**Question: Real-Time Stock Trading Platform Simulation**

**Imagine you are developing a real-time stock trading platform that needs to handle multiple client requests for stock trades, account management, and market analysis, all happening concurrently. This platform will simulate the stock market, where each client can submit buy/sell orders, view account balance updates, and check live stock prices. Since the platform must manage many concurrent clients while ensuring data consistency and fast response times, you will need to implement various Java concurrency tools.**

**Using the following concurrency tools, create a solution that fulfills each requirement:**

**Requirements**

1. **ThreadGroup for Managing Threads by Type**
   * **Organize threads into specific groups based on their task:**
     + **"Buy Orders": handles client requests to buy stocks.**
     + **"Sell Orders": processes sell requests.**
     + **"Market Analysis": analyzes live data for stock trends.**
   * **The platform should be able to monitor or terminate all threads within a group if needed. For example, if the market hits a stop-loss threshold, terminate or pause all threads in the "Sell Orders" group to prevent further losses.**
2. **ThreadLocal for Client Session Management**
   * **Each client session has data specific to that session, such as sessionId, clientName, and accountBalance.**
   * **Use ThreadLocal to store and access each client’s data securely within their session, ensuring no data is accidentally shared or accessed by another client’s thread.**
3. **Locks and ReentrantLock for Shared Resource Management**
   * **Certain shared resources, such as a database for storing trade transactions or an in-memory record of stock prices, need controlled access.**
   * **Use Lock and ReentrantLock to manage access to these resources, ensuring that only one trade can be processed at a time per resource to avoid inconsistencies. For example, use ReentrantLock to allow for reentrant locking when calculating multiple updates for a single stock’s price without causing deadlock.**
4. **Thread Pools for Managing Client Requests**
   * **Implement a thread pool to handle trading requests, ensuring new buy/sell orders are queued and processed without overloading the system.**
   * **The thread pool should optimize resource allocation for high-priority trades while keeping lower-priority analysis or background tasks waiting if necessary.**
5. **Callable and Future for Asynchronous Stock Price and Account Balance Retrieval**
   * **Use Callable to execute tasks that return data, such as checking a stock’s current price or calculating a client’s updated balance after a trade.**
   * **Implement Future to track the progress and retrieve the result of these tasks, allowing clients to query whether their order has been executed or see their account balance after a trade.**
6. **Fork/Join Framework for Real-Time Stock Analytics**
   * **Implement a ForkJoinPool to split complex analysis tasks across multiple threads, such as calculating moving averages or trends for all available stocks.**
   * **Each stock’s price history should be processed in parallel, improving response time for analytics-heavy tasks that involve large datasets.**

**Task**

**Design a Java program that meets the requirements above. Your program should simulate the following workflow:**

1. **Initialization: Set up and initialize ThreadGroup, ThreadLocal, Locks, Thread Pool, Callable, Future, and Fork/Join as required.**
2. **Client Request Simulation: Each client session thread will perform a mix of buy/sell orders, account balance checks, and market analysis queries.**
3. **Concurrency Management: Use the appropriate concurrency controls to manage and synchronize client actions, ensuring data consistency and responsiveness of the platform.**
4. **Error Handling and Task Termination: Demonstrate how the platform handles abrupt changes (e.g., halting all "Sell Orders" if the market crashes) and ensure safe termination of threads where required.**

**Provide a code implementation to simulate the platform's functionality and use comments to explain how each part addresses the concurrency requirement.**