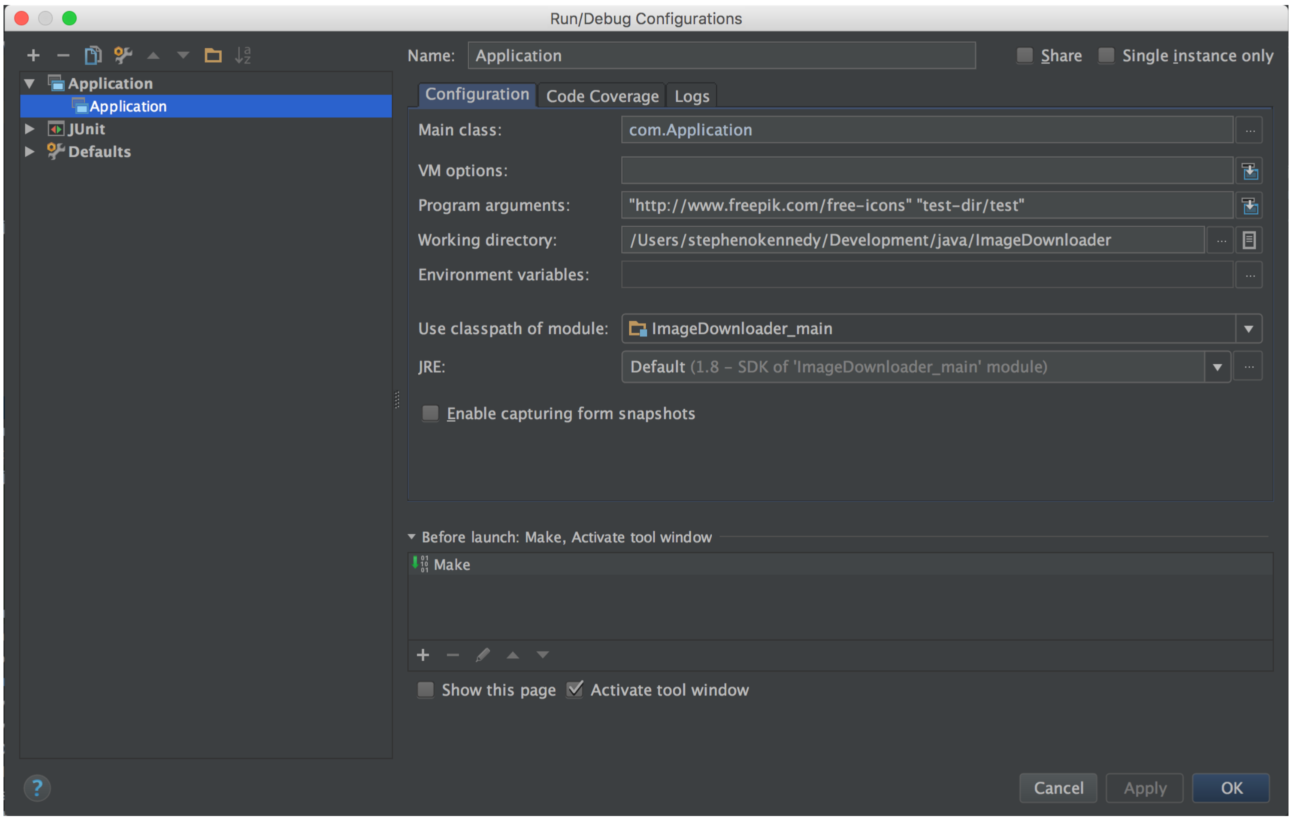
Image Downloader

Stephen O'Kennedy

The program is a command line tool that accepts two arguments. The first, is the URL of the site you want to the download images from and the second is the directory you want to save the images to. Once the images are downloaded they are then transcoded into three different sizes (100px, 220px, and 320px) and are in a jpg, png and gif format. Depending on the size of the image they will be saved in a small, medium or large folder. The tool will not download images that have a width or height less than 10px.

Instructions to run

1. Import project into Intelij and use Gradle to build the project
2. Edit the configurations of the Application.java file by opening the file (OSX "⌘ ⌥ R", Windows "Ctrl ⌥ R" select "Edit configurations…")
3. Enter the URL and directory arguments in the Program arguments textfield.



Name: Application

Main class: com.Application

VM options:

Program arguments: "http://www.freepik.com/free-icons" "test-dir/test"

Working directory: /Users/stephenokennedy/Development/java/ImageDownloader

Environment variables:

Use classpath of module: ImageDownloader_main

JRE: Default (1.8 – SDK of 'ImageDownloader_main' module)

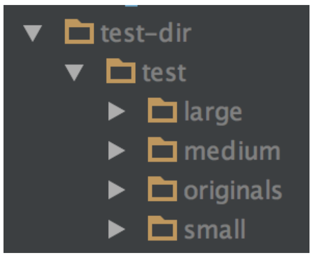
Enable capturing form snapshots

Before launch: Make, Activate tool window

Make

Show this page ✓ Activate tool window

Running the arguments in the above image will create the "test-dir" directory in the root of the project. The program will output the following files and directories each containing images:



```
test-dir
  test
    large
    medium
    originals
    small
```

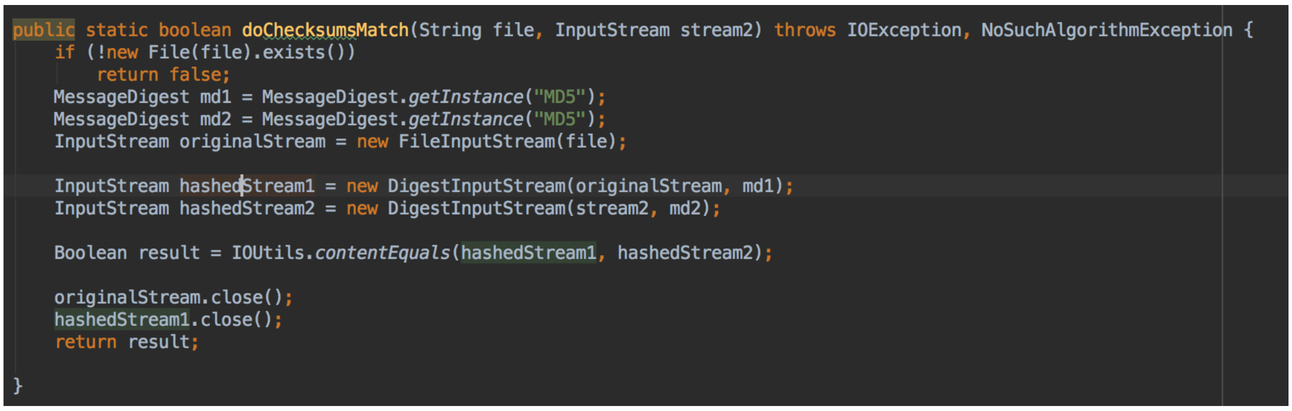
Design Choices

I designed the program taking strong inspiration from the functional programming paradigm. I tried to keep the as many variables and methods as static and pure as possible. As result a lot of the methods in both the ImageDownloader and ImageProcessor classes are static. Therefore many of variables in the program are immutable. The main reason why I decided to do this was to take state management out of the equation and by doing so the program is essentially thread safe.

The use of threads can be found in the ImageDownloader.downloadImages() and ImageProcessor.generateScaledImages() methods. In ImageDownloader.downloadImages() I used Executors.newCachedThreadPool() to enable as many thread as needed to download images from a site concurrently. If the program was to sequentially download the images it would suffer from the same head-of-line blocking issues as in HTTP 1.1. The threads in ImageProcessor.generateScaledImages() were used to generate the new scaled images from the images retrieved from the site.

In order to make the program easier to modify by a future developer I used two enums, one for the image size and the other for image file extensions. If the program is required to produce images that are, for example, 101px wide editing the ImageSize enum is a lot easier than combing through the program and updating the values.

As part of the requirements for the program it should be optimized to download only new or modified images. The simplest way I could see doing was to first of all check if there was an image with the same name in the originals directory. If there was a match I initially thought of doing a comparison of the bytes from the two images. However, while this approach may have been one of the most accurate ways to determine if an image has been modified, it’s also one of the least performant ways. I decided instead to use a checksum to check if a image has been modified, which should prove to be more performant. The MD5 encryption algorithm was used to create a digest for each image and then compare the two digests. While in theory the birthday paradox would determine that it is possible for two different images to produce the same digest the probability of that occurring is low enough that we don’t need to be concerned of clashes. Below is the method that compares the two images.



```java
public static boolean doChecksumsMatch(String file, InputStream stream2) throws IOException, NoSuchAlgorithmException {
    if (!new File(file).exists())
        return false;
    MessageDigest md1 = MessageDigest.getInstance("MD5");
    MessageDigest md2 = MessageDigest.getInstance("MD5");
    InputStream originalStream = new FileInputStream(file);

    InputStream hashedStream1 = new DigestInputStream(originalStream, md1);
    InputStream hashedStream2 = new DigestInputStream(stream2, md2);

    Boolean result = IOUtils.contentEquals(hashedStream1, hashedStream2);

    originalStream.close();
    hashedStream1.close();
    return result;

}
```

Design Pattern

From learning how functional languages like Haskell and to a lesser degree Scala work. Performing IO tasks such as those performed by this program, I feel it would make a lot of sense to use the pipeline pattern (http://parlab.eecs.berkeley.edu/wiki/_media/patterns/pipeline-v1.pdf) to introduce as much concurrency as possible.

Future Work and Reflections

As mentioned earlier in this write up I developed the program as functionally as possible and introduced concurrency to improve performance . I currently used a stopwatch object form Google's Guava util library to gather basic metrics in order to gauge performance. I also used TDD to develop the program, there is a suite of tests that check as many of edge cases as I could think of.

It should also be noted that the image downloading and processing can parallelised and can be mapped to the MapReduce architecture. In a distributed environment this would be a very scalable architecture. Since this exercise didn't specify the context in which the program would be used I had assumed that it would run on a single machine. Therefore I didn't look to implement this architecture, however MapReduce appears to be a good fit for this problem.

If I were to move forward with this project I continue to work on streamlining and fine-tuning the performance of the program. I would also looked at more edge cases and to uncover any hidden bugs in the program.

On reflection I would feel a language like Scala would be a better language to handle this problem. Given that you wouldn't need to worry about state at all.