**Basic Level :-**

**1. Producer-Consumer Problem**

**Scenario:** You are building a messaging system where one thread produces messages and another consumes them.

**Challenge:** Implement thread-safe communication using wait() and notify().

**2. Print Even and Odd Numbers Using Two Threads**

**Scenario:** One thread prints even numbers, another prints odd numbers, in sequence.

**Challenge:** Use synchronization to alternate between threads.

**3. Deadlock Simulation**

**Scenario:** Two threads trying to acquire two locks in different orders.

**Challenge:** Simulate and then resolve a deadlock.

**4. Thread Pool Implementation**

**Scenario:** You need to process multiple tasks concurrently but want to limit the number of active threads.

**Challenge:** Use ExecutorService to manage a thread pool.

**5. Banking System – Account Transfer**

**Scenario:** Multiple threads transferring money between accounts.

**Challenge:** Ensure thread safety and avoid race conditions.

**6. File Downloader Simulation**

**Scenario:** Simulate downloading parts of a file using multiple threads.

**Challenge:** Merge parts correctly and handle thread completion.

**7. Real-Time Stock Price Updater**

**Scenario:** One thread updates stock prices, another displays them.

**Challenge:** Ensure data consistency using volatile or AtomicReference.

**Intermediate Level :-**

**1. Order Processing System**

**Scenario:** Multiple threads process orders from a shared queue. Each order must be processed exactly once.

**Concepts Tested:** Thread-safe queue, synchronization, BlockingQueue.

**Challenge:** Implement a producer-consumer model using LinkedBlockingQueue.

**2. Logging System with Multiple Writers**

**Scenario:** Multiple threads write logs to a single file.

**Concepts Tested:** File I/O with thread safety, synchronized blocks or ReentrantLock.

**Challenge:** Ensure logs are not interleaved or lost.

**3. Parallel Sum Calculator**

**Scenario:** Split a large array into chunks and calculate the sum using multiple threads.

**Concepts Tested:** Thread coordination, Callable, Future, ExecutorService.

**Challenge:** Aggregate results from multiple threads.

**4. Cyclic Barrier for Batch Processing**

**Scenario:** A batch of threads must complete a task before the next batch starts.

**Concepts Tested:** CyclicBarrier, coordination.

**Challenge:** Wait for all threads in a batch to finish before proceeding.

**5. Read-Write Lock for Shared Resource**

**Scenario:** Multiple threads read from a shared resource, but only one can write at a time.

**Concepts Tested:** ReentrantReadWriteLock.

**Challenge:** Maximize concurrency while ensuring data consistency.

**6. Thread-safe Cache with Expiry**

**Scenario:** Implement a cache where entries expire after a certain time.

**Concepts Tested:** ConcurrentHashMap, background thread for cleanup.

**Challenge:** Ensure thread-safe access and automatic cleanup.

**7. Task Scheduler**

**Scenario:** Schedule tasks to run after a delay or periodically.

**Concepts Tested:** ScheduledExecutorService.

**Challenge:** Handle task failures and retries.

**Advance Level :-**

**1. High-Performance Web Crawler**

**Scenario:** Build a multithreaded web crawler that fetches and parses pages concurrently, respects domain rate limits, and avoids duplicate URLs.

**Concepts Tested:**

* Thread pools (ExecutorService)
* Concurrent data structures (ConcurrentHashMap, BlockingQueue)
* Rate limiting
* URL deduplication

**2. Concurrent Banking System with Deadlock Prevention**

**Scenario:** Simulate multiple users transferring money between accounts. Ensure no deadlocks and maintain consistency.

**Concepts Tested:**

* Fine-grained locking
* Deadlock detection/prevention (e.g., lock ordering)
* ReentrantLock with tryLock()

**3. Custom Thread Pool Implementation**

**Scenario:** Implement your own thread pool with task queue, worker threads, and graceful shutdown.

**Concepts Tested:**

* Low-level thread management
* Blocking queues
* Thread lifecycle control

**4. Real-Time Analytics Dashboard**

**Scenario:** Multiple threads ingest data from sensors or logs and update a shared dashboard in real time.

**Concepts Tested:**

* ConcurrentHashMap, AtomicInteger, LongAdder
* Thread-safe UI updates or data aggregation
* Backpressure handling

**5. Multithreaded File Indexer**

**Scenario:** Recursively scan a directory structure and index file metadata using multiple threads.

**Concepts Tested:**

* Producer-consumer pattern
* Recursive task splitting
* ForkJoinPool or custom thread management

**6. Custom Read-Write Lock**

**Scenario:** Implement your own version of a read-write lock from scratch.

**Concepts Tested:**

* Low-level synchronization (synchronized, wait(), notifyAll())
* Fairness and starvation prevention
* Lock upgrading/downgrading

**7. Stress-Tested Order Matching Engine**

**Scenario:** Simulate a stock exchange order book where multiple threads place buy/sell orders.

**Concepts Tested:**

* High-concurrency data structures
* Lock-free algorithms
* Performance tuning and benchmarking

**8. Parallel Genetic Algorithm**

**Scenario:** Implement a genetic algorithm where fitness evaluation and mutation are done in parallel.

**Concepts Tested:**

* Parallel task execution
* Thread-safe random number generation
* ForkJoinPool or CompletableFuture