Managing Threads

These are the ExecutorService classes, and they exist to manage the creation and execution of threads.

Managing Individual Threads

When using a Thread class, you have rudimentary control over that thread.

You can interrupt a thread, and join it to another thread.

You can name the thread, try to prioritize it, and start each manually, one at a time.

You can also pass it an UncaughtExceptionHandler, to deal with exceptions that happen in a thread.

Managing Individual Threads

Managing threads manually can be complex and error-prone.

It can lead to complex issues like resource contention, thread creation overhead, and scalability challenges.

For these reasons, you'll want to use an ExecutorService, even when working with a single thread.

Benefits of Managing Threads with an Implementation of ExecutorService

The ExecutorService type in Java is an interface. Java provides several implementations of this type which provide the following benefits:

- Simplify thread management, by abstracting execution, to the level of tasks which need to be run.
- Use Thread Pools, reducing the cost of creating new threads.
- Efficient Scaling, by utilizing multiple processor cores.
- Built-in synchronization, reducing concurrency-related errors.
- Graceful Shutdown, preventing resource leaks.
- Scheduled implementations exist to further help with management workflows.