code from worksheet 3 if you like. You could use a `JTextArea` or `JList` widget to display the on-screen lists.

If you need to display a message to the user, consider one of the static methods in `JOptionPane` (e.g., `JOptionPane.showMessageDialog(...)`).

Note: you do not need to make the GUI look "pretty". If you would like to nonetheless, do this after you implement the requirements.

5.3 Downloading Web Pages

There are many ways to programmatically download something, but we need a download to be *cancellable*. Java's ordinary IO classes (`InputStream`, `OutputStream`, etc.) *cannot* be interrupted, even if the read/write becomes stalled.

Instead, we need to use Java's "NIO" (new IO) classes, as follows:

```
import java.nio.ByteBuffer;
import java.nio.channels.*;
import java.io.IOException;
import java.net.URL;
...
URL url = new URL(...); // Download source

try(ReadableByteChannel chan = Channels.newChannel(url.openStream()))
{   // "try-with-resources" statement; will automatically call
    // chan.close() when finished.

    ByteBuffer buf = ByteBuffer.allocate(65536);
    byte[] array = buf.array();

    int bytesRead = chan.read(buf); // Read a chunk of data.
    while(bytesRead != -1)
    {
        ... // Get data from array[0 .. bytesRead - 1]
        buf.clear();
        bytesRead = chan.read(buf);
    }
}
catch(ClosedByInterruptException e) { ... } // Thread.interrupt().
catch(IOException e) { ... }                 // An error
```

When you call `chan.read(buf)`, the channel reads a chunk of data into the buffer object, which uses an array to actually store it. Importantly, it does this *interruptibly*. However, rather than throwing `InterruptedException`, it throws `ClosedByInterruptException` (which is actually a subclass of `IOException`, so you *must* catch it *before* `IOException`).

5.4 Parsing Web Pages

This could be a complex topic, but we'll simplify it a lot here. First, you will need to convert the downloaded bytes into a string, which will contain the HTML document representing the front page of the news site.

Next, you need to locate the various heading tags. HTML formally allows for six layers of headings, with the tags `<h1>...</h1>` for the top-level heading down to `<h6>...</h6>` for the lowest-level heading (though in practice six is a bit much, and it doesn't ususually go past about three or four). Not every *heading* in the HTML will be a *headline*, but at least for the news sites mentioned, each headline should be represented as a heading.

You'll need to see for yourself the structure of one of the news sites' web pages. Use your web browser to view the HTML source, and try to identify the headings within.

It's up to you to devise a way to programmatically extract the headings from the HTML source. You could use regular expressions, or even more basic string matching, or (if you really want to) the Document Object Model (DOM). However, your approach does not need to be sophisticated, as long as you get the headlines, and filter out anything that isn't a headline.

6 Scope Limitations

To minimise doubts, here is a list of various things you *do not* need to do, unless you particularly want to:

- Implement any concepts from the unit notes related to Android development, language integration, or domain-specific languages.
- Develop a single fool-proof approach to reading headlines from *any* website, or from RSS feeds. That's not what we're doing here.
- Provide test code, or testing-related information in your build setup.
- Provide a nice looking GUI (though you do need to provide *a* GUI).

7 Submission

Submit the following electronically to the Assignment 2 area on Blackboard:

- A completed Declaration of Originality;
- The project directory, including all build-related files and source code files in their appropriate subdirectories (but omitting the `build/` subdirectory);
- Your report (in .pdf format).

You are responsible for ensuring that your submission is correct and not corrupted. If you make multiple submissions, only your newest submission will be marked. The late submission policy (see the Unit Outline) will be strictly enforced.

Please DO NOT use WinRAR!

8 Marking Criteria

Here is the mark breakdown for this assignment:

1. [4 marks] – Workable GUI.
2. [3 marks] – Clock display.
3. [8 marks] – Plugin mechanism.
4. [3 marks] – Initiating updates: periodic and manual.
5. [4 marks] – Displaying and cancelling downloads.
6. [5 marks] – Headline extraction, sorting and display.
7. [8 marks] – Threading arrangements not otherwise covered.
8. [5 marks] – General design, coding and error handling practices.
9. [5 marks] – Build setup.
10. [5 marks] – Understanding demonstrated in report.

9 Academic Misconduct – Plagiarism and Collusion

This is an assessable task, and as such there are strict rules. You must not ask for or accept help from *anyone* else on completing the tasks. You must not show your work to another student enrolled in this unit who might gain unfair advantage from it.

These things are considered **plagiarism** or **collusion**.

Staff can provide assistance in helping you understand the unit material in general, but nobody is allowed to help you solve the specific problems posed in this document. The purpose of the assignment is for *you* to solve them *on your own*.

Please see Curtin's Academic Integrity website for information on academic misconduct (which includes plagiarism and collusion).

The unit coordinator may require you to provide an oral justification of, or to answer questions about, any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an Academic Misconduct inquiry.

End of Assignment