**Threads:**

**\*Difference between process and thread?**

A **process** is a program under execution i.e an active program. A **thread** is a lightweight **process** that can be managed independently by a scheduler. **Processes** require more time for context switching as they are more heavy. **Threads** require less time for context switching as they are lighter than **processes**.

**\*How to create thread?**

Extends **Thread** class. **Create** a **thread** by a new class that extends **Thread** class and **create** an instance of that class.

Implementing the Runnable Interface. The easiest way to **create** a **thread** is to **create** a class that implements the runnable interface.

**\*What is join () method in thread?**

**java.lang.Thread** class provides the join() method which allows one thread to wait until another thread completes its execution. If **t** is a Thread object whose thread is currently executing, then **t.join()** will make sure that **t** is terminated before the next instruction is executed by the program.

 **join():**It will put the current thread on wait until the thread on which it is called is dead. If thread is interrupted then it will throw InterruptedException.  
**Syntax:** public final void join()

 **join(long millis)**:It will put the current thread on wait until the thread on which it is called is dead or wait for specified time (milliseconds).  
**Syntax:** public final synchronized void join(long millis)

 **join(long millis, int nanos):**It will put the current thread on wait until the thread on which it is called is dead or wait for specified time (milliseconds + nanos).  
**Syntax:** public final synchronized void join(long millis, int nanos)

**\*Wait () vs. notify () and notify All ()?**

The **wait() method** causes the current thread to **wait** until another thread invokes the **notify()** or **notifyAll() methods** for that object. The **notify() method** wakes up a single thread that is **waiting** on that object's monitor. The **notifyAll() method** wakes up all threads that are **waiting** on that object's monitor.

**\*Wait () vs. sleep ()?**

**sleep()** is a method which is used to pause the process for few seconds or the time we want to. But in case of **wait()** method, thread goes in waiting state and it won’t come back automatically until we call the notify() or notifyAll().

The major difference is that wait() releases the lock or monitor while sleep() doesn’t releases the lock or monitor while waiting. wait() is used for inter-thread communication while sleep() is used to introduce pause on execution, generally.

**\*What is synchronization and how to achieve it?**

If you declare any method as synchronized, it is known as synchronized method. Synchronized method is used to lock an object for any shared resource. When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

**Java** provides a way of creating threads and **synchronizing** their task by using **synchronized** blocks. **Synchronized** blocks in **Java** are marked with the **synchronized** keyword. A **synchronized** block in **Java** is **synchronized** on some object

**\*What is false sharing?**

**False sharing** is when the memory cells are different but physically fall on the same cache line.

**\*Volatile keyword?**

**Volatile keyword** is used to modify the value of a variable by different threads. It is also used to make classes thread safe. It means that multiple threads can use a method and instance of the classes at the same time without any problem.

**\*Why wait methods is in object class?**

**wait** - **wait method** tells the current thread to give up monitor and go to sleep. ... That's one reason why these methods are in **Object class**. To reiterate threads **wait** on an **Object's** monitor (lock) and notify() is also called on an **object** to wake up a thread waiting on the **Object's** monitor.

**\*Read – write lock?**

**Read Lock** − If no thread has locked the ReadWriteLock for writing then multiple thread can access the **read lock**.

**Write Lock** − If no thread is reading or writing, then one thread can access the **write lock**.

**\*What is locking mechanism?**

A **locking mechanism** is a mechanical system which provides assistance to the coupling and uncoupling of two connectors and the fixation of the two parts in operating position.

A lock is a thread synchronization **mechanism** like synchronized blocks except locks can be more sophisticated than **Java's** synchronized blocks. ... locks contains several lock implementations, so you may not have to implement your own locks.

**\*How synchronization internally works?**

When we use a **synchronized block**, **internally** Java uses a monitor also known as monitor lock or intrinsic lock, to provide **synchronization**. These monitors are bound to an object, thus all **synchronized** blocks of the same object can have only one thread executing them at the same time.

**\*CountDown latch and cyclic barrier**

As stated in the definitions, **CyclicBarrier allows a number of threads to wait on each other**, whereas CountDownLatch allows one or more threads to wait for a number of tasks to complete. In short, CyclicBarrier maintains a count of threads whereas CountDownLatch maintains a count of tasks