Modul 2 Socket Programming

Tugas Pendahuluan :

1. Jelaskan dengan rinci apa yang dimaksud dengan "Socket" dalam konteks pemrograman jaringan.
2. Gambarkan peran utama dari client dan server dalam model Client-Server Socket. Bagaimana keduanya berinteraksi dalam suatu sistem?
3. Mengapa model Client-Server Socket banyak digunakan dalam pengembangan aplikasi jaringan?
4. Jelaskan apa yang dimaksud dengan IP Address dan mengapa IP Address penting dalam komunikasi jaringan.
5. Gambarkan peran Port dalam koneksi socket dan mengapa kita perlu menggunakan port dalam pengembangan aplikasi.

Materi :

Socket Programming

Socket Programming adalah paradigma pemrograman yang memungkinkan komunikasi antara dua program melalui jaringan menggunakan socket. Socket adalah mekanisme dasar untuk melakukan komunikasi antara dua proses, baik pada satu mesin atau melalui jaringan. Socket memungkinkan pertukaran data antara dua entitas, seperti antara client dan server.

Pada dasarnya, Socket Programming melibatkan dua jenis soket: soket klien (client socket) dan soket server (server socket).

1. Client Socket:

   a) Client socket digunakan oleh program atau aplikasi yang meminta layanan atau informasi dari server.
   b) Ketika program client membutuhkan komunikasi dengan server, ia membuat soket klien.
   c) Soket klien dapat digunakan untuk mengirim permintaan ke server dan menerima respons dari server.

2. Server Socket:

   a) Server socket digunakan oleh program atau aplikasi yang menyediakan layanan atau informasi kepada client.
   b) Server socket mendengarkan permintaan koneksi dari client.
   c) Setiap kali ada permintaan koneksi dari client, server socket membuat soket baru untuk mengelola komunikasi dengan client tersebut.

Penggunaan Socket Programming melibatkan beberapa langkah umum:

1. Membuat Socket:
   a) Program client dan server membuat socket.
   b) Client membuat socket untuk menginisiasi koneksi ke server.
   c) Server membuat socket dan mendengarkan koneksi dari client.
2. Membuat Koneksi:
   a) Client mencoba untuk membuat koneksi dengan server menggunakan alamat IP dan nomor port tertentu.

b) Server menerima permintaan koneksi dan membuat soket baru untuk berkomunikasi dengan client.

3. Mengirim dan Menerima Data:
   a) Setelah koneksi dibuat, client dan server dapat mengirim dan menerima data satu sama lain melalui socket.
   b) Data yang dikirim dapat berupa teks, file, atau format data lainnya.
4. Menutup Koneksi:
   a) Setelah pertukaran data selesai, koneksi dapat ditutup oleh client atau server.
   b) Menutup koneksi membebaskan sumber daya dan mematikan komunikasi antara client dan server.

Socket Programming sering digunakan dalam pengembangan aplikasi jaringan, seperti aplikasi web, permainan online, sistem terdistribusi, dan banyak lagi. Java menyediakan API socket yang kuat untuk memudahkan pengembangan aplikasi jaringan menggunakan Socket Programming.

Prinsip dasar Socket Programming

Prinsip dasar Socket Programming melibatkan beberapa konsep utama yang perlu dipahami untuk mengimplementasikan komunikasi antara dua program melalui jaringan. Berikut adalah prinsip dasar Socket Programming:

1. Inisialisasi dan Pembuatan Socket:
   a) Program client dan server perlu membuat socket untuk memulai komunikasi.
   b) Dalam Java, Anda dapat menggunakan kelas `Socket` untuk membuat socket client dan kelas `ServerSocket` untuk membuat socket server.
   c) Socket client akan terhubung ke alamat IP dan nomor port yang ditentukan untuk server.

```
// Client Socket
Socket clientSocket = new Socket("server_ip", serverPort);

// Server Socket
ServerSocket serverSocket = new ServerSocket(serverPort);
Socket clientSocket = serverSocket.accept(); // Menerima koneksi dari client
```

2. Mengirim dan Menerima Data:
   a) Setelah koneksi terbentuk, client dan server dapat mengirim dan menerima data melalui socket.
   b) Pengiriman dan penerimaan data dapat dilakukan menggunakan input dan output stream yang terkait dengan socket.

```
// Mengirim data dari client
OutputStream outputStream = clientSocket.getOutputStream();
OutputStreamWriter writer = new OutputStreamWriter(outputStream);
writer.write("Hello, Server!");
writer.flush();

// Menerima data di server
InputStream inputStream = clientSocket.getInputStream();
InputStreamReader reader = new InputStreamReader(inputStream);
char[] buffer = new char[1024];
int bytesRead = reader.read(buffer);
```

```
```

3. Penanganan Koneksi:
   a) Server harus mendengarkan permintaan koneksi dari client.
   b) Setiap kali ada permintaan koneksi, server harus membuat socket baru untuk berkomunikasi dengan client tersebut.
   c) Setelah pertukaran data selesai, koneksi dapat ditutup.

```
// Server mendengarkan permintaan koneksi
ServerSocket serverSocket = new ServerSocket(serverPort);
Socket clientSocket = serverSocket.accept(); // Menerima koneksi dari client

// Menutup koneksi
clientSocket.close();
serverSocket.close();
```

4. Penanganan Kesalahan (Error Handling):
   a) Socket Programming dapat melibatkan penanganan kesalahan untuk mengatasi kondisi yang tidak diinginkan, seperti koneksi terputus atau kegagalan pengiriman data.
   b) Penggunaan try-catch blocks sangat penting untuk menangani eksepsi.

```
try {
    // Blok kode Socket Programming
} catch (IOException e) {
    // Penanganan kesalahan
    e.printStackTrace();
}
```

5. Multithreading (Opsional):
   a) Untuk menangani beberapa koneksi secara bersamaan, terutama di server, penggunaan multithreading bisa menjadi solusi.
   b) Setiap koneksi dapat dihandle oleh thread terpisah untuk mencegah blokade pada satu koneksi.

```
// Multithreading di server
while (true) {
    Socket clientSocket = serverSocket.accept();
    Thread clientThread = new ClientHandlerThread(clientSocket);
    clientThread.start();
}
```

Penting untuk diingat bahwa Socket Programming melibatkan komunikasi antara dua program yang berjalan pada mesin yang berbeda atau di mesin yang sama melalui jaringan. Keamanan, penanganan kesalahan, dan manajemen koneksi adalah aspek-aspek penting yang harus dipertimbangkan dalam implementasi.

Socket dan ServerSocket

Kelas `Socket` dan `ServerSocket` dalam Java adalah bagian dari paket `java.net` dan merupakan bagian integral dari API untuk pemrograman jaringan. Berikut adalah penjelasan singkat tentang penggunaan kelas `Socket` dan `ServerSocket`:

1. Socket (Client Side):

Membuat Socket Client:

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

public class ClientExample {
    public static void main(String[] args) {
        try {
            // Membuat socket client dan terhubung ke server di alamat IP dan port tertentu
            Socket clientSocket = new Socket("server_ip", serverPort);

            // Mendapatkan output stream untuk mengirim data ke server
            OutputStream outputStream = clientSocket.getOutputStream();
            OutputStreamWriter writer = new OutputStreamWriter(outputStream);

            // Mengirim data ke server
            writer.write("Hello, Server!");
            writer.flush();

            // Menerima data dari server
            InputStream inputStream = clientSocket.getInputStream();
            InputStreamReader reader = new InputStreamReader(inputStream);
            char[] buffer = new char[1024];
            int bytesRead = reader.read(buffer);

            // Menampilkan data dari server
            System.out.println("Server Response: " + new String(buffer, 0, bytesRead));

            // Menutup socket client
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2. ServerSocket (Server Side):

Membuat Server Socket:

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

public class ServerExample {
    public static void main(String[] args) {
        try {
            // Membuat server socket dan mendengarkan koneksi dari client di port tertentu
            ServerSocket serverSocket = new ServerSocket(serverPort);

            while (true) {
```

```
12.                 // Menerima permintaan koneksi dari client
13.                 Socket clientSocket = serverSocket.accept();
14.
15.                 // Membuat thread untuk menangani koneksi dengan client
16.                 Thread clientThread = new ClientHandlerThread(clientSocket);
17.                 clientThread.start();
18.             }
19.         } catch (IOException e) {
20.             e.printStackTrace();
21.         }
22.     }
23. }
24.
25. class ClientHandlerThread extends Thread {
26.     private Socket clientSocket;
27.
28.     public ClientHandlerThread(Socket clientSocket) {
29.         this.clientSocket = clientSocket;
30.     }
31.
32.     public void run() {
33.         try {
34.             // Menerima data dari client
35.             InputStream inputStream = clientSocket.getInputStream();
36.             InputStreamReader reader = new InputStreamReader(inputStream);
37.             char[] buffer = new char[1024];
38.             int bytesRead = reader.read(buffer);
39.
40.             // Menampilkan data dari client
41.             System.out.println("Received from Client: " + new String(buffer, 0, bytesRead));
42.
43.             // Mengirim data ke client
44.             OutputStream outputStream = clientSocket.getOutputStream();
45.             OutputStreamWriter writer = new OutputStreamWriter(outputStream);
46.             writer.write("Hello, Client!");
47.             writer.flush();
48.
49.             // Menutup socket client
50.             clientSocket.close();
51.         } catch (IOException e) {
52.             e.printStackTrace();
53.         }
54.     }
55. }
56.
```

Penting untuk dicatat bahwa penggunaan `try-catch` sangat penting untuk menangani eksepsi yang mungkin terjadi selama operasi soket. Selain itu, di sisi server, seringkali lebih baik menggunakan multithreading untuk menangani banyak koneksi secara bersamaan.

Untuk menjalankan program Java, Anda dapat mengikuti langkah-langkah umum berikut:

Program Client:

1. Kompilasi Program:
   a) Simpan program client dalam file dengan ekstensi `.java`, misalnya `ClientExample.java`.
   b) Buka terminal atau command prompt dan arahkan ke direktori di mana file program disimpan.
   c) Ketik perintah berikut untuk mengompilasi program:

```
javac ClientExample.java
```

d) Ini akan menghasilkan file `ClientExample.class`.

2. Menjalankan Program:
   a) Setelah berhasil dikompilasi, jalankan program dengan perintah:

```
java ClientExample
```

Program Server:

1. Kompilasi Program:
   a) Simpan program server dalam file dengan ekstensi `.java`, misalnya `ServerExample.java`.
   b) Buka terminal atau command prompt dan arahkan ke direktori di mana file program disimpan.
   c) Ketik perintah berikut untuk mengompilasi program:

```
javac ServerExample.java
```

d) Ini akan menghasilkan file `ServerExample.class`.

2. Menjalankan Program:
   a) Setelah berhasil dikompilasi, jalankan program dengan perintah:

```
java ServerExample
```

Catatan Penting:

- Pastikan bahwa JDK (Java Development Kit) sudah diinstal di sistem Anda.
- Ganti "server_ip" dengan alamat IP sesuai kebutuhan Anda.
- Pastikan bahwa tidak ada firewall atau penghalang lain yang mencegah koneksi antara client dan server.

Setelah program server berjalan, ia akan terus mendengarkan koneksi dan menangani setiap koneksi dari client yang mencoba terhubung. Program client akan membuat koneksi ke server, mengirim pesan, dan menerima respons dari server.

Selama menjalankan program, pastikan bahwa port yang digunakan tidak dikunci oleh program lain dan bahwa tidak ada kesalahan yang muncul selama eksekusi. Jika ada masalah, pesan kesalahan akan membantu Anda menemukan sumber masalahnya.

Contoh :

Berikut adalah contoh lain penggunaan `Socket` dan `ServerSocket` dalam Java, dengan fokus pada pengiriman dan penerimaan objek sederhana:

Program Server (Menerima Objek dari Client):

```
1. import java.io.*;
2. import java.net.ServerSocket;
3. import java.net.Socket;
4.
5. public class ObjectServer {
6.     public static void main(String[] args) {
7.         try {
8.             ServerSocket serverSocket = new ServerSocket(8888);
```

```
 9.
10.             System.out.println("Server waiting for client...");
11.
12.             while (true) {
13.                 Socket clientSocket = serverSocket.accept();
14.                 System.out.println("Client connected: " + clientSocket.getInetAddress());
15.
16.                 ObjectInputStream objectInputStream = new
ObjectInputStream(clientSocket.getInputStream());
17.
18.                 // Menerima objek dari client
19.                 try {
20.                     Object receivedObject = objectInputStream.readObject();
21.                     System.out.println("Received object from client: " + receivedObject);
22.                 } catch (ClassNotFoundException e) {
23.                     e.printStackTrace();
24.                 }
25.
26.                 // Menutup socket client
27.                 clientSocket.close();
28.             }
29.         } catch (IOException e) {
30.             e.printStackTrace();
31.         }
32.     }
33. }
34.
```

Program Client (Mengirim Objek ke Server):

```
 1. import java.io.*;
 2. import java.net.Socket;
 3.
 4. public class ObjectClient {
 5.     public static void main(String[] args) {
 6.         try {
 7.             Socket clientSocket = new Socket("localhost", 8888);
 8.
 9.             // Mengirim objek ke server
10.             ObjectOutputStream objectOutputStream = new
ObjectOutputStream(clientSocket.getOutputStream());
11.             MyObject myObject = new MyObject("Hello from Client!");
12.             objectOutputStream.writeObject(myObject);
13.             objectOutputStream.flush();
14.
15.             // Menutup socket client
16.             clientSocket.close();
17.         } catch (IOException e) {
18.             e.printStackTrace();
19.         }
20.     }
21. }
22.
23. class MyObject implements Serializable {
24.     private String message;
25.
26.     public MyObject(String message) {
27.         this.message = message;
28.     }
```

```
29.
30.     @Override
31.     public String toString() {
32.         return "MyObject{" +
33.                 "message='" + message + '\'' +
34.                 '}';
35.     }
36. }
37.
```

Dalam contoh ini, program client membuat objek `MyObject` dan mengirimkannya ke server menggunakan `ObjectOutputStream`. Server menerima objek tersebut menggunakan `ObjectInputStream` dan mencetaknya ke konsol.

Pastikan untuk menyesuaikan alamat IP dan port sesuai kebutuhan Anda. Program ini menunjukkan cara menggunakan `ObjectInputStream` dan `ObjectOutputStream` untuk mengirim dan menerima objek melalui jaringan.

Latihan

Perbaiki sintaksis program di bawah ini agar dapat dijalankan dengan benar:

Latihan 1: Perbaikan Server Program

Program server mengalami beberapa kesalahan sintaksis yang membuatnya tidak dapat dijalankan. Identifikasi dan perbaiki kesalahan sintaksis tersebut sehingga program dapat berjalan dengan baik.

```
 1.     import java.io.*;
 2.     import java.net.ServerSocket;
 3.
 4.
 5.     public class ObjectServer {
 6.         public static void main(String[] args) {
 7.             try {
 8.                 ServerSocket serverSocket = new ServerSocket(8888);
 9.
10.                 System.out.println("Server waiting for client...");
11.
12.                 while (true) {
13.                     Socket clientSocket = serverSocket.accept();
14.                     System.out.println("Client connected: " +
clientSocket.getInetAddress());
15.
16.                     ObjectInputStream objectInputStream = new
ObjectInputStream(clientSocket.getInputStream());
17.
18.                     // Menerima objek dari client
19.                     try {
20.                         Object receivedObject = objectInputStream.readObject();
21.                         System.out.println("Received object from client: " +
receivedObject);
22.                     } catch (ClassNotFoundException e) {
23.                         e.printStackTrace();
24.                     }
25.
26.                     // Menutup socket client
27.                     clientSocket.close();
```

```
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

```

Latihan 2: Perbaikan Client Program

Program client juga memiliki kesalahan sintaksis yang membuatnya tidak dapat dijalankan. Identifikasi dan perbaiki kesalahan sintaksis tersebut sehingga program dapat berjalan dengan baik.

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

public class ObjectClient {
    public static void main(String[] args) {
        try {
            Socket clientSocket = new Socket("localhost", 9999);

            // Mengirim objek ke server
            ObjectOutputStream objectOutputStream = new
ObjectOutputStream(clientSocket.getOutputStream());
            MyObject myObject = new MyObject("Hello from Client!");
            objectOutputStream.writeObject(myObject);
            objectOutputStream.flush();

            // Menutup socket client
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class MyObject implements Serializable {
    private String message;

    public MyObject(String message) {
        this.message = message;
    }

    @Override
    public String toString() {
        return "MyObject{" +
                "message='" + message + '\'' +
                '}';
    }
}

```

Latihan 3: Membuat Server Program

Buatlah program server sederhana yang akan mendengarkan koneksi dari client dan menampilkan pesan yang diterima dari client.

SimpleServer.java:

```
1.      import java.io.*;
2.      import java.net.ServerSocket;
3.      import java.net.Socket;
4.
5.      public class SimpleServer {
6.          public static void main(String[] args) {
7.              try {
8.                  ServerSocket serverSocket = new ServerSocket(8888);
9.
10.                 System.out.println("Server waiting for client...");
11.
12.                 while (true) {
13.                     Socket clientSocket = serverSocket.accept();
14.                     System.out.println("Client connected: " +
clientSocket.getInetAddress());
15.
16.                     BufferedReader reader = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
17.                     String message = reader.readLine();
18.
19.                     System.out.println("Received from client: " + message);
20.
21.                     // Menutup socket client
22.                     clientSocket.close()
23.                 }
24.             } catch (IOException E) {
25.                 e.printStackTrace();
26.             }
27.         }
28.     }
29.
```

Latihan 4: Membuat Client Program

Buatlah program client sederhana yang akan membuat koneksi ke server dan mengirim pesan ke server.

```
1.      import java.io.*;
2.      import java.net.Socket;
3.
4.      public class SimpleClient {
5.          public static void main(String[] args) {
6.              try {
7.                  Socket clientSocket = new Socket("localhost", 8888);
8.
9.                  // Mengirim pesan ke server
10.                 BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream()));
11.                 writer.write("Hello from Client!");
12.                 writer.newLine();
13.                 writer.flush();
14.
15.                 // Menutup socket client
16.                 clientSocket.close();
17.             } catch (IOException e) {
18.                 e.printStackTrace()
```

```
19.             }
20.         }
21.     }
22.
```

Tugas Rumah:

Deskripsi Tugas:

Anda diminta untuk membuat program chatting sederhana menggunakan Java dengan memanfaatkan Server Socket dan Client Socket. Program ini akan memungkinkan dua pengguna (client) untuk saling berkomunikasi melalui server.

Petunjuk:

1. Implementasikan Server Program:
   - Server harus dapat menerima koneksi dari dua client.
   - Server kemudian akan menyampaikan pesan dari satu client kepada client lainnya.
   - Pastikan server dapat menangani kedua client secara bersamaan.
2. Implementasikan Client Program:
   - Setiap client harus dapat mengirim dan menerima pesan.
   - Pesan yang dikirim oleh salah satu client harus ditampilkan oleh client yang lain.
   - Implementasikan fungsi-fungsi untuk mengirim dan menerima pesan di client.

Langkah-langkah tambahan:

1. Sesuaikan port yang digunakan untuk koneksi server dan alamat IP jika perlu.
2. Coba jalankan server dan dua instance dari client untuk menguji komunikasi.
3. Periksa apakah pesan yang dikirim oleh satu client dapat diterima oleh client yang lain.

Pertanyaan Tugas:

1. Bagaimana mekanisme koneksi antara client dan server pada program Anda?
2. Apa yang terjadi jika salah satu client terputus dari server? Bagaimana Anda dapat menangani situasi ini?
3. Bagaimana cara menangani multiple clients secara bersamaan di server?
4. Apa yang terjadi jika terdapat kesalahan dalam koneksi atau pengiriman pesan? Bagaimana cara Anda menanganinya?