Java Sockets

☑ Favorite	☐
↗ Tag	Java Network Programming

▼ Notes

- Sockets allow the programmer to treat a network connection as just another stream onto which bytes can be written and from which bytes can be read.

▼ What is the difference between Socket and ServerSocket

Socket → Used for client-side programming. Connect to server socket and exchanges data

ServerSocket → Used by servers to listen for incoming connections from clients. Waits for client requests to arrive and establishes a connection when the request is received. Once the connection is received it returns a regular Socket object that represents the connection between client an server

▼ Fundamentals

1. **Socket Basics:**
    - A socket is one endpoint of a communication link between two programs running on a network.
    - Sockets are bound to port numbers, allowing the TCP layer to identify the application to which data is destined.
    - Sockets can perform seven basic operations:
        - connect to a remote machine
        - send data
        - receive data
        - close a connection
        - bind to a port

    - listen for incoming data
    - accept connections from remote machines on the bound port.

2. **Socket Class in Java:**
    - Java's `Socket` class, part of the `java.net.*` package, sits on top of a platform-dependent implementation, abstracting system details.
    - Both clients and servers use the `Socket` class. Clients create socket objects, attempt to connect to remote hosts, establish full-duplex connections, and send/receive data through input and output streams.
    - Servers use the `ServerSocket` class, which allows them to listen for incoming connection attempts on a specified port, accept connections from clients, and interact with clients using input and output streams until the connection is closed.
3. **Server Life Cycle:**
    - The server's basic life cycle involves creating a `ServerSocket` on a specific port, listening for incoming connections using the `accept()` method, and accepting connections from clients, which returns a `Socket` object.
    - Once connected, the server communicates with the client based on an agreed-upon protocol, typically involving sending and receiving data until it's time to close the connection.
    - After the connection is closed, the server returns to listening for new incoming connections.

▼ TCP Server

```
package socket;

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

public class Server {
```

```
    public static void main(String[] args) //throws Exceptio
    {
        try
        {
            ServerSocket serverSocket = new ServerSocket(909
            System.out.println("wainting for clients...");
            boolean stop = false;
            //while(!stop)

            Socket socket = serverSocket.accept();
            PrintWriter out = new PrintWriter(socket.getOutpu
            out.println("Hello  client!");
            BufferedReader input = new BufferedReader(new Inp
            String clientInput = input.readLine();
            System.out.println(clientInput);
            input.close();
            out.close();
            socket.close();
            serverSocket.close();

        } catch (Exception e)
        {
            System.out.println(e.toString());
        }
    }

}
```

- `Socket socket = serverSocket.accept();`: This line blocks the program until a client connects to the server. Once a connection is established, it returns a Socket object representing the connection.
- `PrintWriter out = new PrintWriter(socket.getOutputStream(), true);`: This line creates a PrintWriter object that writes to the output stream of the socket.

It will be used to send messages to the client.

  - `socket.getOutputStream(), true)`: This part of the line specifies the OutputStream as the destination for the output data and sets the second argument to `true`. The second argument in the PrintWriter constructor enables auto-flushing, which means that the output buffer will be automatically flushed whenever a println, printf, or format method is called on the PrintWriter object. This ensures that data is sent to the OutputStream immediately rather than being buffered.

- `BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));`: This line creates a BufferedReader object that reads from the input stream of the socket. It will be used to receive messages from the client.
  - Sets up a mechanism for reading data from the input stream of the socket
  - `new InputStreamReader(socket.getInputStream())`: This part constructs an InputStreamReader object, which serves as a bridge between byte-oriented streams (like the one obtained from `socket.getInputStream()`) and character-oriented streams. It reads bytes and decodes them into characters using a specified charset or the platform's default charset if none is specified.
    - utf-8 is usually the default