**What does atomic mean in threads?**

**a series of operations on a variable can be completed without interference from other threads**.

**1. Synchronized**

* **Definition**: The synchronized keyword in Java is used to lock a block of code or method so only one thread can execute it at a time. It’s a basic mechanism for ensuring thread safety in Java, preventing concurrent access to critical parts of code.
* **Usage**: It’s primarily used to prevent multiple threads from accessing shared resources simultaneously.
*  **Usage**: It’s primarily used to prevent multiple threads from accessing shared resources simultaneously.
*  **Example**:
* public synchronized void increment() {
* // Only one thread can execute this method at a time.
* count++;
* }
* **Drawbacks**: The synchronized block is simple to use but lacks flexibility compared to more advanced locking mechanisms like ReentrantLock.

**2. Intrinsic Lock**

* **Definition**: An intrinsic lock (also known as a "monitor lock") is automatically associated with every object in Java. When a thread enters a synchronized block or method, it acquires the intrinsic lock for that object.
* **How It Works**: When a thread acquires an intrinsic lock on an object, no other thread can access synchronized blocks or methods on that object until the lock is released.
* **Scope**: Intrinsic locks are tied to objects, not specific code sections, so any synchronized block or method on the same object will share the same lock.

**3. Explicit Lock**

* **Definition**: An explicit lock is a more flexible and controllable locking mechanism, implemented with classes like ReentrantLock in Java’s java.util.concurrent.locks package.
* **Advantages**: Explicit locks provide features like timed lock acquisition (tryLock), interruptible locking, and fair ordering.

ReentrantLock lock = new ReentrantLock();

public void increment() {

lock.lock();

try {

count++;

} finally {

lock.unlock(); // Always release lock in a finally block

}

}

**4. Mutual Exclusion**

* **Definition**: Mutual exclusion (or "mutex") is a property ensuring that only one thread can access a critical section at a time.
* **Why It’s Important**: Mutual exclusion prevents data inconsistencies and race conditions by enforcing that shared resources are accessed by one thread at a time.

**5. Critical Section**

* **Definition**: A critical section is a part of the code where shared resources (e.g., variables, data structures) are accessed or modified.
* **Protection**: To prevent concurrent access, critical sections are protected by synchronization or explicit locks, allowing only one thread to enter at a time.

**6. Race Condition**

* **Definition**: A race condition occurs when multiple threads try to access and modify shared data concurrently, leading to unpredictable results.
* **Cause**: It often arises when threads depend on the timing of each other’s execution, without proper synchronization.
* **Prevention**: To prevent race conditions, synchronize access to shared resources or use thread-safe data structures.

**7. Lock**

* **Definition**: A lock is a mechanism that prevents multiple threads from accessing a shared resource simultaneously.
* **Types**:
  + **Intrinsic (Built-In)**: Automatically associated with objects in Java via the synchronized keyword.
  + **Explicit Lock**: Classes like ReentrantLock that provide advanced control over locking.
* **Purpose**: Locks are used to ensure thread safety and prevent data corruption by allowing only one thread to execute a critical section at a time.

**Summary**

* **Synchronized**: Simplified way to acquire an intrinsic lock for thread-safe code.
* **Intrinsic Lock**: Implicitly associated with every object and used in synchronized blocks/methods.
* **Explicit Lock**: More advanced, using ReentrantLock, offering greater flexibility and features.
* **Mutual Exclusion**: Ensures only one thread accesses critical resources at any time.
* **Critical Section**: Code section that requires synchronized access to avoid conflicts.
* **Race Condition**: An issue when multiple threads compete for shared data, leading to unexpected results.
* **Lock**: General mechanism to restrict concurrent access to shared resources.

Together, these concepts form the foundation for managing concurrency in Java, allowing developers to prevent data corruption and ensure reliable multi-threaded execution.