THREADPOOL

- A thread pool reuses previously created threads to execute current tasks and offers a solution to the problem of thread cycle overhead and resource thrashing.
- Since the thread is already existing when the request arrives, the delay introduced by thread creation is eliminated, making the application more responsive.
- Java provides the Executor framework which is centred around the Executor interface, its sub-interface –**ExecutorService** and the class-**ThreadPoolExecutor**, which implements both of these interfaces.
- By using the executor, one only has to implement the Runnable objects and send them to the executor to execute.

- Executor Thread Pool Methods:

Method	Description
`newFixedThreadPool(int)`	Creates a fixed size thread pool.
`newCachedThreadPool()`	Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available
`newSingleThreadExecutor()`	Creates a single thread.

- In case of a fixed thread pool, if all threads are being currently run by the executor then the pending tasks are placed in a queue and are executed when a thread becomes idle.

Risks in using Thread Pools

1. **Deadlock**: While deadlock can occur in any multi-threaded program, thread pools introduce another case of deadlock, one in which all the executing threads are waiting for the results from the blocked threads waiting in the queue due to the unavailability of threads for execution.
2. **Thread Leakage**: Thread Leakage occurs if a thread is removed from the pool to execute a task but not returned to it when the task completed. As an example, if the thread throws an exception and pool class does not catch this exception, then the thread will simply exit, reducing the size of the thread pool by one. If this repeats many times, then the pool would eventually become empty and no threads would be available to execute other requests.
3. **Resource Thrashing**: If the thread pool size is very large then time is wasted in context switching between threads. Having more threads than the optimal number may cause starvation problem leading to resource thrashing as explained.

Tuning Thread Pool

- The optimum size of the thread pool depends on the number of processors available and the nature of the tasks. On a N processor system for a queue of only computation type processes, a maximum thread pool size of N or N+1 will achieve the maximum efficiency.
- But tasks may wait for I/O and in such a case we take into account the ratio of waiting time(W) and service time(S) for a request; resulting in a maximum pool size of N*(1+ W/S) for maximum efficiency.