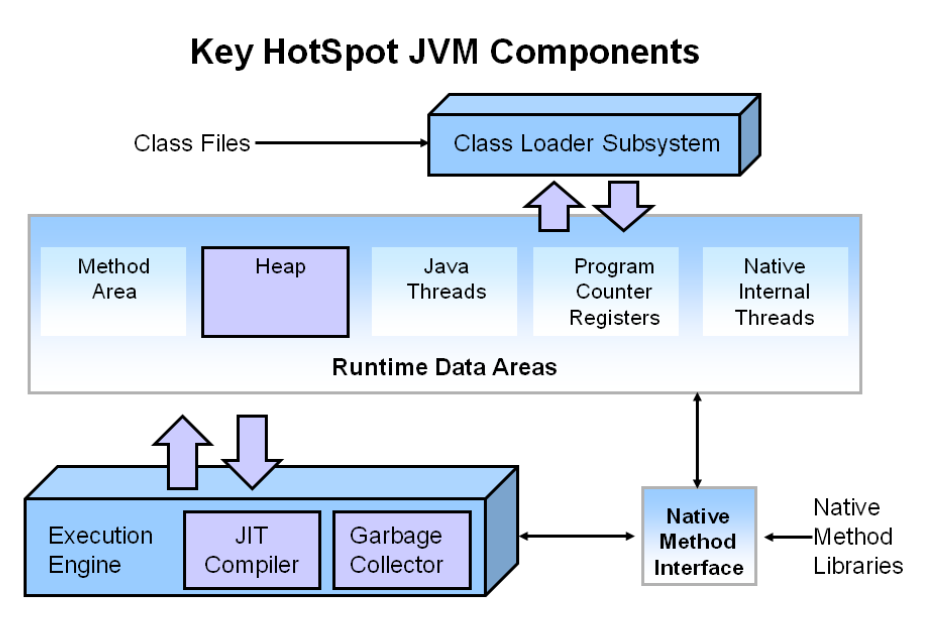
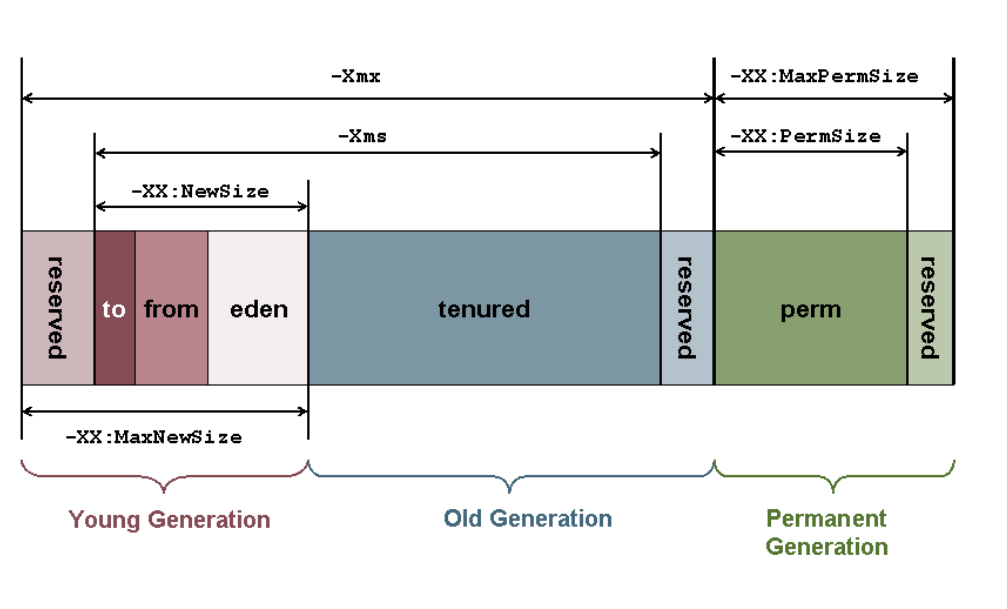
**Memory Architecture Of JVM(Runtime Data Areas)**

During execution of program JVM allocates memory. Some Runtime data areas are specific to thread only.Following is list of different Runtime Data Areas:-  
  
Image Reference:-Oracle

1. [Heap](https://hackthejava.wordpress.com/2015/01/09/memory-architecture-by-jvmruntime-data-areas/)
2. [Stack(Known as Java Stack)](https://hackthejava.wordpress.com/2015/01/09/memory-architecture-by-jvmruntime-data-areas/)
3. [pc register](https://hackthejava.wordpress.com/2015/01/09/memory-architecture-by-jvmruntime-data-areas/)
4. [Method Area](https://hackthejava.wordpress.com/2015/01/09/memory-architecture-by-jvmruntime-data-areas/)
5. [Native Stack](https://hackthejava.wordpress.com/2015/01/09/memory-architecture-by-jvmruntime-data-areas/)

* **HEAP:-**

Heap Memory is created by JVM in start of program and used for storing objects.Heap Memory can be accessed by any thread is further divided into three generations Young Generation, Old & PermGen(Permanent Generation). When object is created then it first go to Young generation(especially Eden space) when objects get old then it moves to Old/tenured Generation. In PermGen space all **static** & **instance variables** name-value pairs(name-references for object) are stored. Below is image showing heap structure of java.



You can manually increase heap size by some JVM parameters as shown in images. Suppose we have a simple class HackTheJava then increasing its memory by following parameters:-

|  |  |
| --- | --- |
| 1 | java -Xms=1M -XmX=2M HackTheJava |

Similarly increase PermGen or NewSize.

* **Stack:-**

Stack is generated with each thread created by program. It is associated by thread. Each Thread has its own stack. All local variables & function calls are stored in stack. It’s life depends upon Thread’s life as thread will be alive it will also and vice-versa. It can also be increased by manually:-

|  |  |
| --- | --- |
| 1 | java -Xss=512M HackTheJava |

It throws StackOverFlow error when stack get full.

* **PC Register**

It is also associated by its thread. It basically is a address of current instruction is being executed. Since each thread some sets of method which is going to be executed depends upon PC Register. It has some value for each instruction and undefined for native methods. It is usually for keep tracking of instructions.

* **Method Area**

It is memory which is shared among all Threads like Heap. It is created on Java Virtual Machine startup. It contains the code actually a compiled code, methods and its data and fields. **Runtime constant pool** is also a part of Method Area. Runtime Constant pool is per class representation of constant Table. It contains all literals defined at compiled time and references which is going to be solved at runtime. Memory for it is by default allotted by JVM and can be increased if needed.

* **Native Method Stack**

Native methods are those which are written in languages other than java. JVM implementations cannot load native methods and can’t rely on conventional stacks . It is also associated with each thread. In short it same as stack but it is used for native methods.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class HackTheJava  {          int instance\_variable=0;          static int static\_variable=0;          void func(int a)          {              int local\_variable=0;              return ;          }  } |

Hope You like this article. Thanks