ForkJoinPool:

All threads in the pool attempt to find and execute tasks submitted to the pool and/or created by other active tasks (eventually blocking waiting for work if none exist)

https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html

AbstractExecutorService (provide default implementations of executor service)

This method helps to manage or produce a Future for tracking the progress of 1 or more asynchronous tasks. The main reason of using this method is so that it can create and return a Future that can be used to cancel execution and/or wait for completion.

An [Executor](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Executor.html) that provides methods to manage termination and methods that can produce a [Future](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html) for tracking progress of one or more asynchronous tasks.

An ExecutorService can be shut down, which will cause it to reject new tasks. Two different methods are provided for shutting down an ExecutorService. The [shutdown()](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorService.html#shutdown()) method will allow previously submitted tasks to execute before terminating, while the [shutdownNow()](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorService.html" \l "shutdownNow()) method prevents waiting tasks from starting and attempts to stop currently executing tasks. Upon termination, an executor has no tasks actively executing, no tasks awaiting execution, and no new tasks can be submitted. An unused ExecutorService should be shut down to allow reclamation of its resources.

Method submit extends base method [Executor.execute(java.lang.Runnable)](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Executor.html" \l "execute(java.lang.Runnable)) by creating and returning a [Future](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html) that can be used to cancel execution and/or wait for completion. Methods invokeAny and invokeAll perform the most commonly useful forms of bulk execution, executing a collection of tasks and then waiting for at least one, or all, to complete. (Class [ExecutorCompletionService](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorCompletionService.html" \o "class in java.util.concurrent) can be used to write customized variants of these methods.)

ScheduledThreadPoolExecutor

<https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ScheduledThreadPoolExecutor.html>

It helps to schedule the commands ad more flexible as compared to ThreadPoolExecutor. However, once the task is being delayed, there is no guarantee when the task will be executed.  A new thread will be created only if the queue is full.

A [ThreadPoolExecutor](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadPoolExecutor.html" \o "class in java.util.concurrent) that can additionally schedule commands to run after a given delay, or to execute periodically. This class is preferable to [Timer](https://docs.oracle.com/javase/7/docs/api/java/util/Timer.html) when multiple worker threads are needed, or when the additional flexibility or capabilities of [ThreadPoolExecutor](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadPoolExecutor.html" \o "class in java.util.concurrent) (which this class extends) are required.

Delayed tasks execute no sooner than they are enabled, but without any real-time guarantees about when, after they are enabled, they will commence. Tasks scheduled for exactly the same execution time are enabled in first-in-first-out (FIFO) order of submission.

ThreadPoolExecutor extends AbstractExecutorService

Thread pools address two different problems: they usually provide improved performance when executing large numbers of asynchronous tasks, due to reduced per-task invocation overhead, and they provide a means of bounding and managing the resources, including threads, consumed when executing a collection of tasks. Each ThreadPoolExecutor also maintains some basic statistics, such as the number of completed tasks.