**JRMP** Java Remote Method Protocol, server and client are running on JVM.

**RMI-IIOP** supporting non-JVM context.

**REMOTE OBJECT** is a Java object whose methods can be called remotely. Its class definition must implement `Remote` interface, which hides the low level mechanism of communication from programmer.

**SERVER** is where remote object created.

**CLIENT** tries to access remote object.

**STUB** a local reference to a non-local object (remote object). But it contains more information than a reference type, etc. Contact information. It is a bridge between server and client.

**REMOTE OBJECT REGISTRY** is a container object on a host, it stores exported remote objects and handles requests from clients.
**Execution Model**

**Server Side:**
1. create remote object
2. export it to local registry
3. waiting

**Client Side**
1. connect to a remote registry
2. lookup remote object by its string reference
3. bind it to stub
4. invoke method of remote object
**Passing Remote Object**  When invoking a remote method with exported remote object as parameter or return value, their stubs are passed.

**Passing non-Remote Object**  Unlike local JVM, passing non-remote object through remote method is by value rather than by reference, since there is no way to access non-remote object by its reference remotely.

**Referential Integrity**  When more references to the same object are passed through remote method, only one object is passed.
Java RMI Specification
http://java.sun.com/j2se/1.5.0/docs/guide/rmi/spec/rmiTOC.html

API
http://java.sun.com/j2se/1.5.0/docs/api/

RMI Tutorial from Sun
http://java.sun.com/docs/books/tutorial/rmi/overview.html
// RMIClient.java
import java.rmi.*;
import java.rmi.server.*;
public class RMIClient{
    public static void main(String argv[]){
        String strName = "TheRMIExample";
        RMIExample RemRMIExample = null;
        try {
            RemRMIExample = (RMIExample)Naming.lookup(strName);
        } catch (Exception e) { System.out.println(e); }
        try {
            long lRes = RemRMIExample.Factorial(Long.parseLong(argv[1]));
        } catch (Exception e) { System.out.println(e); } } }

//RMIExample.java
import java.rmi.*;
public interface RMIExample extends Remote{
    public long Factorial(long lVal) throws RemoteException;}

Remote Method Invocation in Java
// RMIExampleImpl.java
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.io.*;
public class RMIExampleImpl extends UnicastRemoteObject
  implements RMIExample{
    protected static String m_strName;
    public RMIExampleImpl() throws RemoteException{
      super();
    }
    public long Factorial(long lVal) throws RemoteException {
      long lRes = FactorialEx(lVal);
      System.out.println(lRes);
      return lRes;
    }
    protected long FactorialEx(long lVal) {
      if (lVal <= 1) return 1;
      else return lVal * FactorialEx(lVal-1); }
    public static void main(String argv[]) {
      try {
        RMIExampleImpl Example = new RMIExampleImpl();
        Naming.rebind("TheRMIExample", Example);
      } catch (Exception e) {
        System.out.println(e); }
    }
}
Remote Method Invocation in Java

RMIEexampleImplStubs: RMIEexampleImpl
rmic RMIEexampleImpl

RMIEexampleImpl: RMIEexample RMIClient
javac RMIEexampleImpl.java

RMIEexample: RMIEexample.java
javac RMIEexample.java

RMIClient: RMIClient.java
javac RMIClient.java
Modify the example, implement a message board system. You can login the system and post message. The system prints all messages on its terminal with names.