Difference between CountDownLatch and CyclicBarrier in Java

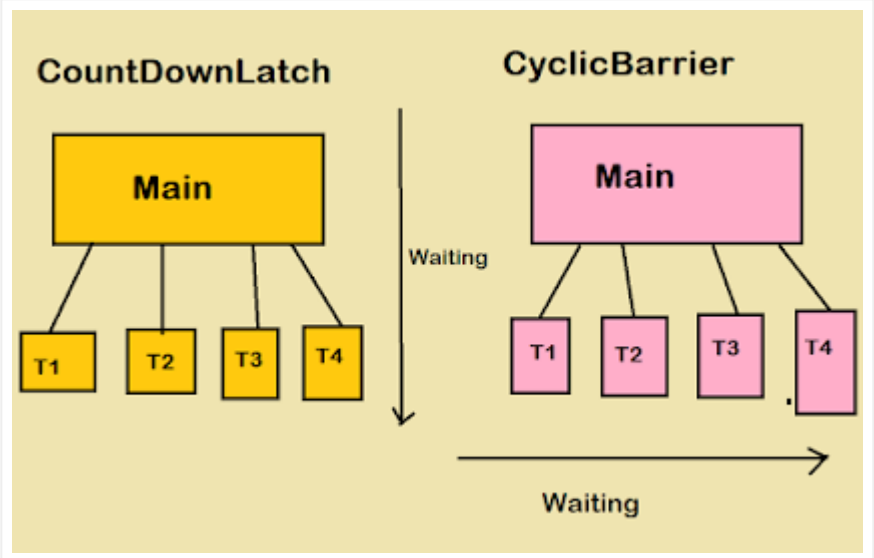
Both CyclicBarrier and `CountDownLatch` are used to implement a scenario where one Thread waits for one or more Thread to complete their job before starting processing but there is one difference between `CountDownLatch` and `CyclicBarrier` in Java which separates them apart and that is, you can not reuse the same `CountDownLatch` instance once count reaches to zero and latch is open, on the other hand, `CyclicBarrier` can be reused by resetting Barrier, Once the barrier is broken.

A useful property of a `CountDownLatch` is that it doesn't require that threads calling `countDown` wait for the count to reach zero before proceeding, it simply prevents any thread from proceeding past an await until all threads could pass.

A `CyclicBarrier` supports an optional `Runnable` command that is run once per barrier point, after the last thread in the party arrives, but before any threads are released. This *barrier action* is useful for updating the shared state before any of the parties continue.

The `CyclicBarrier` uses a fast-fail all-or-none breakage model for failed synchronization attempts: If a thread leaves a barrier point prematurely because of interruption, failure, or timeout, all other threads, even those that have not yet resumed from a previous await(), will also leave abnormally via `BrokenBarrierException` (or `InterruptedException` if they too were interrupted at about the same time).

This diagram also nicely explains the **difference between CountDownLatch and CyclicBarrier** in Java concurrency:

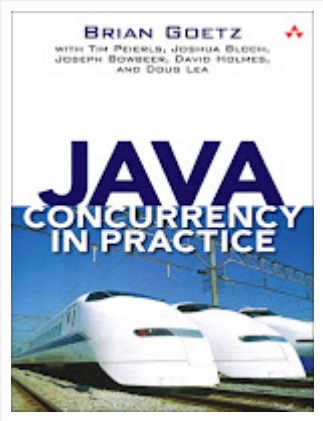


That's all about the **difference between CountDownLatch and CyclicBarrier in Java**.

As I said, the key difference is that you can reuse CyclicBarrier but CountDownLatch cannot be reused once count down reaches zero.

That's why CyclicBarrier is often used when a task is repeated while CountDownLatch is used for a one-time task like loading the initial cache before start accepting client connections.

Btw, If you are serious about improving your understanding of Java concurrency knowledge and skills, I strongly suggest you should read the **Java Concurrency in Practice** book by Brian Goetz, Joshua Bloch, Doug Lea, and team, one of the best resources for Java developers.



If you find trouble understanding the book and need someone to explain to you those concepts with more examples, you can also join the **Java Concurrency in Practice Bundle** course by Heinz Kabutz which is based upon this book and makes it easy to understand those tricky concurrency concepts. It's like someone explaining your Java Concurrency in Practice book live.

Even if you have read the book, going through this course will help you to understand Java Concurrency better and help you to write robust concurrent code with fewer bugs. Sorry, it's almost impossible to write concurrent code without subtle bugs, at least in the first iteration.