Threads

A Thread is a lightweight process and the smallest unit of execution within a program. In Java, threads allow multiple tasks to run concurrently, making the application more efficient, especially when dealing with tasks that require waiting (like I/O operations). Each thread operates independently but shares the process's resources, such as memory and file handles.

Creating Threads in Java

1. By Extending the `Thread` Class
2. By Implementing the `Runnable` Interface

Thread Lifecycle

New: The thread is created but not yet started.
Runnable: The thread is ready to run and is waiting for CPU time. It may be running or waiting to be scheduled.
Blocked: The thread is waiting for a monitor lock to enter a synchronized block/method.
Waiting: The thread is waiting indefinitely for another thread to perform a specific action.
Timed Waiting: The thread is waiting for another thread to perform an action for up to a specified waiting time.
Terminated: The thread has completed its execution.

Thread Synchronization

Synchronization is a way to control access to shared resources in a multithreaded environment to prevent race conditions and ensure data consistency.

Synchronized Methods

A method can be declared `synchronized` to ensure that only one thread can execute it at a time for a particular object instance.

Synchronized Blocks

You can use synchronized blocks to synchronize specific portions of code rather than the entire method, providing finer control over synchronization.

Thread Pool

A Thread Pool is a collection of pre-created threads that can be reused for executing tasks. Instead of creating a new thread every time a task needs to run, the thread pool reuses existing threads, which reduces the overhead of thread creation and destruction, improves performance, and provides better management of system resources.

Types of Thread Pools in Java

1. **Fixed Thread Pool (`newFixedThreadPool(int n)`)**: Creates a thread pool with a fixed number of threads. If all threads are busy, tasks are placed in a queue until a thread is free.
2. **Cached Thread Pool (`newCachedThreadPool()`)**: Creates a thread pool that creates new threads as needed but will reuse previously created threads when available. Suitable for short-lived asynchronous tasks.
3. **Single Thread Executor (`newSingleThreadExecutor()`)**: Creates a single-threaded executor to ensure that tasks are executed sequentially.
4. **Scheduled Thread Pool (`newScheduledThreadPool(int n)`)**: Creates a thread pool that can schedule commands to run after a given delay or periodically.