Data Structures and Algorithms 1

# CA Exercise 2 – Retro Video Games Information System

---

The objective of this CA exercise is to create an application for storing and retrieving information on retro video games and the machines (i.e. video games consoles or home computers) they ran on. The application should include both hashing and sorting functionality, and provide a JavaFX graphical user interface.

The application should allow/support the following:

- Add a new games machine to the system.
    - Machine name/title (e.g. Atari 2600, ZX Spectrum), manufacturer (e.g. Sega, Sinclair), description, type (e.g. video game console, home computer), media (e.g. cartridge, tape, CD; assume just one media type for simplicity), initial launch year, initial RRP/price, and an image/photo (as a URL) are among key data to store.
- Add a new game to the system.
    - Game name/title, publisher (e.g. Activision, Atari; assume just one publisher for simplicity), description, original developer (company or people names), the original games machine it was developed for, the year of first release, and the original cover art/image (as a URL) are among key data to store.
- Add a new game port for a game. This is where an original game was "ported" to run on an alternative games machine.
    - The original game being ported, the new (ported to) games machine, port developer (company or people names), release year of the game port, and the cover art/image (as a URL) are among key data to store.
        - Note that the release year for a game port cannot be earlier than the original game release year.
- Ability to edit and delete games machine, game, and game port information.
    - Note that deleting e.g. a game will also delete any associated ports.
- Ability to search or filter games machines by (partial) name/title, type, year, manufacturer, media, description keyword, etc.
- Ability to search or filter games and game ports by (partial) name/title, publisher, developer, games machine, year of release, description keyword, etc.
- Listings of search results (for both games machines and games/game ports, as noted above) should be appropriately sorted depending on the chosen search parameters/options. Sorting could be by e.g. games machine/game (incl. game port) name/title, machine type, year of release (ascending or descending), price/RRP, publisher, etc.
    - You should provide for sorting of results in at least 3 different ways for both (1) games machines, and (2) games/game ports (which can be listed together; however, original versions should be appropriately highlighted in listings (e.g. by colour or asterisk or similar) to distinguish them from ports).
- Ability to select and view full details of a specific games machine or game/port by selecting it from a search results listing (as described above). Hashing should be appropriately used here to help efficiently retrieve specific games machine or game/port details.
    - Games machine details will provide the name/title, manufacturer, description, type, media, launch year, and RRP/price. It will also display an image/photo of the games

machine. It will also show a list of all games (incl. game ports) made for the games machine sorted in alphabetical order.
    - Game and game port details should show the game/port's: name/title, publisher, developer, description, the games machine it was developed for, and the release year. It will also display the cover art image of the game/port. It will also show a list of all other versions/ports for the game made for all games machines showing both the machine names/titles along with the version/port release years and sorted in ascending release year order (i.e. newest game version/port last). The original game version should, again, be suitably highlighted in the list.
- Overall, the system should support some level of interactive "drill down" for additional detail or navigation/browsing.
    - For instance, searching games machines for "cartridge" should provide a list of all cartridge-based games machines; one games machine could then be clicked on to see more information specifically on that machine (as outlined above); that detail could include a list of games/ports developed for that machine; clicking on a game/port in that list opens up full details on the game/port (as outlined above), including a list of all other versions/ports that have been made of that game for all games machines; one of those games/ports could be clicked on to see the full details of the relevant games machine along with a list of all games/ports released for it (as outlined above), and so on.
    - The navigation/behaviour of your interactive drill down may differ from this example.
- 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 use databases or anything like that. A single data snapshot/image file is fine for our purposes.

**Notes**

- This is a team CA exercise. Teams consist of 2 students. Students should find/choose their own partners.
- You will have to demonstrate this CA exercise together (either in lab or via Zoom) and you may 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 worked on).
- 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 only use regular arrays for your hash tables.
- 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.
    - Note also that you cannot just depend on JavaFX components to do the sorting for you. While some components might facilitate this, you also have to manually sort results as part of this project too.
- 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 games machine or game/port. Again, you have to provide your own implementation of hashing.
    - You don't have to use hashing everywhere, but try and use it in at least 2 or 3 ways/places to reduce reliance on linear search.
- You have to provide a JavaFX based graphical user interface for the system. Exactly what your user 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 one 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 (games machines, games, game ports, etc.) = 8%
- Create/add facilities (games machines, games and game ports) = 12%
- Edit and delete facilities (games machines, games and game ports) = 10%
- Search and listing facilities (multiple search options; games machines and games/ports) = 10%
- Select for further details / interactive drill down facilities = 10%
- Sorting of search results/listings (games machines, game ports, games) = 15%
- Hashing for individual search/lookup (games machines, game ports, games) = 15%
- Persistence facility (saving and loading data) = 5%
- JavaFX graphical user interface = 5%
- JUnit testing (minimum of 6-8 useful unit tests) = 5%
- General (commenting, style, logical approach, completeness, etc.) = 5%