1.

What is Concurrency and Parallelism

What is Process, Program and eventually threads

Software threads and Hardware threads

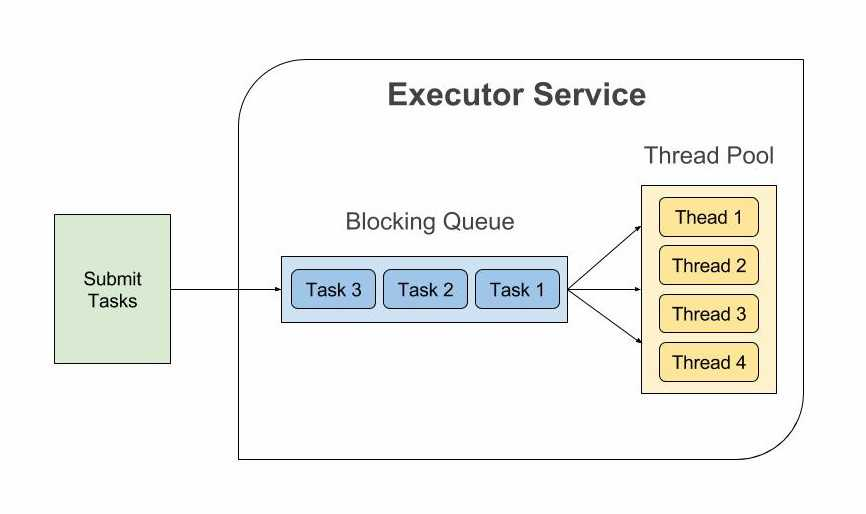
Problem related to concurrency: Memory consistency errors -> solved by acquiring locks -> Leads to starvation and deadlock

2.

Thread creation by extending Thread class and by implementing Runnable interface

Sleep function and join function

3.

ExecutorService Demo  
  


4.

Callable and Future Demo

5.   
Problem of Race Condition:

1. Retrieve the current value of count
2. Increment the retrieved value by 1
3. Store the incremented value back in count

Now let’s assume that two threads - ThreadA and ThreadB, execute these operations in the following order -

ThreadA : Retrieve count, initial value = 0

ThreadB : Retrieve count, initial value = 0

ThreadA : Increment retrieved value, result = 1

ThreadB : Increment retrieved value, result = 1

ThreadA : Store the incremented value, count is now 1

ThreadB : Store the incremented value, count is now 1

Both the threads try to increment the count by one, but the final result is 1 instead of 2 because the operations executed by the threads interleave with each other. In the above case, the update done by ThreadA is lost.

The above order of execution is just one possibility. There can be many such orders in which these operations can execute making the program’s output inconsistent.

The section of the code where a shared variable is accessed is called Critical Section.

When multiple threads try to read and write a shared variable concurrently, and these read and write operations overlap in execution, then the final outcome depends on the order in which the reads and writes take place, which is unpredictable. This phenomenon is called Race condition.

This can be resolved if we ensure that: Only one thread can read and write a shared variable at a time. When one thread is accessing a shared variable, other threads should wait until the first thread is done. This guarantees that the access to a shared variable is Atomic, and multiple threads do not interfere.

Synchronized

Memory Consistency error

Volatile

6.

Java’s concurrency api defines several classes in java.util.concurrent.atomic package that support Atomic operations on single variables.

The AtomicInteger.incrementAndGet() method is atomic, so you can safely call it from several threads simultaneously and be sure that the access to the count variable will be synchronized.

Following are some other atomic classes defined inside java.util.concurrent.atomic package. -

AtomicBoolean

AtomicLong

AtomicReference

You should use these Atomic classes instead of synchronized keyword and locks whenever possible because they are faster, easier to use, readable and scalable.