es/74tdo0adae1d049a4410e9076ee9ee4b55242" class="notecardImageClass"
d="74tdo0adae1d049a4410e9076ee9ee4b55242"></div><br></div><div> <br></div></div><div><br></div>
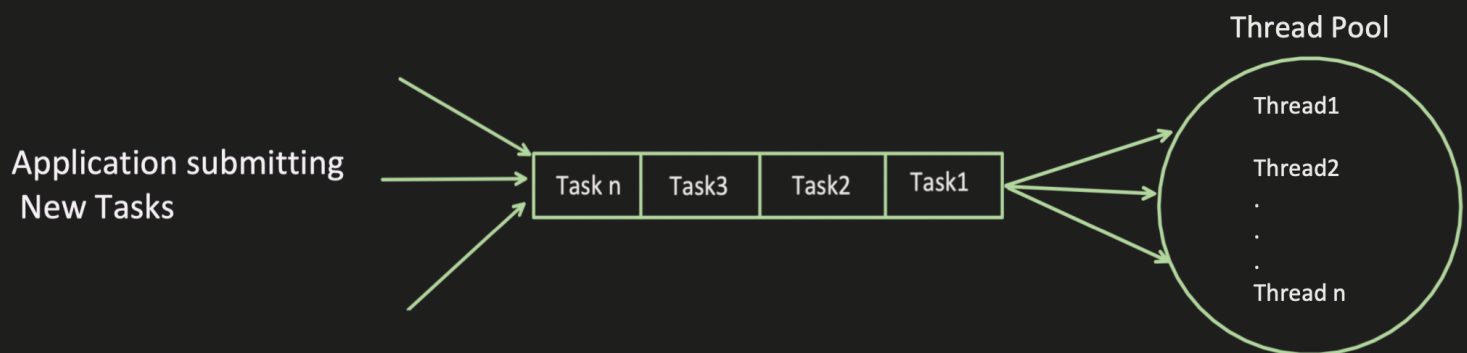/div><div><br></div></div> </div> </div> </div>

Download as PDF

Thread Pools and ThreadPoolExecutor

**"Concept && Coding" YT Video Notes**

## What is ThreadPool:

- It's a collection of threads (aka workers), which are available to perform the submitted tasks.
- Once task completed, worker thread get back to Thread Pool and wait for new task to assigned.
- Means threads can be reused.



## What's the Advantage of Thread Pool?

- **Thread Creation time can be saved:**
  - When each thread created, space is allocated to it (stack, heap, program counter etc..) and this takes time.
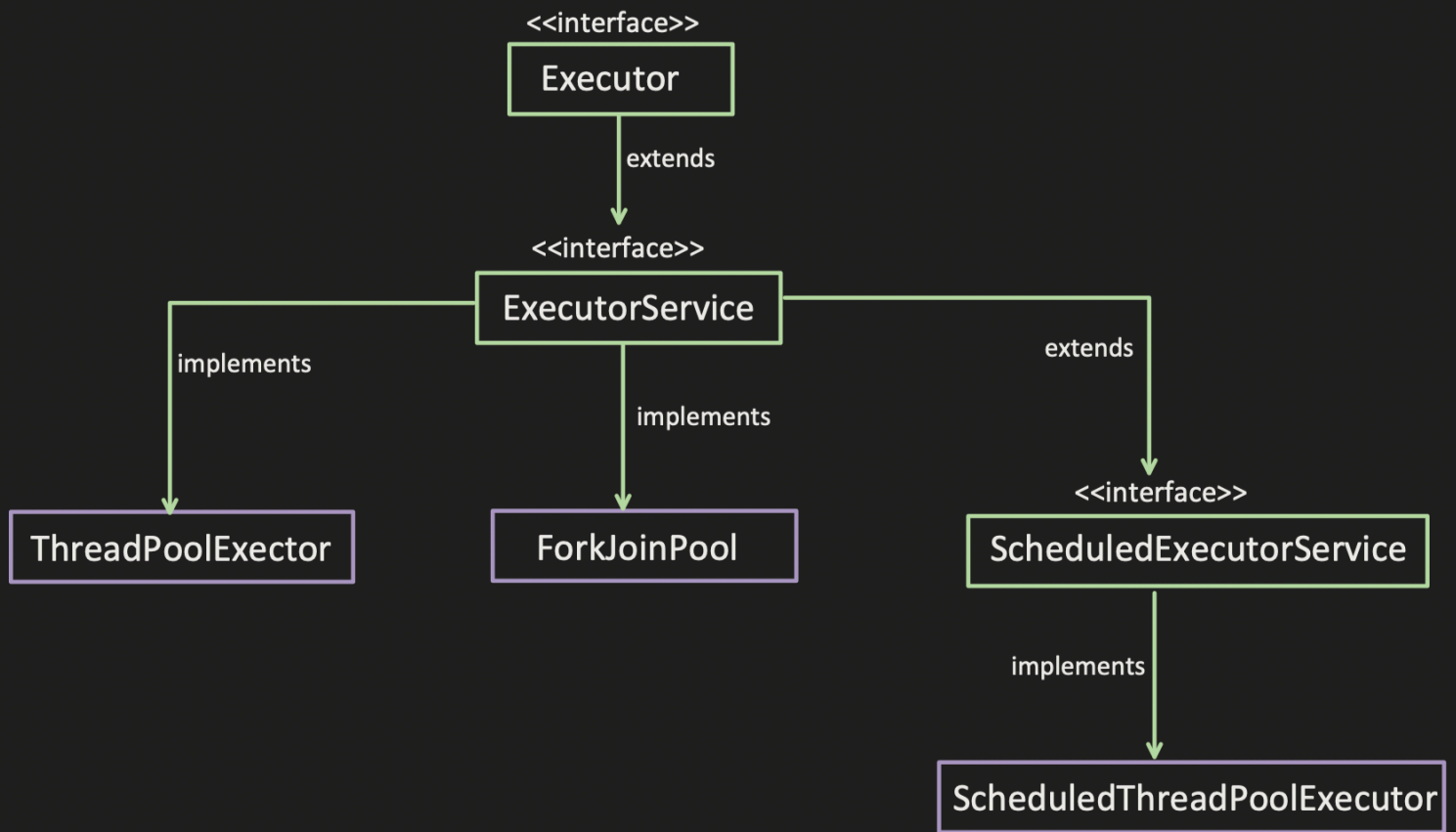  - With thread, this can be avoided by reusing the thread.

- **Overhead of managing the Thread lifecycle can be removed:**
  - Thread has different state like Running, Waiting, terminate etc. And managing thread state includes complexity.
  - Thread pool abstract away this management.

- **Increased the performance:**
  - More threads means, more **Context Switching** time, using control over thread creation, excess context switching can be avoided.

**package** *java.util.concurrent;*

```
                              <<interface>>
                                Executor

                                   │ extends
                                   ▼

                              <<interface>>
          ┌────────────────── ExecutorService ──────────────────┐
          │ implements              │ implements      extends    │
          ▼                         ▼                            ▼
  ThreadPoolExector           ForkJoinPool              <<interface>>
                                                    ScheduledExecutorService

                                                               │ implements
                                                               ▼
                                                  ScheduledThreadPoolExecutor
```

**ThreadPoolExector:**

It's helps to create a customizable ThreadPool.

```
public ThreadPoolExecutor
                         ( int corePoolSize
                           int maximumPoolSize,
                           long keepAliveTime,
                           TimeUnit unit,
                           BlockingQueue<Runnable> workQueue,
                           ThreadFactory threadFactory,
                           RejectedExecutionHandler handler)
```

- *corePoolSize*:
   Number of threads are initially created and keep in the pool, even if they are idle.

- **allowCoreThreadTimeOut**:
   If this property is set to TRUE (by default its FLASE), idle thread kept Alive till time specified
   by 'KeepAliveTime'.

- **KeepAliveTime**:
   Thread, which are idle get terminated after this time.

- **maxPoolSize**:
   Maximum number of thread allowed in a pool.
   If no. of thread are == corePoolSize and queue is also full, then new threads are created
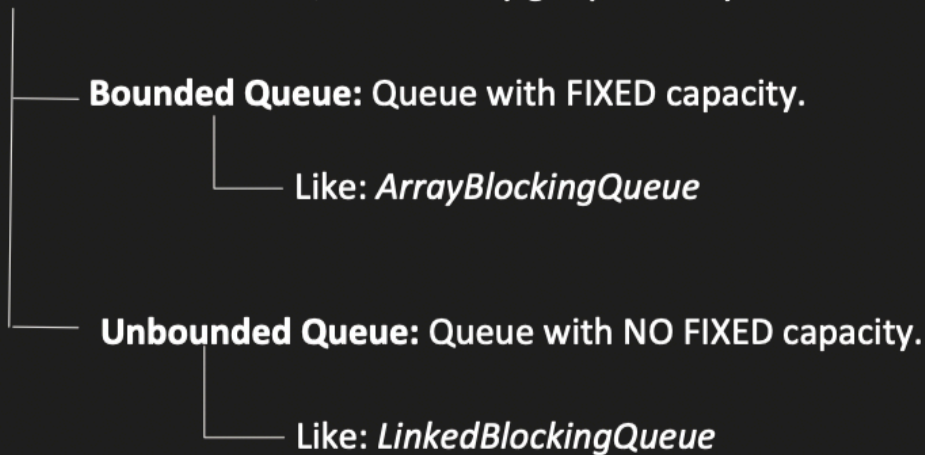(till its less than 'maxPoolSize').

Excess thread, will remain in pool, this pool is not shutdown or if *allowCoreThreadTimeOut* is
set to true, then excess thread get terminated after remain idle for *KeepAliveTime*.

- **TimeUnit:**
   TimeUnit for the keepAliveTime, whether *Millisecond* or *Second* or *Hours* etc.

## - BlockingQueue:
Queue used to hold task, before they got picked by the worker thread.

**Bounded Queue:** Queue with FIXED capacity.

Like: *ArrayBlockingQueue*

**Unbounded Queue:** Queue with NO FIXED capacity.
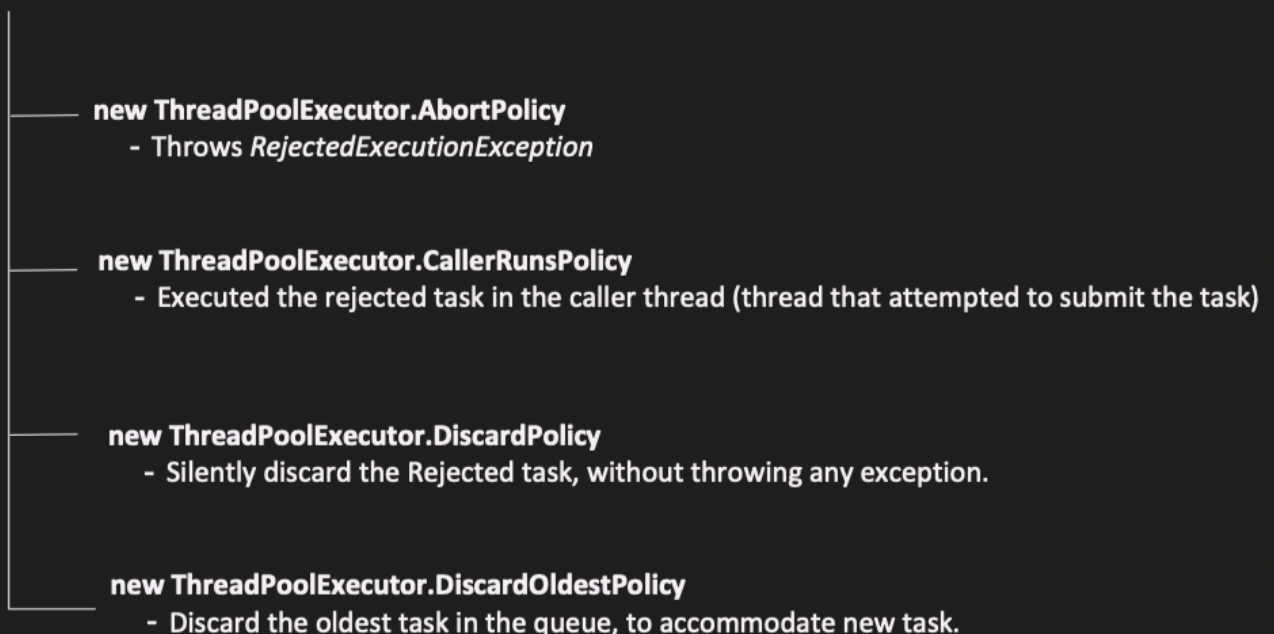
Like: *LinkedBlockingQueue*


## - ThreadFactory:
Factory for creating new thread. ThreadPoolExecutor use this to create new thread, this Factory provide us an interface to:

- To give custom Thread name
- To give custom Thread priority
- To set Thread Daemon flag etc.


## - RejectedExecutionHandler:
Handler for tasks that can not be accepted by thread pool.
Generally logging logic can be put here. For debugging purpose.

**new ThreadPoolExecutor.AbortPolicy**
- Throws *RejectedExecutionException*

**new ThreadPoolExecutor.CallerRunsPolicy**
- Executed the rejected task in the caller thread (thread that attempted to submit the task)

**new ThreadPoolExecutor.DiscardPolicy**
- Silently discard the Rejected task, without throwing any exception.

**new ThreadPoolExecutor.DiscardOldestPolicy**
- Discard the oldest task in the queue, to accommodate new task.

## Example:
--------------

```java
public class Main {

    public static void main(String args[]) {


        ThreadPoolExecutor poolExecutor = new ThreadPoolExecutor( corePoolSize: 2, maximumPoolSize: 5, keepAliveTime: 1,
                TimeUnit.HOURS, new ArrayBlockingQueue<>( capacity: 10), new CustomThreadFactor(),
                new CustomRejectedHandler());

        poolExecutor.allowCoreThreadTimeOut( value: true);

        //submit task
        for(int i=0; i< 25 ; i++){
            poolExecutor.submit(() -> {
                try {
                    Thread.sleep( millis: 5000);
                    System.out.println("Thread name:" + Thread.currentThread().getName());
                }catch (Exception e) {
                }
            });
        }

        poolExecutor.shutdown();

    }
}


class CustomRejectedHandler implements RejectedExecutionHandler{

    @Override
    public void rejectedExecution(Runnable r, ThreadPoolExecutor executor) {
        //logging
        System.out.println("Task denied:" + r.toString());
    }
}

class CustomThreadFactor implements ThreadFactory{

    @Override
    public Thread newThread(Runnable r) {

        Thread th = new Thread(r);
        return th;

    }
}
```

# Interview question:

Why you have taken corePoolSize as 2, why not 10 or 15 or another number, what's the logic?

## My Answer:
------------------

Generally, the ThreadPool min and max size are depend on various factors like:
  - CPU Cores
  - JVM Memory
  - Task Nature (CPU Intensive or I/O Intensive)
  - Concurrency Requirement (Want high or medium or low concurrency)
  - Memory Required to process a request
  - Throughput etc.

And its an iterative process to update the min and max values based on monitoring.

 Formula to find the no. of thread:

Max No of thread = No. of CUP Core  * ( 1 + Request waiting time/processing time)

No. of CUP Core = 4
Request waiting time = 10ms
Processing time = 100ms

But this formula, do not consider Memory yet, which need to be consider...