**Day 1: Introduction to Java Socket Programming — Socket and ServerSocket Basics**

**🔷 What Is Socket Programming?**

**✅ Definition:**

**Socket programming** is a way for two computers (or programs) to **communicate with each other** over a **network** using code.

It helps one computer **send and receive messages** from another. In Java, we use **sockets** to build this connection.

**🔷 Real-Life Analogy:**

Imagine two people talking over walkie-talkies.

* One **speaks first** (Client)
* The other **listens and replies** (Server)

Socket programming works like this:

* The **client** starts the conversation by connecting.
* The **server** waits and responds.

**🔷 Java Classes Used in Socket Programming**

| **Java Class** | **Definition** |
| --- | --- |
| ServerSocket | A Java class that **waits for client connections** on a specific port. |
| Socket | A Java class that **connects to a server** using an IP address and port. |
| InputStream | Used to **read data** (messages) from the connection. |
| OutputStream | Used to **send data** (messages) over the connection. |

Today we’ll focus only on Socket and ServerSocket.

**🔷 Key Networking Terms (Beginner Notes)**

| **Term** | **Definition** |
| --- | --- |
| **IP Address** | Unique address of a computer on the internet (like a house address). |
| **Port** | A number that acts like a **door** into the computer (e.g., 1234). |
| **TCP** | A **reliable** protocol to send data. Makes sure data arrives safely. |
| **UDP** | A **faster**, but less reliable protocol. Not used in basic socket code. |

**🔷 Concept 1: ServerSocket – The Listener**

**✅ Definition:**

ServerSocket is used on the **server side** to wait for incoming connections.

🧠 **Think of it as a shopkeeper** waiting for a customer to walk in.

ServerSocket serverSocket = new ServerSocket(1234);

* This line **opens port 1234**
* The server will now wait for a client to connect

**🔷 Concept 2: Socket – The Connector**

**✅ Definition:**

Socket is used on the **client side** (or by the server after a connection is accepted) to **send and receive data**.

🧠 **Think of it as a customer** walking into a shop and talking to the shopkeeper.

Socket socket = new Socket("localhost", 1234);

* This line **connects to a server** running on the same machine (localhost) and port 1234

**🔷 ✅ Full Code Example: Simple Connection**

**🔹 1. Server Code**

// Server.java

import java.io.\*;

import java.net.\*;

public class Server {

public static void main(String[] args) throws IOException {

ServerSocket serverSocket = new ServerSocket(1234); // Step 1

System.out.println("Server started. Waiting for a client...");

Socket socket = serverSocket.accept(); // Step 2

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

socket.close(); // Step 3

serverSocket.close(); // Step 4

}

}

**🔸 Line-by-Line Explanation:**

1. ServerSocket serverSocket = new ServerSocket(1234);  
   ➤ Opens port 1234 and **starts listening** for clients.
2. Socket socket = serverSocket.accept();  
   ➤ **Waits** for a client to connect. This is a **blocking call** (the program stops here until a client connects).
3. socket.close();  
   ➤ Closes the client connection.
4. serverSocket.close();  
   ➤ Closes the server.

**🔹 2. Client Code**

// Client.java

import java.io.\*;

import java.net.\*;

public class Client {

public static void main(String[] args) throws IOException {

Socket socket = new Socket("localhost", 1234); // Step 1

System.out.println("Connected to the server.");

socket.close(); // Step 2

}

}

**🔸 Line-by-Line Explanation:**

1. Socket socket = new Socket("localhost", 1234);  
   ➤ Tries to connect to the server on the same machine (localhost) at port 1234.
2. socket.close();  
   ➤ Closes the connection.

**⚠️ Common Pitfalls**

| **Issue** | **What Happens** |
| --- | --- |
| Running client first | Client can’t find server → connection error |
| Using busy port | Java will say “port already in use” |
| Not closing socket | May cause memory or network issues |
| Server not started | Client will throw ConnectionRefusedException |

**✅ Best Practices**

* **Always run the server first**, then the client
* Use try-catch blocks to handle errors (we’ll do this in later days)
* Always **close your sockets** to free system resources
* Keep port numbers above 1024 (below that are often reserved)

**🏋️ Practice Task – Try This Today:**

✍️ Create two Java files: Server.java and Client.java using the code above.  
Then:

✅ Step 1: Compile and run Server.java  
✅ Step 2: Then run Client.java  
✅ Step 3: Try changing the port from 1234 to 5678  
✅ Step 4: Try running the client **before** the server — note the error message.