Developing a modern collections library

Usage figures suggest that 56% [1] of all initialisations of a collection library object in java are for some kind of list, with maps and sets being 28% [1] and 15% [1] respectively. This backs up the view that only three collection types are needed. For any others they can be implemented by the three mentioned with relative ease. The library will fill a whole that currently exists inside of java. There are many specialised java collections libraries. This leaves developers in a position where they can struggle to find the correct implementation to suit their needs. Potentially their code could run quicker if they had selected a slightly different collections library. However, in languages like python there exists very few but powerful collection libraries. It is much easier for the developers to find what they are looking to use.

The research I am aiming to undertake is the development of a modern collection’s library. Allowing for development to feel more streamlined in the future. There are a few features that I will be considering for development in this language: Sets, Sequences and Maps. The idea is to have a few very dynamic and powerful collections at the fingertips of developers. Additionally, improving the runtime on java’s native collections library. This would help in the development of more efficient java programs going forward.

Java arrays are a fixed length type of collections. When instantiated a size is declared and it cannot change from there, it is not dynamic. However, in Python the default list is dynamic and can be added to as the developer wishes. In Java this functionality would be achieved by ArrayList and is not a default. Python as a language supports many of Javas specialised collection classes inside of its default library. The sequence for example can act as any number of data types from Strings to Queues and Stacks. It’s a very powerful class.

Typically a dynamic language such as Python will have relatively poor performance at storing collections compared to that of Java which is a statically typed language [3]. I will be aiming to take the benefits of dynamic collections from languages like Python and port them over to Java and see the benefits of a typed language.

The plan for development would be to first get the basic functionality of a collection’s library completed. This functionality would be for example ordered collections. This would need certain features to be functional. These could be working as a queue, a stack, an array list and even the ability to sort unordered lists into ordered lists etc. Storage of the collections will need to be considered. Storage optimisation has been looked into and strategies mentioned in the paper by Carl Friedrich Bolz, Lukas Die kmann, Laurance Tratt will be used in order to speed up development in that section [3].

During development I will be creating test scripts to ensure that all functionality is met. These will ensure robustness and reduce the likelihood that I let any unintended functionality slip through. They will be run automatically on commit to ensure that I do not push untested code to the main branch.

One of the ways that the library will be tested is through the conversion of old programs developed by myself to use the new collections library. This will demonstrate its ease of use and efficiency bonuses. It may also demonstrate the ease of refactoring code to use the new library. In addition to converting old programs over to using the new collections library I would develop new simple algorithms using my collections in order to demonstrate the ease of development with them and show the utility they offer to other developers.

Example algorithms:

* Adding 1000 items to a sorted list with no specified order.
* Testing search times through a HashSet, Depending on the Load Factor (How full the set is) Should have an effect on the performance here.
* Large scale sorting operations being performed on unsorted lists.

After this stage of converting the programs over I will attempt to make any refinements to the library that could improve efficiency and benchmark the programs made with the standard library with programs made with my library.

A timeframe for completion of the research would be about 7 months from October 2024 to April 2025. The first four months would be spent developing the library. November through February writing test scripts that correspond to features available inside of the library. January through April I will work on program conversion. This is scheduled for when most of the core features have been implemented and a general direction is visible. Writing will be through February into April. Finally, improvements that are needed will be added through march and into April these being “finishing touches”.

A screenshot of a calendar

Description automatically generated

The library will be used instead of the built-in java collections library in projects going forwards. Anything that needs parallel processing support and needs to run efficiently will use this library as it will have the features needed to achieve this.

Further research could perhaps delve into code discoverability which seems to be an issue in languages with vast libraries. When there are too many collections to choose from it can become hard to pick the correct one for the specific implementation at hand. So, some research into this would be useful in aiding library designers.

Another area that could use some more research and could potentially follow on quite nicely from this project would be secure and efficient parallel processing. This is done securely by java but not in the most efficient way. “*The simplest solution to ensure correctness is to provide synchronization wrappers as done by Java and C#. However, this approach is rarely efficient and often lacks support for performing multiple operations safely together*” [1]. Javas concurrency support is implemented via the Java.Util.Concurrent class it is secure and works. However, in a paper comparing the Go language to Java, where they tested matrix multiplication with and without concurrency, they found that Java scaled worse that Go when using the concurrent class. Without concurrency Java performed much better with larger matrices [2].

**References**

[1] Marr, S. and Daloze, B. (2018). Few Versatile vs. Many Specialized collections: How to Design a Collection Library for Exploratory programming? *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming*, [online] 6183, pp.135–143. doi:https://doi.org/10.1145/3191697.3214334.

[2] Togashi, N. and Klyuev, V. (2014). Concurrency in Go and Java: Performance Analysis. *2014 4th IEEE International Conference on Information Science and Technology*. doi:https://doi.org/10.1109/icist.2014.6920368.

[3] Carl Friedrich Bolz, Diekmann, L. and Tratt, L. (2013). Storage Strategies for Collections in Dynamically Typed Languages. *OOPSLA ’13: Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*. doi:https://doi.org/10.1145/2509136.2509531.