JAVA 12 Multithreading

Concepts, Kernel, Threads, Synchronization, Concurrent Collections, Debug, Akka, Server Design

[Sample: snippets] This course examines how to use Java 12 to build sophisticated multithreaded architectures. When designed correctly, multithreading can substantially increase application performance and responsiveness to distributed clients and end-users. The kernel provides a number of opaque objects that are the foundation for multithreading – based on this are constructs for the process, thread & various synchronization objects – reentrant locks, event, semaphore, waitable timer and more – each of which targets different needs. Thread activity, lifetimes, inter-thread comms and memory usage must be co-ordinated. Various higher-level design patterns may be used to route workitems in multithreaded servers. Tools may be developed to determine which thread is blocked waiting on which resource, and the state/owner of each resource. A delegate-based configurable pipeline + a cache are often appropriate. The optimal server architecture is one active thread per processor core. A server must efficiently multiplex many I/O requests over a few threads – which is precisely the goal of threadpools in Java 12. This course supplies attendees with a clear understanding of the concepts underlying multi-threading, together with experience of their use in Java.

Target Audience

System architects and experienced developers who need to gain an in-depth understanding of Java multithreading.

Prerequisites

Attendees must have some experience of systems-level programming.

Attendance at our *Java 12 Runtime Programming* course or equivalent experience needed.

Contents of One-Day Training Course

Multithreading Concepts

Scheduling vs. synchronization
Parallelism and concurrency
Compute-bound and I/O bound apps
Race conditions, deadlock, starvation, priority inversion

OS Foundations

Processes & threading in various kernel
Usage counting
Sharing handles among processes

Threads in Java

Java's threading architecture & lifecycle
The Runnable interface
ThreadLocal – each thread has its own copy of the variable / ThreadGroup

Creating Java Threads

OS threads and Java threads
Creating a new thread: java.lang.Thread
Implementing a Runnable
Extending the Thread class
Thread priority and processor affinity

Synchronization

Locks, mutexes (reentrant locks), semaphore, events and timers
Waiting (once, many), idea of spin wait
The "Protect data, not code" principle
Atomics with java.util.concurrent.atomic
A synchronized block

Java Collections And Threads

When do we need to protect collections?
Selecting and using thread-safe collections
java.util.concurrent.locks
java.util.concurrent for concurrency
java.util.concurrent.atomic:*
supports lock-free thread-safe programming on single variables

Thread Pool

The thread pool is a runtime-managed pools of threads for processing I/O, work-items and timer handlers

Debugging with Threads

Querying information about running processes/threads and their attributes
Thread tracing/debugging in tooling

Resource Management

Creating a custom resource browser, to display which thread is waiting on which synchronization resource

Design Issues

Single writer/multiple readers, once-off initialization, Dining Philosopher
Converting legacy code to multithreading

Multithreaded Architectures

Pipeline, Producer-Consumer, Work-Crew and Master-Slave Models
Create threads on demand vs. elastic pool

Akka – The Actor Model

Moving beyond foundational thread constructs, explore higher level frameworks
Akka is based on the actor model and is highly suited to large scale projects
Working without locks – how?

Multithreaded Project

A complete multithreaded embedded HTTP web server that uses a thread pool to efficiently manage very many requests