1. Introduction
   1. What is thread synchronization?
      1. “When multiple threads share an object and it’s modified by one or more of them, indeterminate results may occur (as we’ll see in the examples) unless access to the shared object is managed properly. If one thread is in the process of updating a shared object and another thread also tries to update it, it’s unclear which thread’s update takes effect. When this happens, the program’s behavior cannot be trusted—sometimes the program will produce the correct results, and sometimes it won’t. In either case, there’ll be no indication that the shared object was manipulated incorrectly. The problem can be solved by giving only one thread at a time exclusive access to code that manipulates the shared object. During that time, other threads desiring to manipulate the object are kept waiting. When the thread with exclusive access to the object finishes manipulating it, one of the threads that was waiting is allowed to proceed. This process, called thread synchronization, coordinates access to shared data by multiple concurrent threads. By synchronizing threads in this manner, you can ensure that each thread accessing a shared object excludes all other threads from doing so simultaneously—this is called mutual exclusion.” (Deitel, Paul; Harvey Deitel (2011-07-15). Java How to Program (early objects) (9th Edition) (Deitel) (Kindle Locations 27821-27831). Pearson HE, Inc.. Kindle Edition.)
   2. “A common way to perform synchronization is to use Java’s built-in **monitors**. Every object has a monitor and a monitor lock (or intrinsic lock). The monitor ensures that its object’s monitor lock is held by a maximum of only one thread at any time. Monitors and monitor locks can thus be used to enforce mutual exclusion. If an operation requires the executing thread to hold a lock while the operation is performed, a thread must acquire the lock before proceeding with the operation. Other threads attempting to perform an operation that requires the same lock will be blocked until the first thread releases the lock, at which point the blocked threads may attempt to acquire the lock and proceed with the operation. To specify that a thread must hold a monitor lock to execute a block of code, the code should be placed in a synchronized statement. Such code is said to be guarded by the monitor lock; a thread must acquire the lock to execute the guarded statements. The monitor allows only one thread at a time to execute statements within synchronized statements that lock on the same object, as only one thread at a time can hold the monitor lock. The synchronized statements are declared using the synchronized keyword: synchronized ( object ) {         statements } // end synchronized statement where object is the object whose monitor lock will be acquired; object is normally this if it’s the object in which the synchronized statement appears. If several synchronized statements are trying to execute on an object at the same time, only one of them may be active on the object—all the other threads attempting to enter a synchronized statement on the same object are placed in the blocked state. When a synchronized statement finishes executing, the object’s monitor lock is released and one of the blocked threads attempting to enter a synchronized statement can be allowed to acquire the lock to proceed. Java also allows synchronized methods. Before executing, a non-static synchronized method must acquire the lock on the object that’s used to call the method. Similary, a static synchronized method must acquire the lock on the class that’s used to call the method.
      Deitel, Paul; Harvey Deitel (2011-07-15). Java How to Program (early objects) (9th Edition) (Deitel) (Kindle Locations 27846-27856). Pearson HE, Inc.. Kindle Edition.
   3. Deitel, Paul; Harvey Deitel (2011-07-15). Java How to Program (early objects) (9th Edition) (Deitel) (Kindle Locations 27832-27843). Pearson HE, Inc.. Kindle Edition.
2. Example of thread synchronization in everyday life.
3. Example of thread synchronization in bioinformatics.

References:

Deitel, P. & Deitel H. (2011) Java How to Program 9th Edition p. 572. Pearson HE, Inc.. [Kindle Edition].

Chaichoompu, K., Kittitornkun, S. & Tongsima, S. 2006, "MT-ClustalW: multithreading multiple sequence alignment", Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th InternationalIEEE, , pp. 8 pp.

Khaladkar, C.S. (2009)"An Efficient Implementation of Needleman Wunsch Algorithm on Graphical Processing Units", .

Makino, J. 2011, "GRAPE" in Encyclopedia of Parallel Computing Springer, , pp. 788-796.