**Executor Framework**

A framework having a bunch of components that are used for managing worker threads efficiently is referred to as Executor Framework.

The Executor API reduces the execution of the task from the actual task to be executed through the Executors.

The executor framework is an implementation of the Producer-Consumer pattern. The **java.util.concurrent.Executors class** provides a set of methods for creating **ThreadPools** of worker threads.

**In order to use the executor framework, we have to create a thread pool for executing the task by submitting that task to that thread pool.**

**Why we have to create such thread pools when we already have the java.lang.Thread class for creating an object and Runnable/Callable interface for achieving parallelism by implementing them?**

**Answer: -** The reason for creating such thread pools are as follows:

1. We need to create a large number of threads for adding a new thread without any throttling for each and every process. Due to which it requires more memory and cause wastage of resource. When each thread is swapped, the CPU starts to spend too much time.
2. When we create a new thread for executing a new task cause overhead of thread creation. In order to manage this thread life-cycle, the execution time increase respectively.

## Types of Executors

In Java, there are different types of executors available which are as follows: -

### SingleThreadExecutor

### FixedThreadPool(n)

### CachedThreadPool

### ScheduledExecutor

**Executor Framework Interface List: -**

1. public interface Executor
2. public interface ExecutorService extends Executor
3. public interface ScheduledExecutorService extends ExecutorService

**Executor Framework Class List: -**

1. public abstract class AbstractExecutorService implements ExecutorService
2. public class ForkJoinPool extends AbstractExecutorService
3. public class ThreadPoolExecutor extends AbstractExecutorService
4. public class ScheduledThreadPoolExecutor extends ThreadPoolExecutor implements ScheduledExecutorService

**Utility classes: -**

1. public class Executors extends Object

**Executors Class**

**Module: -** java.base

**Package: -** java.util.concurrent

**Since:** 1.5

public class Executors extends Object

Executors Class provides Factory and utility methods for **Executor, ExecutorService, ScheduledExecutorService, ThreadFactory, and Callable classes** defined in this package.

**This class supports the following kinds of methods: -**

1. Methods that create and return an **ExecutorService** set up with commonly useful configuration settings.
2. Methods that create and return a **ScheduledExecutorService** set up with commonly useful configuration settings.
3. Methods that create and return a "wrapped" **ExecutorService**, that disables reconfiguration by making implementation-specific methods inaccessible.
4. Methods that create and return a **ThreadFactory** that sets newly created threads to a known state.
5. Methods that create and return a **Callable** out of other closure-like forms, so they can be used in execution methods requiring Callable.

**Important methods of Executors Class**

**public static ExecutorService newFixedThreadPool (int nThreads)**

Creates a thread pool that reuses a fixed number of threads operating off a shared unbounded queue.

At any point, at most nThreads threads will be active processing tasks. If additional tasks are submitted when all threads are active, they will wait in the queue until a thread is available.

If any thread terminates due to a failure during execution prior to shutdown, a new one will take its place if needed to execute subsequent tasks. The threads in the pool will exist until it is explicitly shutdown.

**Parameters:** nThreads - the number of threads in the pool

**Returns:** the newly created thread pool

**Throws: IllegalArgumentException** - if nThreads <= 0

**public static ExecutorService newSingleThreadExecutor()**

Creates an Executor that uses a single worker thread operating off an unbounded queue. (Note however that if this single thread terminates due to a failure during execution prior to shutdown, a new one will take its place if needed to execute subsequent tasks.) **Tasks are guaranteed to execute sequentially**, and no more than one task will be active at any given time. Unlike the otherwise equivalent newFixedThreadPool(1) the returned executor is guaranteed not to be reconfigurable to use additional threads.

**Returns**: the newly created single-threaded Executor.

public static ExecutorService newCachedThreadPool()

Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available. These pools will typically improve the performance of programs that execute many short-lived asynchronous tasks. Calls to execute will reuse previously constructed threads if available. If no existing thread is available, a new thread will be created and added to the pool. Threads that have not been used for sixty seconds are terminated and removed from the cache. Thus, a pool that remains idle for long enough will not consume any resources. Note that pools with similar properties but different details (for example, timeout parameters) may be created using ThreadPoolExecutor constructors.

**Returns**: the newly created thread pool

public static ScheduledExecutorService newScheduledThreadPool (int corePoolSize)

Creates a thread pool that can schedule commands to run after a given delay, or to execute periodically.

**Parameters:** corePoolSize - the number of threads to keep in the pool, even if they are idle

**Returns:** the newly created scheduled thread pool

**Throws:** IllegalArgumentException - if corePoolSize < 0