MODUL 3
Socket Programming (Blocking & Non Blocking)

Tugas Pendahuluan :

1. Jelaskan konsep utama dari model blocking dalam pemrograman socket. Bagaimana program berperilaku saat menunggu operasi I/O selesai? Berikan contoh penggunaan dalam Java.
2. Apa perbedaan utama antara model blocking dan non-blocking dalam pemrograman socket? Sebutkan keuntungan dan kerugian masing-masing model.
3. Dalam konteks Java, sebutkan setidaknya tiga operasi yang biasanya menggunakan model blocking socket dan tiga operasi yang menggunakan model non-blocking socket.
4. Apakah situasi di mana kita akan memilih model blocking lebih cocok daripada model non-blocking, dan sebaliknya? Berikan contoh aplikasi yang mungkin lebih baik menggunakan satu model daripada yang lain.
5. Bagaimana Anda menghindari "busy waiting" dalam implementasi model non-blocking? Jelaskan logika yang diterapkan untuk menangani peristiwa dan data masuk tanpa menggunakan loop yang berputar terus menerus.

Materi :

Pemrograman socket menghadirkan dua model utama: blocking dan non-blocking. Dalam model blocking, operasi I/O membuat program berhenti atau 'terblokir' sampai operasi tersebut selesai. Ini cocok untuk aplikasi dengan jumlah koneksi terbatas dan tidak memerlukan skala besar atau responsivitas tinggi. Model ini menawarkan desain yang sederhana dan mudah dimengerti namun tidak efisien untuk menangani banyak koneksi secara bersamaan.

Sebaliknya, model non-blocking memungkinkan program untuk melanjutkan eksekusi tanpa menunggu operasi I/O selesai. Program dapat melakukan beberapa tugas atau menangani banyak koneksi secara bersamaan, cocok untuk aplikasi yang memerlukan responsivitas tinggi dan menangani banyak koneksi. Namun, implementasi model ini lebih kompleks, memerlukan logika pemantauan dan penanganan peristiwa, dan perlu memitigasi "busy waiting" agar tidak terjadi loop yang berputar terus menerus tanpa hasil.

Dalam pemrograman socket Java, contoh blocking socket melibatkan pemakaian `ServerSocket.accept()` dan operasi pembacaan/tulisan blocking, sementara contoh non-blocking menggunakan `SocketChannel.configureBlocking(false)`, `Selector.select()`, serta operasi baca/tulis non-blocking. Pemilihan antara kedua model ini tergantung pada kebutuhan spesifik aplikasi, dengan model blocking cocok untuk skenario yang lebih sederhana dan non-blocking untuk aplikasi yang memerlukan skala besar dan responsivitas tinggi. Berikut adalah ciri dari Blocking dan Non Blocking:

1. Blocking Socket (Socket Blok):

   Ciri Utama: Saat operasi input/output (I/O) dilakukan pada soket, program akan 'terblokir' atau 'menghentikan' eksekusi hingga operasi I/O selesai.

   Keuntungan: Sederhana dan mudah diimplementasikan karena eksekusi program menunggu hingga data tersedia atau operasi selesai.

Kekurangan: Jika ada banyak koneksi, dapat mengakibatkan penundaan karena program menunggu setiap operasi I/O selesai sebelum melanjutkan.

2. Non-Blocking Socket (Socket Non-Blok):

   Ciri Utama: Program dapat melanjutkan eksekusi setelah memulai operasi I/O tanpa menunggu operasi selesai. Program kemudian dapat memeriksa kembali koneksi atau data nantinya.

   Keuntungan: Efisien untuk menangani banyak koneksi sekaligus tanpa menghentikan eksekusi program utama.

   Kekurangan: Lebih kompleks dalam implementasinya karena memerlukan pemantauan aktif terhadap koneksi dan data yang masuk/keluar.

Keputusan untuk menggunakan model blocking atau non-blocking tergantung pada kebutuhan aplikasi Anda. Aplikasi yang membutuhkan penanganan banyak koneksi secara bersamaan dan responsif terhadap peristiwa dapat memilih model non-blocking. Sebaliknya, untuk aplikasi yang lebih sederhana atau tidak memiliki jumlah koneksi yang besar, model blocking mungkin lebih sederhana untuk diimplementasikan.

Contoh aplikasi nyata yang menggunakan model socket, baik blocking maupun non-blocking, sangat beragam. Berikut adalah beberapa contoh:

1. Aplikasi Chat Real-Time:

   Blocking: Aplikasi chat yang sederhana dan tidak memerlukan responsivitas tinggi mungkin menggunakan model blocking karena mudah diimplementasikan.

   Non-blocking: Aplikasi chat real-time yang membutuhkan responsivitas tinggi dan menangani banyak pengguna secara bersamaan dapat memilih model non-blocking untuk menjaga efisiensi dan responsivitas.

2. Server Web:

   Blocking: Server web yang melayani sedikit pengguna pada satu waktu dan tidak memerlukan kinerja tinggi.

   Non-blocking: Server web yang harus menangani banyak permintaan secara bersamaan dan memberikan responsivitas tinggi kepada pengguna.

3. Game Online Multiplayer:

   Blocking: Game sederhana dengan sedikit pemain dan tidak memerlukan responsivitas tinggi.

   Non-blocking: Game online multiplayer yang harus menangani banyak pemain secara bersamaan dan memberikan responsivitas tinggi terhadap peristiwa dalam permainan.

4. Aplikasi Keuangan:

   Blocking: Aplikasi keuangan sederhana yang membutuhkan koneksi ke server untuk memperbarui saldo atau mendapatkan informasi transaksi.

Non-blocking: Aplikasi trading saham online yang harus menghandle banyak transaksi secara bersamaan dengan responsivitas tinggi.

5. Sistem Monitoring Jaringan:

   Blocking: Sistem monitoring jaringan yang hanya perlu memeriksa koneksi jaringan pada interval waktu tertentu.

   Non-blocking: Sistem monitoring jaringan real-time yang harus merespons perubahan dalam jaringan segera setelah terjadi.

Pilihan antara blocking dan non-blocking sangat tergantung pada kebutuhan dan karakteristik aplikasi tersebut. Sebuah aplikasi dapat menggunakan kedua model tergantung pada kebutuhan spesifiknya.

Blocking

Berikut adalah contoh sederhana program menggunakan blocking socket. Program ini adalah server sederhana yang menerima koneksi dari client dan mengirimkan pesan balasan:

BlockingSocketServer.java (Server):

```
 1. import java.io.BufferedReader;
 2. import java.io.IOException;
 3. import java.io.InputStreamReader;
 4. import java.io.PrintWriter;
 5. import java.net.ServerSocket;
 6. import java.net.Socket;
 7.
 8. public class BlockingSocketServer {
 9.     private static final int PORT = 8888;
10.
11.     public static void main(String[] args) {
12.         try {
13.             ServerSocket serverSocket = new ServerSocket(PORT);
14.             System.out.println("Server is listening on port " + PORT);
15.
16.             while (true) {
17.                 // Menunggu koneksi dari client (blocking)
18.                 Socket clientSocket = serverSocket.accept();
19.                 System.out.println("Client connected: " + clientSocket.getInetAddress());
20.
21.                 // Menerima pesan dari client (blocking)
22.                                         BufferedReader   reader   =   new   BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
23.                 String clientMessage = reader.readLine();
24.                 System.out.println("Received from client: " + clientMessage);
25.
26.                 // Mengirim pesan balasan ke client (blocking)
27.                 PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true);
28.                 writer.println("Hello from Server!");
29.
30.                 // Menutup socket client
31.                 clientSocket.close();
32.             }
33.         } catch (IOException e) {
34.             e.printStackTrace();
35.         }
```

```
36.     }
37. }
38.
```

BlockingSocketClient.java (Client):

```
 1. import java.io.BufferedReader;
 2. import java.io.IOException;
 3. import java.io.InputStreamReader;
 4. import java.io.PrintWriter;
 5. import java.net.Socket;
 6.
 7. public class BlockingSocketClient {
 8.     private static final String SERVER_IP = "localhost";
 9.     private static final int SERVER_PORT = 8888;
10.
11.     public static void main(String[] args) {
12.         try {
13.             // Membuat koneksi ke server (blocking)
14.             Socket socket = new Socket(SERVER_IP, SERVER_PORT);
15.             System.out.println("Connected to server");
16.
17.             // Mengirim pesan ke server (blocking)
18.             PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
19.             writer.println("Hello from Client!");
20.
21.             // Menerima pesan balasan dari server (blocking)
22.                                     BufferedReader   reader   =   new   BufferedReader(new
InputStreamReader(socket.getInputStream()));
23.             String serverMessage = reader.readLine();
24.             System.out.println("Received from server: " + serverMessage);
25.
26.             // Menutup socket
27.             socket.close();
28.         } catch (IOException e) {
29.             e.printStackTrace();
30.         }
31.     }
32. }
33.
```

Cara kerja program ini adalah server akan terus menerima koneksi dari client dan menanggapi dengan mengirim pesan balasan. Baik pengiriman pesan dari client ke server maupun dari server ke client adalah operasi blok, yang berarti program akan 'terblokir' sampai operasi tersebut selesai. Program ini sederhana dan hanya dimaksudkan untuk ilustrasi.

Dalam program di atas, karakteristik dari blocking socket dapat dilihat pada beberapa titik di dalam kodenya, terutama pada operasi operasi I/O (Input/Output). Karakteristik ini mencakup kemampuan program untuk menanggapi atau berhenti sementara (blok) saat mengeksekusi operasi I/O tertentu. Mari kita lihat di mana karakteristik tersebut terlihat:

BlockingSocketServer.java (Server):

```
1. // Menunggu koneksi dari client (blocking)
2. Socket clientSocket = serverSocket.accept();
```

```
3.
4. // Menerima pesan dari client (blocking)
5.          BufferedReader          reader          =          new          BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
6. String clientMessage = reader.readLine();
7.
8. // Mengirim pesan balasan ke client (blocking)
9. PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true);
10. writer.println("Hello from Server!");
11.
12. // Menutup socket client
13. clientSocket.close();
14. ```
15.
```

BlockingSocketClient.java (Client):

```
1. // Membuat koneksi ke server (blocking)
2. Socket socket = new Socket(SERVER_IP, SERVER_PORT);
3.
4. // Mengirim pesan ke server (blocking)
5. PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
6. writer.println("Hello from Client!");
7.
8. // Menerima pesan balasan dari server (blocking)
9. BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
10. String serverMessage = reader.readLine();
11.
12. // Menutup socket
13. socket.close();
14. ```
15.
```

Pada kodenya, karakteristik blocking socket tercermin pada:

BlockingSocketServer.java (Server):

1. `serverSocket.accept()`:
   - Program server akan terhenti dan menunggu di sini sampai ada koneksi masuk dari client.
   - Selama menunggu, program server tidak akan melakukan eksekusi lebih lanjut.
   - Baru setelah ada koneksi dari client, program server akan melanjutkan untuk menerima dan menanggapi pesan dari client.
2. `reader.readLine()`:
   - Setelah server menerima koneksi dari client, program server akan memanggil `reader.readLine()` untuk membaca pesan yang dikirim oleh client.
   - Operasi ini bersifat blok, artinya program server akan berhenti di sini sampai ada data yang tersedia untuk dibaca.
   - Setelah pesan berhasil dibaca, program server akan melanjutkan untuk mengirimkan pesan balasan ke client.
3. `writer.println()`:
   - Setelah membaca pesan dari client, program server akan memanggil `writer.println()` untuk mengirimkan pesan balasan ke client.
   - Seperti sebelumnya, operasi ini bersifat blok, dan program server akan berhenti sampai data berhasil ditulis.

- Setelah pesan balasan berhasil dikirim, program server akan melanjutkan untuk menutup socket yang terkait dengan client.

BlockingSocketClient.java (Client):

1. `socket = new Socket(SERVER_IP, SERVER_PORT)`:
   - Program client akan terhenti di sini saat membuat koneksi ke server. Ini adalah operasi blok sampai koneksi berhasil dibuat atau gagal.
2. `writer.println()`:
   - Setelah koneksi berhasil dibuat, program client akan memanggil `writer.println()` untuk mengirimkan pesan ke server.
   - Operasi ini bersifat blok, dan program client akan berhenti sampai data berhasil ditulis.
3. `reader.readLine()`:
   - Setelah mengirim pesan ke server, program client akan memanggil `reader.readLine()` untuk menerima pesan balasan dari server.
   - Operasi ini bersifat blok, dan program client akan berhenti sampai ada data yang tersedia untuk dibaca.
   - Setelah pesan balasan berhasil dibaca, program client akan melanjutkan untuk menutup socket.

Karakteristik utama dari model blocking socket adalah bahwa operasi-operasi ini dapat membuat program 'terblokir' atau berhenti sementara selama menunggu operasi I/O selesai. Ini dapat menghasilkan desain yang sederhana dan mudah dimengerti, tetapi dapat menjadi masalah jika aplikasi harus menangani banyak koneksi secara bersamaan atau jika responsivitas tinggi diperlukan.

Non Blocking

Dalam model non-blocking socket, program dapat melanjutkan eksekusi tanpa harus menunggu operasi I/O selesai. Ini memungkinkan program untuk melakukan beberapa tugas atau menangani banyak koneksi secara bersamaan tanpa harus menunggu setiap operasi I/O selesai. Berikut adalah contoh sederhana program menggunakan non-blocking socket dengan Java menggunakan `java.nio`:

NonBlockingSocketServer.java (Server):

```
 1. import java.io.IOException;
 2. import java.net.InetSocketAddress;
 3. import java.nio.ByteBuffer;
 4. import java.nio.channels.SelectionKey;
 5. import java.nio.channels.Selector;
 6. import java.nio.channels.ServerSocketChannel;
 7. import java.nio.channels.SocketChannel;
 8. import java.util.Iterator;
 9. import java.util.Set;
10.
11. public class NonBlockingSocketServer {
12.     private static final int PORT = 8888;
13.
14.     public static void main(String[] args) {
15.         try {
16.             ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
17.             serverSocketChannel.bind(new InetSocketAddress(PORT));
18.             serverSocketChannel.configureBlocking(false);
19.
```

```
20.             Selector selector = Selector.open();
21.             serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);
22.
23.             System.out.println("Server is listening on port " + PORT);
24.
25.             while (true) {
26.                 selector.select();
27.
28.                 Set<SelectionKey> selectedKeys = selector.selectedKeys();
29.                 Iterator<SelectionKey> keyIterator = selectedKeys.iterator();
30.
31.                 while (keyIterator.hasNext()) {
32.                     SelectionKey key = keyIterator.next();
33.
34.                     if (key.isAcceptable()) {
35.                         // Koneksi baru
36.                      ServerSocketChannel serverChannel = (ServerSocketChannel) key.channel();
37.                         SocketChannel clientChannel = serverChannel.accept();
38.                         clientChannel.configureBlocking(false);
39.                         clientChannel.register(selector, SelectionKey.OP_READ);
40.                                                 System.out.println("Client  connected:  "  +
clientChannel.getRemoteAddress());
41.
42.                     } else if (key.isReadable()) {
43.                         // Membaca data dari client
44.                         SocketChannel clientChannel = (SocketChannel) key.channel();
45.                         ByteBuffer buffer = ByteBuffer.allocate(1024);
46.                         int bytesRead = clientChannel.read(buffer);
47.
48.                         if (bytesRead == -1) {
49.                             // Koneksi ditutup oleh client
50.                                                 System.out.println("Client  disconnected:  "  +
clientChannel.getRemoteAddress());
51.                             clientChannel.close();
52.                         } else if (bytesRead > 0) {
53.                             buffer.flip();
54.                             byte[] data = new byte[buffer.remaining()];
55.                             buffer.get(data);
56.                             String clientMessage = new String(data);
57.                             System.out.println("Received from client: " + clientMessage);
58.
59.                             // Kirim pesan balasan
60.                             String responseMessage = "Hello from Server!";
61.                            clientChannel.write(ByteBuffer.wrap(responseMessage.getBytes()));
62.                         }
63.                     }
64.
65.                     keyIterator.remove();
66.                 }
67.             }
68.         } catch (IOException e) {
69.             e.printStackTrace();
70.         }
71.     }
72. }
73. ```
74.
```

NonBlockingSocketClient.java (Client):

```
1. import java.io.IOException;
2. import java.net.InetSocketAddress;
3. import java.nio.ByteBuffer;
4. import java.nio.channels.SocketChannel;
5.
6. public class NonBlockingSocketClient {
7.     private static final String SERVER_IP = "localhost";
8.     private static final int SERVER_PORT = 8888;
9.
10.     public static void main(String[] args) {
11.         try {
12.             SocketChannel socketChannel = SocketChannel.open(new InetSocketAddress(SERVER_IP,
SERVER_PORT));
13.             socketChannel.configureBlocking(false);
14.
15.             // Mengirim pesan ke server
16.             String message = "Hello from Client!";
17.             socketChannel.write(ByteBuffer.wrap(message.getBytes()));
18.
19.             // Menerima pesan balasan dari server
20.             ByteBuffer buffer = ByteBuffer.allocate(1024);
21.             while (socketChannel.read(buffer) == 0) {
22.                 // Menunggu data tersedia
23.             }
24.
25.             buffer.flip();
26.             byte[] data = new byte[buffer.remaining()];
27.             buffer.get(data);
28.             String serverMessage = new String(data);
29.             System.out.println("Received from server: " + serverMessage);
30.
31.             // Menutup socket
32.             socketChannel.close();
33.         } catch (IOException e) {
34.             e.printStackTrace();
35.         }
36.     }
37. }
38. ```
39.
```

Dalam contoh ini, kita menggunakan `java.nio` untuk menciptakan server dan client non-blocking. Beberapa titik kunci:

1. configureBlocking(false):

Pada saat membuat channel (ServerSocketChannel dan SocketChannel), kita mengonfigurasikannya agar non-blocking dengan memanggil `configureBlocking(false)`.

2. selector.select():

Server menggunakan `Selector` untuk memantau peristiwa I/O. Metode `select()` pada `Selector` akan tetap non-blocking dan akan mengembalikan setiap kunci seleksi yang siap untuk dieksekusi.

3. Operasi I/O non-blocking:

Dalam server, operasi baca dan tulis diimplementasikan secara non-blocking dengan menggunakan `SocketChannel.read()` dan `SocketChannel.write()`. Operasi ini tidak akan memblokir program.

Dengan menggunakan non-blocking socket, program dapat terus melanjutkan eksekusi dan menangani banyak koneksi secara bersamaan tanpa harus menunggu setiap operasi I/O selesai. Namun, perhatikan bahwa model non-blocking membutuhkan pemrograman yang lebih kompleks dibandingkan dengan model blocking.

Terdapat beberapa konsep dan poin-poin penting yang perlu diketahui terkait dengan pemrograman socket baik dalam konteks blocking maupun non-blocking:

Blocking Socket:

1. Sederhana dan Mudah Dimengerti:

   Model blocking lebih sederhana dan mudah dimengerti karena program berhenti atau 'terblokir' saat menunggu operasi I/O selesai.

2. Konservatif terhadap Sumber Daya:

   Karena program berhenti saat menunggu, model blocking dapat lebih efisien dalam penggunaan sumber daya pada jumlah koneksi yang relatif sedikit.

3. Responsif untuk Koneksi Sedikit:

   Cocok untuk aplikasi dengan jumlah koneksi yang terbatas dan tidak memerlukan skala besar atau responsivitas tinggi.

Non-Blocking Socket:

1. Skala Lebih Baik untuk Banyak Koneksi:

   Model non-blocking cocok untuk menangani banyak koneksi secara bersamaan karena program dapat melanjutkan eksekusi tanpa harus menunggu setiap operasi I/O selesai.

2. Responsivitas Tinggi:

   Cocok untuk aplikasi yang memerlukan responsivitas tinggi terhadap peristiwa dan data yang masuk, seperti aplikasi real-time dan game online.

3. Kompleksitas Lebih Tinggi:

   Implementasi model non-blocking lebih kompleks daripada model blocking karena program harus secara aktif memantau koneksi dan data yang masuk/keluar.

4. Kemungkinan Busy Waiting:

   Dalam model non-blocking, perlu menerapkan logika yang tepat untuk menghindari "busy waiting" atau loop yang berputar terus menerus tanpa hasil.

5. Selector:

   Dalam Java, untuk model non-blocking, biasanya digunakan `Selector` untuk memantau beberapa channel dan menanggapi peristiwa I/O.

Umum untuk Keduanya:

1. Buffering:

   Baik dalam model blocking maupun non-blocking, penggunaan buffering untuk membaca dan menulis data sangat umum. Dalam Java, `ByteBuffer` sering digunakan untuk ini.

2. Menangani Koneksi Terputus:

   Baik dalam model blocking maupun non-blocking, program harus dapat menangani koneksi yang terputus atau keluar dari koneksi.

3. Keamanan:

   Dalam kedua model, perhatikan keamanan untuk menghindari kerentanan keamanan seperti serangan DoS (Denial of Service).

Pemilihan antara model blocking dan non-blocking tergantung pada kebutuhan spesifik aplikasi. Beberapa aplikasi mungkin memilih kombinasi dari keduanya sesuai dengan kebutuhan fungsionalitas dan skala yang diinginkan.

Latihan

Latihan untuk Memperbaiki Program Socket

Latihan 1: Memperbaiki Program Blocking Socket

Program server dan client di bawah ini menggunakan model blocking socket. Ada beberapa masalah dalam penulisan dan logika program yang perlu diperbaiki. Identifikasi masalah dan perbaiki agar program dapat berjalan dengan benar.

```
1. // BlockingSocketServer.java
2. // Program Server
3. // ...
4.
5. public class BlockingSocketServer {
6.     private static final int PORT = 8888;
7.
8.     public static void main(String[] args) {
9.         try {
10.             ServerSocket serverSocket = new ServerSocket(PORT);
11.             System.out.println("Server is listening on port " + PORT);
12.
13.             while (true) {
14.                 Socket clientSocket = serverSocket.accept();
15.                 System.out.println("Client connected: " + clientSocket.getInetAddress());
16.
17.                                         BufferedReader   reader   =   new   BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
18.                 String clientMessage = reader.readLine();
19.                 System.out.println("Received from client: " + clientMessage);
20.
```

```
                // Kirim pesan balasan ke client
                PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true);
                writer.println("Hello from Server!");

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

```
// BlockingSocketClient.java
// Program Client
// ...

public class BlockingSocketClient {
    private static final String SERVER_IP = "localhost";
    private static final int SERVER_PORT = 8888;

    public static void main(String[] args) {
        try {
            Socket socket = new Socket(SERVER_IP, SERVER_PORT);
            System.out.println("Connected to server");

            // Mengirim pesan ke server
            PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
            writer.println("Hello from Client!");

            // Menerima pesan balasan dari server
                                    BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            String serverMessage = reader.readLine();
            System.out.println("Received from server: " + serverMessage);

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

Latihan 2: Memperbaiki Program Non-Blocking Socket

Program server dan client di bawah ini menggunakan model non-blocking socket dengan `java.nio`. Ada beberapa masalah dalam penulisan dan logika program yang perlu diperbaiki. Identifikasi masalah dan perbaiki agar program dapat berjalan dengan benar.

```
// NonBlockingSocketServer.java
```

```
// Program Server
// ...

public class NonBlockingSocketServer {
    private static final int PORT = 8888;

    public static void main(String[] args) {
        try {
            ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
            serverSocketChannel.bind(new InetSocketAddress(PORT));
            serverSocketChannel.configureBlocking(false);

            Selector selector = Selector.open();
            serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);

            System.out.println("Server is listening on port " + PORT);

            while (true) {
                selector.select();

                Set<SelectionKey> selectedKeys = selector.selectedKeys();
                Iterator<SelectionKey> keyIterator = selectedKeys.iterator();

                while (keyIterator.hasNext()) {
                    SelectionKey key = keyIterator.next();

                    if (key.isAcceptable()) {
                        // Koneksi baru
                     ServerSocketChannel serverChannel = (ServerSocketChannel) key.channel();
                        SocketChannel clientChannel = serverChannel.accept();
                        clientChannel.configureBlocking(false);
                        clientChannel.register(selector, SelectionKey.OP_READ);
                                                System.out.println("Client  connected:  "  +
clientChannel.getRemoteAddress());

                    } else if (key.isReadable()) {
                        // Membaca data dari client
                        SocketChannel clientChannel = (SocketChannel) key.channel();
                        ByteBuffer buffer = ByteBuffer.allocate(1024);
                        int bytesRead = clientChannel.read(buffer);

                        if (bytesRead == -1) {
                            // Koneksi ditutup oleh client
                                                System.out.println("Client  disconnected:  "  +
clientChannel.getRemoteAddress());
                            clientChannel.close();
                        } else if (bytesRead > 0) {
                            buffer.flip();
                            byte[] data = new byte[buffer.remaining()];
                            buffer.get(data);
                            String clientMessage = new String(data);
                            System.out.println("Received from client: " + clientMessage);

                            // Kirim pesan balasan
                            String responseMessage = "Hello from Server!";
                           clientChannel.write(ByteBuffer.wrap(responseMessage.getBytes()));
                        }
                    }

                    keyIterator.remove();
```

```
60.                 }
61.             }
62.         } catch (IOException e) {
63.             e.printStackTrace();
64.         }
65.     }
66. }
67. ```
68.
```

```
 1. // NonBlockingSocketClient.java
 2. // Program Client
 3. // ...
 4.
 5. public class NonBlockingSocketClient {
 6.     private static final String SERVER_IP = "localhost";
 7.     private static final int SERVER_PORT = 8888;
 8.
 9.     public static void main(String[] args) {
10.         try {
11.             SocketChannel socketChannel = SocketChannel.open(new InetSocketAddress(SERVER_IP,
SERVER_PORT));
12.             socketChannel.configureBlocking(false);
13.
14.             // Mengirim pesan ke server
15.             String message = "Hello from Client!";
16.             socketChannel.write(ByteBuffer.wrap(message.getBytes()));
17.
18.             // Menerima pesan balasan dari server
19.             ByteBuffer buffer = ByteBuffer.allocate(1024);
20.             while (socketChannel.read(buffer) == 0) {
21.                 // Menunggu data tersedia
22.             }
23.
24.             buffer.flip();
25.             byte[] data = new byte[buffer.remaining()];
26.             buffer.get(data);
27.             String serverMessage = new String(data);
28.             System.out.println("Received from server: " + serverMessage);
29.
30.             // Menutup socket
31.             socketChannel.close();
32.         } catch (IOException e) {
33.             e.printStackTrace();
34.         }
35.     }
36. }
37. ```
38.
```

Selamat mencoba memperbaiki program-program tersebut! Jika ada pertanyaan atau kesulitan, jangan ragu untuk bertanya.

Tugas Rumah :

Implementasi Blocking dan Non-Blocking Socket Programming**

Deskripsi Tugas:

Implementasikan dua program jaringan sederhana menggunakan bahasa pemrograman Java, masing-masing menggunakan model blocking dan non-blocking dalam pemrograman socket. Program tersebut harus dirancang untuk menangani koneksi antara server dan beberapa klien. Gunakan konsep pengiriman dan penerimaan pesan antara server dan klien.

Persyaratan Umum:

1. Implementasikan dua program terpisah, yaitu `BlockingServer.java` dan `BlockingClient.java` untuk model blocking, serta `NonBlockingServer.java` dan `NonBlockingClient.java` untuk model non-blocking.
2. Program harus memungkinkan server menerima koneksi dari beberapa klien secara bersamaan dan dapat mengelola pesan yang dikirim dan diterima.
3. Pastikan program memiliki mekanisme untuk menangani koneksi yang terputus dan memberikan notifikasi kepada klien dan server ketika hal tersebut terjadi.
4. Pesan yang dikirim oleh klien ke server harus mencakup identifikasi pengirim.
5. Program harus memberikan keluaran yang jelas di konsol, termasuk pesan yang diterima dan dikirim, serta notifikasi tentang koneksi yang masuk atau terputus.

Blocking Socket Programming:

1. Server harus dapat menerima koneksi dari beberapa klien secara bersamaan.
2. Klien harus dapat mengirim pesan ke server dan menerima pesan dari server.

Non-Blocking Socket Programming:

1. Gunakan `java.nio` untuk mengimplementasikan model non-blocking.
2. Server harus memanfaatkan `Selector` dan `SelectionKey` untuk memonitor koneksi dan menerima pesan dari klien yang terhubung.
3. Klien harus menggunakan `SocketChannel` untuk mengirim dan menerima pesan.

Catatan Penting:

1. Pastikan untuk memberikan komentar yang jelas dalam kode untuk menjelaskan langkah-langkah utama dalam implementasi.
2. Jalankan kedua program dan tunjukkan bahwa mereka dapat beroperasi secara independen dan mengelola koneksi dengan benar.

Pengumpulan Tugas:

1. Kumpulkan dua file Java untuk program blocking (`BlockingServer.java` dan `BlockingClient.java`) dan dua file Java untuk program non-blocking (`NonBlockingServer.java` dan `NonBlockingClient.java`).

2. Kumpulkan program di account github/gitlab masing masing dan salin Alamat dari git tersbut ke tugas di elearning sebagai jawaban.

Penilaian:

Tugas ini akan dinilai berdasarkan implementasi fungsionalitas yang benar dari kedua model, kejelasan dan kerapian kode, serta kemampuan untuk mengelola koneksi dan pesan dengan baik. Juga, diperiksa apakah program dapat menangani kondisi khusus seperti koneksi yang terputus. Jangan ragu untuk menunjukkan kreativitas dalam mengelola solusi.