# Thread dumps

A JVM Thread Dump is a snapshot taken at a given time that provides you with a complete listing of all created Java Threads.

Each individual Java Thread found gives you information such as

– Thread name: often used by middleware vendors to identify the Thread Id along with its associated Thread Pool name and state (running, stuck etc.)

– Thread type & priority ex: daemon prio=3 *\*\* middleware softwares typically create their Threads as daemon meaning their Threads are running in background; providing services to its user e.g. your Java EE application \*\**

– Java Thread ID ex: tid=0x000000011e52a800 *\*\* this is the Java Thread Id obtained via java.lang.Thread.getId () and usually implemented as an auto-incrementing long 1.n\*\**

 – Native Thread ID ex: nid=0x251c*\*\* Crucial information as this native Thread Id allows you to correlate for example which Threads from an OS perspective are using the most CPU within your JVM etc. \*\**

– Java Thread State and detail ex: waiting for monitor entry [0xfffffffea5afb000] java.lang.Thread.State: BLOCKED (on object monitor)  
*\*\* Allows to quickly learn about Thread state and its potential current blocking condition \*\**

 – Java Thread Stack Trace; this is by far the most important data that you will find from the Thread Dump. This is also where you will spent most of your analysis time since the Java Stack Trace provides you with 90% of the information that you need in order to pinpoint root cause of many problem pattern types as you will learn later in the training sessions

– Java Heap breakdown; starting with HotSpot VM 1.6, you will also find at the bottom of the Thread Dump snapshot a breakdown of the HotSpot memory spaces utilization such as your Java Heap (YoungGen, OldGen) & PermGen space. This is quite useful when excessive GC is suspected as a possible root cause so you can do out-of-the-box correlation with Thread data / patterns found.