Data Structures and Algorithms 1

CA Exercise 2 – Drinks/Beverages Information System

The objective of this CA exercise is to create an application for storing and retrieving information on drinks/beverages (e.g. cocktails) and the ingredients/components (e.g. spirits, mixers, ice, sugar, etc.) that comprise them. The application should include both hashing and sorting functionality, and provide a JavaFX graphical user interface.

The application should allow/support the following:

- The addition of new drinks/beverages to the system. These might be cocktails, mocktails, hot beverages (e.g. cup of tea, hot chocolate, Irish coffee), cold non-alcoholic beverages (e.g. rock shandy), or even drinks that consist of just one ingredient (e.g. pint of stout or lager, glass of red or white wine, a glass of lemonade, etc.).
    - Drink name, place/country of origin, textual description, and an image/picture of the completed drink (as a URL) are among key data to store.
- The addition of new drink ingredients/components to the system. Examples are spirits, liqueurs, mixers, soft/fizzy drinks (e.g. cola, lemonade), wines, beers, and other components such as sugar, milk, water, coffee, lemon, etc. Anything that could be used to make up a drink could be added as an ingredient/component.
    - Name, textual description, and alcohol content (ABV) are among key data to store.
- Ability to create recipes for drinks that associate ingredients/components with the drinks they feature in, and the quantities (in millilitres) to use therein. Note that a given ingredient (e.g. vodka) could be used in various drinks (e.g. Bloody Mary, Sea Breeze, etc.) but in different amounts.
- Ability to edit/update/delete drinks and ingredients.
- Ability to search for drinks and ingredients using some parameters/options. The search might be: by name, by description keyword, by maximum alcohol content/ABV, etc. The key thing is that a search facility is provided with *some* (minimum of 2) search options.
- Listings of search results (for both drinks and ingredients) should be appropriately sorted depending on the chosen search parameters/options. The sorting could be (1) alphabetical by name, or (2) by total alcohol content/ABV (note that ABV would have to be calculated for a drink comprising various ingredients of varying quantities and ABVs). Other sorting options/parameters can also be provided, but alphabetical and ABV sorting are the key ones to implement.
- The system should support some level of interactive "drill down" for additional detail or navigation/browsing. For instance, searching for drinks containing vodka should provide a list of any vodka-containing drink; one drink could then be clicked on to see more information specifically on that drink; that detail could include a list of ingredients in the drink; clicking on any ingredient opens up details on the ingredient, including a list of all drinks containing that ingredient; and so on.
- The system should support some form of persistence whereby the internal data is saved to a file on exit, and reloaded for use on the next execution. You could use e.g. binary files, XML files, plain text files, or anything else that provides an image/snapshot of all internal data.
    - There is no need to look to databases or anything like that. A single snapshot/image file is fine for our purposes.

Notes

- This is a team CA exercise. Teams consist of 2 students only. Students should find/choose their own partners.
- You will have to demonstrate this CA exercise in the lab sessions and you will be interviewed individually on various aspects of it. You are expected to be able to answer all questions on all code individually (so make sure that you understand all code, including the parts that your teammate wrote).
- This CA exercise is worth approximately 35% of your overall module mark.
- As with CA Exercise 1, you cannot use any existing Java collections or data structures classes (e.g. ArrayList, LinkedList, or any other class that implements the Collection interface or any of its children – if in doubt, ask me!). You essentially have to implement the required data structures and algorithms from scratch and from first principles (in line with the module learning outcomes).
    - You are free to reuse any of your generic code from CA Exercise 1.
    - You can use a regular array for your hash table(s) only. You cannot use regular arrays beyond that.
- You also cannot use any sorting or searching methods provided by Java in e.g. the Arrays or Collections class (or any third party library equivalents). You should implement any searching and sorting routines from scratch. Again, this is a learning exercise, and in line with the module learning outcomes.
    - Note that you cannot use a bubble sort either (as this is the full example provided in the course notes, and I don't want you to just copy-paste that). Use any other sorting algorithm except a regular bubble sort for this reason.
- You have to use hashing as part of the project to avoid (excessive) linear/sequential searching. For instance, a custom hash function should be used for retrieving details on a specific drink or ingredient when searching. Again, you have to provide your own implementation of hashing.
- You have to provide a JavaFX based graphical user interface for the system. Exactly what your interface looks like, though, is up to you.
    - You can choose to implement a command line interface if you wish, but you will lose the marks allocated specifically for the JavaFX GUI.
- Remember that the key point of this CA exercise is to demonstrate knowledge and proficiency with hashing and sorting in particular. Keep this in mind!

Indicative Marking Scheme

- Appropriate custom ADTs for drinks, ingredients, etc. = 10%
- Create/add facilities (drinks and ingredients) = 10%
- Edit/update/delete facilities (drinks and ingredients) = 10%
- Search and listing facilities (multiple search options; drinks and ingredients) = 15%
- Sorting of search results/listings (alphabetical and ABV; drinks and ingredients) = 10%
- Hashing for individual search/lookup (drinks and ingredients) = 10%
- Persistence facility (saving and loading data) = 10%
- JavaFX graphical user interface = 10%
- JUnit testing (minimum of 5-6 useful unit tests) = 5%
- General (commenting, style, logical approach, completeness, etc.) = 10%