**Main Components/Utilities of concurrent package**

1. Executor - It used to run the Runnable objects without creating new threads every time and mostly re-using the already created threads.
2. ExecutorService - The Java ExecutorService is the interface which allows us to execute tasks on threads asynchronously. The ExecutorService helps in maintaining a pool of threads and assigns them tasks. It also provides the facility to queue up tasks until there is a free thread available if the number of tasks is more than the threads available.
3. ScheduledExecutorService - The Java ExecutorService is the interface which allows us to execute tasks on threads asynchronously. The Java ExecutorService interface is present in the java.util.concurrent package. The ExecutorService helps in maintaining a pool of threads and assigns them tasks. It also provides the facility to queue up tasks until there is a free thread available if the number of tasks is more than the threads available.
4. Future - In Java, the Callable interface was introduced in Java 5 as an alternative to existing Runnable interface. It wraps a task and pass it to a Thread or thread pool for asynchronous execution. The Callable represents an asynchronous computation, whose value is available through a Future object. All the code that needs to be executed asynchronously goes into the call() method. Both Callable and Future are parametric types and can be used to wrap classes like Integer, String, or anything else.
5. CountDownLatch - CountDownLatch is used to make sure that a task waits for other threads before it starts. To understand its application, let us consider a server where the main task can only start when all the required services have started. When we create an object of CountDownLatch, we specify the number of threads it should wait for, all such threads are required to be counted down by calling CountDownLatch.countDown() once they are completed or ready to the job. As soon as count reaches zero, the waiting task starts running.
6. CyclicBarrier - CyclicBarrier is used to make threads wait for each other. It is used when different threads process a part of computation and when all threads have completed the execution, the result needs to be combined in the parent thread. In other words, a CyclicBarrier is used when multiple threads carry out different sub tasks and the output of these sub tasks need to be combined to form the final output.
7. Semaphore - A semaphore controls access to a shared resource using a counter. If the counter is greater than zero, then access is allowed. If it is zero, then access is denied. What the counter is counting are permits that allow access to the shared resource. Thus, to access the resource, a thread must be granted a permit from the semaphore.
8. ThreadFactory - An object that creates new threads on demand. Using thread factories removes hardwiring of calls to new Thread, enabling applications to use special thread subclasses, priorities, etc.
9. BlockingQueue - BlockingQueue is a queue that additionally supports operations that wait for the queue to become non-empty when we are trying to retrieve an element and wait for the space to be empty when an element is to be inserted in the queue. Java 1.5 provides support for BlockingQueue interface along with other concurrent utility classes. All the implementations related to the BlockingQueue are thread safe. All the methods achieve their events using internal locks or other forms of concurrency control.
10. DelayQueue - DelayQueue implements the BlockingQueue interface. DelayQueue is a specialized Priority Queue that orders elements based on their delay time. It means that only those elements can be taken from the queue whose time has expired. DelayQueue head contains the element that has expired in the least time. If no delay has expired, then there is no head, and the poll will return null.
11. Lock - DelayQueue head contains the element that has expired in the least time. If no delay has expired, then there is no head, and the poll will return null. The lock() and unlock() methods of the interface can be called in different methods.
12. Phaser - Phaser’s primary purpose is to enable synchronization of threads that represent one or more phases of activity. It lets us define a synchronization object that waits until a specific phase has been completed. It then advances to the next phase until that phase concludes. It can also be used to synchronize a single phase, and in that regard, it acts much like a CyclicBarrier.