Garbage Collection in Python and Java

Garbage Collection is a very important part of memory management in programming. It ensures that memory is not wasted and properly managed. In this project, we delved into GC implementations in two common programming languages; Python and Java.

Python Garbage Collection Simulator

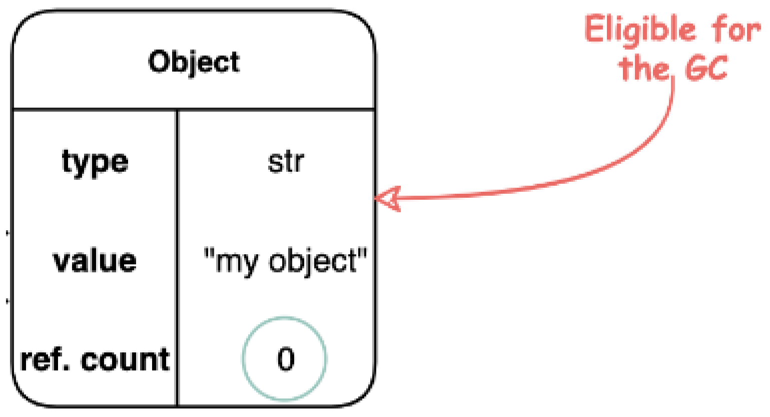
The Python GC simulator is made up of a straightforward algorithm for object allocation and collection. The 'allocate_object' function sequentially scans the heap to find an empty slot, simulating object allocation. The 'collect_garbage' function then processes the provided root indices, updating moved roots and swapping the old heap with the new heap. This simplistic approach to allocation and garbage collection makes this code most suitable for small-scale applications, or more so learning the general idea of how a complete garbage collector really works.

Memory Usage and Edge Cases

The simulator is based on a heap structure, moved roots set, and a mapping table. Memory usage mainly depends on the size of the heap and the number of allocated objects.
Edge cases, such as dealing with cyclic references or complex object graphs, are not addressed in this simulator as it is not a complete garbage collector. While Python's automatic garbage collector handles such cases, this simulator serves as a simple representation.

Programming Paradigms

The simulator makes use of procedural and object-oriented paradigms. The procedural parts are evident in the sequential execution of steps, while object-oriented principles can be seen in encapsulating heap-related functionalities within a class.



Java Garbage Collection

Computational Complexity and Execution Time

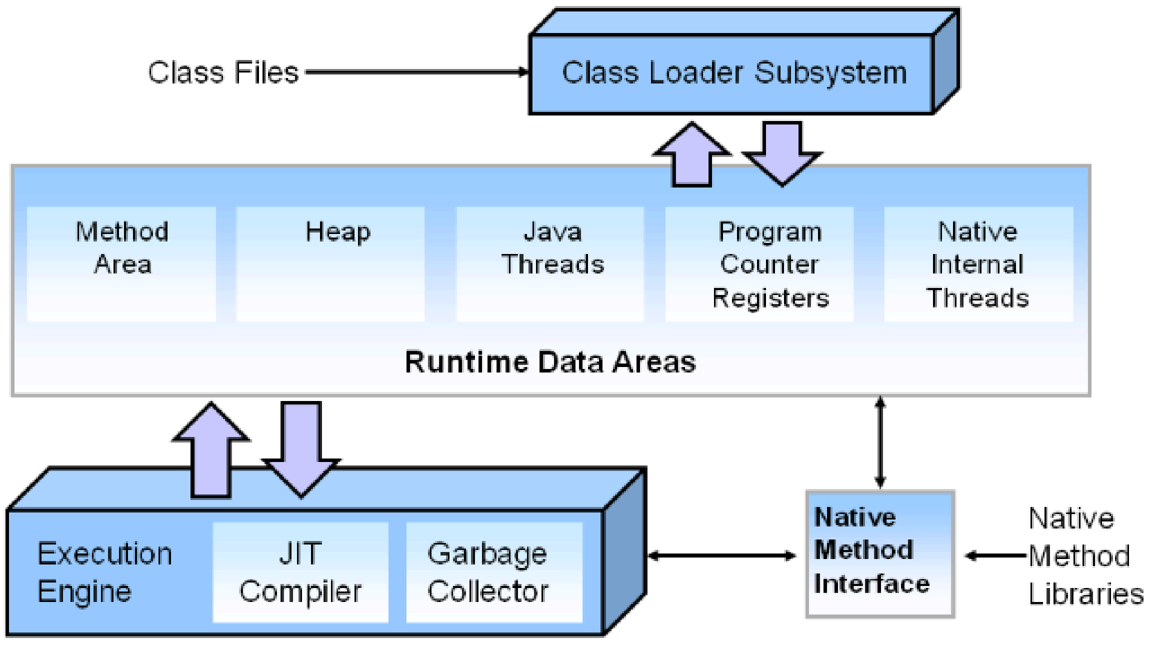
In the Java implementation, the ‘CustomGarbageCollector’ class uses a simple list to manage allocated objects. The ‘collectGarbage’ method iterates through the list, applying a custom disposal condition. This simplistic approach results in a linear time complexity, similar to the Python simulator.

Memory Usage and Edge Cases

Java's automatic garbage collector handles complex scenarios, including cyclic references and circular dependencies. However, the custom garbage collector presented here is not as featureful. Memory usage is tied to the size of the allocated objects and the disposal condition, which, in this case, is a random test condition.

Programming Paradigms

The Java implementation follows an object-oriented paradigm. The CustomGarbageCollector class encapsulates memory management functionalities, matching the principles of encapsulation and abstraction.



Comparisons

Existing Solutions
Both Python and Java have automatic garbage collectors. Python's cyclic garbage collector handles circular references effectively, while Java's Garbage-First collector is best when it comes to large-scale applications with dynamic memory requirements. The custom collectors presented here, however, are not as sophisticated as the automatic counterparts, they are more so aimed to help simulate and understand the concept of a garbage collector.

Execution Time and Memory Usage
While both implementations show linear time complexity, Python's dynamic typing and memory management may introduce additional overhead. Java, with its static typing and powerful garbage collection mechanisms, may provide more predictable performance.

Breaking Points and Heuristic Approach
Both implementations face challenges in handling complex scenarios, especially cyclic references. Python's simulator has limitations with large heaps, while Java's custom collector lacks the heuristics of the automatic garbage collector.

Importance of GC
This project highlights the important role of GC in managing resources effectively. Efficient garbage collection ensures optimal memory utilisation, preventing memory leaks and enhancing overall system performance. The automatic collectors in Python and Java are an example of the importance of efficient memory management in modern software development.

Conclusion (What we learned)
Throughout the process of learning how and why garbage collection is used, not only did we learn the technical aspect of garbage collection but also the real-world use of it. Prior to this project we were under the impression that custom garbage collection would almost always be the better option but after finding out the ins and outs of the process, we came to realisation that in almost any case, utilising the built-in java and/or python garbage collector is most definitely better, and in real world cases the vast majority of developers do not bother to create a custom one. This is because it is not worth the complexity and time taken to create such custom garbage collector, while also adding more lines of code when not necessary, contradicting the standard practice of a developer which is minimising code length, and also because of the fact that the garbage collectors built into java and python are incredibly efficient, innovative, and useful as they already are. On the other

hand, we can understand how in very few cases a custom garbage collector can prove useful. Such as when the use of a programming language which does not provide automatic garbage collection is required.

References for diagrams:

- https://jenaiz.com/2022/02/understanding-how-the-python-garbage-collector-works/
- https://performancestack.in/garbage-collection-in-java/