Implementing Efficient Data Structures for SCUPI+

Chen Qixuan

Introduction

In the development of the SCUPI+ Java application, efficient data storage and retrieval mechanisms were crucial to ensure smooth querying of extensive film data, crew credits, and user ratings. This report discusses the design decisions and implementation details behind the chosen data structures for three primary data stores: Movies, Credits, and Ratings. The aim was to balance performance, scalability, and ease of integration with existing application components.

Justification of the Custom ArrayList Selection

1. It is mainly because of the reference to the keywards class. As It was my first time doing coursework, I had no clue at first, and it took me a long time to look at the written keywards class before I understood what to do. Because the ArrayList is less difficult to code, so I directly uses ArrayList like the keywards class, and create a custom ArrayList.
2. The underlying principle of the ArrayList is array, with efficient indexes for traversal and random access.
3. It's easy to add data. You can put data on top of the stack in turn, just like a stack.
4. Finding data is simple, because all the different data are stored in alignment (such as the time, country, and actors of a movie, although they are placed in different containers, but their indexes are the same), so you only need to iterate through the ArrayList that stores one data(such as the ID of the movie) to find its index, and then you can find the corresponding other data, and then perform various processes on the data (such as modification, deletion, etc.).
5. ArrayList can be automatically scaled to make reasonable use of storage space.

Implementation of Operations in the 3 Data Stores

6. There are three main types of methods to be achieved： **add() remove() get()** and **set()**.
7. Before you can implement these methods, you first need to initialize the container. However, an issue was found when initializing the ArrayList that holds Float types. The Float class exists in multiple packages, and Eclipse (my IDE) recognizes it by default in the package I don't want, resulting in an error that

takes most of the time to solve.

8. The implementation of the **add** method is very simple: call the add method in the ArrayList directly, and the data will be added to the array in turn.
9. The **remove** method is also very simple, traversal to find its index, and the following ones can be filled in order (the same is true for the **set** method).
10. When it comes to the **get** method, if you only look for a specific piece of data, then you can still iterate. However, if the return value is based on a set of data (such as the top 10 highest-rated movies), you need to summarize the ratings of each movie first, and then sort them according to the ratings, and finally return an array. One of the most confusing is the determination of the size of the array, just initializing a large array is not possible, because the length of the array is the size of the entire array rather than the size of the stored data, if it is confused, it will lead to a lot of errors, such as there is no data under the corresponding index, the array is out of bounds, and the return is not an empty array and other problems (all of which are my own experience). So, for the details of the get method, I added the Arrays class to the structure, which includes a sorting algorithm and a **copyOf** method (which allows the array to store exactly the same amount of data as the length of the array), and finally solved the problems.

Conclusion

The adoption of ArrayList in the SCUPI+ Java application is rooted in its simplicity and efficiency for managing extensive film data. Inspired by the existing Keywords class implementation, ArrayList was chosen due to their ease of use, rapid data addition, and direct access via indexing. They automatically handle storage expansion and facilitate straightforward removal and retrieval operations. Challenges arose with initializing a float-type ArrayList due to namespace conflicts, resolved after considerable debugging. Implementing core methods like add, remove, get, and set required careful consideration, particularly for the get method which involves sorting and dynamically sizing arrays. Utilizing the Arrays class for operations like sorting and resizing ensured effective handling of data arrays, overcoming issues like incorrect size assumptions and out-of-bounds errors. Overall, the ArrayList enabled efficient data structuring and manipulation within the Movies, Credits, and Ratings classes.