UNIT-8 [Around 10 marks]

RMI and CORBA

⊛Introduction of RMI: (RMI def^n Imp)

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. A distributed system, also known as distributed computing, is a system with multiple components located on different machines that communicate and coordinate actions in order to appear as a single coherent system to the end-user. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects stub and skeleton.

Goals of RMI:

→To minimize the complexity of the application.
→To preserve type safety.
→Distributed garbage collection.
→Minimize the difference between working with local and remote objects.

Requirements of distributed applications:

If any application performs these tasks, it can be distributed application.
→The application need to locate the remote method.
→It need to provide the communication with the remote objects, and
→The application need to load the class definitions for the objects.

Understanding stub and skeleton:

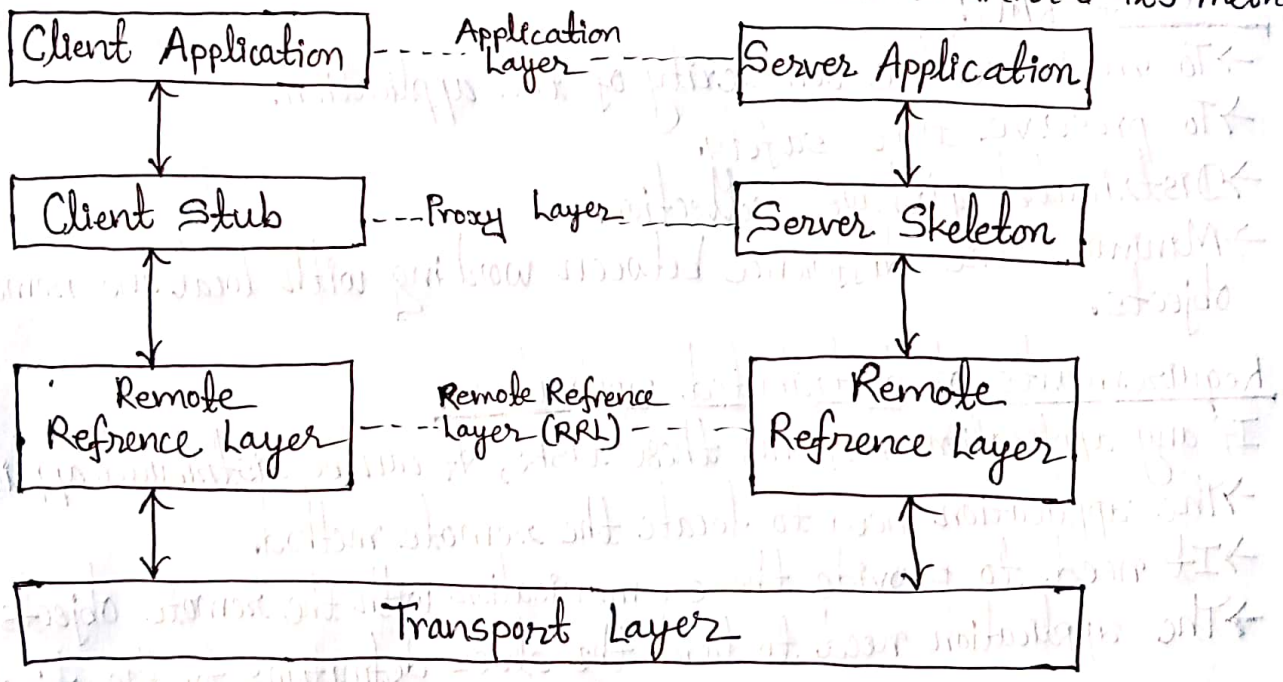
Stub: The stub is an object, that acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:
→It initiates a connection with remote Virtual Machine (JVM).
→It writes and transmits the parameters to remote Virtual Machine.
→It waits for the result.
→It reads the return value.
→ Finally returns the value to the caller.

Skeleton: The skeleton is an object, that acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

→It reads the parameter for the remote method.
→It invokes the method on the actual remote object.
→It writes and transmits the result to the caller.

⊛Architecture of RMI: [Imp]

In an RMI application, we write two programs, a server program and a client program. Inside the server program, a remote object is created and refrence of that object is made available for the client. The client program requests the remote objects on the server and tries to invoke its methods.



Application Layer: In this layer client and server are involved in communication. The java program on client side communicates with the java program on server side.

Proxy layer: This layer contains the client stub and server skeleton objects.

Stub: A stub is an object that acts as a gateway for client side. All the outgoing requests are routed through it.

Skeleton: The skeleton is an object, that acts as a gateway for server side. All the incomming requests are routed through it.

Remote Refrence Layer (RRL): This layer is responsible to maintain session during the method call. It is also responsible for handling duplicated objects.

Transport Layer: It is responsible for setting up communication between two machines. This layer uses standard TCP/IP protocol for connection.

⊛ Creating and Executing RMI Applications:

Q/ How can you use RMI to develop a program that runs in different machine? Discuss with suitable example.

OR Q/. Describe the process to run the RMI application.

Solution:

To create an RMI application in Java, we will need to do the following steps:

→ Define the remote interface
→ Implement the remote interface.
→ Create the server.
→ Create the client.

Example:

```
import java.rmi.Remote;
import java.rmi.RemoteException;
//Define the remote interface
public interface Hello extends Remote{
        String sayHello() throws RemoteException;
}
//Implement the remote interface
public class HelloImpl extends UnicastRemoteObject implements Hello{
        public HelloImpl() throws RemoteException{
                super();
        }
        public String sayHello(){
                return "Hello, World!";
        }
}
//Create the server
public class Server{
        public static void main(String[] args){
                try{ HelloImpl obj=new HelloImpl(); //Create remote object
                //Bind the remote object to the RMI registry
                Naming.rebind("//localhost/Hello", obj);
                System.out.println("Hello Server ready.");
        }catch(Exception e){
```

```
        System.out.println("Server failed:" +e);
    }
  }
}

//Create the client
  public class Client {
    public static void main (String[] args){
      try{
        //Lookup the remote object
        Hello obj = (Hello) Naming.lookup("//localhost/Hello");
        //Invoke a method on the remote object
        String message = obj.sayHello();
        System.out.println(message);
      }catch (Exception e){
        System.out.println("Hello Client exception: "+e);
      }
    }
  }
```

To compile and run, we need to include the 'rmi.jar' file in our classpath and start the RMI registry with 'rmiregistry' command. Then, we can compile the server and client classes using 'javac' and run the client and server using 'java' as follows:

```
javac Server.java HelloImpl.java
java Server

javac Client.java
java Client
```

This will cause the client to connect.

⊛ Introduction to CORBA: [Imp]

The Common Object Request Broker Architecture (CORBA) is a standard developed by the Object Management Group (OMG) to provide interoperability among distributed objects. CORBA is the world's leading middleware solution enabling the exchange of information, independent of hardware platforms, programming languages and operating systems. CORBA is often described as a "software bus" because it is a software-based communications interface through which objects are located and accessed.

⊛ RMI vs CORBA: [Imp]

RMI	CORBA
i) RMI is a Java-specific technology.	i) CORBA is independent of programming languages.
ii) It uses Java interface for implementation.	ii) It uses Interface Definition Language (IDL) to seperate interface from implementation.
iii) RMI programs can download new classes from remote JVMs.	iii) CORBA doesn't have this code sharing mechanism.
iv) RMI passes objects by remote refrence or by value.	iv) CORBA passes objects by refrence.
v) Distributed garbage collection is available.	v) Distributed garbage collection is not available.
vi) Generally simpler to use	vi) More complicated.
vii) It is free of cost	vii) Cost money according to the vendor.

⊛CORBA Architecture:

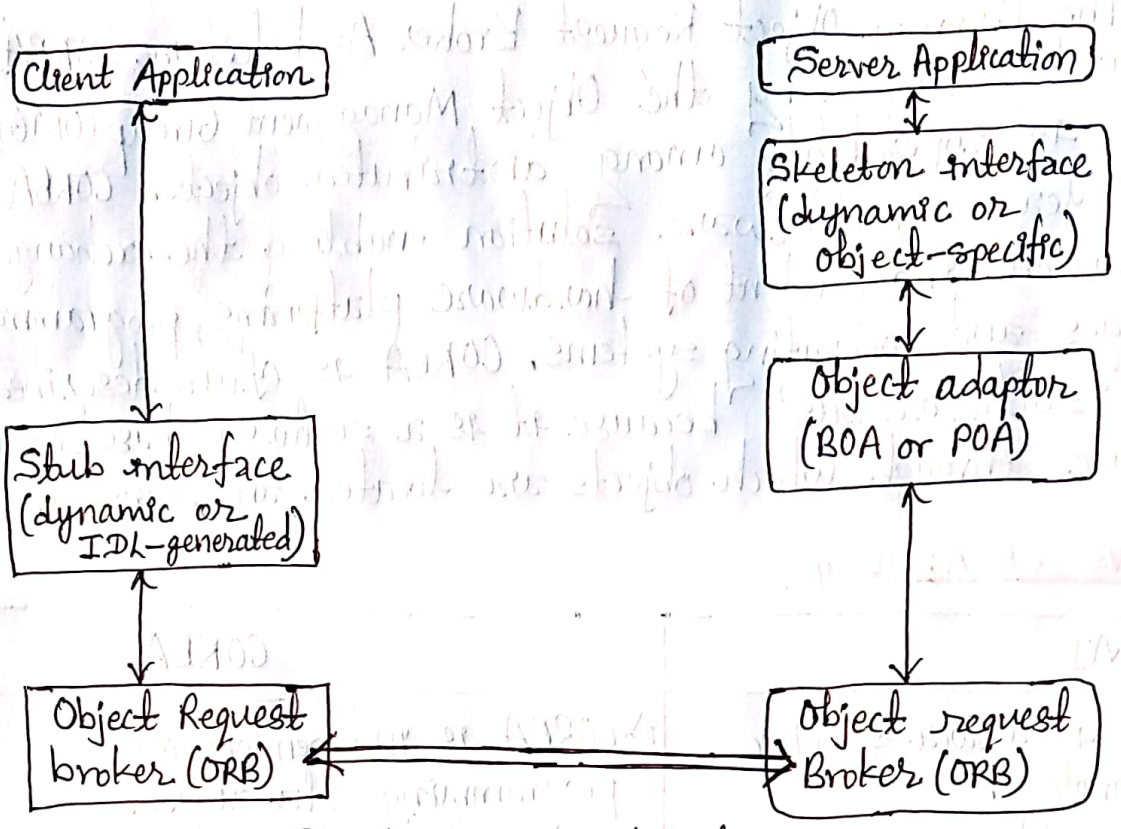


Fig: The CORBA Architecture

The CORBA architecture consists of following components:

Object Request Broker (ORB): The ORB is the core component of the CORBA architecture. It is responsible for handling communication between objects and for locating and activating remote objects.

IDL compiler: The IDL compiler is used to generate code from IDL interfaces. It generates code for the client-side and server-side stubs and skeletons, which are used to communicate with the remote object.

Client: The client is a program that invokes operations on a remote object. It uses the client-side stub to communicates with the ORB, which in-turn communicates with server-side skeleton.

Server: The server is a program that implements the operations of a remote object. It uses the server-side skeleton to communicate with the ORB, which in turn communicates with the client-side stub.

⊛ Interface Definition Language (IDL): [Imp]

The Interface Definition Language (IDL) is a language that is used to define interfaces for distributed objects in the CORBA. IDL is a language-neutral way of specifying the interface to a remote object, and it allows software components written in different programming languages to work together.

In IDL an interface is defined as a set of operations that can be invoked on a remote object. Each operation has a name, return type, and a list of parameters. Here is an example of an IDL interface that defines a single operation, 'sayHello', which takes no parameters and returns a string:

```
interface Hello{
    string sayHello();
}
```

⊛ Simple CORBA Program:

```
module HelloApp
{
    interface Hello
    {
        string sayHello("Hello");
    };
};
```

If my notes really helped you,then you can support me on esewa for my hardwork.

Esewa ID: 9806470952