C# Multithreading Cheat Sheet

Thread vs Task

- Thread: Low-level, used for dedicated background work. Manual start and control.
- Task: Higher-level abstraction. Uses ThreadPool internally. Prefer Task for most use cases.

ThreadPool

- A shared pool of threads managed by .NET.
- Used by Task, timers, async/await, and parallel operations.
- Avoid long-running tasks in ThreadPool to prevent starvation.

async/await

- Simplifies asynchronous programming.
- Use for I/O-bound operations, not CPU-bound.
- Avoid blocking calls like .Result or .Wait() on async code to prevent deadlocks.

lock, Monitor, and Mutex

- lock(obj): Simple way to ensure mutual exclusion in a code block.
- Monitor: Provides more control (TryEnter, Pulse, Wait).
- Mutex: Use for cross-process synchronization.

Concurrent Collections

- Thread-safe alternatives to List, Dictionary, etc.
- Use ConcurrentQueue, ConcurrentDictionary, BlockingCollection.
- Avoid manual locks when using these collections.

Parallel Programming

- Use Parallel.For / Parallel.ForEach for CPU-bound work.
- Use PLINQ (Parallel LINQ) for data-parallel operations.
- Be aware of overhead and throttling; test performance benefits.

CancellationToken

- Enables cooperative cancellation of tasks.
- Pass token to methods and check token.IsCancellationRequested or throw if needed.
- Used with Task.Run, async/await, and timers.

Thread Safety Tips

- Avoid shared mutable state.
- Prefer immutability and stateless functions.
- Use proper synchronization primitives or thread-safe collections.
- Watch out for race conditions and deadlocks.