Stack vs Heap Memory Allocation

Difficulty Level : Easy Last Updated : 21 Jun, 2021

Memory in a C/C++/Java program can either be allocated on a stack or a heap.
Prerequisite: Memory layout of C program.

Stack Allocation: The allocation happens on contiguous blocks of memory. We call it a stack memory allocation because the allocation happens in the function call stack. The size of memory to be allocated is known to the compiler and whenever a function is called, its variables get memory allocated on the stack. And whenever the function call is over, the memory for the variables is de-allocated. This all happens using some predefined routines in the compiler. A programmer does not have to worry about memory allocation and de-allocation of stack variables. This kind of memory allocation also known as Temporary memory allocation because as soon as the method finishes its execution all the data belongs to that method flushes out from the stack automatically. Means, any value stored in the stack memory scheme is accessible as long as the method hasn't completed its execution and currently in running state.

Key Points:

- It's a temporary memory allocation scheme where the data members are accessible only if the method( ) that contained them is currently is running.
- It allocates or de-allocates the memory automatically as soon as the corresponding method completes its execution.
- We receive the corresponding error Java. lang. StackOverFlowError by JVM, If the stack memory is filled completely.
- Stack memory allocation is considered safer as compared to heap memory allocation because the data stored can only be access by owner thread.
- Memory allocation and de-allocation is faster as compared to Heap-memory allocation.
- Stack-memory has less storage space as compared to Heap-memory.

```
int main()
{
   // All these variables get memory
   // allocated on stack
   int a;
   int b[10];
```

```
    int n = 20;
    int c[n];
}
```

Heap Allocation: The memory is allocated during the execution of instructions written by programmers. Note that the name heap has nothing to do with the heap data structure. It is called heap because it is a pile of memory space available to programmers to allocated and de-allocate. Every time when we made an object it always creates in Heap-space and the referencing information to these objects are always stored in Stack-memory. Heap memory allocation isn't as safe as Stack memory allocation was because the data stored in this space is accessible or visible to all threads. If a programmer does not handle this memory well, a memory leak can happen in the program.

The Heap-memory allocation is further divided into three categories:- These three categories help us to prioritize the data(Objects) to be stored in the Heap-memory or in the Garbage collection.

- **Young Generation –** It's the portion of the memory where all the new data(objects) are made to allocate the space and whenever this memory is completely filled then the rest of the data is stored in Garbage collection.
- **Old or Tenured Generation –** This is the part of Heap-memory that contains the older data objects that are not in frequent use or not in use at all are placed.
- **Permanent Generation –** This is the portion of Heap-memory that contains the JVM's metadata for the runtime classes and application methods.

Key Points:

- We receive the corresponding error message if Heap-space is entirely full, java. lang.OutOfMemoryError by JVM.
- This memory allocation scheme is different from the Stack-space allocation, here no automatic de-allocation feature is provided. We need to use a Garbage collector to remove the old unused objects in order to use the memory efficiently.
- The processing time(Accessing time) of this memory is quite slow as compared to Stack-memory.
- Heap-memory is also not threaded-safe as Stack-memory because data stored in Heap-memory are visible to all threads.
- Size of Heap-memory is quite larger as compared to the Stack-memory.
- Heap-memory is accessible or exists as long as the whole application(or java program) runs.

```
int main()
{
   // This memory for 10 integers
   // is allocated on heap.
   int *ptr  = new int[10];
}
```

Intermixed example of both kind of memory allocation Heap and Stack in java:

```
class Emp {
    int id;
    String emp_name;

    public Emp(int id, String emp_name) {
        this.id = id;
        this.emp_name = emp_name;
    }
}

public class Emp_detail {
    private static Emp Emp_detail(int id, String emp_name) {
        return new Emp(id, emp_name);
    }

    public static void main(String[] args) {
        int id = 21;
        String name = "Maddy";
        Emp person_ = null;
        person_ = Emp_detail(id, emp_name);
    }
}
```

Following are the conclusions on which we'll make after analyzing the above example:

- As we start execution of the have program, all the run-time classes are stored in the Heap-memory space.
- Then we find the main() method in the next line which is stored into the stack along with all it's primitive(or local) and the reference variable Emp of type Emp_detail will also be stored in the Stack and will point out to the corresponding object stored in Heap memory.
- Then the next line will call to the parameterized constructor Emp(int, String) from main( ) and it'll also allocate to the top of the same stack memory block. This will store:
  - The object reference of the invoked object of the stack memory.
  - The primitive value(primitive data type) int id in the stack memory.
  - The reference variable of String emp_name argument which will point to the actual string from string pool into the heap memory.
- Then the main method will again call to the Emp_detail() static method, for which allocation will be made in stack memory block on top of the previous memory block.
- So, for the newly created object Emp of type Emp_detail and all instance variables will be stored in heap memory.

Pictorial representation as shown in the Figure.1 below:

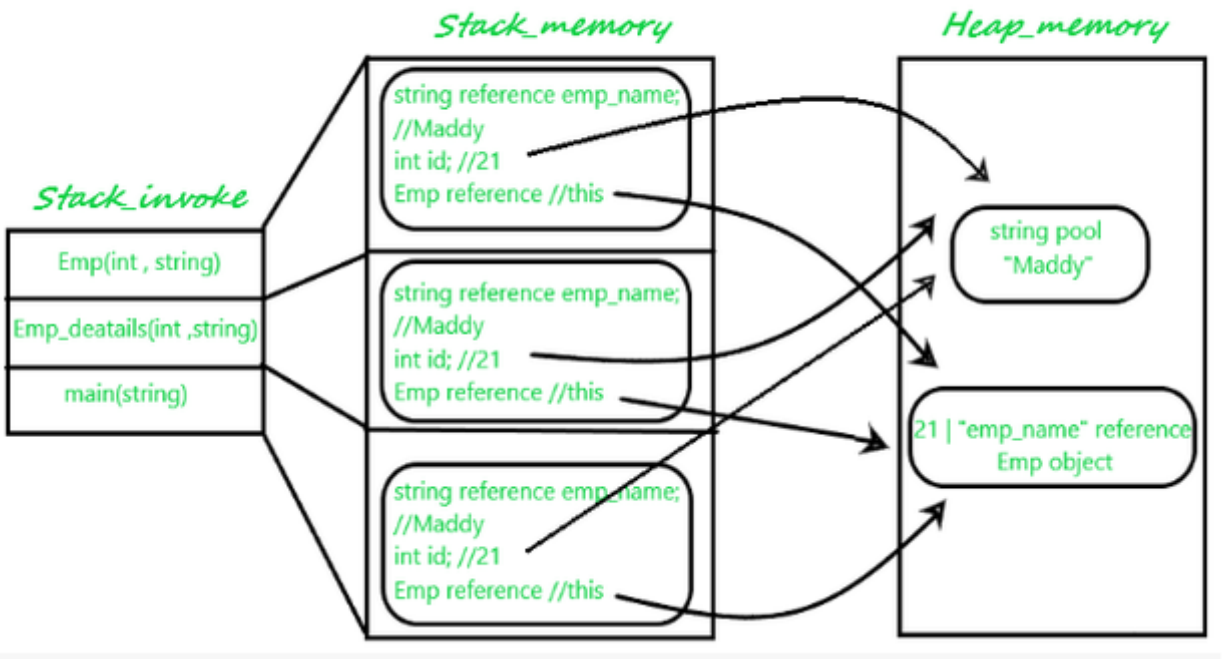

Fig.1

Key Differences Between Stack and Heap Allocations

1. In a stack, the allocation and de-allocation are automatically done by the compiler whereas in heap, it needs to be done by the programmer manually.
2. Handling of Heap frame is costlier than the handling of the stack frame.

3. Memory shortage problem is more likely to happen in stack whereas the main issue in heap memory is fragmentation.
4. Stack frame access is easier than the heap frame as the stack have a small region of memory and is cache-friendly, but in case of heap frames which are dispersed throughout the memory so it causes more cache misses.
5. A stack is not flexible, the memory size allotted cannot be changed whereas a heap is flexible, and the allotted memory can be altered.
6. Accessing time of heap takes is more than a stack.

Comparison Chart

Parameter	STACK	HEAP
Basic	Memory is allocated in a contiguous block.	Memory is allocated in any random order.
Allocation and De-allocation	Automatic by compiler instructions.	Manual by the programmer.
Cost	Less	More
Implementation	Easy	Hard
Access time	Faster	Slower
Main Issue	Shortage of memory	Memory fragmentation
Locality of reference	Excellent	Adequate
Safety	Thread safe, data stored can only be accessed by owner	Not Thread safe, data stored visible to all threads
Flexibility	Fixed-size	Resizing is possible
Data type structure	Linear	Hierarchical