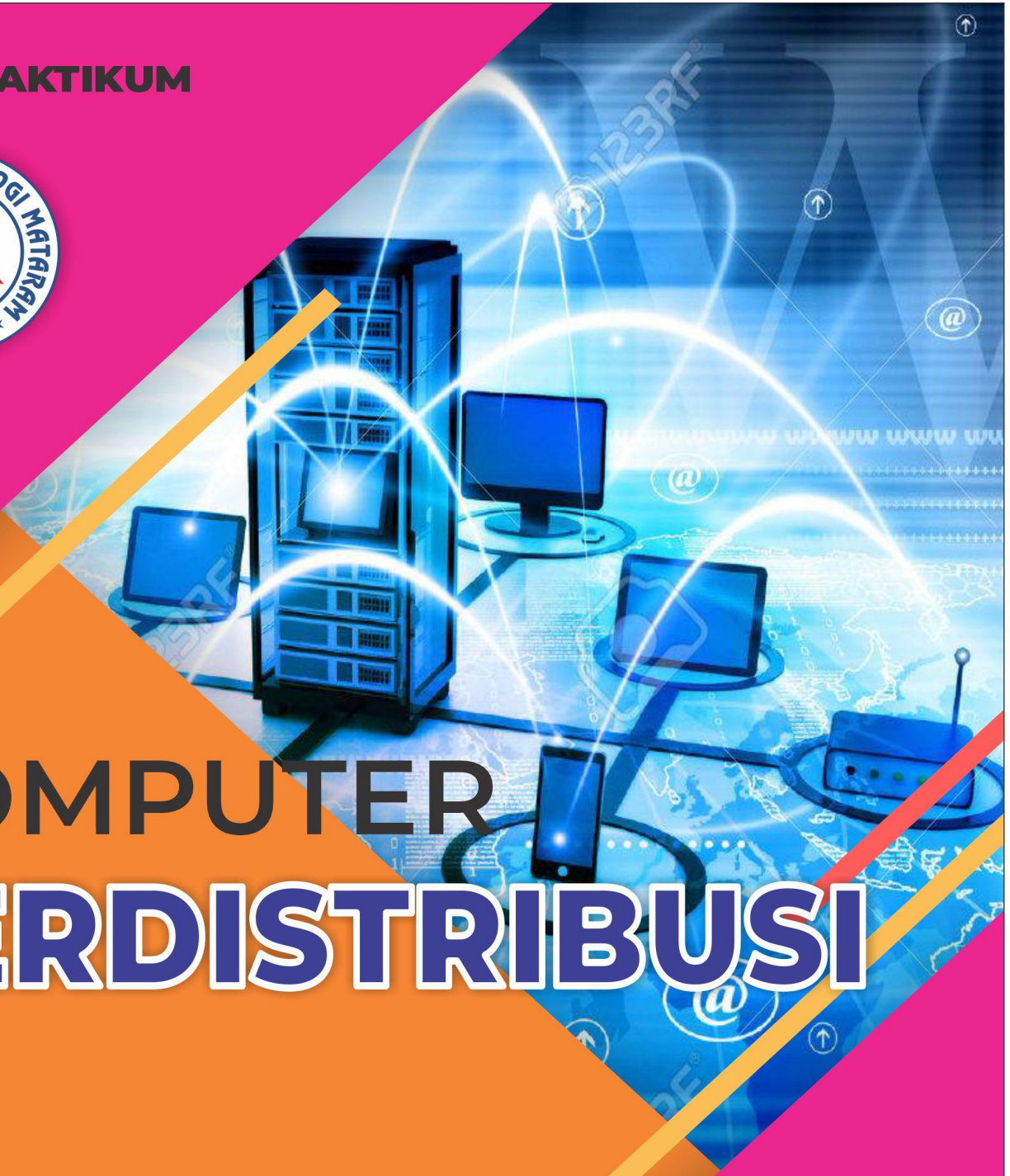
MODUL PRAKTIKUM



KOMPUTER TERDISTRIBUSI

UNIVERSITAS TEKNOLOGI MATARAM

2019

Jl. Kampus Universitas Teknologi Mataram, Kekalik, Mataram

MODUL 1

INTERNET ADDRESSING

1.1 Pembahasan

Salah satu konsep yang paling mendasar dalam pemrograman jaringan adalah pengalamatan jaringan (*network address*). Tanpa alamat jaringan, tidak akan ada cara untuk mengidentifikasi pengirim paket data atau dimana paket harus dikirim. Modul ini membahas representasi *IP address* dan *domain name* pada Java.

Sebuah host di Internet dapat direpresentasikan dalam format *dotted decimal* sebagai alamat IP atau sebagai nama host seperti www.uin-malang.ac.id. Dalam pemrograman jaringan menggunakan java, alamat tersebut direpresentasikan oleh kelas `java.net.InetAddress`. Tidak seperti kebanyakan kelas lain, di dalam kelas ini tidak ada public Constructor, akan tetapi di dalamnya terdapat dua static method yang mengembalikan nilai `InetAddress`. **Tabel 1.1** menunjukkan method utama yang ada dalam kelas `InetAddress`. Method lain dapat juga dipelajari pada situs resmi oracle.

Tabel 1.1 Beberapa method pada kelas `InetAddress`

Method	***Deskripsi***
`byte[] getAddress()`	Mengembalikan nilai Alamat IP dalam format byte.
`static InetAddress[] getAllByName (String hostname)` throws `java.net.UnknownHostException, java.lang.SecurityException`	Mengembalikan nilai `InetAddress` dari hostname dalam bentuk array. Kebanyakan mesin memiliki satu alamat IP, namun ada beberapa situasi dimana satu hostname dapat dipetakkan ke beberapa banyak mesin dan/atau beberapa alamat dalam satu mesin
`static InetAddress getByName (String hostname)` throws `java.net.UnknownHostException, java.lang.SecurityException`	Mengembalikan nilai `InetAddress` dari hostname tertentu yang dapat direpresentasikan baik dalam bentuk *text hostname* (misal: uin-malang.ac.id) maupun IP address dalam format *dotted decimal*.
`static InetAddress getLocalHost()` throws	Mengembalikan nilai `InetAddress` dari localhost

`java.net.UnknownHostException, java.lang.SecurityException`	
`String getHostAddress()`	Mengembalikan nilai String alamat IP dalam format *dotted decimal*.
`String getHostName()` throws `java.lang.SecurityManager`	Mengembalikan nilai String hostname dari `InetAddress`.
`boolean isMulticastAddress()`	Mengembalikan nilai *true* jika `InetAddress` tersebut adalah alamat multicast atau alamat kelas D
`String toString()`	Mengembalikan nilai String dari `InetAddress`. Disarankan menggunakan method `getHostAddress()` dan `getHostName()` untuk mengontrol jenis data yang diminta.

Contoh 1.1. Memperoleh alamat IP localhost

Berikut ini contoh program untuk mendapatkan alamat IP dari mesin lokal *(localhost)*

```
import java.net.InetAddress;
import java.net.UnknownHostException;

public class LocalHostDemo {

    public static void main(String[] args) throws UnknownHostException {
        System.out.println("Looking up local host");
        InetAddress localAddress = InetAddress.getLocalHost();
        System.out.println("Name: " + localAddress.getHostName());
        System.out.println("IP Address: " + localAddress.getHostAddress());
        System.out.println(localAddress);
    }
}
```

Output:

```
run:
Looking up local host
Name: Khiznia
IP Address: 192.168.29.1
Khiznia/192.168.29.1
BUILD SUCCESSFUL (total time: 1 second)
```

Contoh 1.2. Mendapatkan alamat dari host/ip dan cek koneksi

Contoh program ke-2 ini menunjukkan penggunaan `InetAddress` untuk mendapatkan alamat lain baik dalam bentuk text hostname maupun *dotted decimal*. Ditunjukkan pula cara melakukan check koneksi dengan menggunakan method `isReachable()`. Method `isReachable()` memberikan nilai "*true*" jika host tertentu dapat dicapai dengan batas timeout tertentu dalam satuan *millisecond*.

```
import java.net.InetAddress;

public class inAddress {

    public static void main(String[] args) {
        try {
            System.out.println("# Get hostname from current ip");
            String ip = "173.252.120.6";
            System.out.println("Hostname from ip \"" + ip + "\": "
                    + InetAddress.getByName(ip).getHostName());

            System.out.println("\n# Get host address from current name");
            String host = "www.google.com";
            System.out.println("Host/IP: "
                    + InetAddress.getByName(host).getHostAddress());
            System.out.println("Host/IP: "
                    + InetAddress.getByName(host));

            System.out.println("\n# Check connection");
            InetAddress ia = InetAddress.getByName(host);
            if (ia.isReachable(3000)) {
                System.out.println(ia +" is Reachable");
            }else{
                System.out.println(ia +" is unReachable");
            }
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
}
```

Output:

```
run:
# Get hostname from current ip
Hostname from ip "173.252.120.6": edge-star-shv-12-frc3.facebook.com

# Get host address from current name
Host/IP: 74.125.200.104
Host/IP: www.google.com/74.125.200.104

# Check connection
www.google.com/74.125.200.104 is unReachable
BUILD SUCCESSFUL (total time: 4 seconds)
```

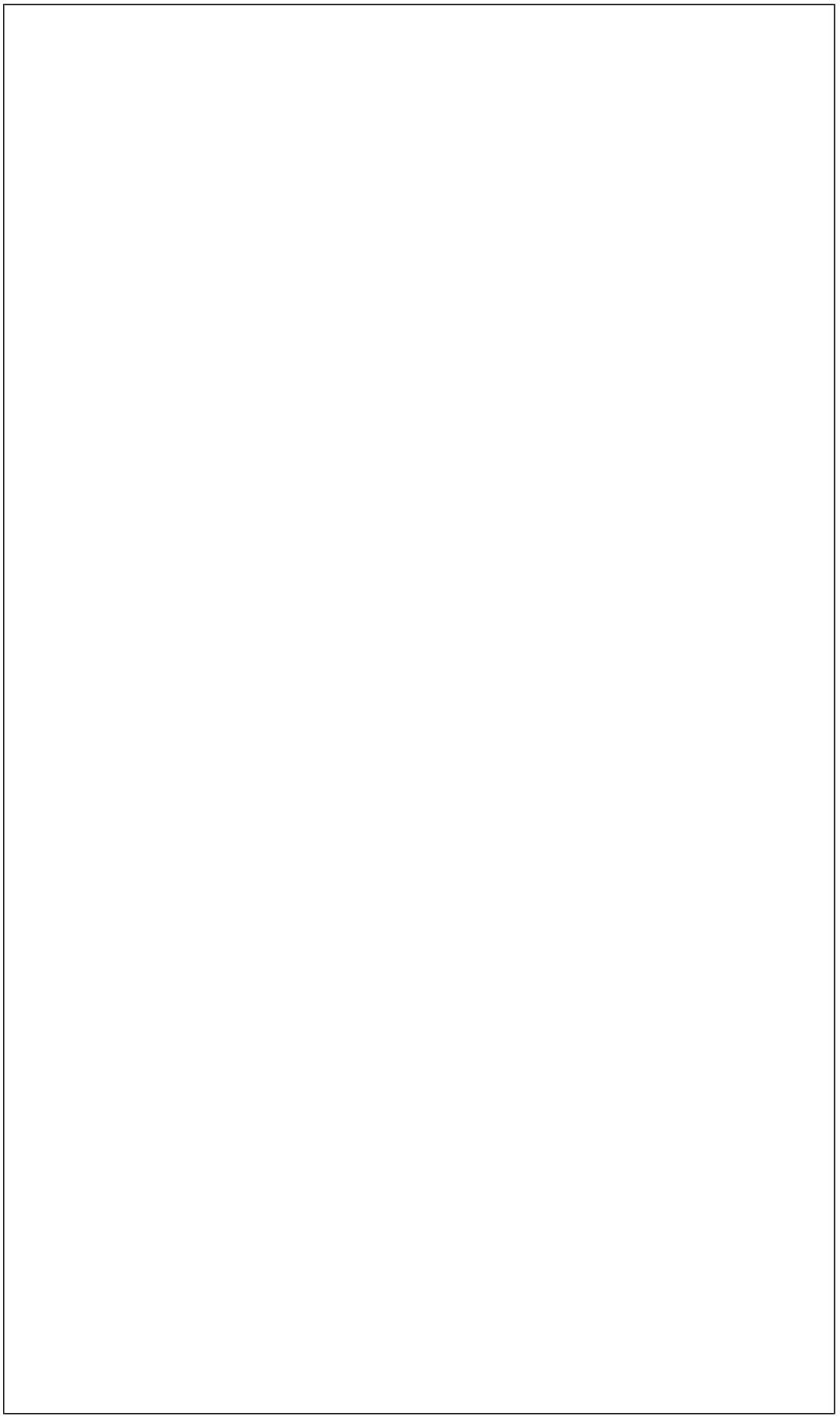
1.2 Praktikum

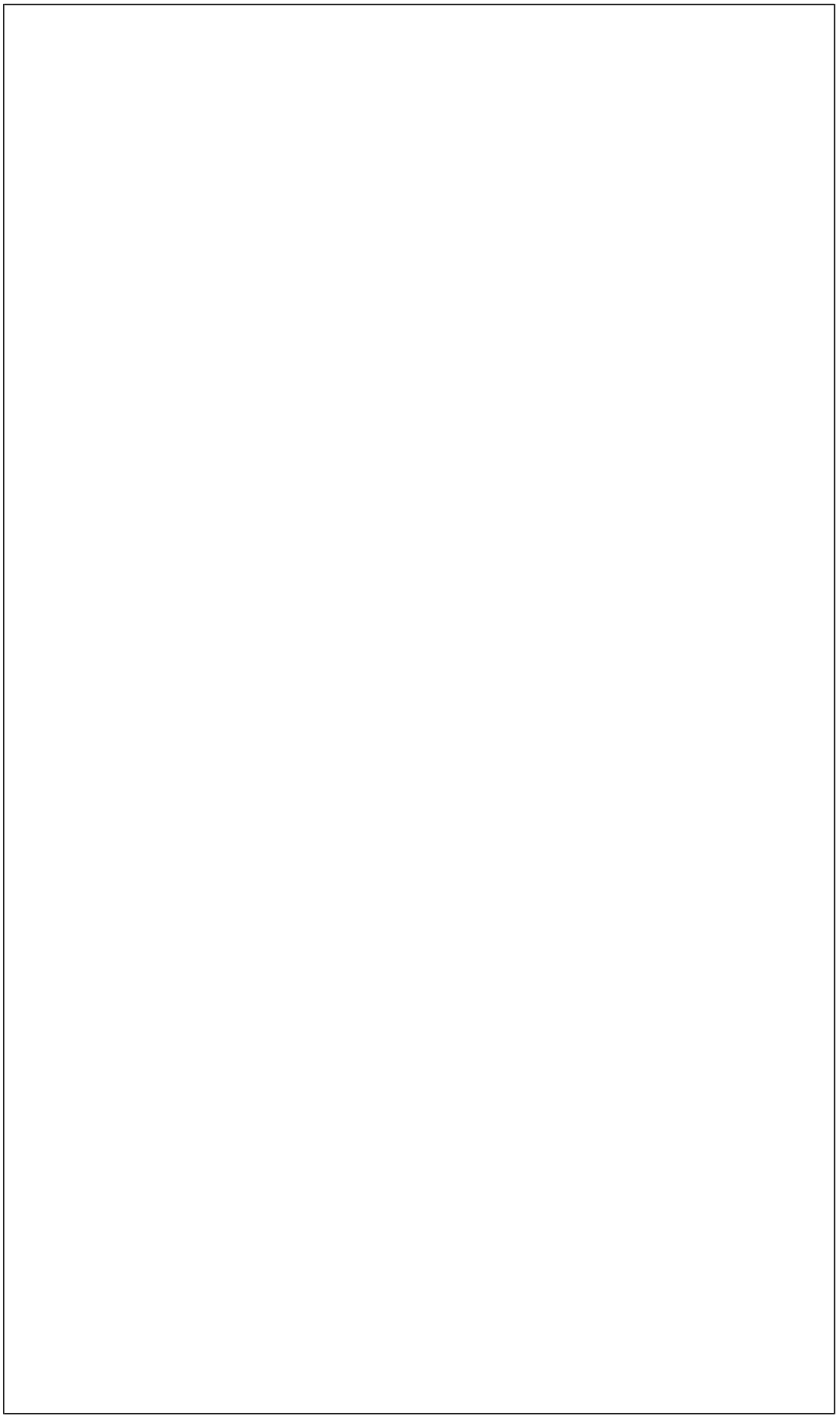
1. Tulis dan jalankan **Contoh 1.1** dan **Contoh 1.2** pada modul ini dan pahami tiap barisnya!
2. Buatlah sebuah program untuk ping alamat komputer lain. Inputan bisa dilakukan dengan dua cara:
    a. Inputkan alamat hostnya
    b. Inputkan nama hostnya
3. Adakalanya dalam suatu jaringan terdapat satu nama host dengan beberapa alamat host. Buatlah program untuk mendapatkan semua alamat host tersebut. Sebagai contoh:
    a. Nama Host : www.google.com
    b. Alamat Host :
        1. www.google.com/74.125.235.52
        2. www.google.com/74.125.235.51
        3. www.google.com/74.125.235.49
        4. www.google.com/74.125.235.48
        5. www.google.com/74.125.235.50

1.3 Tugas

1. Jelaskan konsep *internet addressing*!
2. Jelaskan konsep dan bagaimana kerja dari *Domain Name System* (DNS)!
3. Buatlah program aplikasi untuk mengecek koneksi jaringan lokal atau **scan ip** dari ip pertama hingga akhir secara berulang-ulang sehingga memungkinkan user dapat mengetahui PC mana yang aktif dan tidak sewaktu-waktu.
4. Buatlah program untuk mendapatkan MAC address!

Jawaban Tugas Pengembangan





MODUL 2

DATA STREAMS

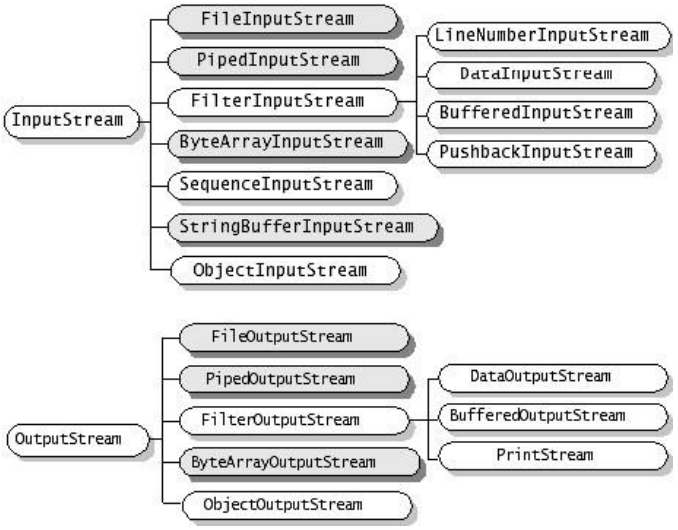
2.1 Pembahasan

Komunikasi melalui jaringan, melalui file, ataupun aplikasi, direpresentasikan dalam Java dengan menggunakan Stream. Konsep Stream sangat penting ketika membahas aplikasi jaringan. Hampir semua *network communication* (kecuali komunikasi UDP) dibangun menggunakan Steams.

2.1.1 Streams

Stream adalah aliran data yang berupa aliran data byte atau karakter dari device input ke device output pada saat program di eksekusi. Stream juga dapat diartikan sebagai representasi abstrak dari input dan output device dimana aliran data bytes akan ditransfer seperti transfer file ke dalam harddisk. Operasi input dan output pada java menggunakan konsep aliran data, yaitu aliran data dari device output dengan memanfaakan method `println()` pada objek `System.out`, dan device input pada objek `System.in`.

Proses stream untuk membaca diimplementasikan dengan memanggil kelas turunan dari superclass **`java.io.InputStream`**. Sedangkan proses stream untuk menulis diimplementasikan dengan memanggil kelas turunan dari superclass java.io.**`OutputStream`**. `InputStream` dan `OutputStream` merupakan kelas abstrak yang tidak bisa digunakan secara langsung, namun bisa menggunakan subclass dari masing-masing sesuai kebutuhan. Kelas turunan dari `InputStream` dan `OutputStream` ditunjukkan pada **Gambar 2.1**.



Gambar 2.1 Subclass dari input Stream dan output Stream

Input Stream

Diantara method pada kelas `InputStream` ditunjukkan pada **Tabel 2.1**.

Tabel 2.1 Method pada kelas InputStream

Method	Diskripsi
int available() throws java.io.IOException	Mengembalikan jumlah byte yang tersedia untuk dibaca
void close() throws java.io.IOException	Menutup input stream dan mengabaikan semua resource yang berhubungan dengan input stream
void mark(int readLimit)	Mencatat posisi tertentu pada input stream sehingga input stream dapat meninjau kembali pada urutan posisi tersebut dengan menggunakan method `InputStream.reset()`. Tapi tidak semua input stream mendukung fungsi ini.
boolean markSupported()	Mengembalikan nilai "true" jika input stream mendukung method `mark()` dan `reset()`.
int read() throws java.io.IOException	Mengembalikan byte data berikutnya dari stream. Ketika mencapai akhir dari stream maka akan mengembalikan nilai -1.
int read(byte[] byteArray) throws java.io.IOException	Membaca rangkaian byte dan menempatkan pada array byte yang ditentukan dengan memanggil method read() berulangkali hingga array terisi atau tidak ada lagi data yang diperoleh. Method ini mengembalikan jumlah byte yang berhasil dibaca atau -1 jika telah mencapai akhir stream.
int read(byte[] byteArray, int offset, int length) throws java. io.IOException, java.lang.IndexOutOfBoundsException	Membaca rangakain byte dan menempatkan pada array yang ditentukan. Berbeda dengan method sebelumnya, int read(byte[] byteArray), method ini memulai pengisian byte pada offset array yang ditentukan dan dengan panjang array tertentu.
void reset() throws java.io.IOException	Mengembalikan posisi input stream pada tanda (mark) yang telah ditetapkan.
long skip(long amount) throws java.io.IOException	Membaca, tetapi mengabaikan sejumlah byte tententu. Byte dibuang dan posisi input stream diperbarui.

Contoh 2.1. Menggunakan input stream

Berikut ini contoh implementasi input stream untuk menampilkan isi sebuah file. Setiap byte pada satu waktu dibaca dari file dan ditampilkan pada layar. Meskipun hal ini tidak cukup efisien dibandingkan dengan mempercepat kinerja menggunakan *buffering* –misalkan–, namun ini bisa memberi ilustrasi penggunaan *low-level-stream*.

```java
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;

public class FileInputStreamDemo {

    public static void main(String[] args) throws FileNotFoundException, IOException {
        //create an input stream and reading from specified file
        InputStream fileInput = new FileInputStream("e:/document.txt");

        //read the first byte of data
        int data = fileInput.read();

        //repeat: until end of file reached
        while (data != -1) {

            //convert byte to char and send it to standart output
            System.out.print((char)data);

            //read next byte
            data = fileInput.read();
        }
        //close the file input stream
        fileInput.close();

    }
}
```

Output:

```
run:
file input stream demoBUILD SUCCESSFUL (total time: 0 seconds)
```

Output Stream

Sejumlah output stream tersedia dalam paket java.io dengan berbagai tugas, misalkan menulis ke suatu struktur data seperti string dan array, atau ke bentuk file, dan sebagainya. Diantara method yang dapat digunakan pada kelas output stream ditunjukkan pada ***Tabel 2.2***.

Tabel 2.2 Method pada kelas output stream

Method	**Diskripsi**
void close() throws java.io.IOException	Menutup output stream
void flush() throws java.io.IOException	Melakukan “flush” terhadap suatu data yang tidak terkirim dan mengirimkan ke penerima output stream
void write(int byte) throws java.io.IOException	Menulis byte tertentu
void write(byte[] byteArray) throws java.io.IOException	Menulis isi byte array ke output stream. Seluruh isi array akan ditulis

void write(byte[] byteArray, int offset, int length) throws java.io.IOException	Menulis isi subset dari byte ke output stream. Dengan method ini maka developer dapat menentukan berapa banyak array yang dikirim serta bagiannya.

Contoh 2.2. Menggunakan output stream

Berikut ini contoh sederhana yang menunjukkan bagaimana penggunaan output stream.

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

public class FileOutputStreamDemo {

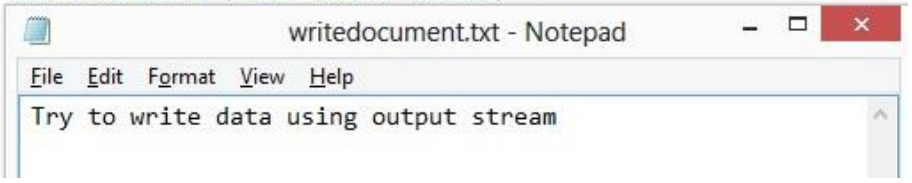
    public static void main(String args[]) throws FileNotFoundException, IOException {
        String destination = "e:/writedocument.txt";
        OutputStream output = new FileOutputStream(destination);
        System.out.println("Opened " + destination + " for writing.");

        String data = "Try to write data using output stream";
        for (int i = 0; i < data.length(); i++) {
            output.write((byte) data.charAt(i));
        }

        output.close();
        System.out.println("output stream closed");
    }
}
```

Output:

```
run:
Opened e:/writedocument.txt for writing.
output stream closed
BUILD SUCCESSFUL (total time: 2 seconds)
```



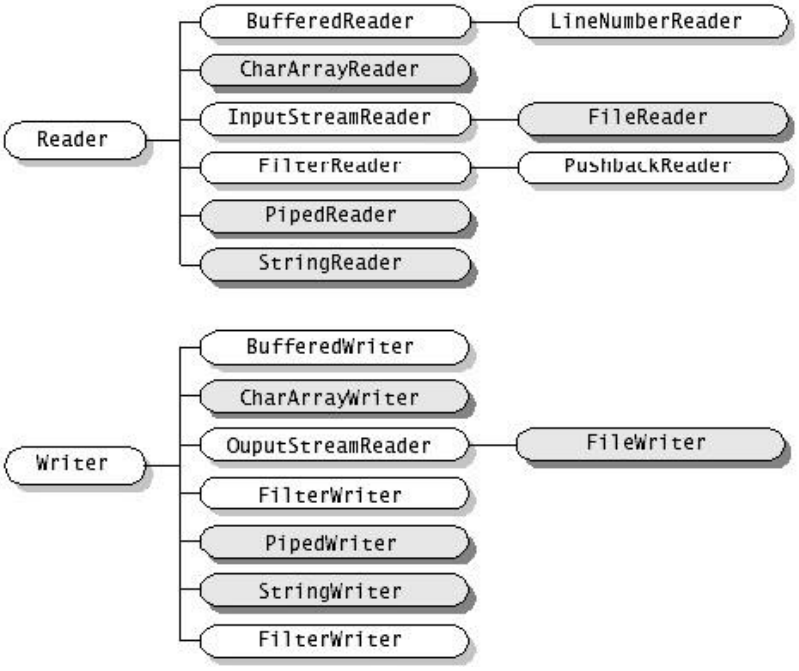
2.1.2 *Reader* dan *Writers*

Input Stream dan output Stream memang dapat digunakan untuk membaca dan menulis text, baik dalam bentuk byte maupun tipe data primitive. Namun, ada alternatif yang lebih baik dengan menggunakan *reader* dan *writers*. Reader dan writers dikenalkan pada JDK 1.1 agar lebih mendukung stream karakter Unicode (pembahasan tentang Unicode dapat diperoleh pada situs resmi dengan alamat http://www.unicode.org/).

Reader dan writers adalah alternatif yang lebih baik daripada input stream dan output stream ketika menggunakan text data. Input stream dan output stream bisa tetap digunakan jika hanya

memproses tipe data primitive. Namun untuk aplikasi yang memproses text data maka perlu menggunakan reader dan writers sehingga mendukung karakter Unicode.

Subclass dari `java.io.Reader` dan `java.io.Writer` ditunjukkan pada **Gambar 2.2**. Pada dasarnya, terdapat kesetaraan fungsi antara kelas pada `java.io.Reader` dengan kelas pada `java.io.InputStream`. Kelas `Reader` memiliki tanda method yang sama dengan kelas `InputStream`. Hanya terdapat beberapa perubahan kecil pada tanda method untuk mendukung karakter, bukan byte. Selain itu, method `available()` telah dihapus dan digantikan dengan method `ready()`.



Gambar 2.2 Subclass dari Reader dan Writer

Reader

Diantara method pada kelas reader ditunjukan pada **Tabel 2.3**.

Tabel 2.3 Method pada kelas Reader

Method	**Diskripsi**
void close() throws java.io.IOException	Menutup/mengakhiri *reader*
void mark(int amount) throws java.io.IOException	Menandai posisi tertentu pada reader dan menggunakan spesifik karakter tertentu sebagai *buffer*. Tidak semua reader mendukung method `mark(int)` dan `reset()`.
boolean markSupported()	Mengembalikan nilai "true" jika reader mendukung operasi method `mark(int)` dan `reset()`.

int read() throws java.io.IOException	Membaca dan mengembalikan sebuah karakter. Jika sudah mencapai akhir dari stream reader maka mengembalikan nilai -1.
int read(char[] characterArray) throws java.io.IOException	Mengisi array karakter dengan data. Method ini mengembalikan nilai int yang mewakili jumlah byte yang dibaca. Jika mencapai akhir dari stream reader maka mengembalikan nilai -1 dan array tidak dirubah.
int read(char[] characterArray int offset, int length) throws java.io.IOException	Mengisi subset dari array dengan data, mulai dari offset yang telah ditentukan dan dengan panjang tertentu. Method ini mengembalikan nilai int, mewakili jumlah bte yang dibaca, atau -1 jika tidak ada byte yang bisa diperoleh.
boolean ready() throws java.io.IOException	Mengembalikan nilai "true" jika da data yang tersedia
void reset() throws java.io.IOException	Mencoba untuk mengembalikan stream reader dengan memindahkan kembali ke posisi awal yang sudah ditentukan dengan method `mark(int)`
long skip(long amount) throws java.io.IOException	Membaca dan mengabaikan sejumlah karakter yang ditentukan. Method ini mengembalikan jumlah karakter yang berhasil dilewati.

Contoh 2.3. Kombinasi Input Streams dan Readers

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class InputStreamToReaderDemo {

    public static void main(String[] args) {
        try {
            System.out.print("Please enter your name : ");

            // Get the input stream representing standard input
            InputStream input = System.in;

            // Create an InputStreamReader
            InputStreamReader reader = new InputStreamReader(input);

            // Connect to a buffered reader, to use the readLine() method
            BufferedReader bufReader = new BufferedReader(reader);
            String name = bufReader.readLine();
            System.out.println("Pleased to meet you, " + name);
        } catch (IOException ioe) {
            System.err.println("I/O error : " + ioe);
        }

    }
}
```

Output:

```
run:
Please enter your name : Khadijah
Pleased to meet you, Khadijah
BUILD SUCCESSFUL (total time: 3 seconds)
```

Writer

Diantara method yang ada pada kelas writer ditunjukkan pada

Tabel *2.4*. Pada dasarnya method pada kelas Writer memiliki kemiripan fungsi dengan method pada kelas Output Stream namun dengan penyesuaian sehingga dapat digunakan untuk karakter Unicode.

Tabel 2.4 Method pada kelas Writer

Method	**Diskripsi**
void close() throws java.io.IOException	Method flush() untuk mengirim data buffer kemudian menutup writer
void flush() throws java.io.IOException	Melakukan "flush" terhadap data yang tidak terkirim kemudian mengirimkannya.
void write(int character) throws java.io.IOException	Menulis karakter tertentu
void write(char[] charArray) throws java.io.IOException	Membaca semua konten dari array karakter yang ditentukan dan menuliskannya
void write(char[] charArray int offset, int length) throws java.io.IOException	Membaca subset dari array karakter, mulai dari offset yang ditentukan hingga panjang tertentu kemudian menuliskannya
void write(String string) throws java.io.IOException	Menulis string tertentu
void write(String string, int offset, int length) throws java.io.IOException	Menulis sebuah subset string, dimulai dari offset tertentu hingga panjang yang ditentukan

Contoh 2.4. Kombinasi Output Streams dan Writers

```
import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;

public class OutputStreamToWriterDemo {

    public static void main(String args[]) {
        try {
            // Get the output stream representing standard output
            OutputStream output = System.out;

            // Create an OutputStreamWriter
            OutputStreamWriter writer = new OutputStreamWriter(output);

            // Write to standard output using a writer
            writer.write("Hello world");

            // Flush and close the writer, to ensure it is written
            writer.flush();
            writer.close();
        } catch (IOException ioe) {
            System.err.println("I/O error : " + ioe);
        }
    }
}
```

Output:

```
run:
Hello worldBUILD SUCCESSFUL (total time: 0 seconds)
```

2.1.3 Membaca *Image File* dan *Audio File*

Berikut ini dipaparkan contoh program untuk membaca file berupa gambar (Contoh 2.5) dan berupa suara (Contoh 2.6).

Contoh 2.5. Membaca Image File

Untuk membaca input berupa data image, dapat menggunakan method `ImageIO.read(sourceimage)`. Berikut ini contoh program yang akan menunjukkan bagaimana menangani proses input data image.

```java
import java.awt.*;
import java.io.*;
import javax.imageio.*;
import javax.swing.*;

public class ImageInputStreamDemo {
public static void main(String [] args ){
        Image image = null;
        try {
            // you may try both of this techniques

            // Read from a file
            File sourceimage = new File("e:/be the one.jpg");
            image = ImageIO.read(sourceimage);

            // Read from an input stream
            InputStream is = new BufferedInputStream(new FileInputStream("e:/be the one.jpg"));
            image = ImageIO.read(is);
        } catch (IOException e) {
        }
        // Use a label to display the image
        JFrame frame = new JFrame();
        JLabel label = new JLabel(new ImageIcon(image));
        frame.getContentPane().add(label, BorderLayout.CENTER);
        frame.pack();
        frame.setVisible(true);
    }
}
```

Output:

Contoh 2.6. Membaca Audio File

```
import java.io.FileInputStream;
import java.io.InputStream;
import sun.audio.AudioPlayer;
import sun.audio.AudioStream;

public class AudioInputStreamDemo {

    public static void main(String[] args) throws Exception {
        // open the sound file as a Java input stream
        String audioFile = "e:/bubbles.wav";
        InputStream in = new FileInputStream(audioFile);

        // create an audiostream from the inputstream
        AudioStream audioStream = new AudioStream(in);

        // play the audio clip with the audioplayer class
        AudioPlayer.player.start(audioStream);
    }
}
```

Output:

~ ~ Suara bubble ~ ~

2.2 Praktikum

1. Tulis dan jalankan **Contoh 2.1** dan **Contoh 2.2** pada modul ini dan pahami tiap barisnya!
2. Tulis dan jalankan **Contoh 2.3** dan **Contoh 2.4** pada modul ini dan pahami tiap barisnya!
3. Buatlah program untuk membaca input text dari console kemudian menuliskan kedalam sebuah file pada directory!
4. Buatlah program yang membaca input text dari console. Inputan berupa satu kalimat atau paragraph. Program mampu membaca dan menampilkan secara berulang dan akan keluar ketika user mengetikkan kata “exit”. Contoh:

```
run:
Masukkan text. Ketik "exit" untuk keluar
1. mengawali dengan basmalah
anda menulis: 1. mengawali dengan basmalah

2. menjalani dengan ikhtiar dan tawakkal 'alallah
anda menulis: 2. menjalani dengan ikhtiar dan tawakkal 'alallah

3. mengakhiri dengan hamdalah
anda menulis: 3. mengakhiri dengan hamdalah

exit
anda menulis: exit

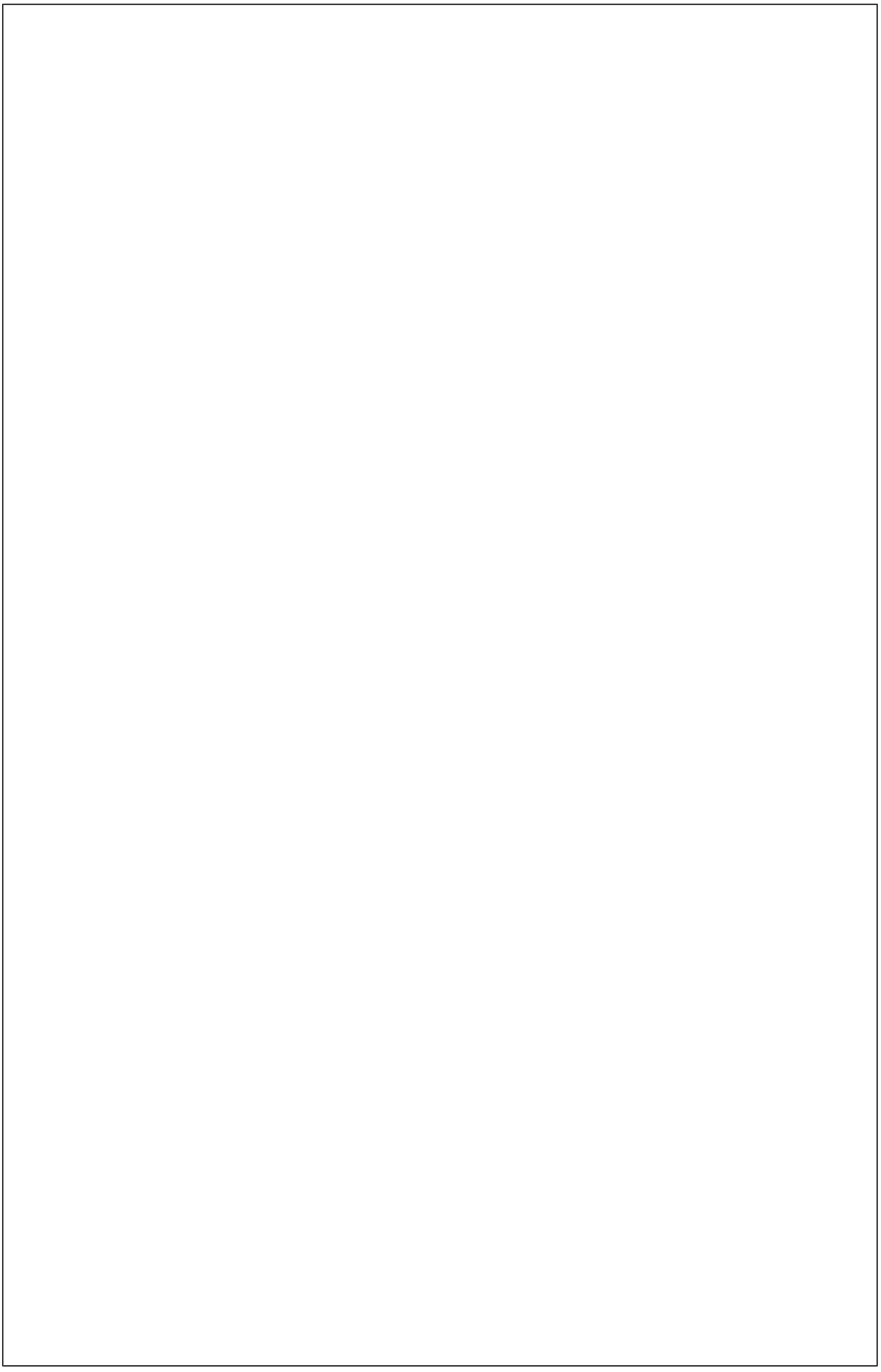
terimakasih
BUILD SUCCESSFUL (total time: 43 seconds)
```

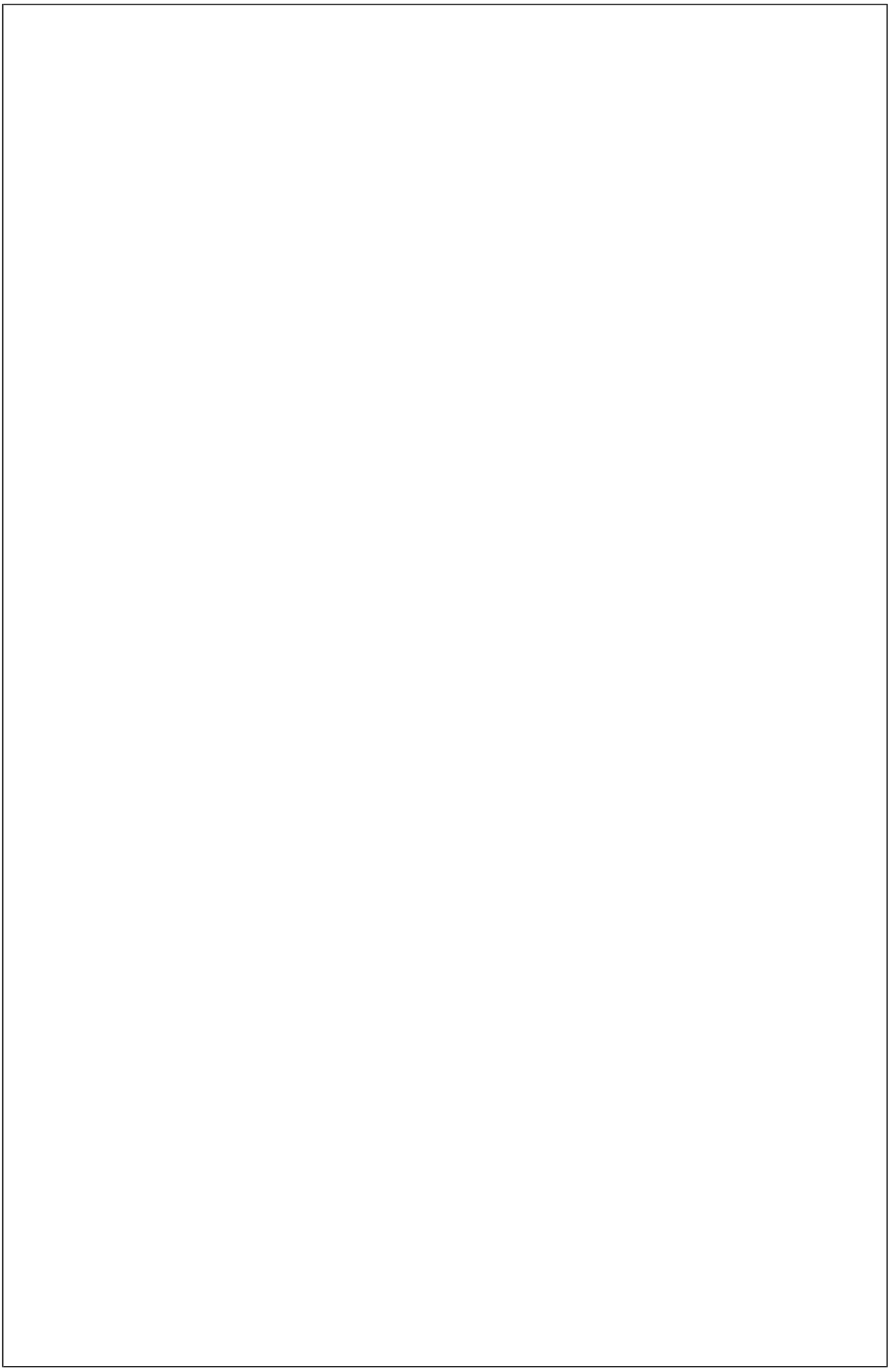
5. Tulis dan jalankan **Contoh 2.5** dan **Contoh 2.6** pada modul ini dan pahami tiap barisnya!

2.3 Tugas

1. Buatlah program untuk membuat file baru pada directory kemudian mengkopi isi sebuah file lain yang sudah ada ke file yang baru dibuat tersebut!
2. Buatlah program dengan GUI untuk membaca dan menulis Image file dari dan pada file yang dipilih!
3. Buatlah program dengan GUI untuk membaca dan menulis audio file dari dan pada file yang dipilih!

Jawaban Tugas Pengembangan





MODUL 3

OBJECT SERIALIZATION

3.1 Pembahasan

Serialization adalah proses mengkonversi sebuah *object* menjadi *stream* (urutan bytes). Objek yang telah diserialisasi dapat dikirim melalui jaringan, disimpan dalam memori, atau disimpan kedalam suatu file. Sebaliknya, deserialization adalah proses merekonstruksi sebuah *object* yang telah diserialisasi sebelumnya, yaitu mengembalikan urutan bytes kedalam bentuk objek.

Suatu objek agar dapat diserialisasi maka kelas object tersebut harus mengimplementasikan interface **`java.io.Serializable`** atau **`java.io.Externalizable.`** Interface **`Serializable`** tidak memiliki method untuk di-*override* tetapi harus diimplementasikan sebagai indikasi bahwa kelas objek tersebut dapat diserialisasi. Semua data atau object pada kelas yang mengimplementasikan interface **`Serializable`** dapat diserialisasi kecuali jika pada data atau object tersebut ditambahkan keyword **`transient`**. Suatu tipe data primitive ataupun object yang ditandai dengan keyword **`transient`** tidak akan diserialisasi. Sebagai contoh pada kelas UserAccount field password tidak dapat diserialisasi.

```
public class UserAccount implements java.io.Serializable {
        protected String username;
        protected transient String password;
        public UserAccount()
        {
                ......
        }
}
```

Keyword **`transient`** juga dapat digunakan untuk suatu field yang selalu diupdate, seperti timer. Tidak perlu melakukan serialisasi pada field yang menyimpan data tampilan GUI. Tampilan GUI dapat dibuat secara dinamis tanpa serialisasi.

Reading and Writing Object to Stream

Inti dari serialiasi adalah untuk menuliskan objek kedalam bentuk stream dan membacanya kembali. Hal ini dapat dilakukan dengan menggunakan kelas **`java.io.ObjectOutputStream`** dan **`java.io.ObjectInputStream`**. Kelas tersebut mampu menuliskan objek yang bisa diserialisasi kedalam suatu output stream dan membaca kembali dari suatu input Stream.

Kelas **`ObjectInputStream`** digunakan untuk membaca sebuah serial objek dari sebuah aliran byte. Sebalikanya, kelas **`ObjectOutputStream`** digunakan untuk melakukan serialisasi sebuah objek menjadi sebuah aliran byte. **`ObjectOutputStream`** dapat dihubungkan dengan output stream tertentu seperti sebuah file atau sebuah stream jaringan untuk transmisi melalui internet.

Konstruktor dan method yang paling sering digunakan pada kelas **`ObjectOutputStream`** dan **`ObjectInputStream`** disebutkan pada **Tabel 3.1**.

Tabel 3.1 Konstruktor dan Method pada kelas ObjectInputStream dan ObjectOutputStream

Konstruktor/Method	**Diskripsi**
ObjectInputStream(InputStream input) throws java.io.IOException	Membuat membuat aliran input objek yang terhubung dengan suatu input stream tertentu, seperti **`FileInputStream`**, **`ByteArrayInputStream`**, dsb.
public final Object readObject() throws java.io.OptionalDataException java.io.IOException, java.lang.ClassNotFoundException	Digunakan untuk melakukan deserialization, yaitu untuk membaca sebuah serial objek dari stream dan merekonstruksi ke bentuk semula (kecuali bagi field **`transient`** dan **`static`**)
ObjectOutputStream (OutputStream output) throws java.io. IOException	Membuat sebuah aliran output objek yang mampu menserialisasi objek kepada output stream tertentu.
void writeObject (Object object) throws java.io.IOException java.io.InvalidClassException, java.io.NotSerializableException	Menuliskan objek tertentu pada output stream. Semua variable yang tidak ditandai dengan **`transient`** atau **`static`** akan ditulis.

Contoh 3.1. Simple Serialization

Berikut ini contoh sederhana object serialization. Program ini dapat menyimpan sebuah objek ke dalam sebuah file kemudian membaca dan menampilkan objek tersebut pada console. Pada contoh ini, objek yang diserialisasi disimpan pada sebuah file, sehingga konstruktor **`ObjectOutputStream`** dihubungkan dengan **`FileOutputStream`**. Sedangkan untuk membaca file dan melakukan deserialisasi, konstruktor **`ObjectInputStream`** dihubungkan dengan **`FileInputStream`**.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

public class ParticipantSer implements Serializable {

    private String firstName;
    private String lastName;
    private int age;

    public ParticipantSer(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    public void printData() {
        System.out.println("Nama\t: " + firstName + " " + lastName);
        System.out.println("Usia\t: " + age);
    }
```

```
    //to serialize object and save in a file
    public void saveObject(ParticipantSer Obj) {
        try {
            FileOutputStream fos = new FileOutputStream("data.ser");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(Obj);
            oos.flush();
            fos.close();
        } catch (IOException ioe) {
            System.out.println("a problem accured during serialize.\n "
                    + ioe.getMessage());
        }
    }

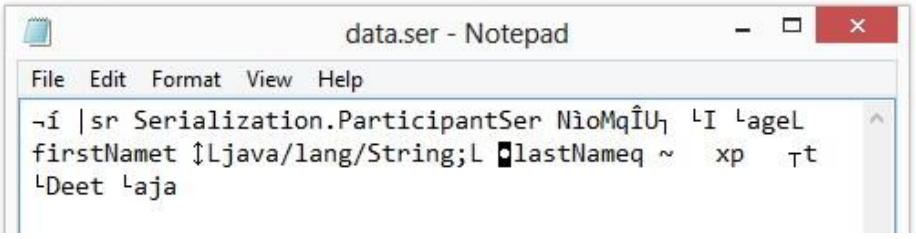
    //read a file and deserialize into object
    public void readObject(ParticipantSer Obj) {
        try {
            FileInputStream fis = new FileInputStream("data.ser");
            ObjectInputStream ois = new ObjectInputStream(fis);
            Obj = (ParticipantSer) ois.readObject();
            Obj.printData();
            fis.close();
        } catch (IOException | ClassNotFoundException ex) {
            System.out.println("a problem accured during deserialize.\n" + ex);
            System.exit(1);
        }
    }

    public static void main(String[] args) {
        ParticipantSer is = new ParticipantSer("Dee", "aja", 22);
        is.saveObject(is);
        is.readObject(is);
    }

}
```

Output:

```
run:
Nama    : Dee aja
Usia    : 22
BUILD SUCCESSFUL (total time: 1 second)
```



Contoh 3.2. Serialization menggunakan ArrayList

Pada contoh sebelumnya (Contoh 3.1), telah ditunjukkan bagaimana melakukan serialisasi terhadap sebuah objek dan menyimpan kedalam sebuah file. Berikut ini contoh lain dari serialisasi terhadap class object dengan ArrayList. Terdapat tiga kelas pada contoh program ini, yaitu:

a. *Participant.java*. Kelas ini sebagai class object yang mengimplementasikan interface Serializable.
b. *SerializationApp.java*. Kelas ini sebagai kelas utama yang memiliki main method untuk menjalankan program. Inisiasi variable dan pemanggilan method untuk serialize dan deserialize dilakukan pada kelas ini.
c. *SerializationDemo.java*. Method serialize dan deserialize dibangun dalam kelas ini.

Berikut ini langkah implementasi serialization menggunakan ArrayList. Tulis listing program sesuai dengan tahapan berikut untuk memahami langkah implementasinya.

a. Buat class object dengan nama ***Participant*** serta *implements Serializable*. Deklarasikan variable *firstName*, *lastName* berupa *String* dan *age* berupa *integer*. Buat kontruktor dan *setter-getter method*. Untuk menampilkan data, lakukan *override* terhadap method *toString*. Berikut ini listing program untuk kelas *Participant*:

```
import java.io.Serializable;

public class Participant implements Serializable {
    private final String firstName;
    private final String lastName;
    private int age;

    public Participant(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return firstName +" "+ lastName + " (" + age + ")";
    }

}
```

b. Buat kelas ***SerializationApp***. Pada *main method*, deklarasikan sebuah List dengan nama *participants* yang akan menampung sejumlah object Participant. Tambahkan beberapa object pada list *participants* dan tampilkan.

```
import java.util.ArrayList;
import java.util.List;

public class SerializationApp {

    public static void main(String[] args) {
        List<Participant> participants = new ArrayList<Participant>();
        participants.add(new Participant("Dee", "aja", 22));
        participants.add(new Participant("Ami", "Fahmi", 21));
        participants.add(new Participant("Haya", "Hayati", 20));
        participants.add(new Participant("Aya", "Hayati", 19));

        System.out.println("Participants : " + participants);

    }
}
```

Output tampilan list object *participants*:

```
run:
Participants : [Dee aja (22), Ami Fahmi (21), Haya Hayati (20), Aya Hayati (19)]
BUILD SUCCESSFUL (total time: 6 seconds)
```

c. Buat method ***serialize*** pada kelas ***SerializationDemo***. Method serialize berisi baris program untuk melakukan serialisasi terhadap list object participant kemudian menyimpan dalam sebuah file.

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.List;

public class SerializationDemo {

    public void serialize(List<Participant> pList, String fileName) {
        try (ObjectOutputStream out =
                new ObjectOutputStream(new FileOutputStream(fileName))) {
            out.writeObject(pList);
        } catch (IOException ex) {
            System.out.println("a problem accured during serialization \n"
                    + ex.getMessage());
        }
    }

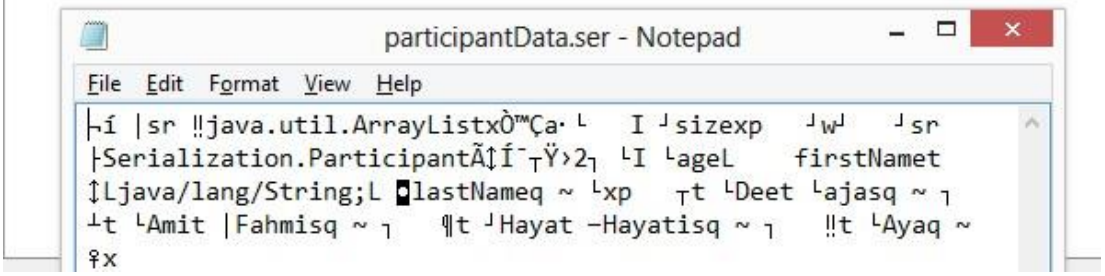
}
```

Method *serialize* tersebut dapat dipanggil dan dijalankan, dengan menambahkan baris program berikut pada **main method** kelas ***SerializationApp***:

```
        SerializationDemo demo = new SerializationDemo();
        demo.serialize(participants, "participantData.ser");
        System.out.println("serialization is done");
```

Output tahap serialisasi:

```
run:
Participants : [Dee aja (22), Ami Fahmi (21), Haya Hayati (20), Aya Hayati (19)]
serialization is done
BUILD SUCCESSFUL (total time: 4 seconds)
```



d. Tambahkan method ***deserialize*** pada kelas ***SerializationDemo***. Hingga tahap sebelumnya, program yang dibuat sudah mampu menyimpan objek serialisasi kedalam sebuah file. Tahap ini menunjukkan cara untuk melakukan deserialize, yaitu membaca dan merekonstruksi objek yang diserialisasi sebelumnya.

Berbeda dengan method *serialize* yang merupakan void method, method *deserialize* merupakan method yang mengembalikan nilai dari *list participant*. Method *deserialize* ini berisi baris program untuk membaca suatu file dan merekonstruksi byte stream dari file tersebut kedalam bentuk objek. Object disimpan dalam sebuah list. Berikut ini baris program untuk method *deserialize*.

```
public List<Participant> deserialize(String fileName) {
        List<Participant> pList=null;
        try (ObjectInputStream in =
                new ObjectInputStream(new FileInputStream(fileName))){
            pList = (List<Participant>) in.readObject();
        }catch(IOException | ClassNotFoundException ex){
            System.out.printf("a problem accured deserializing %s%n", fileName);
            System.out.println(ex.getMessage());
        }
        return pList;
    }
```

Method *deserialize* tersebut dapat dipanggil dan dijalankan dengan menambahkan baris program berikut pada **main method** kelas ***SerializationApp***:

```
        System.out.println("Deserialize obejct...");
        List<Participant> newList = demo.deserialize("participantData.ser");
        System.out.println("New List: " + newList);
```

Output akhir:

```
run:
Participants : [Dee aja (22), Ami Fahmi (21), Haya Hayati (20), Aya Hayati (19)]
serialization is done
Deserialize obejct...
New List: [Dee aja (22), Ami Fahmi (21), Haya Hayati (20), Aya Hayati (19)]
BUILD SUCCESSFUL (total time: 2 seconds)
```

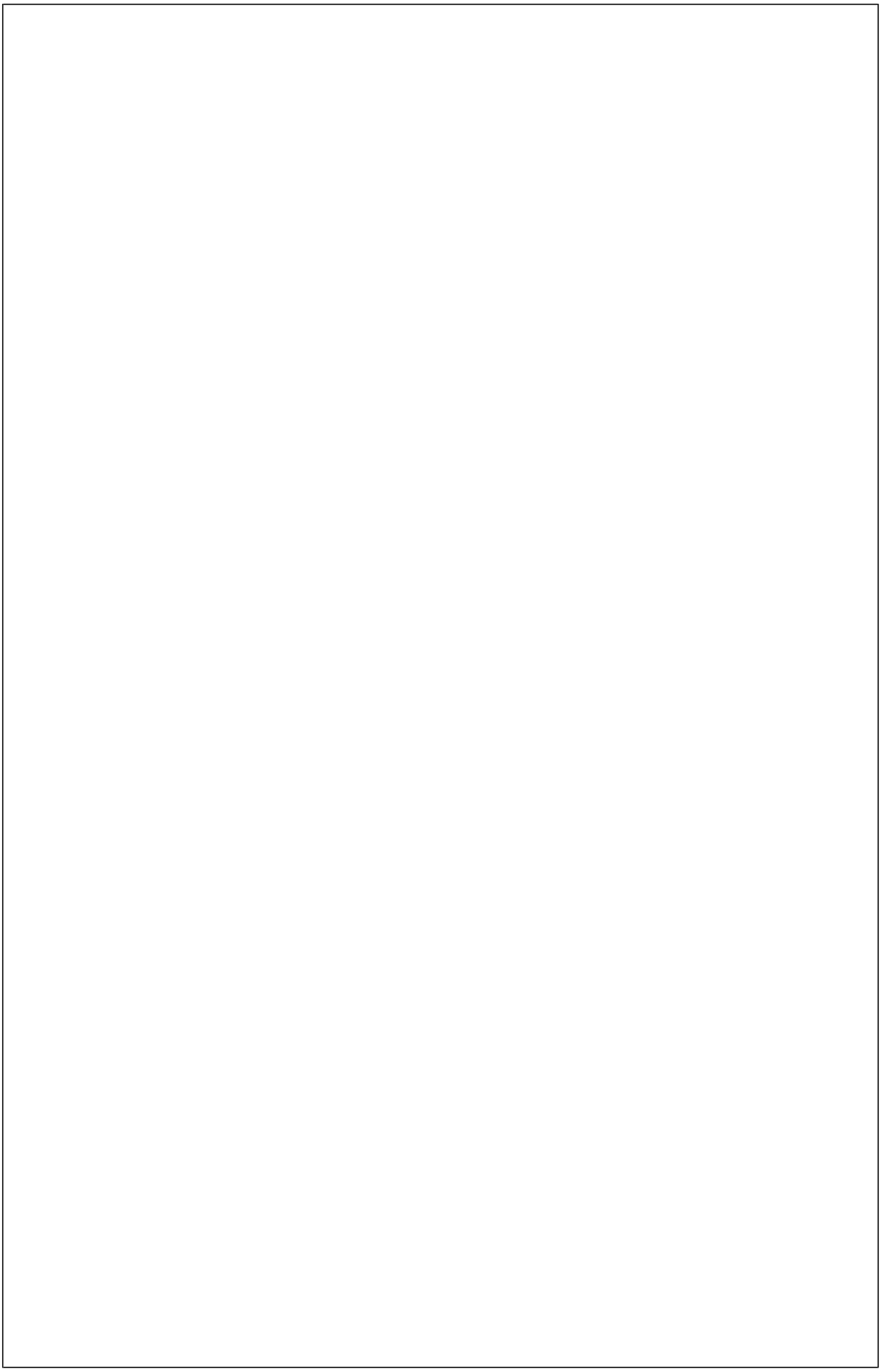
3.2 Praktikum

1. Tulis dan jalankan **Contoh 3.1** pada modul ini dan pahami tiap barisnya!
2. Tulis dan jalankan **Contoh 3.2** sesuai langkah pada modul ini dan pahami tiap barisnya!
3. Buat program semisal dengan **Contoh 3.2**, dimana data participant merupakan input user dari keyboard.

3.3 Tugas

1. Sebutkan dan jelaskan penggunaan object Serialization pada pemrograman jaringan!
2. Buatlah program penyimpanan data mahasiswa dengan mengimplemantasikan Object Serialization. Data mahasiswa terdiri dari: nim, nama, asal, dan kelas praktikum. Data diinputkan melalui keyboard. Terdapat pilihan menu *insert, update*, *delete* untuk pengelolaan objek serta *print* dan *save* untuk menampilkan dan menyimpan objek dari/kedalam suatu file.

Jawaban Tugas Pengembangan



MODUL 4

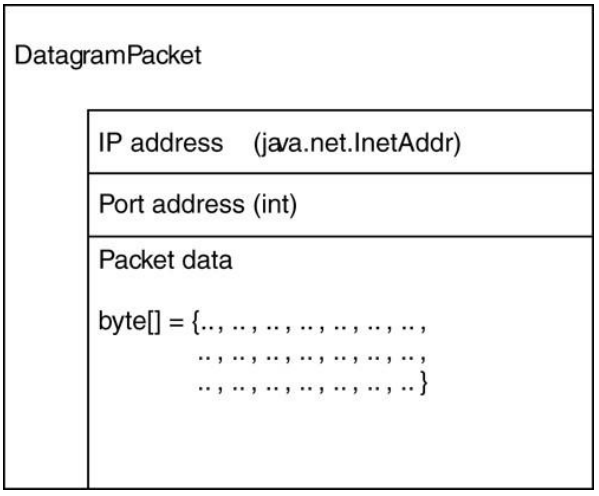
USER DATAGRAM PROTOCOL

4.1 Pembahasan

User Datagram Protocol (UDP) adalah protokol transport yang umum digunakan dalam berbagai jenis aplikasi. UDP adalah protokol transport *connectionless*, yang berarti bahwa itu tidak menjamin paket yang dikirim dan paket yang tiba berurutan. Pada UDP, byte data dikelompokkan dalam paket diskrit, yang dikirim melalui jaringan. Java mendukung User Datagram Protocol dengan adanya 2 kelas, yaitu `java.net.DatagramPacket` dan `java.net.DatagramSocket`

4.1.1 Kelas DatagramPacket

Kelas DatagramPacket merupakan representasi paket data yang dimaksudkan untuk transmisi menggunakan User Datagram Protocol (lihat *Gambar 4.1*). Paket adalah wadah untuk urutan kecil byte, dan termasuk informasi seperti alamat IP dan port.



Gambar 4.1 DatagramPacket sebagai representasi sebuah paket UDP

Arti dari data yang disimpan dalam DatagramPacket ditentukan oleh konteksnya. Ketika DatagramPacket dibaca dari socket UDP, maka alamat IP dari paket merupakan alamat pengirim (juga dengan nomor port). Namun, ketika DatagramPacket yang digunakan untuk mengirim paket UDP, maka alamat IP yang disimpan di DatagramPacket merupakan alamat penerima (juga dengan nomor port).

Untuk membuat DatagramPacket maka perlu memanggil constructor dari DatagramPacket. Contructor yang digunakan dapat disesuaikan dengan tujuannya. Untuk DatagramPacket yang digunakan untuk **menerima paket UDP**, maka harus menggunakan constructor berikut: DatagramPacket(byte[] buffer, int length). Sebagai contoh:

```
DatagramPacket packet = new DatagramPacket(new byte[256], 256);
```

Sedangkan untuk **mengirim sebuah DatagramPacket**, maka lebih utama menggunakan constructor berikut DatagramPacket(byte[] buffer, int length, InetAddress dest_addr, int dest_port). Sebagai contoh:

```
InetAddress addr = InetAddress.getByName("192.168.0.1");

DatagramPacket packet = new DatagramPacket (new byte[128], 128, addr, 2000);
```

Diantara method yang dapat digunakan pada kelas DatagramPacket ditunjukkan pada *Tabel 4.1*.

Tabel 4.1 Diantara Method pada kelas DatagramPacket

Method	Diskripsi
InetAddress getAddress()	Mengembalikan alamt IP dari DatagramPacket yang dikirim, atau alamat IP tujuan (jika paket akan dikirim)
byte[] getData()	Mengembalikan isi/konten dari DatagramPacket dalam bentuk array byte
int getLength()	Mengembalikan panjang data yang tersimpan pada DatagramPacket
int getPort()	Mengembalikan nomor port dari DatagramPacket yang telah dikirim atau nomor port tujuan (jika paket akan dikirim)
void setAddress(InetAddress addr)	Memberlakukan alamat tujuan yang baru untuk DatagramPacket
void setData(byte[] buffer)	Memberlakukan buffer data yang baru untuk DatagramPacket
void setLength(int length)	Memberlakukan panjang baru untuk DatagramPacket. Panjang (length) ini harus kurang dari atau sama dengan ukuran maksimal dari buffer data. Jika tidak maka IllegalArgumentException akan diberlakukan
void setPort(int port)	Memberlakukan port tujuan yang baru untuk DatagramPacket

4.1.2 Kelas DatagramSocket

Kelas DatagramSocket memberikan akses pada socket UDP, sehingga memungkinkan paket UDP dikirim dan diterima. Untuk membuat **DatagramSocket client**, maka dapat menggunakan constructur berikut: DatagramSocket() throws java.net.SocketException, sedangkan untuk membuat **DatagramSocket server**, maka menggunakan contructur berikut: DatagramSocket(int port) throws java.net.SocketException. Diantara method pada kelas ini ditunjukkan pada *Tabel 4.2*.

Tabel 4.2 Beberapa Method pada kelas DatagramSocket

Method	**Diskripsi**
void close()	Menutup socket
void connect(InetAddress remote_addr int remote_port)	Membatasi akses ke alamat dan port tertentu
void disconnect()	Memutus DatagramSocket dan menghapus semua batasan yang ditetapkan sebelumnya
InetAddress getInetAddress()	Mengembalikan alamat yang terhubung dengan socket, atau null jika tidak ada koneksi
int getPort()	Mengembalikan port yang terhubung dengan socket, atau -1 jika tidak ada koneksi
InetAddress getLocalAddress()	Mengembalikan alamat local
int getLocalPort()	Mengembalikan port local
int getReceiveBufferSize() throws java.net.SocketException	Mengembalikan ukuran buffer maksimum yang digunakan untuk paket UDP yang masuk
int getSendBufferSize() throws java.net.SocketException	Mengembalikan ukuran buffer maksimum digunakan untuk paket UDP yang keluar/dikirim
int getSoTimeout() throws java.net.SocketException	Mengembalikan nilai batasan waktu socket.
void receive(DatagramPacket packet) throws java.io.IOException	Membaca sebuah paket UDP dan menyimpan isi dalam paket yang ditentukan.
void send(DatagramPacket packet) throws java.io.IOException	Mengirim paket UDP
void setReceiveBufferSize(int length) throws java.net. SocketException	menetapkan ukuran buffer maksimum yang digunakan untuk paket UDP yang masuk.
void setSendBufferSize(int length) throws java.net.SocketException	menetapkan ukuran buffer maksimum yang digunakan untuk paket UDP yang keluar
void setSoTimeout(int duration) throws java.net.Socket Exception	menetapkan nilai opsi batas waktu socket. Nilai ini adalah jumlah milidetik operasi baca akan memblok sebelum paket dianggap tidak terbaca sehingga menetapkan java.io.InterruptedIOException a.

4.1.3 Contoh User Datagram Protocol

Implementasi transmisi data menggunakan UDP pada Java ditunjukkan dengan adanya dua kelas, yaitu kelas yang berfungsi untuk menerima pesan (**Contoh 4.1**) dan kelas

yang berfungsi untuk mengirim pesan (**Contoh 4.2**). Program berjalan dengan menjalankan kelas penerima dan disusul dengan menjalankan kelas pengirim (bukan sebaliknya).

Contoh 4.1. Code sederhana untuk Menerima Paket

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class SimplePacketReceive {

    public static void main(String[] args) throws IOException {

        DatagramSocket socket = new DatagramSocket(2000);
        // Create a datagram packet, containing a maximum buffer of 256 bytes
        DatagramPacket packet = new DatagramPacket(new byte[256], 256);
        socket.receive(packet);

        String message = new String(packet.getData(), 0, packet.getLength());
        System.out.println(message);

    }
}
```

Contoh 4.2. Code sederhana untuk Mengirim Paket

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;

public class SimplePacketSend {

    public static void main(String[] args) throws SocketException,
            UnknownHostException, IOException {

        DatagramSocket socket = new DatagramSocket();
        String message = "Assalamu'alaikum";
        DatagramPacket packet = new DatagramPacket(message.getBytes(),
                message.length(), InetAddress.getLocalHost(), 2000);
        socket.send(packet);
        socket.close();

    }
}
```

Output:

Setelah menjalankan kelas SimplePacketReceive dan kelas SimplePacketSend maka pada pada kelas penerima akan tampak pesan yang dikirimkan oleh kelas pengirim sebagaimana berikut ini:

```
run:
Assalamu'alaikum
BUILD SUCCESSFUL (total time: 10 seconds)
```

Contoh lain dari implementasi transmisi data menggunakan UDP pada Java ditunjukkan pada **Contoh 4.3** dan **Contoh 4.4** sebagimana berikut ini.

Contoh 4.3. Code Menerima Paket UDP beserta beberapa informasi pengirim

```
import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class PacketReceiveDemo {

    public static void main(String[] args) {
        System.out.println("Packet Receive\n================");
        System.out.println("Binding to local port 2000");
        try (DatagramSocket socket = new DatagramSocket(2000)) {
            System.out.println("Bound to local port " + socket.getLocalPort());

            // Create a datagram packet, containing a maximum buffer of 256 bytes
            DatagramPacket packet = new DatagramPacket(new byte[256], 256);

            // Receive a packet
            socket.receive(packet);
            System.out.println("Packet received!");

            // Display packet information
            InetAddress remote_addr = packet.getAddress();
            System.out.println("Sent by  : " + remote_addr.getHostAddress());
            System.out.println("Sent from: " + packet.getPort());

            // Display packet contents, by reading from byte array
            ByteArrayInputStream bin = new ByteArrayInputStream(packet.getData());
            BufferedReader br = new BufferedReader(new InputStreamReader(bin));
            System.out.print("Message  : " + br.readLine());

        } catch (IOException ex) {
            System.err.println("Error: " + ex.getMessage());
        }
    }
}
```

Contoh 4.4. Code mengirim paket UDP dan pengaturan identitas pengirim

```
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

public class PacketSendDemo {

    public static void main(String[] args) {
        System.out.println("Packet Send\n==================");
        String hostname = "localhost";

        System.out.println("Binding to a local port");
        // Create a datagram socket, bound to any available local port
        try (DatagramSocket socket = new DatagramSocket()) {
            System.out.println("Bound to local port " + socket.getLocalPort());

            // Create a message to send using a UDP packet
            ByteArrayOutputStream bout = new ByteArrayOutputStream();
            PrintStream pout = new PrintStream(bout);
            pout.print("Assalamu'alaikum");

            // Get the contents of our message as an array of bytes
            byte[] barray = bout.toByteArray();

            // Create a datagram packet, containing our byte array
            DatagramPacket packet = new DatagramPacket(barray, barray.length);

            // Lookup the specified hostname, and get an InetAddress
            System.out.println("Looking up hostname " + hostname);
            InetAddress remote_addr = InetAddress.getByName(hostname);
            System.out.println("Hostname resolved as " + remote_addr.getHostAddress());

            // Address packet to sender
            packet.setAddress(remote_addr);

            // Set port number to 2000
            packet.setPort(2000);

            // Send the packet - remember no guarantee of delivery
            socket.send(packet);
            System.out.println("Packet sent!");

        } catch (SocketException ex) {
            System.err.println("Error: " + ex.getMessage());
        } catch (IOException ex) {
            System.err.println("Error: " + ex.getMessage());
        }
    }
}
```

Output:

Kelas PacketReceiveDemo dijalankan terlebih dahulu kemudian kelas PacketSendDemo, sehingga tampak output program dari sisi receiver dan sender sebagaimana berikut ini:

```
run:
Packet Receive
==================
Binding to local port 2000
Bound to local port 2000

run:
Packet Send
==================
Binding to a local port
Bound to local port 51862
Looking up hostname localhost
Hostname resolved as 127.0.0.1
Packet sent!
BUILD SUCCESSFUL (total time: 3 seconds)

run:
Packet Receive
==================
Binding to local port 2000
Bound to local port 2000
Packet received!
Sent by   : 127.0.0.1
Sent from: 51862
Message   : Assalamu'alaikum
```

4.1.4 Membangun UDP Client/Server

Contoh sebelumnya menggambarkan teknik bagaimana paket tunggal dapat dikirim dan diterima. Namun, applikasi membutuhkan kumpulan dari paket, bukan hanya satu paket. Jenis layanan yang menangani kasus tersebut biasa disebut dengan layanan *echo*. Layanan echo berjalan pada port 7. Jika sistem memiliki server echo, maka server dapat diakses oleh client apakah sistem sedang berjalan (mirip dengan cara kerja ping).

Berikut ini dicontohkan bagaimana membangun sebuah Server UDP, sebuah sistem yang terus berjalan yang mampu melayani banyak request selama sistem tersebut dijalankan.

Contoh 4.5. Code Echo Server

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class EchoServer {
    // UDP port to which service is bound
    public static final int SERVICE_PORT = 7;
    // Max size of packet, large enough for almost any client
    public static final int BUFSIZE = 4096;
    // Socket used for reading and writing UDP packets
    private DatagramSocket socket;
```

```
    public EchoServer() {
        try {
            // Bind to the specified UDP port, to listen for incoming data packets
            socket = new DatagramSocket(SERVICE_PORT);
            System.out.println("Server active on port " + socket.getLocalPort());
        } catch (Exception e) {
            System.err.println("Unable to bind port");
        }
    }

    public void serviceClients() {
        // Create a buffer large enough for incoming packets
        byte[] buffer = new byte[BUFSIZE];
        for (;;) {
            try {
                // Create a DatagramPacket for reading packets
                DatagramPacket packet = new DatagramPacket(buffer, BUFSIZE);
                // Receive incoming packets
                socket.receive(packet);
                System.out.println("Packet received from "
                        + packet.getAddress() + ":"
                        + packet.getPort()
                        + " of length " + packet.getLength());

                socket.send(packet);
            } catch (IOException ioe) {
                System.err.println("Error : " + ioe);
            }
        }
    }

    public static void main(String args[]) {
        EchoServer server = new EchoServer();
        server.serviceClients();
    }
}
```

Contoh 4.6. Code Echo Client

```
import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.InterruptedIOException;
import java.io.PrintStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

public class EchoClient {

    public static final int SERVICE_PORT = 7;
    public static final int BUFSIZE = 256;

    public static void main(String[] args) throws SocketException, IOException {
        String hostname = "localhost";
        InetAddress addr = InetAddress.getByName(hostname);
```

```
            DatagramSocket socket = new DatagramSocket();
            socket.setSoTimeout(2000);

            BufferedReader reader
                    = new BufferedReader(new InputStreamReader(System.in));
            for (;;) {
                System.out.println("Write your message here..");
                ByteArrayOutputStream bout = new ByteArrayOutputStream();
                PrintStream pout = new PrintStream(bout);
                pout.print(reader.readLine());
                // Get the contents of our message as an array of bytes
                byte[] barray = bout.toByteArray();
                // Create a datagram packet, containing our byte array
                DatagramPacket packet
                        = new DatagramPacket(barray, barray.length, addr, SERVICE_PORT);
                System.out.println("Sending packet to " + hostname);
                socket.send(packet);

                System.out.println("Waiting for packet....");
                // Create a small packet for receiving UDP packets
                byte[] recbuf = new byte[BUFSIZE];
                DatagramPacket receivePacket = new DatagramPacket(recbuf, BUFSIZE);
                // Declare a timeout flag
                boolean timeout = false;
                // Catch any InterruptedIOException that is thrown
                // while waiting to receive a UDP packet
                try {
                    socket.receive(receivePacket);
                } catch (InterruptedIOException ioe) {
                    timeout = true;
                }
                if (!timeout) {
                    System.out.println("packet received!");
                    System.out.println("Details : " + receivePacket.getAddress());
                    // Obtain a byte input stream to read the UDP packet
                    ByteArrayInputStream bin = new ByteArrayInputStream(
                            receivePacket.getData(), 0,
                            receivePacket.getLength());
                    // Connect a reader for easier access
                    BufferedReader reader2 =
                            new BufferedReader(new InputStreamReader(bin));
                    System.out.println(reader2.readLine());

                } else {
                    System.out.println("packet lost!");
                }
            }
        }
    }
```

Output:

Berikut ini kondisi awal ketika echo server dijalankan:

```
run:
Server active on port 7
```

Begitu echo client dijalankan maka user dapat menuliskan pesan secara berulang dan mendapatkan konfirmasi dari server tentang status pesan yang dikirim

```
run:
Write your message here..
first message
Sending packet to localhost
Waiting for packet....
packet received!
Details : /127.0.0.1
first message
Write your message here..
second message
Sending packet to localhost
Waiting for packet....
packet received!
Details : /127.0.0.1
second message
Write your message here..
and so on..
Sending packet to localhost
Waiting for packet....
packet received!
Details : /127.0.0.1
and so on..
Write your message here..
```

Berikut ini notifikasi pada server bahwa paket diterima.

```
run:
Server active on port 7
Packet received from /127.0.0.1:61689 of length 13
Packet received from /127.0.0.1:61689 of length 14
Packet received from /127.0.0.1:61689 of length 11
```

4.2 Praktikum

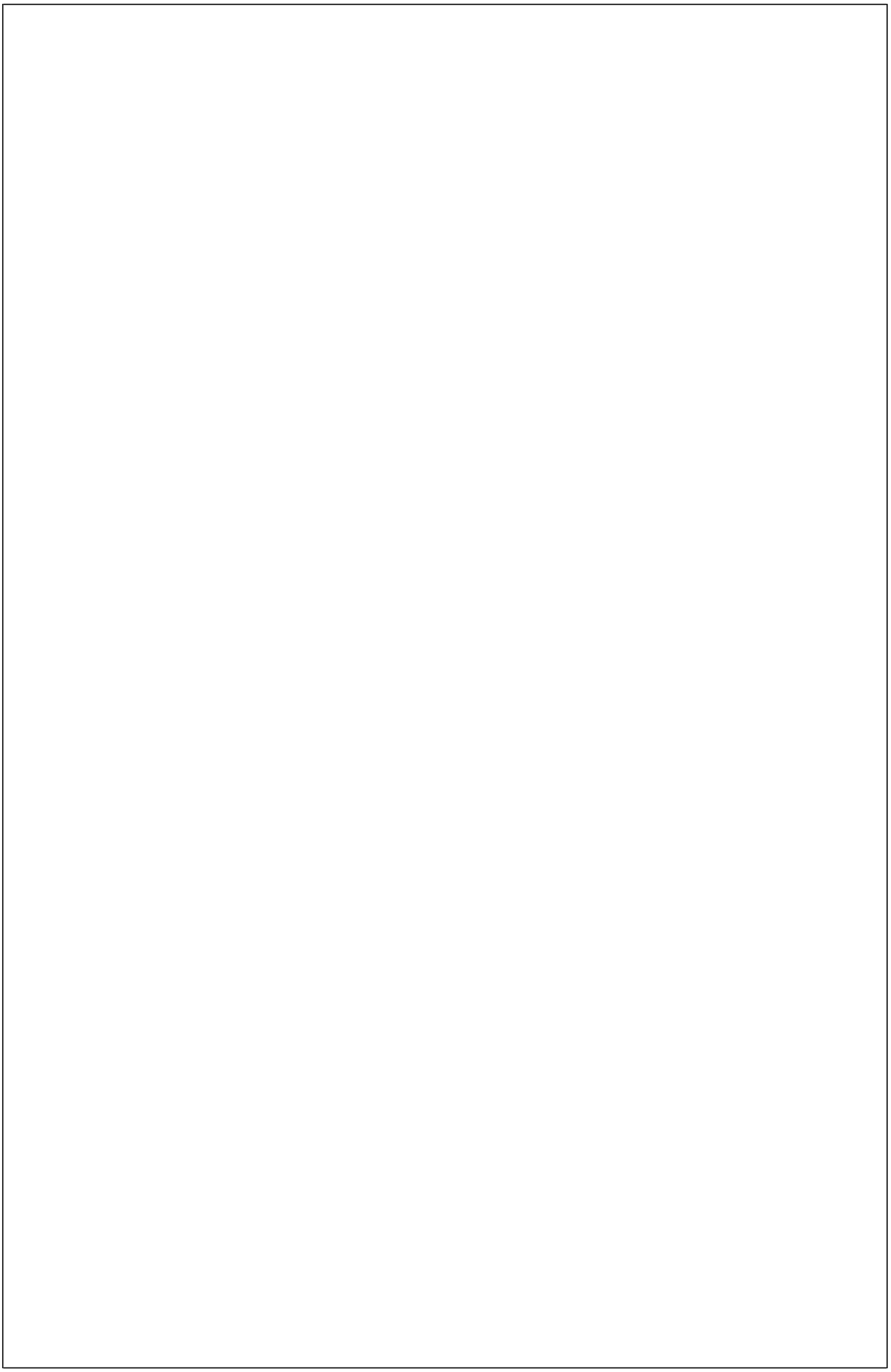
1. Tulis dan jalankan **Contoh 4.1** dan **Contoh 4.2** pada modul ini dan pahami tiap barisnya!
2. Tulis dan jalankan **Contoh 4.3** dan **Contoh 4.4** pada modul ini dan pahami tiap barisnya! Tambahkan baris program pada kelas packetSendDemo sehingga pesan yang dikirim dibaca dari inputan keyboard!
3. Tulis dan jalankan **Contoh 4.5** dan **Contoh 4.6** pada modul ini dan pahami tiap barisnya! Tambahkan kondisi dimana ketika client menuliskan “exit” maka socket client akan ditutup dan program client berakhir.

4.3 Tugas

1. Sebutkan kekurangan (minimal 1) dan kelebihan (minimal 5) dari UDP! Jelaskan masing-masing!

2. Buatlah program penyimpanan data mahasiswa client-server UDP dengan mengimplemantasikan Object Serialization. Data mahasiswa terdiri dari: nim, nama, asal, dan kelas praktikum. Data diinputkan dari sisi client melalui keyboard. Terdapat pilihan menu *insert, update, delete* untuk pengelolaan objek serta *print* dan *save* untuk menampilkan dan menyimpan objek dari/kedalam suatu file. Data object dari client dikirim ke server kemudian server menyimpan dalam suatu file.

Jawaban Tugas Pengembangan



MODUL 5

TRANSMISSION CONTROL PROTOCOL

5.1 Pembahasan

Transmission Control Protocol (TCP) adalah metode berbasis aliran (*stream*) pada komunikasi jaringan. Modul ini membahas TCP stream dan bagaimana implementasinya pada Java. Berbeda dengan UDP, TCP menjamin data terkirim dan berurutan, menyediakan aliran komunikasi byte antara client dan server yang mendukung komunikasi dua arah. TCP menggunakan Internet Protocol (IP) untuk membuat koneksi antar mesin.

5.1.1 Socket pada TCP

Socket pada TCP dapat melakukan beberapa operasi lebih dibandingkan dengan UDP, yaitu mampu untuk:

- Membangun koneksi ke sebuah remote host
- Mengirim data ke remote host
- Menerima data dari remote host
- Menutup koneksi

Selain itu, terdapat pula tipe socket yang biasa digunakan sebagai server dengan kemampuan untuk mengikat (*bind*) nomor port tertentu. Socket ini mampu melakukan operasi berikut:

- Mengikat (*bind*) ke port local
- Menerima koneksi yang datang dari remote host
- Melepaskan (*unbind*) dari port local

Dua tipe socket ini masing-masing digunakan sebagai client atau sebagai server. Java mendukung implementasi dari TCP socket dengan adanya dua kelas socket, yaitu: java.net.Socket dan java.net.ServerSocket. Kelas Socket digunakan untuk membuat software client yang berhubungan dengan layanan tertentu, sedangkan kelas ServerSocket digunakan untuk membuat software server yang mengikat ke port local dan menyediakan layanan tertentu. Socket ini berbeda dengan DatagramSocket yang bekerja pada UDP dengan adanya fungsi koneksi ke server dan fungsi menerima data dari client.

5.1.2 Kelas Socket

Kelas Socket merepresentasikan socket client. Socket TCP tidak bisa berkomunikasi dengan lebih dari dua mesin. Jika dibutuhkan melakukan komunikasi dengan lebih dari dua mesin maka sebuah aplikasi client harus membangun sejumlah koneksi socket dimana masing-masing untuk tiap mesin.

Terdapat beberapa constructor untuk kelas java.net.Socket yang dapat digunakan dalam situasi yang berbeda. Hampir semua constructor bersifat public, kecuali beberapa. Constructor kelas Socket ditunjukkan pada **Tabel 5.1**.

Tabel 5.1 Constructor pada kelas Socket

Constructor	**Diskripsi**
protected Socket ()	Membuat sebuah socket yang tidak berhubungan menggunakan implementasi default. Tidak disarankan menggunakan method ini karena tidak menentukan hostname atau port tertentu
Socket (InetAddress address, int port) throws java.io.IOException, java.lang.SecurityException	Membuat socket yang terhubung ke alamat IP dan port tertentu
Socket (InetAddress address, int port, InetAddress localAddress, int localPort) throws java.io.IOException, java.lang.SecurityException	Membuat socket yang terhubung ke alamat dan port yang ditentukan, dan terikat ke alamat local dan port local tertentu. Method ini biasa digunakan pada kasus multihomed hosts, yaitu sebuah mesih dimana localhost dikenal dengan dua alau lebih alamat IP.
protected Socket (SocketImpl implementation)	Membuat socket yang tidak berhubungan menggunakan implementasi socket tertentu. Pengembang tidak disarankan menggunakan method ini.
Socket (String host, int port) throws java.net.UnknownHostException, java.io.IOException, java.lang.SecurityException	Membuat socket yang terhubung ke host dan port tertentu. Method ini memungkinkan memberikan parameter String, bukan InetAddress sebagaimana method sebelumnya.
Socket (String host, int port, InetAddress localAddress, int localPort) throws java.net.UnknownHostException, java.io. IOException, java.lang.SecurityException	Membuat socket yang terhubung ke host dan port tertentu serta terikat pada alamat dan port local. Method ini biasa digunakan pada kasus multihomed hosts, yaitu sebuah mesih dimana localhost dikenal dengan dua alau lebih alamat IP.

Setelah membuat socket dengan penulisan constructor kelas socket, maka programmer dapat menggunakan socket untuk beberapa tugas seperti membaca informasi, mengirim data, menutup koneksi, dan mengatur opsi socket **Tabel 5.2** berikut ini memuat beberapa method yang dapat digunakan pada kelas socket.

Tabel 5.2 Method pada kelas Socket

Method	**Diskripsi**
void close() throws java.io.IOException	Menutup koneksi socket
InetAddress getInetAddress()	Mengembalikan alamat dari *remote machine* yang terhubung dengan socket
InputStream getInputStream() throws java.io.IOException	Mengembalikan sebuah input stream, yang dibaca dari aplikasi socket yang terhubung
OutputStream getOutputStream() throws java.io.IOException	Mengembalikan sebuah output stream, yang ditulis ke aplikasi dimana socket terhubung dengannya
InetAddress getLocalAddress()	Mengembalikan alamat local yang terkait dengan socket (sangat berguna pada kasus *multihomed machines*)
int getLocalPort()	Mengembalikan nomor port dimana socket terikat pada mesin local.
int getPort()	Mengembalikan nomor port dari layanan remote dimana socket terhubung dengannya
void shutdownInput() throws java.io.IOException	Menutup input stream yang berhubungan dengan socket
void shutdownOutput() throws java.io.IOException	Menutup output stream yang berhubungan dengan socket

Setelah socket dibuat dan terhubung, maka socket siap digunakan untuk membaca atau menulis dengan menggunakan input dan output stream socket. Stream ini tidak perlu dibuat karena sudah disediakan oleh socket dengan method Socket. getInputStream() dan Socket.getOutputStream(). Potongan kode berikut menunjukkan TCP klien sederhana yang menghubungkan BufferedReader untuk input stream socket, dan PrintStream ke output stream socket.

```
try {
        // Connect a socket to some host machine and port
        Socket socket = new Socket ( somehost, someport );
        // Connect a buffered reader
        BufferedReader reader = new BufferedReader (
                new InputStreamReader ( socket.getInputStream() ) );
        // Connect a print stream
        PrintStream pstream = new PrintStream( socket.getOutputStream() );
} catch (Exception e) {
        System.err.println ("Error – " + e);
}
```

5.1.3 Kelas ServerSocket

Server socket digunakan untuk menyediakan layanan TCP. Socket pada client hanya mampu mengikat satu port dimana client terhubung dengannya, sedangkan server socket mampu memenuhi permintaan koneksi dari beberapa client.

Begitu server socket dibuat maka socket mengikat port local dan siap menerima permintaan koneksi. Terdapat beberapa constructor yang dapat digunakan untuk membuat ServerSocket sebagaimana yang ditunjukkan pada **Tabel 5.3**.

Tabel 5.3 Constructor kelas ServerSocket

Constructor	**Diskripsi**
ServerSocket(int port) throws java.io.IOException, java.lang. SecurityException	membuat socketserver dengan nomor port yang ditentukan
ServerSocket(int port, int numberOfClients) throws java.io. IOException, java.lang.SecurityException	Membuat socketserver dengan nomor port tertentu serta mengalokasikan jumlah socket client
ServerSocket(int port, int numberOfClients, InetAddress address) throws java.io.IOException, java.lang.SecurityException	Membuat socketserver dengan nomor port tertentu, mengalokasikan jumlah socket client dan memungkinkan socketserver untuk mengikatkan koneksi pada alamat IP tertentu. Ini bisa digunakan pada kasus *multihomed machine*

Method pada kelas ServerSocket tidak sebanyak method pada kelas socket. Method yang paling penting pada kelas ini adalah accept(). **Tabel 5.4** berikut ini menunjukkan beberapa method yang bisa digunakan pada kelas ServerSocket.

Tabel 5.4 Method pada kelas ServerSocket

Method	**Diskripsi**
Socket accept() throws java.io.IOException, java.lang.Security Exception	Menunggu klient untuk meminta koneksi ke server socket dan menerimanya.
void close() throws java.io.IOException	Menutup socket server, dimana socket server melepaskan port TCP dan mengijinkan layanan lain menggunakannya
InetAddress getInetAddress()	Mengembalikan alamat dari socket server
int getLocalPort()	Mengembalikan nomor port dimana server socket terikat
int getSoTimeout() throws java.io.IOException	Mengembalikan nilai timeout, yang menentukan berapa millisecond sebuah operasi accept() dapat memblok.

void setSoTimeout(int timeout) throws java.net.SocketException	Memberikan nilai timeout dalam hitungan millisecond untuk bloking operasi accept(). Jika menggunakan nilai nol berarti timeout dinonaktivkan.

5.1.4 Membuat *TCP Client/Server*

Pada subbab ini ditunjukkan kode untuk membuat TCP client dengan pemanggilan kelas *Socket* dan kode untuk membuat TCP server dengan pemanggilan kelas *ServerSocket*. TCP client dan server yang dicontohkan disini adalah layanan daytime. *Daytime client* dapat terhubung dengan *daytime server* untuk membaca hari dan waktu saat itu. Secara default, layanan daytime berjalan pada port 13.

Contoh 5.1. Kode untuk *Daytime Client*

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;

public class DaytimeClient {

    public static final int SERVICE_PORT=13;
    public static void main(String[] args) {
        try {
            String hostname= "localhost";

            //Get a socket to daytime service
            Socket daytime = new Socket (hostname, SERVICE_PORT);
            System.out.println("Connection established");

            //Set the socket option just in case server stalls
            daytime.setSoTimeout(2000);

            //Read from the server
            BufferedReader reader= new BufferedReader(
                    new InputStreamReader(daytime.getInputStream()));
            System.out.println("Result: "+reader.readLine());

            //Close the connection
            daytime.close();
        } catch (IOException ioe) {
            System.err.println("Error "+ioe);
        }
    }
}
```

Contoh 5.2. Kode untuk *Daytime Server*

```
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.BindException;
import java.net.ServerSocket;
import java.net.Socket;

public class DaytimeServer {

    public static final int SERVICE_PORT = 13;

    public static void main(String[] args) {
        try {
            //Bind to the service port,
            //to grant clients access to the TCP daytime service
            ServerSocket server = new ServerSocket(SERVICE_PORT);
            System.out.println("Daytime service started");

            //Loop indefinitely, accepting clients
            for (;;) {
                //Get the next TCP client
                Socket nextClient = server.accept();

                //Display connection details
                System.out.println("Received request from "
                        + nextClient.getInetAddress() + " : "
                        + nextClient.getPort());

                //just write the message
                OutputStream out = nextClient.getOutputStream();
                PrintStream pout = new PrintStream(out);

                //Write the current date out to the user
                pout.print(new java.util.Date());

                //Flust unsent bytes
                out.flush();

                //close stream
                out.close();

                //close the connection
                nextClient.close();
            }
        } catch (BindException be) {
            System.err.println("Service already running on port " + SERVICE_PORT);
        } catch (IOException ioe) {
            System.err.println("I/O error - " + ioe);
        }
    }
}
```

Output:

Jalankan daytime server untuk memulai layanan daytime

```
run:
Daytime service started
```

Begitu server siap, jalankan daytime client sehingga client mendapatkan informasi hati dan waktu saat itu sebagaimana berikut:

```
run:
Connection established
Result: Wed Mar 09 07:44:19 ICT 2016
BUILD SUCCESSFUL (total time: 0 seconds)
```

Berikut ini output pada server ketika ada client yang terhubung.

```
run:
Daytime service started
Received request from /127.0.0.1 : 49768
```

Server tetap berjalan dan menerima client yang mengaksesnya.

```
run:
Connection established
Result: Wed Mar 09 07:44:40 ICT 2016
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run:
Daytime service started
Received request from /127.0.0.1 : 49768
Received request from /127.0.0.1 : 49769
```

```
run:
Connection established
Result: Wed Mar 09 07:45:16 ICT 2016
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run:
Daytime service started
Received request from /127.0.0.1 : 49768
Received request from /127.0.0.1 : 49769
Received request from /127.0.0.1 : 49770
```

begitu seterusnya server tetap berjalan dan menerima koneksi client. Hanya ada satu server dengan satu port, ketika server dijalankan lagi pada waktu bersamaan maka BindException akan bekerja.

```
run:
Service already running on port 13
BUILD SUCCESSFUL (total time: 0 seconds)
```

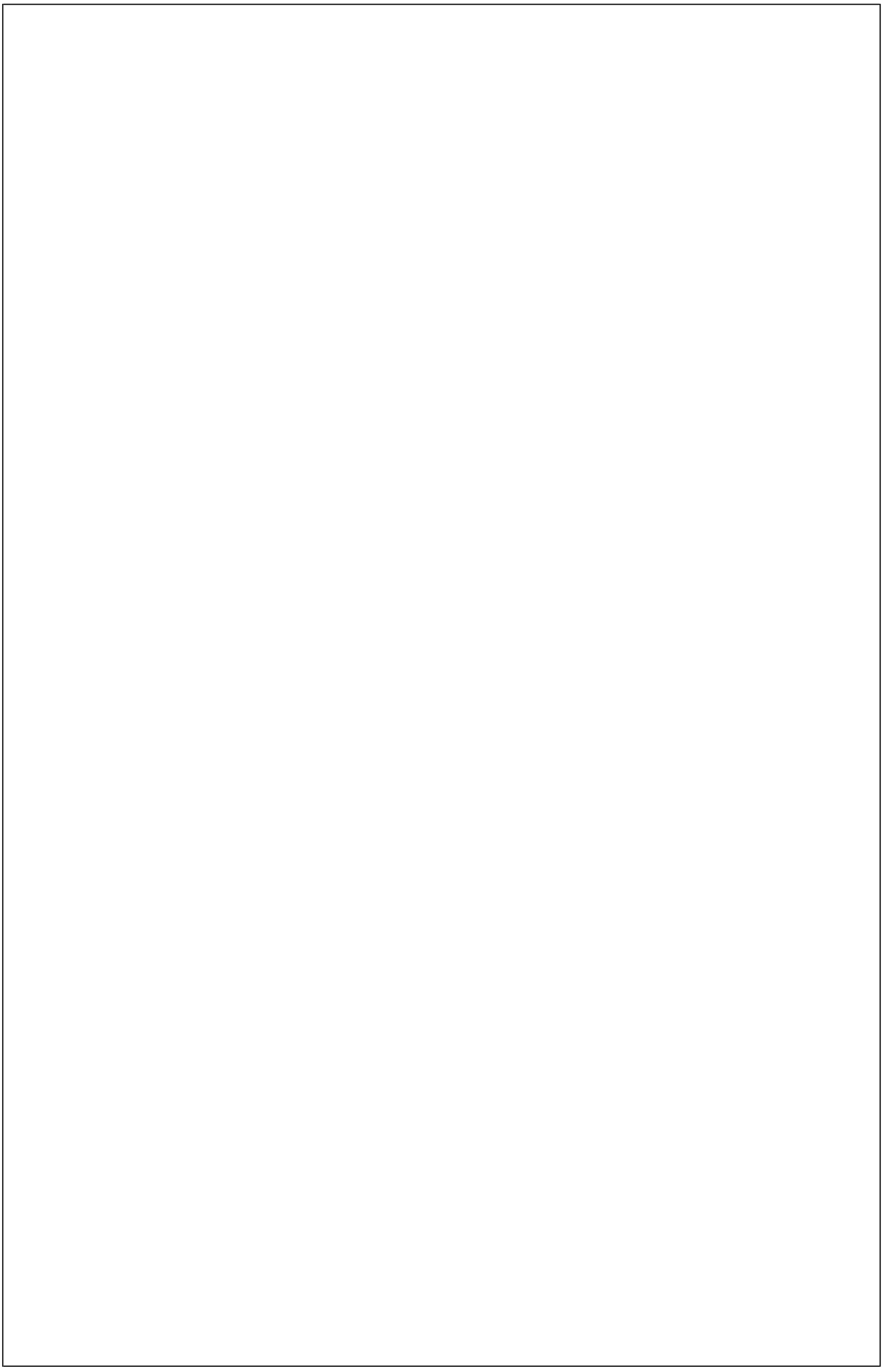
5.2 Praktikum

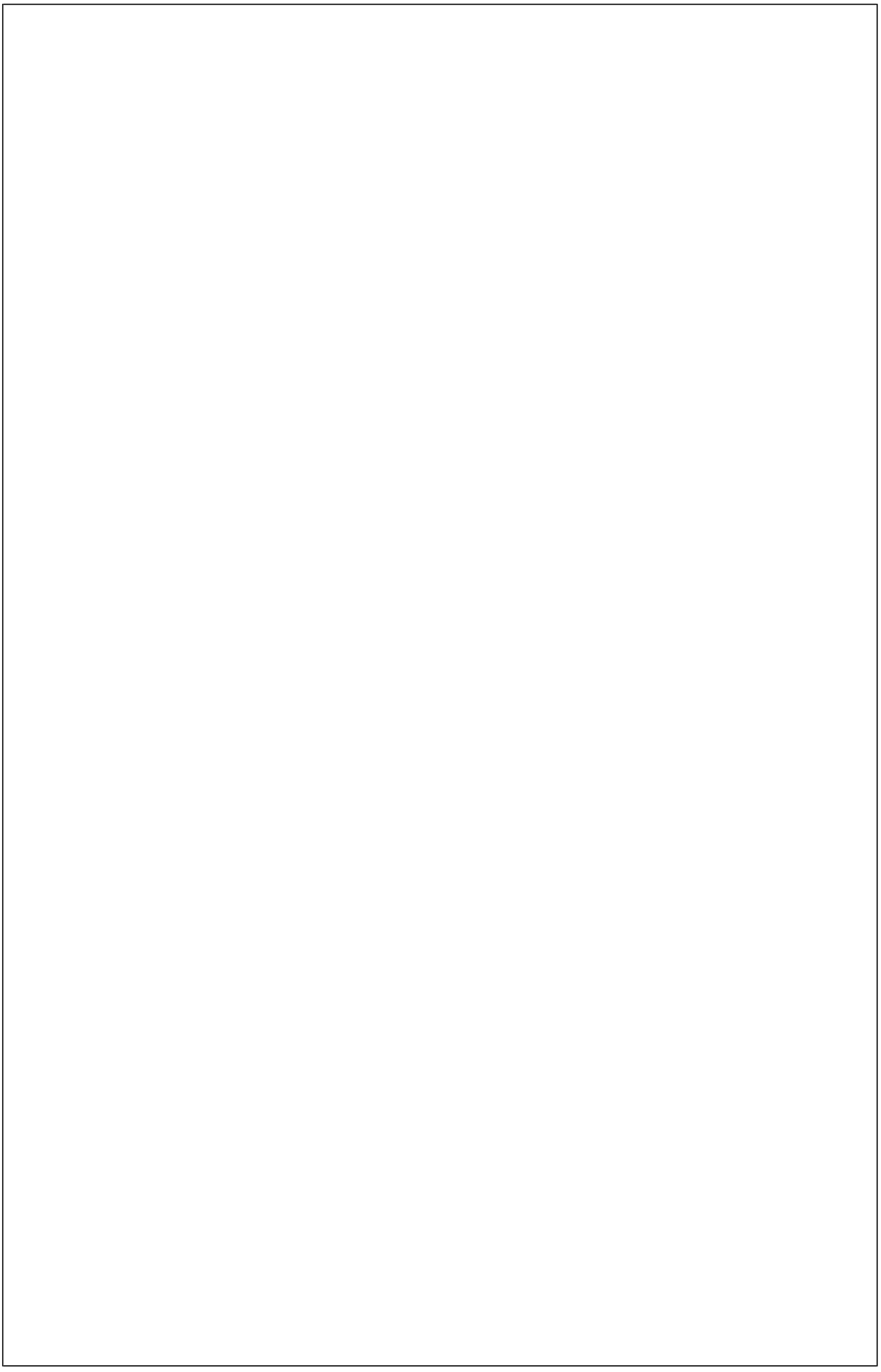
1. Tulis dan jalankan **Contoh 5.1** dan **Contoh 5.2** pada modul ini dan pahami tiap barisnya!
2. Buatlah program TCP client-server sederhana dimana client dapat mengirimkan pesan secara berulang yang diperoleh dari input keyboard sedangkan server dapat menerima pesan dan menapilkan pesan tersebut berserta informasi alamat dan port client!
3. Buatlah program seperti nomor 2 dimana server dapat mengirimkan pesan balasan kepada client sesuai pesan yang dikirim client. Sebagai contoh: client mengirim “Assalamu’alaikum” server membalas “Wa’alaikumussalam”.

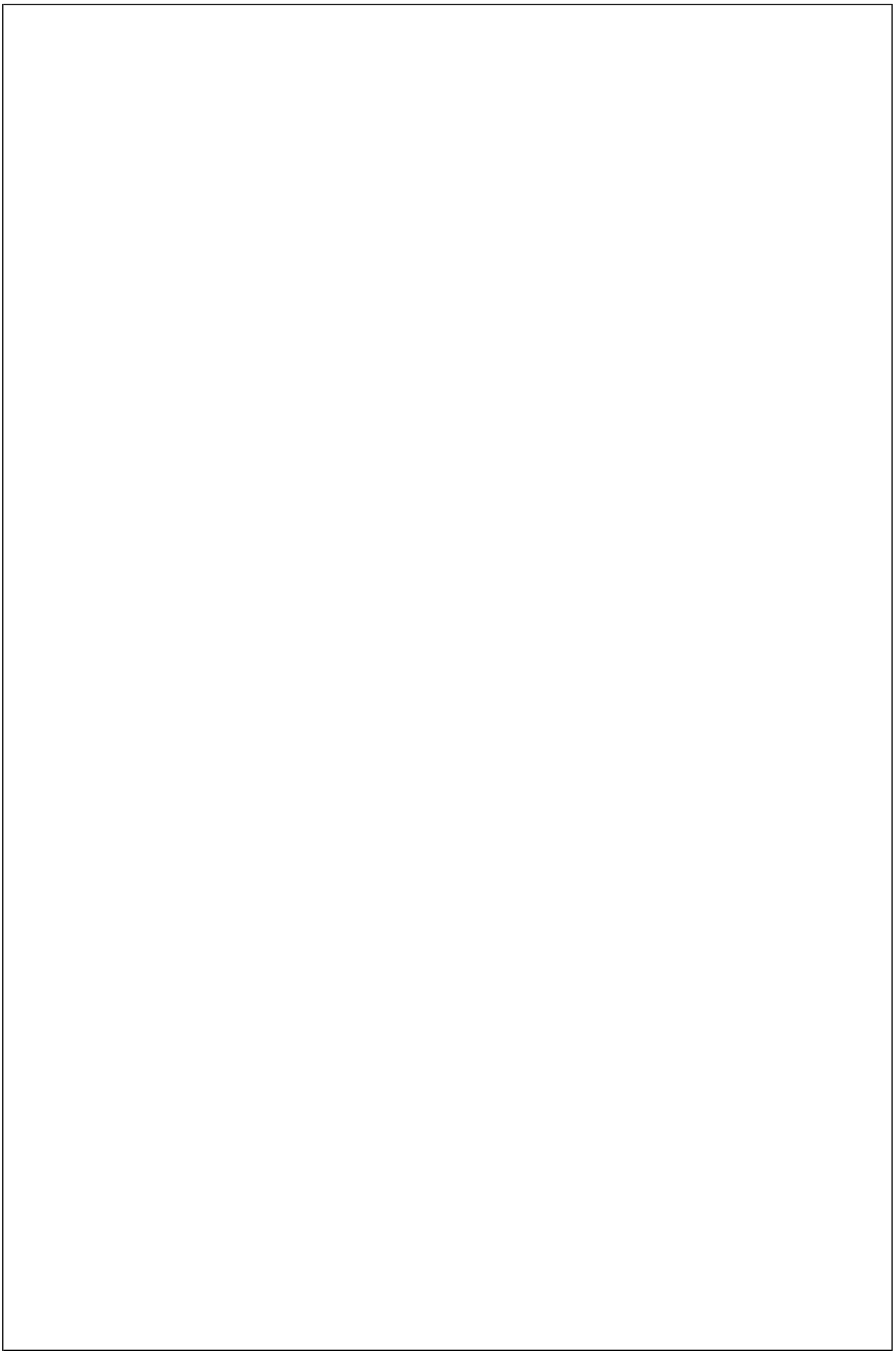
5.3 Tugas

1. Jelaskan persamaan dan perbedaan TCP dan UDP!
2. Sebutkan kelebihanTCP dibandingkan UDP (minimal 3)!
3. Buatlah program penyimpanan data mahasiswa client-server TCP dengan mengimplemantasikan Object Serialization. Data mahasiswa terdiri dari: nim, nama, asal, dan kelas praktikum. Data diinputkan dari sisi client melalui keyboard. Data object dari client dikirim ke server kemudian server menyimpan dalam suatu file.
4. Buatlah program dengan GUI untuk mengirim dan menerima pesan gambar yang mengimplmentasikan TCP
5. Buatlah program dengan GUI untuk mengirim dan menerima pesan suara yang mengimplmentasikan TCP

Jawaban Tugas Pengembangan







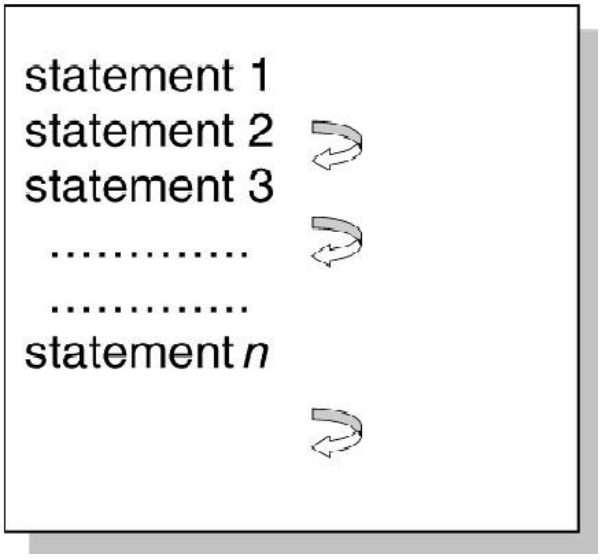
MODUL 6

MULTITHREAD PROGRAMMING

6.1 Pembahasan

Pemrograman multithread merupakan konsep penting dalam networking menggunakan Java, misalkan untuk melakukan beberapa tugas yang berbeda dalam satu waktu pada jaringan client-server. Sebelum pembahasan multithread, maka penting untuk memahami perbedaan antar pemrograman single-thread, pemrograman miltiprocess, dan pemrograman multi-thread.

Dalam pemrograman single-thread, setiap statemen masing-masing akan dieksekusi satu persatu layaknya sebuah antrian. Pemrograman seperti ini biasa digunakan pada bahasa pemrograman lawas, dimana setiap kode akan dibaca oleh *Central Processing Unit* (CPU). Pada pemrogaman single-thread ini, apabila suatu statement tidak tereksekusi, maka statement berikutnya juga tidak akan tereksekusi. Dengan kata lain, apabila sebuah variable sedang diakses oleh salah satu statement maka dapat dijamin jika variable itu tidak diakses oleh statement yang lain. Jadi, programmer dapat dengan mudah menentukan statement mana yang mengalami masalah apabila terjadi error pada baris program tersebut.



Berbeda dengan single-thread yang mengeksekusi setiap statement secara berurutan, pada multiproses setiap statement dapat dieksekusi sendiri-sendiri. Sedangkan untuk pemrograman multithread, maka dapat dilihat contohnya pada layar desktop ketika user menjalankan beberapa aplikasi GUI. Berbeda dengan pemrograman multiproses, pada multithread, statement berjalan terpisah namun masih menggunakan memory yang sama pada komputer, sedangkan pada pemrograman multiproses, memory yang digunakan oleh setiap eksekusi statement akan berbeda.

6.1.1 Multithread pada Java

Eksekusi thread pada java diimplementasikan oleh kelas java.lang.Thread. sedangkan kode untuk tugas-tugas yang dirancang untuk dijalankan di thread direpresentasikan pada kelas interface java.lang.Runnable.

Thread Class

Kelas java.lang.Thread menyediakan method untuk memulai (*start*), menghentikan (*stop*), dan melanjutkan (*resume*) thread, serta untuk mengontrol aspek lain seperti prioritas thread. Diantara method-method yang didefinisikan kelas thread ditunjukkan pada **Tabel** *6.1*.

Tabel 6.1 diantara Method pada kelas Thread

Method	**Deskripsi**
static Thread currentThread()	Mendapatkan referensi object Thread yang sedang dieksekusi.
String getName()	Mendapatkan nama dari thread.
int getPriority()	Mendapatkan prioritas dari thread nilainya 1 – 10. Semakin tinggi nilainya maka prioritas thread tsb semakin tinggi.
Boolean isAlive()	Mengetes thread apakah masih aktif!
void join() void join(long millis) void join(long millis, int nanos)	Menunggu hingga thread ini selesai dieksekusi. .
void run()	Method yang pertama kali akan dieksekusi saat thread dibuat.
String setName()	Menset nama dari thread.
static void sleep(long millis) static void sleep(long millis, int nanos)	Menangguhkan eksekusi dari thread yang sedang berjalan untuk sementara waktu
void start()	Byte output Stream yang menambahkan method untuk memudahkan proses menulis ke suatu output. Method yang ditambahkan adalah print() dan println(). Object yang referensinya dipegang oleh **Sytem.out** juga bertipe **PrintStream**.
void setDaemon(Boolean on)	Bila nilai on adalah true maka akan memanggil thread ini terlebih dahulu sebelum menjalankan

Cara mudah untuk menggunakan kelas Thread adalah dengan melakukan *extend* dan meng-*override* method run(). Berikut ini contoh yang menunjukkan bagaimana melakukan extend terhadap kelas Thread dan memulai beberapa kejadian yang berkelanjutan, masing-masing pada Thread yang berbeda.

Contoh 6.1. membuat aplikasi multithread dengan kelas Thread

```
public class ExtendThread extends Thread {

    int threadNumber;

    public ExtendThread(int num) {
        threadNumber = num;
    }

    //Run method is executed when thread first started
    public void run() {
        System.out.println("I'm thread number: " + threadNumber);
        try {
            Thread.sleep(5000);
        } catch (InterruptedException ex) {
            System.out.println(threadNumber + " is finished");
        }
        System.out.println(threadNumber + " is finished!");
    }

    public static void main(String[] args) {
        System.out.println("Creating thread 1");
        Thread t1 = new ExtendThread(1);

        System.out.println("Creating thread 2");
        Thread t2 = new ExtendThread(2);

        //start both threads
        t1.start();
        t2.start();
    }
}
```

Output:

Contoh ini menunjukkan Thread yang dibuat dan output dari masing-masing thread. Thread sleep selama 5 second untuk mensimulasikan suatu pekerjaan kemudian menghentikannya.

```
run:
Creating thread 1
Creating thread 2
I'm thread number: 1
I'm thread number: 2
1 is finished!
2 is finished!
BUILD SUCCESSFUL (total time: 5 seconds)
```

Runnable Interface

Melakukan extend kelas Thread tidak selalu bisa digunakan untuk membuat aplikasi multithread. Hal ini karena java hanya mendukung *single inheritance* sehingga jika suatu kelas telah *extend* kelas Thread maka tidak bisa melakukan extend dari kelas lain yang bisa jadi juga diperlukan. Untuk itu, maka interface Runnable dapat digunakan untuk membuat aplikasi multithread. Berikut ini contoh implementasi dari interface runnable.

Contoh 6.2. membuat aplikasi multithread dengan *Runnable Interface*

```
public class RunnableThread implements Runnable{

    // Run method is executed when thread first started
    public void run() {
        System.out.println("I am an instance of the java.lang.Runnable interface");
    }

    // Main method to create and start threads
    public static void main(String args[]) {
        System.out.println("Creating runnable object");

        // Create runnable object
        Runnable run = (Runnable) new RunnableThread();

        // Create a thread, and pass the runnable object
        System.out.println("Creating first thread");
        Thread t1 = new Thread(run);

        // Create a second thread, and pass the runnable object
        System.out.println("Creating second thread");
        Thread t2 = new Thread(run);

        // Start both threads
        System.out.println("Starting both threads");
        t1.start();
        t2.start();
    }
}
```

Output:

```
run:
Creating runnable object
Creating first thread
Creating second thread
Starting both threads
I am an instance of the java.lang.Runnable interface
I am an instance of the java.lang.Runnable interface
BUILD SUCCESSFUL (total time: 0 seconds)
```

Perbedaan kelas Thread dan Runnable pada java:

1. Didalam kelas thread pada java, kita tidak bisa memperpanjang thread yang telah kita buat. Dikarenakan java sendiri tidak mendukung banyak turunan layaknya bahasa pemrograman lain seperti C++. Maka dari itu, implementasi kelas Runnable lebih dianjurkan dikarenakan kita lebih diberi kebebasan untuk memperpanjang kelas sebanyak yang kita mau
2. Ketika kita menggunakan kelas Runnable, kita mendapatkan kebebasan untuk menggunakan kembali behavior class yang sudah tidak terpakai. Berbeda dengan thread class, behavior yang sudah tidak terpakai tidak akan dapat unutk di gunakan lagi.
3. Runnable thread lebih cocok bagi programmer yang berorientasi objek.

6.1.2 Mengontrol Thread

Interrupting

Fungsi dari interrupt ini digunakan untuk membangunkan thread yang sedang tertidur atau dalam posisi idle atau tidak digunakan. Apabila program ingin mengaktifkan thread yang sedang dalam masa idlenya, maka fungsi Thread.Interrupt() ini dapat digunakan.

Contoh 6.3. Melakukan *Interrupt* pada Thread

```
public class InterruptingThread extends Thread {

    // Run method is executed when thread first started
    public void run() {
        System.out.println("I feel sleepy. Wake me in eight hours");
        try {
            // Sleep for eight hours
            Thread.sleep(1000 * 60 * 60 * 8);
            System.out.println("That was a nice nap");
        } catch (InterruptedException ie) {
            System.err.println("Just five more minutes....");
        }
    }

    public static void main(String args[]) throws java.io.IOException {
        // Create a 'sleepy' thread
        Thread sleepy = new InterruptingThread();
        // Start thread sleeping
        sleepy.start();
        // Prompt user and wait for input
        System.out.println("Press enter to interrupt the thread");
        System.in.read();
        // InterruptingThread the thread
        sleepy.interrupt();
    }
}
```

Output:

```
run:
Press enter to interrupt the thread
I feel sleepy. Wake me in eight hours
```

Dan jika kita ketikan “enter” maka:

```
run:
Press enter to interrupt the thread
I feel sleepy. Wake me in eight hours

Just five more minutes....
BUILD SUCCESSFUL (total time: 21 seconds)
```

Didalam program diatas, thread ditidurkan dengan waktu yang tak terbatas, sedangkan thread tidak bisa untuk membangunkan dirinya sendiri. Maka dari itu apabila kita ketikan enter maka thread akan terbangun.

Stopping

Sebagian thread dapat mengirimkan perintah stop kepada thread yang lain, dengan cara menggunakan perintah Thread.Stop().

Contoh 6.4. Menghentikan Thread

```
public class StoppingThread extends Thread {

    // Run method is executed when thread first started
    public void run() {
        int count = 1;
        System.out.println("I can count. Watch me go!");
        for (;;) {
            // Print count and increment it
            System.out.print(count++ + " ");
            // Sleep for half a second
            try {
                Thread.sleep(500);
            } catch (InterruptedException ie) {
            }
        }
    }

    public static void main(String args[]) throws java.io.IOException {
        // Create and start counting thread
        Thread counter = new StoppingThread();
        counter.start();
        // Prompt user and wait for input
        System.out.println("Press any enter to stop the thread counting");
        System.in.read();
        // Interrupt the thread
        counter.stop();
    }
}
```

Output:

```
run:
Press any enter to stop the thread counting
I can count. Watch me go!
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
```

Begitu menekan “enter” maka Thread akan terhenti

```
run:
Press any enter to stop the thread counting
I can count. Watch me go!
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
BUILD SUCCESSFUL (total time: 10 seconds)
```

Suspending/resuming

Didalam java terdapat fungsi untuk memberhentikan kerja thread sementara, sehingga thread masih dapat digunakan kembali. Mekanisme pemberhentian thread sementara didalam java dikenal dengan fungsi “suspend” dengan menggunakan method Thread.Suspend(). Dan untuk membangkitakannya maka menggunakan method Thread.Resume().

Contoh 6.5. Suspending/Resuming Thread

```
public class SuspendingThread extends Thread {

    // Run method is executed when thread first started
    public void run() {
        int count = 1;
        System.out.println("I can count. Watch me go!");
        for (;;) {
            System.out.print(count++ + " ");
            try {
                Thread.sleep(500);
            } catch (InterruptedException ie) {
            }
        }
    }

    public static void main(String args[]) throws java.io.IOException {
        Thread counter = new SuspendingThread();
        counter.start();
        System.out.println("Press any enter to SUSPEND the thread counting");
        System.in.read();
        counter.suspend();
        System.out.println("Press any enter to RESUME the thread counting");
        System.in.read();
        counter.resume();
        System.out.println("Press any enter to STOP the thread counting");
        System.in.read();
        counter.stop();
    }
}
```

Output:

```
run:
Press any enter to SUSPEND the thread counting
I can count. Watch me go!
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
```

Begitu menekan enter maka thread akan terhenti sementara

```
run:
Press any enter to SUSPEND the thread counting
I can count. Watch me go!
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 Press any enter to RESUME the thread counting
```

Thread dapat dilanjutkan setelah menekan “enter” lagi

```
run:
Press any enter to SUSPEND the thread counting
I can count. Watch me go!
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 Press any enter to RESUME the thread counting

Press any enter to STOP the thread counting
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
```

Dan kali ini thread berhenti begitu menekan “enter”

```
run:
Press any enter to SUSPEND the thread counting
I can count. Watch me go!
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 Press any enter to RESUME the thread counting

Press any enter to STOP the thread counting
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
39 BUILD SUCCESSFUL (total time: 47 seconds)
```

Waiting

Terkadang kita tidak tahu apakah thread sedang aktif atau tidak, maka untuk mengeceknya kita dapat menggunakan method isAlive() sebagai fungsi untuk mengecek apakah thread sedang aktif atau tidak. Tetapi cara ini sangat tidak efektif dikarenakan akan membenani CPU terlalu berat dengan pengulangan untuk cek apakah thread sedang aktif atau tidak. Maka dari itu, terdapat method yang lebih pas digunakan yaitu dengan menggunakan method Thread.Join(), dengan menggunakan method ini thread yang sedang aktif akan dipantau sampai thread tersebut berhenti bekerja, dan setelah thread tersebut berhenti bekerja maka aka nada pemberitahuan bahwa thread tersebut sudah tidak aktif.

Contoh 6.6. Menunggu hingga Thread mati

```
public class WaitingThreadDead extends Thread {

    // Run method is executed when thread first started
    public void run() {
        System.out.println("This thread feels a little ill....");
        // Sleep for five seconds
        try {
            Thread.sleep(5000);
        } catch (InterruptedException ie) {
        }
    }

    public static void main(String args[]) throws java.lang.InterruptedException {
        // Create and start dying thread
        Thread dying = new WaitingThreadDead();
        dying.start();
        // Prompt user and wait for input
        System.out.println("Waiting for thread death");
        // Wait till death
        dying.join();
        System.out.println("Thread has died");
    }
}
```

Output:

```
run:
Waiting for thread death
This thread feels a little ill....
```

Method join() terus memantau Thread yang berjalan. Setelah 5 second (batas waktu thread tidur yang telah ditentukan) maka tampil pemberitahuan bahwa Thread telah tidak aktiv.

```
run:
Waiting for thread death
This thread feels a little ill....
Thread has died
BUILD SUCCESSFUL (total time: 5 seconds)
```

6.2 Praktikum

1. Tulis dan jalankan **Contoh 6.1** dan **Contoh 6.2** pada modul ini dan pahami tiap barisnya!
2. **Program 1**
    a. Buatlah buatlah sebuah class **Kirim** yang merupakan turunan dari class thread dimana dalam kelas ini terdapat prosedur yang pertama kali dijalankan. Keluaran

dari proses merupakan bilangan ganjil dan ditampilkan ke layar yang di ulang sebanyak 10 kali.

b. Lalu buatlah class **Terima** yang juga merupakan turunan dari class thread dimana dalam kelas ini terdapat prosedur yang pertama kali dijalankan. Keluaran dari proses merupakan bilangan genap dan ditampilkan ke layar yang di ulang sebanyak 10 kali.

c. Buatlah kelas utama dengan nama **ThreadUtama** yang didalamnya menjalankan kelas terima dan kelas kirim secara bersama sama.

3. Buatlah class-class seperti pada **program 1** hanya saja class **Kirim** dan class **Terima** mengimplementasikan interface **Runnable**
4. Tulis dan jalankan **Contoh 6.3**, **Contoh 6.4**, **Contoh 6.5**, dan **Contoh 6.6** pada modul ini dan pahami tiap barisnya!

6.3 Tugas

1. Buatlah program yang mampu menjumlahkan 3 buah Array bertipe integer berikut:

   Array 1: 6, 9, 1, 2, 3, 5

   Array 2: 7, 11, 6, 4, 3, 1

   Array 3: 5, 4, 3, 2, 1, 12

   ---------- +

   Hasil = 18, 24, 8, 7, 18

   Dengan aturan terdapat 6 Thread dimana Thread 1-6 tugasnya menjumlahkan array sesuai dengan indexnya...

   Misalkan:
   Thread 1 = 6 + 7 + 5 = 18

   Thread 2 = 9 + 11 + 4 = 24

   …dst

   Total: 75

2. Buatlah program java dengan menggunakan thread dengan hasil output menceritakan kisah sebagai berikut:

   Disuatu rumah sakit terdapat 1 dokter, 2 perawat dan 1 kasir. Datanglah 10 pasien kemudian diperiksa oleh dokter. Setelah 10 pasien diperiksa dokter, perawat 1 datang

dan merawat 5 pasien dan perawat 2 merawat sisa pasien lain. Setelah dirawat, pasien langsung membayar biaya perawatan dikasir.

3. Mengontrol Thread dapat dilakukan dengan menggunakan method *stop()*, *suspend()*, ataupun *resume()*. Selain menggunakan method tersebut, kita bisa menggunakan variable bertipe boolean untuk pengontrolnya.

   Buatlah program menggunakan thread. Saat program dijalankan maka akan tampil counter bilangan genap, misal: 2 4 6 8 10 12 14 16….dst. Ketika user menekan "enter" maka dilakukan counter mundur bilangan ganjil, misal: 15 13 11 9 7 5 3 1…dst. hingga user kembali menekan "enter" sehingga counter dihentikan. Pengontrolan thread menggunakan variable Boolean.

4. Buatlah aplikasi TCP client-server sederhana yang mendukung multiclient. Agar server mampu menangani lebih dari satu client, maka thread harus diimplementasikan pada server. Sehingga server mampu menjalankan lebih dari satu pemanggilan method yang sama.

   Server terus berjalan. Setiap kali ada client yang terhubung, maka server menampilkan IP address dan port client tersebut.

   Client yang berhasil terhubung dengan server akan mendapatkan pesan dari server beserta nilai counter yang menunjukkan urutan client yang terhubung.

   Misal sebagaimana berikut:

Server

```
run:
Client connect: /127.0.0.1 - on port 63538
Client connect: /127.0.0.1 - on port 63539
Client connect: /127.0.0.1 - on port 63540
Client connect: /127.0.0.1 - on port 63545
Client connect: /127.0.0.1 - on port 63548
```

Output pada server sebagai notifikasi client yang terhubung

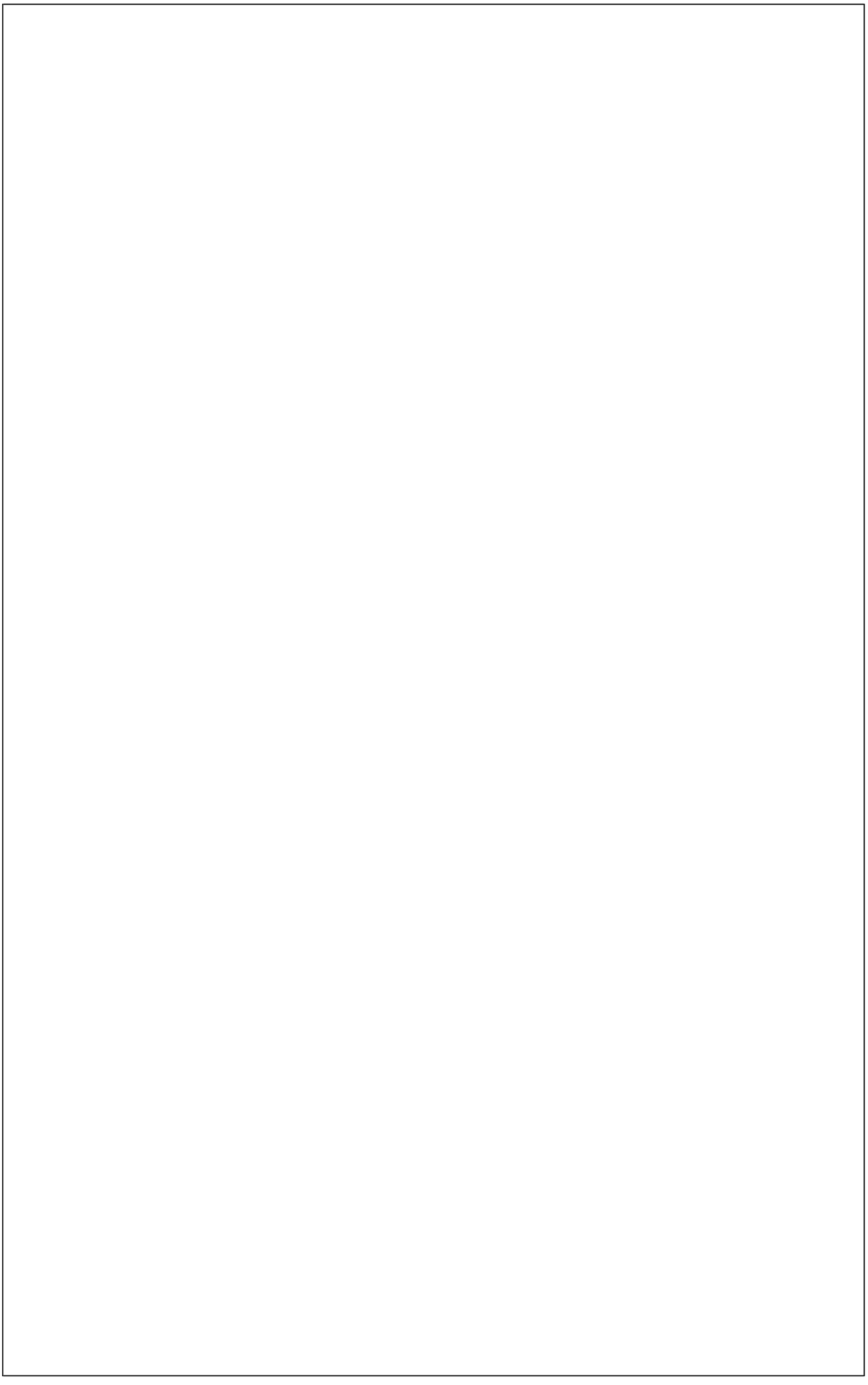
Client ke-n

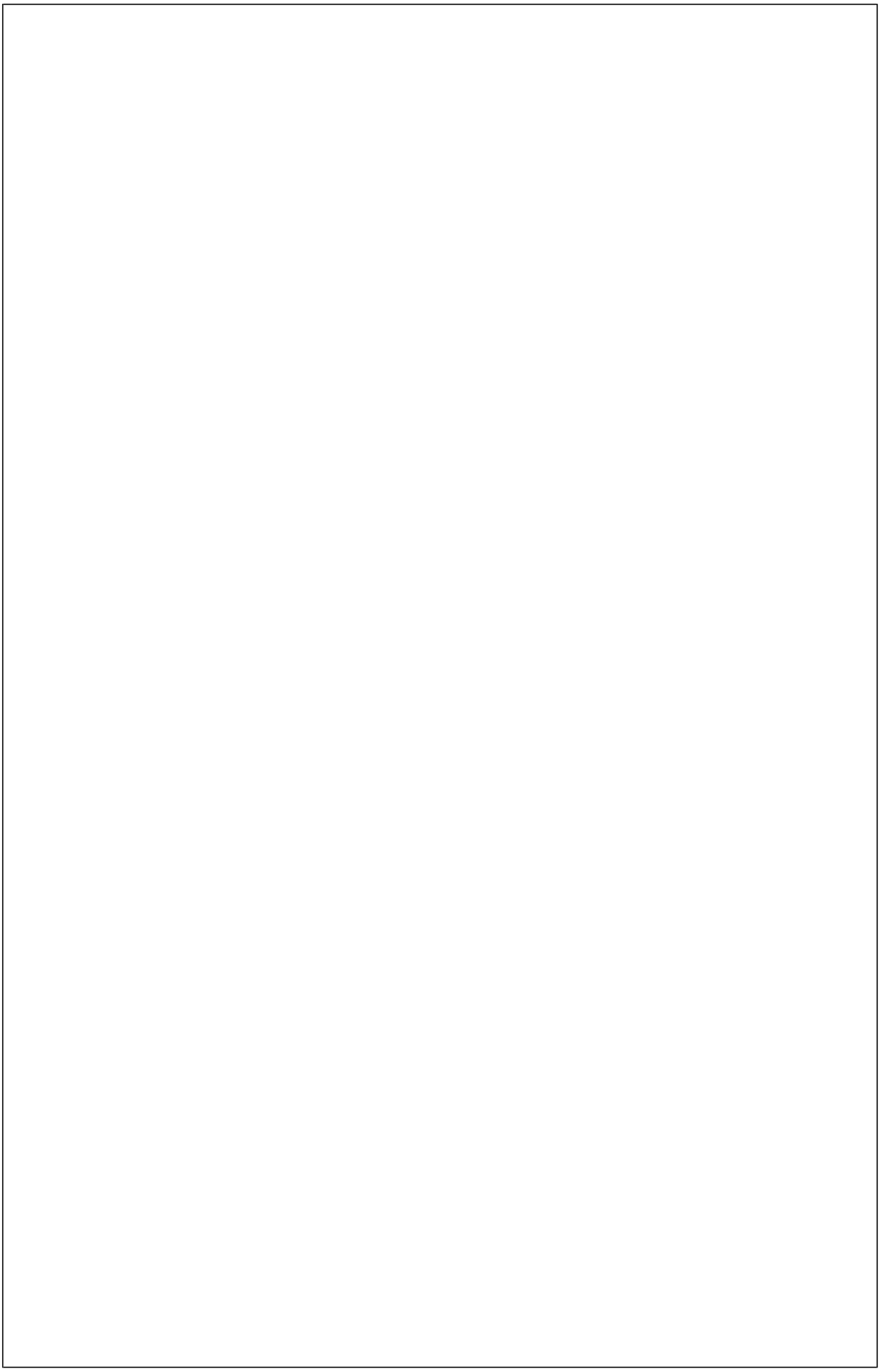
```
run:
Ahlan wa sahlan, anda pengunjung ke-1
BUILD SUCCESSFUL (total time: 0 seconds)
run:
Ahlan wa sahlan, anda pengunjung ke-2
BUILD SUCCESSFUL (total time: 1 second)
run:
Ahlan wa sahlan, anda pengunjung ke-3
BUILD SUCCESSFUL (total time: 0 seconds)

run:
Ahlan wa sahlan, anda pengunjung ke-4
BUILD SUCCESSFUL (total time: 0 seconds)
run:
Ahlan wa sahlan, anda pengunjung ke-5
BUILD SUCCESSFUL (total time: 0 seconds)
```

Output pada client, balasan dari server setalah client ke-n terhubung dengan server. Beberapa client dapat connect server bersamaan

Jawaban Tugas Pengembangan





MODUL 7

MULTITHREAD PROGRAMMING (LANJUTAN)

7.1 Pembahasan

Modul sebelumnya (modul 6) telah membahas pengelanan thread, membuat thread pada java, dan mengontrol thread. Lebih lanjut, modul ini membahas tentang sinkronisasi pada multithread dan komunikasi antar thread.

7.1.1 Sinkronisasi

Sinkronisasi adalah suatu proses pengendalian akses dari sumber daya terbagi pakai (*shared resource*) oleh banyak thread sedemikian sehingga hanya satu thread yang dapat mengakses sumber daya tertentu pada satu waktu.

Dalam aplikasi multithreaded yang tidak tersinkronisasi, sangat mungkin terjadi adanya satu thread memodifikasi suatu obyek yang dipakai bersama pada saat thread lain sedangkan dalam proses menggunakan atau mengupdate nilai obyek tersebut. Sinkronisasi mencegah jenis kerusakan data demikian, jika tidak disinkronkan maka dapat mengakibatkan pembacaan yang buruk dan error yang signifikan. Secara umum bagian kritis (*critical sections*) dari kode biasanya ditandai dengan kata kunci *synchronized*.

Terdapat 2 (dua) bagian kode yang dapat dikenakan sinkronisasi di dalam Java:

- *synchronized method*
- *synchronized block*

Sinkronisasi pada method mencegah dua thread mengeksekusi method pada object dan dan waktu yang sama. Method yang disinkronisasi ditandai dengan kata kunci *synchronized*. Ketika method tersebut dipanggil maka thread akan mengaktivkan sebuah *object-lock* atau monitor. Jika thread lain mencoba untuk memanggil method tersebut maka akan didapati bahwa method tersebut terkunci dan berada dalam keadaan suspense sampai kunci/monitor terbuka.

Sinkronisasi pada method umumnya digunakan untuk mengsinkronisasi akses pada resource. Pada aplikasi multhithread, method dapat disinkronisasi untuk mencegah hilang dan rusaknya data. Contohnya adalah sebuah counter yang disimpan dalam sebuah file, misalkan counter nilai berapa kali sebuah aksi seperti mengakses web terjadi. Counter ini dapat bertambah (dengan cara membaca nilai saat ini dan menulis yang baru) atau dibaca

oleh beberapa thread. Jika satu thread mencoba untuk menambah nilai counter sebelum thread yang lain selesai memodifikasi counter, dengan kata lain nilai nilai tersebut di-*set* oleh satu thread dan di-*override* oleh yang lain. Itu berarti counter akan membaca nilai yang tidak valid. Terlebih lagi, jika ada dua upaya untuk meng-*override* nilai tersebut, maka file dapat rusak. Jika method untuk mengakses dan memodifikasi nilai counter disinkronisasikan, maka hanya ada satu thread yang bisa melakukan satu tindakan menulis pada satu waktu tertentu.

Contoh 7.1. Sinkronisasi pada level method

Kelas berikut mendefinisikan sebuah counter dengan method akses dan modifikasi yang disinkronisasi.

```
public class Counter {

    private int countValue;

    public Counter() {
        countValue = 0;
    }

    public Counter(int start) {
        countValue = start;
    }

    // Synchronized method to increase counter
    public synchronized void increaseCount() {
        int count = countValue;
        // Simulate slow data processing and modification
        try {
            Thread.sleep(5);
        } catch (InterruptedException ie) {
        }
        count = count + 1;
        countValue = count;
    }

    // Synchronized method to return counter value
    public synchronized int getCount() {
        return countValue;
    }
}
```

Kelas berikut adalah sebuah aplikasi yang menggunakan beberapa thread dengan counter tunggal.

```
public class CountingThread implements Runnable {

    Counter myCounter;
    int countAmount;
    // Construct a counting thread to use the specified counter
    public CountingThread(Counter counter, int amount) {
        myCounter = counter;
        countAmount = amount;
    }

    public void run() {
        // Increase the counter the specified number of times
        for (int i = 1; i <= countAmount; i++) {
            // Increase the counter
            myCounter.increaseCount();
            System.out.println(""+myCounter.getCount());
        }
    }

    public static void main(String args[]) throws Exception {
        // Create a new, thread-safe counter
        Counter c = new Counter();
        // Our runnable instance will increase the counter
        // ten times, for each thread that runs it
        Runnable runner = new CountingThread(c, 10);
        System.out.println("Starting counting threads");
        Thread t1 = new Thread(runner);
        Thread t2 = new Thread(runner);
        Thread t3 = new Thread(runner);
        t1.start(); t2.start(); t3.start();
        // Wait for all three threads to finish
        t1.join(); t2.join(); t3.join();
        System.out.println("Counter value is " + c.getCount());
    }
}
```

Output:

Tiga thread dijalankan dimana masing-masing menambahkan counter 10 kali. Ini berarti dengan adanya sinkronisasi maka method getCount() akan mengembalikan nilai total sejumlah 30. Sedangkan tanpa adanya sinkronisasi maka total nilai akan sangat kurang.

```
run:
Starting counting threads
 1 2 3 4 5 6 7 8 9 10 11 12 13 15 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Counter value is 30
BUILD SUCCESSFUL (total time: 0 seconds)
```

7.1.2 Komunikasi Antar Thread *(Inter-thread Communication)*

Pada pemrograman multithread adakalanya penting memperhatikan urutan eksekusi dan pengontrolan agar sebuah tugas tidak dapat dilakukan sebelum tugas yang lain selesai dilakukan. Pada dasarnya memungkinkan bagi thread untuk menunggu sampai thread yang lain mati (yang

mengindikasikan bahwa pekerjaannya telah selesai), hal ini dilakukan dengan menggunakan method join(). Namun, bagaimana jika sebuah thread melakukan tugas yang sedang berlangsung dan tidak pernah berakhir?

Solusi kondisi tersebut adalah dengan adanya komunikasi antar thread. Yaitu memberi tahu thread lain bahwa tugas telah selesai sedangkan thread lain menunggu sampai mereka diberi tahu. Notifikasi ini bisa menjadi proses berulang (dengan beberapa siklus menunggu dan memberitahu). Hal ini memungkinkan bagi thread untuk menyinkronkan tindakan dan komunikasi mereka. Untuk mengimplementasikan tindakan ini maka dilakukan dengan penggunaan method wait() untuk thread yang menunggu dan method notify() atau naotifyAll() untuk thread yang memberitahu, ketengan method-method ini dapat dilihat pada **Tabel *7.1***.

Tabel 7.1 Method untuk komunikasi antar thread

Method	**Diskripsi**
public final void wait()	Menyebabkan thread ini menunggu sampai thread yang lain memanggil *notify* atau *notifyAll* method dari object ini. Hal ini dapat menyebabkan *InterruptedException*.
public final void notify()	Membangunkan thread yang telah memanggil method *wait* dari object yang sama.
public final void notifyAll()	Membangunkan semua thread yang telah memanggil method *wait* dari object yang sama.

Contoh 7.2. Komunikasi antar thread

Contoh berikut mendemontrasikan penggunaan method wait() dan notify(). Contoh ini cukup sederhana. Thread aplikasi utama menunggu sampai diberitahu oleh thread kedua bahwa suatu peristiwa telah terjadi. Sampai user menekan “enter”, thread kedua tidak akan mengirimkan pesan pemberitahuan. Contoh ini tanpa menggunakan perulangan, dapat juga ditambahkan perulangan yang menunjukkan rangakaian siklus menunggu dan memeberitahu.

```
public class WaitNotify extends Thread {

    public static void main(String args[]) throws Exception {
        Thread notificationThread = new WaitNotify();
        notificationThread.start();
        // Wait for the notification thread to trigger event
        synchronized (notificationThread) {
            notificationThread.wait();
        }
        // Notify user that the wait() method has returned
        System.out.println("The wait is over");
    }

    public void run() {
        System.out.println("Hit enter to stop waiting thread");
        try {
            System.in.read();
        } catch (java.io.IOException ioe) {
        }
        // Notify any threads waiting on this thread
        synchronized (this) {
            this.notifyAll();
        }
    }
}
```

Output:

```
run:
Hit enter to stop waiting thread

The wait is over
BUILD SUCCESSFUL (total time: 5 seconds)
```

7.2 Praktikum

1. Tulis dan jalankan **Contoh 7.1** pada modul ini dan pahami tiap barisnya!

   Pada **Contoh 7.1** yang telah dituliskan, hilangkan kata kunci *synchronized* pada method di kelas counter. Jalankan program dan lihat bagaimana hasilnya!
2. Tulis dan jalankan **Contoh 7.2** pada modul ini dan pahami tiap barisnya!
3. Buatlah layanan *chef-waitress* sederhana dengan memuat kelas *waitress* dan kelas *chef*.

   Thread pada kelas *waitress* menunggu sampai thread pada kelas chef memberitahu bahwa tugasnya (membuat makanan sesuai jumlah pesanan) telah selesai. Setiap kali menetima notifikasi dari chef maka muncul pesan bahwa “pelayan mengantarkan makanan”

Pada kelas *chef*, thread melakukan penambahan counter yang menunjukkan jumlah pesanan yang telah dimasak dan menampilkan pesan, misalkan “pesanan ke 1 selesai”. Begitu nilai counter penambahan mencapai jumlah pesanan pesanan tertentu maka *chef* akan memberitahu *waitress*.

Tambahkan kelas *kitchen* yang menjalankan masing-masing thread pada kelas *chef* dan *waitress*.

Tidak ada tugas pengembangan untuk modul 7.

Mahasiswa diharap mempersiapkan diri untuk mengikuti evaluasi tengah semester matakuliah praktikum Sistem Terdistribusi dan Keamanan dalam bentuk Live coding.



MODUL 8
JADE PROGRAMMING

8.1 Pembahasan

Istilah *middleware* dalam dunia komputer biasanya diberikan kepada suatu obyek yang bertugas menjembatani heterogenitas lingkungan seperti sistem operasi, aplikasi, bahasa pemrograman, jaringan, dan lain-lain sehingga kerumitan menjadi transparan bagi pengguna atau aplikasi yang menggunakannya.

JADE (Java Agent Development Framework) adalah *middleware* yang dapat digunakan untuk mengembangkan dan menjalankan aplikasi *peer to peer* yang berdasarkan pada paradigma agent. Sesuai namanya bahasa pemrograman yang digunakan untuk mengembangkan agent dalam JADE adalah Java.

8.1.1 Pengenalan JADE

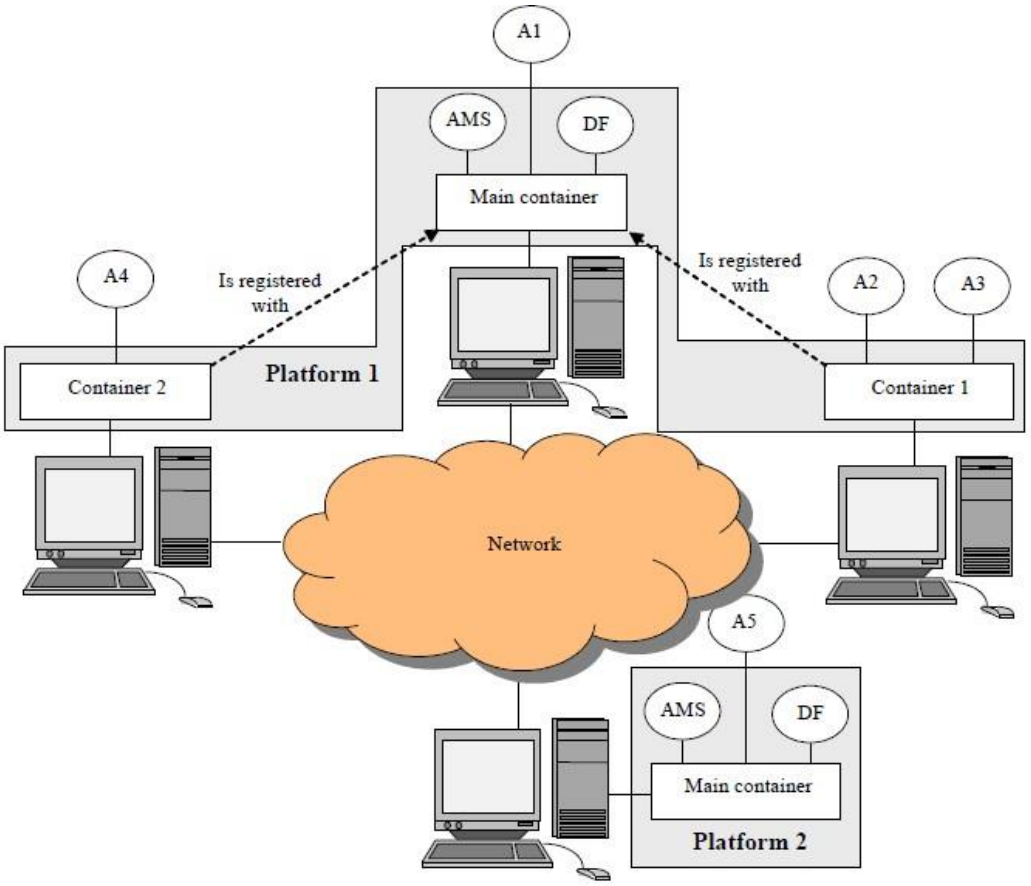
JADE sebagai middleware yang memberikan fasilitas untuk pengembangan sistem berbasiskan agent menyediakan:

1. *Runtime environment* yang menjadi tempat di mana agent dapat berjalan dan harus aktif pada host dimana agent akan bekerja.
2. *Library* berupa kelas-kelas yang dapat/harus digunakan untuk mengembangkan agent.
3. Sekumpulan *graphic tool* yang digunakan untuk melakukan administrasi dan monitoring terhadap aktivitas agent yang sedang berjalan pada runtime environment.

Runtime environement dalam JADE dikenal dengan istilah **container**. Satu host dapat menjalankan lebih dari satu container dan setiap container bisa menangani beberapa agent. Sekumpulan container yang aktif disebut sebagai **platform**. Sebuah platform dapat memiliki container yang berasal dari host yang berbeda-beda. Satu platform harus memiliki satu container yang memiliki atribut sebagai main container yang aktif. Semua container yang aktif dan ingin bergabung dalam sebuah platform harus bergabung dengan mendaftarkan diri pada main container dan tidak boleh beratribut sebagai main container atau disebut juga normal container (non-main/ bukan utama).

Gambar 8.1 menggambarkan konsep di atas melalui sebuah contoh skenario yang menunjukkan dua platform JADE yang masing-masing terdiri dari 3 dan 1 container. Agent JADE didefinisikan dengan nama yang unik dan agent tersebut dapat berkomunikasi secara transparan dimanapun lokasi mereka yang sebenarnya, baik pada container yang sama

(misalnya agen A2 dan A3 pada **Gambar 8.1**), pada container yang berbeda dalam platform yang sama (misalnya A1 dan A2) atau platform yang berbeda (misalnya A4 dan A5).



Gambar 8.1 Container dan Platform

Selain menerima pendaftaran dari normal container sebuah main container selalu memiliki dua buah agent yang aktif secara otomatis ketika main container dijalankan. Kedua agent itu adalah:

1. AMS (*Agent Management System*) yang menyediakan naming service yang memastikan setiap agent dalam platform memiliki identitas yang unik. Selain itu AMS dapat merepresentasikan otoritas dalam platform di mana melalui AMS kita dapat menjalankan atau menghentikan agent dalam container yang terdaftar.
2. DF (*Directory Facilitator*) adalah agent yang berfungsi sebagai "yellow pages" bagi platform. Melalui DF sebuah agent dapat mencari agent yang aktif dan layanan yang diberikan agent tersebut.

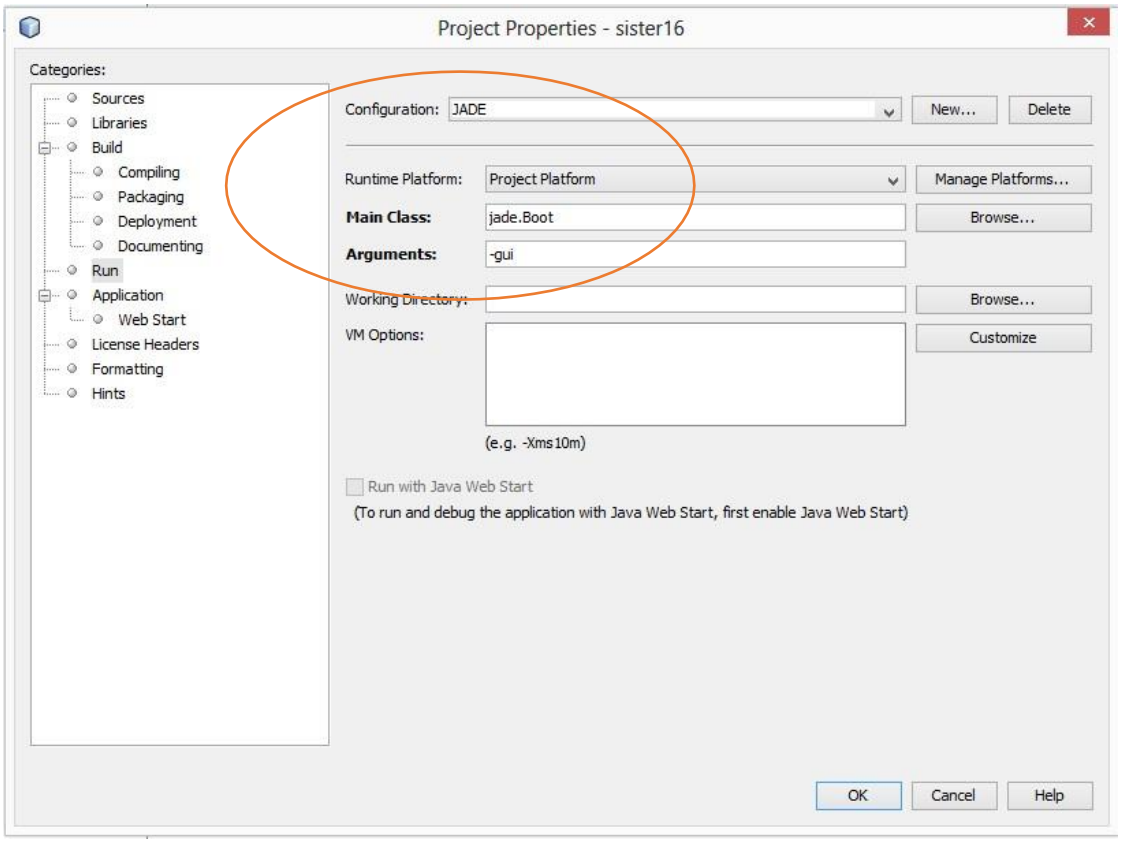
8.1.2 Menjalankan JADE pada IDE Netbeans

Berikut ini langkah-langkah untuk menjalankan JADE dengan menampilkan RMA GUI:

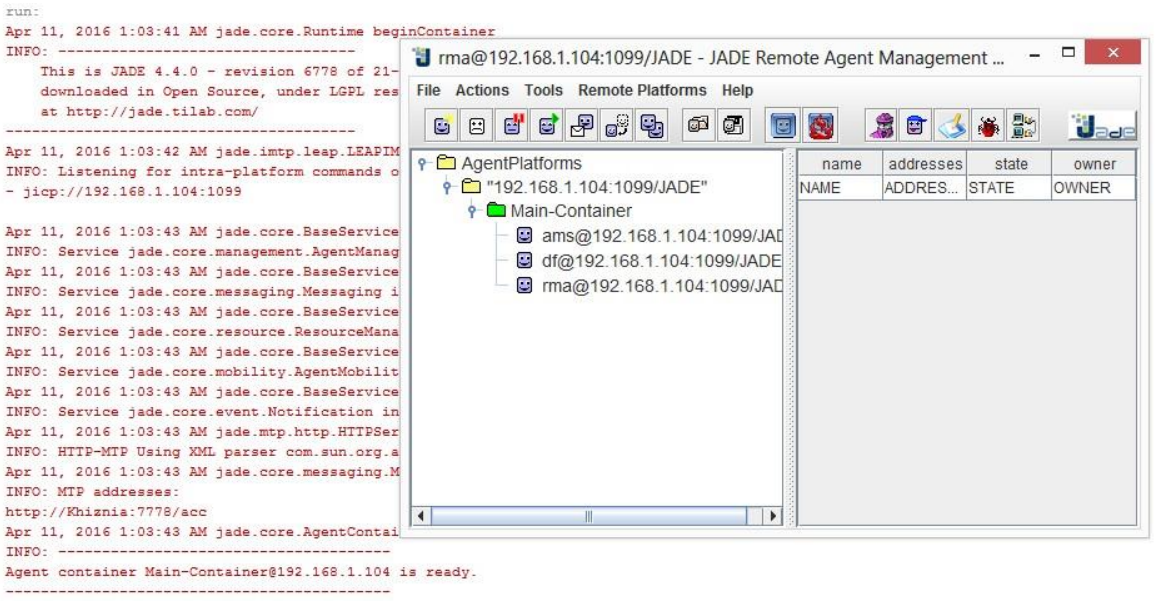
1. Mempersiapkan libraries sesuai dengan versi yang digunakan. Framework JADE dapat diunduh pada situs http://jade.tilab.com/. Untuk JADE versi 4 hanya perlu jade.jar dan commons-codec-1.3.jar, sedangkan versi di bawahnya terdapat tambahan iiop.jar, http.jar, jadeTools.jar.



2. Melakukan konfigurasi dengan cara klik kanan di project → set configuration → customize → kemudian buat configurasi baru dan mengisi main class dengan **jade.Boot** dan arguments **-gui** lalu **OK**



3. Jalankan project dengan klik kanan lalu **Run** sehingga JADE aktiv dan muncul tampilan RMA GUI sebagaimana berikut:



8.1.3 Membuat Agent

Membuat sebuah agent menggunakan JADE dilakukan dengan mendefinisikan sebuah kelas yang *extends* pada kelas **`jade.core.Agent`** dan mengimplementasikan method **`setup()`** sebagaimana listing program berikut ini:

```
public class HelloWorldAgent extends Agent {

        protected void setup() {
            System.out.println("Hello World. I'm an agent!");
        }
}
```

Setiap agen diidentifikasi oleh sebuah "agen identifier". Dalam JADE agen identifier diwakili sebagai instance dari kelas jade.core.AID. Adanya metode **`getAID()`** pada kelas Agen memungkinkan untuk memperoleh identifikasi agen lokal. Sebuah objek AID memuat nama unik global (GUID) dan sejumlah alamat. Kelas AID menyediakan metode untuk mengambil nama lokal (**`getLocalName()`**), GUID (**`getName()`**) dan alamat (**`getAllAddresses()`**).

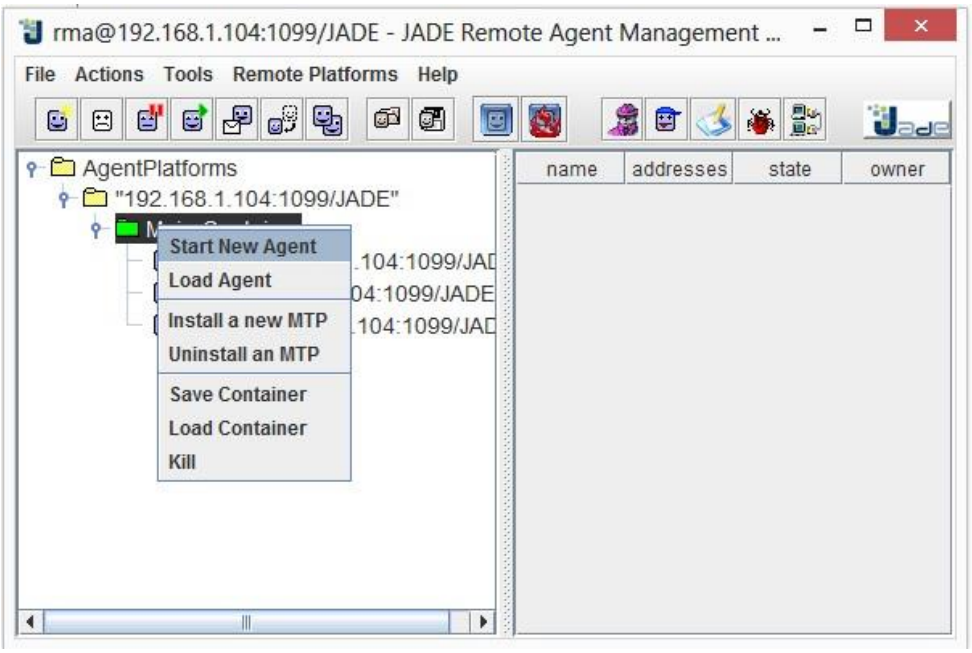
Contoh 8.1. Membuat Agent

```
import jade.core.Agent;
import java.util.Iterator;

public class HelloWorldAgent extends Agent {

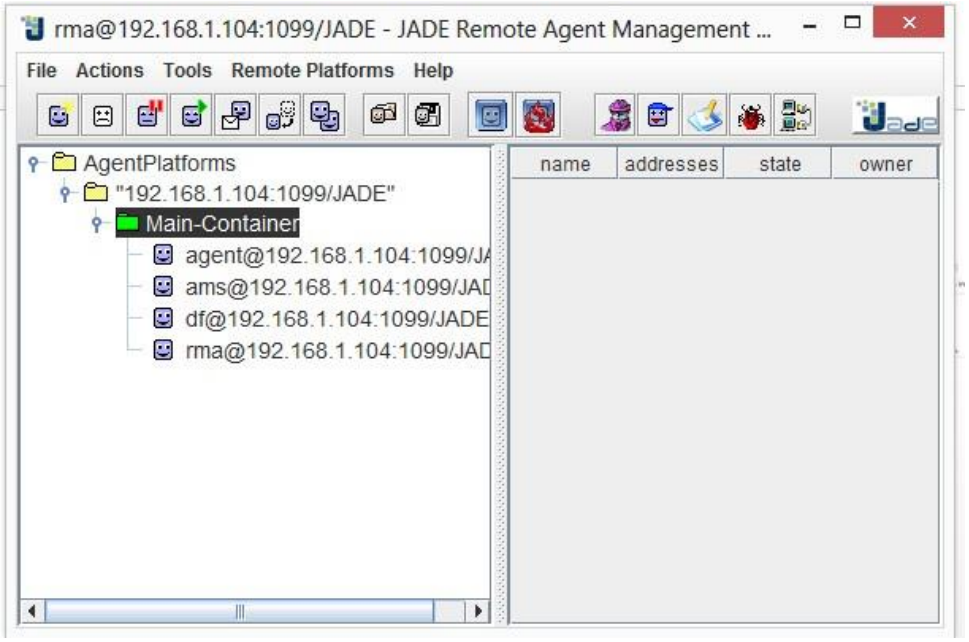
    @Override
    protected void setup() {
        // Printout a welcome message
        System.out.println("Hello World. I'm an agent!");
        // Printout agent identifiers
        System.out.println("My local-name is " + getAID().getLocalName());
        System.out.println("My GUID is " + getAID().getName());
        System.out.println("My addresses are:");
        Iterator it = getAID().getAllAddresses();
        while (it.hasNext()) {
            System.out.println("- " + it.next());
        }
    }
}
```

Untuk menjalankan agent tersebut dapat dilakukan melalui RMA GUI. Klik kanan pada container, pilih start new agent sehingga muncul form untuk mengisi parameter agent. Isikan nama agent dan pilih kelas yang akan dijalankan, kemudian OK.





Begitu agent dijalankan maka nama agent yang dibuat akan muncul pada list agent pada container serta tampak output dari program agent tersebut pada console.



Output:

```
INFO: ----------------------------------------
Agent container Main-Container@192.168.1.104 is ready.
--------------------------------------------
Hello World. I'm an agent!
My local-name is agent
My GUID is agent@192.168.1.104:1099/JADE
My addresses are:
- http://Khiznia:7778/acc
```

8.1.4 Tugas Agent

Pekerjaan yang sebenarnya dari agen terletak dalam suatu "Behaviour". Behaviour merupakan suatu tugas yang agen dapat melakukan dan diimplementasikan sebagai objek dari suatu kelas dari kelas turunan Behaviour. Terdapat beberapa macam behaviour dengan sifat yang berbeda-beda, seperti:

- ***One-Shoot Behaviour*** yang memiliki sifat pengerjaan hanya satu fase eksekusi. Ketika OneShootBehaviour dijalankan maka pada akhirnya secara otomatis akan menjalankan metode **`done()`** yang berada didalam kelas OneShootBehaviour sehingga proses akan berhenti.

```
public class MyOneShotBehaviour extends OneShotBehaviour {

	public void action() {

		// perform operation X

	}

}
```

Pada contoh tersebut operasi X hanya berjalan sekali

- ***Cyclic Behaviour*** yang memiliki sifat pengerjaan yang tidak akan pernah berhenti dan berulang-ulang.

```
public class MyCyclicBehaviour extends CyclicBehaviour {
	public void action() {
		// perform operation Y
	}
}
```

- ***Ticker Behaviour*** memiliki sifat pengerjaan berulang-ulang dan dalam satu proses terdapat waktu jeda yang ditentukan sebelum menjalankan proses selanjutnya.

Selain ketiga behavior tersebut, dapat juga menggunakan ***Generic Behaviour*** dimana behavior dapat menjalankan operasi yang berbeda sesuai dengan status.

```
public class ThreeStepBehaviour extends Behaviour {
private int step = 0;
	public void action() {
		switch (step) {
		case 0:
			// perform operation X
		step++;
		break;
		case 1:
			// perform operation Y
		step++;
		break;
		case 2:
			// perform operation Z
		step++;
		break;
		}
	}
public boolean done() {
	return step == 3;
}
}
```

Untuk membuat agent dengan tugas tertentu, maka Behaviour dapat didefinisikan dalam kelas agent sebagaimana **Contoh 8.2** atau juga dapat didefinisikan pada kelas lain kemudian dipanggil dalam kelas Agent sebagaimana **Contoh 8.3**.

Contoh 8.2. Mendefinisikan behavior pada kelas Agent

```
import jade.core.Agent;
import jade.core.behaviours.Behaviour;

public class BehaviourAgent extends Agent {

    protected void setup() {
        addBehaviour(new Behaviour(this) {

            @Override
            public void action() {
                System.out.println("local name: " + getAID().getLocalName());
            }

            @Override
            public boolean done() {
                return true;
            }
        });
    }
}
```

Contoh 8.3. Mendefinisikan behavior pada kelas Behaviour

```
import jade.core.Agent;

public class BehaviourAgent extends Agent {

    protected void setup() {
        addBehaviour(new sifat(this) );
    }
}
```

```
import jade.core.Agent;
import jade.core.behaviours.Behaviour;

class sifat extends Behaviour {

    public sifat(Agent a) {
        super(a);
    }

    @Override
    public void action() {
        System.out.println("local name " + myAgent.getLocalName());
    }

    @Override
    public boolean done() {
        return true;
    }

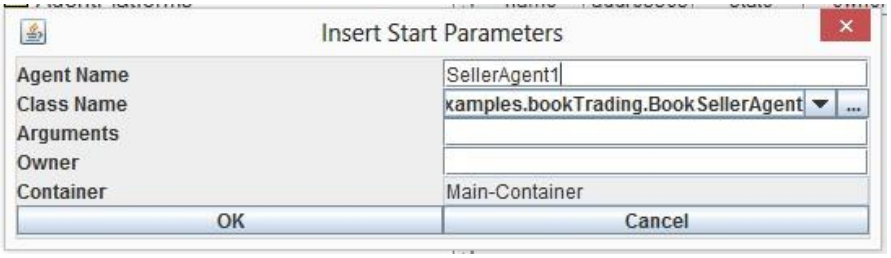
}
```

Kedua contoh tersebut menunjukkan pemanggilan kelas Behaviour, sedangkan untuk turunan kelas tersebut, yaitu OneShotBehaviour, CyclicBehaviour, dan TickerBehaviour, dapat dilakukan dengan cara yang sama dengan menyesuaikan implementasi abstrak method pada masing-masing behavior.

8.1.5 Menjalankan Contoh "Book Trading"

Sebagai gambaran bagaimana implementasi dari software agent, JADE sendiri menyediakan contoh yang dapat dipelajari, misalkan pada package **examples.bookTrading.** Subbab ini menunjukkan langkah-langkah menjalankan program book Trading menggunakan JADE.

1. Masukkan package bookTrading pada project netbeans. Jalankan JADE
2. Buat agent Seller dengan cara klik kanan pada container → start new agent. Tuliskan nama agent, misal "SellerAgent1" dan pilih kelas BookSellerAgent, kemudian OK

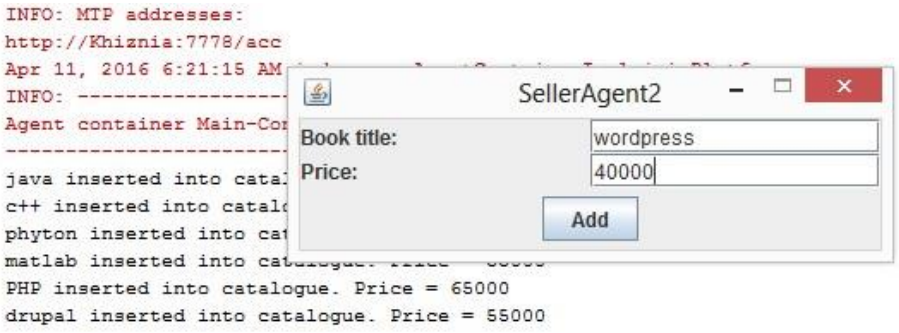


3. Masukkan beberapa data buku (judul dan harga) lewat SellerAgent1

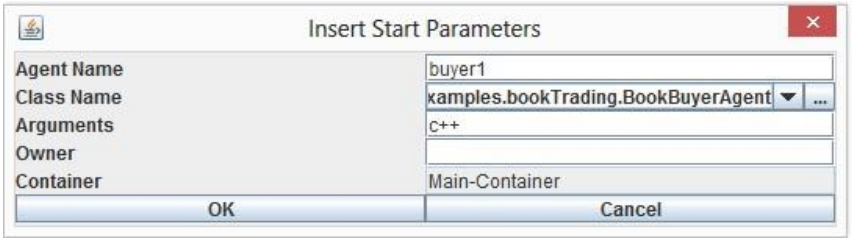


Catatan: biarkan agent tetap berjalan dengan tidak menutup (*close*) tampilan form sellerAgent1

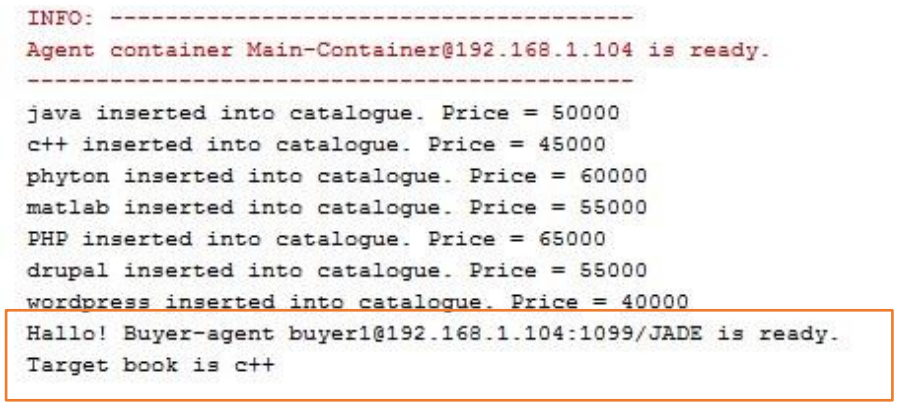
4. Lakukan lagi langkah 2 dan 3. Buat agent seller lain misal dengan nama "SellerAgent2". Kemudian masukkan beberapa data buku lewat SellerAgent2.



5. Anda juga dapat mencoba memasukkan nama buku yang sama dari SellerAgent3.
6. Sekarang buat agent sebagai pembeli, misal "buyer1" dengan memanggil kelas BookBuyerAgent. Sertakan pula argument yang berisi judul buku yang ingin dibeli. Kemudian OK.



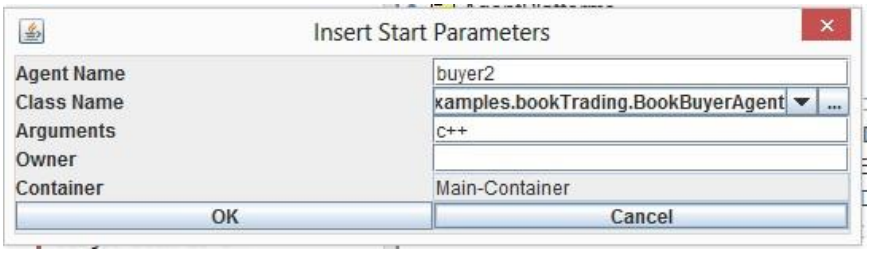
Output:



Tunggu beberapa saat, ketersediaan buku diperiksa baik pada sellerAgent1 maupun sellerAgent2 hingga akhirnya buku yang menjadi target buyer1 berhasil dibeli dari sellerAgent1.

```
Hallo! Buyer-agent buyer1@192.168.1.104:1099/JADE is ready.
Target book is c++
Trying to buy c++
Found the following seller agents:
sellerAgent1@192.168.1.104:1099/JADE
SellerAgent2@192.168.1.104:1099/JADE
c++ sold to agent buyer1@192.168.1.104:1099/JADE
c++ successfully purchased from agent sellerAgent1@192.168.1.104:1099/JADE
Price = 45000
Buyer-agent buyer1@192.168.1.104:1099/JADE terminating.
```

7. Buat agent pembeli yang lain, misal "buyer2" yang juga ingin membeli buku yang sama dengan buyer1.



Lihat bagaimana outputnya.

```
Hallo! Buyer-agent buyer2@192.168.1.104:1099/JADE is ready.
Target book is c++
Trying to buy c++
Found the following seller agents:
sellerAgent1@192.168.1.104:1099/JADE
SellerAgent2@192.168.1.104:1099/JADE
Attempt failed: c++ not available for sale
```

Buyer2 masih dalam keadaan mencari buku yang diinginkan kepada sellerAgent1 dan sellerAgent2

8. Tambahkan buku yang menjadi target buyer2 melalui sellerAgent2. Setelah buku tersimpan dalam katalog, lihat bagaimana hasilnya?

8.2 Praktikum

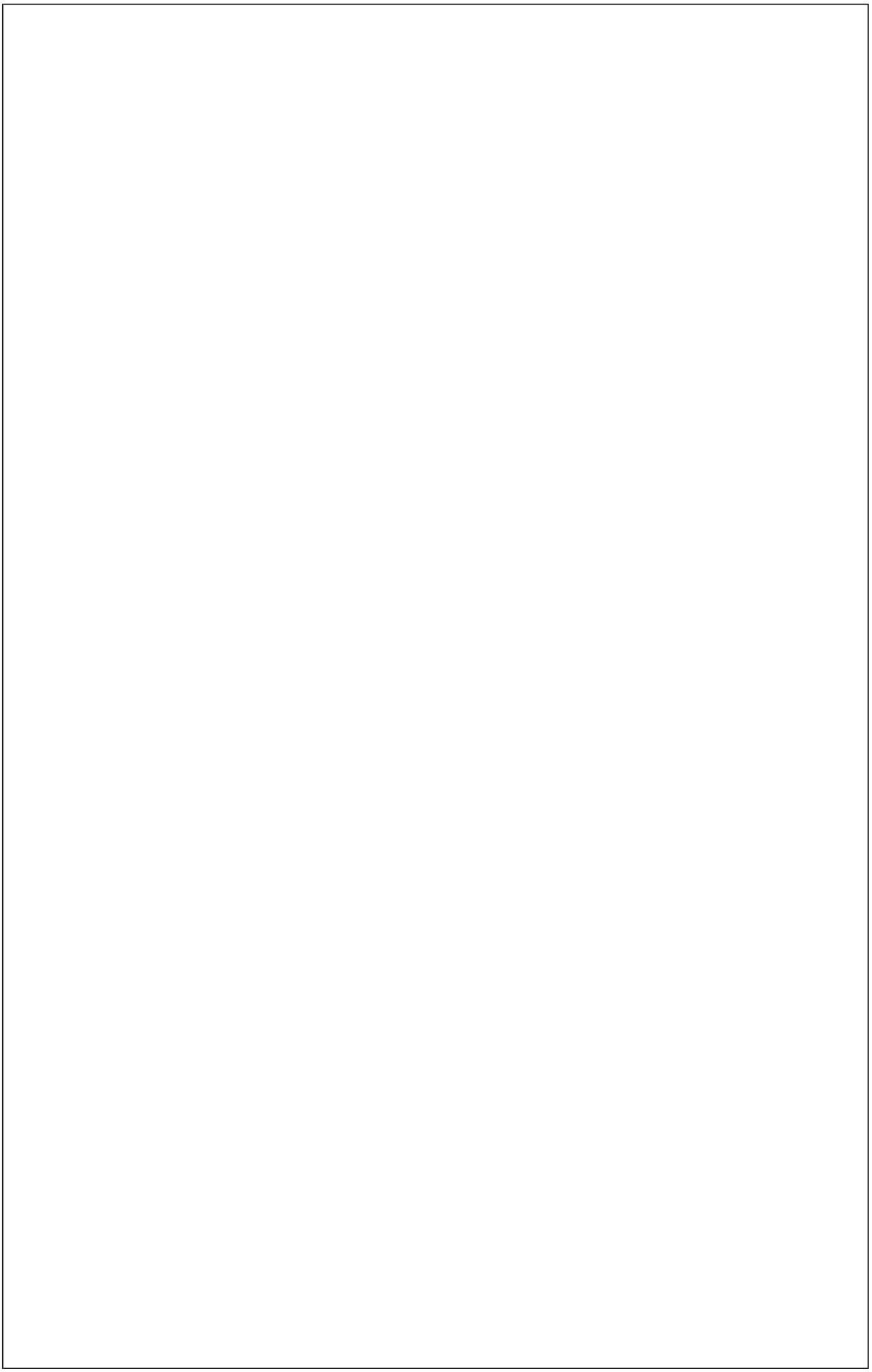
1. Tulis dan jalankan **Contoh 8.1** pada modul ini dan pahami tiap barisnya!
2. Ubahlah kode program behavior (**Contoh 8.2** atau **Contoh 8.3**) menjadi OneShootBehaviour, TickerBehaviour, dan CyclicBehaviour sehingga diketahui perbedaan dari ketiga behaviour tersebut!

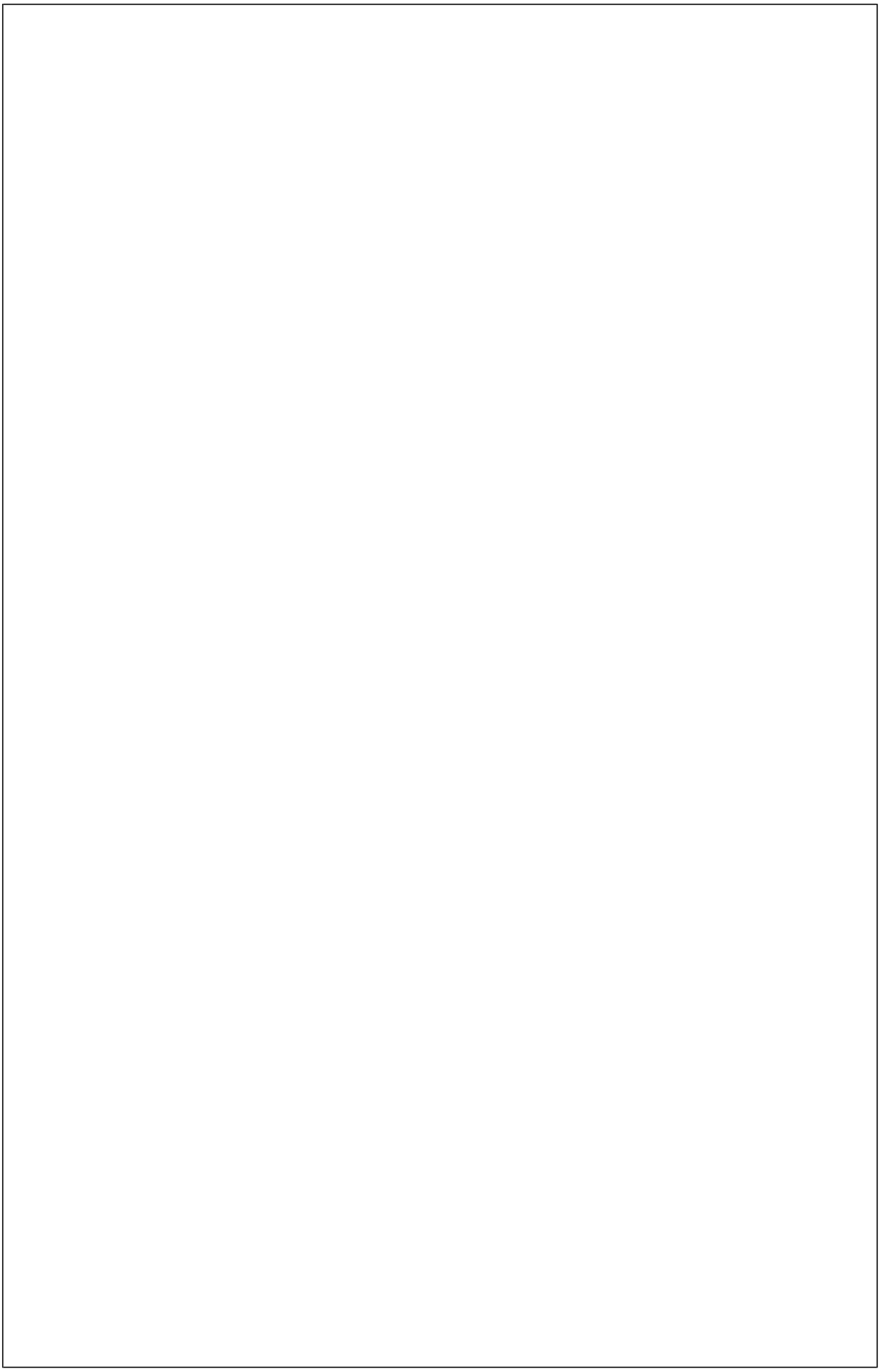
3. Kerjakan tiap langkah pada sub pembahasan **8.1.5 Menjalankan Contoh "Book Trading"** pada modul ini. Pahami bagaimana contoh implementasi software agent menggunakan JADE.

8.3 Tugas

1. Jelaskan apa yang dimaksud dengan software agent!
2. Sebutkan contoh software agent dan jelaskan bagaimana jika diimplementasikan menggunakan JADE!
3. Pelajari lagi listing program dari Book Trading. Tuliskan algoritma/flowchart dari program tersebut!
4. Buatlah program timer menggunakan salah satu sifat dari agent menggunakan JADE!

Jawaban Tugas Pengembangan





MODUL 9
JADE PROGRAMMING (LANJUTAN)

9.1 Pembahasan

Pemrograman Agent menggunakan JADE pada modul 9 ini membahas tentang komunikasi agent serta layanan *yellow pages*.

9.1.1 Komunikasi Agent

Komunikasi antar agent merupakan fitur penting dalam JADE. Untuk komunikasi antar *platform*, pesan akan diubah dari representasi java internal JADE, menjadi sintak-sintak, kode-kode, dan protokol-protokol *transport* yang mengikuti aturan FIPA (*Foundation for Intellegent Physical Agent*). FIPA sendiri adalah sebuah lembaga internasional yang mengembangkan standar-standart terkait dengan teknologi agen. Komunikasi antar agent dapat tercapai ketika terdapat pengiriman dan penerimaan pesan antara dua atau lebih agent.

Pertukaran pesan oleh agent JADE menggunakan format ACL (*Agent Communication Language*) yang didefinisikan oleh FIPA. Format ini mencangkup beberapa bidang, khususnya:

- Pengirim pesan (***sender***)
- Daftar ***receiver***
- Maksud dari komunikasi (***performative***), mengindikasikan tujuan dari pengirim mengirimkan pesan. Permormatif dapat berupa REQUEST jika pengirim mengiginkan penerima melakukan suatu tindakan, INFORM jika pengirim menginginkan penerima mengetahui suatu fakta, CFP (call for proposal), PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL jika pengirin dan penerima terlibat dalam negoisasi, dan masih banyak lagi jenis permormatif lain.
- ***Content***, yaitu isi dari pesan
- Content ***language***, yaitu syntax yang digunakan untuk mengungkapkan content
- ***Ontology***, yaitu kosakata symbol yang digunakan dalam content dan makna yang tercangkup di dalamnya.
- Selain itu, ada beberapa cakupan lain yang diguankan untuk mengontrol percakapan dan menentukan timeout untuk menerima balasan, seperti ***conversation-id***, ***reply-with***, ***in-reply-to***, ***reply-by***

Pesan di dalam JADE diimplementasikan sebagai sebuah objek dari kelas `jade.lang.acl.ACLMessage` yang menyediakan method `set` dan `get` untuk menangani setiap bidang pesan. Code berikut ini menunjukkan pesan yang menginformasikan kepada sebuah agent bernama *peter* bahwa *today it's raining* :

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID("Peter", AID.ISLOCALNAME));
msg.setLanguage("English");
msg.setOntology("Weather-forecast-ontology");
msg.setContent("Today it's raining");
send(msg);
```

Pengiriman Pesan

Tabel 9.1 berikut ini menunjukkan method yang umum digunakan untuk pengiriman pesan.

Tabel 9.1 Method pada JADE untuk pengiriman pesan

Method / Procedure	**Deskripsi**

`ACLMessage msg = new ACLMessage(ACLMessage.INFORM)`
`thows`
`jade.lang.acl.ACLMessage`
	Memanggil Class Object ACLMessage untuk tempat dari pesan yang akan dikirim. Nama dari Class Object ACLMessage yaitu "msg"

`Public void setContent()`
`throws`
`jade.lang.acl.ACLMessage`
	Memanggil Methode dari kelas ACLMessage untuk pengisian pesan

`Public void addReceiver()`
`throws`
`jade.lang.acl.ACLMessage`
	Methode yang digunakan untuk menambah nama agent sebagai penerima pesan

`Public void send()`
`throws`
`jade.lang.acl.ACLMessage`
	Mengirimkan pesan sesuai dengan agent tujuan

Penerimaan Pesan

Tabel 9.2 berikut ini menunjukkan method dan konstraktor yang umum digunakan pada penerima pesan.

Tabel 9.2 Method pada JADE untuk menerima pesan

Method / Procedure	Deskripsi

`Public void receive()`
`throws`
`jade.lang.acl.ACLMessage`
	Methode untuk penerima kiriman

`Public void getContent()`
`throws`
`jade.lang.acl.ACLMessage`
	Methode untuk membaca isi dari pesan yang didapat

`Public void createReply()`
`throws`
`jade.lang.acl.ACLMessage`
	Methode untuk membalas kiriman pesan secara otomatis.

`ACLMessage msg =new ACLMessage()`
`throws`
`jade.lang.acl.ACLMessage`
	Memanggil Class Object ACLMessage untuk memanggil methode yang digunakan untuk penerimaan dan membalas pesan otomatis

Seleksi Pesan menggunakan Message Template

Message Template digunakan untuk menjamin bahwa pesan yang diterima oleh suatu agent, adalah yang tepat dan di proses dengan behaviour yang tepat pula. Oleh karena itu dengan adanya *Message Template* pada setiap behaviour yang menerima pesan, maka penyaringan pesan yang masuk akan lebih mudah. **Tabel 9.3** berikut ini menunjukkan method dan konstraktor dasar penggunaan Message Template

Tabel 9.3 Method dasar penggunaan Message Template

Method / Procedure	Deskripsi

`MessageTemplate nama = MessageTemplate.and();`
`thows`
`jade.lang.acl.*;`
	Memanggil Class Object MessageTemplate untuk tempat dari template. Nama dari Class Object MessageTemplate yaitu “nama”

`Public voidMatchSender(new AID("broker", AID.ISLOCALNAME))`
`throws`
`jade.lang.acl.*;`
	Memanggil Method dari kelas MessageTemplate untuk penyeleksian agen pengirim pesan. nama agen yang diperbolehkan : broker

`Public void MatchPerformative(ACLMessage.INFORM)`
`throws`
`jade.lang.acl.*;`
	Methode yang digunakan untuk pengecekkan performativenya

Sebagai contoh, berikut ini ditunjukkan pada behavior agent yang bersifat cyclic dilakukan penyaringan pesan dengan tipe pervormatif CFP sehingga aksi dilakukan ketika agent menerima pesan CFP.

```
public void action() {
 MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
 ACLMessage msg = myAgent.receive(mt);
 if (msg != null) {
       // CFP Message received. Process it
       ...
 }
 else {
       block();
 }
}
```

Contoh 9.1. Komunikasi Agent

Kelas Pengirim

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;

public class sender extends Agent {

    @Override
    protected void setup() {
        addBehaviour(new TickerBehaviour(this, 1000) {

            @Override
            protected void onTick() {
                System.out.println(myAgent.getLocalName() + " mengirim pesan");
                ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                msg.addReceiver(new AID("AgentPenerima", AID.ISLOCALNAME));
                msg.setContent("Ini pesan dari " + myAgent.getLocalName());
                send(msg);
            }
        });
    }

}
```

Kelas Penerima

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class receiver extends Agent {
    protected void setup() {
        addBehaviour(new CyclicBehaviour(this) {
            public void action() {
                //seleksi pesan berdasarkan performative
                MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.INFORM);
                //seleksi pesan berdasarkan nama agent pengirim dan permormative
                MessageTemplate mt2 = MessageTemplate.and(MessageTemplate.MatchSender(
                        new AID("pengirim", AID.ISLOCALNAME)),
                        MessageTemplate.MatchPerformative(ACLMessage.INFORM));

                ACLMessage msg = myAgent.receive(mt);
                String content;
                if (msg != null) {
                    content = msg.getContent();
                    System.out.println("Pesan yang diterima: " + content);
                } else {
                    block();
                }
            }
        });
    }
}
```

Konfigurasi untuk Menjalankan Agent

Untuk menjalankan kedua kelas tersebut secara bersamaan maka lakukan pengaturan konfigurasi dengan menuliskan **jade.Boot** pada main class, sedangkan pada argurment tuliskan:

–gui namaAgent:packace.namaKelas

misal:

-gui AgentPenerima:JADE.receiver;AgentPengirim:JADE.sender

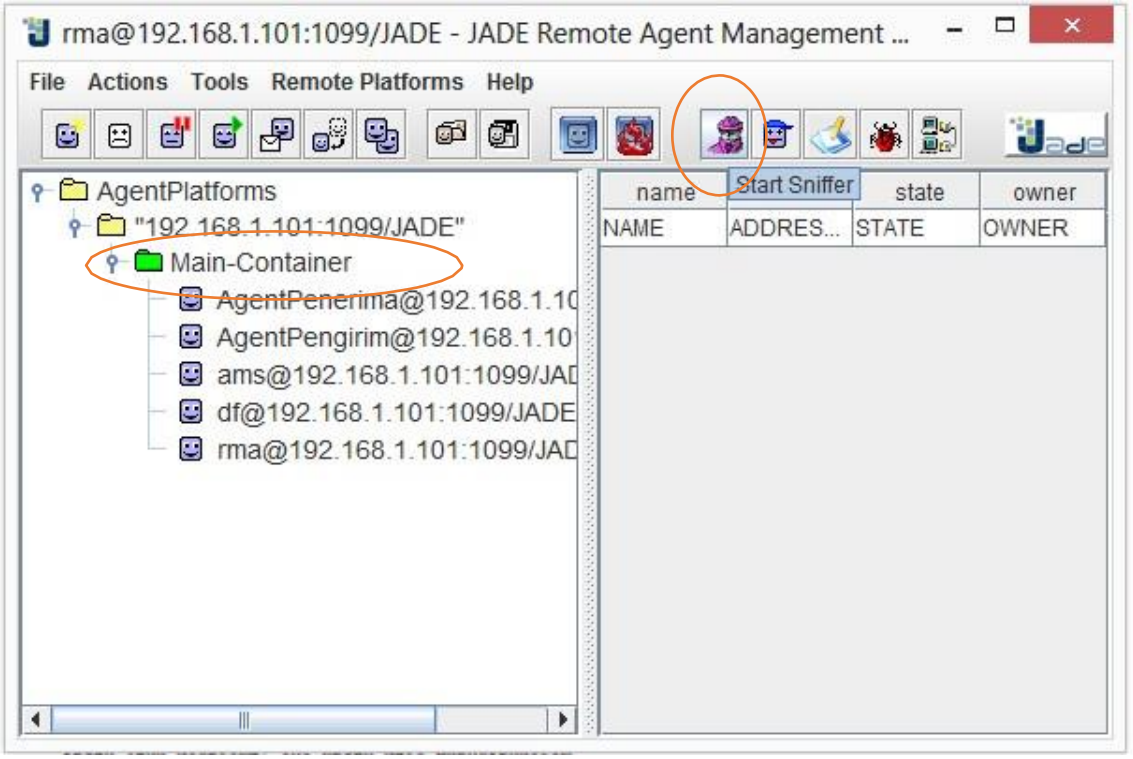
Begitu dijalankan akan tampil output:

```
----------------------------------------
Apr 18, 2016 7:16:40 AM jade.imtp.leap.LEAPIMTPManager initialize
INFO: Listening for intra-platform commands on address:
- jicp://192.168.1.101:1099

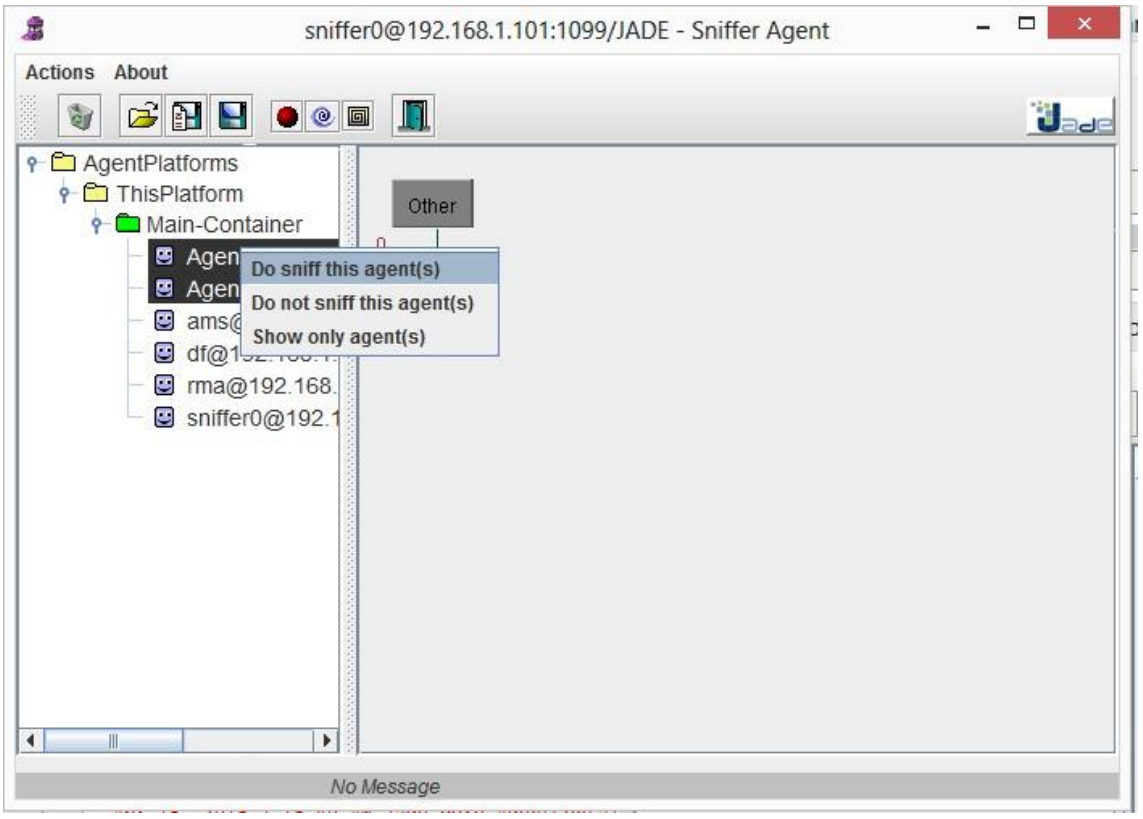
Apr 18, 2016 7:16:40 AM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
Apr 18, 2016 7:16:40 AM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
Apr 18, 2016 7:16:40 AM jade.core.BaseService init
INFO: Service jade.core.resource.ResourceManagement initialized
Apr 18, 2016 7:16:40 AM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
Apr 18, 2016 7:16:40 AM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
Apr 18, 2016 7:16:40 AM jade.mtp.http.HTTPServer <init>
INFO: HTTP-MTP Using XML parser com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser
Apr 18, 2016 7:16:40 AM jade.core.messaging.MessagingService boot
INFO: MTP addresses:
http://Khiznia:7778/acc
Apr 18, 2016 7:16:40 AM jade.core.AgentContainerImpl joinPlatform
INFO: ----------------------------------------
Agent container Main-Container@192.168.1.101 is ready.
--------------------------------------------
AgentPengirim mengirim pesan
Pesan yang diterima: Ini pesan dari AgentPengirim
AgentPengirim mengirim pesan
Pesan yang diterima: Ini pesan dari AgentPengirim
AgentPengirim mengirim pesan
Pesan yang diterima: Ini pesan dari AgentPengirim
AgentPengirim mengirim pesan
Pesan yang diterima: Ini pesan dari AgentPengirim
AgentPengirim mengirim pesan
```

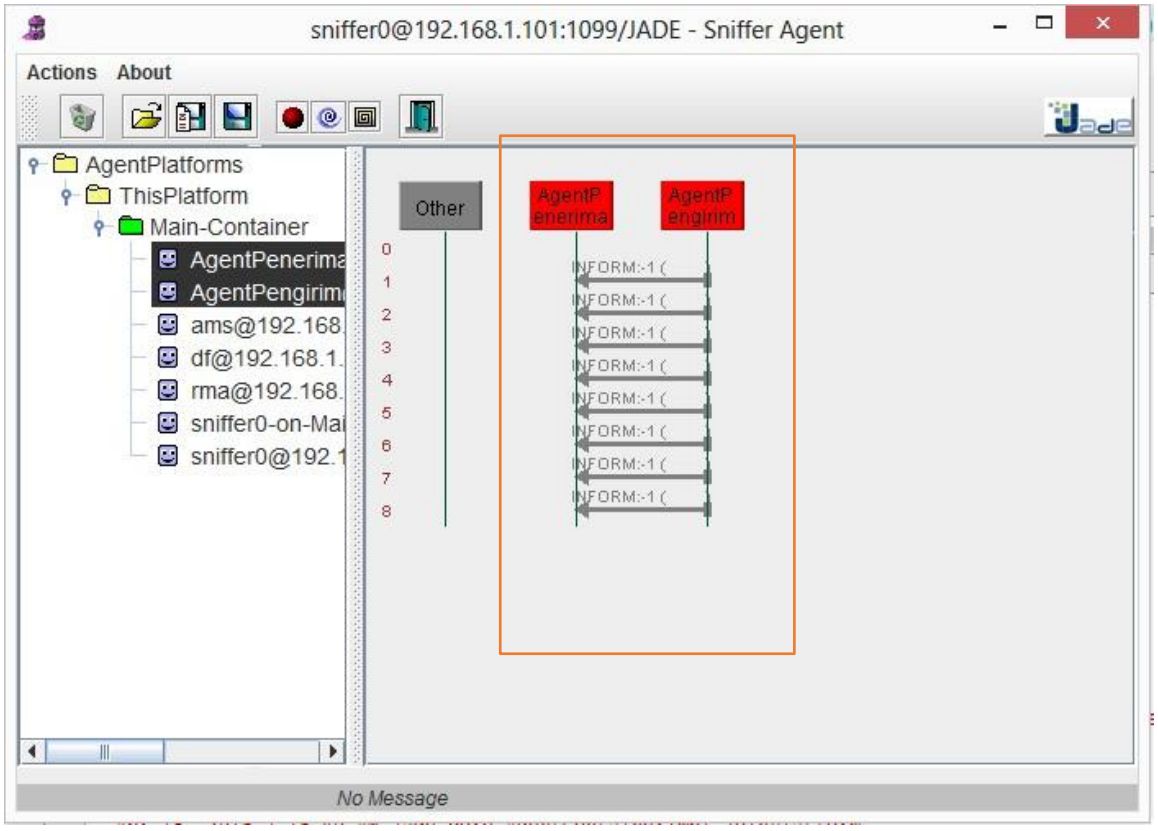
Sniffing

Untuk melihat komunikasi antar agent, dapat menggunakan *sniff* pada RMA. Pastikan kursor aktiv pada main-container kemudian klik icon "*sniffer*" untuk melakukan *sniffing*.



Klik kanan pada agent kemudian pilih “Do sniff this agent(s)” sehingga tampak arah komunikais dan jenis pesan yang dikirim.

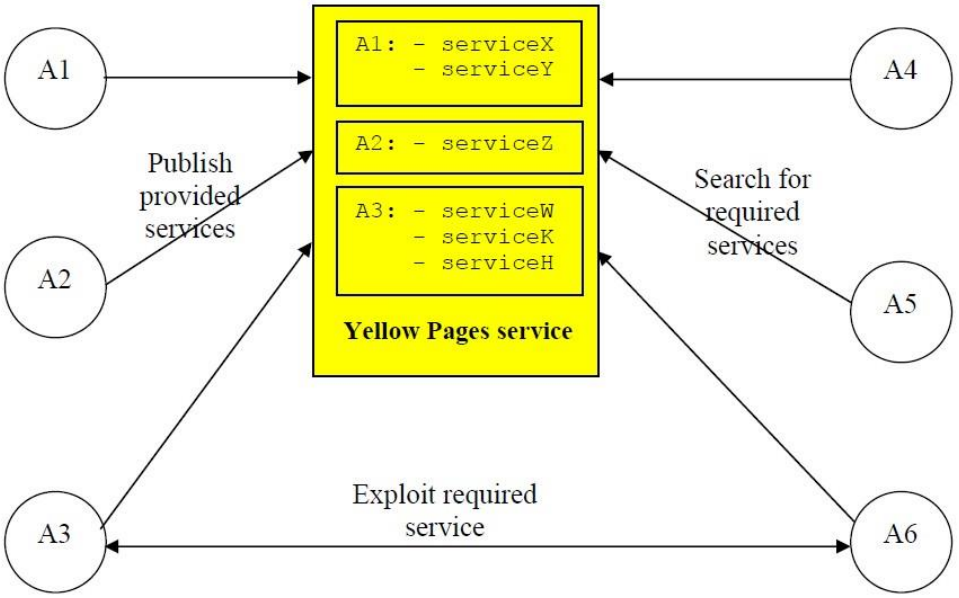




9.1.2 Yellow Pages

Layanan *yellow pages* memungkinkan agen untuk mempublikasikan deskripsi dari satu atau lebih layanan yang mereka berikan agar agen lain dapat dengan mudah menemukan mereka. Layanan yellow pages di JADE, sesuai dengan spesifikasi Agen Manajemen FIPA, disediakan oleh agen khusus yang disebut DF (*Directory Fasilitator*).

Setiap agen bisa mendaftar (*publish*) layanan dan mencari (*discover*) jasa. Pendaftaran, *deregistrations*, modifikasi dan pencarian dapat dilakukan setiap saat selama agen masih hidup. Hal ini digambarkan dalam gambar berikut:



Gambar 9.1 Layanan Yellow Pages

Selama DF adalah agen, hal tersebut memungkinkan untuk berinteraksi dengan agen lainnya dengan bertukar *ACLMessage* menggunakan bahasa konten yang seperti yang didefinisikan dalam spesifikasi FIPA.

Contoh 9.2. Yellow Pages

Pada contoh ini terdapat tiga kelas yang ditulis terpisah, yaitu: (1) kelas “kirim” sebagai pengirim informasi, (2) kelas “broker” sebagai perantara yang mengirimkan informasi dari pengirim ke beberapa penerima serta sebagai pencari agent penerima yang masih hidup, dan (3) kelas “terima” sebagai penerima pesan.

Kelas Kirim

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;

public class kirim extends Agent{
protected void setup(){
    addBehaviour(new CyclicBehaviour(this) {

        @Override
        public void action() {
            ACLMessage msg= new ACLMessage(ACLMessage.SUBSCRIBE);
            msg.setConversationId("kirim");
            msg.setContent("dari "+myAgent.getLocalName()+" apa kaabr? \n");
            msg.addReceiver(new AID("broker", AID.ISLOCALNAME));
            myAgent.send(msg);
            block(2000);
        }
    });
}
}
```

Kelas Broker

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;

public class broker extends Agent {

    Vector aTerimaBaru = new Vector();

    @Override
    protected void setup() {
        addBehaviour(new behav(this, aTerimaBaru));
        addBehaviour(new cari_df(this, 10000, aTerimaBaru));
    }

    class behav extends CyclicBehaviour {...40 lines }

    class cari_df extends TickerBehaviour {...28 lines }
}
```

```
class behav extends CyclicBehaviour {
    MessageTemplate mt_Kirim;
    boolean StaKirim = false, StaTerima = false;
    Vector vTerima;
    int iTerima = 0;
    public behav(broker aThis, Vector aTerimaBaru) {
        vTerima = aTerimaBaru;
        mt_Kirim = MessageTemplate.MatchConversationId("kirim");
    }

    public void action() {
        //terima pesan dari terima
        ACLMessage msgKrmPesan = myAgent.receive(mt_Kirim);
        // terima pesan dari kirim
        if (msgKrmPesan != null) {
            if (vTerima.size() > 0) {
                ACLMessage Krmbalas = new ACLMessage(ACLMessage.PROPOSE);
                Krmbalas.setContent(msgKrmPesan.getContent());
                Krmbalas.addReceiver((AID) vTerima.elementAt(iTerima++));
                myAgent.send(Krmbalas);
                if (iTerima >= vTerima.size()) {
                    iTerima = 0;
                }
                StaKirim = true;
                System.out.print("size ok\n");
            }
            System.out.print("msg ok\n");
        } else {
            StaKirim = false;
        }
        if (!(StaKirim || StaTerima)) {
            block(50);
        }
    }
}
```

```
class cari_df extends TickerBehaviour {

    String[] aTerimaBaru;
    DFAgentDescription tm = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    Vector vTerima;

    public cari_df(Agent aThis, int i, Vector aTerimaBaru) {
        super(aThis, i);
        sd.setType("terima");
        tm.addServices(sd);
        vTerima = aTerimaBaru;

    }
```

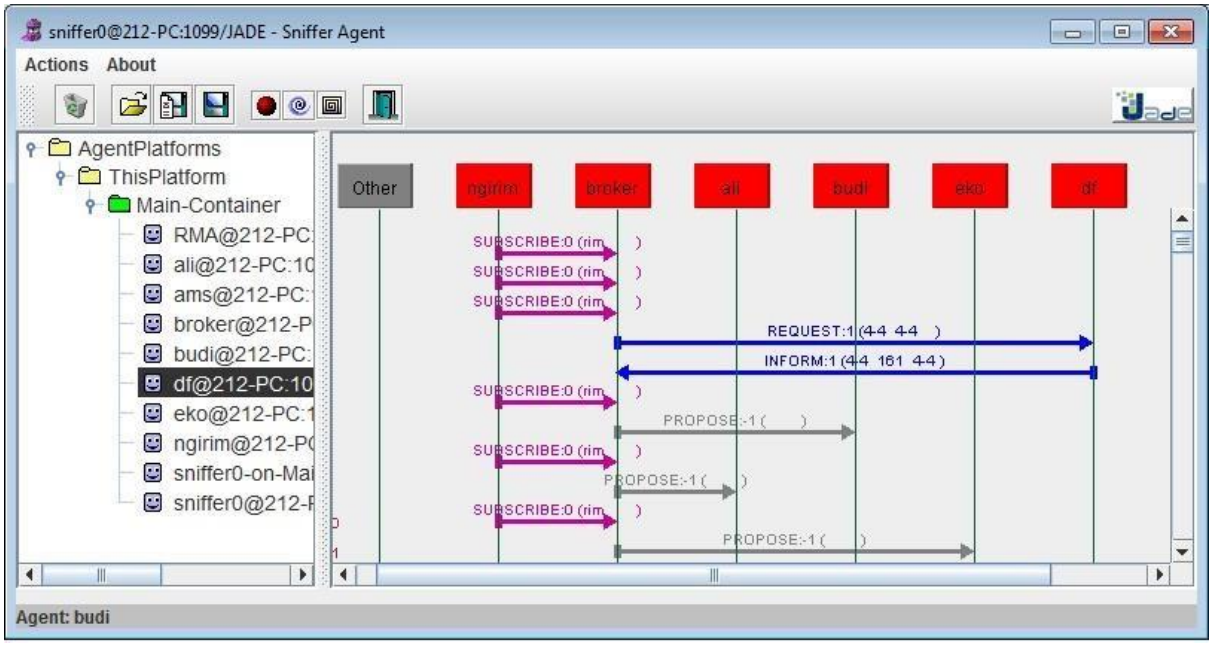
```
        @Override
        protected void onTick() {
            try {
                DFAgentDescription[] dfPenerima = DFService.search(myAgent, tm);
                vTerima.clear();
                for (int i = 0; i < dfPenerima.length; i++) {
                    vTerima.addElement(dfPenerima[i].getName());
                }
            } catch (FIPAException ex) {
                Logger.getLogger(broker.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
```

Kelas Terima

```
public class terima extends Agent {
    protected void setup() {
        //registrasi agent
        DFAgentDescription dfad = new DFAgentDescription();
        dfad.setName(getAID());
        ServiceDescription sd = new ServiceDescription();
        sd.setType("terima");
        sd.setName(getLocalName());
        dfad.addServices(sd);
        try {
            DFService.register(this, dfad);
        } catch (FIPAException ex) {
            Logger.getLogger(terima.class.getName()).log(Level.SEVERE, null, ex);
        }

        addBehaviour(new CyclicBehaviour(this) {
            public void action() {
                ACLMessage pesan = myAgent.receive();
                if (pesan != null) {
                    String msg = pesan.getContent();
                    System.out.print("terima : " + myAgent.getLocalName() +
                            " : " + msg + "\n");
                } else {
                    block(200);
                }
            }
        });
    }
}
```

Hasil dari kode program diatas jika dijalankan dengan isi dari argumentnya: '**-gui ngirim:JADE.kirim;broker:JADE.broker;budi:JADE.terima;eko:JADE.terima;ali:JADE.terima**', sebagai berikut :



Berdasarkan hasil tersebut dapat di jabarkan, bahwa broker me-*request* ke DF (*Directory Fasilitator*) tentang pencarian agent penerima dengan type "terima". Kemudian DF mengirim kembali kepada broker dan broker melakukan pekerjaannya yaitu sebagai perantara informasi dari 1 pengirim ke 3 penerima sekaligus.

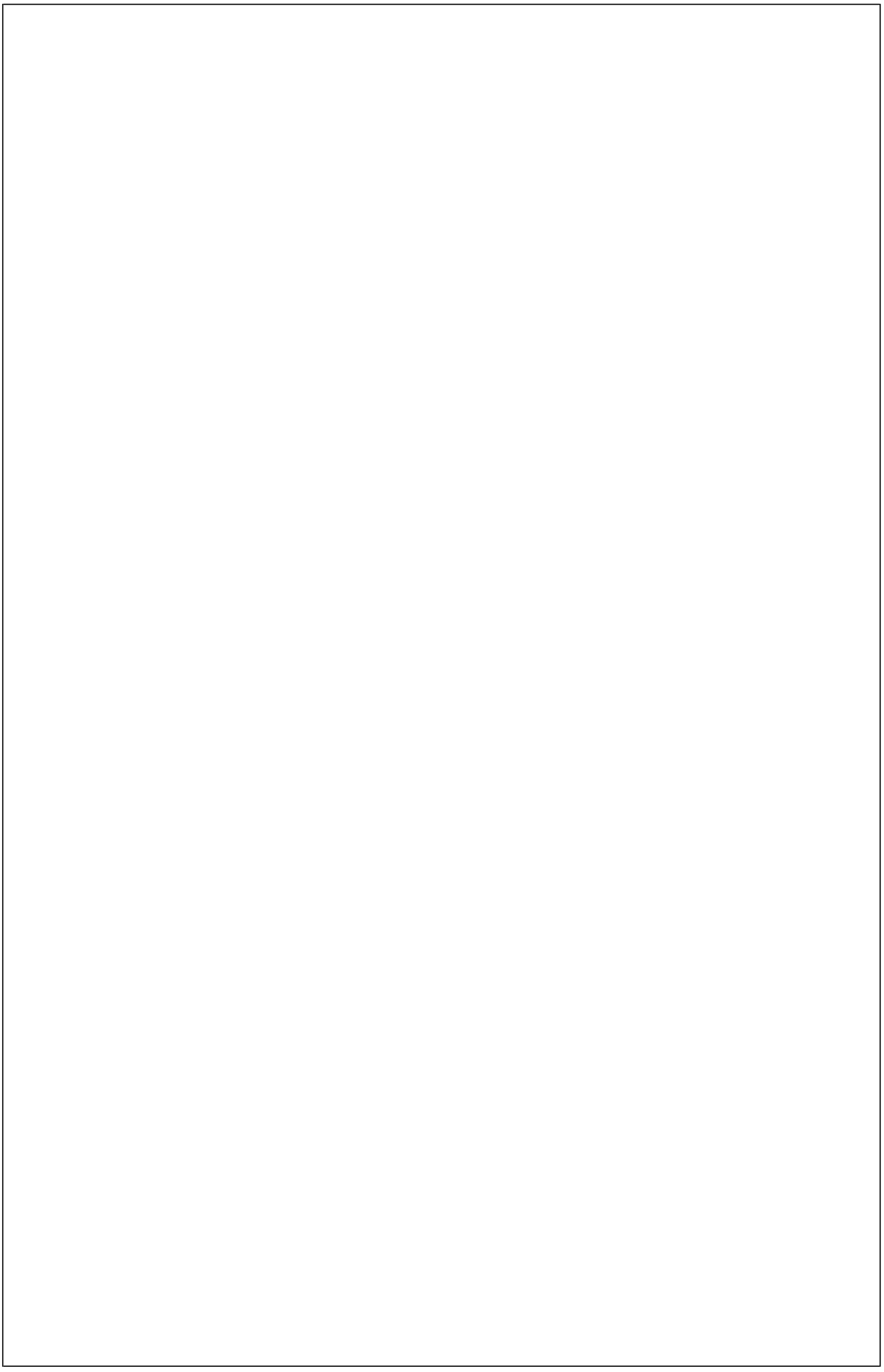
9.2 Praktikum

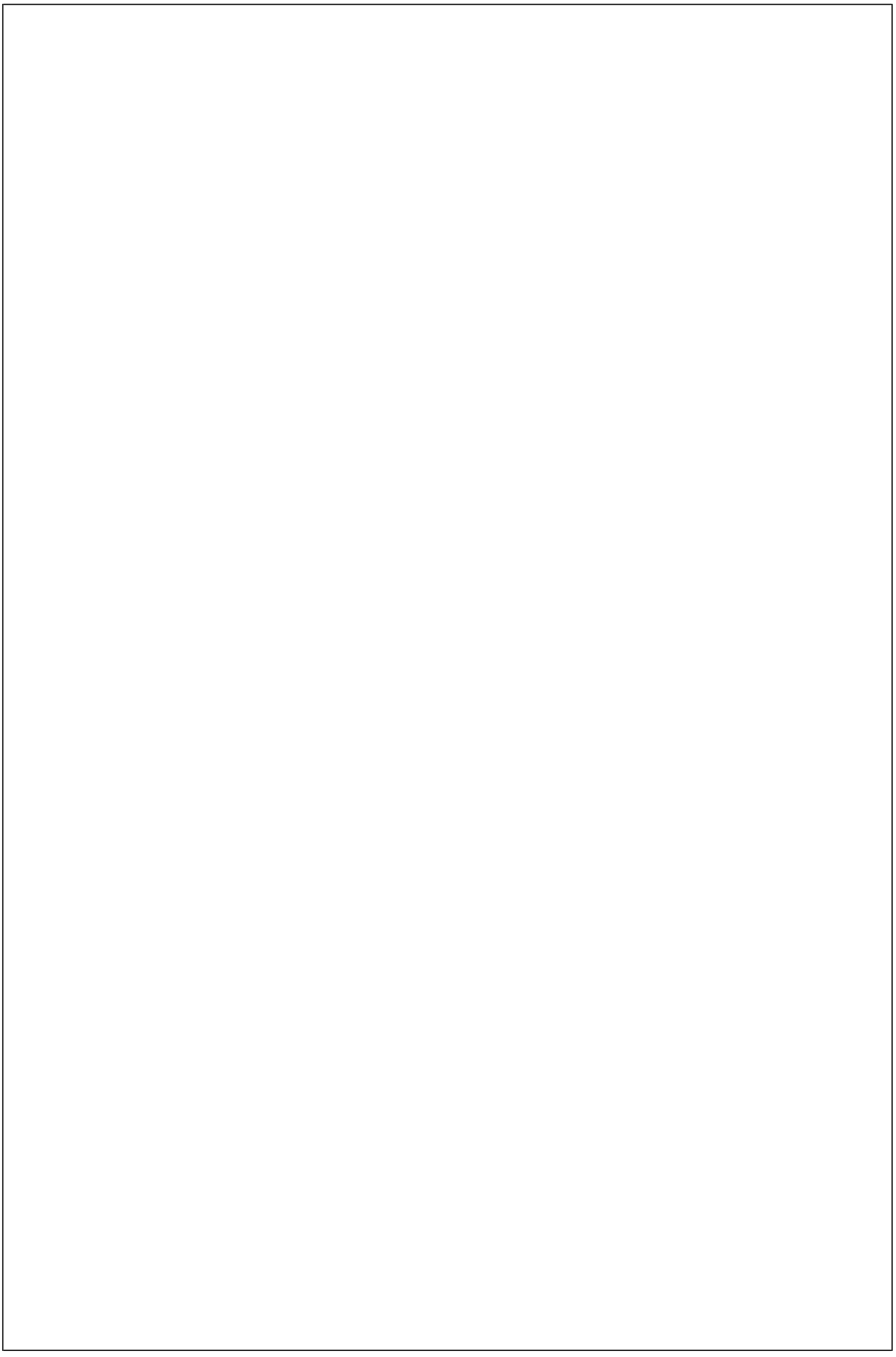
1. Tulis dan jalankan **Contoh 9.1** sesuai langkah pada modul ini dan pahami tiap barisnya!
2. Tulis dan jalankan **Contoh 9.2** sesuai langkah pada modul ini dan pahami tiap barisnya!

9.3 Tugas

1. Pelajari contoh book trading untuk menjawab pertanyaan berikut:
   a. Jelaskan baris program yang menunjukkan komunikasi agent. Jelaskan ACLMessage dan MessageTemplate yang digunakan.
   b. Jelaskan alur kumunikasi agent dengan menjalankan Sniff
   c. Jelaskan layanan DF pada book trading dan jelaskan baris programnya
2. Buatlah sebuah aplikasi travel berbasis agent!
3. Bagaimana konfigurasi project agar aplikasi yang anda buat pada tugas 2 dapat dijalankan di beberapa device? Simulasikan jalannya aplikasi menggunakan 4 device (minimal)

Jawaban Tugas Pengembangan



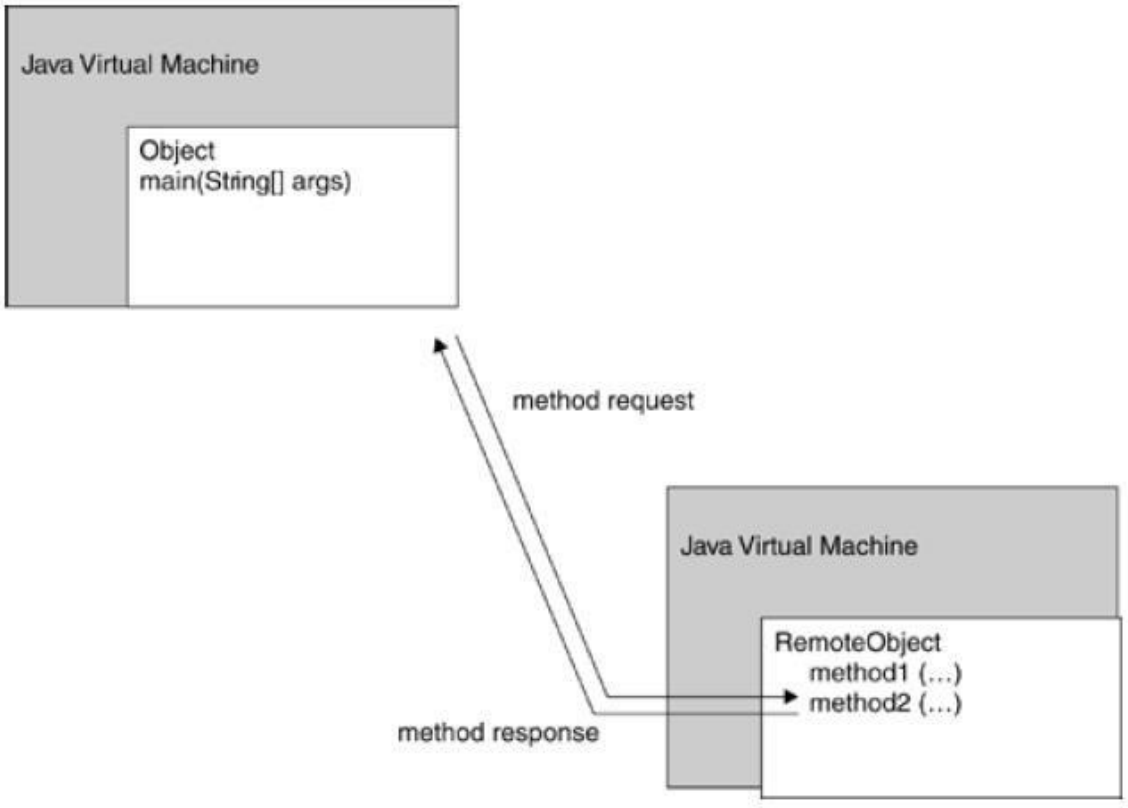


MODUL 10

REMOTE METHOD INVOCATION

10.1 Pembahasan

RMI (*Remote Method Invocation*) adalah sebuah API yang menyediakan mekanisme untuk membuat aplikasi terdistribusi di Java. RMI memungkinkan sebuah object untuk memanggil metode pada object yang berjalan di JVM lain.



Gambar 10.1 Permohonan akses method pada *remote object* dan eksekusi pada *remote machine*

Sistem yang menggunakan RMI untuk komunikasi biasanya dibagi menjadi dua kategori: klien dan server. Sebuah server menyediakan layanan RMI, dan klien memanggil metode object layanan ini.

Server RMI harus mendaftar pada layanan pencarian (lookup service) sehingga klien dapat menemukan mereka. Dalam platform Java terdapat sebuah aplikasi yang disebut *rmiregistry*, yang berjalan sebagai proses terpisah dan memungkinkan aplikasi untuk mendaftar layanan RMI atau mendapatkan referensi ke nama layanan tertentu. Setelah server telah terdaftar, maka ia akan menunggu permintaan RMI masuk dari klien.

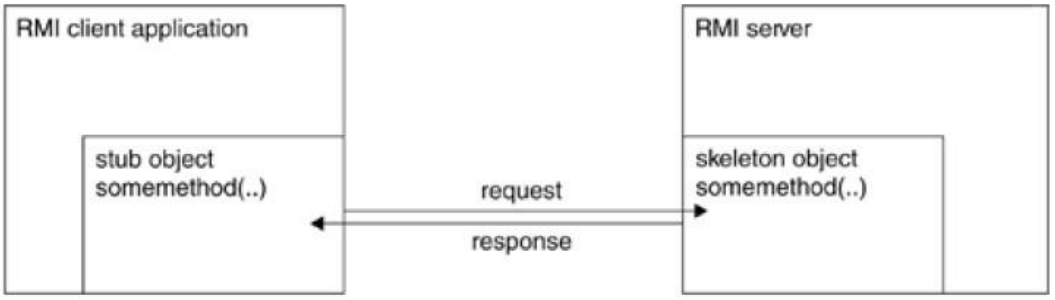
Client RMI akan mengirimkan pesan RMI untuk meminta object method dari jarak jauh. Untuk melakukan permintaan, client harus memiliki referensi object remote yang bisa diperoleh dengan mencari layanan pada *RMI registry*. Aplikasi client meminta nama layanan tertentu dan menerima URL. Format berikut digunakan untuk merepresentasikan sebuah referensi *remote object*.

rmi://hostname:port/servicename

dimana hostname merepresentasikan nama server (atau ip address), port lokasi layanan pada mesin dan servicename sebuah string yang mendiskripsikan layanan.

RMI menyediakan komunikasi jarak jauh antara aplikasi menggunakan dua object *stub* dan *skeleton*. RMI menggunakan object stub dan object skeleton untuk komunikasi dengan object remote. Sebuah remote object adalah obyek yang metodenya dapat dipanggil dari JVM lain.

Stub adalah obyek yang bertindak sebagai gateway untuk sisi klien. Semua permintaan keluar disalurkan melalui object ini. Sedangkan *Skeleton* adalah obyek, bertindak sebagai gateway untuk object sisi server. Semua permintaan masuk akan dialihkan melalui object ini.



Gambar 10.2 Stub pada client RMI memanggil skeleton pada server RMI

Implementasi Layanan RMI

Suatu sistem yang menggunakan RMI harus menggunakan sebuah layanan interface. Layanan interface mendefinisikan method object yang dapat dikontrol dari jarak jauh, parameter tertentu, mengembalikan (*return*) tipe, dan *exception*. Object stub dan skeleton, begitu juga dengan layanan RMI diimplementasikan pada interface ini.

Berikut ini implementasi layanan RMI dengan Java, yang terdiiri dari: membuat object RMI, membuat server RMI, dan membuat client RMI.

1. Membuat Object RMI

a. Membuat class yang implements Interface

Yang dimaksud dengan implements interface yaitu object yang dibuat dari suatu class harus merupakan turunan dari sebuah interface. Misalnya, terdapat class "Data", agar dapat diakses lewat RMI maka class Data tersebut harus merupakan turunan dari sebuah Interface, misalkan saja interface DataInterface.

```
package RMI;
public interface DataInterface{

}
```

```
package RMI;
public class Data implements DataInterface{

}
```

b. Menambahkan extends java.rmi.Remote pada Interface

Selain class harus turunan interface, interface tersebut harus extend `java.rmi.Remote`, sehingga baris program menjadi sebagai berikut:

```
package RMI;
import java.rmi.Remote;

public interface DataInterface extends Remote{

}
```

c. Menambahkan extend java.rmi.server.UnicastRemoteObject pada class

Selain interface yang harus extend `java.rmi.Remote`, class yang mengimplementasikan Interface tersebut harus extend `java.rmi.server.UnicastRemoteObject`, sehingga baris program tampak sebagaimana berikut:

```
package RMI;
import java.rmi.server.UnicastRemoteObject;

public class Data extends UnicastRemoteObject implements DataInterface{

}
```

d. Menambahkan konstruktor yang thow java.rmi.RemoteException pada Class

class object yang akan diakses lewat RMI harus memiliki konstruktor yang throws `java.rmi.RemoteException`, sehingga baris program tampak sebagaimana berikut:

```
package RMI;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Data extends UnicastRemoteObject implements DataInterface{
      public Data() throws RemoteException{
      }
}
```

e. *Menambahkan method Interface yang throw java.rmi.RemoteException*

Method pada Interface memiliki object yang akan diakses lewat RMI. Seluruh method yang dimiliki Interface tersebut harus throw `java.rmi.RemoteException`.

Dengan demikian maka maka baris program lengkap untuk interface dan class pada contoh langkah ini adalah sebagai berikut:

```
package RMI;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface DataInterface extends Remote {

    public void FirstMethod() throws RemoteException;

    public void SecondMethod() throws RemoteException;
}
```

```
package RMI;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Data extends UnicastRemoteObject implements DataInterface {

    public Data() throws RemoteException {
    }

    @Override
    public void FirstMethod() throws RemoteException {
        System.out.println("Running First Method");
    }

    @Override
    public void SecondMethod() throws RemoteException {
        System.out.println("Running Second Method");
    }
}
```

2. Membuat Server RMI

Server digunakan sebagai tempat untuk melakukan sharing object, sehingga client dapat mengakses object RMI melalui servre tersebut. Dalam RMI, server direpresentasikan sebagai interface `java.rmi.registry.Registry`. Karena merupakan interface, maka untuk membuat object `java.rmi.registry.registry` dapat dilakukan dengan instansiasi "new". Selain itu, dapat pula menggunakan method `createRegistry(int port)` yang berapa pada class `java.rmi.registry.LocateRegistry`.

```
Registry registry = LocateRegistry.createRegistry(1099);
```

Baris kode tersebut menyatakan membuat server (registry) RMI pada port 1099, ini adalah port yang biasa digunakan oleh RMI.

Setelah server terbentuk, perlu dilakukan penyimpanan object dari class yang telah dibuat ke server agar bisa diakses oleh client. Untuk menyimpan object ke server dilakukan dengan perintah:

```
Data data= new Data();
registry.bind("data", data);
```

Method `bind()` memiliki dua parameter, pertama adalah String dan yang kedua adalah Object. String tersebut adalah nama object yang disimpan, nama object tersebut bebas sesuai yang diinginkan namun nantinya harus sama dengan string nama object pada client agar bisa diakses. Sedangkan parameter kedua, yaitu object yang dishare. Contoh diatas menunjukkan proses penyimpanan data object class Data dengan nama "data".

Perlu diperhatikan pula pada penggunakan method `bind()` adalah jika dilakukan penyimpanan object dengan nama yang telah ada di server, maka akan terjadi error. Misal dilakukan penyimpanan terhadap object "data" kemudian menyimpan object baru dengan maksud memperbarui object sebelumnya dengan nama yang sama, maka ini akan menyebabkan error. Oleh karena itu, untuk penyimpanan object yang sewaktu-waktu bisa berubah maka disarankan menggunakan method `rebind()`.

Cara kerja method `rebind()` adalah pertama server akan mendeteksi apakah object dengan nama yang sama telah ada, jika belum maka object yang baru akan disimpan, namun jika object dengan nama yang sama telah ada maka object yang lama akan dihapus dan digantikan oleh object yang baru.

```
package RMI;

import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Server {

    public static void main(String[] args) throws RemoteException,
            NotBoundException {
        Registry registry = LocateRegistry.createRegistry(1099);

        Data data = new Data();
        registry.rebind("data", data);

        System.out.println("Server is running");
    }
}
```

3. Membuat Client RMI

Client sebagai pihak yang mengakses object yang ada pada server. Untuk membuat client dilakukan dengan menggunakan class java.rmi.Naming. Naming merupakan class utilities, artinya tidak bisa membuat object Naming dengan instansiasi "new", sehingga cukup mengetikkan java.rmi.Naming untuk membuat client.

Selanjutnya untuk mengakses Object pada server maka harus membuat object interfacenya, bukan classnya. Misal pada contoh pengenalan RMI ini dengan membuat object interface DataInterface yang merupakan interface yang diimplementasikan oleh class Data. Untuk membuat object interface tersebut dilakukan dengan:

```
DataInterface data = (DataInterface)
                    Naming.lookup("rmi://localhost:1099/data");
```

Untuk mengakses data tersebut dapat menggunakan method lookup yang dimiliki oleh class Naming dengan parameter url yang sama dengan yang didefinisikan pada server. Berikut ini format url pengaksesan object di server:

```
"rmi://"+host+":"+port+"/"+nama_object
```

Dimana host adalah tempat server, baik berupa nama host maupun IP Address, untuk port adalah port yang digunakan oleh server dan untuk nama_object adalah nama object yang ada di server.

```java
package RMI;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

public class Client {

    public static void main(String[] args) throws NotBoundException,
            MalformedURLException, RemoteException {
        DataInterface data = (DataInterface) Naming.lookup("rmi://localhost:1099/data");
        System.out.println("Client is connect to Server");

        data.FirstMethod();
        data.SecondMethod();

        System.out.println("finish");
    }
}
```

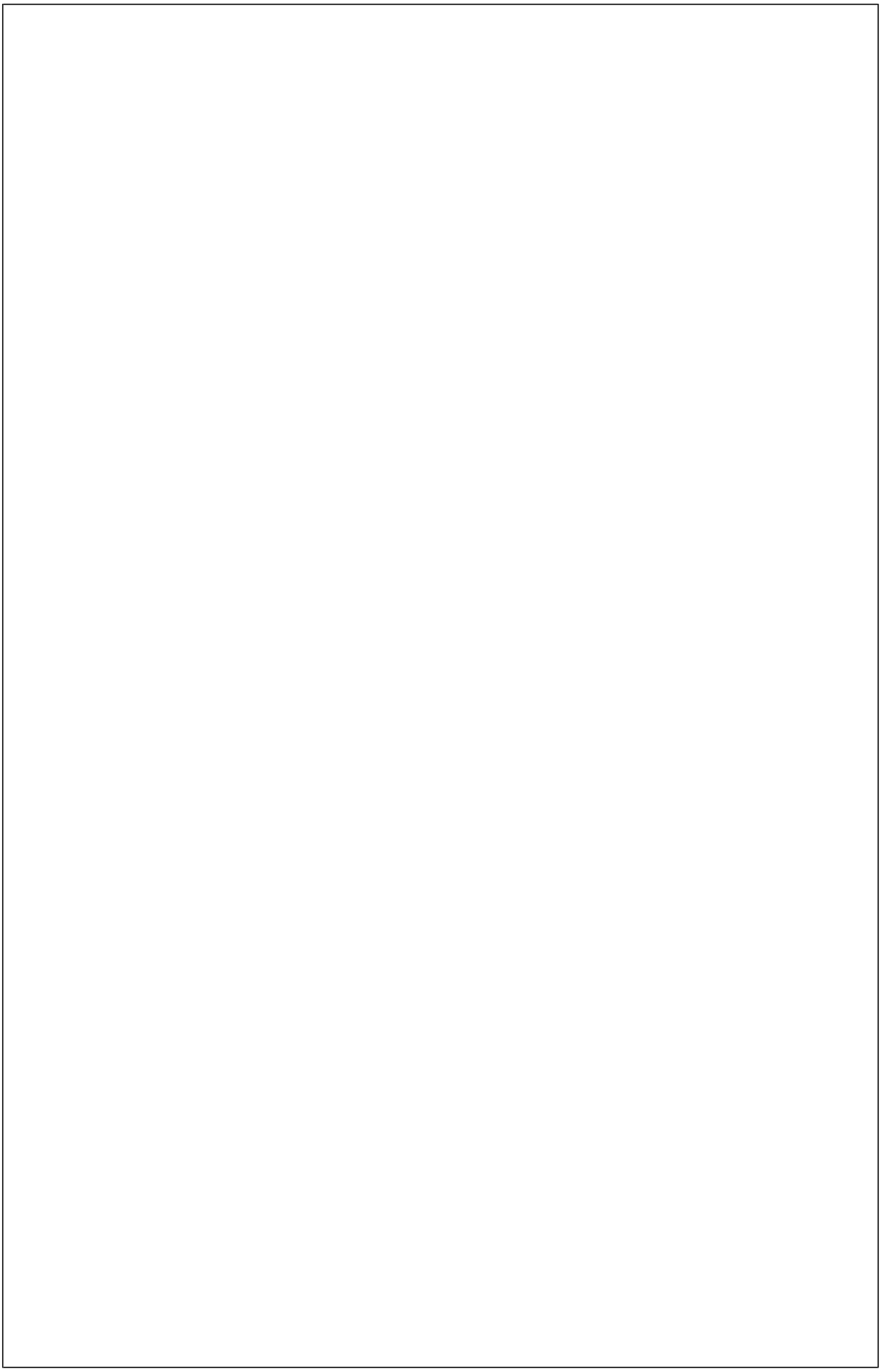
10.2 Praktikum

1. Tulis dan jalankan tiap langkah contoh layanan RMI pada modul ini dan pahami tiap barisnya!
2. Buat program *asSalam* dengan RMI. Dimana client menuliskan namanya kemudian mendapat balasan dari server: “Assalamu’alaikum [nama_client]”

10.3Tugas

1. Sebutkan dan jelaskan contoh sistem/aplikasi yang menggunakan layanan RMI!
2. Buatlah program calculator client-server menggunakan RMI. (tugas individu)
3. Progres projek tugas akhir. Minimal 20% dari target *full-project*. (dikerjakan sesuai kelompok. Bila anggota kelompok di kelas yang berbeda maka progres project dilakukan masing-masing ke tiap asisten atau sesuai kesepakatan dengan asisten)

Jawaban Tugas Pengembangan



MODUL 11

KRIPTOGRAFI

Modul ini membahas tentang kriptografi, enkripsi, dan deskripsi. Kriptografi termasuk dalam dasar keamanan jaringan untuk mengamankan system computer. Oleh karena itu, kriptografi menjadi topik yang harus dibahas ketika ingin meningkatkan keamanan jaringan pada system terdistribusi.

11.1 Pembahasan

Kriptografi berasal dari Bahasa yunani, yaitu *kripto* (rahasia) dan *graphia* (tulisan). Beberapa istilah penting yang perlu diketahui dalam pembahasan kriptografi yaitu: *plaintext*, *chipertext*, enkripsi, deskripsi. ***Plaintext*** adalah pesan asli yang tersusun sehingga dapat dimengerti, baik sebelum dikodekan mapupun setelah disusun kembali dari kode. Dapat pula diartikan dengan informasi yang dapat dimengerti. ***Chipertext*** adalah pesan berupa kode atau informasi yang tidak dapat dimengerti. Proses merubah plaintext menjadi kode (cipher) disebut dengan proses **enkripsi**, sedangkan proses mengembalikan kode (cipher) ke bentuk informasi yang bias dimengerti (plaintext) disebut dengan proses **deskripsi**.

11.1.1 Kriptografi Klasik

Diantara teknik kriptografi klasik yaitu teknik subtitusi cipher serta perkembangannya seperti *shift cipher*, *vigenere cipher*, *hill cipher*. Selain subtitusi terdapat pula teknik transposisi cipher seperti enigma (rotor mesin).

Teknik kriptografi pertama yang dikenalkan pada pemerintahan Yulius Caesar dikenal dengan *Caesar Cipher* yang menggunakan teknik subtitusi. Subtitusi merupakan penggantian setiap karakter dari plaintext dengan karakter lainnya. Contoh sederhana dengan mengganti poisisi huruf awal dari alphabet sebaimana berikut ini:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Plaintext : “SAYA INTELEKTUAL BERJIWA ULUL ALBAB”

Ciphertext : “VDBDLQWHDHNWXDOEHUMLZDXOXODOEDE”

Proses enkripsi Algoritma Caesar chipper tersebut (jika A=0, B=1, dan seterusnya) dapat dirumuskan sebagaimana berikut ini:

$$C = E(P) = (P+K) \bmod (26)$$

sedangkan proses deskripsi dirumuskan dengan:

$$P = D(C) = (C-K) \bmod (26)$$

dengan P adalah plaintext, C adalah ciphertext dan K adalah kunci. Sehingga dari contoh diatas, rumus untuk melakukan enkripsi dan deskripsi adalah sebagai berikut:

$$C = E(P) = (P+3) \bmod (26)$$

$$P = D(C) = (C-3) \bmod (26)$$

Pada perkembangannya, Caesar Cipher bisa menggunakan kunci lain atau bisa disebut dengan polyalphabetic. Kunci bisa jadi nama, alamat, atau apa saja yang diinginkan oleh pengirim pesan.

Contoh 11.1. Kriptografi klasik dengan Modulo 26

```
public class Modulo26Crypto {

    public static void main(String[] args) {
        String plainText = "SISTER";
        String secretKey = "SECRETKEY";
        System.out.println("Plain Text Before Encryption\t: "
                + plainText);
        String encryptedText = encrypt(plainText, secretKey);
        System.out.println("Encrypted Text After Encryption\t: "
                + encryptedText);
        String decryptedText = decrypt(encryptedText, secretKey);
        System.out.println("Decrypted Text After Decryption\t: "
                + decryptedText);
    }

    private static String encrypt(String plainText, String secretKey) {
        StringBuilder encryptedString = new StringBuilder();
        int encryptedInt;
        for (int i = 0; i < plainText.length(); i++) {
            int plainTextInt = (int) (plainText.charAt(i) - 'A');
            int secretKeyInt = (int) (secretKey.charAt(i) - 'A');
            encryptedInt = (plainTextInt + secretKeyInt) % 26;
            encryptedString.append((char) ((encryptedInt) + (int) 'A'));
        }
        return encryptedString.toString();
    }
```

```
    private static String decrypt(String decryptedText, String secretKey) {
        StringBuilder decryptedString = new StringBuilder();
        int decryptedInt;
        for (int i = 0; i < decryptedText.length(); i++) {
            int decryptedTextInt = (int) (decryptedText.charAt(i) - 'A');
            int secretKeyInt = (int) (secretKey.charAt(i) - 'A');
            decryptedInt = decryptedTextInt - secretKeyInt;
            if (decryptedInt < 0) {
                decryptedInt += 26;
            }
            decryptedString.append((char) ((decryptedInt) + (int) 'A'));
        }
        return decryptedString.toString();
    }
}
```

Output:

```
run:
Plain Text Before Encryption    : SISTER
Encrypted Text After Encryption : KMUKIK
Decrypted Text After Decryption : SISTER
BUILD SUCCESSFUL (total time: 2 seconds)
```

11.1.2 Kriptografi Modern

Enkripsi modern menggunakan computer dalam pengoperasiannya. Algoritma kriptografi modern dapat diklasifikasikan menjadi 2 berdasarkan kunci (key) dalam proses enkripsi dan deskripsi. Kriptografi simentris menggunakan kunci yang sama pada proses enkripsi dan deskripsi, sedangkan kriptografi asimetris menggunakan kunci yang berbeda. Symetric Key disebut juga dengan *secret key* sedangkan asymmetric key juga dikenal dengan *public key*.

Simetris

Algoritma yang bayak dipakai disini adalah DES (Data Descryption Standard). Agoritma tersebut dikembangkan menjadi 3DES (Triple Data Descryption Standard). Selain itu, terdapat pula IDEA (International Data Encryption Algorithm).

Implementasi kriptografi modern pada Bahasa Java dapat dilakukan menggunakan *Java Cryptography Extension* (JCE). JCE sudah tersedia pada JAVA SE 8. Untuk versi dibawahnya maka perlu tambahan library JCE. Berikut ini contoh sederhana tentang menggunakan kriptografi "*secret key/symetric*" dengan JCE.

Contoh 11.2 berikut ini menggunakan kunci simetris untuk enkripsi dan deskripsi. Algoritma yang digunakan adalah DESede atau triple DES (3DES) yang merupakan variasi

pengembangan dari algoritma DES. Berikut ini langkah implementasi kriptografi 3DES menggunakan JCE:

- Tambahkan Provider Keamanan. Contoh 11.2 menggunakan Provider SunJCE yang tersedia dengan JDK.
- *Generate secret key*. Gunakan Keygenerator dan algoritma untuk menghasilkan kunci rahasia. Contoh 11.2 menggunakan DESede. Ada algoritma lain seperti Blowfish.
- *Encode Text*. Untuk konsistensi di seluruh platform maka encode plain text bentuk byte menggunakan UTF-8 encoding.
- Proses enkripsi text. Definisikan Cipher sebagai ENCRYPT_MODE, serta gunakan *secret key* kemudian lakukan enkripsi byte dari plain text.
- Proses dekripsi teks. Definisikan Cipher sebagai DECRYPT_MODE, serta gunakan secret key yang sama kemudian dekripsi byte cipher text.

Contoh 11.2. Kriptografi simetrik metode triple DES (3DES) menggunakan JCE

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

public class Crypto {

    private static Cipher cipher = null;

    public static void main(String[] args) throws Exception {

        KeyGenerator keyGenerator = KeyGenerator.getInstance("DESede");
        // keysize must be equal to 112 or 168 for this provider
        keyGenerator.init(168);
        SecretKey secretKey = keyGenerator.generateKey();
        cipher = Cipher.getInstance("DESede");

        String plainText = "Kriptografi menggunakan Java Cryptography Extension";
        System.out.println("Plain Text Before Encryption\t: " + plainText);

        byte[] plainTextByte = plainText.getBytes("UTF8");
        byte[] encryptedBytes = encrypt(plainTextByte, secretKey);

        String encryptedText = new String(encryptedBytes, "UTF8");
        System.out.println("Encrypted Text After Encryption\t: " + encryptedText);

        byte[] decryptedBytes = decrypt(encryptedBytes, secretKey);
        String decryptedText = new String(decryptedBytes, "UTF8");
        System.out.println("Decrypted Text After Decryption\t: " + decryptedText);
    }
```

```
    static byte[] encrypt(byte[] plainTextByte, SecretKey secretKey)
            throws Exception {
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(plainTextByte);
        return encryptedBytes;
    }

    static byte[] decrypt(byte[] encryptedBytes, SecretKey secretKey)
            throws Exception {
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
        return decryptedBytes;
    }
}
```

Output:

```
run:
Plain Text Before Encryption    : Kriptografi menggunakan Java Cryptography Extension
Encrypted Text After Encryption : ^□□Gn□A□□□pv6_`□□□ |□□□ N$4□□□□□Č □m□□ □□□ □ □□e□□ □
Decrypted Text After Decryption : Kriptografi menggunakan Java Cryptography Extension
BUILD SUCCESSFUL (total time: 36 seconds)
```

Data Encryption Standard (DES) rentan terhadap serangan brute-force. Oleh karena itu, DES digantikan dengan *Advanced Encryption Standard* (AES). Masalah utama pada DES adalah pendeknya ukuran kunci sehingga mudah untuk dipecahkan dengan serangan brute force. AES dapat menggunakan 128, 192 atau 256 bit enkripsi. Blok ukuran yang digunakan oleh DES adalah 64 bit sedangkan AES 128 bit. Selain itu, AES menggunakan Permutasi substitusi sedangkan DES menggunakan *Feistel network.*

Berikut ini contoh kriptografi metode AES menggunakan JCE. Pada dasarnya program pada contoh 11.3 ini sama dengan contoh 11.2. Hanya terdapat dua perbedaan, contoh 11.3 ini menggunakan enkripsi AES dan tahap encoding menggunakan Base64. Untuk Java 8, class Base64 sudah tersedia pada package java.util. sedangkan untuk versi dibawah 8 maka perlu menambahkan bundle “Apache commons-code”. Tahap encoding-decoding ini penting untuk mengkodekan data biner dengan Base64 untuk memastikan itu masih utuh tanpa perubahan ketika disimpan atau ditransfer.

Contoh 11.3. Kriptografi Simetris metode AES

```
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

public class AESExample {

    static Cipher cipher;

    public static void main(String[] args) throws Exception {
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init(128);
        SecretKey secretKey = keyGenerator.generateKey();
        cipher = Cipher.getInstance("AES");

        String plainText = "AES Symmetric Encryption Decryption";
        System.out.println("Plain Text Before Encryption\t: " + plainText);

        String encryptedText = encrypt(plainText, secretKey);
        System.out.println("Encrypted Text After Encryption\t: " + encryptedText);

        String decryptedText = decrypt(encryptedText, secretKey);
        System.out.println("Decrypted Text After Decryption\t: " + decryptedText);
    }

    public static String encrypt(String plainText, SecretKey secretKey)
            throws Exception {
        byte[] plainTextByte = plainText.getBytes();
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedByte = cipher.doFinal(plainTextByte);
        Base64.Encoder encoder = Base64.getEncoder();
        String encryptedText = encoder.encodeToString(encryptedByte);
        return encryptedText;
    }

    public static String decrypt(String encryptedText, SecretKey secretKey)
            throws Exception {
        Base64.Decoder decoder = Base64.getDecoder();
        byte[] encryptedTextByte = decoder.decode(encryptedText);
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedByte = cipher.doFinal(encryptedTextByte);
        String decryptedText = new String(decryptedByte);
        return decryptedText;
    }
}
```

Output:

```
run:
Plain Text Before Encryption    : AES Symmetric Encryption Decryption
Encrypted Text After Encryption : ZYDgX/xfsxj/+mTC50y1HSZ3EBFvF0eyJYxYI4oN+4QFkwAUGqh4ILWFP0F/rXH4
Decrypted Text After Decryption : AES Symmetric Encryption Decryption
BUILD SUCCESSFUL (total time: 25 seconds)
```

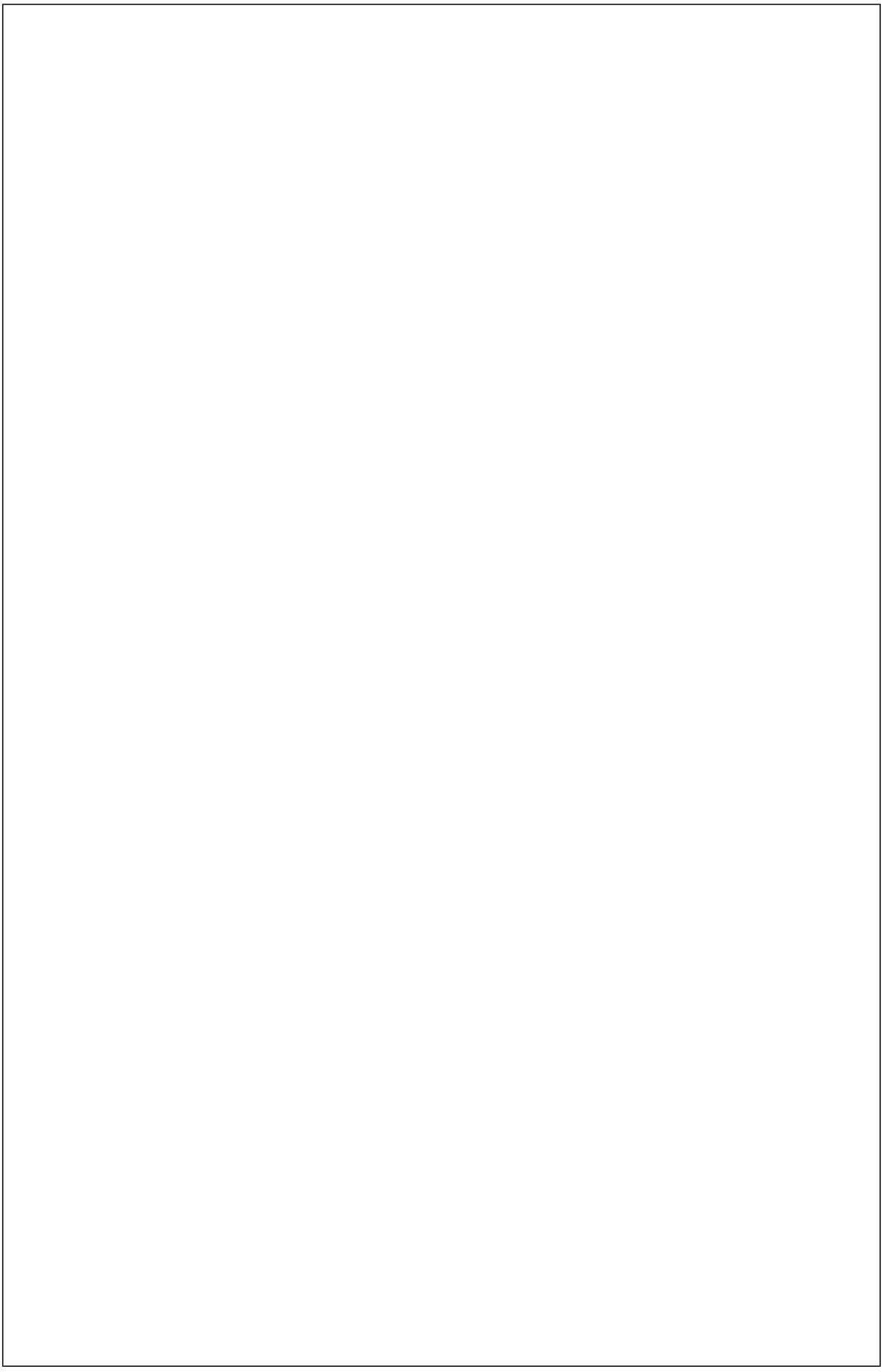
11.2 Praktikum

1. Tulis dan jalankan tiap langkah contoh kriptografi pada modul ini dan pahami tiap barisnya!
2. Pahami kembali contoh kriptografi klasik pada modul ini. Bagaimana proses enkripsi-deskripsi sesuai program tersebut?

11.3 Tugas

1. Modifikasi **Contoh 11.1** sedemikian sehingga menjadi kriptografi Caesar Cipher polyalphabetic.
   a. Jalankan program dengan plaintext nama Anda dan kunci berupa kota lahir Anda.
   b. Tuliskan perhitungan/langkah manual dari poin (a) yang anda jalankan.
2. Buat contoh program yang mengimplementasikan kriptografi asimetris menggunakan RSA!
3. Progres projek tugas akhir. Minimal 40% dari target *full-project*. (dikerjakan sesuai kelompok. Bila anggota kelompok di kelas yang berbeda maka progres project dilakukan masing-masing ke tiap asisten atau sesuai kesepakatan dengan asisten)

Jawaban Tugas Pengembangan



DAFTAR PUSTAKA

[1] Bellifemine, Fabio Luigi, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*. Vol. 7. John Wiley & Sons, 2007.

[2] Harold, Elliotte Rusty. *Java I/O*. "O'Reilly Media, Inc.", 2006.

[3] Harold, Elliotte Rusty. *Java network programming*. "O'Reilly Media, Inc.", 2013.

[4] Kshemkalyani, Ajay D., and Mukesh Singhal. *Distributed computing: principles, algorithms, and systems*. Cambridge University Press, 2011.

[5] Reilly, David, and Michael Reilly. *Java network programming and distributed computing*. Addison-Wesley Professional, 2002.