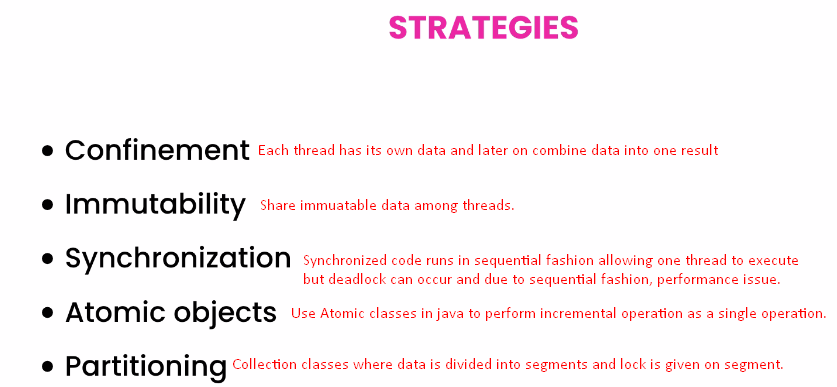
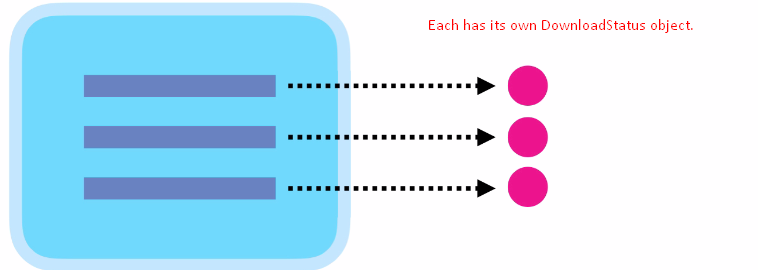
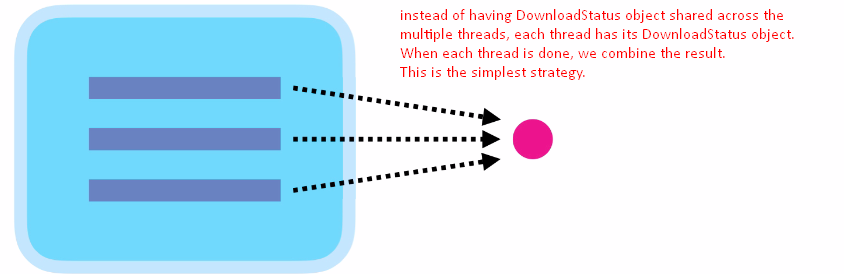
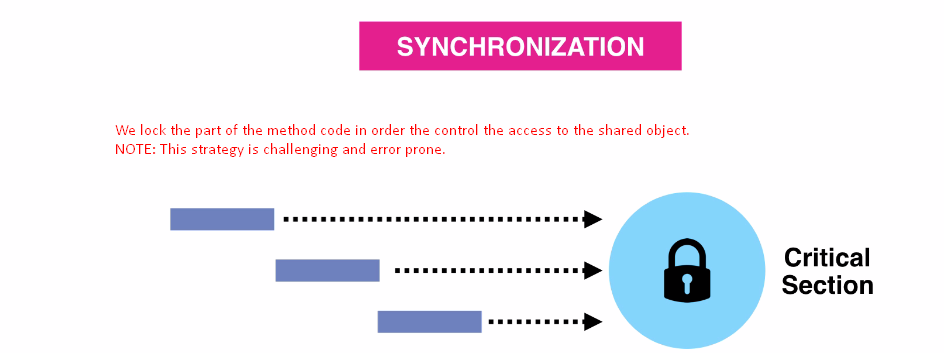
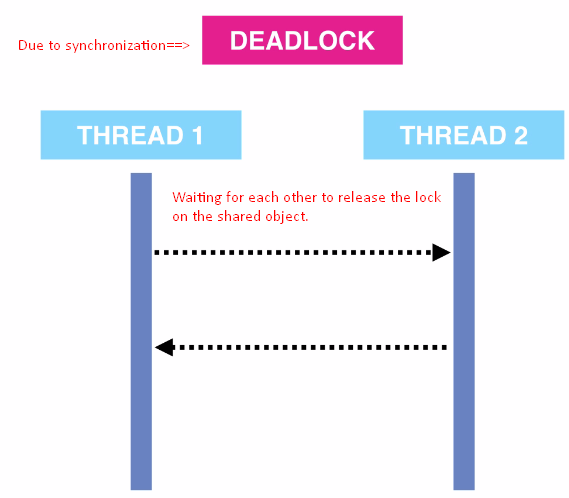
1.   
   
2. We have strategies to write **Thread-Safe** Code. That is we can safely execute multiple threads.
3. **Confinement**: The simple solution is not to share data across the multiple threads.   
   We want to confine/restrict each thread to its own data.   
   We will show you this in the next video.
4. **Immutability:** After creating object, its state can’t be changed. For example, String class.
5. **Synchronization:** To prevent multiple threads to access the shared object concurrently. We do this using **lock. **
6. **Problems with Synchronization:** 
   1. Code runs in sequential fashion.
   2. Deadlock can occur.
7. 
8. **Atomic Objects:** Another solution is to use Atomic class such as **AtomicInteger**. Such classes allow us to achieve thread-safely without using locks. If you increment AtomicInteger, the JVM will execute the increment operation as one single atomic operation. It will not break down into 3 smaller operations. That is why it’s called **Atomic** (atom) which can’t be broken down.
9. **Partitioning**: To partitioning data into segments that can be accessed concurrently. Java provides a number of collection classes that use concurrency using partitioning.   
   So multiple threads can access collection object but at a time one thread can access a segment of that collection. 