Java Web Application Report

Implemented features

The application I have developed allows the user to create basic text notes, with different categories to store their notes under. Each note has a label and can be viewed in the index, wherein clicking on an index entry displays the corresponding note. The notes are stored in .json files, and they can be deleted, edited and renamed as necessary. The categories are also stored in .json files, and a category can be deleted without the loss of notes stored in said category. Free-form text search is also supported, so that results are shown based on string matches with any text in a note.

Design and programming process

When designing my classes I first considered which ones would be required for the most basic features – hence I started with the Note.java class, the NoteIndex.java class (for displaying the notes) and the Category.java class. I made Note.java an abstract class since it would be used to define the common set of method signatures and variable definitions needed for its subclasses (TextNote, ImageNote, and URLNote, the latter two I did not end up implementing fully). There was some overlap between the classes for notes and categories, such as the getters and setters for IDs, names, and the times at which the note/category had been created and last updated. They also both had a 'containsText' method to return a Boolean value if a string that had been searched for was within the note or category. However, the implementation of this method in the Note.java class was abstract, left to defined in its subclasses, while in the Category.java class it was fully defined.

Following from this I added the FileStorage.java class and the NotesModel.java class to handle storing notes and categories in memory. The latter would have methods for updating, adding and deleting indices, notes and categories, add these methods would in turn call the required methods on an instance of the FileStorage.java class. New subdirectories would be created for notes, categories and indices respectively, and for each one of the three that was created, it would be written to a .json file in the corresponding directory.

In terms of overall object-oriented design practice, my project did contain some flaws. For example, I implemented classes for which the final functionality was never made, and some methods that were made public could perhaps have been made private to increase the robustness of my code. Despite these minor flaws, however, I believe my project was well structured overall. I made good use of abstraction as touched upon earlier, my classes were cohesive, and there was thorough error testing throughout, highlighted by the used of try/catch statement across the codebase. I also made use of synchronised methods to prevent race conditions between threads. All in all, this led to a coherent development cycle that produced a polished web application.