Monitor, mutex, and semaphores can be confusing concepts initially. A monitor is made up of a mutex and a condition variable. One can think of a mutex as a subset of a monitor. Differences between a monitor and semaphore are discussed below.

Difference between Semaphore and Monitor#

- A monitor and a semaphore are interchangeable and theoretically, one can be constructed out of the other or one can be reduced to the other. However, monitors take care of atomically acquiring the necessary locks whereas, with semaphores, the onus of appropriately acquiring and releasing locks is on the developer, which can be error-prone.

- Semaphores are lightweight when compared to monitors, which are bloated. However, the tendency to misuse semaphores is far greater than monitors. When using a semaphore and mutex pair as an alternative to a monitor, it is easy to lock the wrong mutex or just forget to lock altogether. Even though both constructs can be used to solve the same problem, monitors provide a pre-packaged solution with less dependency on a developer's skill to get the locking right.

- Java monitors enforce correct locking by throwing the `IllegalMonitorState` exception object when methods on a condition variable are invoked without first acquiring the associated lock. The exception is in a way saying that either the object's lock/mutex was not acquired at all or that an incorrect lock was acquired.

- A semaphore can allow several threads access to a given resource or critical section, however, only a single thread at any point in time can **own** the monitor and access associated resource.

- Semaphores can be used to address the issue of **missed signals**, however with monitors additional state, called the predicate, needs to be maintained apart from the condition variable and the mutex which make up the monitor, to solve the issue of missed signals.