1. **Read the CSV File**: Read the CSV file containing the stock information (e.g., Symbol, Security Name, Price) and parse the data into your Java program. You can use libraries like Apache Commons CSV or Java's built-in FileReader and BufferedReader to accomplish this task.
2. **Store Stock Information**: Store the parsed stock information in a suitable data structure. Since you need to efficiently access and update stock prices and track bids, a suitable collection for this purpose could be a HashMap or ConcurrentHashMap. You can use the Symbol as the key and store the stock information (e.g., Security Name, Price) as the value.
3. **Server Implementation**:
   * Implement a server application that listens for incoming connections on port 2000.
   * Use multithreading to handle multiple client connections simultaneously. Each client connection should be handled in a separate thread to ensure concurrency.
   * Upon receiving a connection, the server should expect the first message to be the name of the client. Store this name for future reference.
   * Once the client provides the Symbol of the security they want to bid on, the server should check if the Symbol is valid by looking it up in the data structure you created in step 2. If the Symbol is found, the server should reply with the current price of the security; otherwise, it should reply with -1.
   * Implement functionality to update the price of stock items and track all changes to the stock items, including bids made by clients.
4. **GUI for Displaying Stock Prices**:
   * Implement a GUI to display the stock items (Symbol, Security Name) together with their current prices.
   * You can use Java's Swing or JavaFX libraries to create the GUI. Update the GUI periodically (e.g., every 500ms) to reflect changes in stock prices.
5. **Client Communication**:
   * Clients should be able to connect to the server using a common communication tool (e.g., nc, telnet).
   * Clients should provide their name and the Symbol of the security they want to bid on.
   * After receiving the current price of the security from the server, clients can make bids.
6. **Synchronization**:
   * Ensure proper synchronization when accessing and updating shared data structures (e.g., the collection storing stock information) to prevent data corruption in a multi-threaded environment.
7. **Testing**:
   * Test your server application with multiple clients to ensure it functions correctly under various scenarios (e.g., concurrent bids, updates to stock prices).

By following these steps, you can design and implement the auction server software system using threads, synchronization primitives, sockets, objects, classes, and collections as described in your project requirements.