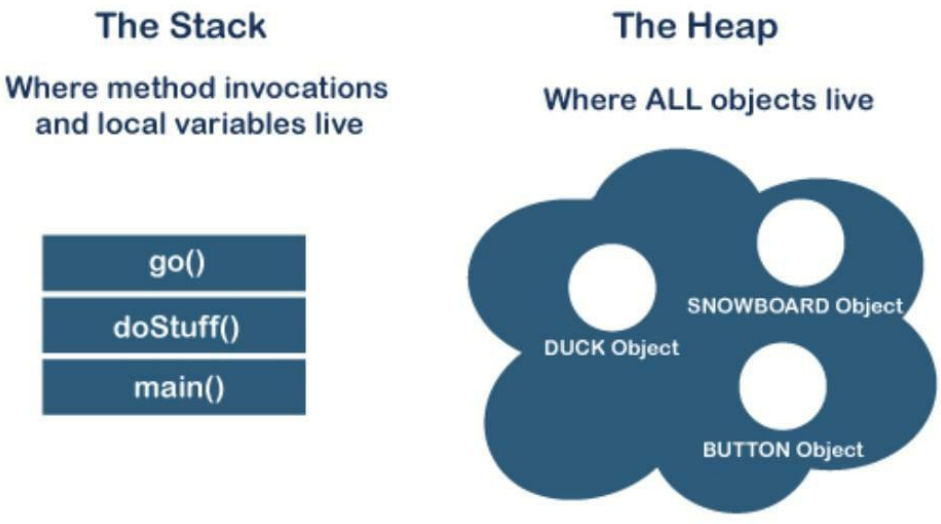
The major difference between Stack memory and heap memory is that the stack is used to store the order of method execution and local variables while the heap memory stores the objects and it uses dynamic memory allocation and deallocation. In this section, we will discuss the differences between stack and heap in detail




Stack Vs Heap

## Stack Memory

The stack memory is a physical space (in RAM) allocated to each thread at run time. It is created when a thread creates. Memory management in the stack follows LIFO (Last-In-First-Out) order because it is accessible globally. It stores the variables, references to objects, and partial results. Memory allocated to stack lives until the function returns. If there is no space for creating the new objects, it throws the java.lang.StackOverFlowError. The scope of the elements is limited to their threads. The JVM creates a separate stack for each thread

## Heap Memory

It is created when the JVM starts up and used by the application as long as the application runs. It stores objects and JRE classes. Whenever we create objects it occupies space in the heap memory while the reference of that object creates in the stack. It does not

follow any order like the stack. It dynamically handles the memory blocks. It means, we need not to handle the memory manually. For managing the memory automatically, Java provides the garbage collector that deletes the objects which are no longer being used. Memory allocated to heap lives until any one event, either program terminated or memory free does not occur. The elements are globally accessible in the application. It is a common memory space shared with all the threads. If the heap space is full, it throws the java.lang.OutOfMemoryError. The heap memory is further divided into the following :memory areas


| Parameter | Stack | Heap |
|---|---|---|
| Type of data structures | A stack is a linear data structure. | Heap is a hierarchical data structure. |
| Access speed | High-speed access | Slower compared to stack |
| Space management | Space managed efficiently by OS so memory will never become fragmented. | Heap Space not used as efficiently. Memory can become fragmented as blocks of memory first allocated and then freed. |
| Access | Local variables only | It allows you to access variables globally. |
| Limit of space size | Limit on stack size dependent on OS. | Does not have a specific limit on memory size. |
| Resize | Variables cannot be resized | Variables can be resized. |
| Memory Allocation | Memory is allocated in a contiguous block. | Memory is allocated in any random order. |
| Allocation and Deallocation | Automatically done by compiler instructions. | It is manually done by the programmer. |
| Deallocation | Does not require to de-allocate variables. | Explicit de-allocation is needed. |
| Cost | Less | More |
| Implementation | A stack can be implemented in 3 ways simple array based, using dynamic memory, and Linked list based. | Heap can be implemented using array and trees. |
| Main Issue | Shortage of memory | Memory fragmentation |
| Locality of reference | Automatic compile time instructions. | Adequate |
| Flexibility | Fixed size | Resizing is possible |
| Access time | Faster | Slower |