Advantages of using an ExecutorService

The job of managing threads is simplified.

ExecutorService implementations let you stay focused on tasks that need to be run, rather than thread creation and management.

Creating Threads is Expensive

Creating threads, destroying threads, and then creating them again can be expensive.

A thread pool mitigates the cost, by keeping a set of threads around, in a pool, for current and future work.

Threads, once they complete one task, can then be reassigned to another task, without the expense of destroying that thread and creating a new one.

The Mechanics of a Thread Pool

A thread pool consists of three components.

Worker Threads are available in a pool to execute tasks. They're pre-created and kept alive, throughout the lifetime of the application.

Submitted Tasks are placed in a First-In First-Out queue. Threads pop tasks from the queue, and execute them, so they're executed in the order they're submitted.

The Thread Pool Manager allocates tasks to threads, and ensures proper thread synchronization.

Java's Thread Pool classes

Java has five variations of the Thread Pool.

Class	Description	Executors method
FixedThreadPool	Has a fixed number of threads.	newFixedThreadPool
CachedThreadPool	Creates new threads as needed, so its a variable size pool	newCachedThreadPool
ScheduledThreadPool	Can schedule tasks to run at a specific time or repeatedly at regular intervals.	newScheduledThreadPool
WorkStealingPool	Uses a work-stealing algorithm to distribute tasks among the threads in the pool.	newWorkStealingPool
ForkJoinPool	Specialized WorkStealingPool for executing ForkJoinTasks.	n/a