COMP1551 Core Programming

Coursework 4

Introduction

Your task is to write a Java program that analyzes and visualizes earthquake data. You have three different options for your solution, offering progressively greater difficulty (and higher marks). The assignment is worth 10% of your overall module grade.

Preparation

1. Visit the 'Feeds & Notifications' page of the USGS Earthquake Hazards Program web site:

   ```
   http://earthquake.usgs.gov/earthquakes/feed/v1.0/
   ```

   Click on the 'Spreadsheet Format' link, which will take you a page detailing how the USGS makes its earthquake data available as CSV files. Study the information on this page. Click on one of the Data Feed links on the right of the page (e.g., the one for M4.5+ quakes ocurring in the last 7 days). Open the downloaded CSV file in a text editor and study it.

2. Download `cwk4files.zip` from the VLE and put it into a directory that you have created for this coursework.

3. The final preparatory step depends on whether you are using Eclipse or not.

   - **If not using Eclipse**, unpack the Zip archive, e.g., by using `unzip cwk4files.zip` on the command line. This will create several files and a `src` directory tree. Note that any Java source code files must go under here, in `src/comp1551/cwk4`.

     The provided file `build.xml` is an **Ant buildfile** that can be used to compile and run your code. For example, you can enter `ant compile` to compile your code. Enter `ant -p` to see a list of all the tasks that can be performed with it.

   - **If using Eclipse**, create a new Java project for the coursework in the usual way, then right-click on the project folder and choose *Import*. On the resulting dialog, open up the 'General' folder, select the 'Archive File' option and click *Next*. Use the *Browse* button to find and select `cwk4files.zip`, then click *Finish*.

     After the import has completed, right-click on the file `jmustache-1.10.jar` and choose *Build Path → Add to Build Path*.

     Eclipse will still show errors in the project. These relate to the provided file `QuakeList.java` and are expected. They will disappear once you have created a suitable `Quake` class as part of the basic solution.

Basic Solution

This is worth **20 marks**, out of a possible 40 marks.

There are three stages:

1. Create a new class called `Quake`, in a file called `Quake.java`. Your class should be put in the `comp1551.cwk4` package, which is achieved with a line like this at the top of the file:

   ```
   package comp1551.cwk4;
   ```

   (Eclipse can create this for you if you are using the *New Class* wizard.)

   Your `Quake` class should have fields to represent latitude, longitude, magnitude and depth for a single earthquake. It should have getter methods for each of these attributes. It should also have a `toString` method that returns a string giving magnitude, depth, latitude and longitude, in that order. The string's format should match this example:

```
M4.8, 28.3 km, (-0.5908,122.4168)
```

Finally, it should have a constructor that takes a string as its only parameter. This string is a single record from the CSV data feed produced by the USGS, in which the different values characterising an earthquake are separated by commas.

When implementing the constructor, please note that Java strings have a `split` method that works in much the same way as the corresponding method for Python strings—i.e., it breaks up one string into an array of strings, on a given delimiter (such as a comma!) Note also that conversion of strings into `double` values can be accomplished using static method `Double.parseDouble`, e.g.

```
double value = Double.parseDouble(text);
```

2. Next, edit the file `QuakeList.java` that we have provided. The `QuakeList` class in this file represents a list of earthquake data acquired from a USGS data feed. The data feed is specified by supplying two pieces of information as strings: the severity level (`"1.0"`, `"2.5"`, `"4.5"`, `"significant"`) and the time period (`"hour"`, `"day"`, `"week"`, `"month"`). Each earthquake is represented by a `Quake` object.

   You will need to finish the implementation of the `acquireData` method in `QuakeList`. You will also need to add two new methods called `size` and `print`; see the Basic Solution 'To Do'comments for further details.

   When creating the `Scanner` object in `acquireData`, note that an `InputStream` is required. A suitable stream can be obtained by calling the `getInputStream` method on the `connection` object. Once you've created the `Scanner`, you can use it to read lines of text in the normal way (as described in the workbook), only this time the data are being downloaded over the network instead of being read from disk or the keyboard.

3. Finally, add a `main` method to `QuakeList` containing a test program that creates a `QuakeList` object for earthquakes of magnitude 4.5 and above, occurring in the last 7 days. Your test program should also call the `print` method to print the details of all these earthquakes.

Make sure that you follow a sensible coding style and that you do so consistently. Please write Javadoc comments for `Quake` and for the methods you add to `QuakeList`. The latter already uses this style of commenting, which you can use to guide you; see also `http://bit.ly/jdochowto`.

Marks for this part will be awarded as follows:

Feature	**Marks**
`Quake` constructor	2
`Quake` fields and getters	4
`toString` method in `Quake`	2
Completion of `acquireData` in `QuakeList`	2
`size` method in `QuakeList`	1
`print` method in `QuakeList`	2
`main` method in `QuakeList`	2
Successful compilation & execution (no errors/warnings)	2
Coding style and use of Javadoc comments	3

Intermediate Solution

This builds on the basic solution and is worth a further **12 marks**.

1. Add the following features to the `QuakeList` class

   - A method `strongest` that finds and returns the `Quake` object having the largest magnitude
   - A method `shallowest` that finds and returns the `Quake` object having the smallest depth
   - A method `deepest` that finds and returns the `Quake` object having the largest depth

   Each of these methods is worth up to 2 marks.

2. In a new class called `QuakeInfo`, write a small program that uses the `QuakeList` class to display information about an earthquake dataset with a severity level and time period that are specified on the command line. Here is an example of the expected output:

```
79 quakes analyzed
Strongest: M6.4, at (-18.384,-69.146)
Shallowest: 8.4 km, at (29.219,142.015)
Deepest: 649.7 km, at (-26.306,178.760)
```

Your program should handle cases where two command line arguments have not been supplied by displaying an appropriate error message. It should also intercept any exceptions generated by `QuakeList` and print the exception object (so that the error message appears on screen rather than the full traceback).

4 marks will be awarded for program source code. A further 2 marks will be awarded for successful compilation and execution of the program without errors or warnings.

Advanced Solution

This builds on the the intermediate solution and is worth a further **8 marks**.

Note: you should not attempt this if you found the basic or intermediate stages difficult!

1. For 2 marks, add a method called `asGoogleMap` to `QuakeList`. This method should generate a Google map of earthquakes, returning the HTML for the map as a string. Figure 1 shows what this HTML will look like when displayed in a browser.

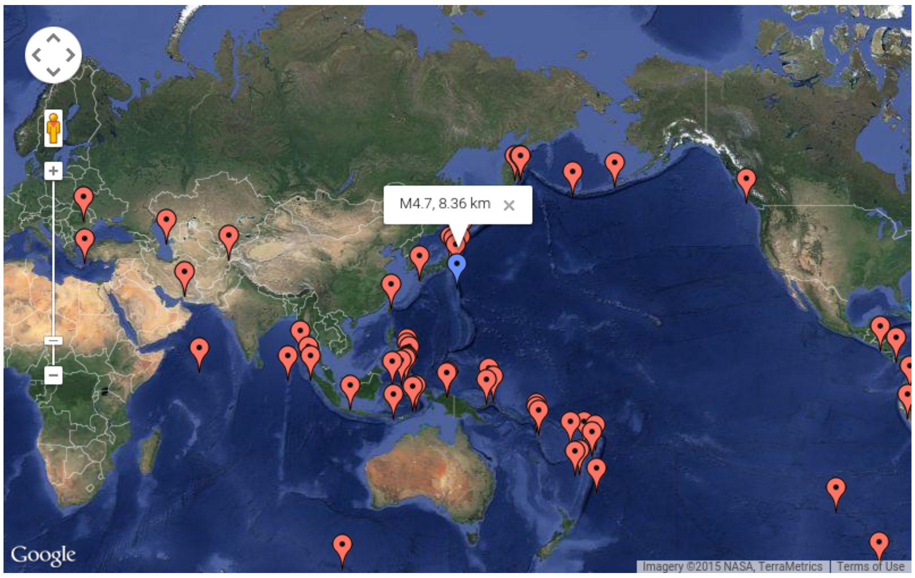

Figure 1: Earthquakes plotted as a Google Map

Note: you don't have to figure out how Google maps actually work, as the HTML & JavaScript you need has already been provided, in the template `map-template.html`. Existing code in `QuakeList` loads this into a JMustache `Template` object. You can merge your data with the template by calling its `execute` method, passing in the data as a `Map` collection of some kind. The severity level and period strings must be put into this map, using keys of `level` and `period`, respectively. The list of `Quake` objects should be put into the map using a key of `quakes`. The `execute` method returns the output you need, as a string.

For more information on JMustache, see its GitHub page:

`https://github.com/samskivert/jmustache`

2. For 3 marks, add a method called `asImageMap` to `QuakeList`. This method should read an image showing a map of the world, provided for you as the file `map-base.png`. It should do this using the `read` method `ImageIO` class, which returns a `BufferedImage` object.

Your method should draw a small coloured square into the image, at the location of each earthquake. Choose a colour that stands out from the map background here. You method should then return the now annotated image to the caller. Figure 2 shows what this image will look like when displayed.

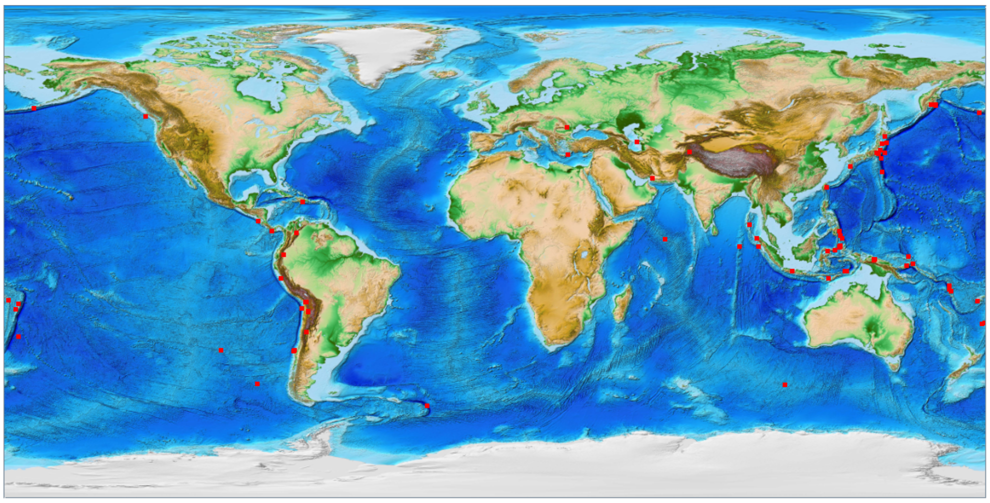

Figure 2: Earthquake map as an image

You may need to consult `http://bit.ly/workimages` for more on working with images. You will find that `http://bit.ly/javagraphics` is a useful source of information on how to draw things into an image. Drawing essentially involves retrieving a `Graphics2D` object from the image by calling its `createGraphics` method. You can then call the `fill` method on this `Graphics2D` object to draw filled shapes into it.

Note: the image dimensions are 1080×540, so there are 3 pixels per degree of longitude and 3 pixels per degree of latitude. You'll need to take this into account when converting longitude and latitude into pixel x and y coordinates. You'll also need to allow for the fact that longitude and latitude run from $-180°$ to $180°$ and from $-90°$ to $90°$, respectively, whereas pixel coordinates cannot be negative. Not only that, but y increases *down* the screen for images.

3. In a new class called `QuakeMap`, write a small program that uses `QuakeList` to generate a map of earthquakes. Use either `asImageMap` or `asGoogleMap` for this, as you wish. Your program should output the generated map to a suitable file[1]

   3 marks are available here for appropriate code that compiles and executes successfully, without errors or warnings.

Submission

Before you submit anything, check your coding style and use of comments. Make sure that you remove any remaining 'To Do' comments from the `QuakeList` class definition.

Your files must be submitted as a Zip archive. To generate this archive, go to the directory containing your project files in a terminal window. There should be a file called `build.xml` there. Enter the following command:

```
ant zip
```

Alternatively, from within Eclipse, double-click on `build.xml` to open it, then find the 'zip' item in the Outline view on the right of the workbench. Right-click on this item and choose *Run As → Ant Build*.

The deadline for submissions is **10 am on Thursday 7 May 2015**. The standard university penalty of 5% of available marks per day will apply to late work, unless an extension has been arranged due to genuine extenuating circumstances.

Note that all submitted code will be run through plagiarism detection software.

[1] If you want an added challenge, have your program load the Google map into a browser, or display the image in a simple GUI using Java's Swing toolkit. There are no extra marks for this, but we'd be interested in seeing how you approach it!