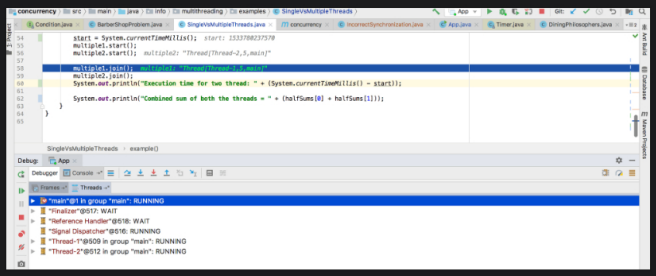
Introduction

This lesson details the reasons why threads exist and what benefit do they provide. We also discuss the problems that come with threads.

What good is concurrency?

Understanding of how threading works and knowledge of concurrent programming principles exhibit maturity and technical depth of a candidate and can be an important differentiator in landing a higher leveling offer at a company. First, we have to understand why threading models exist and what good do they provide?

Threads like most computer science concepts aren't physical objects. The closest tangible manifestation of threads can be seen in a debugger. The screen-shot below, shows the threads of our program suspended in the debugger.



Suspended threads in a debugger

The simplest example to think of a concurrent system is a single-processor machine running your favorite IDE. Say you edit one of your code files and click save, that clicking of the button will initiate a workflow which will cause bytes to be written out to the underlying physical disk. However, IO is an expensive operation, and the CPU will be idle while bytes are being written out to the disk.

Whilst IO takes place, the idle CPU could work on something useful and here is where threads come in - the IO thread is ***switched out*** and the UI thread gets scheduled on the CPU so that if you click elsewhere on the screen, your IDE is still responsive and does not appear hung or frozen.

Threads can give the illusion of multitasking even though at any given point in time the CPU is executing only one thread. Each thread gets a slice of time on the CPU and then gets switched out either because it initiates a task which requires waiting and not utilizing the CPU or it completes its time slot on the CPU. There are much more nuances and intricacies on how thread scheduling works but what we just described, forms the basis of it.

With advances in hardware technology, it is now common to have multi-core machines. Applications can take advantage of these architectures and have a dedicated CPU run each thread.

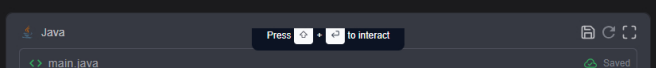
Benefits of Threads

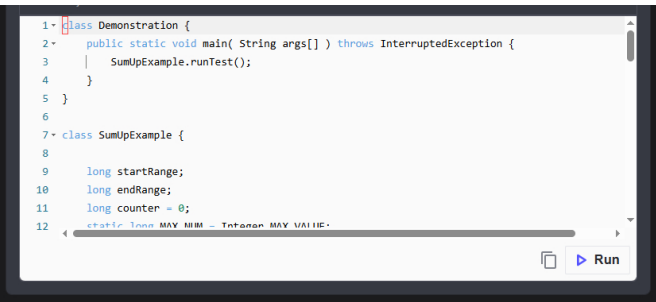
1. Higher throughput, though in some pathetic scenarios it is possible to have the overhead of context switching among threads steal away any throughput gains and result in worse performance than a single-threaded scenario. However such cases are unlikely and an exception, rather than the norm.
2. Responsive applications that give the illusion of multi-tasking.
3. Efficient utilization of resources. Note that thread creation is light-weight in comparison to spawning a brand new process. Web servers that use threads instead of creating new processes when fielding web requests, consume far fewer resources.

All other benefits of multi-threading are extensions of or indirect benefits of the above.

Performance Gains via Multi-Threading

As a concrete example, consider the example code below. The task is to ***compute the sum of all the integers from 0 to*** `Integer.MAX_VALUE`. In the first scenario, we have a single thread doing the summation while in the second scenario we split the range into two parts and have one thread sum for each range. In the end, we add the two half sums to get the combined sum. We measure the time taken for each scenario and print it.





In my run, I see the two threads scenario run within **652 milliseconds** whereas the single thread scenario runs in **886 milliseconds**. You may observe different numbers but the time taken by two threads would always be less than the time taken by a single thread. The performance gains can be many folds depending on the availability of multiple CPUs and the nature of the problem being solved. However, there will always be problems that don't yield well to a multi-threaded approach and may very well be solved efficiently using a single thread.

Problems with Threads

However, as it is said, there's no free lunch in life. The premium for using threads manifests in the following forms:

1. ***Usually very hard to find bugs***, some that may only rear head in production environments
2. ***Higher cost of code maintenance*** since the code inherently becomes harder to reason about
3. ***Increased utilization of system resources***. Creation of each thread consumes additional memory, CPU cycles for book-keeping and waste of time in context switches.
4. ***Programs may experience slowdown*** as coordination amongst threads comes at a price. Acquiring and releasing locks adds to program execution time. Threads fighting over acquiring locks cause lock contention.

With this backdrop lets delve into more details of concurrent programming about which you are likely to be quizzed in an interview.

Completed Next →
Program vs Process vs Thread