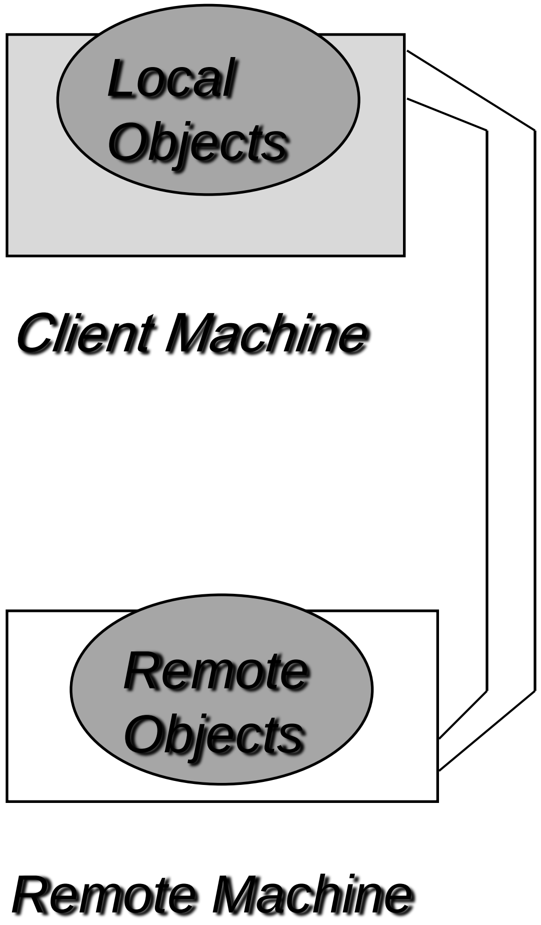
Unit 3

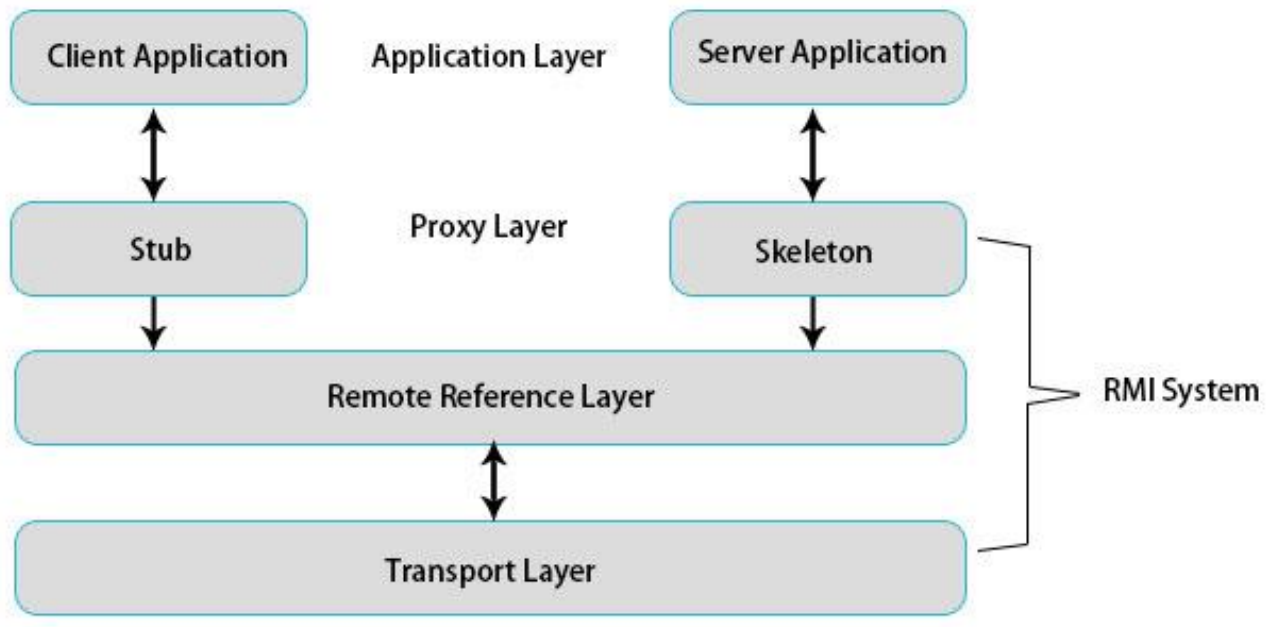
Java RMI

What is java RMI

- Remote Method Invocation (RMI) is an API which allows an object to invoke a method on an object on a remote machine.
- Object running in a JVM present on a computer (Client side) can invoke methods on an object present in another JVM (Server side).
- RMI creates a public remote server object that enables client and server side communications through simple method calls on the server object.



RMI Architecture



RMI Architecture

RMI Architecture

- Application Layer - Client and Server Program
- Proxy Layer – Stub and Skeleton
- Remote Reference Layer
- TCP
- IP
- Data Link
- Physical

RMI Architecture

- Application Layer
  - This layer is the actual systems i.e. client and server which are involved in communication.
  - The java program on the client side communicates with the java program on the server-side.
- Proxy Layer
  - Initiates connection with remote JVM,
  - Writes and transmits (Marshals) parameters to remote JVM,
  - Waits for the result,
  - Reads (Unmarshalls) the returned result,
  - Pass the received result to the caller.

RMI Architecture

- Remote Reference Layer
  - provides a RemoteRef object that represents the link to the remote service implementation object.
  - encodes and decodes the on-wire protocol
  - implements the remote object protocols
- Transport layer
  - The Transport Layer makes the connection between JVMs. All connections are stream-based network connections that use TCP/IP.
  - handles the underlying socket handling required for communications
    - sets up and maintains connections
    - communications related error handling

Working of java RMI

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called **invoke()** of the object **remoteRef**. It passes the request to the RRL on the server side.
- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

Working of java RMI

- Create the Interface to the server
- Create the Server
- Create the Client
- Compile the Interface (javac)
- Compile the Server (javac)
- Compile the Client (javac)
- Generate Stubs and Skeletons (rmic)

Deploying the Application

- Start the RMI registry
  - rmiregistry is in the JSDK bib directory
- Start the RMI Server
- Start the RMI Client

Activatable Objects

- Added in Java 2 SDK
- Standard RMI objects exported as UnicastRemoteObject must run continuously
- Activatable the object can be deactivated and reactivated remotely when a method call is made
- Must use the rmid server process to take advantage of this capability

Example :Remote interface

```
import java.rmi.*;
public interface Adder extends Remote
{
		public int add(int x,int y)throws RemoteException;
}
```

Example: Implementing interface

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements
    Adder {
AdderRemote()throws RemoteException{
super();
}
public int add(int x,int y){return x+y;}
}
```

Example: Binding to registry

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
```

Example:Client application

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```

Thank You