**ExecutorService in Java**



Photo by [Bozhin Karaivanov](https://unsplash.com/@bkaraivanov?utm_source=medium&utm_medium=referral" \t "_blank) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

Multi-threaded program is the process of dividing the tasks among multiple threads to improve performance. When you are running any program, it will be done by a single thread called the main thread. To make a program multi-threaded, we need to spawn threads from the main thread. There are two ways of doing it.

1. Extend Thread class
2. Implement Runnable interface

Internally Thread class implements the Runnable interface and has its own methods, to have more hold on created threads. More details can be found [here](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html).

Starting from Java 5, JDK provides API ExecutorService that simplifies running tasks in asynchronous mode. It provides a pool of threads and an API to assign tasks to it.

ExecutorService is an interface provided by JDK. Its object can be obtained using class Executors. There are 5 types of thread pools we can create

1. **ExecutorService execService = Executors.*newSingleThreadExecutor*(10); //**Creates an Executor that uses a single thread
2. **ExecutorService execService = Executors.newFixedThreadPool(10); //**Creates a thread pool that reuses a fixed number of threads
3. **ExecutorService execService = Executors.*newScheduledThreadPool*(10); //**Creates a thread pool that can schedule commands to run after a given delay, or to execute periodically.
4. **ExecutorService execService = Executors.*newCachedThreadPool*(); //**Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available.
5. **ExecutorService execService = Executors.*newWorkStealingPool()*; //**Creates a work-stealing thread pool using all [**available processors**](https://docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#availableProcessors--) as its target parallelism level

Once the thread pool is created, we can perform the required action on them. Most used methods and their functionalities are as below:

1. **execute(Runnable) :** Executes the given command at some time in the future. It does not return anything.
2. **submit(Runnable) :** executes the passed task and returns a Future Object representing the executed task.
3. [**invokeAll**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#invokeAll-java.util.Collection-)([**Collection**](https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html)<? extends [**Callable**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html)<T>> tasks): Executes the given tasks, returning a list of Futures holding their status and results when all complete.
4. [**invokeAny**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#invokeAny-java.util.Collection-)([**Collection**](https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html)<? extends [**Callable**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html)<T>> tasks): Executes the given tasks, returning the result of one that has completed successfully (i.e., without throwing an exception), if any do.
5. **awaitTermination(long timeout, TimeUnit unit):**Blocks until all tasks have completed execution after a shutdown request, or the timeout occurs, or the current thread is interrupted, whichever happens first.
6. [**isShutdown**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#isShutdown--)(): Returns true if this executor has been shut down.
7. [**isTerminated**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#isTerminated--)(): Returns true if all tasks have been completed following shut down.
8. **shutdown() :** Initiates the shutdown so that previously submitted tasks are executed, but no new task will be accepted. Invocation has no additional effect if already shut down.
9. **shutdownNow() :** Attempts to stop all actively executing tasks, halts the processing of waiting tasks, and returns a list of the tasks that were awaiting execution.

*More details on ExecutorService class can be found in*[*official documentation*](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html)

One thing, what is Future now?

A Future is used to track the progress of one or more asynchronous tasks. It represents the result of an asynchronous computation. Methods are provided to check if the computation is complete, to wait for its completion, and to retrieve the result of the computation.

Once tasks are completed we can use the *get()* method on it to get all data processed. For more details on the Future class, [check here](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html).

Below is the code for cached thread pool executor service, considering the Task class implements Runnable interface:

The sample code with possible types of executors can be found on Github [here](https://github.com/oldfr/ExecutorServiceOverview).

Thank you for reading. Happy exploring!!!