

TASK 2

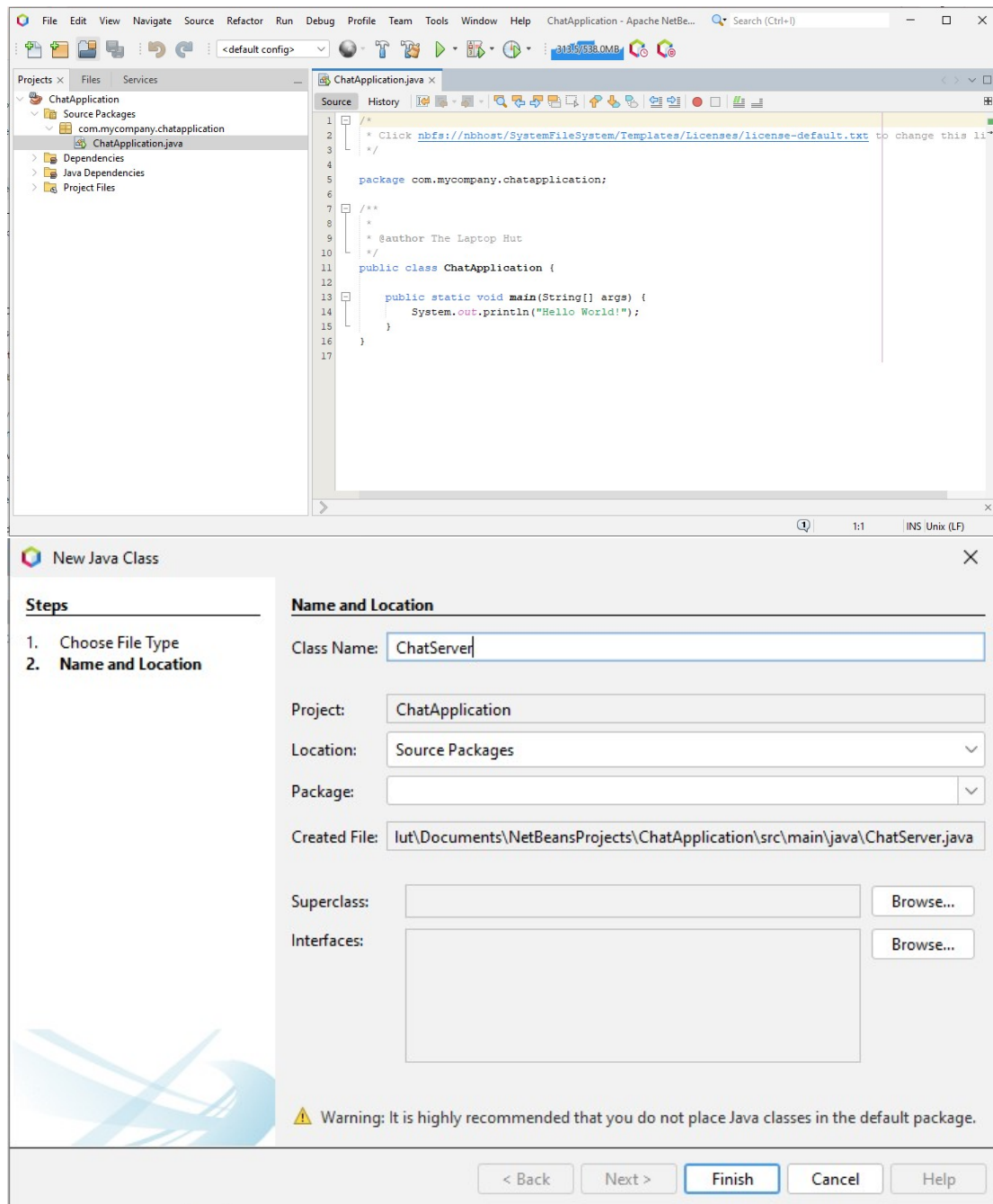
Develop a multithreaded chat application where multiple clients can connect to a server and exchange messages in real-time.

Steps of Solution

Create a new java project

The first screenshot shows the 'New Project' dialog. The 'Steps' pane on the left lists '1. Choose Project' and '2. ...'. The 'Choose Project' pane shows a tree of categories with 'Java with Maven' selected. To the right, a list of projects includes 'Java Application', 'Web Application', 'EJB Module', 'Enterprise Application', 'Enterprise Application Client', 'OSGi Bundle', 'Micronaut Project', 'NetBeans Module', 'NetBeans Application', 'Payara Micro Application', 'FXML JavaFX Maven Archetype', and 'Simple JavaFX Maven Archetype'. The 'Description' pane at the bottom states: 'A simple Java SE application using Maven. You are recommended to begin here, if you are new to Java!'. The bottom navigation bar contains buttons for '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

The second screenshot shows the 'New Java Application' dialog. The 'Steps' pane on the left lists '1. Choose Project' and '2. Name and Location'. The 'Name and Location' pane contains the following fields: 'Project Name' (ChatApplication), 'Project Location' (C:\Users\The Laptop Hut\Documents\NetBeansProjects) with a 'Browse...' button, 'Project Folder' (aptop Hut\Documents\NetBeansProjects\ChatApplication), 'Artifact Id' (ChatApplication), 'Group Id' (com.mycompany), 'Version' (1.0-SNAPSHOT), and 'Package' (com.mycompany.chatapplication) with an '(Optional)' label. The bottom navigation bar contains buttons for '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.



Server Code

```
import java.io.*;
import java.net.*;
import java.util.*;
```

```
public class ChatServer {
    // Set to hold client PrintWriter objects
    private static final Set<PrintWriter> clientWriters = new HashSet<>();
```

```
    public static void main(String[] args) {
        System.out.println("The chat server is running...");
        int port = 12345; // Choose your port number
```

```

try (ServerSocket serverSocket = new ServerSocket(port)) {
    while (true) {
        // Accept new client connections and handle them in a new thread
        new ClientHandler(serverSocket.accept()).start();
    }
} catch (IOException e) {
    System.err.println("Error starting server: " + e.getMessage());
}
}

private static class ClientHandler extends Thread {
    private final Socket socket;
    private PrintWriter out;
    private BufferedReader in;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try {
            // Initialize input and output streams
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new PrintWriter(socket.getOutputStream(), true);

            // Add the PrintWriter to the set of client writers
            synchronized (clientWriters) {
                clientWriters.add(out);
            }

            String message;
            while ((message = in.readLine()) != null) {
                System.out.println("Received: " + message);
                // Broadcast the message to all clients
                synchronized (clientWriters) {
                    for (PrintWriter writer : clientWriters) {
                        writer.println(message);
                    }
                }
            }
        } catch (IOException e) {
            System.err.println("Error handling client: " + e.getMessage());
        } finally {
            try {
                // Close the socket when done
                socket.close();
            } catch (IOException e) {
                System.err.println("Error closing socket: " + e.getMessage());
            }
            // Remove the client's PrintWriter from the set
            synchronized (clientWriters) {
                clientWriters.remove(out);
            }
        }
    }
}

```

```
}  
}  
}
```

Run the Server code

The screenshot displays the IDE interface for running a chat server application. The top panel shows the 'Output - Run (ChatApplication)' window with a 'BUILD SUCCESS' message and a total time of 7.110 s. The middle panel shows the 'Source' editor with the 'ChatServer.java' file open, displaying the server's main logic. The bottom panel shows the 'Output - Run (ChatServer)' window with a 'BUILD SUCCESS' message and a total time of 2.505 s.

```
Output - Run (ChatApplication) x
BUILD SUCCESS
Total time: 7.110 s
Finished at: 2024-07-29T21:36:05-07:00

Source
ChatApplication.java x ChatServer.java x
50 writer.println(message);
51 }
52 }
53 }
54 } catch (IOException e) {
55     System.err.println("Error handling client: " + e.getMessage());
56 } finally {
57     try {
58         // Close the socket when done
59         socket.close();
60     } catch (IOException e) {
61         System.err.println("Error closing socket: " + e.getMessage());
62     }
63     // Remove the client's PrintWriter from the set
64     synchronized (clientWriters) {
65         clientWriters.remove(out);
66     }
67 }

Output - Run (ChatServer) x
Skip non existing resourceDirectory C:\Users\Rut\Documents\NetBeansProjects\ChatApplication\src\main
--- compiler:3.11.0:compile (default-compile) @ ChatApplication ---
Nothing to compile - all classes are up to date
--- exec:3.1.0:exec (default-cgi) @ ChatApplication ---
The chat server is running...

Run (ChatServer) x
66:5 INS

Source
ChatApplication.java x ChatServer.java x
23 private static class ClientHandler extends Thread {
24     private final Socket socket;
25     private PrintWriter out;
26     private BufferedReader in;
27
28     public ClientHandler(Socket socket) {
29         this.socket = socket;
30     }
31
32     @Override
33     public void run() {
34         try {
35             // Initialize input and output streams
36             in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
37             out = new PrintWriter(socket.getOutputStream(), true);
38         } catch (IOException e) {
39             System.err.println("Error initializing streams: " + e.getMessage());
40         }
41     }
42 }
```

Chat Client 1

```
import java.io.*;
import java.net.*;

public class ChatClient {
    private final String serverAddress;
    private final int serverPort;

    public ChatClient(String address, int port) {
        this.serverAddress = address;
        this.serverPort = port;
    }

    public void start() {
        try (Socket socket = new Socket(serverAddress, serverPort);
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader consoleInput = new BufferedReader(new InputStreamReader(System.in))) {

            // Thread to read messages from server
            new Thread(() -> {
                try {
                    String serverMessage;
                    while ((serverMessage = in.readLine()) != null) {
                        System.out.println("Server: " + serverMessage);
                    }
                } catch (IOException e) {
                    System.err.println("Error reading from server: " + e.getMessage());
                }
            }).start();

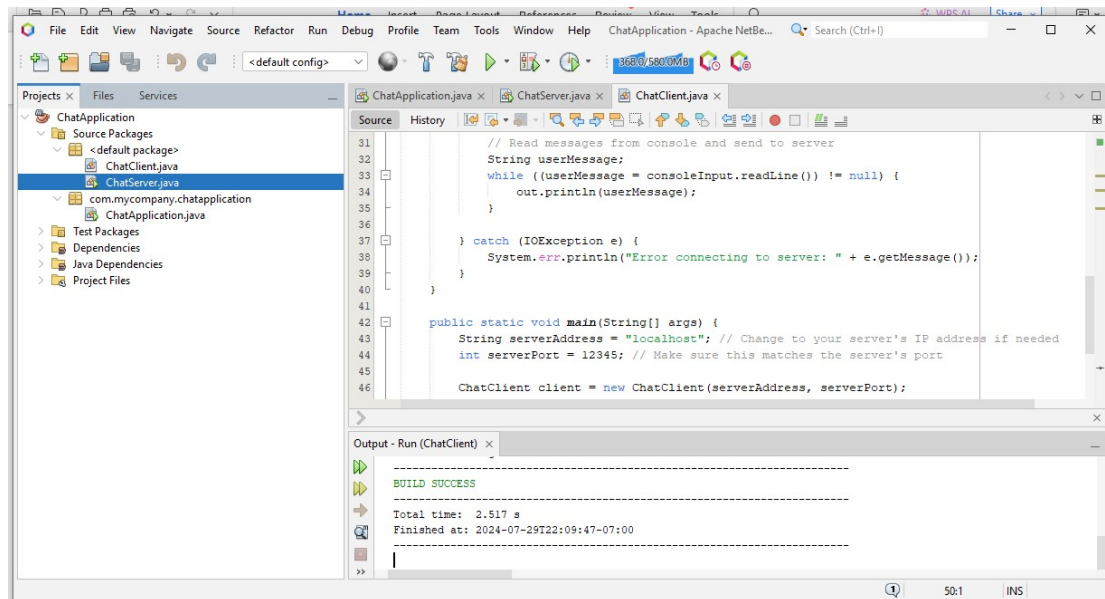
            // Read messages from console and send to server
            String userMessage;
            while ((userMessage = consoleInput.readLine()) != null) {
                out.println(userMessage);
            }

        } catch (IOException e) {
            System.err.println("Error connecting to server: " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        String serverAddress = "localhost"; // Change to your server's IP address if needed
        int serverPort = 12345; // Make sure this matches the server's port

        ChatClient client = new ChatClient(serverAddress, serverPort);
        client.start();
    }
}
```

Run ChatClient



Chat Client 2

```
import java.io.*;
import java.net.*;

public class ChatClient2 {
    private final String serverAddress;
    private final int serverPort;

    public ChatClient2(String address, int port) {
        this.serverAddress = address;
        this.serverPort = port;
    }

    public void start() {
        try (Socket socket = new Socket(serverAddress, serverPort);
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader consoleInput = new BufferedReader(new InputStreamReader(System.in))) {

            // Thread to read messages from server
            new Thread(() -> {
                try {
                    String serverMessage;
                    while ((serverMessage = in.readLine()) != null) {
                        System.out.println("Server: " + serverMessage);
                    }
                } catch (IOException e) {
                    System.err.println("Error reading from server: " + e.getMessage());
                }
            }).start();

            // Read messages from console and send to server
            String userMessage;
            while ((userMessage = consoleInput.readLine()) != null) {
                out.println(userMessage);
            }

        } catch (IOException e) {
            System.err.println("Error connecting to server: " + e.getMessage());
        }
    }
}
```

```

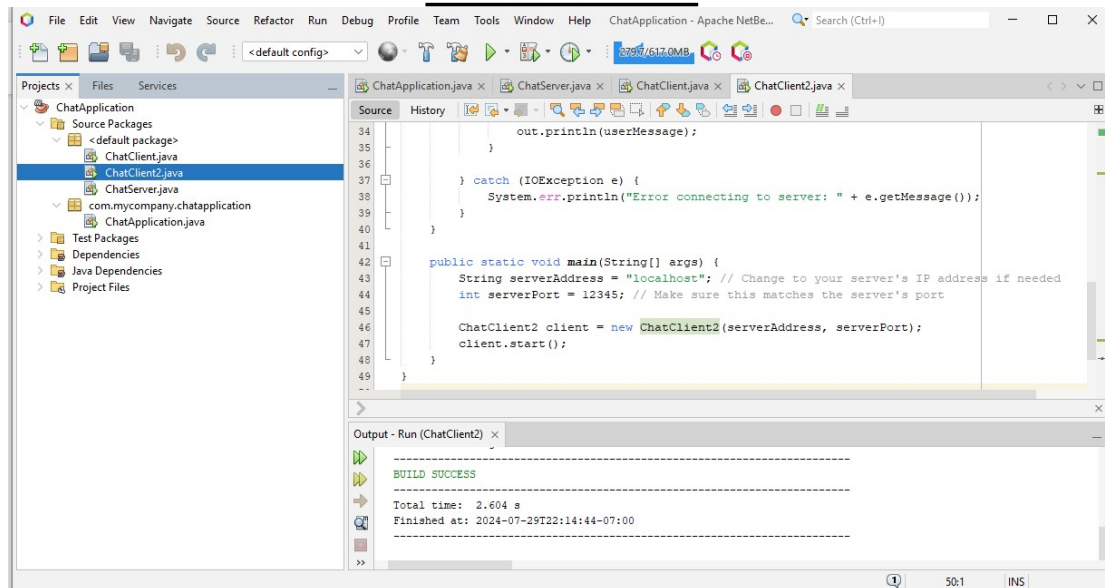
    }
}

public static void main(String[] args) {
    String serverAddress = "localhost"; // Change to your server's IP address if needed
    int serverPort = 12345; // Make sure this matches the server's port

    ChatClient2 client = new ChatClient2(serverAddress, serverPort);
    client.start();
}
}

```

Run ChatClient2



Successful Run both Clients and Server

