A **memory leak** occurs when a [computer program](http://en.wikipedia.org/wiki/Computer_program) incorrectly manages [memory allocations](http://en.wikipedia.org/wiki/Memory_allocation). *Memory leak* is an existence of objects that are not needed anymore according to the application logic, but still retain memory and cannot be collected because they are referenced from other live objects, due to a bug in application itself.

1. [memory structure in JVM](#Memory_structure_JVM)
2. [GarbageCollection](#GarbageCollection)
3. [MemoryLeaksJava](#MemoryLeaksJava)
4. Java memory profiling tools - Eclipse platform, the Memory Analyzer ,JHat, JProfiler,  JProbe, YourKit, AD4J or JRockit Mission Control.
5. [JMX AND MBean](#JMX_MBeans)

<http://java-espresso.blogspot.com/2011/04/memory-structure-in-jvm.html>

### Memory Structure in JVM

So, If we summarise

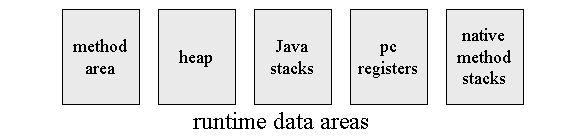
* Instance variables are stored in Heap
* Local variables are stored on stack
* Static variables are stored in Method Area
* Arrays are stored on heap

Error:

When no more memory is remaining, an OutOfMemoryError alert will be thrown and generate an exception like this:  
  
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space at  
MemoryLeakDemo.main(MemoryLeakDemo.java:14)

Tools:

Verbose log (part of servers) Or JDK Provided , you can use JConsole or [VisualVM](https://visualvm.dev.java.net/" \t "_blank) to attach to the process. Note that later JDK 6 versions include VisualVM.

JVM has various Runtime data areas some of which are created at the startup of JVM and are destroyed at the shutdown of JVM. Remaining data areas are per-thread.  
Figure below explains the **Runtime data areas that JVM has:**  
   
JVM has only two Runtime memory area types.:

1. Heap
2. Non-Heap
   1. PC register
   2. JVM stack

Let us explore every Memory Area one by one.

1. **JVM Heap**:
   1. shared across all the Threads
   2. Created on JVM start-up
   3. Memory is reclaimed using the Garbage Collection.
   4. Heap can be of fixed size or dynamically expandable. Users can expand the memory between permissible limits.
   5. If JVM Heap is configured for dynamic expansion and sufficient memory is not available then JVM will throw *OutOfMemoryError* Error
2. **PC(program Counter) Register**: PC is created per Thread. At one time, it executes code of one method for the associated thread.
   1. If that method is not native, It contains the address of the JVM instruction currently being executed.
   2. If the method currently being executed by the thread is native, the value of the Java virtual machine's pc register is undefined.
3. **JVM stack**: Each JVM thread has a private Java virtual machine stack, created at the same time as the thread
   1. It stores frames.[frames](http://java-espresso.blogspot.com/2011/04/memory-structure-in-jvm.html#1)
   2. Stacks are similar to conventional language(example,) stacks
   3. These store local variables, partial results and plays part in method invocation and return
   4. Size of JVM stack can be fix or expandable. If the size is fixed and for some operation memory required is more than the size then JVM throws *StackOverFlowError*. If Stack size is dynamically expandable and now if the required memory is more then the available then JVM will throw *OutOfMemoryError* Error
4. **Method Area**: It is shared among all Java virtual machine threads. It is created on virtual machine start-up.
   1. It stores per-class structures such as the runtime constant pool, field and method data, and the code for methods and constructors, including the special methods used in class and instance initialization and interface type initialization
   2. **It is logically part of the heap** but Specification doesn't restrict this.So it can be independent part also
   3. if JVM runs short of Method Area memory for a request then *OutOfMemoryError* Error will be thrown.
5. **Runtime Constant Pool**: Runtime constant Pool memory is allocated from the Method Area. The runtime constant pool for a class or interface is constructed when the class or interface is created by the Java virtual machine.
   1. It is a per-class or per-interface runtime representation of the constant\_pool table in a class file. Similar to symbol table in conventional languages .
   2. It stores different kinds of constants, ranging from numeric literals known at compile time to method and field references that must be resolved at run time.
6. **Native Method Stacks**: These stacks are provided by JVMs that support native methods (methods written in languages other than Java). They are created per thread.

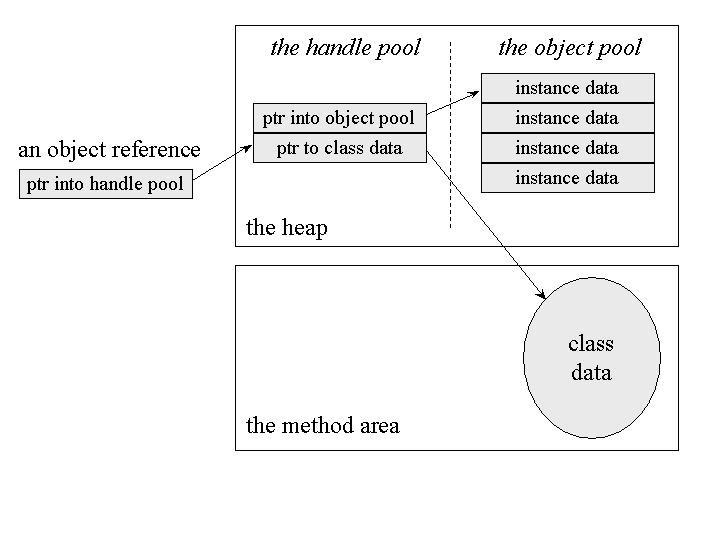
Discussing about JVM stacks I mentioned about ***Frames***. Lets discuss them now. ***Frames*** A frame is used to store data and partial results, as well as to perform dynamic linking , return values for methods, and dispatch exceptions.A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes, whether that completion is normal or abrupt. Each frame has its own array of local variables , its own operand stack (§3.6.2), and a reference to the runtime constant pool of the class of the current method. The memory of these frames is determined at compile time and allocated when the method is actually invoked at runtime. Frames have again three parts:

1. Local variables's stack
2. Operand stack
3. Frame Data

**So, If we summarize**

* **Instance variables are stored in Heap**
* **Local variables are stored on stack**
* **Static variables are stored in Method Area**
* **Arrays are stored on heap**

Now take a look at the diagram below to make things more clear:



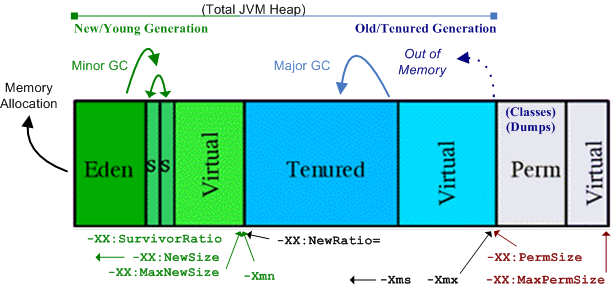
[original source of above image](http://www.cs.unc.edu/~stotts/144/lectures/lect32.pdf) Now finally references:

* [java papers blog](http://javapapers.com/core-java/java-jvm-memory-types/)
* [Lean,Mean JVM](http://www.artima.com/underthehood/leanmean.html)
* [Lean,Mean JVM](http://www.artima.com/underthehood/leanmean.html)
* [JVM Spec Overview](http://java.sun.com/docs/books/jvms/second_edition/html/Overview.doc.html)

<http://javadecodedquestions.blogspot.com/2012/12/java-memory-management.html>

### Garbage Collection:

Garbage collection (GC) is a form of automatic [memory management](http://en.wikipedia.org/wiki/Memory_management). The garbage collector, or just collector, attempts to reclaim [garbage](http://en.wikipedia.org/wiki/Garbage_(computer_science)), or memory occupied by [objects](http://en.wikipedia.org/wiki/Object_(computer_science)) that are no longer in use by the [program](http://en.wikipedia.org/wiki/Application_software). Garbage collection was invented by [John McCarthy](http://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist)) around 1959 to solve problems in [Lisp](http://en.wikipedia.org/wiki/Lisp_(programming_language)).

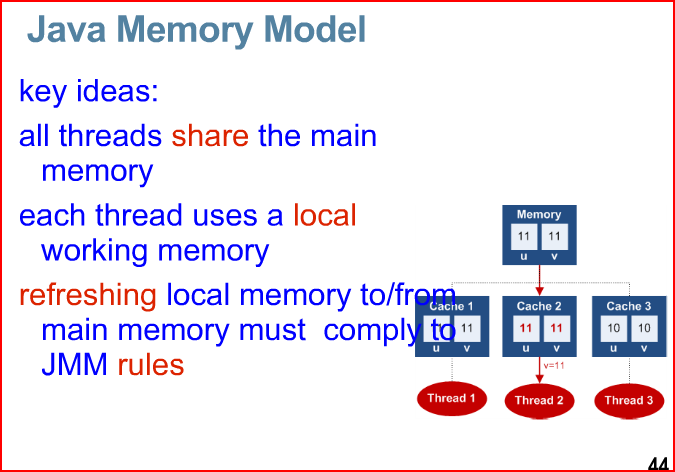
[](http://3.bp.blogspot.com/-KLUmCgw0yG0/UWn3DoDAkuI/AAAAAAAAAJ8/CroGLX4wlaM/s1600/java_memory.png)Garbage collection is often portrayed as the opposite of [manual memory management](http://en.wikipedia.org/wiki/Manual_memory_management), which requires the programmer to specify which objects to deallocate and return to the memory system. However, many systems use a combination of approaches, including other techniques such as [stack allocation](http://en.wikipedia.org/wiki/Stack-based_memory_allocation) and [region inference](http://en.wikipedia.org/wiki/Region_inference).

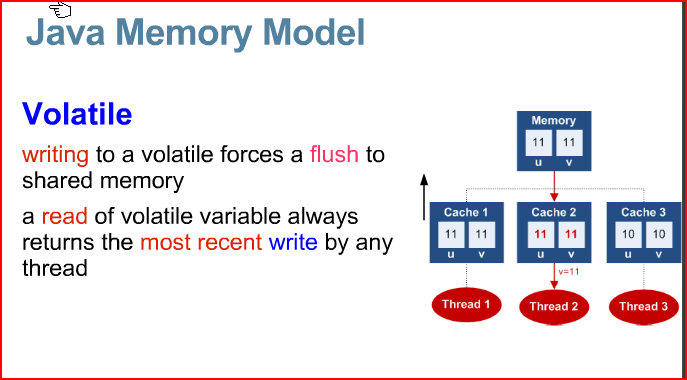
Resources other than memory, such as [network sockets](http://en.wikipedia.org/wiki/Network_socket), database [handles](http://en.wikipedia.org/wiki/Handle_(computing)), user interaction windows, and file and device descriptors, are not typically handled by garbage collection. Methods used to manage such resources, particularly [destructors](http://en.wikipedia.org/wiki/Destructor_(computer_science)), may suffice to manage memory as well, leaving no need for GC. Some GC systems allow such other resources to be associated with a region of memory that, when collected, causes the other resource to be reclaimed; this is called [finalization](http://en.wikipedia.org/wiki/Finalizer). Finalization may introduce complications limiting its usability, such as intolerable latency between disuse and reclaim of especially limited resources, or a lack of control over which thread performs the work of reclaiming.

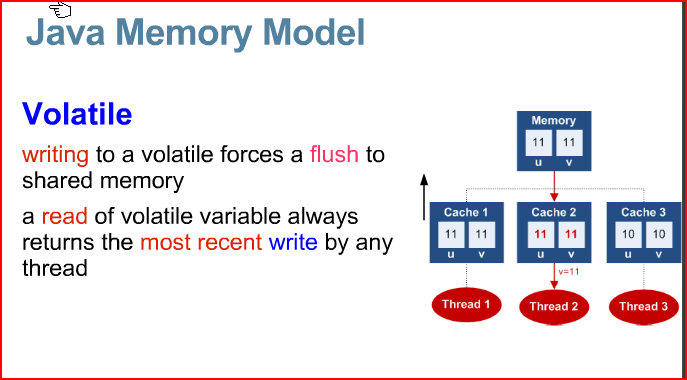
**Memory is divided in mainly 3 areas:**

* **Young\Eden Generation** : for newly created objects.
* **Tenured Generation** : for old objects which are survived after minor gc.
* **Perm Space** : for class definition, meta data and string pools.

<http://www.slideshare.net/caroljmcdonald/java-concurrency-memory-model-and-trends-4961797>







### How would you detect and minimize memory leaks in Java?

In Java, memory leaks are caused by poor program design where object references are long lived and the garbage collector is unable to reclaim those objects.

#### Detecting memory leaks:

* Use tools like JProbe, OptimizeIt etc to detect memory leaks.
* Use operating system process monitors like task manager on NT systems, ps, vmstat, iostat, netstat etc on UNIX systems.
* Write your own utility class with the help of totalMemory() and freeMemory() methods in the Java Runtime class. Place these calls in your code strategically for pre and post memory recording where you suspect to be causing memory leaks. An even better approach than a utility class is using dynamic proxies or Aspect Oriented Programming (AOP) for pre and post memory recording where you have the control of activating memory measurement only when needed.

#### Minimizing memory leaks:

In Java, typically memory leak occurs when an object of a longer lifecycle has a reference to objects of a short life cycle. This prevents the objects with short life cycle being garbage collected. The developer must remember to remove the references to the short-lived objects from the long-lived objects. Objects with the same life cycle do not cause any issues because the garbage collector is smart enough to deal with the circular references

* Design applications with an object’s life cycle in mind, instead of relying on the clever features of the JVM. Letting go of the object’s reference in one’s own class as soon as possible can mitigate memory problems. Example: myRef = null;
* Unreachable collection objects can magnify a memory leak problem. In Java it is easy to let go of an entire collection by setting the root of the collection to null. The garbage collector will reclaim all the objects (unless some objects are needed elsewhere).
* Use weak references if you are the only one using it. The WeakHashMap is a combination of HashMap and WeakReference. This class can be used for programming problems where you need to have a HashMap of information, but you would like that information to be garbage collected if you are the only one referencing it.
* Free native system resources like AWT frame, files, JNI etc when finished with them. Example: Frame, Dialog, and Graphics classes require that the method dispose() be called on them when they are no longer used, to free up the system resources they reserve.

<http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html>

**What is Heap space in Java?**

When a Java program started Java Virtual Machine gets some memory from Operating System. Java Virtual Machine or JVM uses this memory for all its need and part of this memory is call **java heap memory**. Heap in Java generally located at bottom of address space and move upwards. whenever we create object using new operator or by any another means object is allocated memory from Heap and When object dies or garbage collected ,memory goes back to **Heap space** in Java, to learn more about garbage collection see [how garbage collection works in Java](http://javarevisited.blogspot.com/2011/04/garbage-collection-in-java.html).

**How to increase size of Java Heap**

*Default size of Heap space  in Java is 128MB* on most of 32 bit Sun's [JVM](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html) but its highly varies from JVM to JVM  e.g. default maximum and start heap size for the 32-bit Solaris Operating System (SPARC Platform Edition) is -Xms=3670K and -Xmx=64M and Default values of heap size parameters on 64-bit systems have been increased up by approximately 30%. Also if you are using throughput garbage collector in Java 1.5 default maximum heap size of JVM would be Physical Memory/4 and  default initial heap size would be Physical Memory/16. Another way to find default heap size of JVM is to start an application with default heap parameters and monitor in using JConsole which is available on JDK 1.5 onwards, on VMSummary tab you will be able to see maximum heap size

By the way you can **increase size of java heap space** based on your application need and I always recommend this to avoid using default JVM heap values. if your application is large and lots of object created you can change size of heap space by using JVM options **-Xms and -Xmx**.  Xms denotes starting size of Heap while -Xmx denotes maximum size of Heap in Java. There is another parameter called -Xmn which denotes Size of new generation of **Java Heap Space**. Only thing is you can not change the size of Heap in Java dynamically, you can only provide Java Heap Size parameter while starting JVM. I have shared some more useful JVM options related to Java Heap space and Garbage collection on my post [10 JVM options Java programmer must know](http://javarevisited.blogspot.sg/2011/11/hotspot-jvm-options-java-examples.html), you may find useful.  
  
Read more: <http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html#ixzz35ECa5qeK>

**Java Heap dump**

**Java Heap dump**is a snapshot of Java Heap Memory at a particular time. This is very useful to analyze or troubleshoot any memory leak in Java or any Java.lang.OutOfMemoryError. There are tools available inside JDK which helps you to take heap dump and there are heap analyzer available tool which helps you to analyze java heap dump. You can use "jmap" command to get java heap dump, this will create heap dump file and then you can use *"jhat - Java Heap Analysis Tool"* to analyze those heap dumps.

Read more: <http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html#ixzz35ECyx6MQ>

Get heap Size programmatically in java

you can use either JConsole or Runtime.maxMemory(), Runtime.totalMemory(), Runtime.freeMemory() to query about Heap size programmatic in Java.   
  
Read more: <http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html#ixzz35EEHHiIY>

### Cause of OutOfMemoryError in PermGen space in Tomcat:

PermGen Space of heap is used to store classes and Meta data about [classes in Java](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html). When a class is loaded by a classloader it got stored in PermGen space, it gets unloaded only when the classloader which loaded this class got garbage collected. If any object retains reference of classloader than its not garbage collected and Perm Gen Space is not freed up. This causes memory leak in PermGen Space and eventually cause java.lang.OutOfMemoryError**: PermGen space.** Another important point is that when you deploy your web application a new Clasloader gets created and it loads the classes used by web application. So if Classloader doesn't get garbage collected when your web application stops you will have memory leak in tomcat.

### Solution of Tomcat: OutOfMemroyError:

1) Find the offending classes which are retaining reference of [Classloader](http://javarevisited.blogspot.com/2012/12/how-classloader-works-in-java.html) and preventing it from being garbage collected. Tomcat provides memory leak detection functionality *after tomcat 6* onwards which can help you to find when particular library, framework or class is causing memory leak in tomcat. Here are some of the commoncauses of **java.lang.OutOfMemoryError: PermGen space in tomcat server**:

1) **JDBC Drivers:**

JDBC drivers are most common cause of java.lang.OutOfMemoryError: PermGen space in tomcat if web app doesn't unregister during stop. One hack to get around this problem is that JDBC driver to be loaded by common class loader than application classloader and you can do this by transferring driver's jar into tomcat lib instead of bundling it on web application's war file.

2) **Logging framework:**

Similar solution can be applied to prevent logging libraries like Log4j causing java.lang.OutOfMemoryError: PermGen space in tomcat.

3) **Application Threads which have not stopped.**

Check your code carefully if you are leaving your [thread](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) unattended and running in while loop that can retain classloader's reference and cause java.lang.OutOfMemoryError: PermGen space in tomcat web server. Another common culprit is **ThreadLocal**, avoid using it until you need it absolutely, if do you make sure to set them null or free any object [ThreadLocal variables](http://javarevisited.blogspot.com/2012/05/how-to-use-threadlocal-in-java-benefits.html) are holding.

Another Simple Solution is to increase PermGen [heap size](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html) in catalina.bat or catalina.sh of tomcat server; this can give you some breathing space but eventually this will also return in *java.lang.OutOfMemoryError: PermGen space* after some time.

### Steps to increase PermGen Heap Space in Tomcat:

1) Go to Tomcat installation directory i.e. C:\Program Files\Apache Software Foundation\Apache Tomcat 7.0.14\bin in Windows and something similar in linux.

2) Add JAVA\_OPTS in your catalina.bat or Catalina.sh

In Windows:

set JAVA\_OPTS="-Xms1024m -Xmx10246m -XX:NewSize=256m -XX:MaxNewSize=356m -XX:PermSize=256m -XX:MaxPermSize=356m"

In linux:

export JAVA\_OPTS="-Xms1024m -Xmx10246m -XX:NewSize=256m -XX:MaxNewSize=356m -XX:PermSize=256m -XX:MaxPermSize=356m"

You can change the actual heap size and PermGen Space as per your requirement.

3) Restart Tomcat.

As I said earlier **increasing PermGen space** can prevent **java.lang.OutOfMemoryError: PermGen** in tomcat only for some time and it will eventually occur based on how many times you redeploy your web application, its best to find the offending class which is causing memory leak in tomcat and fix it.

**Related Java Tutorials**

Read more: <http://javarevisited.blogspot.com/2012/01/tomcat-javalangoutofmemoryerror-permgen.html#ixzz36pmYmzXB>

HOW TO FIND MEMORY LEAKS In java

<http://stackoverflow.com/questions/40119/how-to-find-a-java-memory-leak>

general approach:

* 1. First get the memory dump by using jconsole under the JDK bin folder
  2. The heap dump file will have extension such as .hprof or .phd

I use following approach to finding memory leaks in Java. I've used jProfiler with great success, but I believe that any specialized tool with graphing capabilities (diffs are easier to analyze in graphical form) will work.

1. Start the application and wait until it get to "stable" state, when all the initialization is complete and the application is idle.
2. Run the operation suspected of producing a memory leak several times to allow any cache, DB-related initialization to take place.
3. Run GC and take memory snapshot.
4. Run the operation again. Depending on the complexity of operation and sizes of data that is processed operation may need to be run several to many times.
5. Run GC and take memory snapshot.
6. Run a diff for 2 snapshots and analyze it.

Basically analysis should start from greatest positive diff by, say, object types and find what causes those extra objects to stick in memory.

For web applications that process requests in several threads analysis gets more complicated, but nevertheless general approach still applies.

I did quite a number of projects specifically aimed at reducing memory footprint of the applications and this general approach with some application specific tweaks and trick always worked well.

invoke GC with a click of a button

Questioner here, I got to say getting a tool that does not take 5 minutes to answer any click makes it a lot easier to find potential memory leaks.

Since people are suggesting several tools ( I only tried visual wm since I got that in the JDK and JProbe trial ) I though I should suggest a free / open source tool built on the Eclipse platform, the Memory Analyzer (sometimes referenced as the SAP memory analyzer) available on <http://www.eclipse.org/mat/> .

What is really cool about this tool is that it indexed the heap dump when I first opened it which allowed it to show data like retained heap without waiting 5 minutes for each object (pretty much all operations was tons faster than the other tools I tried).

When you open the dump the first screen shows you a pie chart with the biggest objects (counting retained heap) and one can quickly navigate down to the objects that are to big for comfort. It also has a Find likely leak suspects which I reccon can come in handly, but since the navigation was enough for me I did not really get into it.

**About JMX and MBeans**

<http://www.javalobby.org/java/forums/t49130.html>

**JMX** stands for Java Management Extensions, and is a facility to allow for remote clients to connect to a JVM, and manage/monitor running applications in that JVM. This management is typically done through MBeans. MBeans are the heart of the JMX specification. For an application developer, MBeans are the controllable end-points of your application where remote clients can observe application activity as well as control the inner workings. Like any good application design, determining how your mbeans should be structured and designed is not an exact science, but they also aren't nearly as scary as most people think.

Difference between JIT and JVM

Main difference between JIT and JVM is that, JIT is part of JVM itself and used to improve performance of JVM.

JIT stands for Just In time compilation and JVM stands for Java Virtual Machine.

JVM is a virtual machine used in Java programming platform to execute or run Java programs. Main advantage of JVM is that, it makes Java platform independent by executing byte codes.

Java source code is compiled into class files, which contains byte code. These byte codes are then executed by JVM. Now here comes JIT. Since execution of byte code is slower than execution of machine language code, because JVM first needs to translate byte code into machine language code.

JIT helps JVM here by compiling currently executing byte code into machine language. JIT also offers caching of compiled code which result in improved performance of JVM. by the way difference between JVM and JIT is also a good Java interview question to ask. Well, this is just a simple explanation, JIT is lot more complex than this. There are sophisticated algorithm which helps JIT to pick most executed code for compiling into machine code.