Master Java Concurrency and Multi-Threading

# Source

1. Java
2. Java Concurrency & Multithreading in Practice by Tatiana Fesenko

# Thread Fundamentals: Creation and Coordination

A thread is to execute a particular task. This task can be implemented by:

1. Runnable object
2. RecursiveTask.compute() : T 🡺 when you’re interested in the result
3. RecursiveAction.compute() : void 🡺 when you’re NOT interested in the result
4. ForkJoinTask 🡺 ‘divide & conquer’ a task, as in: binary search, merge sort, closest pair pof points, Karatsuba algorithm for integer multiplication

To execute a task in a separate thread, you can:

1. Create a subclass of thread and override the run() method with the intended task. NOTE: if you override start() method, make sure you call super.start() at the end.
2. Instantiate a thread object with a Runnable object passed as an argument

Best practice: use a thread pool instead for thread creation and management:

1. ForkJoinPool.commonPool().invokeAll(Collection<Callable<T>>) : List<Future<T>>

* Fork() 🡺 delegate a subtask to another thread
* Join() 🡺 join results by waiting for other threads

1. A

# Performance Optimisation: Latency and Throughput

# Data Sharing between Threads

# Concurrency Challenges and Solutions

# Advanced Locking

# Inter-thread Communication

# Lock-free Algorithms, Data Structures and Techniques

# Threading Models for High Performance IO

# Virtual Threads and High Performance IO

# Structured Concurrency

Shutdown policies 🡺 heart of StructuredTaskScope