Data Structures and Algorithms 1

CA Exercise 1 – Jewellery Store Management System

The objective of this CA exercise is to create a "jewellery store management system" in Java that makes heavy use of custom-built internal data structures. The system should allow the user to manage a jewellery store that consists of multiple display cases. Every display case contains multiple display trays. Every tray can contain multiple items of jewellery (e.g. rings, watches, necklaces, etc.). Every item of jewellery consists of one or more materials/components (e.g. gold, platinum, diamonds, etc.), and these materials/components will vary in quantity (by a suitable unit of weight e.g. grams or carats) and quality (e.g. karat for gold, clarity for diamonds, etc.) for different items of jewellery. Therefore, in summary, an item of jewellery is comprised of one-or-more materials/components of varying quantity and quality; an item of jewellery has to be stored on a display tray; a display tray must be stored in a display case.

The system should allow the user to:

- Add a new <u>display case</u> to the system. The following information should be stored for each display case: unique display case number/identifier, type (wall-mounted or freestanding), and lighting (lit or unlit).
- Add a new <u>display tray</u> to a display case. The following information should be stored for each display tray: display tray identifier (one letter followed by a number e.g. A12), inlay material colour (e.g. green), and the display tray dimensions (width x depth) in centimetres.
    - Note that the display tray identifier must be unique in the system, so check that no other display tray in any other display case has the same identifier when creating a new display tray.
- Add an <u>item of jewellery</u> to a display tray. Properties that should be stored: item description (free text to include style, etc.), type (ring, watch, necklace, etc.), target gender (male, female, unisex, etc.), image (as a URL), and retail price.
- Add a <u>material/component</u> to an item of jewellery (in a display tray on a display case). Material/component records should have properties: name/type (e.g. gold, platinum, diamond, emerald, silver, etc.), description/information (free text e.g. type of cut/shape/colour if a gemstone), quantity/amount/weight (in a numeric form), and quality (in a suitable numeric form).
- <u>View all stock</u> in the jewellery store. This would show/list all jewellery items (with suitable information, photos, etc.) on all display trays in all display cases.
- <u>Interactively "drill down"</u> through display cases, display trays, and jewellery items to see full details (material/components etc.) of a specific item of jewellery using an appropriate GUI.
- <u>Search for items of jewellery by text</u>. The system will systematically search for all matches for the given search term in every item of jewellery and its materials/components information. It will then report the search results (in list form) on where matches are stored in the jewellery store (i.e. which display cases and display trays). The user should then be able to choose a search result from the list to see a full description of the given item of jewellery (including photo, materials/components information, retail price, etc.).

- A “smart add” facility for jewellery items. Given the jewellery item’s description, type, value, etc., the system will automatically identify a suitable place (i.e. display case and tray) to store them. For instance, the system might try to store “similar items” together in the same display tray and/or case. For instance, all platinum diamond rings might logically share a tray (or case), as might similarly-priced gents watches, etc.
    - You can have the “smart add” behave any way you think appropriate.
- Remove item of jewellery. The user should be able to identify the item of jewellery to remove in an easy/appropriate way.
- Value stock facility. This facility should systematically value (1) the contents of every display case and display tray individually, and (2) the total value of all cases/trays in the store.
- Reset facility. Clears all system data.
- Save and load the entire system data to support persistence between executions.
    - This can be done using any suitable file format (e.g. CSV, XML, binary, etc.). There is no need to use any database system beyond this.
- Other appropriate facilities to manage the jewellery store as you see fit.
    - Remember that you will be strictly marked according to the marking scheme below, so be careful not to deviate too far from the project specification.

Notes

- This is an individual CA exercise, and students should work by themselves to complete it.
- You will have to demonstrate this CA exercise in the lab sessions and be interviewed on various aspects of it. You must be able to answer all questions on any code you are forwarding as your own.
- This CA exercise is worth 35% of your overall module mark.
- You should implement a suitable JavaFX graphical user interface to interact with your system.
    - The GUI does not have to be particularly fancy, but should nevertheless be functional.
- You should implement a set of JUnit classes to test your code systematically as you develop it. Exactly what you test for is up to you, but make sure that you demonstrate a test-driven approach to your development.
- It is important to note that 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 also cannot use regular arrays directly. You essentially have to implement the required data structures and algorithms from scratch and from first principles (in line with the module learning outcomes). This is deliberate, as I want you to get your hands dirty.
    - You will have to create numerous custom ADTs such as DisplayCase, DisplayTray, JewelleryItem, etc. (your classes/class names may differ) and use some form of linked-list implementation.
    - You should also avoid simply storing data using the JavaFX components themselves. The JavaFX components can be used to display data/information as required, but the main data should also be stored separate from the JavaFX components.

Indicative Marking Scheme

- Appropriate custom ADTs = 10%
- Create/add facilities (display case, display tray, item of jewellery, material/component) = 12%
- View all stock (listing) = 8%
- Interactive view stock = 7%
- Search for and select/view items of jewellery = 10%
- Smart add for items of jewellery = 10%
- Remove item of jewellery = 5%
- Value stock facility = 10%
- Reset facility = 5%
- Save/load facility = 8%
- Appropriate JavaFX user interface = 5%
- JUnit testing (minimum of 8-10 useful unit tests) = 5%
- General (commenting, style, logical approach, completeness, etc.) = 5%