Socket Programming in Java

1. Introduction to Socket Programming

Socket programming enables communication between computers over a network. A socket is an endpoint for sending or receiving data. Java provides built-in libraries in the java.net package to support both TCP and UDP socket programming. This lecture focuses on TCP, which is reliable and connection-oriented.

2. Learning Objectives

By the end of this lecture, you will be able to:

- Understand how socket communication works.

- Write simple Java TCP client and server programs.

- Implement multithreaded servers to handle multiple clients.

- Explain the roles of key Java networking classes.

- Identify real-world applications of socket communication.

3. Java Networking Classes Used

- `ServerSocket`: Listens for incoming client connections.

- `Socket`: Used to create a connection between the client and server.

- `InputStream`, `OutputStream`: Read and write data.

- `BufferedReader`, `PrintWriter`: Simplify text-based communication.

4. Example 1: Basic TCP Server Program

This program listens on port 5000, accepts a client, and echoes received messages.

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

public class Server {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(5000);
        System.out.println("Server started. Listening on port 5000...");

        Socket clientSocket = serverSocket.accept();
```

```
        System.out.println("Client connected.");

        BufferedReader reader = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true);

        String message;
        while ((message = reader.readLine()) != null) {
            System.out.println("Received: " + message);
            writer.println("Echo: " + message);
        }

        clientSocket.close();
        serverSocket.close();
    }
}
```

Explanation:

- The server binds to port 5000.

- It waits for a client to connect using `accept()`.

- Reads text from the client and echoes it back using `PrintWriter`.

5. Example 2: Basic TCP Client Program

This client connects to the server at localhost:5000 and sends messages to it.

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

public class Client {
    public static void main(String[] args) throws IOException {
        Socket socket = new Socket("localhost", 5000);
        System.out.println("Connected to server.");

        PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
        BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        BufferedReader input = new BufferedReader(new InputStreamReader(System.in));

        String userInput;
        while ((userInput = input.readLine()) != null) {
            writer.println(userInput);
            System.out.println(reader.readLine());
```

```
        }

        socket.close();
    }
}
```

Explanation:

- The client connects to the server using `Socket`.

- It takes input from the keyboard, sends it to the server, and prints the server's response.

6. Key Concepts to Explain

Concept	Explanation
`accept()`	Waits for a client to connect
`readLine()`	Reads data sent from client
`PrintWriter/BufferedReader`	Used for reading/writing strings
`Socket`	Represents the client connection
`ServerSocket`	Waits for client requests
Ports	Communication channels (0–65535)

7. Real-World Applications

- Messaging Apps (WhatsApp, Messenger)
- File Transfer (FTP)
- Web Servers
- Online Multiplayer Games
- IoT Communication

8. Common Errors

Error	Cause
`Connection Refused`	Server not running or wrong port
`SocketException`	Broken connection
`BindException`	Port already in use

9. Assignment/Practice Ideas

- Build a TCP client-server chat app.
- Build a file transfer system.
- Use multi-threading for multiple client handling.

10. Summary

This lecture introduced basic socket communication using TCP in Java. You learned how to write simple client and server programs that exchange text data, laying the foundation for more complex networked applications like chat servers, file transfers, and multiplayer games.