**Chapter 7: Synchronization Constructs**

Simple blocking methods

These wait for another thread to finish or for a period of time to elapse. Sleep, Join, and Task.Wait are simple blocking methods.

Locking constructs

These limit the number of threads that can perform some activity or execute a section of code at a time. Exclusive locking constructs are most common — these allow just one thread in at a time, and allow competing threads to access common data without interfering with each other. The standard exclusive locking constructs are lock (Monitor.Enter/Monitor.Exit), Mutex, and SpinLock. The nonexclusive locking constructs are Semaphore, SemaphoreSlim, and the reader/writer locks.

Signaling constructs

These allow a thread to pause until receiving a notification from another, avoiding the need for inefficient polling. There are two commonly used signaling devices: event wait handles and Monitor’s Wait/Pulse methods. Framework 4.0 introduces the CountdownEvent and Barrier classes.

Nonblocking synchronization constructs

These protect access to a common field by calling upon processor primitives. The CLR and C# provide the following nonblocking constructs: Thread.MemoryBarrier, Thread.VolatileRead, Thread.VolatileWrite, the volatile keyword, and the Interlocked class.

We can start with an example of multi threaded code that fails due to lack of synchronization I.e., we can showcase the need of synchronization and then we can take it forward. Since using stand alone thread is not recommended, so we can talk less on join, wait etc. We can talk on How do we synchronize async calls, Auto reset and manual reset events apart from what you mentioned above with example.