Remote Method Invocation

Goal of RMI

Implement distributed objects.

Have a program running on one machine invoke a method belonging to an object whose execution is performed on another machine.

Remote Object Technologies

CORBA

- Common Object Request Broker Architecture
- Designed for interoperability between different languages as well as different machines.

DCOM or COM

- Distributed Component Object Model
- Interoperability between different machines and languages as long as they are Wintel.

RMI

- Java only.
- Interoperability for any machines that are running the Java virtual machine.

Remote versus Local Objects

Although we would like remote objects to behave exactly the same as local object, that is impossible for several reasons:

- Networks can be unreliable.
- References on one machine (memory addresses) have no meaning on another machine.

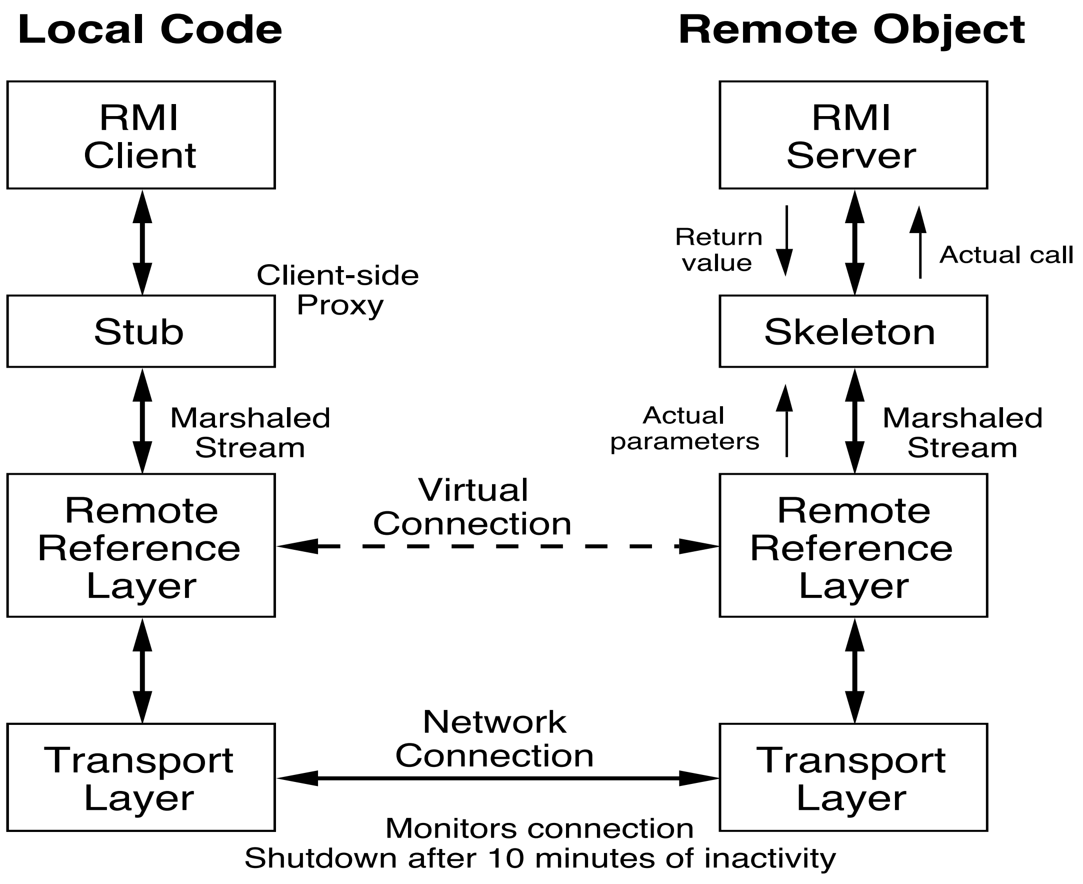
Consequences

- Special exceptions will indicate network failures.
- Objects passed as parameters and returned as results are passed by copy mode (pass by value).

Reasons for Using RMI

- Some operations can be executed significantly faster on the remote system than the local client computer.
- Specialized data or operations are available on the remote system that cannot be replicated easily on the local system, for example, database access.
- Simpler than programming our own TCP sockets and protocols.

Overview of an RMI System



Recipe for RMI

1. Create and compile an interface that specifies the methods that will have remote access.
   - Must be public
   - Must extend the interface java.rmi.Remote
   - Each remote method must have a ***throws*** *java.rmi.RemoteException* clause
   - Interface must reside on both the server and the client side
   - Any method parameters or return value of a reference type must implement Serializable.

2. Write and compile a class that implements the remote interface in 1.
   - Must extend java.rmi.server.UnicastRemoteObject
   - Must implement the remote interface in 1
   - Its constructors must be defined explicitly since they each may throw java.rmi.RemoteException
   - Resides on the server side only

3. Create the stub class using the rmic tool.

   % **rmic -v1.2 ImplementationClass**

   - Leave the stub on the server side
   - Put a copy of the stub class on the client side

4. Write and compile a server class that
   - Installs a security manager using

```
System.setSecurityManager(
        new java.rmi.RMISecurityManager());
```

This step may be optional.

- Instantiates an object of the implementation class in 2; this is the remote object
- Binds the remote object *implObj* to a unique identifier (a String) for the rmi registry

```
java.rmi.Naming.rebind("uniqueID", implObj);
```

Note: The implementation class and the server class can be combined into one class, particularly if there is no need for the server class to extend another class.

5. Write and compile a client class that

- Installs a security manager as in 4 (this step may be optional also).
- Requests an object from the remote server using its hostname and the unique identifier of the object, and then casts that object to the interface type from 1.

```
InterfaceType it =
        (InterfaceType)java.rmi.Naming.lookup(
                "rmi://r-lnx233.cs.uiowa.edu:1099/uniqueID");
```

1099 is the default port and if used, its specification here can be omitted.

- If the client is running on the same machine as the remote object (the server), use

```
    InterfaceType it =
        (InterfaceType)java.rmi.Naming.lookup(
                                     "rmi://localhost/uniqueID");
```
or
```
    InterfaceType it =
      (InterfaceType)java.rmi.Naming.lookup("rmi:///uniqueID");
```

- Now methods on the remote object can be called using the interface class object *it* when the rmi system is up and running.

6. Start the bootstrap rmi registry in the background on the server side.

```
% rmiregistry &               // assumes port 1099
% rmiregistry 1492 &          // specifies a port
```

7. Execute the server class on the machine that the client named, which was r-lnx233.

```
% java ServerClass
```

Note: The rmi registry and the server must run on the same machine and in the same directory.

8. Execute the client class on another machine

```
% java ClientClass
```

Note: Any objects that will be passed as parameters to the remote object method or returned as its result must be Serializable.

Class Naming Convention

Employing a uniform pattern of names can help simplify the use of RMI:

	Name	**File**
Interface:	TryRmi	TryRmi.java
Implementation:	TryRmiImpl	TryRmiImpl.java
Server	TryRmiServer	TryRmiServer.java
Client	TryRmiClient	TryRmiClient.java

Server Side Classes

TryRmi.class	
TryRmiImpl.class	
TryRmiServer.class	
TryRmiImpl_Stub.class	the stub

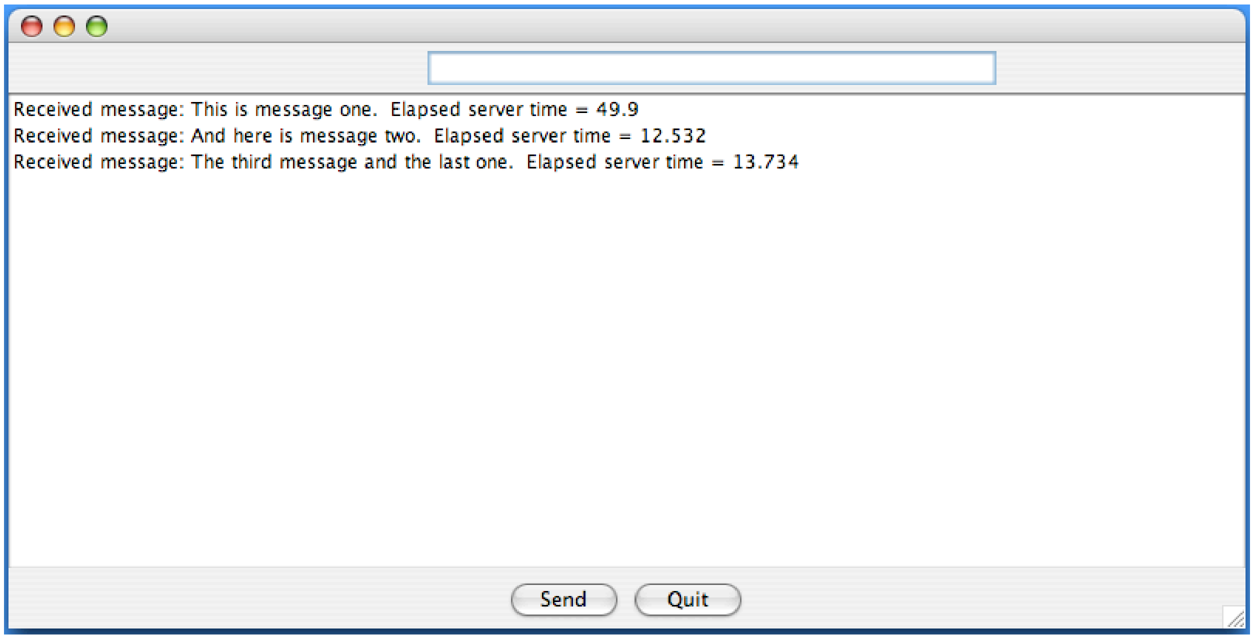
Client Side Classes

TryRmi.class	
TryRmiClient.class	
TryRmiImpl_Stub.class	the stub

Example

A server receives a String message from a client and returns a String containing the message and an indication of the time elapsed since the last message it handled.

The client provides a window in which a user can type a message to send to the server and a text area where the Strings returned from the server are displayed.



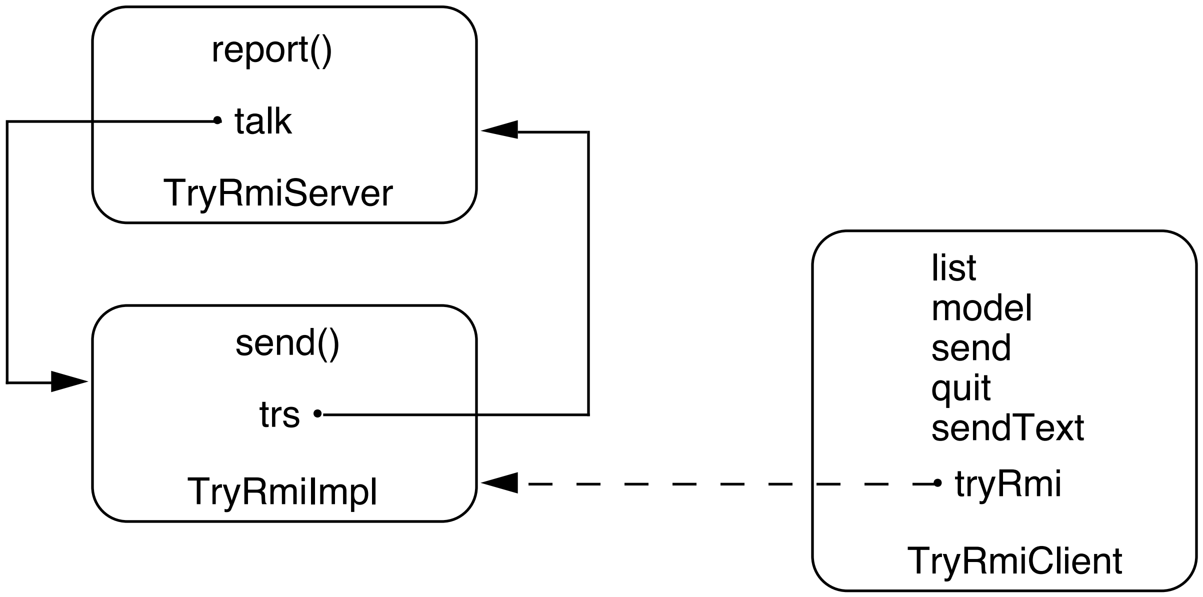
```
Received message: This is message one.  Elapsed server time = 49.9
Received message: And here is message two.  Elapsed server time = 12.532
Received message: The third message and the last one.  Elapsed server time = 13.734
```

The Code

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

public interface TryRmi extends Remote
{
    String send(String s) throws RemoteException;
}

//------------------------------------------------------------

import java.rmi.*;                   // TryRmiImpl.java
import java.rmi.server.UnicastRemoteObject;

public class TryRmiImpl extends UnicastRemoteObject
                                        implements TryRmi
{
    private TryRmiServer trs;
    private long startTime;
    TryRmiImpl(TryRmiServer r) throws RemoteException
    {
        trs = r;
        startTime = System.currentTimeMillis();
    }

    public String send(String message)
                                        throws  RemoteException
    {
        trs.report(message);
        long now =  System.currentTimeMillis();
        double time = (now - startTime)/1000.0;
        startTime = now;
        return ("Received message: " + message
                    + "  Elapsed server time = " + time);
    }
}

//--------------------------------------------------------------

import java.rmi.*; // TryRmiServer.java
import java.rmi.server.*;
import java.net.InetAddress;
import java.rmi.registry.*;           // Registry, LocateRegistry

public class TryRmiServer
{
    TryRmiServer()
    {
        System.setSecurityManager(
                                    new RMISecurityManager());
        try
        {   TryRmi talk = new TryRmiImpl(this);
            Naming.rebind("TryRmi", talk);

//---------------------------------------------------------------
//          Registry reg =
//                  LocateRegistry.createRegistry(5050);
//          reg.bind("TryRmi", talk);
//---------------------------------------------------------------

            System.out.println("Server ready on "
               + InetAddress.getLocalHost().getHostName());
            System.out.println("Stop server using control-c");
        }
        catch (Exception e)
        {  System.out.println(e); }
    }

    void report(String text)
    {
        System.out.println("Message from client: " + text);}

    public static void main(String [] args)
    {
        new TryRmiServer();
    }
}

//------------------------------------------------------------------
```

```
import java.rmi.*;                    // TryRmiClient.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TryRmiClient extends JFrame
                                        implements ActionListener
{
    private DefaultListModel model;
    private JButton send, quit;
    private JTextField sendText;
    private TryRmi tryRmi;

    private String url ="rmi://r-lnx233.cs.uiowa.edu:5050/TryRmi";

    TryRmiClient()
    {
        System.setSecurityManager(
                                        new RMISecurityManager());
        Container cp = getContentPane();
        JPanel p = new JPanel();
        p.add(new JLabel("Enter message: "));
        p.add(sendText = new JTextField(30));
        cp.add(p, "North");

        model = new DefaultListModel();
        JList list = new JList(model);
        JScrollPane jsp = new JScrollPane(list);
        cp.add(jsp, " Center ");

        p = new JPanel();
        p.add(send = new JButton("Send"));
        send.addActionListener(this);
        p.add(quit = new JButton("Quit"));
        quit.addActionListener(this);
        cp.add(p, "South");
        setBounds(100, 100, 800, 400);
        setVisible(true);
        try
        {    tryRmi = (TryRmi)Naming.lookup(url);
        }
        catch (Exception e)
        { System.out.println("Naming Exception"+e); }
    }

    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == quit) System.exit(0);
        if (e.getSource() == send) sendClicked();
    }
    private void sendClicked()
    {
        String message = "";
        String text = sendText.getText();
        sendText.setText("");
        sendText.requestFocus();
        try
        {    message = tryRmi.send(text);   }
        catch (Exception e)
        {  System.out.println(e); }
```

```
            model.addElement(message);
        }

        public static void main(String [] args)
        {
            TryRmiClient trc = new TryRmiClient();
            trc.sendText.requestFocus();
        }
}
```



Running TryRmi

```
% javac TryRmi.java
% javac TryRmiImpl.java
% javac TryRmiServer.java
% javac TryRmiClient.java
% rmic -v1.2 TryRmiImpl
```

Put server classes on server machine, whose name (r-lnx233.cs.uiowa.edu) is hard-coded into the client.

Put client classes on client machine.

```
% rmiregistry 5050 &      // on r-lnx233.cs.uiowa.edu
% java TryRmiServer       // on r-lnx233.cs.uiowa.edu
% java TryRmiClient       // on any machine
```

Alternative to rmiregistry (in TryRmiServer)

```
import java.rmi.registry.*;
//----------------------------------------------------------
    Registry reg = LocateRegistry.createRegistry(5050);
    reg.bind("TryRmi", talk);
//----------------------------------------------------------
```

Parameter Passing

Compare passing an object to an instance method for a local object with passing it to a remote object.

The object to be passed will implement an interface so that the example can be developed in a remote context.
The methods in the interface all throw RemoteException eventually for the same reason.

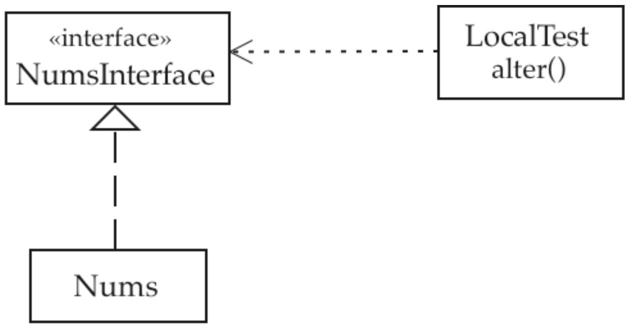
```
                                          // NumsInterface.java
public interface NumsInterface
{
    String mkStr();
    int getM();
    int getN();
    void setM(int k);
    void setN(int k);
}
```

The class that implements NumsInterface will provide the objects that are passed as parameters.

```
                                    // Nums.java
public class Nums implements NumsInterface
{
   private int m, n;

   Nums(int k1, int k2)
   {   m = k1;  n = k2;   }

   public String mkStr()
   {
      return "Nums[m = " + m + ", n = " + n + "]";
   }

   public int getM()
   {  return m;  }

   public int getN()
   {  return n;  }

   public void setM(int k)
   {  m = k;  }

   public void setN(int k)
   {  n = k;  }
}
```

Calling a Method on a Local Object

```
import java.io.*;              // LocalTest.java

public class LocalTest
{
    public static void main(String [] args) throws Exception
    {
        Nums nums = new Nums(5, 13);
        LocalTest lt = new LocalTest();
        System.out.println("nums = " + nums.mkStr());
        lt.alter(nums);  // pass object to local object
        System.out.println("---------------------------------");
        System.out.println("nums = " + nums.mkStr());
    }
    void alter(NumsInterface nm) throws Exception
    {
            nm.setM(2*nm.getM());
            nm.setN(3*nm.getN());
    }
}
```



```
/**************************************************
nums = Nums[m = 5, n = 13]
-------------------------------------
nums = Nums[m = 10, n = 39]
**************************************************/
```

Calling a Method on a Remote Object

Now put the *alter* method into a remote object.

Need an interface for the remote object:

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

public interface Remo extends Remote
{
      void alter(NumsInterface nm) throws RemoteException;
}
```

Then implement the interface with a class that provides the remote object and acts as a server, creating an instance of the remote object and registering it.

```
import java.rmi.*;                       // RemoImpl.java
import java.rmi.server.UnicastRemoteObject;

public class RemoImpl extends UnicastRemoteObject
                                         implements Remo
{
   public void alter(NumsInterface nm)
                                    throws RemoteException
   {
         nm.setM(2*nm.getM());
         nm.setN(3*nm.getN());
   }

   RemoImpl() throws RemoteException
   {
      System.out.println("Initializing Remote Test");
   }
/*********************************************************/
```

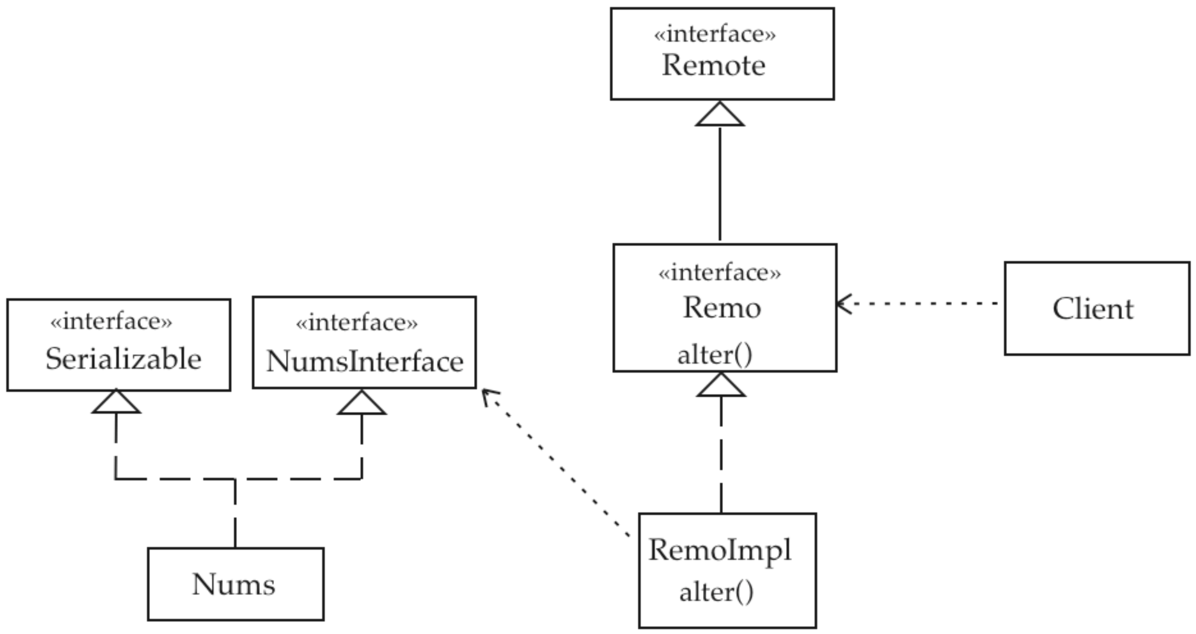
```
  public static void main(String [] args)
  {                                          // Acts as server also
    System.setSecurityManager(new RMISecurityManager());
    try
    {
      Remo ri = new RemoImpl();
      Naming.rebind("remoteServer", ri);
      System.out.println("Registered with registry");
    }
    catch (RemoteException e)
    { System.out.println("Remote Error: " + e);
    }
    catch (java.net.MalformedURLException e)
    { System.out.println("URL Error: " + e);
    }
  }
}
```

We need a client that creates a Nums object and calls *alter* on the remote object just created, passing the Nums object.

Nums must implement Serializable now.

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

public class Client
{
  public static void main(String [] args)
  {
    System.setSecurityManager(
                             new RMISecurityManager());
    try
    {
      NumsInterface nums = new Nums(5, 13);
      System.out.println("nums = " + nums.mkStr());
```

```
         Remo remo = (Remo)Naming.lookup(
                                  "rmi://localhost/remoteServer");
         remo.alter(nums);
         System.out.println("------------------------------");
         System.out.println("nums = " + nums.mkStr());
         System.out.println("Done");
      }
      catch (NotBoundException e)
      {  System.out.println("Server not found");  }
      catch (RemoteException e)
      {  System.out.println("Remote error: " + e);  }
      catch (java.net.MalformedURLException e)
      {  System.out.println("URL error: " + e);  }
   }
}
```



```
/**********************************************
% rmiregistry &
[1] 14550
% java RemoImpl &
[2] 14558
Initializing Remote Test
Registered with registry
% java Client
nums = Nums[m = 5, n = 13]
------------------------------
nums = Nums[m = 5, n = 13]
Done
**********************************************/
```

Nums as a Remote Object

Make NumsInterface into a remote interface called RNumsInterface.

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

public interface RNumsInterface extends Remote
{
    String mkStr() throws RemoteException;
    int getM() throws RemoteException;
    int getN() throws RemoteException;
    void setM(int k) throws RemoteException;
    void setN(int k) throws RemoteException;
}
```

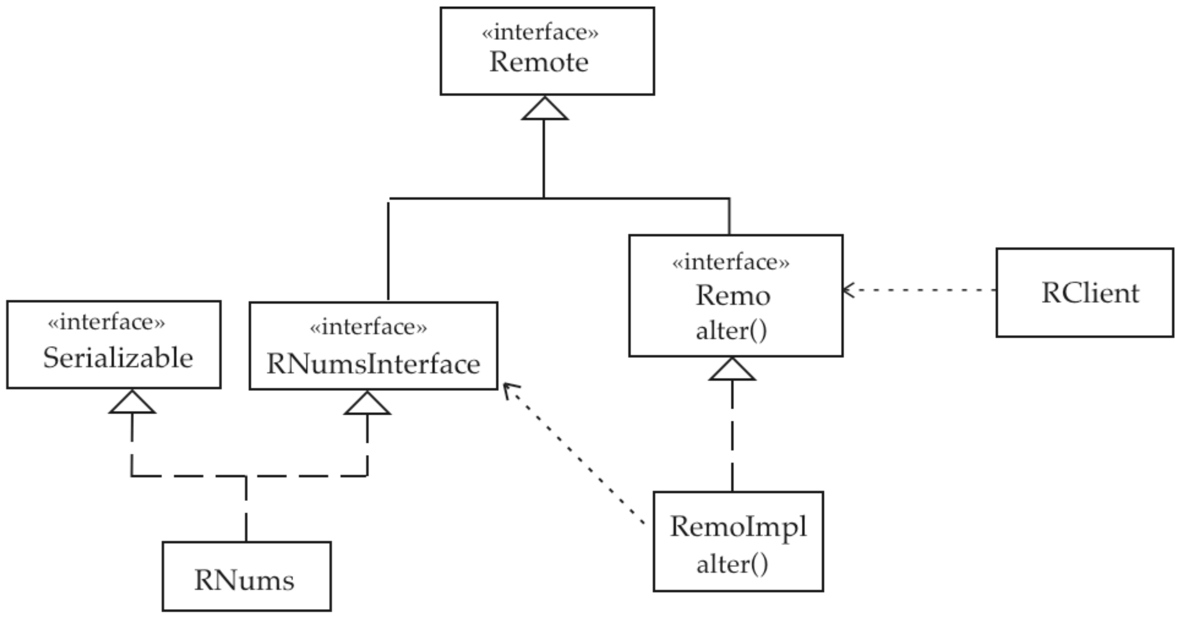
Change Nums into RNums, a subclass of UnicastRemoteObject.

```
import java.rmi.*;                    // RNums.java
import java.io.*;
import java.rmi.server.UnicastRemoteObject;

public class RNums extends UnicastRemoteObject
                    implements Serializable, RNumsInterface
{
   private int m, n;

   RNums(int k1, int k2) throws RemoteException
   {   m = k1; n = k2;  }

   public String mkStr() throws RemoteException
   {  return "Nums[m = " + m + ", n = " + n + "]";  }

   public int getM() throws RemoteException
   {  return m;  }

   public int getN() throws RemoteException
   {  return n;  }

   public void setM(int k) throws RemoteException
   {  m = k;  }

   public void setN(int k) throws RemoteException
   {  n = k;  }
}
```

Change the type of the parameter of *alter* to RNumsInterface in Remo and RemoImpl.

Change Client into RClient, which creates an RNums object to pass to alter.

```
import java.rmi.*;                    // RClient.java
public class RClient
{
   public static void main(String [] args)
   {
      System.setSecurityManager(new RMISecurityManager());
      try
      {
         NumsInterface nums = new RNums(5, 13);
         System.out.println("nums = " + nums.mkStr());
         Remo remo = (Remo)Naming.lookup(
                                  "rmi://localhost/remoteServer");
         remo.alter(nums);
         System.out.println("------------------------------");
         System.out.println("nums = " + nums.mkStr());
         System.out.println("Done");
      }
      catch (NotBoundException e)
      {  System.out.println("Server not found");  }
      catch (RemoteException e)
      {  System.out.println("Remote error: " + e);  }
      catch (java.net.MalformedURLException e)
      {  System.out.println("URL error: " + e);  }
   }
}
```



```
/*************************************************
% rmiregistry &
[1] 14550
% java RemoImpl &
[2] 1558
Initializing Remote Test
Registered with registry
% java RClient
nums = Nums[m = 5, n = 13]
--------------------------------
nums = Nums[m = 10, n = 39]
Done
*************************************************/
```

Client side	**Server side**
1. Client	RemoImpl implements Remo (Remote)
remo ------------------------->	alter(NumsInterface)
2. RClient	RemoImpl implements Remo (Remote)
remo ------------------------->	alter(RNumsInterface)

Calls remo.alter(nums)

where

1. nums = **new** Nums implements NumsInterface

2. nums = **new** RNums
implements RNumsInterface (Remote)

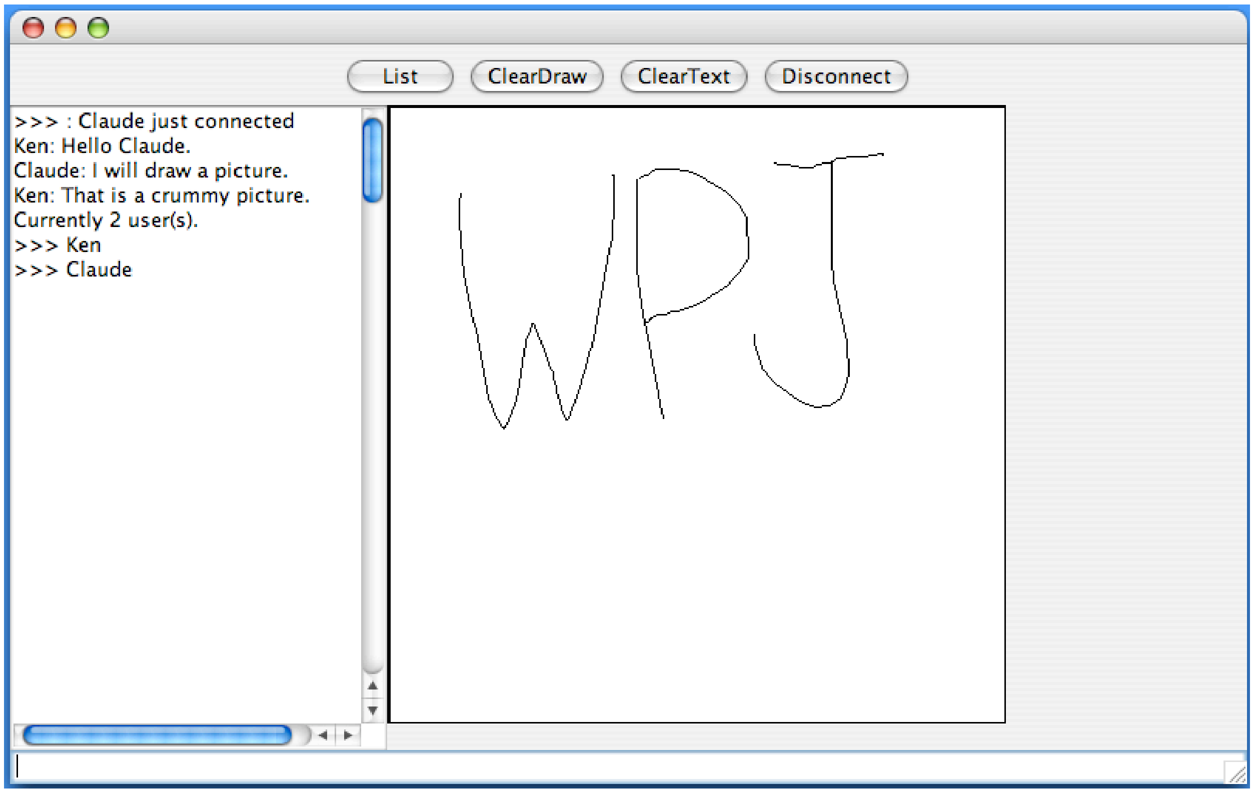
client files

RNumsInterface.class	RNums.class
RClient.class	
RNums_Stub.class	
Remo.class	RemoImpl_Stub.class

server files

Remo.class	RemoImpl.class
RemoImpl_Stub.class	
RNumsInterface.class	
RNums_Stub.class	

A Chat System with a Whiteboard



```
>>> : Claude just connected
Ken: Hello Claude.
Claude: I will draw a picture.
Ken: That is a crummy picture.
Currently 2 user(s).
>>> Ken
>>> Claude
```

Server running on r-lnx233.cs.uiowa.edu.

One client (Ken) running on cornfed.cs.uiowa.edu (a Mac).

Another client (Claude) running on r-lnx233.cs.uiowa.edu.

Note: I had trouble with getting the JDialog to show on *cornfed*, so I hard-coded the name of the user on that machine.

Actions

A client obtains a reference to a remote object (ChatServer implemented by ChatServerImpl) that provides three remote methods:

register

postMessage

listChatters

When a client registers with the server, it passes a remote reference to itself (ChatClient implemented by ChatClientImpl) that provides one method: *chatNotify*.

Note that the remote reference passed to *register* must be of the interface type, ChatClient, not of the object type ChatClientImpl.

Therefore, the server has a remote reference to each of its clients so that it can notify each, using *chatNotify*, when a new message has arrived.

The *chatNotify* method acts as a callback function that the server uses to pass messages to the clients.

Since clients will be notified of messages that are broadcast, they do not have to poll the server but just wait until the server notifies them of a new message.

On the client side, each client has a thread, an instance of ServerTalker, that maintains a queue of messages to be set to the server using *postMessage*, one of the methods that the server remote object recognizes.

As long as the message queue is not empty, the messages are removed and sent to the server.

The threads ensure that no client is delayed waiting for a remote method invocation on the server to complete.

Each client can be attentive to user input.

On the server side, the server also has a thread for each client, an instance of ClientTalker, so that a slow client will not delay the system.

Each ClientTalker object maintains a queue of messages to be sent to its client using the callback method *chatNotify*, the one method that the client remote object recognizes.

When a client calls *postMessage*, that method calls a boolean method *addMessage* in each ClientTalker to add the message to its queue.

This method returns *false* if its client is no longer connected.

The ChatServerImpl object starts the registry in its main method using

```
      Registry reg = LocateRegistry.createRegistry(5050);
```

before binding itself to the unique identifier “ChatServerImpl”.

Message Objects

- The class Message implements Serializable so that its objects can be sent as parameters from a client to the server and then back to each of the clients.
- Message objects include the name of the client who initiated the message.
- Message objects may encapsulate a String or a Line.

Since ChatClientImpl extends JFrame, it cannot be a subclass of UnicastRemoteObject.

Therefore, the remote interface must be exported manually using the command:

```
UnicastRemoteObject.exportObject(this);
```

Once a client is registered with the server, it remains idle until either

- A user event (text or a line) causes a message to be sent to the server by ServerTalker or a button is pressed.

or

- Its method *chatNotify* is called by the server, passing a message back.

When a message comes back by means of *chatNotify*, either

- Text data is appended to the TextArea

or

- A Line object is added to an off-screen buffer, which is shown when the panel of the whiteboard is redrawn.

When the ServerTalker has emptied its queue of messages, the client becomes idle again.

The Whiteboard

When the mouse is dragged on the panel that makes up the whiteboard, lines are created and placed in an internal buffer, a List, and then the *render* method is called.

Those lines will be drawn on an off-screen image buffer and sent as messages to the server when the *render* method is executed.

The offscreen image is drawn on the panel inside *paintComponent*, provided the image exists.

The Code

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

public interface ChatServer extends Remote
{
    void register(ChatClient c,String name)
                              throws RemoteException;

    void postMessage(Message m)
                          throws RemoteException;

    String [] listChatters() throws RemoteException;
}

//------------------------------------------------------------

import java.rmi.*;           // ChatServerImpl.java
import java.rmi.server.*;
import java.rmi.registry.*;
import java.util.*;

public class ChatServerImpl extends UnicastRemoteObject
                                  implements ChatServer
{
  private List<ClientTalker> chatters =
                              new ArrayList<ClientTalker>();

  ChatServerImpl() throws RemoteException
  {
    System.out.println("Initializing Server...");
  }

  public static void main(String [] args)
  {
    System.setSecurityManager(new RMISecurityManager());
    try
    {
       ChatServer csi = new ChatServerImpl();

       Registry reg =
                LocateRegistry.createRegistry(5050);

       reg.rebind("ChatServerImpl", csi);

       System.out.println("Server Ready");
    }
    catch (RemoteException e)
    {  System.out.println("ChatServer Error: " + e);  }
 }

 public synchronized void register(ChatClient c, String n)
                                  throws RemoteException;
 {
    chatters.add(new ClientTalker(c, n));

    System.out.println(n + " added to chat group.");
 }

// Note remote object parameter of type ChatClient.

 public synchronized void postMessage(Message m)
                                  throws RemoteException;
 {
    for (Iterator it=chatters.iterator(); it.hasNext;  )
    {
       ClientTalker ct = (ClientTalker)it.next();
       boolean alive = ct.addMessage(m);
       if (!alive) it.remove();
    }
 }

 public String [] listChatters()throws RemoteException;
 {
       String [] list = new String[chatters.size()];
```

```
      for (int k=0; k< list.length; k++)
      {
         ClientTalker ct = chatters.get(k);
         list[k] = ct.getChatterName();
      }
      return list;
   }

//------------------------------------------------------------

import java.util.*;     // ClientTalker.java
import java.rmi.*;

public class ClientTalker extends Thread
{
   private List<Message> messages =
                                    new LinkedList<Message>();
   private ChatClient chat;
   boolean isActive = true;
   private String name;

   ClientTalker(ChatClient c, String n)
   {  chat = c;
      name = n;
      start();
   }

   synchronized boolean addMessage(Message e)
   {
      if (!isActive) return false;
      messages.add(e);
      notifyAll();
      return true;
   }
   public synchronized void run()
   {
      boolean running = true;
      while (running)
      {  try
         {  if (messages.isEmpty()) wait();
            chat.chatNotify((Message)messages.get(0));
            messages.remove(0);
         }
         catch (InterruptedException e)
         {  System.out.println("Interrupted: " + e);  }
         catch (RemoteException e)
         {
            System.out.println("Removing "+ name);
            isActive=false;
            running = false;
         }
         yield();
      }
   }

   String getChatterName()
   {  return name;  }
}

//-----------------------------------------------------------------

import java.io.*;              // Message.java

public class Message implements Serializable
{
   private String sender = null;
   private String text = null;
   private Line line = null;
   Message(String s, Line ln)
   {  sender = s;
      line = ln;
   }

   Message(String s, String m)
   {  sender = s;
      text = m;
   }

   String getSender()
   {  return sender;  }

   String getText()
   {  return text;  }

   Line getLine()
   {  return line;  }

   public String toString()
   {  return sender + ":" + text;  }
}

//------------------------------------------------------------

import java.io.*;              // Line.java

class Line implements Serializable
{
   int x1, y1, x2, y2;

   Line(int x1, int y1, int x2, int y2)
   {     this.x1 = x1;              this.y1 = y1;
         this.x2 = x2;              this.y2 = y2;
   }
}
```

```
//-------------------------------------------------------------

import java.rmi.*;           // ChatClient.java

public interface ChatClient extends Remote
{
    void chatNotify(Message m) throws RemoteException;
}

//-------------------------------------------------------------


import java.rmi.*;           // ChatClientImpl.java
import java.rmi.server.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;

public class ChatClientImpl extends JFrame
    implements ChatClient, LineSender, ActionListener
{
  private JTextArea ta;       // display chatter
  private JTextField tf;      // message to send

  private ChatServer cs;      // reference to server
  private String name = null; // user's name

  private DrawPad dp;               // whiteboard
  private ServerTalker st;

  ChatClientImpl() throws RemoteException
  {
    JButton list, cDraw, cText, dis;
    System.out.println("Starting up Chat Client");

    Container cp = getContentPane();
```

```
   JPanel p = new JPanel();
   p.add(list = new JButton("List"));   // list users
   list.addActionListener(this);
   p.add(cDraw = new JButton("ClearDraw"));
   cDraw.addActionListener(this);
   p.add(cText = new JButton("ClearText"));
   cText.addActionListener(this);
   p.add(dis = new JButton("Disconnect"));
   dis.addActionListener(this);
   cp.add(p, "North");

   ta = new JTextArea(150, 20);          // transcript
   cp.add(ta, "West");

   dp = new DrawPad(this);               // whiteboard
   cp.add(dp, "Center");

   tf = new JTextField(40);              // text entry field
   tf.addActionListener(this);
   cp.add(tf, "South");
}

void registerChatter()
{
   NameDialog nd = new
         NameDialog(new JFrame("Enter Name"),
                        "Enter your name", false);
   name = nd.getName();
   nd.setVisible(false);      // get rid of NameDialog
   nd = null;
   String url =
         "rmi://r-lnx233.cs.uiowa.edu:5050/ChatServerImpl";
   System.setSecurityManager(new RMISecurityManager());

   try
   {
      UnicastRemoteObject.exportObject(this);
                                   // export remote methods
      cs = (ChatServer)Naming.lookup(url);
      st = new ServerTalker(cs, name);
      cs.register(this, name);          // remote call
    }
    catch (RemoteException e)
    {
      System.out.println("Cannot find Registry ");
      System.exit(0);
    }
    catch (MalformedURLException e)
    {
      System.out.println("Bad URL: " + url);
      System.exit(0);
    }
    catch (NotBoundException e)
    {
      System.out.println("Service not bound." );
      System.exit(0);
    }
    tf.requestFocus();
  }

  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() == tf)
    {
      boolean alive = st.addMessage(
                new Message(name, tf.getText().trim()));
      if (!alive) ta.append("*** Server Error ***\n");
      tf.setText("");
    }
    if (e.getSource() instanceof JButton)
    {
        String arg = e.getActionCommand();
        if (arg.equals("ClearText"))  ta.setText("");
        else if (arg.equals("ClearDraw"))  dp.clearScreen();
        else if (arg.equals("Disconnect"))
        {
            st.addMessage(new Message(">>> " + name,
                                        "Logged off. Bye"));
            System.exit(0);
        }
        else if (arg == "List") getUserList();
    }
    tf.requestFocus();
}

public void sendLine(Line ln)
{
    boolean alive = st.addMessage(new Message(name, ln));
    if (!alive) ta.append("*** Server Error ***\n");
}

void getUserList()
{
    String [] users = null;
    try
    {     users = cs.listChatters();
    }
    catch (RemoteException e)
    {
        System.out.println(e);
        users = new String[1];
        users[0] = "*** Error ***";
    }
    for (int k = 0; k < users.length; k++)
            ta.append(">>> "+users[k]+"\n");
}
```

```
  public void chatNotify(Message m)
                                throws RemoteException
  {
    if (m.getText() != null)
      ta.append(m.getSender() + ": " + m.getText()+"\n");
    if (m.getLine() != null && !m.getSender().equals(name))
      dp.addLine(m.getLine());
  }

  public static void main(String [] args)
                                throws RemoteException
  {
    ChatClientImpl cc = new ChatClientImpl();
    System.out.println("Created ChatClientImpl");
    cc.registerChatter();       // reg. client with server
    cc.setSize(700, 400);
    cc.setVisible(true);
  }
}

//-------------------------------------------------------------

import java.util.*;     // ServerTalker.java
import java.rmi.*;

class ServerTalker extends Thread
{
  private List<Message> messages =
                                new LinkedList<Message>();
  private ChatServer cs;

  public ServerTalker(ChatServer c, String name)
  {
    cs = c;
    messages.add(new Message(">>> ",
                                name + " just connected"));
    start();
  }

  synchronized boolean addMessage(Message e)
  {
    if (cs == null)
    {  System.out.println("Server reference is null");
       return false;
    }
    messages.add(e);
    notifyAll();
    return true;
  }

  public synchronized void run()
  {
    boolean running = true
    while (running)
    {
        try
        {  if (messages.isEmpty()) wait();
           cs.postMessage(messages.get(0));
           messages.remove(0);
        }
        catch (InterruptedException e)
        {  System.out.println("Interrupted: " + e);  }
        catch (RemoteException e)
        {  System.out.println("Server down?" + e);
           cs = null;
           running = false;
        }
        yield();
    }
  }
}
```

```
//------------------------------------------------------------

public interface LineSender  // LineSender.java
{
      void sendLine(Line ln);
}

//------------------------------------------------------------

import java.awt.*;                  // DrawPad.java
import java.awt.event.*;
import java.util.*;
import javax.swing.*;

class DrawPad extends JPanel
         implements MouseListener, MouseMotionListener
{
   private int lastx, lasty;
   private int lastIndex = 0;    // used to mark drawing list

   private java.util.List<Line> lines =
                                  new ArrayList<Line>(500);
   private LineSender ls;           // client created DrawPad

   private Image img;               // for double buffering
   private Graphics db;             // to access offscreen buffer

   DrawPad(LineSender s)
   {
      ls = s;
      addMouseListener(this);
      addMouseMotionListener(this);
   }
```

```
public void paintComponent(Graphics g)
{
   super.paintComponent(g)
   if (img != null)
         g.drawImage(img, 0, 0, this);
   else
   {     g.drawString("Initializing...", 5, 30);
         render();
   }
}

void render()
{
   if (img == null)
   {  img = this.createImage(400, 400);
      db = img.getGraphics();
      db.setColor(Color.white);
      db.fillRect(0, 0, 400, 400);
      db.setColor(Color.black);
      db.drawRect(1, 1, 398, 398);   db.drawRect(0, 0, 400, 400);
   }
   for (Line ln : lines)
      db.drawLine(ln.x1, ln.y1, ln.x2, ln.y2);
   lastIndex = lines.size();
   repaint();
}

void clearScreen()
{
   img = null;
   lines.clear();
   lastIndex=0;
   render();
}
```

```
   public void mousePressed(MouseEvent e)
   {
      lastx = e.getX();  lasty = e.getY();
   }

   public void mouseReleased(MouseEvent e)
   {
      int x = e.getX(), y = e.getY();
      drawLine(x,y);
   }
   public void mouseEntered(MouseEvent e) { }
   public void mouseExited(MouseEvent e) { }
   public void mouseClicked(MouseEvent e) { }

   public void mouseDragged(MouseEvent e)
   {
      int x = e.getX(), y = e.getY();
      drawLine(x, y);
   }

   public void mouseMoved(MouseEvent e) { }


   void drawLine(int x, int y)
   {
      if ((Math.abs(lastx-x)>0) || (Math.abs(lasty-y)>0))
      {
         Line ln = new Line(x, y, lastx, lasty);
         ls.sendLine(ln);
         lastx=x;      lasty=y;
         lines.add(ln);
         render();
      }
   }
//-------------------------------------------------------------------
```

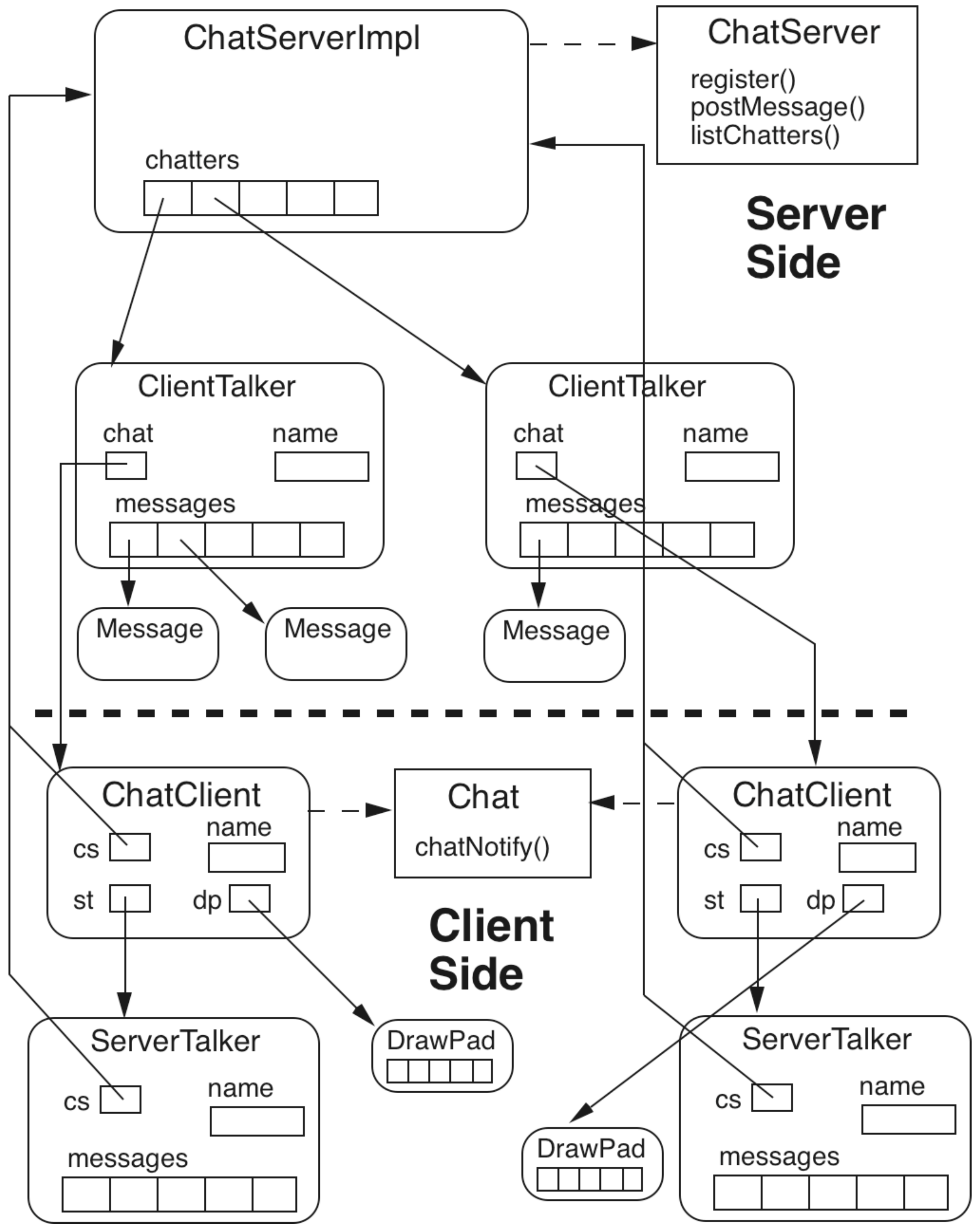
```
import java.awt.*;                    // NameDialog.java
import java.awt.event.*;
import javax.swing.*;
class NameDialog extends JDialog implements ActionListener
{
      private JTextField tf = new JTextField(20);
      private String value = null;

      NameDialog(JFrame f, String s, boolean modal)
      {
         super(f, s, modal);
         Container cp = getContentPane();
         cp.setLayout(new FlowLayout());
         add(new JLabel("Enter your name:"));
         cp.add(tf);
         tf.addActionListener(this);
         JButton ok = new JButton("OK");
         cp.add(ok);
         ok.addActionListener(this);
         pack();        setVisible(true);
         tf.requestFocus();
      }

      public void actionPerformed(ActionEvent e)
      {
         if (tf.getText().length() >= 1)
               value = tf.getText().trim();
      }

      public String getName()
      {
         while (value==null || value.equals(""))
               tf.requestFocus();
         return value;
      }
}
```

Server Side

ChatServer.java	ChatServer.class
ChatServerImpl.java	ChatServerImpl.class
	ChatServerImpl_Stub.class
ClientTalker.java	ClientTalker.class
Message.java	Message.class
Line.java	Line.class
	ChatClient.class
	ChatClientImpl_Stub.class

Client Side

ChatClient.java	ChatClient.class
ChatClientImpl.java	ChatClientImpl.class
	ChatClientImpl_Stub.class
ServerTalker.java	ServerTalker.class
DrawPad.java	DrawPad.class
LineSender.java	LineSender.class
NameDialog.java	NameDialog.class
	Message.class
	Line.class
	ChatServer.class
	ChatServerImpl_Stub.class


ChatServerImpl
chatters
ChatServer
register()
postMessage()
listChatters()
Server Side
ClientTalker
chat
name
messages
ClientTalker
chat
name
messages
Message
Message
Message
ChatClient
cs
name
st
dp
Chat
chatNotify()
ChatClient
cs
name
st
dp
Client Side
ServerTalker
cs
name
messages
DrawPad
DrawPad
ServerTalker
cs
name
messages

go Files

```
% more server/go1
\javac Message.java
\javac Line.java
echo "Message and Line compiled."

cp Message.class ../client
echo "Message.class copied to client."

cp Line.class ../client
echo "Line.class copied to client."
-----------------------------------------------------------
% more client/go2
\javac ChatClient.java
\javac LineSender.java
echo "Interfaces ChatClient and LineSender compiled."

cp ChatClient.class ../server
echo "ChatClient.class copied to server."
-----------------------------------------------------------
% more server/go3
\javac ChatServer.java
echo "Interface ChatServer compiled."

cp ChatServer.class ../client
echo "ChatServer.class copied to client."
-----------------------------------------------------------
% more server/go4
\javac ChatServerImpl.java
\javac ClientTalker.java
echo "ChatServerImpl and ClientTalker compiled."

rmic -v1.2 ChatServerImpl
echo "Stub file created."

cp ChatServerImpl_Stub.class ../client
echo "Stub copied to client."
-----------------------------------------------------------
```

```
% more client/go5
\javac ChatClientImpl.java
\javac ServerTalker.java
\javac DrawPad.java
\javac NameDialog.java
echo "ChatClientImpl, ServerTalker, DrawPad,
                                        NameDialog compiled."
rmic -v1.2 ChatClientImpl
echo "Stub file created."

cp ChatClientImpl_Stub.class ../server
echo "Stub copied to server."
-------------------------------------------------------
% more server/go6
java ChatServerImpl&
echo "Server started."
echo "Do not forget to kill the process when done."
-------------------------------------------------------
% more client/go7
java ChatClientImpl&
echo "Client started."
```

go (in Chat)

```
cd server
go1
cd ../client
go2
cd ../server
go3
go4
cd ../client
go5
cd ../server
go6
cd ../client
go7
```