Let us build a utility class whose instances can be used to log messages in a

multithreaded environment.

A simple message logging utility class can be designed as in Listing 9.1.

Other application objects can log messages to the log ﬁle by invoking the log

method on a FileLogger instance. Inside the log method, the FileLogger

performs the necessary ﬁle operations required to log an input message. It is to

be noted that the log method is declared with the synchronized keyword. Without

synchronization, when multiple threads simultaneously try to log messages by

invoking the log method on the same FileLogger object, it could result in

unpredictable behavior. This is because multiple threads try to perform the same

set of open, write and close operations on the same log ﬁle at the same time.

The synchronized keyword ensures that only one thread is allowed to execute

the log method on a given FileLogger object at any given point in time. This

guarantee comes at a price. Declaring an object’s methods as synchronized can

have negative impact on the performance of an application that makes use of

those methods. In general, synchronized methods run many times slower than

their nonsynchronized counterparts. Hence an object’s methods should be

designed as synchronized methods only after careful consideration.

An object can have any number of synchronized methods. For a thread to

execute a synchronized method on an object, it needs to get a lock on that object.

The thread holds the lock on the object as long as the method execution continues.

While a thread holds a lock on an object, no other thread is given a lock on the

same object and hence other threads cannot execute any of the synchronized

methods on the same object.