Laboratory Sheet 5

This Lab Sheet contains material primarily based on Lectures 9 – 10.

The deadline for Moodle submission is 17:00 on Thursday 27 October.

Aims and objectives

- Further practice with object-oriented modelling in Java (especially inheritance and constructors)
- Practice with arrays and ArrayLists
- Processing command-line arguments in Java
- Practice catching and dealing with exceptions

Set Up

1. Download Laboratory5.zip from Moodle and launch Eclipse.
2. In Eclipse, select File → Import ... to launch the import wizard – this will be used to import the starter projects into your workspace. Then select General → Existing Projects into Workspace and click Next.
3. Choose Select archive file and then browse to the location where you downloaded Laboratory4.zip. Select Laboratory5.zip and click Open.
4. You should now see a new project in the Projects window: Submission5_1. Ensure that the checkbox beside this project is selected, and then press Finish.
5. In the *Package Explorer view* (on the left of the Eclipse window), you should now see an icon representing the new project (in addition to any projects that were there before).

Please refer to the lab sheet from Laboratory 1 for additional details on how to open and run the code in the new project. Note that the starter code for this lab consists only of the test cases – you should create all of the remaining classes yourself. Refer to the lab sheet for Laboratory 4 for details on creating classes in Eclipse if necessary.

Submission material

Preparatory work for these programming exercises, prior to your scheduled lab session, is expected and essential to enable you to submit satisfactory solutions.

Unit Tests: I have supplied some simple test cases to check that your code is working properly, in the file TestWordFinder.java. **Note that the tests may not test all aspects of the program's behaviour – passing all of the tests does not mean that your code is perfect (although failing a test will definitely indicate a problem to be investigated).** You should also read through the specification and make sure that you have implemented everything correctly before submitting.

Submission

Your task this week is to design a set of classes for **reading text files into memory** (from a file on the local file system or from a URL on the internet) and then **searching through the file contents** to count the **number of occurrences of a given word**.

You should construct four classes in total: a base WordFinder class; two subclasses, UrlWordFinder and FileWordFinder; and a final class, WordFinderMain, which contains a main method that exercises the three other classes.

WordFinder

The base WordFinder class should include two **`protected`** fields:

- `source`: a String indicating the source that the file contents was read from
- `contents`: an ArrayList<String> consisting of the lines of the text file read from source

It should also include the following instance methods:

- A protected constructor `WordFinder(String source)` that sets the `source` field
- An overridden `toString()` method that returns a string including the `source` value
- A method `public int count (String word)` that counts the number of occurrences of the given word in the file contents and returns it. More details of this method are given on the following page.

Finally, WordFinder should include a **factory** method as follows:

```
public static WordFinder createWordFinder(String source)
    throws java.io.IOException
```

This method should check the source string, and then construct an appropriate WordFinder subclass as follows:

- if `source` starts with "http", the method should construct and return a new UrlWordFinder
- otherwise, the method should construct and return a new FileWordFinder

You can use the `String.startsWith()` method to carry out the required check. The exception is included in the method signature because both of the subclass constructors may potentially throw an exception if there is a failure reading the file contents – see below for details.

FileWordFinder and UrlWordFinder

Both of these subclasses should extend the base WordFinder class, and should consist only of a constructor taking a single String argument and declaring a thrown IOException. That is:

- `public FileWordFinder (String fileName) throws java.io.IOException`
- `public UrlWordFinder (String url) throws java.io.IOException`

Both constructors should do essentially the same two things:

- Call the super-class constructor to set the `source` field, and then
- Open the given source and read all of the lines into the `contents` ArrayList

However, the process for reading from a file and reading from a URL differ.
To download from a URL, use a `java.util.Scanner` like this:

```
java.util.Scanner s = new java.util.Scanner
            (new java.net.URL(url).openStream());
```

To read the content of the URL, check the documentation for `Scanner.hasNextLine()` and `Scanner.nextLine()` to see how to read lines of input as Strings. You should store each line in the contents ArrayList as they are read. Don't forget to close the Scanner at the end.

To read from a file, you could use a similar Scanner process as above, but initializing the Scanner as follows on the file name:

```
java.util.Scanner s = new java.util.Scanner
            (new java.io.File(fileName));
```

You could also look into the documentation of `Files.readAllLines()` for a more efficient option.

Note that reading from a URL or a file may throw an IOException if there is a problem (e.g., network down, file does not exist, etc). As indicated above, you should not attempt to handle these exceptions in the constructors; rather, you should include the "throws" clause in the constructor signature and let the calling class deal with any I/O problems.

Counting words

After a WordFinder has been created – either from a URL or from a file – the `contents` field will contain a list of strings, one corresponding to each line read from a file. The `count()` method should iterate through the list of lines, and then iterate through each word on each line, and count up every time that a given word appears in the file.

Iterating through the list of lines should be straightforward. However, there are a few details to be considered when iterating through the words on a line. The easiest way to go through a line is to use the `String.split()` method, which splits a string around a given pattern and returns the result in a String array. In other words, if `s == "boo:and:foo"`, `s.split(":")` will return the array `{ "boo", "and", "foo" }`.[1]

For our purposes, we want to split each line on both whitespace and punctuation, so that we obtain only the words in the file. The correct expression to use in `split` is as follows (if you are curious, this is a **regular expression** that represents all whitespace and all punctuation characters; the backslashes are doubled to ensure that it is processed correctly by the Java parser):

```
"[\\s\\p{Punct}]+"
```

One final consideration: the `count()` method should count all occurrences of the given word regardless of case – that is, the results for `count("the")`, `count("THE")`, `count("The")`, and `count("tHe")` should all be the same. You could address this, for example, by appropriate use of either the `String.toLowerCase()` method or the `String.equalsIgnoreCase()` method – see https://docs.oracle.com/javase/8/docs/api/java/lang/String.html for documentation of both.

[1] Documentation of this method is available at https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#split-java.lang.String-

WordFinderMain

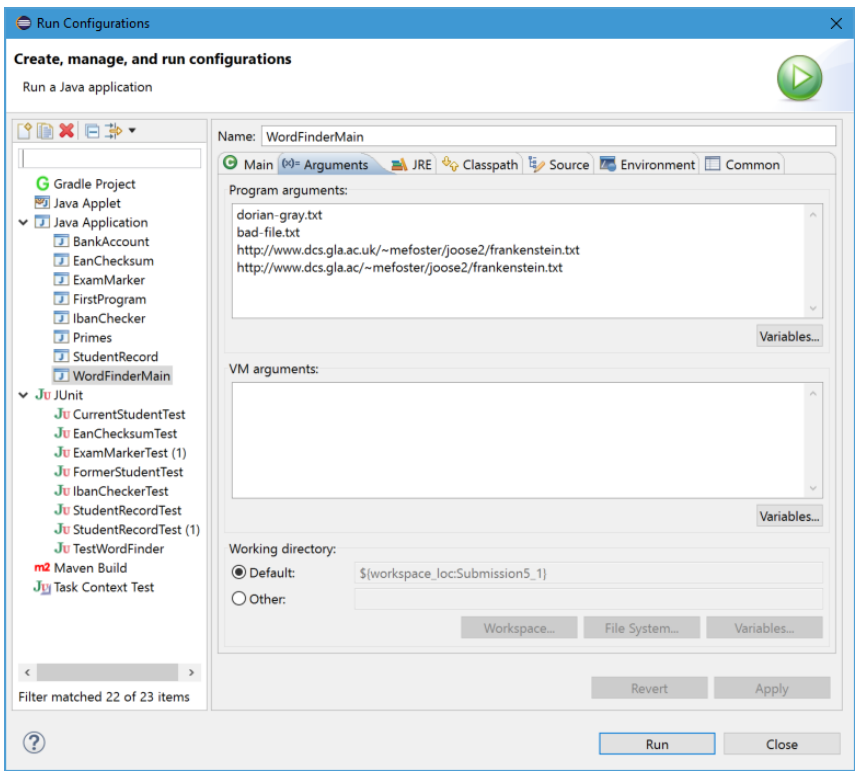
The final class that you should create is WordFinderMain, which should have only a main method that functions as follows:

1. First, create an empty ArrayList of WordFinder
2. For each string in the command-line arguments (i.e., the String[] args passed to the main method), attempt to create a WordFinder from that argument using the createWordFinder method
    a. If an exception is thrown, print an appropriate error message and continue
    b. Otherwise, add the created WordFinder to the list
3. Prompt the user for a word to search for (use a Scanner as in previous labs)
4. For each WordFinder in the list, print out the details of the word finder followed by the count of the given word

Here is how to specify command-line arguments in Eclipse.

1. Choose "Run – Run configurations …"
2. Look for WordFinderMain in the list on the left under Java Application. If it is already there, skip to step 4.
3. If WordFinderMain is not in the list, click on Java Application and then press the "New" button (the white rectangle with the plus sign on the left). If you have created WordFinderMain with an appropriate main method, Eclipse should automatically create a launch configuration with WordFinderMain as the main class.
4. With the WordFinderMain configuration selected, click the "Arguments" tab. You can then specify the command-line arguments to your program in the top "Program arguments" box. Each string entered here will be passed to your main method through the args array.

Here is an example of the run configurations window with some arguments specified:



For possible command-line arguments, I have provided a text file dorian-gray.txt in the Eclipse project, and you should also be able to use the URL

http://www.dcs.gla.ac.uk/~mefoster/joose2/frankenstein.txt as a valid URL for your program.[2] You should also test your program with a mixture of invalid and valid sources (as in the screenshot above).

Here is some possible output for your program using the above command-line arguments. Your output might appear different depending on how you choose to implement things, but this is the basic expectation.

```
Warning: couldn't create WordFinder for bad-file.txt:
java.io.FileNotFoundException: bad-file.txt (The system cannot find
the file specified)

Warning: couldn't create WordFinder for
http://www.dcs.gla.ac/~mefoster/joose2/frankenstein.txt:
java.net.UnknownHostException: www.dcs.gla.ac

What word to search for?

monster

Testing WordFinder(File = dorian-gray.txt)

- Occurrences of 'monster': 3

Testing WordFinder(URL =
http://www.dcs.gla.ac.uk/~mefoster/joose2/frankenstein.txt)

- Occurrences of 'monster': 31
```

[2] Both of these text files were downloaded from Project Gutenberg, which provides free ebooks, mainly of books with expired copyright. See https://www.gutenberg.org/ for more details.

How to submit

You should submit your work before the deadline no matter whether the program is fully working or not.

When you are ready to submit, go to the JOOSE2 moodle site. Click on Laboratory 5 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code ...\Submission5_1\ and drag *only* the four Java files WordFinder.java, UrlWordFinder.java, FileWordFinder.java, WordFinderMain.java into the drag-n-drop area on the moodle submission page. **Your markers only want to read your java files, not your class file.** Then click the blue save changes button. Check the three .java file is uploaded to the system. Then click submit assignment and fill in the non-plagiarism declaration. Your tutor will inspect your file and return feedback to you via moodle.

Guidelines for Mark Scheme

A1: All methods completed, with correct types and behaviour. All JUnit tests pass.

B1: Majority of methods completed, with correct types and behaviour. Some JUnit tests pass.

C1: Some methods attempted, possible problems with types and behaviour. Code may not compile.

D1: Minimal effort.

For this assignment (and all future ones), we will also be considering code style – comments, formatting, variable names, etc – in addition to correctness. You will be deducted some marks (e.g., from an A1 to and A3) if your code style is not appropriate. Please check with the tutors and demonstrators during your lab session if you are uncertain about style, or look at the BankAccount class from Laboratory 3 for an example.