Socket server:

- **Server:**
  - Listens on a predefined port for incoming client connections.
  - Utilizes a fixed thread pool to manage multiple client threads efficiently.
  - Implements stream-based I/O for handling client data transmission.
  - Incorporates a switch-case mechanism to process at least six different request types (e.g., data retrieval, data submission, status inquiries, etc.).
  - Ensures continuous operation, capable of handling client connections and requests indefinitely.
  - Maintains data persistence throughout its runtime, allowing data to be saved and accessed across various client interactions.
- **Client:**
  - Connects to the server using the specified IP address and port.
  - Sends various types of requests to the server based on user input or predefined actions.
  - Receives and processes responses from the server, displaying relevant information to the user.
  - Handles graceful termination by notifying the server upon completion or exit commands.

Thread Pool Implementation:

- Utilizes Java's `ExecutorService` with a fixed thread pool size to manage client threads.
- Ensures optimal performance by reusing threads for multiple client connections, reducing overhead associated with thread creation and destruction.

Stream-Based Communication:

- Employs `InputStream` and `OutputStream` for reading from and writing to client sockets.
- Uses `BufferedReader` and `PrintWriter` for efficient text-based data handling.
- Ensures data is correctly serialized and deserialized during transmission.

Switch-Case Request Handling:

- Defines a clear and organized structure for processing different client requests.
- Each case within the switch statement corresponds to a specific request type, executing relevant server-side operations.
- Facilitates easy expansion for additional request types in the future.

Data Management:

- Implements in-memory data structures (like `ArrayList`) to store and manage data during server runtime.

- Ensures thread-safe operations when accessing or modifying shared data to prevent race conditions.
- Provides mechanisms for data retrieval, insertion, updating, and deletion based on client requests.

Architecture and Design:

1. **Server Architecture:**
    - **Main Server Thread:**
        - Listens for incoming client connections on a specified port.
        - For each new connection, submits a `ClientHandler` task to the thread pool.
    - **ClientHandler Class:**
        - Implements `Runnable` or `Callable` to define the task for handling individual client interactions.
        - Manages communication streams with the connected client.
        - Processes client requests using a switch-case structure.
        - Ensures proper closure of client connections after request processing.
2. **Client Architecture:**
    - Establishes a socket connection to the server using the server's IP address and port.
    - Provides an interface (console-based or GUI) for users to send requests and receive responses.
    - Manages input/output streams for data transmission.
    - Handles user commands for sending different types of requests and terminating the connection.
3. **Data Flow:**
    - **Request Flow:**
        - Client sends a request to the server through the output stream.
        - Server reads the request via the input stream, processes it, and sends back a response.
        - Client receives the response and displays it to the user.
    - **Response Flow:**
        - Server processes the request based on the switch-case logic.
        - Executes the corresponding operation (e.g., data retrieval, storage).
        - Sends a response back to the client indicating the result of the operation.