Data Structures and Algorithms 1 (2023)

CA Exercise 2 – Baking Information System

The objective of this CA exercise is to create an application for storing and retrieving information on baked goods (e.g. cakes, breads, biscuits, etc.) and the ingredients/components (e.g. flour, eggs, sugar, etc.) that comprise them. The application should include both hashing and sorting functionality, and provide a graphical user interface.

The application should allow/support the following:

- The addition of new baked goods to the system. These might be cakes, breads, biscuits, tarts, pies, etc.
    - Baked good name, place/country of origin, textual description, and an image/picture of the completed baked good (as a URL) are among key data to store.
- The addition of new baking ingredients/components to the system. Examples are flour, eggs, milk, chocolate, sugar, alcoholic spirits, fruits, nuts, etc. Anything that could be used to make a baked good could be added as an ingredient/component.
    - Name, textual description, and calories (as kcal per 100g/ml) are among key data to store.
- Ability to create recipes for baked goods that associate ingredients/components with the baked goods they feature in, and the quantities (in grams or millilitres as appropriate) to use therein. Note that a given ingredient could be used in various baked goods but in different amounts/quantities.
- Ability to edit/update/delete baked goods, recipes, and ingredients.
- Ability to search for baked goods (including recipes) and ingredients using some parameters/options. The search might be: by name, by description keyword, etc. The key thing is that a search facility is provided with *some* (minimum of 2) search options.
- Listings of search results (for both baked goods and ingredients) should be appropriately sorted depending on the chosen search parameters/options. The sorting could be (1) alphabetical by name or (2) by calories.
    - Note that calories would have to be calculated for a baked good comprising various ingredients of varying quantities and calories e.g. using 50g of dark chocolate when dark chocolate has 500kcal per 100g => 250kcal is added to the total calories for the baked good (and so on for all other ingredients in the baked good to arrive at the final calories count).
    - Other sorting options/parameters can also be provided, but alphabetical and calories 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 baked goods containing chocolate should provide a list of any chocolate-containing baked good. One baked good in the list could then be clicked on to see more information specifically on that baked good; that detail could include a list of ingredients in the baked good; clicking on any ingredient opens up details on the ingredient, including a list of all baked good 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. 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 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 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 baked good or ingredient when searching. Again, you have to provide your own implementation of hashing.
- You have to provide a 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.
- 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 baked goods, ingredients, etc. = 10%
- Create/add facilities (baked goods, recipes and ingredients) = 10%
- Edit/update/delete facilities (baked goods, recipes and ingredients) = 10%
- Search and listing facilities (multiple search options; baked goods and ingredients) = 15%
- Sorting of search results/listings (alphabetical and total calories; baked goods and ingredients) = 10%
- Hashing for individual search/lookup (baked goods and ingredients) = 10%
- Persistence facility (saving and loading data) = 10%
- graphical user interface = 10%
- JUnit testing (minimum of 6-8 useful unit tests) = 5%
- General (commenting, style, logical approach, completeness, etc.) = 10%