1. **Explain about Thread Synchronisation?**

Thread synchronization is a mechanism to ensure that multiple threads do not interfere with each other while accessing shared resources. It prevents race conditions, maintains data consistency, and allows proper execution order.

**Example: Synchronizing a shared counter**

Consider a scenario where multiple threads increment a shared counter. Without synchronization, inconsistent results may occur due to race conditions.

class Counter {

private int count = 0;

public synchronized void increment() {

count++; // Only one thread can execute this at a time

}

public synchronized int getCount() {

return count;

}

}

class Worker extends Thread {

private Counter counter;

public Worker(Counter counter) {

this.counter = counter;

}

@Override

public void run() {

for (int i = 0; i < 1000; i++) {

counter.increment();

}

}

}

public class ThreadSyncExample {

public static void main(String[] args) throws InterruptedException {

Counter counter = new Counter();

Thread t1 = new Worker(counter);

Thread t2 = new Worker(counter);

t1.start();

t2.start();

t1.join();

t2.join();

System.out.println("Final counter value: " + counter.getCount());

}

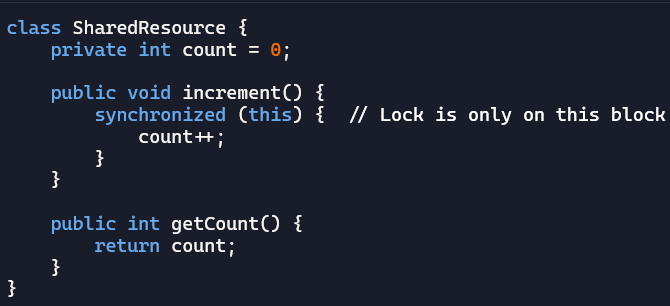
}

**Explanation:**

* synchronized ensures that only one thread modifies count at a time.
* join() ensures that both threads complete execution before printing the final count.
* Without synchronization, unpredictable values may appear due to concurrent access.

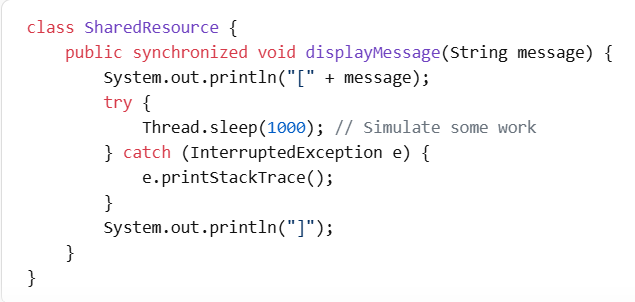
1. **Explain Synchronization block?**

Java programming language provides a very handy way of creating threads and [synchronizing](https://www.tutorialspoint.com/java/java_thread_synchronization.htm) their task by using synchronized blocks. You keep shared resources within this block.



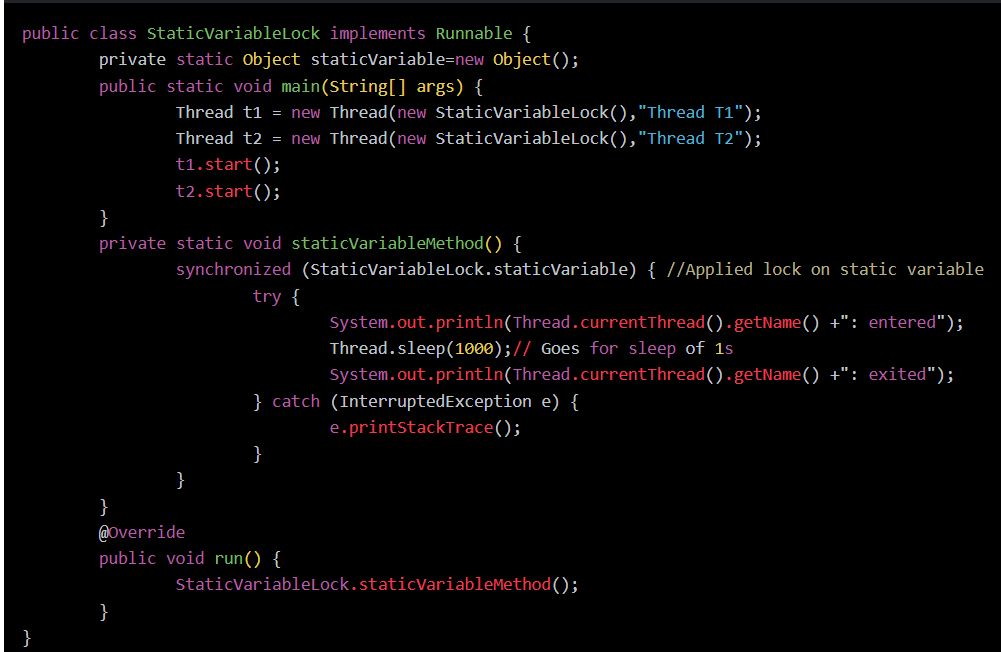
1. **Explain Synchronised Method?**

In Java, method synchronization is achieved by adding the **synchronized k**eyword. Then the entire method is treated as a critical section. Ensuring that only one thread can execute that method at any given time. This approach locks the entire method, preventing other threads from accessing it until the first thread finishes its execution.



1. **Explain Class level synchronisation?**

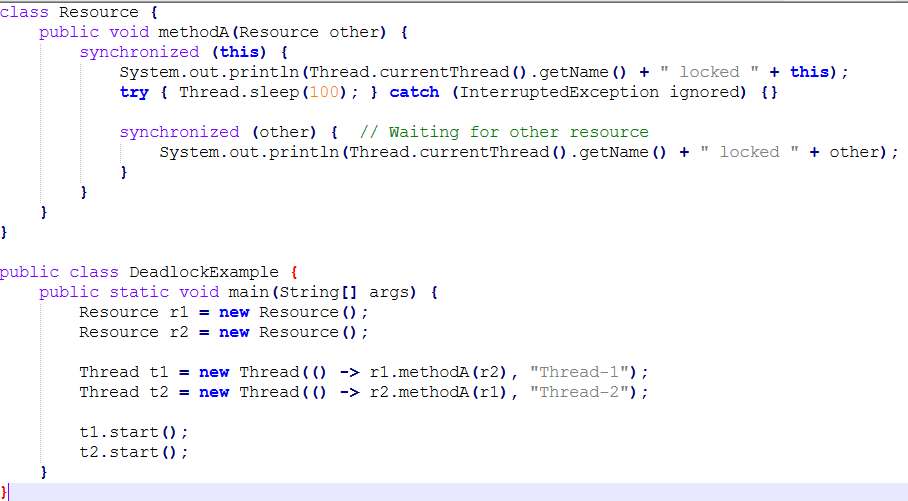
Synchronization can be applied at the class level, meaning all threads accessing the class share the same lock.

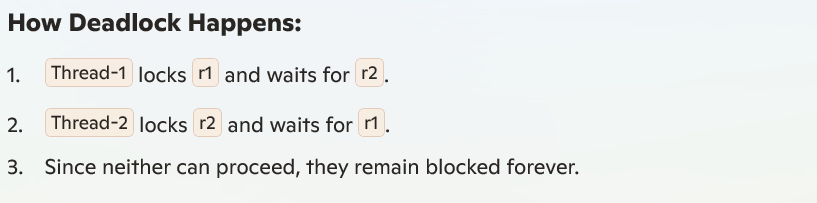




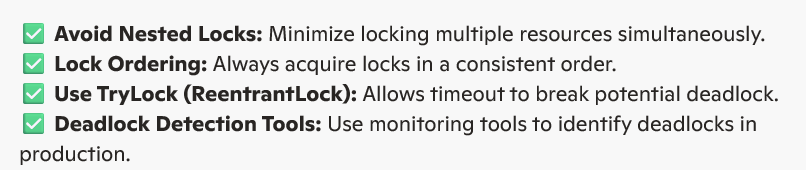
1. **Is deadlock occurs during thread synchronisation?**

Yes, deadlocks can occur during thread synchronization when multiple threads hold locks on shared resources and wait indefinitely for each other to release their locks. This results in a situation where no thread can proceed, leading to a system freeze or performance degradation.



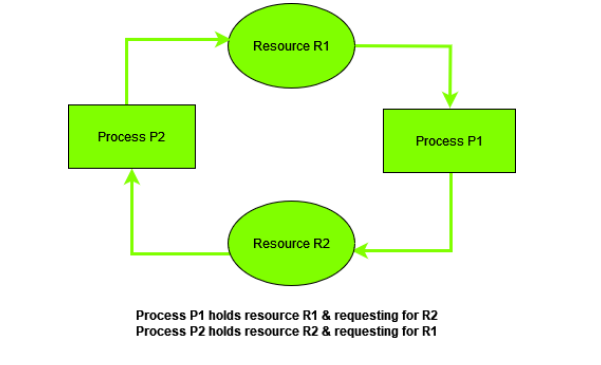


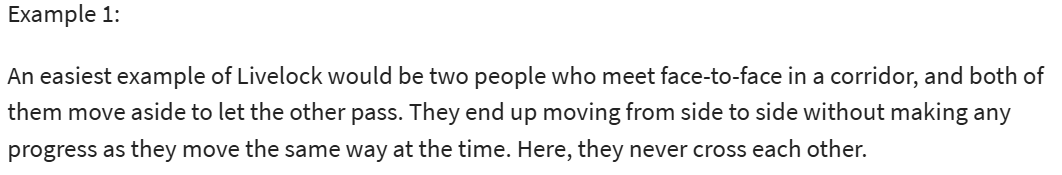
1. **How to avoid deadlock in thread synchronisation?**



1. **Explain about Live Lock ?**

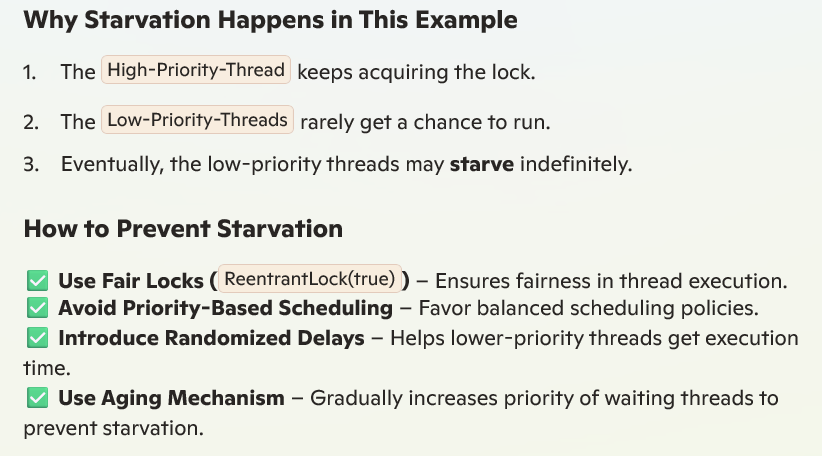
**Livelock** occurs when two or more processes continually repeat the same interaction in response to changes in the other processes without doing any useful work. These processes are not in the waiting state, and they are running concurrently. This is different from a deadlock because in a deadlock all processes are in the waiting state.





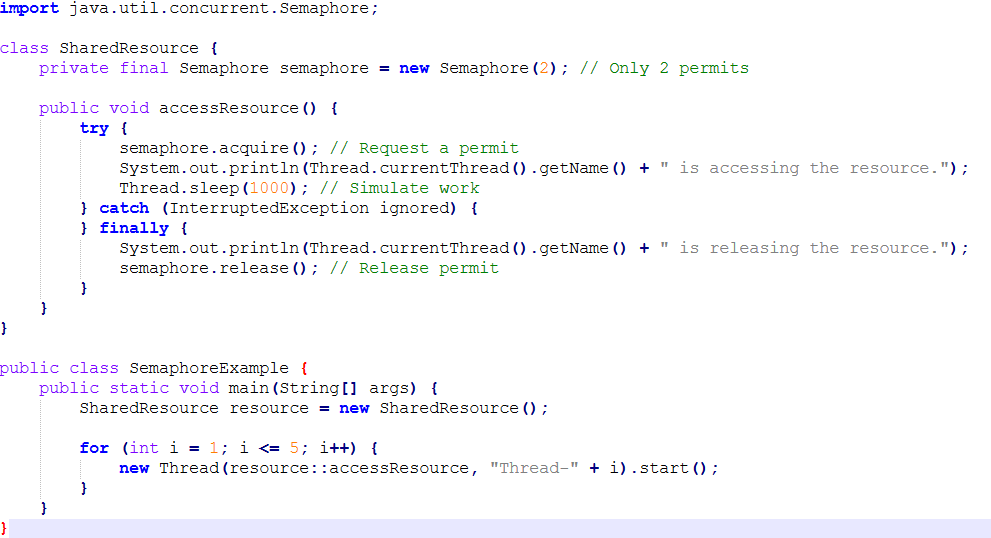
1. **What is Starvation?**

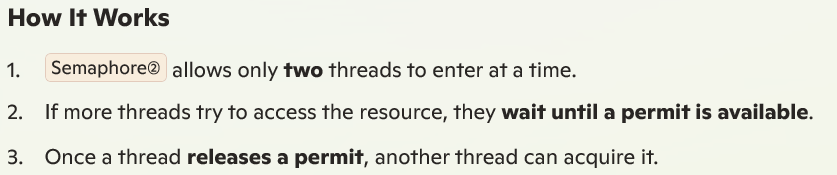
Starvation is a situation where all the low priority processes got blocked, and the high priority processes proceed. In any system, requests for high/low priority resources keep on happening dynamically. Thereby, some policy is require to decide who gets support when.



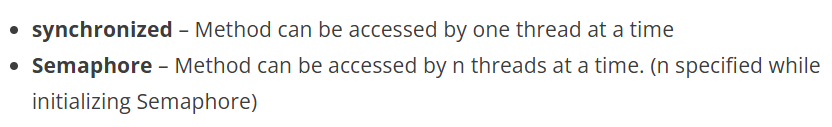
1. **Explain Semaphore in threads?**

A **Semaphore** is a synchronization mechanism that controls access to a shared resource by multiple threads using **permits**. It helps **limit concurrent access** and prevents race conditions.





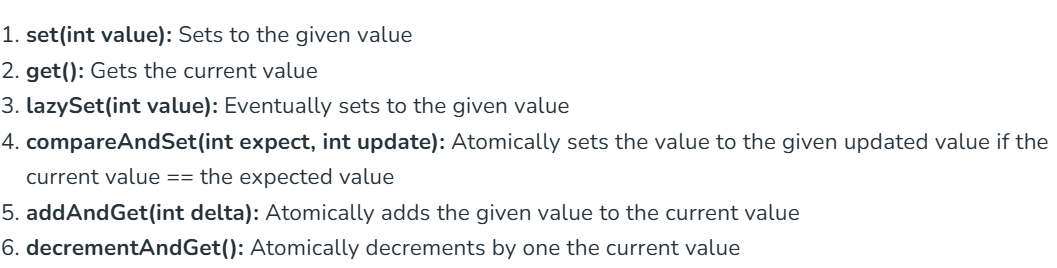
1. **Difference between Semaphore and synchronised in threads?**

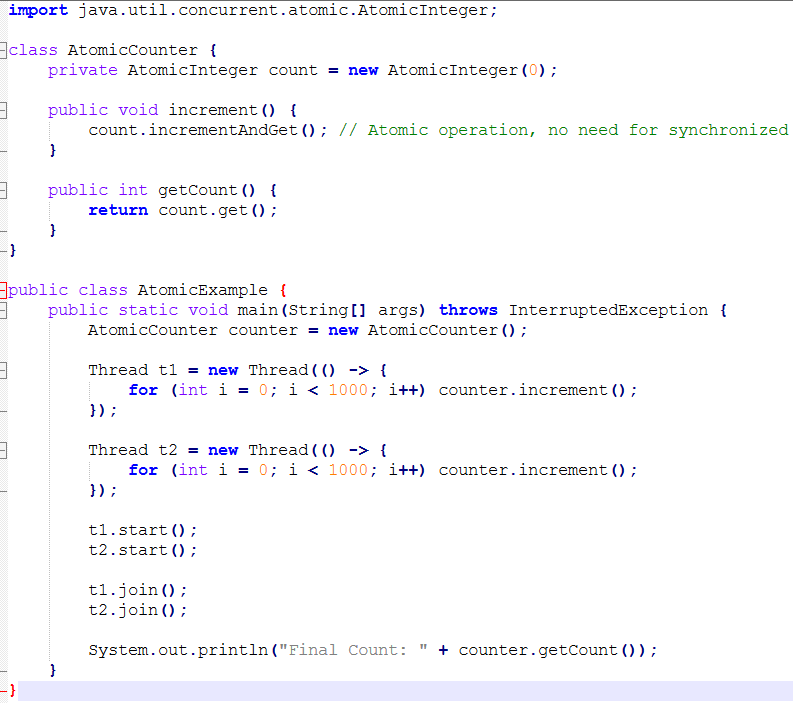


1. **Explain Atomic Variables in threads?**

Regular variables like int or long can be modified by multiple threads simultaneously, causing inconsistencies. Atomic variables ensure **thread-safe** updates without using synchronized.

Java provides the java.util.concurrent.atomic package, which offers **atomic variables** that help prevent race conditions **without using explicit synchronization**.

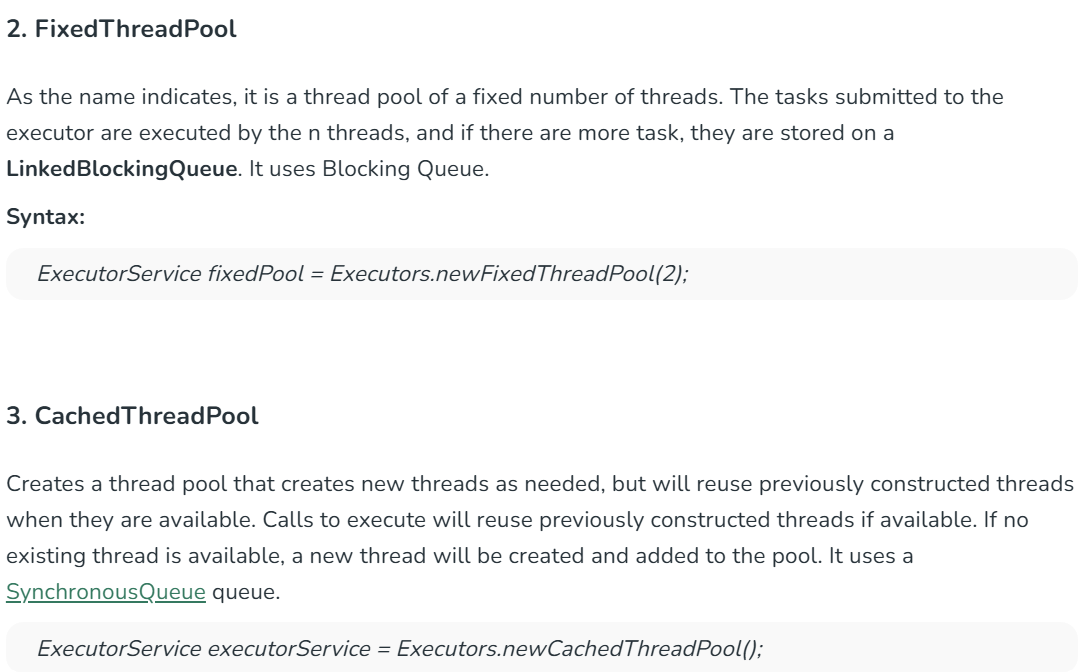


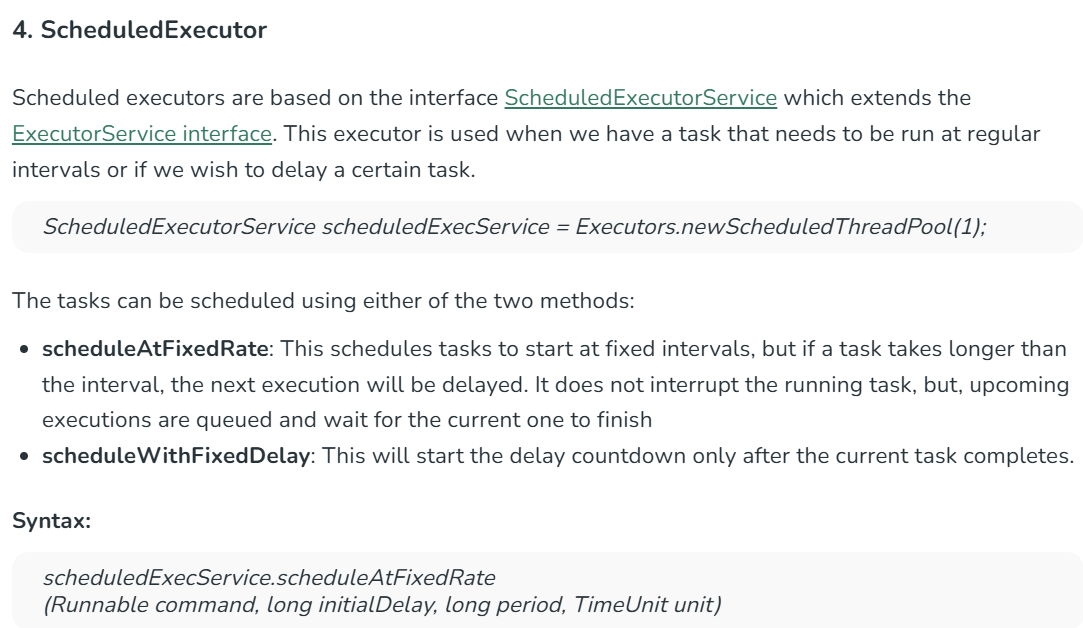


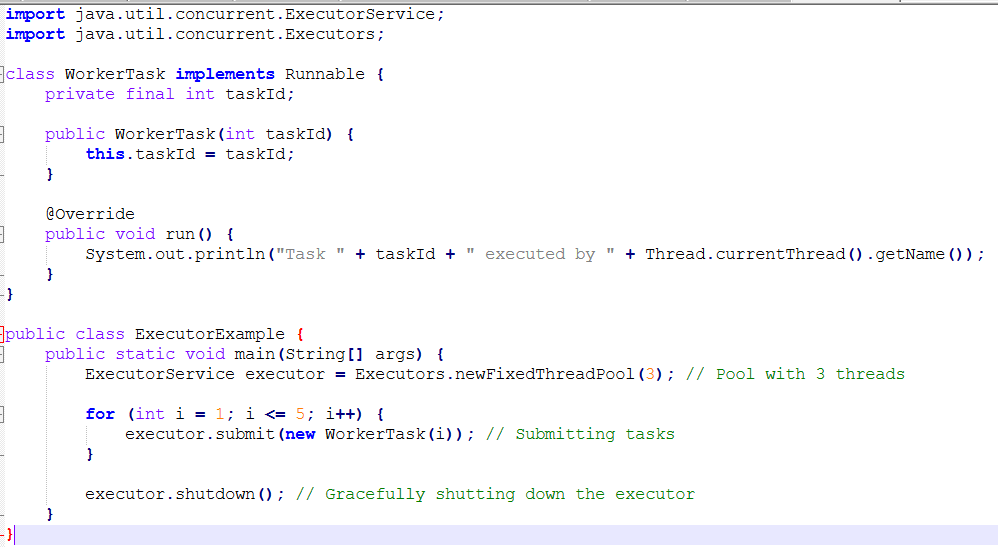
1. Explain about Executor framework and its example?

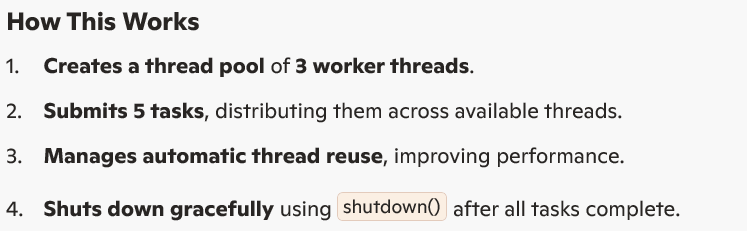
The Java Executor framework, introduced in JDK 5, is a powerful tool for managing and executing tasks asynchronously. It is used to run the Runnable objects without creating new threads every time and mostly reusing the already created threads.



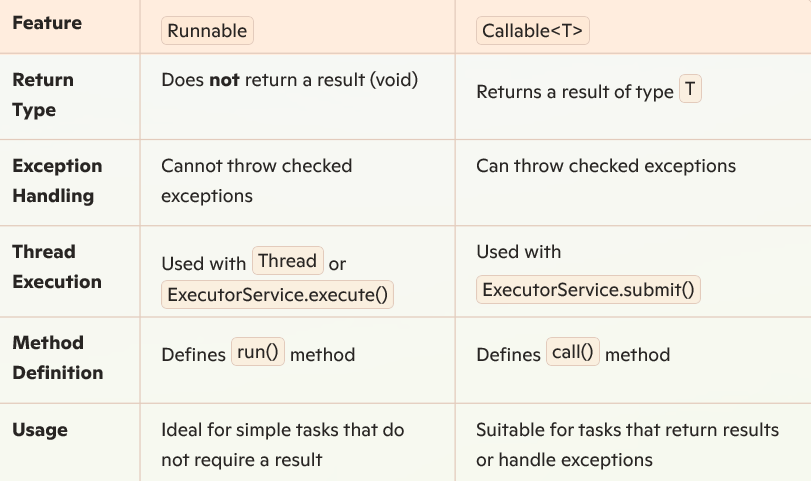




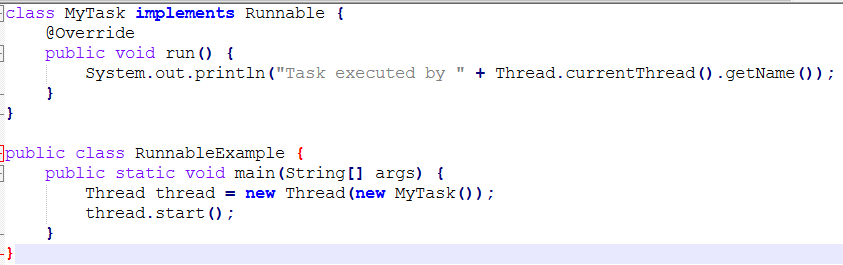




1. **Difference between runnable and callable?**

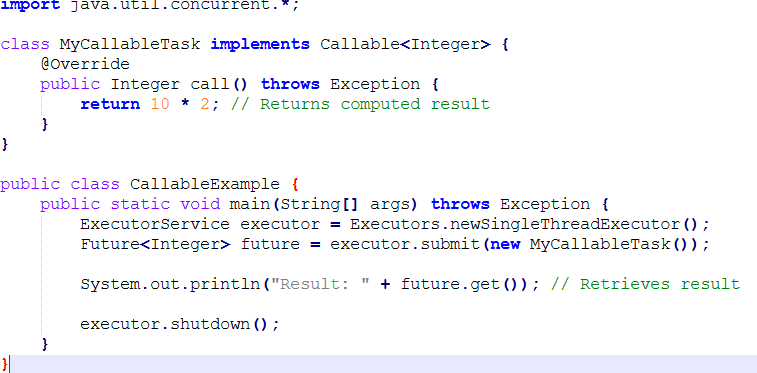


**Example Runnable:**



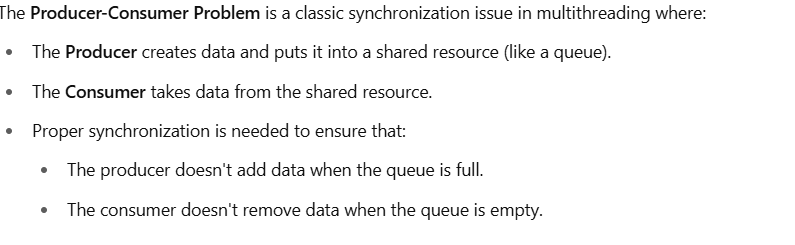
✅ **Does not return a result. Simply executes a task.**

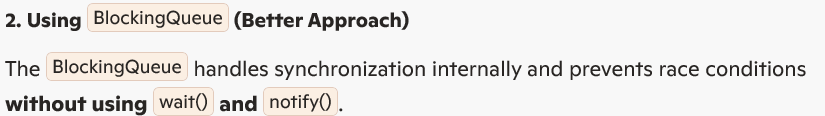
**Example Callable:**

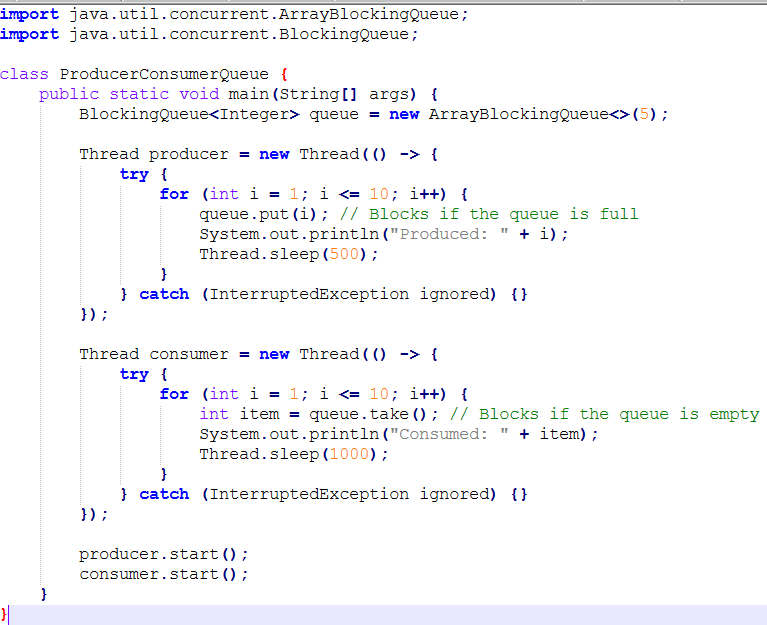
****

**✅ Returns a value and handles exceptions. Uses Future to get the result asynchronously.**

1. **How to resolve producer and consumer problem in the threads?**

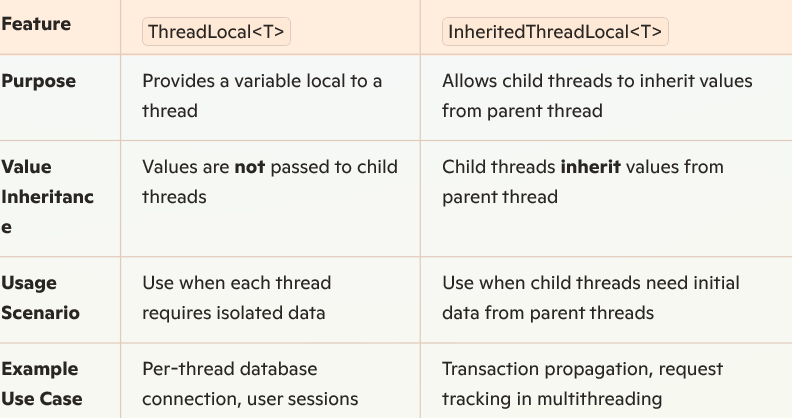


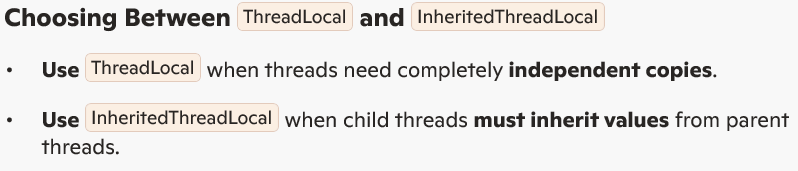




1. **Difference between thread local and inherited thread local?**

Both ThreadLocal and InheritedThreadLocal provide thread-local variables that help maintain separate copies for each thread. However, InheritedThreadLocal extends the concept by allowing **child threads to inherit values** from their parent threads.

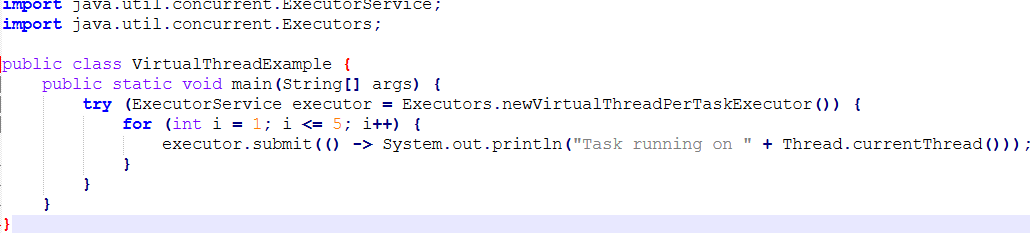


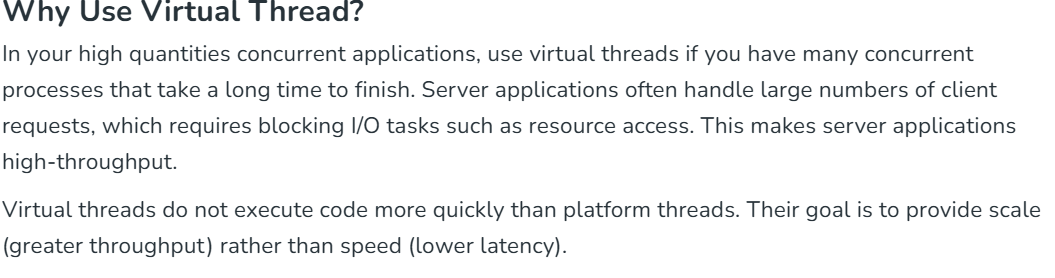


1. **Explain about virtual threads?**

In Java, Virtual threads are now supported by the Java Platform. Virtual threads are lightweight threads that greatly minimize the effort required to create, operate, and manage high volumes systems that are concurrent. As a result, they are more efficient and scalable than standard platform threads.

A virtual thread is an instance of java.lang.Thread, independent of any OS thread, is used to run programs. The Java runtime suspends the virtual thread until it resumes when the code calls a blocked I/O operation. Virtual threads have a limited call stack and can only execute one HTTP client call or JDBC query. They are suitable for delayed operations, but not for extended CPU-intensive tasks.





1. **Difference between synchronised, volatile and Atomic?**

