MODUL 3
PEMROGRAMAN SOCKET DENGAN KONSEP MULTI-THREADING
(Menggunakan Pemrograman Java)

A. Tujuan

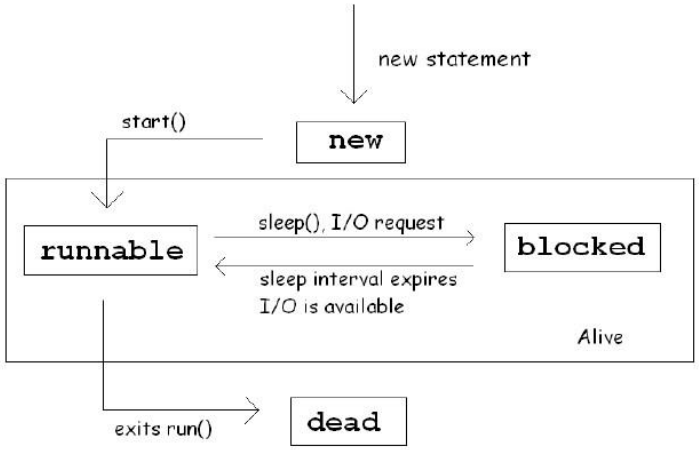
1. Menjelaskan mengenai kosep pemrograman Multi-threading
2. Membangun aplikasi socket dengan paket TCP/UDP dengan konsep multi-threading

B. Pendahuluan

Pemrograman Thread

Suatu proses dikontrol oleh paling sedikit satu thread. Namun, sebagian besar proses yang ada sekarang biasanya dijalankan oleh beberapa buah thread. Multithreading adalah sebuah mekanisme di mana dalam suatu proses, ada beberapa thread yang mengerjakan tugasnya masing-masing pada waktu yang bersamaan. Contohnya, sebuah web browser harus menampilkan sebuah halaman yang memuat banyak gambar. Pada program yang single-threaded, hanya ada satu thread untuk mengatur suatu gambar, lalu jika gambar itu telah ditampilkan, barulah gambar lain bisa diproses. Dengan multithreading, proses bisa dilakukan lebih cepat jika ada thread yang menampilkan gambar pertama, lalu thread lain untuk menampilkan gambar kedua, dan seterusnya, di mana thread-thread tersebut berjalan secara paralel.

Saat sebuah program Java dieksekusi, yaitu saat main() dijalankan, ada sebuah thread utama yang bekerja. Java adalah bahasa pemrograman yang mendukung adanya pembentukan thread tambahan selain thread utama tersebut. Thread dalam Java diatur oleh Java Virtual Machine (JVM) sehingga sulit untuk menentukan apakah thread Java berada di user-level atau kernel-level. Diagram state pemrograman multithread pada java dapat ditunjukkan sebagai berikut :



Suatu thread bisa berada pada salah satu dari status berikut:

1. *New*. Thread yang berada di status ini adalah objek dari kelas Thread yang baru dibuat, yaitu saat instansiasi objek dengan statement new. Saat thread berada di status new, belum ada sumber daya yang dialokasikan, sehingga thread belum bisa menjalankan perintah apapun.
2. *Runnable*. Agar thread bisa menjalankan tugasnya, method start() dari kelas Thread harus dipanggil. Ada dua hal yang terjadi saat pemanggilan method start(), yaitu alokasi memori untuk thread yang dibuat dan pemanggilan method run(). Saat method run() dipanggil, status thread berubah menjadi runnable, artinya thread tersebut sudah memenuhi syarat untuk dijalankan oleh JVM. Thread yang sedang berjalan juga berada di status runnable.
3. *Blocked*. Sebuah thread dikatakan berstatus blocked atau terhalang jika terjadi blocking statement, misalnya pemanggilan method sleep(). sleep() adalah suatu method yang menerima argumen bertipe integer dalam bentuk milisekon. Argumen tersebut menunjukkan seberapa lama thread akan "tidur". Selain sleep(), dulunya dikenal method suspend(), tetapi sudah disarankan untuk tidak digunakan lagi karena mengakibatkan terjadinya deadlock. Di samping blocking statement, adanya interupsi M/K juga dapat menyebabkan thread menjadi blocked. Thread akan menjadi runnable kembali jika interval method sleep()-nya sudah berakhir, atau pemanggilan method resume() jika untuk menghalangi thread tadi digunakan method suspend() atau M/K sudah tersedia lagi.
4. *Dead*. Sebuah thread berada di status dead bila telah keluar dari method run(). Hal ini bisa terjadi karena thread tersebut memang telah menyelesaikan pekerjaannya di method run(), maupun karena adanya pembatalan thread. Status jelas dari sebuah thread tidak dapat diketahui, tetapi method isAlive() mengembalikan nilai boolean untuk mengetahui apakah thread tersebut dead atau tidak.

Java Thread

Sebuah thread mulai dibuat, ketika dilakukan pembuatan objek baru dari class java.lang.Thread. Sebuah objek Thread mewakili thread sesungguhnya dalam interpreter java dan juga memberikan penangan control dan sinkronisasi eksekusinya. Sebuah objek yang diperlakukan sebagai sebuah thread harus menerapksi dari sebuah interface java.lang.runnable. Dalam interface ini menyediakan suatu method kunci run().

```
public interface Runnable{
        Abstract public void run();
}
```

Sebuah thread dimulai siklus hidupnya dengan memanggil method run(). Untuk membuat sebuah class yang akan bertindak sebagai sebuah thread, didefinisikan dengan bentuk sebagai berikut :

```
Class namaClass implements Runnable {
        Void run(){
        //implementasi kode.....
        }
}
```

Sedangkan cara pemanggilannya dengan

- Buat instance objek dari class yang mengimplemntasikan interface Runnable tersebut.
- Buat object dari class java.lang.Thread dengan parameter konstruktor object dari class yang mengimplemnetasikan interface Runnable
- Panggil method start() dari objek yang mengimplementasikan interface Runnable.

Cara lain untuk membuat thread adalah dengan membuat class turunan dari class lain yang telah menerapkan Runnable. Class Thread merupakan class yang mengimplementasikan interface Runnable. Berikut contoh :

```
class namaClass extends Thread {
        public void run(){
                //kode .............
        }
}
```

Pengontrolan Thread

Method start() digunakan untuk memulai sebuah thread dieksekusi. Method stop() digunakan untuk menghentikan eksekusi suatu thread. Method suspend() digunakan untuk menghentikan sementara waktu (pause) untuk kemudian dapat dilanjutkan kembali dengan menggunakan method resume(). Jika beban yang harus dikerjakan oleh thread berat/mahal, maka dapat digunakan method suspend() dan resume(). Method static sleep(n_msec) digunakan untuk menghentikan thread yang active selama n millisecond.

```
try {
        Thread.sleep(1000);
} catch(Exception e ){ }
```

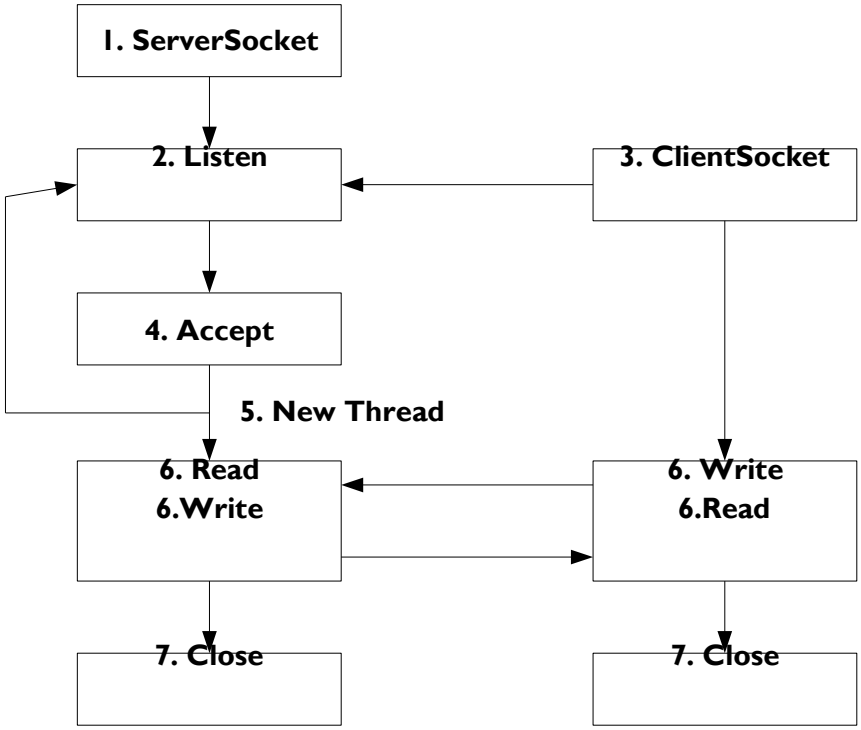
Dalam pembuatan aplikasi nantinya kita mungkin akan menggunakan beberapa thread untuk mengakses resource yang sama. Untuk menghindari masalah pada satu pengaksesan resource yang sama antar thread, maka satu thread harus tahu akan thread yang lain. Inilah yang disebut dengan synchronization. Struktur penggunaan synchronization adalah

[public] **synchronized** [type_data] namaMethod (parameter masukkan){

```
        //koding.....
}
```

Penggunaan Multithreading dalam pemrograman socket

Penggunaan multithreading untuk pemrograman socket sangat penting sekali, karena suatu server harus melayani banyak client. Seperti pada modul yang pertama, aplikasi socket untuk server hanya dapat melayani client tunggal, setalah itu server berhenti. Dengan menggunakan konsep pemrograman multithreading, maka setiap client koneksi ker server akan diciptakan suatu thread baru, demikian seterusnya untuk client selanjutnya. Sehingga server selalu running dan siap menerima request dari client (seperti program web server apache). Jika kita gambarakan dalam diagram blok sebagai berikut



C. Praktikum

Praktikum 1 :

```
package th;
public class myThread implements Runnable {
        private static int i=0;
        @Override
        public void run() {
                while(i<100){
                        System.out.print(i+" , ");
                        i++;
                }
        }
}
```

Berikutnya contoh pemanggilan clas myThread tersebut :

```
package th;
public class myThreadTest {
	public static void main(String[] args) {
		myThread ot = new myThread();
		Thread th = new Thread(ot);
		th.start();
		for(int j=0;j<10;j++){
			System.out.println("From Main "+j);
		}
	}
}
```

Praktikum 2:

```
package th;
public class myThread2 implements Runnable {
	private static int i=0;
	Thread myth;
	public myThread2(){
		myth = new Thread(this);
		myth.start();
	}
	@Override
	public void run() {
		while(i<100){
			System.out.print(i+", ");
			i++;
		}
	}
}
```

Buat program/method main untuk menjalankan thread tersebut.

Praktikum 3 :

```
package th;
public class myThread3 extends Thread {
	private static int i=0;
	public void run(){
		while(i<100){
			System.out.print(i + " , ");
			i++;
		}
	}
}
```

Buatlah program untuk menjalankan thread teraebut.

Praktikum 4 :

```
package th;
public class myThreadTest2 {
	public static void main(String[] args) {
		myThread ot  = new myThread();
		Thread th = new Thread(ot);
		th.start();
		for(int j=0;j<10;j++){
			if (j==5){
				try {
					th.sleep(1000);
				} catch(Exception e){
					e.printStackTrace();
				}
			}
			System.out.println("From Main "+j);
		}
	}
}
```

Amati cara kerja dari program tersebut !.

Praktikum 5 :

```
package th;
public class mhThreadTest3 {
	public static void main(String[] args) {
		boolean isSuspend = false;
		myThread ot = new myThread();
		Thread th = new Thread(ot);
		th.start();
		for(int j=0;j<10000;j++){
			if (j%250 ==0){
				if (isSuspend){
					th.resume();
					System.out.println("Setelah resume i : "+ot.getI());
				} else {
					th.suspend();
					System.out.println("Setelah suspend i : "+ot.getI());
				}
				isSuspend = ! isSuspend;
			}
		}
		th.stop();
```

```
        }
}
```

Praktikum 6 : (Sebelum menggunakan synchronized)

```
package ok;
public class TwoStrings {
        static void print(String str1,  String str2) {
                System.out.print(str1);
                try {
                        Thread.sleep(500);
                } catch (InterruptedException ie) {}
                System.out.println(str2);
        }
}
```

Method print(String str1, String str2) akan diakses oleh lebih dari satu thread, sedangkan program yang terdiri lebih dari satu thread yang akan mengakses method tersebut adalah

```
package ok;
public class PrintStringsThread implements Runnable {
        Thread thread;
          String str1, str2;
          PrintStringsThread(String str1, String str2) {
            this.str1 = str1;
            this.str2 = str2;
            thread = new Thread(this);
            thread.start();
          }
          public void run() {
            TwoStrings.print(str1, str2);
          }
          public static void main(String args[]) {
                    new PrintStringsThread("Hello ", "there.");
                    new PrintStringsThread("How are ", "you?");
                    new PrintStringsThread("Thank you ",  "very much!");
                }

}
```

Kemudian Anda jalankan, dan amati hasilnya

Praktikum 7 : (menggunakan synchronized)

```
package ok;
public class TwoStrings {
        synchronized static void print(String str1,  String str2) {
                System.out.print(str1);
                try {
                        Thread.sleep(500);
                } catch (InterruptedException ie) {}
                System.out.println(str2);
        }
}
```

Kemudian program yang menggunakan method dalam synchronized adalah sama dengan praktikum 6, coba Anda amati outputnya, kemudian anda cerna konsep synchronization tersebut.

Praktikum 8 : (Penggunaan pemrograman Multi-threading pada socket programming dengan paket TCP)
Class turuna dari class Thread :

```
package tcp;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.net.Socket;
public class Connection extends Thread {
        DataInputStream in;
        DataOutputStream out;
        Socket clientSocket;
        public Connection (Socket aClientSocket) {
           try {
                clientSocket = aClientSocket;
                in = new DataInputStream( clientSocket.getInputStream());
                out =new DataOutputStream( clientSocket.getOutputStream());
                this.start();
            } catch(IOException e)  {System.out.println("Connection:"+e.getMessage());}
        }
        public void run(){
           try {                                        // an echo server
                String data = in.readUTF();
                out.writeUTF(data);
           } catch(EOFException e) {
                System.out.println("EOF:"+e.getMessage());
           } catch(IOException e) {
```

```
                System.out.println("IO:"+e.getMessage());
            } finally{
                    try {
                            clientSocket.close();
                    }catch (IOException e){}
                }
        }
}
```

Kemudian TCP Servernya :

```
package tcp;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
public class TCPServer {
        public static void main(String[] args) {
                try{
                        int serverPort = 7896;
                        ServerSocket listenSocket = new ServerSocket(serverPort);
                        System.out.println("Server sedang running.....");
                        while(true) {
                                Socket clientSocket = listenSocket.accept();
                                Connection c = new Connection(clientSocket);
                        }
                } catch(IOException e) {
                        System.out.println("Listen :"+e.getMessage());
                }
        }
}
```

Kemudian TCP Clientnya :

```
package tcp;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;
public class TCPClient {
        public static void main(String[] args) {
                Socket s = null;
```

```
                try{
                    int serverPort = 7896;
                    String host="127.0.0.1";//host server, silahkan diganti jika server
tidak lokal
                    s = new Socket(host, serverPort);
                    DataInputStream in = new DataInputStream( s.getInputStream());
                    DataOutputStream out =       new DataOutputStream(
s.getOutputStream());
                    Scanner sc = new Scanner(System.in);
                    System.out.print("Inputkan Data yang akan di submit : ");
                    String input = sc.nextLine();
                    out.writeUTF(input);
                    String data = in.readUTF();
                    System.out.println("Received: "+ data) ;
                }catch (UnknownHostException e){
                    System.out.println("Sock:"+e.getMessage());
                }catch (EOFException e){
                    System.out.println("EOF:"+e.getMessage());
                }catch (IOException e){
                    System.out.println("IO:"+e.getMessage());
            }finally {
                    if(s!=null)
                            try {
                                    s.close();
                            }catch (IOException e){
                                    System.out.println("close:"+e.getMessage());
                                    }
                    }
        }
}
```

Kemudian Anda compile TCPClient dan TCPServer, setelah itu running untuk TCPServer baru kemudian Anda running TCPClientnya. Amati hasil dan prosesnya.

D. Latihan

1. Buatlah socket programming client server (TCP) dengan konsep multi-threading untuk proses penjumlahan, maksudnya adalah buat suatu socket server yang didalamnya terdapat suatu method untuk melakukan penjumlahan suatu data array bertype integer (ambil indeks maksimal 5)x, pada socket client mengirimkan data yang akan di jumlahkan, dan socket server malakukan proses penjumlahan dan mengirimkan hasilnya ke socket client.
2. Buatlah untuk kasus yang sama dengan paket Datagram (UDP).