Week 1

Java Memory Management

**What Is Java Stack?**

A Java stack is part of your computer’s memory where temporary variables, which are created by all functions you do, are stored. It is used to execute a thread and may have certain short-lived values as well as references to other objects. It uses LIFO data structure, or [last in first out](http://www.journaldev.com/4098/java-heap-space-vs-stack-memory).

What does this mean? When a method is invoked, it creates a new block in the stack for that particular method. The new block will have all the local values, as well as references to other objects that are being used by the method. When the method ends, the new block will be erased and will be available for use by the next method. The objects you find here are only accessible to that particular function and will not live beyond it.

This makes it very easy to keep track of the stack, where the latest reserved block is also the first to be freed. The variables created for the method are directly stored in the memory, allowing for fast access.

The memory size of a Java stack is generally much less than in a Java heap space because when a method ends, all the variables created on the stack are erased forever.

## What Is Java Heap?

Java objects are in an area, which is called the heap. It is created when the program is run, and its size may decrease or increase as your program runs. It can easily become full, and when it does, garbage collection is initiated. This is when objects that are no longer used are deleted to make way for new objects.

Unlike in a Java stack where memory allocation is done when your program is compiled, in a heap it is allocated as your program is run. Accessing variables placed here is a bit slower compared to a stack’s direct and fast access.

Heap is likened to a [global memory pool](https://www.quora.com/How-is-java-memory-pool-divided). A method or function will use the heap for memory allocation if you need the data or variables to live longer than the method or function in question. The objects you find here are accessible to all the functions.

Also, there is no specific order in reserving blocks in a heap. You can allocate blocks at any time, and then you can free it when you wish. As you can imagine, it is much more complex to keep track of the parts that are free and can be allocated, but it can also be divided into two generations or sub-areas.

These sub-areas are called the young space (or nursery) and the old space. The young space is typically earmarked for the memory allocation for new objects. When the young space becomes full, [garbage collection](https://stackify.com/what-is-java-garbage-collection/) happens. Short-lived or temporary objects typically use the young space. This help makes garbage collection faster when compared to a heap without any divisions.

## In a Nutshell…

### **Stack**

* The size of the stack will vary as methods and functions create and delete local variables as needed.
* Memory is allocated and then subsequently freed without you needing to manage the memory allocation.
* Stack has size limits, which can vary according to the operating system you use.
* Variables that are stored on the stack exist for as long as the function that created them are running.

### **Heap**

* Memory is not managed automatically nor is it as tightly managed by the central processing unit the way stack is managed. You would need to free allocated memory yourself when these blocks are no longer needed.
* The heap is prone to memory leaks, where memory is allocated to unused objects and will not be available to processes other than that.
* There is no size limit in the heap.
* Compared to stack, objects in the heap are much slower to access. It is also slower to write to the memory on the heap.

Stack is easier and faster to use, but it comes with a lot of limitations that you can ignore if you use heap.

[**When do you use stack?**](https://softwareengineering.stackexchange.com/questions/65281/stack-and-heap-memory-in-java) Stack can only be used for local variables that use up small amounts of memory. The good news is that memory allocation and management is not going to be your problem and access to these objects is very fast. It does suffer from size limitations and the fact that you cannot resize variables on the stack.

[**When do you use heap?**](https://softwareengineering.stackexchange.com/questions/65281/stack-and-heap-memory-in-java) You use the heap to allocate memory if there are variables that you need to be accessed globally, as opposed to just being available only to the methods and functions that created it. Heap is also good when you have a need for a lot of memory since it has no limit on memory size. You can also resize the variables on the heap.

# **Asynchronous and Synchronous Callbacks in Java**

CallBack Function is a function which passed into another function as an argument and is expected to execute after some kind of event. The purpose of the callback is to inform a class Sync/Async if some work in other class is done

**Synchronous Callback**

The code execution will block or wait for the event before continuing. Until your event returns a response your program will not execute any further. So Basically the callback performs all its work before returning to the call statement. The problem with synchronous callback is that they appear to lag.

**Asynchronous Callback**

An Asynchronous call do not block the program from the code execution. when the call returns from the event the call returns back to the callback function. So in the context of java we have to Create a new thread invoke the callback method inside that thread. Callback may be invoked from a thread but is not a requirement. A Callback may also start a new thread thus making themselves asynchronous.