

UNIT II JAVA NETWORKING FUNDAMENTALS

- Overview of Java Networking
 - TCP
 - UDP
 - InetAddress and Ports
- Socket Programming
- Working with URLs
- Internet Protocols simulation
 - HTTP
 - SMTP
 - POP
 - FTP
- Remote Method Invocation
- Multithreading Concepts.

Presented by,
B.Vijayalakshmi
Computer Centre
MIT Campus
Anna University

RMI

(Remote method Invocation)

- **Remote method** → A method of a java program on any other system (not in our system) in the world
- **Invocation** → Sending Request to the other system (say, a server) to execute the method (say, to call as remote method) available on it by passing sufficient parameters for the execution of the method and asking to return the values of the method.
- RMI is communication between 2 JVMs loaded on two different systems
- **RMI is an example of distributed communication**, where systems are distributed across the globe.

RMI

(Remote method Invocation)

- **RMI is not a web based technology** so it does not use browsers and HTTP protocol
- RMI comes with its own protocol known as RMI protocol with default port number 1099
- The protocol involves,
 - RMI runtime environment
 - Remote references for Remote objects
 - RMI registry
 - Some naming service etc.
- **In RMI, objects communicate** an object on client side communicates with another object on server side. i.e. object to object communication.
- Through distributed network, objects pass in between client and server. For this purpose RMI internally serialize objects

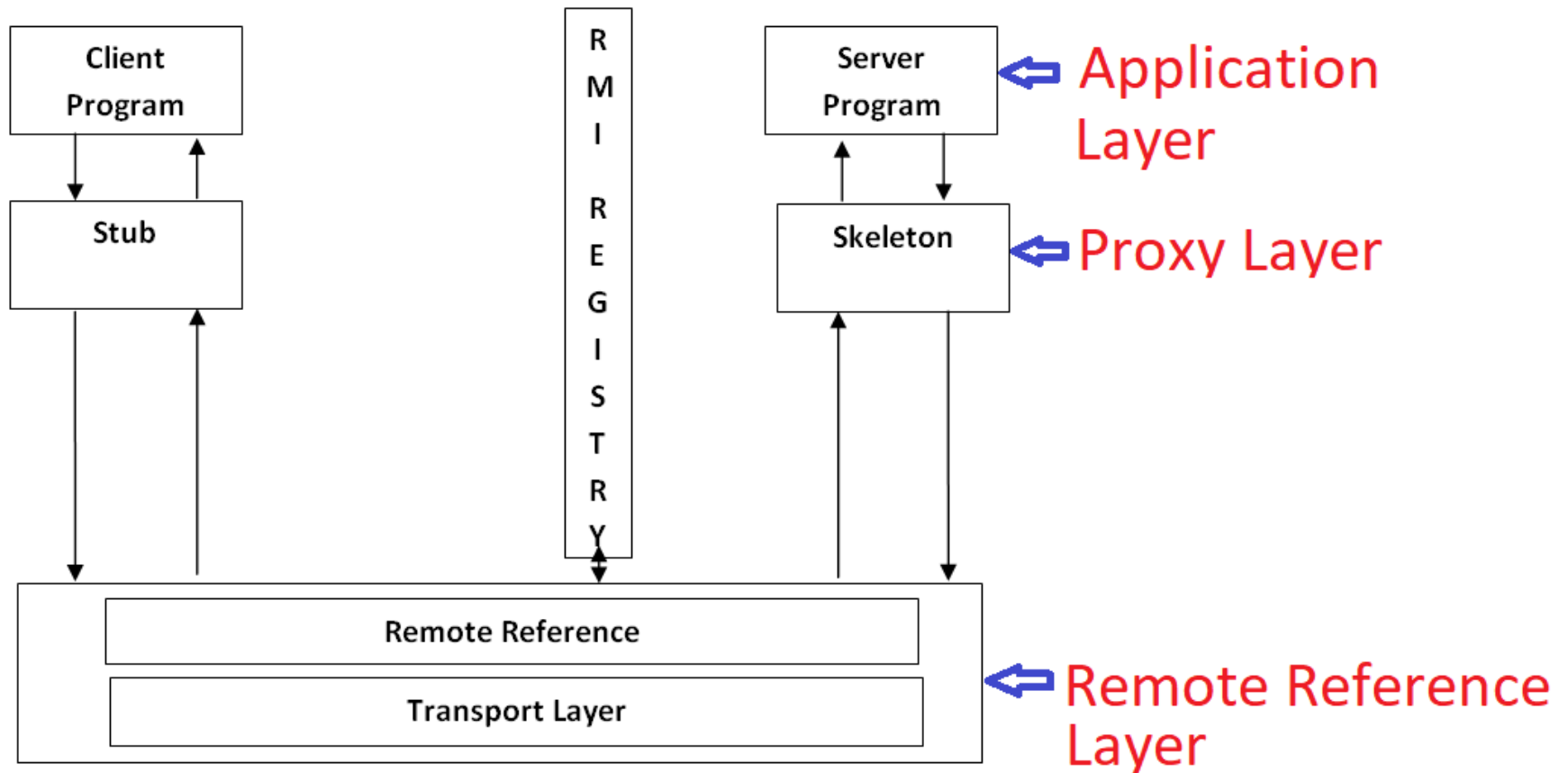
Serialization

- It is the **process converting an object into a special stream of bytes** so that they can be written to a file (or) send across distributed networks.
- It guarantee that when the object is returned back either from the file (or) network, the object maintains the same properties

Remote server, Remote program, Remote method

- The method existing on the server is a remote method to the client (because client and server may be far away, anywhere in the globe). But for the server, it is local method only.
- A program on the remote server is remote program and finally the method in the remote program is remote method

RMI Architecture



RMI Architecture Cont'd

- In RMI, the client and server do not communicate directly, instead communicates through stub and skeleton.
- Stub and Skeleton:
 - Special programs generated by RMI Compiler
 - Treated as proxies for the client and server
- Stub program resides on client side & skeleton program resides on Server
- Client sends a method call with appropriate parameters to stub, stub in turn calls the skeleton on the server
- Skeleton passes the stub request to the server to execute the remote method. The return value of the method is sent to the skeleton by the server. Skeleton sends back to stub and finally the return value reaches client through stub.
- One more job of proxies is marshaling and unmarshaling.
- Marshalling means converting data into special format suitable to pass through the distributed environment without losing object persistence.

RMI Application

- A typical RMI application includes 4 programs,
 1. **Remote interface**: Write an interface that should **extend an interface called Remote** from java.rmi package
 2. **Implementation program**: Write a concrete class which **extends UnicastRemoteObject** and **implements the above remote interface**
 3. **Server Program**: Write a server program to register the object with **RMI registry**
 4. **Client program**: Write a client program to request for object and method invocation
- For communication, the other supporting programs are stub, skeleton and RMI registry

Example Application – Interest Calculation

- This application is for Simple interest calculation
- The client sends principle amount, Number of years and rate of interest to server
- The server, calculates the interest amount to be paid to clear the complete debt

1. Remote interface

[This file is placed in both server and client]


```
import java.rmi.*;
public interface Interest extends Remote
{
    public abstract int calcInterest(int p,int n,int r) throws RemoteException;
    public abstract String calcTotal() throws RemoteException;
}
```

- Interface should extend Remote interface, therefore Interest interface is remote interface
- All the methods involved in communication should be written in the remote interface.
- It is a contract between client and server to use only the methods included in the remote interface, i.e. client should invoke only these remote methods on the server
- All the methods should throw java.rmi.RemoteException
- The methods parameters are used by the client to send data to server and the return type is used by the server to send data client.

2. Implementation Program

[This file is placed in server]

InterestImpl.java

```
import java.rmi.*;
import java.rmi.server.*;
class InterestImpl extends UnicastRemoteObject implements Interest
{
    int p,si;
     Default constructor
    public InterestImpl() throws RemoteException
    {
        super();
    }
    public int calcInterest(int p,int n,int r) throws RemoteException
    {
        this.p=p;
        si=(p*n*r)/100;
        return(si);
    }
    public String calcTotal() throws RemoteException
    {
        String str="Pay Rs."+(p+si)+" to clear the dept completely";
        return(str);
    }
}
```

Implementation Program Cont'd

- Implementation file **should extend an abstract class called `java.rmi.server.UnicastRemoteObject` & implements the `Interest` interface**
- **`UnicastRemoteObject` makes the server object exportable to the client system** through remote reference layer (here, client gets a reference of remote object of server (or) to say, server passes the remote object by reference (pass by reference) to client.
- **Default constructor should be included**
- All the methods and constructors should throw `java.rmi.RemoteException`

3. Server Program

[This file is placed in server]

[This program creates a remote object, links with an alias name & binds with the RMI registry, linked to RMI runtime mechanism]

```
import java.rmi.*;           InterestServer.java
public class InterestServer
{
    public static void main(String a[]) throws Exception
    {
        Interest i1=new InterestImpl();
        Naming.rebind("simple",i1);
        System.out.println("Server is ready for remote invocation by client");
    }
}
```

Server Program Cont'd

- The Remote interface object `i1` is returned by the implementation program constructor, `new InterestImpl()`
- **`i1` is known as remote object in RMI technology.**
- RMI uses a naming service where an alias name is maintained and this alias name should be known to client to invoke the remote method
- The naming service is represented by a class called `java.rmi.Naming`
- The `rebind()` method of `Naming` class binds the object `i1` along alias name with the RMI registry.
- Alias name “simple” refers the object `i1` on server side object `i1`.
- RMI registry is connected internally with the Remote Reference Layer

4. Client Program

[This file is placed in Client]

```
import java.rmi.*;
import java.rmi.registry.*;
public class InterestClient
{
    public static void main(String a[]) throws Exception
    {
        Registry reg = LocateRegistry.getRegistry("192.168.1.7", 1099);
        Interest i2 = (Interest) reg.lookup("simple");
        //Object obj1=Naming.lookup("simple");
        //Interest i2=(Interest)obj1;
        int intAmt1=i2.calcInterest(5000,2,12);
        System.out.println("Simple interest:"+intAmt1);
        String str1=i2.calcTotal();
        System.out.println(str1);
        int intAmt2=i2.calcInterest(6000,3,10);
        System.out.println("Simple interest:"+intAmt2);
        String str2=i2.calcTotal();
        System.out.println(str2);
    }
}
```

InterestClient.java

[This program obtains a reference of the remote object of server and invokes remote methods]


Client Program Cont'd

- The variable i2 on client side is reference object that refers i1 on server side.
 - i1 on the server is known as remote object
 - i2 refers the remote object
- Every method called with i2.calcInterest(5000,2,12) become i1.calcInterest(500,2,12) on the server and return the value to local variable intAmt1
- Every operation done on i2 on client will be executed on the server with object i1
- i2 is reference of i1
- i2 and i1 resides on 2 different systems. i2 and i1 communicates each other is known as object communication

Compilation and Execution procedure

1) Compile all 4 programs with javac command.

javac Interest*.java

2) Generate stub & skeleton  **rmic InterestImpl**
copy InterestImpl_Stub class at client side.

3) **start rmiregistry**

4) From server prompt run Server program

java InterestServer

5) Run Client program in client prompt

java InterestClient

Warning: rmic has been deprecated and is subject to removal in a future release. Generation and use of skeletons and static stubs for JRMP is deprecated. Skeletons are unnecessary, and static stubs have been superseded by dynamically generated stubs. Users are encouraged to migrate away from using this tool to generate skeletons and static stubs. See the documentation for `java.rmi.server.UnicastRemoteObject`.

RMI registry



```
Command Prompt - java InterestServer

G:\JAVA_PGMS>javac Interest*.java

G:\JAVA_PGMS>start rmiregistry

G:\JAVA_PGMS>java InterestServer
Server is ready for remote invocation by client
```

RMI Server

RMI Client

```
Command Prompt

F:\VIJJI\JAVA>javac Interest*.java

F:\VIJJI\JAVA>java InterestClient
Simple interest:1200
Pay Rs.6200 to clear the dept completely
Simple interest:1800
Pay Rs.7800 to clear the dept completely

F:\VIJJI\JAVA>
F:\VIJJI\JAVA>
```