Week 1 Exercise (0%): Java Refresher and Automated Checking

CS2001 - Jon Lewis <jon.lewis@st-andrews.ac.uk>

Due date: Friday 22nd September, 21:00

This exercise does not count towards your final grade for the module. However, it is highly recommended that you complete the exercise as it gives you the opportunity to refresh your Java skills and crucially, use and familiarise yourself with our new automated checking tools (which will be used for many assignments in future).

Objective

- to refresh your Java skills from last year
- to use and familiarise yourself with our new automated checking tools

Learning Outcomes

By the end of this practical you should understand:

- Java arrays
- programming to an interface
- the typical structure of second level practicals
- how to run the automated checker on your solution prior to submission
- the format of automatically generated feedback

Getting started

To start with, you should create a suitable assignment directory such as `CS2001/W01-Exercise` on the Linux lab clients. You should decompress the `zip` file at

`https://studres.cs.st-andrews.ac.uk/CS2001/Practicals/W01-Exercise/code.zip`

to your assignment directory. Please note that the `zip` file contains a number of files in the `src` directory. **Once you have extracted the `zip` file, you should probably delete it to avoid accidentally overwriting your `src` directory with files contained in the `zip`.**

Requirements

You are to write part of a very simplistic spelling checker. The overall operation of the spelling checker is as follows:

1. a list of words is read in from a file `words.txt` (provided) and is treated as a dictionary
2. your program should take a command-line argument (via the `main` method in your `SpellChecker` class) representing a word that is to be checked

3. your `SpellChecker` class should check whether the given word is in the dictionary or not
4. if the given word is found, your program should print out a line containing the word and "correct"
5. if the word is not found, your program should print out a line containing the word and "not found", and the words which would have come before and after it in alphabetical order in the exact format shown below
6. if no command-line arguments are supplied, then your program should print the line containing usage information as shown below
7. your program should treat all words as lower-case words

Below are some examples of the expected output for some different command-line arguments indicating the functionality that you should provide:

```
java SpellChecker computer
computer correct

java SpellChecker Computer
computer correct

java SpellChecker computronic
computronic not found - nearest neighbour(s) computist and computus

java SpellChecker
Usage: java SpellChecker <word_to_check>
```

The `zip` file you can download and extract as indicated above contains source code for `DictionaryLoader` and `SpellCheckResult` classes and the `ISpellChecker` interface. Your job is to implement a `SpellChecker` class which implements the `ISpellChecker` interface (you must not change the interface at all). Further details are provided below.

- `DictionaryLoader` provides a method `loadDictionary` which reads the words from a file `words.txt` (also on Student resources) and returns the words as an array of lower-case strings in alphabetical order.
- `SpellChecker` should implement the `ISpellChecker` interface and should provide a zero-argument constructor, which calls `DictionaryLoader.getInstance().loadDictionary` and stores the result in a suitable attribute. As mentioned above, the `main` method in your `SpellChecker` class should take a command-line argument representing a word that is to be checked.

  The `SpellChecker.runChecker` method should run the checker for the given argument. The `SpellChecker.check` method should search the dictionary for the given word. You may find the `binarySearch` method in the `java.util.Arrays` class useful for this.

  Note: It is generally a good idea to use standard API methods and data structures, unless of course the objective of a practical is to implement these for yourself. In this case, we are not asking to implement your own data structure, or your own sort or search methods, so please feel free to use the standard API.
- `ISpellChecker` is an interface defining the `check` and `runChecker` methods of the `SpellChecker` class
- `SpellCheckResult` is a class used by `SpellChecker.check` to return results. A `SpellCheckResult` object has fields indicating whether the word was correctly spelt and, if not, what were the words immediately before and after it.

Extensions

Here are some enhancements you could try for this simple unassessed exercise. You could alter your program such that

- it accepts one or more words on the command-line and checks each word in turn
- it does not print out a word before or after a checked word if it was before the first dictionary entry or after the last one, in this case one of the fields in the SpellCheckResult might be null
- your `SpellChecker` removes any duplicates from the dictionary prior to checking

Compiling and Running

In order for your program to be compatible with the automated checker that we (and you) are going to use, please observe the following:

- All your java source files should be in a `src` directory in your assignment directory.
- You must include a `main` method in your `SpellChecker` class which takes a command-line argument representing a word that should be checked as outlined above.
- It must be possible to compile all your program `.java` source files to `.class` files using the command `javac SpellChecker.java` from within the `src` directory from a terminal window on the Linux lab clients/servers.
- It must be possible to run your program using e.g. the command `java SpellChecker computer` from within the `src` directory from a terminal window on the Linux lab clients/servers as indicated above.

Running the Automated Checker

One of the main objectives of this exercise is to expose you to the automated checking system that we are going to use for a number of assignments. As such, please have a go at this prior to submitting your attempt. It should help you see how well your program performs on the tests we have made public and will hopefully give you an insight into issues prior to submission. In order to run the automated checking system on your program, open a terminal window connected to the Linux lab clients/servers and execute the following commands:

```
cd ~/CS2001/W01-Exercise
stacscheck /cs/studres/CS2001/Practicals/W01-Exercise/Tests
```

assuming `W01-Exercise` is your assignment directory and is in a `CS2001` folder in your home directory on the Linux lab machines. If all goes well, then you should see output similar to the one below

```
Testing CS2001/CS2101 Week 1 Exercise
- Looking for directory 'src': found in current directory
* BUILD TEST - basic/build : pass
* COMPARISON TEST - basic/Test01_computer/progRun-expected.out : pass
* COMPARISON TEST - basic/Test02_computronic/progRun-expected.out : pass
* COMPARISON TEST - basic/Test03_Computer/progRun-expected.out : pass
* COMPARISON TEST - basic/Test04_COMpuTER/progRun-expected.out : pass
* COMPARISON TEST - basic/Test05_no_arguments/progRun-expected.out : pass
* INFO - basic/TestQ_CheckStyle/infoCheckStyle : pass
--- output ---
Starting audit...
Audit done.
```

If it is not going so well, here are some things to consider

- If the automated checker cannot be started, then you have most likely mis-typed the commands to invoke the checker shown above.
- If the automated checker runs but fails at the build stage, then no other tests can be conducted, so you will have to fix this issue first. Likely reasons for the build failure include:
  - executing the checker from some directory other than your `W01-Exercise` directory or specifying an incorrect path to the tests
  - your program cannot be compiled as required by the checker and as specified above from a `src` directory in your assignment directory. Try to modify your program, your directory naming, or directory structure such that your program can be compiled using the simple command `javac SpellChecker.java` from within the `src` directory.
- If the automated checker runs and the build succeeds, but all comparison tests fail, it could be that your program cannot be run using the simple command `java SpellChecker computer` from within the `src` directory in your submission. Please consider the following:
  - Ensure you have written a `main` method in your `SpellChecker` class.
  - Ensure your program uses the `args` command-line arguments that are passed to your `main` method.
  - Changing the package of your Java classes will cause problems for this assignment, make sure you do not have a `package` statement in your `.java` files.
- The names of the comparison tests 1 to 4 indicate the word that is being passed to your program and each test expects your program to produce a specific output as outlined above. Test 5 launches your spell checker without any command-line arguments and expects the usage message to be printed. If one or more of the tests fail, then it may be that your program has a bug or is simply not printing out exactly what is expected and shown in the sample execution runs above.
  - Maybe you have included some additional debug messages in your output or additional new lines, these will also cause tests to fail.
  - Try to ensure your output matches the one shown above exactly when executing your program from the command-line.
- The final test `TestQ CheckStyle` runs a program called Checkstyle over your source code and uses a style adapted from our St Andrews coding style (informally known as the *Kirby Style*). You may receive a lot of output from the style checker for your program. In order to address these, you can look at the published guide at

  `https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/programming-style.html`

  The style recommends the use of spaces as opposed to TABs for indentation. Don't worry too much about this.

If nothing is working and you don't know why, please don't suffer in silence, ask one of the demonstrators in the lab. This is the first time you will have tried to use the automated checking system, so there are bound to be some issues.

Testing

Running the automated checker will run some basic tests on your program. We would encourage you to perform more rigorous testing, either manually or by writing your own tests. If you adopt the latter approach, it is probably a good idea to start by looking at the tests we have made available to you at `/cs/studres/CS2001/Practicals/W01-Exercise/Tests`. You can create new tests in a local sub-directory in your assignment directory and pass the directory of your own tests to `stacscheck` when you run it from your `W01-Exercise` directory. Also, you should look at the documentation for the automated checker at

`https://studres.cs.st-andrews.ac.uk/Library/stacscheck/`

Deliverables

Hand in via MMS, by the deadline of 9pm on Friday of Week 1:

- A zip file containing your assignment directory in which there is a `src` directory containing all source code files.

Marking

The submission does not count towards your module grade and you will not receive a mark on MMS. However, we will upload the output from the automated checker for all our tests to MMS.

I would remind you to ensure you are following the relevant guidelines on good academic practice as outlined at

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`