ADVANCED REAL TIME CHAT APPLICATION

NAME – RN Umashankar

(Personal project Summary)

ABOUT THIS PROJECT

1.**Real-time Communication**: The application enables instant messaging between users through a client-server model.
2.**Java Sockets**: Utilizes Java's Socket and ServerSocket for creating the communication channel between clients and the server.
3.**Multi-threading**: Supports multiple clients by using multi-threading, allowing simultaneous interactions.
4.**Graphical User Interface (GUI)**: Provides a user-friendly interface using Java's Swing or JavaFX for seamless chat experience.
5.**Message Broadcasting**: Allows for individual or group message broadcasting between users connected to the server.
6.**Persistent Connection**: Maintains continuous communication with persistent socket connections for real-time messaging.
7.**User Authentication**: Ensures secure communication by implementing basic login and user authentication features.

METHODOLOGY

•**Server Setup (ServerSocket)**:
•A ServerSocket is created on the server side to listen for incoming client connections.
•Example: ServerSocket serverSocket = new ServerSocket(12345); listens on port 12345 for client connections.

•**Client Connection (Socket)**:
•Each client establishes a connection to the server using a Socket.
•Example: Socket socket = new Socket("localhost", 12345); connects the client to the server running on the same machine (localhost) at port 12345.

•Create **Multi-threading (Handling Multiple Clients)**:
•te a new thread for each client using Thread or ExecutorService for concurrent handling.
•Example: new Thread(new ClientHandler(clientSocket)).start();

•**Message Communication via Streams**:
•Use ObjectOutputStream and ObjectInputStream for sending and receiving messages.
•Example: ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

•**Broadcasting/Private Messaging**:
•Server sends messages either to all clients (broadcast) or to specific clients (private messages).

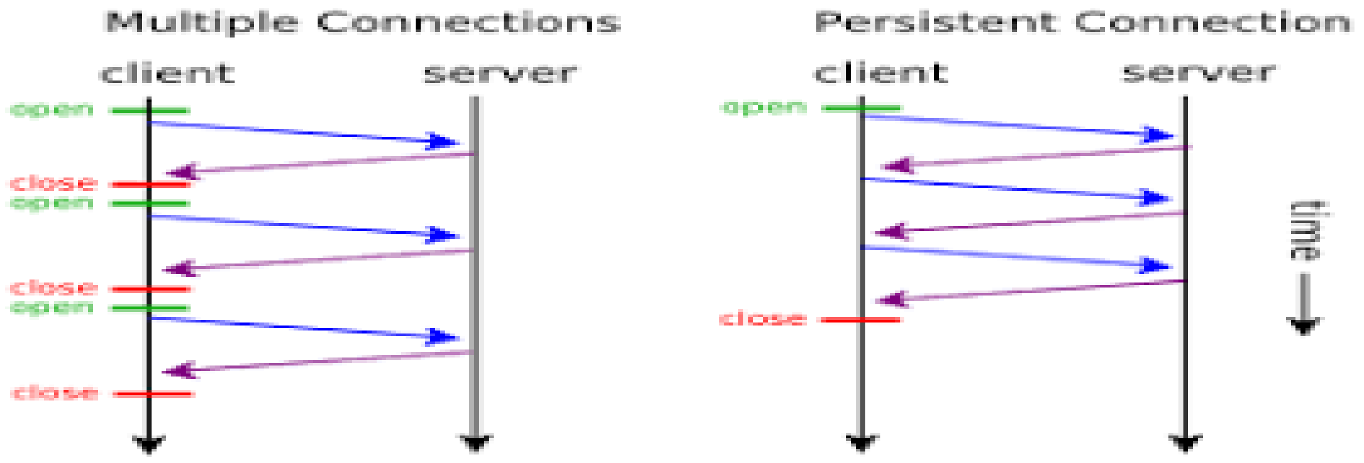
•Example: Loop over clients and send messages: client.sendMessage(message);

•**User Authentication**:
•Implement basic username/password validation before allowing communication.

•Example: Validate login credentials before accepting the client connection.

•**GUI Interface (Swing/JavaFX)**:
•Use Java Swing/JavaFX for client GUI with text fields for input and text areas for message display.
•Example: JTextArea chatArea = new JTextArea();

•**Persistent Connection (Continuous Communication)**:
•Keep the connection open for real-time chat without reconnection.
•Example: Use while (true) loop to listen for messages continuously.



1.Disconnection Handling:

- Properly close sockets and streams when a client disconnects.
- Example: socket.close(); and remove client from the list.

2.Exception Handling:

- Manage network or I/O errors using try-catch blocks.
- Example: catch (IOException e) { System.out.println("Error: " + e.getMessage()); }

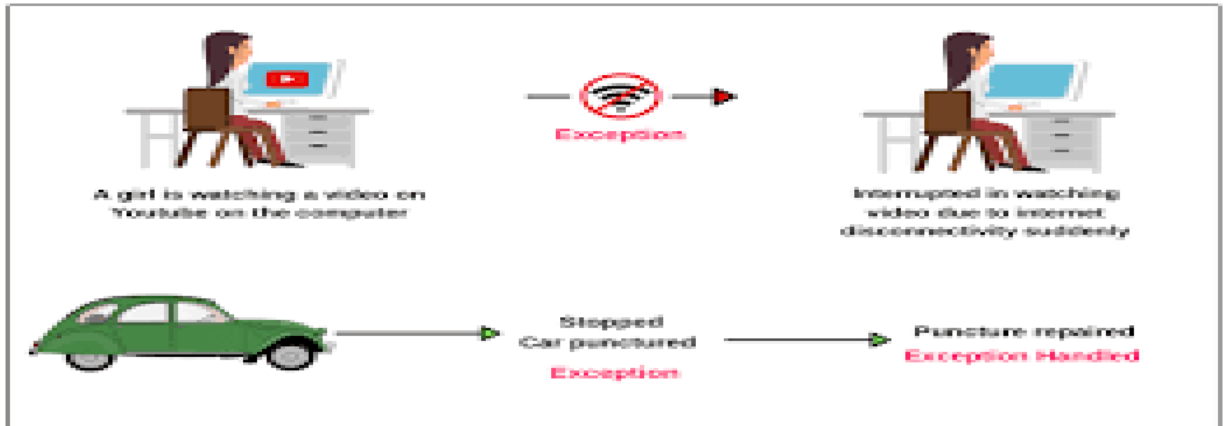

A girl is watching a video on Youtube on the computer
Exception
Interrupted in watching video due to internet disconnectivity suddenly
Stopped
Car punctured
Exception
Puncture repaired
Exception Handled

Fig: Realtime Example of Exception Handling