Principles of Programming
Tutorial and Practical - Set 4

Name: Ashok Malhotra

Enrolment no: IIB2020501

Ans:1,

Ans:2nd Golang garbage collection has all goroutines reach a garbage collection safe point with a process called stop the world. This temporarily stops the program from running and turns a write barrier on to maintain data integrity on the heap. This allows for concurrency by allowing goroutines and the collector to run simultaneously.

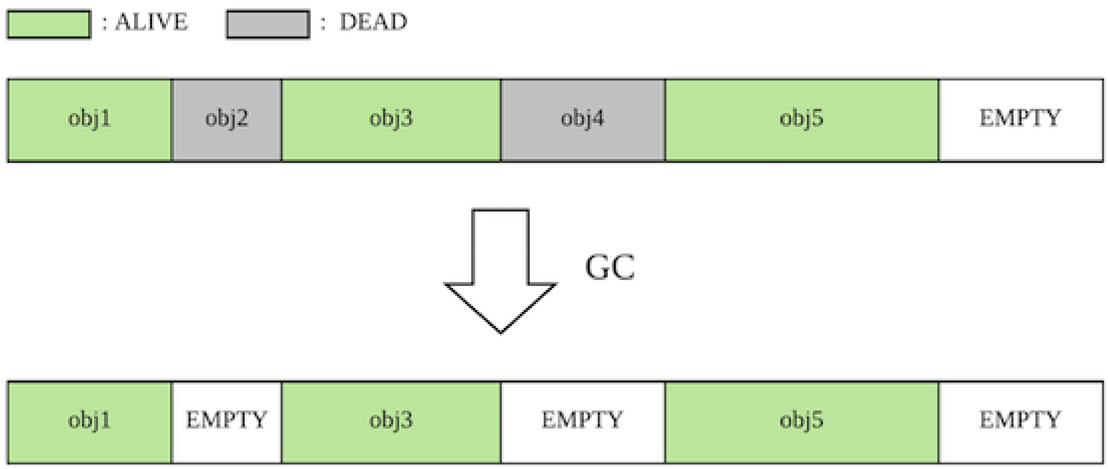
	Java (Java8 HotSpot VM)	**Golang collection**
Collector	Several collectors (Serial, Parallel, CMS, G1)	CMS
Compaction	Compacts	Does not compact
Generational GC	Generational GC	Non-generational GC
Tuning parameters	Depends on the collector. Multiple parameters available.	GOGC only

Compaction

Garbage collection can either be non-moving or moving.

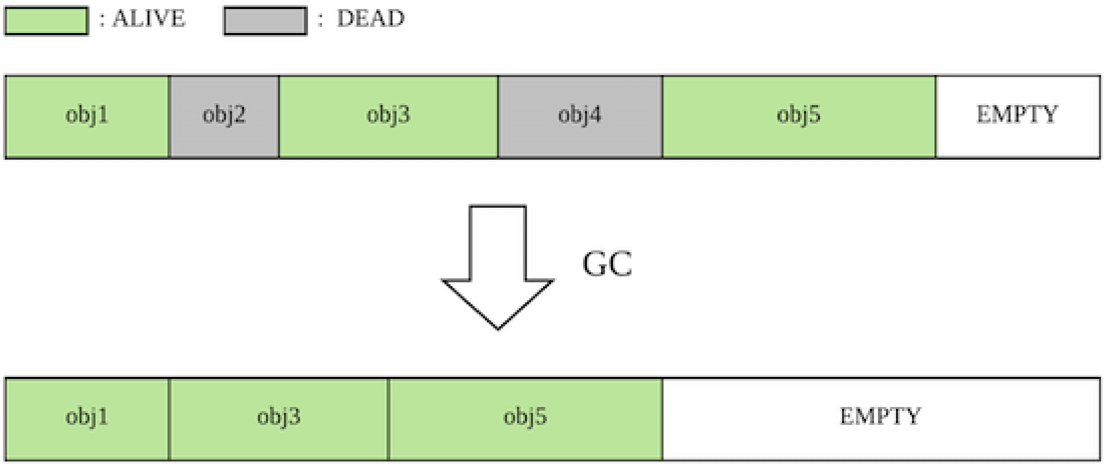
Non-moving GC

Non-moving garbage collectors do not relocate objects in a heap. CMS, the collector Go uses, is non-moving. Generally, if you repeat memory allocation and deallocation in non-moving garbage collection, you end up with heap fragmentation, and thus lessened performance for allocation. But, of course, this will depend on how you implement the memory allocator.



Moving GC

Moving garbage collectors compact the heap by relocating alive objects to the end of the heap. An example of a moving garbage collection is Copying GC, which is used on HotSpot VM.



Compaction has the following merits:

- Avoiding fragmentation
- Being able to implement a high-performance memory allocator thanks to bump allocation (Since all objects are located at the end of the heap, we can increment right at the end for new memory allocation.)

Ans:3rd

Heap space is used for the dynamic memory allocation of Java objects and JRE classes at runtime. New objects are always created in heap space, and the references to these objects are stored in stack memory.

When an object is allocated from the managed heap, the new operator returns the memory address of the object. You usually store this address in a variable. This is called a reference type variable because the variable does not actually contain the object's bits; instead, the variable refers to the object's bits.

In addition to reference types, the virtual object system supports lightweight types called value types. Value type objects cannot be allocated on the garbage-collected heap, and the variable representing the object does not contain a pointer to an object; the variable contains the object itself. Since the variable contains the object, a pointer does not have to be dereferenced in order to manipulate the object. This, of course, improves performance.

Parameter	**Stack Memory**	**Heap Space**
Application	Stack is used in parts, one at a time during execution of a thread	The entire application uses Heap space during runtime
Size	Stack has size limits depending upon OS,	

	and is usually smaller than Heap	There is no size limit on Heap
Storage	Stores only primitive variables and references to objects that are created in Heap Space	All the newly created objects are stored here
Efficiency	Much faster to allocate when compared to heap	Slower to allocate when compared to stack
Order	It's accessed using Last-in First-out (LIFO) memory allocation system	This memory is accessed via complex memory management techniques that include Young Generation, Old or Tenured Generation, and Permanent Generation.
Life	Stack memory only exists as long as the current method is running	Heap space exists as long as the application runs
Allocation /Deallocation	This Memory is automatically allocated and deallocated when a method is called and returned, respectively	Heap space is allocated when new objects are created and deallocated by Gargabe Collector when they're no longer referenced