SOCKET PROGRAMMING ASSIGNMENT

1. What is socket programming?

Socket Programming in Java refers to the technique used to enable communication between two machines over a network. It allows data exchange between client and server applications using **TCP/IP** or **UDP** protocols.

- **Client**: Opens a socket and connects to the server.
- **Server**: Listens on a port for incoming client connections.

2. Explain the difference between TCP and UDP protocols. How does it relate to socket programming?

TCP	UDP
Connection-oriented: Requires a connection before data transfer.	**Connectionless**: No connection is established before sending data.
Reliable: Guarantees delivery of data in order.	**Unreliable**: No guarantee of delivery or order.
Flow Control: Manages data flow to avoid congestion.	**No Flow Control**: Does not manage data flow.
Slower: Due to overhead of error checking and flow control.	**Faster**: Less overhead, ideal for real-time applications.

Relation to Socket Programming in Java:

- **TCP**: Uses Socket class for reliable, connection-oriented communication (e.g., new Socket()).
- **UDP**: Uses DatagramSocket class for faster, connectionless communication (e.g., new DatagramSocket()).

3. In Java, what is the purpose of the Socket class?

he **Socket** class in Java is used to create a client-side socket that allows communication between a client and a server over a network using the **TCP protocol**.

Eg. Socket socket = new Socket("localhost", 8080); // Connects to the server at localhost on port 8080.

4. What is a server socket in Java? How is it different from a regular socket?

A **ServerSocket** in Java is used to create a server-side socket that listens for incoming client connections on a specific port. It is used to accept client requests.

ServerSocket listens for incoming connections and Uses the accept() method to establish a connection with a client where as **Socket** connects to the server and Establishes a connection after the server socket accepts it.

5. Explain the role of the ServerSocket class in socket programming.

The **ServerSocket** class in Java is used for **server-side** communication in socket programming. Its main role is to listen for incoming client connections on a specific port. **ServerSocket** listens for connections, while **Socket** is used to establish communication after a connection is accepted.

Eg.
ServerSocket serverSocket = new ServerSocket(8080); // Listen on port 8080
Socket clientSocket = serverSocket.accept(); // Accepts a client connection

6. What is the purpose of the InputStream and OutputStream in socket programming?

In socket programming, **InputStream** and **OutputStream** are used to read and write data through a socket connection. **InputStream** reads data from the socket. **OutputStream** writes data to the socket.

Eg.
OutputStream out = socket.getOutputStream();
out.write(data);

InputStream in = socket.getInputStream();
int data = in.read();

7. How does a server in socket programming distinguish different clients?

In socket programming, a server distinguishes different clients by creating a **new Socket** object for each client connection. Each connection is assigned a unique **IP address** and **port number**, which can be accessed using the getInetAddress() and getPort() methods of the Socket class. In Short The server uses **IP address** and **port number** to distinguish between different clients.

Eg.
ServerSocket serverSocket = new ServerSocket(8080);
Socket clientSocket = serverSocket.accept(); // Accept a new client connection
InetAddress clientAddress = clientSocket.getInetAddress(); // Get client IP
int clientPort = clientSocket.getPort(); // Get client port number

8. What is the significance of the accept() method in the ServerSocket class?

The **accept()** method in the **ServerSocket** class is used to **accept incoming client connections**. It blocks the execution until a client establishes a connection, then returns a **Socket** object representing the connection between the server and the client.

9. Explain the terms "blocking" and "non-blocking" in the context of socket programming.

In socket programming, **blocking** and **non-blocking** refer to whether a method call waits for a specific condition to be met before proceeding.
A blocking method **waits** for an event (like a client connection or data arrival) to occur before it returns control to the program. A non-blocking method **does not wait** and immediately returns control, even if the event hasn't occurred yet.

10. How do you handle exceptions in socket programming in Java? Provide examples of common exceptions.

In socket programming, exceptions are handled using **try-catch blocks**. Common exceptions include:

1. **IOException**: Occurs during input/output operations, like reading or writing data.
2. **UnknownHostException**: Happens when the server's IP address is unknown.
3. **SocketException**: Raised when there's an error in socket creation or communication.

Eg.

```
try {
    Socket socket = new Socket("localhost", 8080); // May throw UnknownHostException
    InputStream in = socket.getInputStream();      // May throw IOException
} catch (UnknownHostException e) {
    System.out.println("Host not found: " + e.getMessage());
} catch (IOException e) {
    System.out.println("I/O error: " + e.getMessage());
}
```

11. What is the role of the PrintWriter class in socket programming, and how is it used?

The **PrintWriter** class in socket programming is used to write formatted text (such as strings) to an output stream. It is typically used to send data from the client to the server or vice versa through a socket connection.

Eg.

```
Socket socket = new Socket("localhost", 8080);
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
out.println("Hello, Server!"); // Sends data to the server
```

12. Explain the concept of a port number in socket programming. Why is it necessary?

In socket programming, a **port number** is a 16-bit integer used to uniquely identify a specific service or application on a host within a network. It ensures that data sent to a specific IP address is directed to the correct application or process.

Port numbers are used to direct network traffic to the appropriate service on a machine. Distinguishes services and Allows multiple connections.

13. How can you ensure proper communication between a Java server and client if they are on different machines?

To ensure proper communication between a Java server and client on different machines, follow these steps:

1. **Use the Correct IP Address**: Ensure the client connects to the **server's external IP address** (not localhost).
2. **Open the Server Port**: The server's port (e.g., 8080) must be open and accessible via the firewall on both machines.

3. **Ensure Network Connectivity**: Both machines should be on the same network or have proper routing (e.g., via VPN, public IP).
4. **ServerSocket Binding**: Bind the server to the correct network interface and port.

Example:

Socket socket = new Socket("server_ip_address", 8080); // Use the server's external IP

14. Discuss the steps involved in creating a simple client-server application in Java using sockets.

Creates ServerSocket, listens on a port, accepts client connections, exchanges data.

Create Server (ServerSocket):

- **Initialize ServerSocket** to listen for incoming connections.
- **Accept client connections** using accept() method.
- **Read/write data** from/to client using InputStream/OutputStream.

Creates Socket, connects to server, sends/receives data.

Create Client (Socket):

- **Initialize Socket** to connect to server using the server's IP and port.
- **Send/receive data** using OutputStream/InputStream.

15. What is the purpose of the close() method in socket programming? Why is it important to close sockets properly?

The `close()` method in socket programming is used to **close the socket connection**, releasing the resources associated with it. The purpose is to **Closes input/output streams** linked to the socket and **Releases the network resources** (e.g., port number) used by the socket.
Proper Close is required as it **Prevents resource leakage and Allows the port to be reused.**

16. Describe the differences between synchronous and asynchronous socket communication.

In Synchronous Socket Communication The application waits for the completion of each operation before proceeding. Operations like reading and writing block the thread until completed.
In **Asynchronous Socket Communication**: The application does not wait and can continue execution while waiting for operations to complete. It **Requires callbacks or event-driven mechanisms.** It uses features like **Selectors** or external libraries (e.g., **NIO**) to handle multiple connections.

17. How can you handle multiple client connections in a Java server using socket programming?

To handle multiple client connections in a Java server using socket programming, use **multi-threading**. Each client connection is handled in a separate thread, allowing the server to manage multiple clients concurrently.

18. Explain the purpose of the BufferedReader and BufferedWriter classes in socket programming.

BufferedReader: Reads text data from an input stream (e.g., from a socket) more efficiently by buffering characters. Used for reading text data line-by-line or character-by-character.

BufferedWriter: Writes text data to an output stream (e.g., to a socket) more efficiently by buffering characters before writing them to the stream. Used for sending data to the client/server in a buffered manner.

Eg.

```
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
String message = in.readLine();
BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
out.write("Message from server");
```

19. Discuss the advantages and disadvantages of using sockets for communication in a distributed system.

Advantages of Using Sockets in Distributed Systems:

- **Direct Communication**: Allows direct communication between applications.
- **Flexibility**: Supports both **TCP** and **UDP** protocols.
- **Platform Independence**: Java's socket API is platform-independent, allowing communication across different systems and networks.
- **Customization**: Provides full control over communication.

Disadvantages of Using Sockets:

- **Complexity**: Requires managing network issues.
- **Scalability**: Handling multiple clients can become challenging and resource-intensive without proper management.
- **Security**: Sockets alone don't provide built-in security.
- **Error-prone**: Network failures, timeouts, and disconnections need to be handled manually.

20. What security considerations should be taken into account when implementing socket programming, and how can you address them?

1. **Data Encryption**:
    - **Consideration**: Data transmitted over sockets can be intercepted.
    - **Solution**: Use **SSL/TLS** to encrypt data.
2. **Authentication**:
    - **Consideration**: Ensure the server and client are who they claim to be.
    - **Solution**: Implement **authentication mechanisms** like certificates or tokens.
3. **Authorization**:
    - **Consideration**: Prevent unauthorized access to sensitive resources.
    - **Solution**: Implement **access control** using role-based or user-based permissions.
4. **Data Integrity**:
    - **Consideration**: Ensure data is not tampered with during transmission.
    - **Solution**: Use cryptographic **hash functions** (e.g., SHA) to verify message integrity.