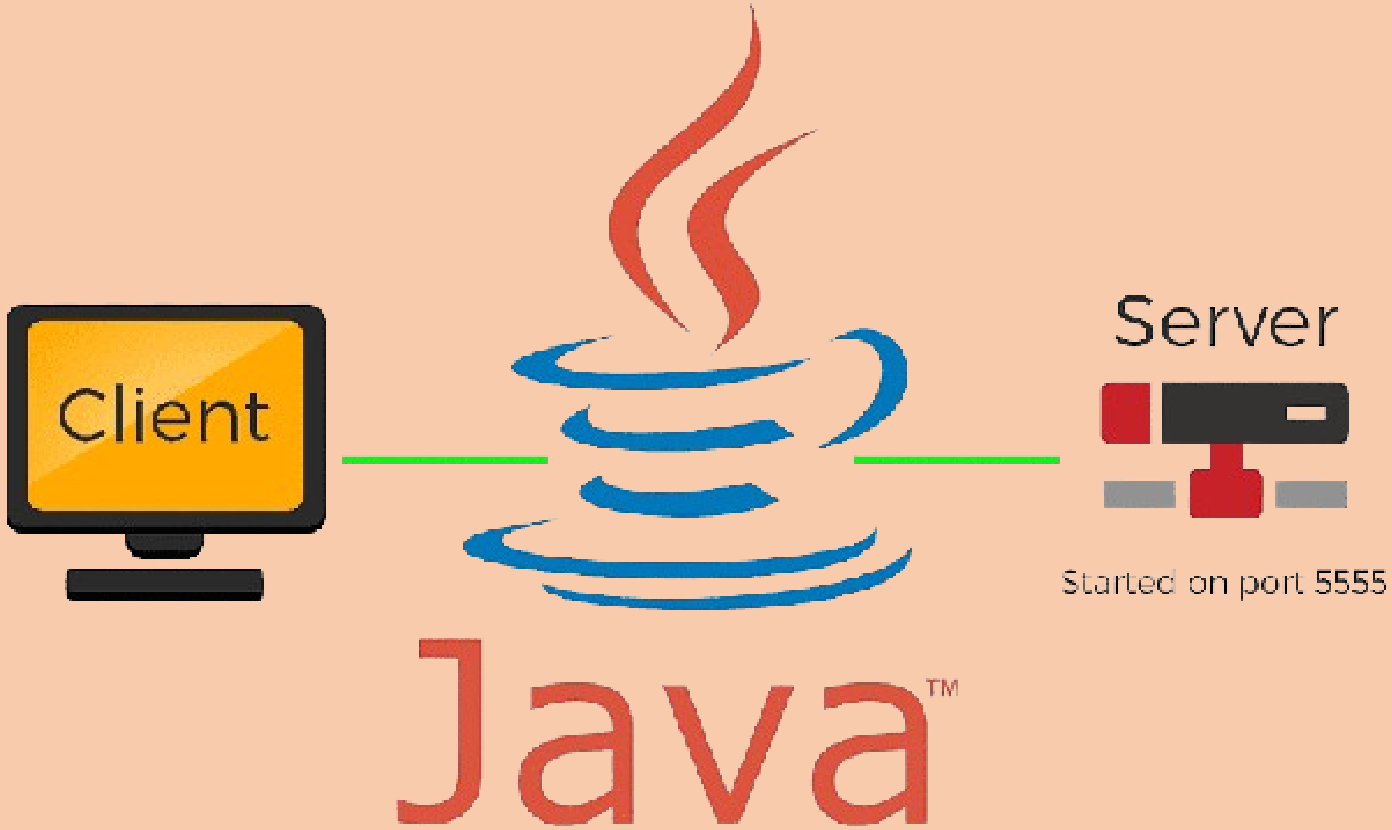
AP Lesson

Topic: socket

Instructor: Dr.Vahidi

by Sahar Shahrajabian

Fall 1403


Client
Server
Started on port 5555
Java™

What is socket

A **socket** is a software endpoint that facilitates communication between two devices over a network.

It allows programs to send and receive data, enabling network communication, typically over TCP (Transmission Control Protocol) or UDP (User Datagram Protocol).

How Socket Pairing Works:

1. Server Side (Passive Socket):

- The server creates a socket and binds it to a specific **IP address** and **port** using the “**bind()”** function.
- The server listens for incoming connections using the “**listen()”** function.
- When a client tries to connect, the server accepts the connection using the “**accept()”** function, which creates a new socket specifically for this connection. This new socket is paired with the client’s socket.

2. Client Side (Active Socket):

- The client creates a socket and connects to the server using the server's IP address and port via the “**connect()”** function.
- This action initiates the pairing process with the server.

3. Establishing a Connection (TCP):

- A **TCP connection** involves a **three-way handshake**:
  1. **SYN** (Synchronize): The client sends a SYN packet to the server, requesting to open a connection.
  2. **SYN-ACK** (Synchronize-Acknowledge): The server acknowledges the request and responds with a SYN-ACK packet.
  3. **ACK** (Acknowledge): The client sends an ACK packet, confirming the connection.

4. Communication:

- The server and client can now send and receive data using their respective sockets.

TCP (Transmission Control Protocol)

- **Connection-Oriented**: Establishes a connection between client and server before transmitting data.
- **Reliable**: Guarantees that data will be delivered in the correct order and without errors.
  If a packet is lost, it is retransmitted.
- **Error Checking**: Includes mechanisms for error detection and correction.
- **Flow Control**: Ensures that the sender doesn't overwhelm the receiver with too much data at once.
- **Use Cases**: Web browsing (HTTP/HTTPS), file transfers (FTP), email (SMTP),
  and other services where reliability is crucial.

UDP (User Datagram Protocol)

- **Connectionless**: No formal connection is established between the client and server.
- **Unreliable**: There is no guarantee that packets will arrive in order, or even arrive at all. Lost packets are not retransmitted.
- **Faster**: Since there’s no overhead for managing connections, UDP is faster but at the cost of reliability.
- **No Flow Control**: Data is sent regardless of the receiver’s ability to handle it.
- **Use Cases**: Online gaming, live video streaming, VoIP, real-time communications where speed is more critical than reliability.

Differences between TCP and UDP

Feature	TCP	UDP
Connection Type	Connection-Oriented (requires a connection)	Connectionless (no connection needed)
Reliability	Reliable, ensures delivery and order	Unreliable, no guarantees of delivery or order
Speed	Slower due to overhead of ensuring reliability	Faster, minimal overhead
Error Handling	Performs error checking and correction	Error checking is optional, no correction
Data Transmission	Stream-based (continuous flow of data)	Packet-based (individual, discrete packets)
Use Cases	Web browsing, file transfers, emails	Online games, live streaming, VoIP, real-time apps

Types of sockets

Stream Sockets (TCP Sockets)

- **Protocol**: TCP (Transmission Control Protocol)
- **Connection-Oriented**: A connection must be established between the client and server before data can be transmitted. This connection remains open until explicitly closed by either side.
- **Reliability**: Ensures reliable, ordered, and error-checked delivery of data. If a packet is lost, it will be retransmitted.
- **Use Case**: Web browsing (HTTP/HTTPS), file transfers (FTP), emails (SMTP), and any situation where reliable, continuous communication is essential.
- **Data Transmission**: Continuous, like a stream of bytes. The data arrives in the same order it was sent.

Types of sockets

Datagram Sockets (UDP Sockets)

- **Protocol**: UDP (User Datagram Protocol)

- **Connectionless**: No connection is established between client and server. Data is sent as independent packets (datagrams) without guaranteeing arrival, order, or integrity.

- **Speed**: Faster than TCP because there's no overhead for establishing a connection, retransmitting lost packets, or ensuring data integrity.

- **Use Case**: Applications where speed is more critical than reliability, like online gaming, video streaming, voice over IP (VoIP), and real-time communications.

- **Data Transmission**: Data is sent in discrete chunks (datagrams), which may arrive out of order, be duplicated, or get lost entirely.

Types of sockets

Raw Sockets

- **Protocol**: Typically, allows direct sending and receiving of packets at the network layer (IP).

- **Direct Access**: Provides access to the underlying network protocols. It’s used for sending custom IP packets or for protocols that aren't supported by the operating system's socket API.

- **Use Case**: Network monitoring tools (like ping, traceroute), custom protocols, or low-level network programming.

- **Privileges**: Often requires administrative or root privileges to create a raw socket.

Types of sockets

Sequenced Packet Sockets (SCTP Sockets)

- **Protocol**: SCTP (Stream Control Transmission Protocol)

- **Connection-Oriented**: Like TCP, it establishes a connection before data transfer,
  but it supports multi-streaming (multiple streams of data in a single connection).

- **Reliability**: Ensures reliable, ordered delivery of data,
  but also supports sending partial messages and unordered messages within a stream.

- **Use Case**: Telecommunications, messaging systems, and applications that need multi-stream,
  reliable delivery of messages.

- **Data Transmission**: Message-oriented, and supports multi-streaming.

Types of sockets

Unix Domain Sockets

- **Protocol**: Used for inter-process communication (IPC) within the same host machine, without involving the network stack.

- **Local Communication**: Communicates between processes running on the same machine, offering lower latency and more efficient communication than network sockets.

- **Use Case**: Local applications that need to exchange data between processes, like system services or local daemons.

Types of sockets

Bluetooth Sockets

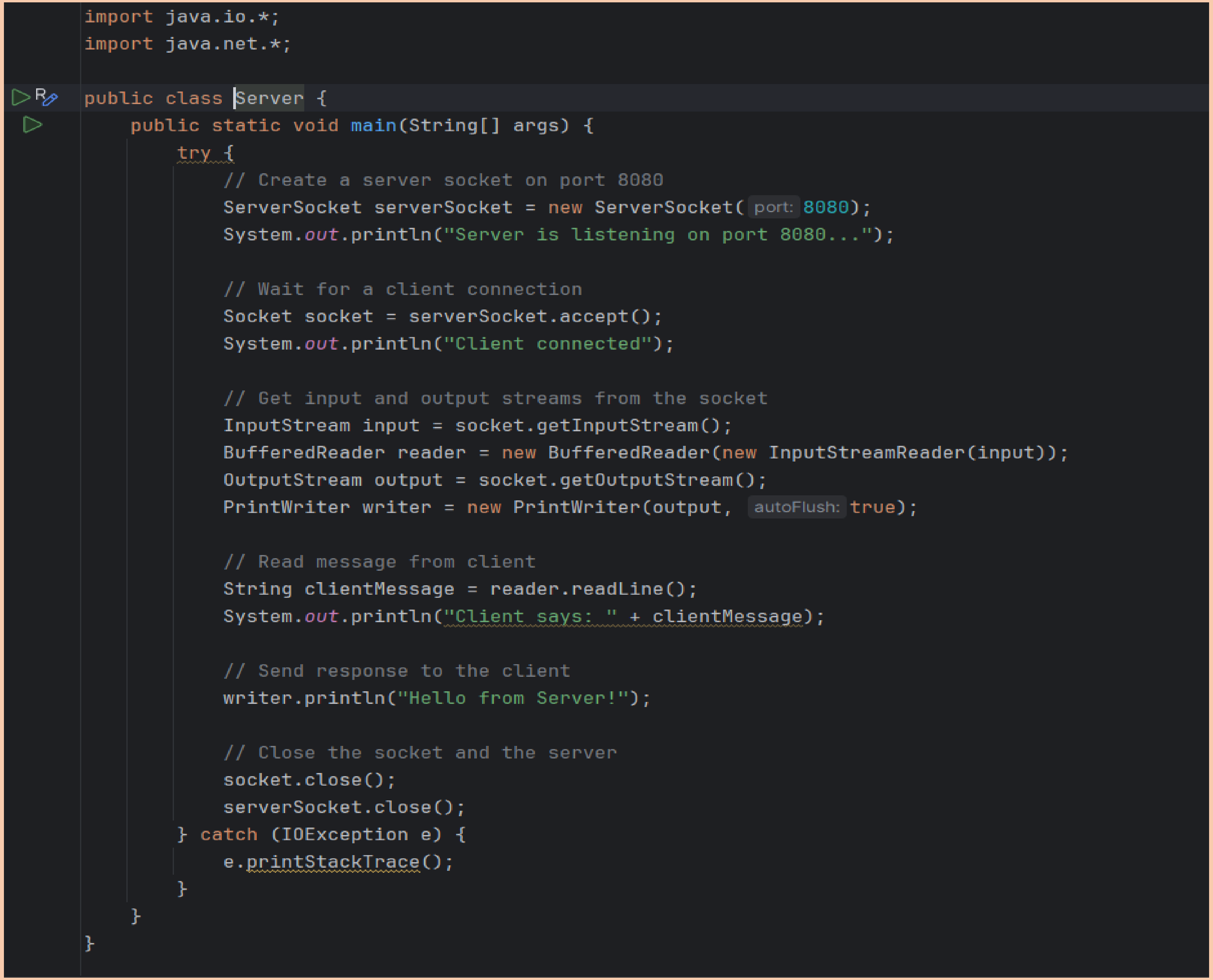
- **Protocol**: L2CAP (Logical Link Control and Adaptation Protocol)
- **Connection-Oriented/Connectionless**: Bluetooth supports both reliable, connection-oriented and connectionless communication, depending on the application.
- **Use Case**: Wireless communication between devices, such as connecting headphones to a smartphone, or sending files between devices.

Types of sockets

Multicast Sockets

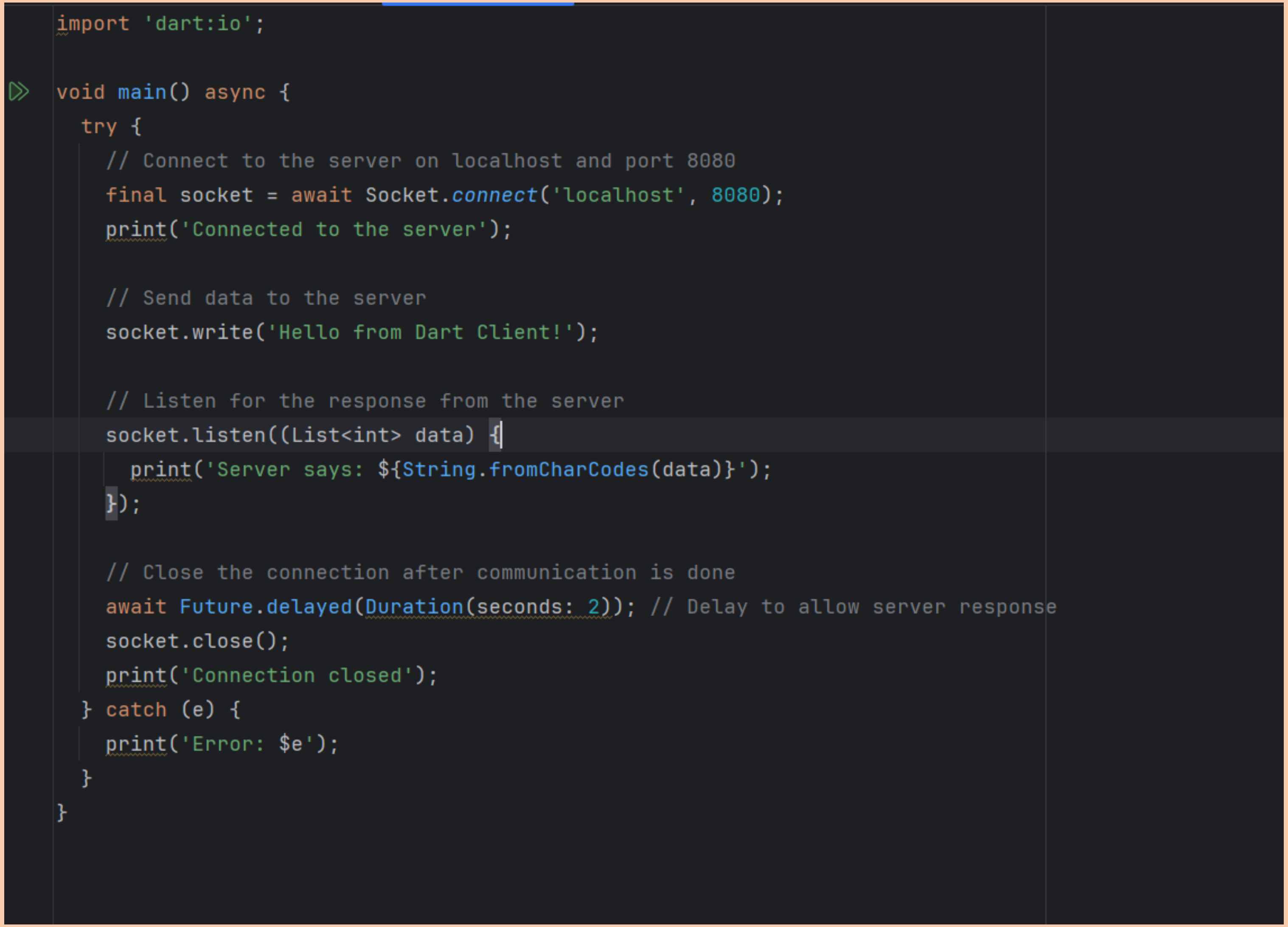
- **Protocol**: Typically uses UDP for sending the same data to multiple recipients (multicasting).
- **Broadcasting**: Multicast sockets are used to broadcast data to multiple clients simultaneously over UDP, where a single packet is sent to multiple receivers.
- **Use Case**: Streaming video or audio to multiple clients, online radio, or real-time stock quote systems.

Java server code



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

public class Server {
    public static void main(String[] args) {
        try {
            // Create a server socket on port 8080
            ServerSocket serverSocket = new ServerSocket( port: 8080);
            System.out.println("Server is listening on port 8080...");

            // Wait for a client connection
            Socket socket = serverSocket.accept();
            System.out.println("Client connected");

            // Get input and output streams from the socket
            InputStream input = socket.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(input));
            OutputStream output = socket.getOutputStream();
            PrintWriter writer = new PrintWriter(output,  autoFlush: true);

            // Read message from client
            String clientMessage = reader.readLine();
            System.out.println("Client says: " + clientMessage);

            // Send response to the client
            writer.println("Hello from Server!");

            // Close the socket and the server
            socket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Client socket code in dart



```
import 'dart:io';

void main() async {
  try {
    // Connect to the server on localhost and port 8080
    final socket = await Socket.connect('localhost', 8080);
    print('Connected to the server');

    // Send data to the server
    socket.write('Hello from Dart Client!');

    // Listen for the response from the server
    socket.listen((List<int> data) {
      print('Server says: ${String.fromCharCodes(data)}');
    });

    // Close the connection after communication is done
    await Future.delayed(Duration(seconds: 2)); // Delay to allow server response
    socket.close();
    print('Connection closed');
  } catch (e) {
    print('Error: $e');
  }
}
```

Read more on

https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

https://www.codejava.net/java-se/networking/java-socket-client-examples-tcp-ip