Modul 9. Remote Procedure Call

Tugas Pendahuluan :

1. Apa yang dimaksud dengan Remote Procedure Call (RPC) dalam konteks pemrograman jaringan? Jelaskan konsep dasar dari RPC dan bagaimana ia memfasilitasi komunikasi antara proses yang berjalan di jaringan yang berbeda.
2. Bagaimana RPC berbeda dengan lokal procedure call (pemanggilan prosedur lokal)?
3. Apa peran stub dalam RPC, dan mengapa diperlukan? Jelaskan peran stub dalam memediasi panggilan prosedur antara klien dan server dalam arsitektur RPC.
4. Mengapa keselamatan dan keamanan menjadi perhatian utama dalam implementasi RPC?

Meteri :

Remote Procedure Call (RPC) adalah sebuah protokol komunikasi antarproses yang memungkinkan suatu program komputer untuk memanggil prosedur atau fungsi pada komputer lain dalam jaringan yang terhubung melalui jaringan komputer, seperti Internet. Dengan menggunakan RPC, program di satu komputer dapat meminta layanan atau informasi dari program di komputer lain dan menjalankan fungsi-fungsi di sana secara transparan, seolah-olah program tersebut berjalan pada komputer yang sama.

Pada dasarnya, RPC berfungsi sebagai penghubung antara client dan server, dimana client meminta layanan tertentu dan server menjalankan fungsi tersebut dan mengirimkan hasil kembali ke client. Penggunaan RPC biasanya digunakan dalam arsitektur Client-Server dalam sistem distribusi, seperti pada aplikasi web, database, sistem operasi, dan sebagainya.

Di Java, implementasi RPC dapat dilakukan menggunakan beberapa teknologi, seperti RMI (Remote Method Invocation), CORBA (Common Object Request Broker Architecture), dan JSON-RPC (JSON Remote Procedure Call).

RMI memungkinkan objek Java untuk dipanggil secara remote melalui protokol TCP/IP, dengan menggunakan objek sebagai parameter atau hasil. CORBA menggunakan ORB (Object Request Broker) sebagai mekanisme untuk memanggil objek yang didefinisikan oleh IDL (Interface Definition Language). Sedangkan JSON-RPC menggunakan format JSON untuk encoding data dan request/response. Berikut adalah contoh sederhana menggunakan RMI di Java:

Buat interface remote

```
1. import java.rmi.Remote;
2. import java.rmi.RemoteException;
3.
4. public interface MyRemoteInterface extends Remote {
5.     public String sayHello() throws RemoteException;
6. }
7.
8. Buat implementasi interface remote
9.
10. import java.rmi.RemoteException;
11. import java.rmi.server.UnicastRemoteObject;
12.
13. public class MyRemoteImpl extends UnicastRemoteObject implements MyRemoteInterface {
14.     public MyRemoteImpl() throws RemoteException {}
15.
```

```
16.     public String sayHello() throws RemoteException {
17.         return "Hello, World!";
18.     }
19. }
20.
```

Buat server

```
 1. import java.rmi.registry.LocateRegistry;
 2. import java.rmi.registry.Registry;
 3.
 4. public class MyRemoteServer {
 5.     public static void main(String[] args) {
 6.         try {
 7.             MyRemoteInterface stub = new MyRemoteImpl();
 8.             Registry registry = LocateRegistry.createRegistry(1099);
 9.             registry.bind("MyRemoteInterface", stub);
10.             System.out.println("Server ready");
11.         } catch (Exception e) {
12.             e.printStackTrace();
13.         }
14.     }
15. }
16.
```

Buat Client

```
 1. import java.rmi.registry.LocateRegistry;
 2. import java.rmi.registry.Registry;
 3.
 4. public class MyRemoteClient {
 5.     public static void main(String[] args) {
 6.         try {
 7.             Registry registry = LocateRegistry.getRegistry("localhost");
 8.             MyRemoteInterface stub = (MyRemoteInterface)
registry.lookup("MyRemoteInterface");
 9.             String response = stub.sayHello();
10.             System.out.println(response);
11.         } catch (Exception e) {
12.             e.printStackTrace();
13.         }
14.     }
15. }
16.
```

Dalam contoh di atas, server akan membuat objek MyRemoteImpl sebagai implementasi dari interface MyRemoteInterface, dan mendaftarkannya ke dalam RMI registry dengan nama "MyRemoteInterface". Sedangkan pada client, akan mengambil objek dari RMI registry dengan menggunakan nama yang sama dan memanggil metode sayHello() yang ada pada interface MyRemoteInterface.

Untuk menjalankan contoh implementasi RPC menggunakan RMI di Java, langkah-langkahnya adalah sebagai berikut:

1. Buat file Java dengan nama MyRemoteInterface.java dan isi dengan kode pada langkah 1 di atas.
2. Buat file Java dengan nama MyRemoteImpl.java dan isi dengan kode pada langkah 2 di atas.
3. Buat file Java dengan nama MyRemoteServer.java dan isi dengan kode pada langkah 3 di atas.
4. Buat file Java dengan nama MyRemoteClient.java dan isi dengan kode pada langkah 4 di atas.

5. Kompilasi semua file Java menggunakan perintah javac pada terminal/command prompt:

```
javac MyRemoteInterface.java
javac MyRemoteImpl.java
javac MyRemoteServer.java
javac MyRemoteClient.java
```

6. Jalankan RMI registry pada terminal/command prompt:

```
start rmiregistry
```

7. Jalankan server pada terminal/command prompt:

```
java MyRemoteServer
```

8. Jalankan client pada terminal/command prompt:

```
java MyRemoteClient
```

Output yang diharapkan adalah "Hello, World!" pada client.

Dalam implementasi RPC di Java, terdapat beberapa kelas dan API yang bisa digunakan, antara lain:

1. RMI (Remote Method Invocation) API, yaitu kelas-kelas yang terdapat pada paket java.rmi, seperti Remote, RemoteException, UnicastRemoteObject, Naming, dan sebagainya. RMI digunakan untuk memanggil metode pada objek yang berada di komputer yang berbeda.
2. CORBA (Common Object Request Broker Architecture) API, yaitu kelas-kelas yang terdapat pada paket org.omg.CORBA, seperti ORB, NamingContextExt, NamingContextExtHelper, dan sebagainya. CORBA digunakan untuk memanggil objek yang didefinisikan oleh IDL (Interface Definition Language) melalui ORB (Object Request Broker).
3. JSON-RPC API, yaitu kelas-kelas yang terdapat pada paket com.thetransactioncompany.jsonrpc2, seperti JSONRPC2Request, JSONRPC2Response, dan sebagainya. JSON-RPC digunakan untuk memanggil fungsi-fungsi yang terdapat pada server melalui protokol JSON.
4. Spring Remoting API, yaitu kelas-kelas yang terdapat pada paket org.springframework.remoting, seperti RemoteInvocation, RemoteInvocationHandler, HttpInvokerProxyFactoryBean, HessianServiceExporter, dan sebagainya. Spring Remoting digunakan untuk mempermudah penggunaan teknologi RPC di Java dengan framework Spring.

Berikut adalah contoh implementasi RPC menggunakan Common Object Request Broker Architecture (CORBA) di Java:

Buat IDL file

```
1. module HelloApp {
2.     interface Hello {
3.         string sayHello();
4.     };
5. };
6.
```

Compile IDL file menggunakan command idlj

```
idlj -fall Hello.idl
```

Implementasi server

```
 1. import org.omg.CORBA.*;
 2. import HelloApp.*;
 3.
 4. public class HelloServer extends HelloPOA {
 5.     private ORB orb;
 6.
 7.     public void setORB(ORB orb_val) {
 8.         orb = orb_val;
 9.     }
10.
11.     public String sayHello() {
12.         return "Hello, World!";
13.     }
14.
15.     public void shutdown() {
16.         orb.shutdown(false);
17.     }
18.
19.     public static void main(String args[]) {
20.         try {
21.             ORB orb = ORB.init(args, null);
22.             HelloServer helloRef = new HelloServer();
23.             helloRef.setORB(orb);
24.
25.             org.omg.CORBA.Object objRef = orb.resolve_initial_references("RootPOA");
26.             POA rootpoa = POAHelper.narrow(objRef);
27.             rootpoa.the_POAManager().activate();
28.
29.             org.omg.CORBA.Object obj = rootpoa.servant_to_reference(helloRef);
30.             Hello href = HelloHelper.narrow(obj);
31.
32.             org.omg.CORBA.Object objRef2 = orb.resolve_initial_references("NameService");
33.             NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef2);
34.
35.             NameComponent path[] = ncRef.to_name("Hello");
36.             ncRef.rebind(path, href);
37.
38.             System.out.println("Server ready");
39.
40.             orb.run();
41.         } catch (Exception e) {
42.             System.err.println("ERROR: " + e);
43.             e.printStackTrace(System.out);
44.         }
45.         System.out.println("HelloServer Exiting ...");
46.     }
47. }
48.
```

Implementasi client

```
 1. import org.omg.CosNaming.*;
 2. import org.omg.CosNaming.NamingContextPackage.*;
 3. import org.omg.CORBA.*;
 4.
 5. public class HelloClient {
 6.     public static void main(String args[]) {
 7.         try {
 8.             ORB orb = ORB.init(args, null);
 9.             org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
10.             NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
11.
```

```
12.             Hello helloRef = (Hello) HelloHelper.narrow(ncRef.resolve_str("Hello"));
13.
14.             String response = helloRef.sayHello();
15.             System.out.println(response);
16.
17.             helloRef.shutdown();
18.
19.         } catch (Exception e) {
20.             System.out.println("ERROR : " + e);
21.             e.printStackTrace(System.out);
22.         }
23.     }
24. }
25.
```

Dalam contoh di atas, IDL file digunakan untuk mendefinisikan antarmuka objek remote yang akan digunakan. Setelah IDL file di-compile menggunakan command idlj, akan dihasilkan beberapa kelas Java, termasuk Hello, HelloHelper, dan HelloPOA. Kelas-kelas ini akan digunakan pada implementasi server dan client.

Pada implementasi server, objek HelloServer merupakan implementasi dari objek remote yang di-definisikan pada IDL file. Objek tersebut kemudian didaftarkan ke dalam NameService menggunakan NamingContextExt. Sedangkan pada implementasi client, objek HelloClient akan mengambil objek remote dari NameService menggunakan NamingContextExt dan memanggil metode sayHello() yang ada pada objek remote tersebut.

Untuk menjalankan contoh di atas, pastikan sudah mengikuti langkah-langkah yang sudah dijelaskan pada jawaban sebelumnya.

JSON-RPC

JSON-RPC (Remote Procedure Call) adalah sebuah protokol yang digunakan untuk memanggil metode pada objek yang berada pada jarak jauh melalui HTTP atau protokol transportasi lainnya. Seperti namanya, JSON-RPC menggunakan format data JSON untuk mengirimkan permintaan dan menerima respons dari server jarak jauh.

Dalam JSON-RPC, metode-metode pada objek remote didefinisikan dengan cara yang sama seperti pada RPC lainnya, di mana setiap metode memiliki parameter masukan dan keluaran. Setiap permintaan JSON-RPC berisi objek JSON yang berisi nama metode, parameter masukan, dan ID transaksi. Setiap respons JSON-RPC berisi objek JSON yang berisi hasil dari metode yang dipanggil atau pesan kesalahan jika terjadi kesalahan pada server.

JSON-RPC biasanya digunakan dalam lingkungan aplikasi yang terdistribusi, di mana komponen-komponen aplikasi berjalan pada mesin yang berbeda dan perlu saling berkomunikasi secara efisien. JSON-RPC memiliki banyak keuntungan, antara lain:

1. Mudah digunakan dan dipelajari karena menggunakan format JSON yang populer.
2. Mendukung berbagai jenis protokol transportasi, seperti HTTP, TCP/IP, dan WebSocket.
3. Dapat digunakan pada berbagai jenis bahasa pemrograman, tidak terbatas pada Java atau platform tertentu.
4. Lebih ringan dan lebih efisien daripada protokol SOAP yang lebih kompleks.
5. Dapat digunakan untuk mengakses sumber daya web yang terdistribusi, seperti REST API dan layanan web lainnya.

JSON-RPC dapat digunakan untuk berbagai jenis aplikasi, seperti aplikasi web, mobile, dan perangkat lunak terdistribusi lainnya. Beberapa kerangka kerja seperti Spring Framework dan Express.js menyediakan dukungan bawaan untuk JSON-RPC, sehingga memudahkan pengembang untuk mengintegrasikan protokol ini ke dalam aplikasi mereka.

Berikut adalah contoh implementasi Remote Procedure Call (RPC) menggunakan JSON-RPC di Java:

Tambahkan dependensi JSON-RPC pada file pom.xml

```
<dependencies>
    <dependency>
        <groupId>com.googlecode.json-rpc</groupId>
        <artifactId>json-rpc</artifactId>
        <version>1.1.2</version>
    </dependency>
</dependencies>
```

Buat antarmuka objek remote

```
1. public interface CalculatorService {
2.     int add(int a, int b);
3.     int subtract(int a, int b);
4. }
5.
```

Implementasi objek remote

```
 1. public class CalculatorServiceImpl implements CalculatorService {
 2.     public int add(int a, int b) {
 3.         return a + b;
 4.     }
 5.
 6.     public int subtract(int a, int b) {
 7.         return a - b;
 8.     }
 9. }
10.
```

Buat server

```
 1. import com.thetransactioncompany.jsonrpc2.JSONRPC2Error;
 2. import com.thetransactioncompany.jsonrpc2.JSONRPC2Request;
 3. import com.thetransactioncompany.jsonrpc2.JSONRPC2Response;
 4. import com.thetransactioncompany.jsonrpc2.server.Dispatcher;
 5. import com.thetransactioncompany.jsonrpc2.server.RequestHandler;
 6. import com.thetransactioncompany.jsonrpc2.server.MessageContext;
 7. import com.thetransactioncompany.jsonrpc2.server.NotificationHandler;
 8. import java.util.HashMap;
 9. import java.util.Map;
10.
11. public class CalculatorServer {
12.     public static void main(String[] args) {
13.         CalculatorService calculatorService = new CalculatorServiceImpl();
14.         Map<String, RequestHandler> handlerMap = new HashMap<>();
15.         handlerMap.put("add", calculatorService::add);
16.         handlerMap.put("subtract", calculatorService::subtract);
17.         Dispatcher dispatcher = new Dispatcher(handlerMap);
18.         try {
19.             JSONRPC2Request req =
JSONRPC2Request.parse("{\"jsonrpc\":\"2.0\",\"method\":\"add\",\"params\":[3, 5],\"id\":1}");
```

```
            JSONRPC2Response res = dispatcher.process(req, null);
            System.out.println(res);
        } catch (JSONRPC2Error e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Buat client

```
import com.thetransactioncompany.jsonrpc2.JSONRPC2Error;
import com.thetransactioncompany.jsonrpc2.JSONRPC2Request;
import com.thetransactioncompany.jsonrpc2.JSONRPC2Response;
import com.thetransactioncompany.jsonrpc2.client.*;
import java.net.*;

public class CalculatorClient {
    public static void main(String[] args) throws Exception {
        URL serverURL = new URL("http://localhost:8080");
        JSONRPC2Session session = new JSONRPC2Session(serverURL);
        int a = 3;
        int b = 5;
        JSONRPC2Request req = new JSONRPC2Request("add", new Object[]{a, b}, 1);
        JSONRPC2Response res = session.send(req);
        if (res.indicatesSuccess()) {
            int sum = (int) res.getResult();
            System.out.println(a + " + " + b + " = " + sum);
        } else {
            JSONRPC2Error err = res.getError();
            System.out.println(err.getMessage());
        }
    }
}
```

Dalam contoh di atas, dependensi JSON-RPC digunakan untuk membantu dalam mengirim dan menerima request dan response JSON-RPC. Antarmuka objek remote pada contoh ini adalah CalculatorService yang memiliki dua metode, yaitu add() dan subtract(). Implementasi objek remote pada contoh

Untuk menjalankan contoh implementasi Remote Procedure Call (RPC) menggunakan JSON-RPC di Java, Anda dapat mengikuti langkah-langkah berikut:

1. Pastikan Anda telah menginstal JDK dan Maven pada komputer Anda.
2. Unduh contoh implementasi JSON-RPC dari repository GitHub atau salin kode program dari contoh di atas dan simpan pada direktori lokal.
3. Buka terminal atau command prompt, lalu navigasi ke direktori program yang telah disimpan.
4. Jalankan perintah mvn clean package untuk membersihkan dan membangun program.
5. Setelah proses pembangunan selesai, jalankan server dengan menjalankan kelas CalculatorServer menggunakan perintah java -cp target/json-rpc-example-1.0-SNAPSHOT.jar CalculatorServer.
6. Untuk menjalankan client, jalankan kelas CalculatorClient menggunakan perintah java -cp target/json-rpc-example-1.0-SNAPSHOT.jar CalculatorClient.
7. Perhatikan output yang ditampilkan di terminal atau command prompt untuk melihat hasil dari operasi RPC.

Catatan: Pastikan server telah berjalan sebelum menjalankan client. Juga pastikan untuk menyesuaikan URL server pada kode program client sesuai dengan URL server yang digunakan pada kode program server.

Spring Remoting API

Spring Remoting API adalah sebuah fitur di kerangka kerja Spring Framework yang memungkinkan pengembang untuk membuat aplikasi yang dapat berkomunikasi melalui Remote Procedure Call (RPC) menggunakan berbagai teknologi, seperti HTTP, RMI, Hessian, Burlap, dan lain-lain.

Dalam Spring Remoting, kelas-kelas yang dijuluki sebagai "proxy" (atau "adaptor") digunakan untuk memfasilitasi komunikasi antara klien dan server. Proxy ini menyediakan antarmuka yang sama antara klien dan server, sehingga klien dapat memanggil metode pada server seolah-olah itu adalah objek lokal.

Dalam Spring Remoting, klien dan server diimplementasikan sebagai dua aplikasi yang berbeda, yang biasanya dijalankan pada mesin yang berbeda atau dalam container aplikasi yang berbeda. Dalam aplikasi klien, proxy digunakan untuk menghubungi server jarak jauh dan memanggil metode pada objek remote. Sedangkan pada aplikasi server, proxy digunakan untuk mengekspos objek remote kepada aplikasi klien.

Spring Remoting API memberikan banyak manfaat, antara lain:

1. Memudahkan pengembang untuk membuat aplikasi yang dapat berkomunikasi melalui RPC menggunakan berbagai teknologi.
2. Membuat pengembangan aplikasi lebih mudah dan lebih produktif dengan menyembunyikan kompleksitas dari komunikasi jarak jauh.
3. Membuat aplikasi lebih modular dan memungkinkan pengembang untuk mengganti teknologi komunikasi jarak jauh tanpa mengubah kode aplikasi yang ada.
4. Memudahkan pengembang untuk memonitor dan mengelola aplikasi dengan menyediakan banyak fitur di Spring Framework, seperti manajemen transaksi, keamanan, dan lain-lain.

Spring Remoting API dapat digunakan untuk berbagai jenis aplikasi, seperti aplikasi bisnis, aplikasi web, dan aplikasi mobile. Berikut adalah contoh implementasi Remote Procedure Call (RPC) menggunakan Spring Remoting API di Java:

1. Tambahkan dependensi Spring Remoting pada file pom.xml

```
1. <dependencies>
2.     <dependency>
3.         <groupId>org.springframework</groupId>
4.         <artifactId>spring-webmvc</artifactId>
5.         <version>5.3.16</version>
6.     </dependency>
7. </dependencies>
8.
```

2. Buat antarmuka objek remote

```
1. public interface CalculatorService {
2.     int add(int a, int b);
3.     int subtract(int a, int b);
```

```
4. }
5.
```

3. Implementasi objek remote

```
1. public class CalculatorServiceImpl implements CalculatorService {
2.     public int add(int a, int b) {
3.         return a + b;
4.     }
5.
6.     public int subtract(int a, int b) {
7.         return a - b;
8.     }
9. }
10.
```

4. Konfigurasi Spring untuk mengaktifkan Spring Remoting

```
1. <beans xmlns="http://www.springframework.org/schema/beans"
2.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.     xmlns:context="http://www.springframework.org/schema/context"
4.     xmlns:util="http://www.springframework.org/schema/util"
5.     xmlns:remoting="http://www.springframework.org/schema/remoting"
6.     xsi:schemaLocation="
7.         http://www.springframework.org/schema/beans
8.         http://www.springframework.org/schema/beans/spring-beans.xsd
9.         http://www.springframework.org/schema/context
10.        http://www.springframework.org/schema/context/spring-context.xsd
11.        http://www.springframework.org/schema/util
12.        http://www.springframework.org/schema/util/spring-util.xsd
13.        http://www.springframework.org/schema/remoting
14.        http://www.springframework.org/schema/remoting/spring-remoting.xsd">
15.
16.    <context:component-scan base-package="com.example.rpc"/>
17.
18.    <bean id="calculatorService" class="com.example.rpc.CalculatorServiceImpl"/>
19.
20.    <remoting:service id="calculatorServiceExporter" service-ref="calculatorService"
21.                     service-interface="com.example.rpc.CalculatorService"
22.                     service-name="CalculatorService"/>
23.
24. </beans>
25.
```

Buat server

```
1. import org.springframework.context.ApplicationContext;
2. import org.springframework.context.support.ClassPathXmlApplicationContext;
3. import com.example.rpc.CalculatorService;
4.
5. public class CalculatorServer {
6.     public static void main(String[] args) {
7.         ApplicationContext context = new ClassPathXmlApplicationContext("spring-
config.xml");
8.         CalculatorService calculatorService = (CalculatorService)
context.getBean("calculatorService");
9.         System.out.println(calculatorService.add(3, 5));
10.    }
11. }
12.
```

Buat client

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.example.rpc.CalculatorService;

public class CalculatorClient {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("spring-
config.xml");
        CalculatorService calculatorService = (CalculatorService)
context.getBean("calculatorService");
        System.out.println(calculatorService.add(3, 5));
    }
}

```

Dalam contoh di atas, dependensi Spring Remoting digunakan untuk membantu dalam implementasi Remote Procedure Call (RPC) menggunakan Spring. Antarmuka objek remote pada contoh ini adalah CalculatorService yang memiliki dua metode, yaitu add() dan subtract(). Implementasi objek remote pada contoh ini adalah CalculatorServiceImpl. Konfigurasi Spring pada contoh ini menggunakan Spring XML configuration. Konfigurasi tersebut mengaktifkan Spring Remoting dan mengekspor.

Latihan 1 :

Tugas Anda adalah menemukan dan memperbaiki kesalahan sintaks, logika, atau implementasi yang mungkin ada dalam program-program ini agar program dapat berjalan dengan benar. Program ini melibatkan Remote Method Invocation (RMI) untuk mengimplementasikan RPC di Java.

```
// Server.java
import java.rmi.*;
import java.rmi.server.*;

public class Server {
    public static void main(String[] args) {
        try {
            HelloImpl obj = new HelloImpl();
            Naming.rebind("HelloServer", obj);
            System.out.println("Server is ready.");
        } catch (Exception e) {
            System.out.println("Server error: " + e.getMessage());
        }
    }
}


```

```
// HelloImpl.java
import java.rmi.*;
import java.rmi.server.*;

public class HelloImpl extends UnicastRemoteObject implements Hello {
    public HelloImpl() throws RemoteException {
        super();
    }

    public String sayHello(String name) {
        return "Hello, " + name + "!";
    }
}

```

```java
// Hello.java
import java.rmi.*;

public interface Hello extends Remote {
    public String sayHello(String name) throws RemoteException;
}

```

```java
// Client.java
import java.rmi.*;

public class Client {
    public static void main(String[] args) {
        try {
            Hello obj = (Hello) Naming.lookup("HelloServer");
            String message = obj.sayHello("John");
            System.out.println("Response from server: " + message);
        } catch (Exception e) {
            System.out.println("Client error: " + e.getMessage());
        }
    }
}

```

Latihan 2:

Berikut adalah contoh program Java menggunakan library Java RMI (Remote Method Invocation). Program ini mencakup beberapa kesalahan sintaks dan logika yang perlu diperbaiki:

```java
import java.rmi.*;
import java.rmi.server.*;

public class Server {
    public static void main(String[] args) {
        try {
            HelloImpl obj = new HelloImpl();
            Naming.rebind("HelloServer", obj);
            System.out.println("Server is ready.");
        } catch (Exception e) {
            System.out.println("Server error: " + e.getMessage());
        }
    }
}

class HelloImpl extends UnicastRemoteObject implements Hello {
    public HelloImpl() throws RemoteException {
        super();
    }

    public String sayHello(String name) {
        return "Hello, " + name + "!";
    }
}

interface Hello extends Remote {
    String sayHello(String name) throws RemoteException;
}

public class Client {
    public static void main(String[] args) {
        try {
            Hello obj = (Hello) Naming.lookup("HelloServer");
            String message = obj.sayHello("John");
```

```
35.             System.out.println("Response from server: " + message);
36.         } catch (Exception e) {
37.             System.out.println("Client error: " + e.getMessage());
38.         }
39.     }
40. }
41.
```

Tugas Rumah :

1. Tujuan Tugas:

   Mengembangkan program Java yang melibatkan Remote Procedure Call (RPC) dalam konteks Network Programming.

2. Deskripsi Tugas:

   Anda diminta untuk membuat program sederhana yang melibatkan RPC untuk berkomunikasi antara klien dan server melalui jaringan. Program ini akan mensimulasikan pengiriman pesan antara dua entitas.

3. Spesifikasi Program:

   1) Implementasikan dua program terpisah: ChatServer dan ChatClient.
   2) ChatServer:
      - Menangani penerimaan koneksi dari ChatClient.
      - Memiliki metode sendMessage yang dapat dipanggil oleh ChatClient untuk mengirim pesan ke server.
      - Mengirim pesan yang diterima dari ChatClient kepada semua ChatClient yang terhubung.
   3) ChatClient:
      - Menghubungkan ke ChatServer.
      - Memiliki metode receiveMessage untuk menerima pesan dari ChatServer dan menampilkannya di konsol.
      - Memiliki metode sendMessage untuk mengirim pesan ke ChatServer.

Catatan:

Gunakan RPC atau metode komunikasi jarak jauh yang sesuai untuk mengimplementasikan panggilan antara klien dan server. Pastikan untuk menangani pengecualian dan situasi error yang mungkin terjadi selama eksekusi program.

Uji program Anda dengan membuat beberapa instance ChatClient yang terhubung ke ChatServer dan dapat saling mengirim pesan.

4. Penting:

   Jangan lupa menyertakan komentar di dalam kode program untuk menjelaskan logika atau alur kerja yang diimplementasikan.