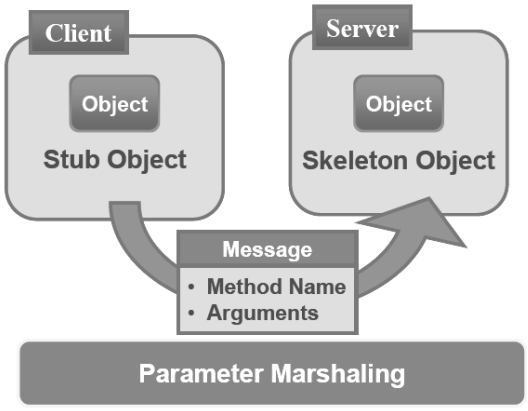
Aturan Praktikum:

- Keterlambatan maksimal 10 menit
- Hasil praktikum, selesai atau tidak selesai ditanggung mahasiswa
- Nilai praktikum adalah nilai perorangan, sebagai tambahan nilai test atau quiz jika nilai test atau quiz kurang mencukupi
- Mahasiswa login ke komputer
- Persiapan dan penjelasan praktikum 10 menit
- Praktikum dilakukan selama 70 menit terdiri dari praktikum sesuai modul
- Hasil praktikum di-zip dan dikumpulkan
- Sisa waktu 20 menit digunakan untuk tes kecil hasil praktikum, saat tes tampilan komputer adalah background desktop dan mahasiswa tidak diperkenankan menggunakan komputer dan melihat modul praktikum
- Mahasiswa logout dari komputer
- Praktikum selesai

A. REMOTE METHOD INVOCATION (RMI) (Minggu 1 – Praktikum II)

1. Sekilas RMI

RMI (*Remote Method Application*) pada Java adalah mekanisme yang mengijinkan sebuah objek untuk mengakses metode objek lainnya yang berada pada alamat lain pada mesin yang sama ataupun mesin yang berbeda.
Arsitektur RMI adalah seperti pada gambar berikut:



Terdapat tiga proses yang ada pada mekanisme RMI, yaitu:

1. Client berupa proses yang melakukan invokasi metode ke sebuah objek *remote*.

2. Server berupa proses yang memiliki objek *remote*. Objek *remote* adalah objek biasa yang berada pada server.
3. *Object Registry* adalah penamaan objek pada server.

Pada Java, sebuah objek *remote* adalah instans dari suatu kelas yang mengimplementasikan antarmuka *remote*.

2. Praktikum

a. Persiapan

- Membuat direktori kerja dengan nama SI319-P2-Kelas-NIM misalnya SI319-P2-A-23507024
- Di dalam direktori di atas, buat direktori RMI untuk menyimpan file-file yang akan dibuat.

b. Membuat Kelas Remote dan antarmuka (*interfaces*)

Kelas *remote* merupakan kelas yang digunakan sebagai antarmuka objek *remote* agar dapat diakses oleh klien. Kelas *remote* memiliki dua buah bagian yaitu sebuah antarmuka dan kelas *remote* itu sendiri. Sebuah antarmuka *remote* harus memiliki properti berikut:

1. antarmuka (*interface*) harus merupakan kelas public
2. antarmuka (*interface*) harus merupakan turunan dari `java.rmi.Remote`
3. setiap metode pada antarmuka (*imterface*) harus melemparkan `java.rmi.RemoteException` atau eksepsi lainnya.

Kelas *remote* harus memiliki properti sebagai berikut:

1. harus merupakan implementasi dari antarmuka *remote*
2. harus merupakan turunan dari kelas java.rmi.server.UnicastRemoteObject.
3. kelas *remote* dapat memiliki metode yang tidak terdapat pada antarmuka *remote* tapi metode ini tidak dapat dipanggil oleh klien.

Nama file: HelloInterface.java

```
import java.rmi.*;
/**
 * antarmuka remote
 */
public interface HelloInterface extends Remote {
  /**
   * metode yang dapat diinvokasi klien
   */
  public String say() throws RemoteException;
}
```

Nama file: Hello.java

```
import java.rmi.*;
import java.rmi.server.*;
/**
 * kelas remote
 */
public class Hello extends UnicastRemoteObject implements
HelloInterface {
  private String message;
  /**
   * konstruktor
   */
  public Hello (String msg) throws RemoteException {
    message = msg;
  }
  /**
   * Implementasi metode say()
   */
  public String say() throws RemoteException {
    return message;
  }
}
```

Antarmuka dan kelas *remote* diatas harus dikompilasi dengan cara sebagai berikut:

1. Buka command prompt, masuk ke direktori RMI
2. Coba ketik perintah: javac
   Jika muncul pesan:

```
'javac' is not recognized as an internal or external command,
operable program or batch file.
```

   Maka ketik perintah:

   path=<tempat direktori instalasi java>\bin;%path%

   contoh:

   path= C:\Program Files\Java\jdk1.5.0_05\bin;%path%
3. Ketik perintah: javac *.java
4. Ketik perintah: rmic Hello
   (perintah ini akan berjalan dengan baik jika rmic menemukan file HelloInterface.class dan Hello.class, oleh karena itu sebelumnya perlu diset path-nya dengan perintah sebelumnya)
5. Ketik perintah: rmiregistry &

c. Membuat Client

Klien yang akan dibuat merupakan program java biasa. Sebuah klien pada RMI membutuhkan kelas *remote* untuk mengetahui metode apa saja yang boleh diakses oleh klien.

Object Registry merupakan sebuah penamaan untuk memanggil metode di server oleh klien. Penamaan *object registry* harus terdiri dari informasi-informasi berikut:

1. alamat internet mesin yang menjalankan *object registry* dimana metode *remote* registrasi, jika klien dan *object registry* berjalan pada mesin yang sama maka nama ini dapat diabaikan
2. port dimana *object registry* menunggu klien (*listening*), jika menggunakan port standar 1099, maka nilai port bisa tidak ditulis
3. nama lokal dari objek *remote* yang ada dalam *object registry*

Nama file: Client.java

```
import java.rmi.*;

class HelloClient{
/**
   * kelas klien
   */
  public static void main (String[] argv) {
    try {
      // memanggil metode dengan melihat object registry
      HelloInterface hello =
        (HelloInterface) Naming.lookup ("//localhost/Hello");
      System.out.println (hello.say());
    } catch (Exception e) {
      System.out.println ("HelloClient exception: " + e);
    }
  }
}
```

`Naming.lookup` pada kode di atas merupakan metode untuk mengakses metode secara *remote* dengan alamat objek yang ada pada *object registry* dan port standar. Hasil dari Naming.lookup harus diubah tipenya menjadi tipe dari antarmuka *remote*.

d. Membuat Server

Server juga hanya berupa program Java. Objek *remote* harus telah terdapat pada objek *registry* karena server akan mengikat objek *remote*, dengan menggunakan `Naming.rebind (objectName, object);` dimana object adalah objek yang telah diregister dan objectName adalah nama objek *remote* yang telah diregister.

Nama file: HelloServer.java

```
import java.rmi.*;

class HelloServer{
/**
   * server remote
   */
  public static void main (String[] argv) {
    try {
          Naming.rebind ("Hello", new Hello ("Hello, world!"));
      System.out.println ("Hello Server is ready.");
    } catch (Exception e) {
      System.out.println ("Hello Server failed: " + e);
    }
  }
}
```

e. Menjalankan Aplikasi

1. Buka command prompt dan ketik perintah: rmiregistry &
2. Buka command prompt kedua, masuk ke direktori RMI
3. Kompilasi server dengan mengetik perintah: javac HelloServer.java
4. Kemudian jalankan server dengan mengetik perintah: java HelloServer &
5. Buka command prompt ketiga, masuk ke direktori RMI
6. kompilasi klien dengan mengetik perintah: javac HelloClient.java
7. jalankan klien dengan mengetik perintah: java HelloClient
8. Ketik ctrl+c untuk menghentikan aplikasi

B. RMI-IIOP (Internet InterORB Protocol)

RMI-IIOP adalah antarmuka RMI yang menggunakan IIOP(*Internet Inter-ORB Protocol*) sebagai koneksi komunikasi. RMI-IIOP menyediakan antarmuka operasi dengan objek CORBA(*Common Object Request Broker Architecture*) (akan dibahas pada minggu ke 4). Dengan menggunakan IIOP, RMI dapat diakses oleh klien dari CORBA yang dapat ditulis dengan menggunakan berbagai macam bahasa pemrograman.

1. Praktikum

a. Persiapan

- Di dalam direktori SI319-P2-Kelas-NIM, buat direktori RMIIIOP untuk menyimpan file-file yang akan dibuat.

b. Membuat antarmuka RMI

Nama file: HelloInterface.java

```
import java.rmi.Remote;

public interface HelloInterface extends java.rmi.Remote {
   public void sayHello( String from ) throws
     java.rmi.RemoteException;
}
```

c. Membuat kelas implementasi objek *remote* (*Servant*)

Kelas implementasi objek *remote* merupakan kelas yang dapat membuat objek *remote* dengan menggunakan IIOP sebagai protokol komunikasi

Nama file: HelloImpl.java

```
import javax.rmi.PortableRemoteObject;

public class HelloImpl extends PortableRemoteObject implements
HelloInterface {
   public HelloImpl() throws java.rmi.RemoteException {
       super();     // invoke rmi linking dan remote object
initialization
   }

   public void sayHello( String from ) throws
java.rmi.RemoteException {
       System.out.println( "Hello from " + from + "!!" );
       System.out.flush();
   }
}
```

d. Membuat Server

Kelas server merupakan kelas yang memiliki metode main untuk membuat instans dari implementasi objek *remote* dan mengikat instans itu dengan penamaan dalam Naming Services.

Nama file: HelloServer.java

```
import javax.naming.InitialContext;
import javax.naming.Context;


public class HelloServer {
    public static void main(String[] args) {
        try {
            // 1: menginstansiasi objek implementasi objek remote
            HelloImpl helloRef = new HelloImpl();

            // 2: mempublikasikan referensi Naming Services
```

```
            // dengan JNDI API
            Context initialNamingContext = new InitialContext();
            initialNamingContext.rebind("HelloService", helloRef );

            System.out.println("Hello Server: Ready...");

         } catch (Exception e) {
            System.out.println("Trouble: " + e);
            e.printStackTrace();
         }
     }
}
```

e. Membuat Client

Aplikasi klien memanggil metode sayHello().

Nama file: HelloClient.java

```
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;
import javax.rmi.*;
import java.util.Vector;
import javax.naming.NamingException;
import javax.naming.InitialContext;
import javax.naming.Context;

public class HelloClient {

    public static void  main( String args[] ) {
        Context ic;
        Object objref;
        HelloInterface hi;

        try {
            ic = new InitialContext();

        // 1: mengambil referensi objek dari Naming Services
        // dengan JNDI call.
            objref = ic.lookup("HelloService");
            System.out.println(
              "Client: Obtained a ref. to Hello server.");

        // 2: mengkhususkan referensi objek ke tipe yang lebih konkret
        // dan invoke metode.
            hi = (HelloInterface) PortableRemoteObject.narrow(
                objref, HelloInterface.class);
            hi.sayHello( " MARS " );

        } catch( Exception e ) {
            System.err.println( "Exception " + e + "Caught" );
            e.printStackTrace( );
            return;
```

```
        }
    }
}
```

f. Menjalankan aplikasi

1. Buka command prompt, masuk ke direktori RMIIIOP
2. Ketik perintah: javac -d . -classpath . HelloImpl.java
   dimana "–d ." berarti hasil file yang digenerasi disimpan di dalam direktori dimana kompilasi dilakukan, "-classpath ." berarti semua file yang diperlukan HelloImpl.java ada pada direktori kerja tempat kompilasi dilakukan
3. Ketik perintah: rmic -iiop HelloImpl
   perintah ini akan menghasilkan file:
   - `_HelloInterface_Stub.class` – berperan sebagai stub milik klien pada CORBA
   - `_HelloImpl_Tie.class` – berperan sebagai skeletons milik server pada CORBA
4. Ketik perintah: javac -d . -classpath . HelloInterface.java HelloServer.java HelloClient.java
5. Ketik perintah: start orbd -ORBInitialPort 1050 hingga muncul jendela ORB
6. Buka commad prompt kedua, masuk ke direktori RMIIIOP
7. Ketik perintah:
   ```
   java -classpath .
   -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
   -Djava.naming.provider.url=iiop://localhost:1050
   HelloServer
   ```

   hingga muncul

   ```
   Hello Server: Ready ...
   ```
8. Buka command prompt ketiga, masuk ke direktori RMIIIOP
9. Ketik perintah:
   ```
   java
   -classpath .
   -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
   -Djava.naming.provider.url=iiop://localhost:1050
   HelloClient
   ```

   hingga muncul

   `Client: Obtained a ref. to Hello server.` [pada jendela client]
   `Hello from  MARS` [pada jendela server]

10. Tekan ctrl+c pada command prompt kedua untuk membunuh server dan jendela ORB.