Advanced Java Programming Course

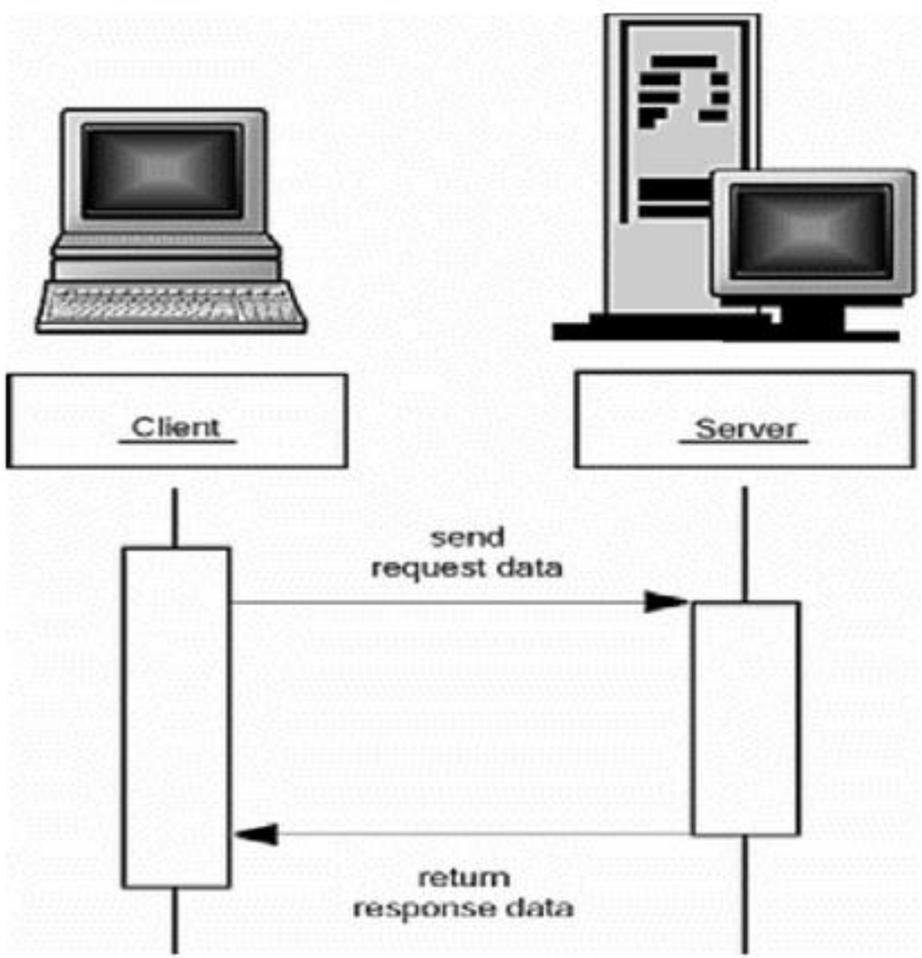
Faculty of Information Technologies
Industrial University of Ho Chi Minh City

Session objectives

- Networking introduction
- URL Class
- InetAddress Class
- Working with Socket
  - connection-based protocol
  - connectionless-oriented protocol
- Remote Method Invocation (RMI)

Introduction

Client/Server working model



Internet architecture: Protocols & Layers

- Internet architecture based on Internet protocol suite concept also called TCP/IP suite.
- TCP/IP suite contains many protocols
  - SMTP, HTTP, FTP, BitTorrent, POP3, UDP, IRC, SNMP, TCP, IP, Wi-Fi..
- The various protocols are often categorized in Layers

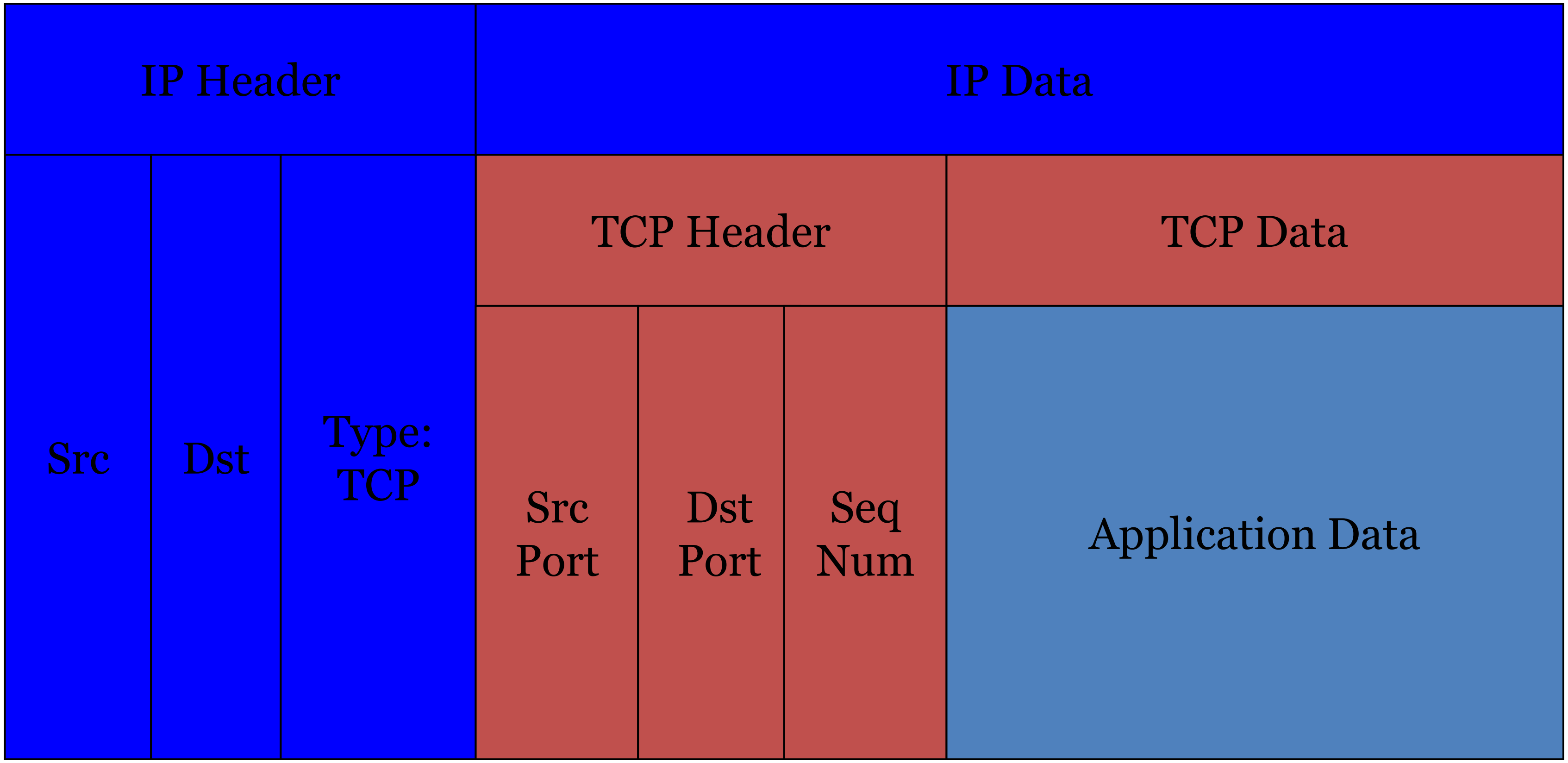
Layer	Name	Functionality	Description
IV	Application	HTTP, FTP	This is where the "high level" protocols such as File Transfer Protocol (FTP) and Hypertext Transfer Protocol (HTTP) operate.
III	Transport	TCP, UDP(User Datagram Protocol)	This layer deals with opening and maintaining connections, and ensures that packets are transmitted and received.
II	Network	IP	This layer defines Internet Protocol (IP) addresses, and deals with packet transmission from one IP address to another.
I	Link	IP	This layer describes the physical equipment required for communications, such as twisted pair cables.

TCP/IP Overview

- TCP
    - TCP is a stream-based protocol over IP
    - TCP promises a reliable transport
    - TCP takes care of packet ordering, packet loss and data corruption.
    - A TCP stream is a connection.
- IP
    - IP is a protocol for connecting networks
    - IP is a packet-based protocol
    - Packets can be dropped, garbled or re-ordered and duplicated:
    - IP promises nothing but best-effort delivery
    - IP usually runs over ethernet, but not always

TCP/IP Overview

TCP over IP



How Computers Communicate

- When two applications (A and B) on remote machines want to communicate with each other

- Using TCP
  - A contacts B and wait for B's response
  - If B agree, will then respond to A (a connection is established)
  - A & B now can send data back and forth over that connection.

- Using UDP
  - A only needs to know B's address.
  - A send data to B (do not care B is available or not)

Networking Application Basics
Host & Internet Address

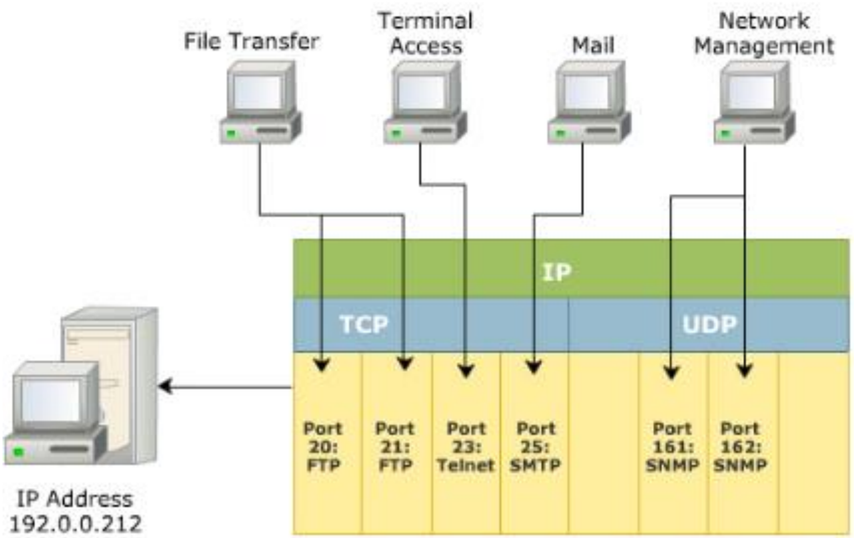
- Host
  - Devices connected to the Internet are called hosts
  - Most hosts are computers, but hosts also include routers, printers, fax machines, soda machines, bat houses, etc.
- Internet addresses
  - Every host on the Internet is identified by a unique, four-byte Internet Protocol (IP) address.
  - This is written in dotted quad format like 199.1.32.90 where each byte is an unsigned integer between 0 and 255.
  - There are about four billion unique IP addresses, but they aren't very efficiently allocated

Networking Application Basics
Concept of Ports

- A computer generally has a single physical connection available for the network.
- Several applications running on a machine need to use this single physical connection for communication.
- If data arrives at these physical connections, there is no means to identify the application to which it should be forwarded.
- Hence, data being sent and received on the physical connection are based on the *concept of ports*.

Networking Application Basics
Concept of a Ports

- The physical connection is logically numbered within a range of 0 to 65535 called as Ports.
- The port numbers ranging from 0 to 1023 are reserved
  - for well known services such as HTTP, FTP
  - Your applications should not use any of the port numbers in this range.
- Data transmitted over the Internet is accompanied with the destination address and the port number.
- The destination address identifies the computer
- The port number identifies the application.



The InetAddress Class

- The InetAddress represents an Internet Protocol (IP) address
  - It retrieves the host address/name.
  - It provides methods to resolve host names to their IP addresses and vise versa.

```java
import java.net.InetAddress;
public class TestInetAddress {
    public static void main(String[] args) {
        try {
            InetAddress add=InetAddress.getByName("www.google.com.vn");
            System.out.println("google IP address:"+add.getHostAddress());

            add=InetAddress.getByName("74.125.128.94");
            System.out.println("google host:"+add.getHostName());

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
Problems  @ Javadoc  Declaration  Console
<terminated> TestInetAddress [Java Application] C:\jdk1.6.0_16\bin\javav
google IP address:74.125.128.94
google host:hg-in-f94.1e100.net
```

The URL class

URL

- A URL, short for "Uniform Resource Locator", is a way to unambiguously identify the location of a resource on the Internet.
  - The resource can be an HTML page, an image or simply any file.
- URL form protocol://resource

- Protocol Identifier
  - Name of the protocol, which is used to fetch the resource from the Internet.
- Resource Name
  - Address of the resource
  - Separated from the protocol identifier with a colon and two forward slashes
  - Resource format depends on the protocol being used.

URL examples

- ✓ http://java.sun.com/
- ✓ file:///Macintosh%20HD/Java/Docs/JDK%201.1.1%20docs/api/java.net.InetAddress.html#_top_
- ✓ http://www.macintouch.com:80/newsrecent.shtml
- ✓ ftp://ftp.info.apple.com/pub/
- ✓ mailto:elharo@metalab.unc.edu
- ✓ telnet://utopia.poly.edu
- ✓ ftp://mp3:mp3@138.247.121.61:21000/c%3a/stuff/mp3/
- ✓ http://elharo@java.oreilly.com/
- ✓ http://metalab.unc.edu/nywc/comps.phtml?category=Choral+Works

The java.net.URL class

- The URL class represents an URL object.
- The URL class contains methods to
    - create new URLs
    - parse the different parts of a URL
    - get an input stream from a URL so you can read data from a server
    - get content from the server as a Java object
- Supported Protocols

file	ftp	gopher	http	mailto
appletresource	doc	netdoc	systemresource	verbatim

The java.net.URL class

- Construct URL object

Constructor Summary
URL(String spec) Creates a URL object from the String representation.
URL(String protocol, String host, int port, String file) Creates a URL object from the specified protocol, host, port number, and file.
URL(String protocol, String host, int port, String file, URLStreamHandler handler) Creates a URL object from the specified protocol, host, port number, file, and handler.
URL(String protocol, String host, String file) Creates a URL from the specified protocol name, host name, and file name.
URL(URL context, String spec) Creates a URL by parsing the given spec within a specified context.
URL(URL context, String spec, URLStreamHandler handler) Creates a URL by parsing the given spec with the specified handler within a specified context.

Parsing URLs

- The java.net.URL class has five methods to split a URL into its component parts. These are:
  - public String getProtocol()
  - public String getHost()
  - public int getPort()
  - public String getFile()
  - public String getRef()

```
try {
  URL u = new URL("http://www.poly.edu/fall97/grad.html#cs ");
  System.out.println("The protocol is " + u.getProtocol());
  System.out.println("The host is " + u.getHost());
  System.out.println("The port is " + u.getPort());
  System.out.println("The file is " + u.getFile());
  System.out.println("The anchor is " + u.getRef());
}
catch (MalformedURLException e) { }
```

Reading Data from a URL

- The openStream() method connects to the server specified in the URL and returns an InputStream object fed by the data from that connection.

```
public final InputStream openStream() throws IOException
```

- Any headers that precede the actual data are stripped off before the stream is opened.
- Network connections are less reliable and slower than files. Buffer with a BufferedReader or a BufferedInputStream.

The URLConnection class

URLConnections

- The java.net.URLConnection class is an abstract class that handles communication with different kinds of servers like ftp servers and web servers.
- Protocol specific subclasses of URLConnection handle different kinds of servers.
- By default, connections to HTTP URLs use the GET method.
- Capacities:
  - Can send output as well as read input
  - Can post data
  - Can read headers from a connection

URLConnection five steps

1. The URL is constructed.
2. The URL's openConnection() method creates the URLConnection object.
3. The parameters for the connection and the request properties that the client sends to the server are set up.
4. The connect() method makes the connection to the server.
5. The response header information is read using getHeaderField().

I/O Across a URLConnection

- Data may be read from the connection in one of two ways
  - raw by using the input stream returned by getInputStream()
  - through a content handler with getContent().
- Data can be sent to the server using the output stream provided by getOutputStream().

```
void doReadSampel()throws Exception{
    URL u = new URL("http://www.wwwac.org/");
    URLConnection uc = u.openConnection();
    uc.connect();
    InputStream in = uc.getInputStream();
    // read the data...
    in.close();
}
```

Send request to server

- Since a URLConnection doesn't allow output by default, you have to call setDoOutput(true) before asking for an output stream.

```
public InputStream sendRequest(URL url, String method,
        String params)throws Exception{
    HttpURLConnection con=(HttpURLConnection)url.openConnection();
    con.setRequestMethod(method.toUpperCase());
    con.setDoOutput(true);

    OutputStreamWriter ow=new OutputStreamWriter(con.getOutputStream(),"UTF-8");
    ow.write(params);//send data to server
    ow.flush(); ow.close();

    int rc=con.getResponseCode();//response code from server
    if(rc==HttpURLConnection.HTTP_OK){
        return con.getInputStream();
    }
    return null;
}
```

Get response from server

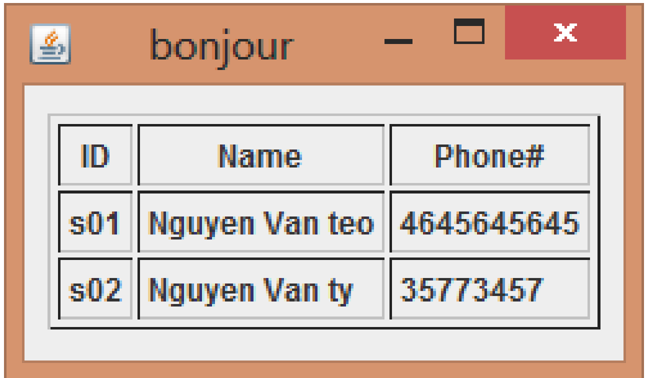
```
public String getResponse(InputStream is)throws Exception{
    BufferedReader br=new BufferedReader(new InputStreamReader(is));
    String line="",html="";
    while((line=br.readLine())!=null){
        html+=line+"\n";
    }
    br.close();
    return html;
}
```

HTML in Swing

HTML on Components

- Most text-based Swing components, such as labels, buttons, menu items, tabbed panes, and tool tips, can have their text specified as HTML. The component will display it appropriately.

```java
JLabel lbl=new JLabel("",JLabel.CENTER);
String text="<html><body><table width='100%' border='1'>" +
        "<tr><th>ID</th><th>Name</th><th>Phone#</th></tr>" +
        "<tr><td>s01</td><td>Nguyen Van teo</td><td>4645645645</td>" +
        "</tr><tr><td>s02</td><td>Nguyen Van ty</td><td>35773457" +
        "</td></tr></table></body></html>";
lbl.setText(text);
```



The JEditorPane class

- This class provides an even more complete HTML 3.2 renderer that can handle frames, forms, hyperlinks, and parts of CSS Level 1.
- The JEditorPane class also supports plain text and basic RTF

Constructors

```
public JEditorPane( )
public JEditorPane(URL initialPage) throws IOException
public JEditorPane(String url)  throws IOException
public JEditorPane(String mimeType, String text)
```

Navigate/Display contents

```
public void setPage(URL page) throws IOException
public void setPage(String url) throws IOException
public void setText(String html)
```

The JEditorPane class example


```java
private void navigate(String url) {
    try {
        contentPane.setPage(url);
    } catch (IOException e1) {
        statusLabel.setText(e1.getMessage());
    }
}
```

```java
contentPane.addHyperlinkListener(new HyperlinkListener() {
    @Override
    public void hyperlinkUpdate(HyperlinkEvent e) {
        try {
            if(e.getEventType( ) == HyperlinkEvent.EventType.ACTIVATED)
                contentPane.setPage(e.getURL());
        } catch (Exception e2) {
            e2.printStackTrace();
        }
    }
});
```

Socket programming

TCP Socket

Implementing server

Socket programming

- The Java interface with a TCP connection is a socket.
- A socket is a Java object which has methods to connect, accept connections and transfer data.
- Core Networking is in java.net.*
- TCP Sockets come in two flavors : ServerSocket and Socket.

Implementing server

The ServerSocket Class

- Represent the server side of the two-way communication.

- The ServerSocket has to bind to a specific port which should be free and available.
  - If the port is being used by any other application, an exception is thrown.

- If the ServerSocket class is successful in binding to a port, it can then wait and listen for client request to establish connections

Implementing server

The ServerSocket Class (cont.)

Constructor Summary
`ServerSocket()` Creates an unbound server socket.
`ServerSocket(int port)` Creates a server socket, bound to the specified port.
`ServerSocket(int port, int backlog)` Creates a server socket and binds it to the specified local port number, with the specified backlog.
`ServerSocket(int port, int backlog, InetAddress bindAddr)` Create a server with the specified port, listen backlog, and local IP address to bind to.

Implementing server

Working with the ServerSocket class

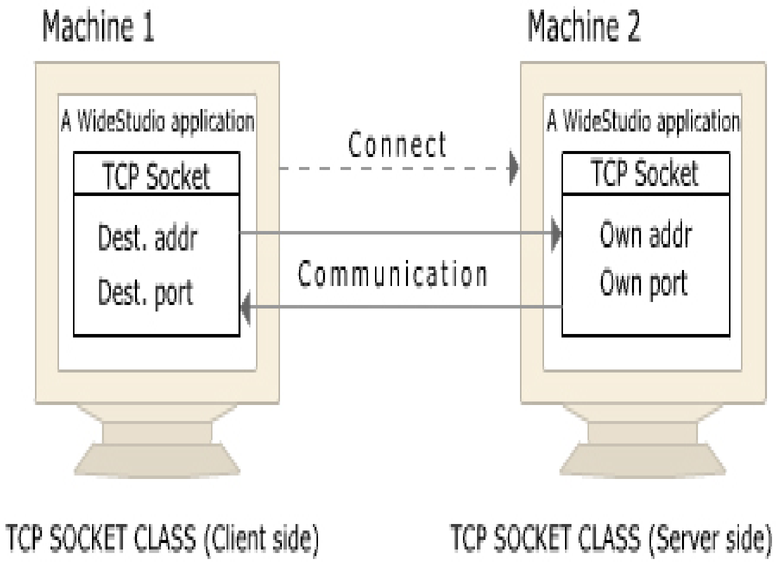
- Once a ServerSocket instance is created, you can invoke the accept() method to listen for client request.
- The accept () method is a blocking method that is once invoked it will wait till a client requests for a connection.
- When a client requests for a connection, the accept() method creates a socket object of class Socket and returns it.
- This returned object represents a proxy of a client.
- To communicate with the client, you retrieve the Inputstream and Outputstream of this proxy socket object.

Implementing server

Working with the Socket class

- Represent the connection between a client program and a server program.

- Represents the client side of the connection



Implementing server

Logic diagram to implements a server

1) It gets a command from the client through an incoming data stream.

2) It somehow fetches the information.

3) It sends the information to the client through the outgoing data stream.

Implementing server

Example

```java
import java.net.ServerSocket;
import java.net.Socket;
public class MyTestServer {
    public static void main(String[] args) throws Exception{
        ServerSocket svr=new ServerSocket(9999);
        System.out.println("server waiting on port 9999...");
        while(true){
            Socket clientSocket=svr.accept();
            processClientRequest(clientSocket);
        }
    }
    private static void processClientRequest(Socket clientSocket) {
        //do your works
    }
}
```

Problems | @ Javadoc | Declaration | Console

<terminated> MyTestServer [Java Application] C:\jdk1.6.0_16\bin\javaw.exe (Mar 12, 2012 9:05:10 AM)

```
server waiting on port 9999...
```

Implementing server

Serving Multiple clients

- One ServerSocket can accept multiple clients simultaneously.
- Use multithreading to handle them.
- One thread waits for “accept()”.
- Open a new thread for each connection.

Implementing server

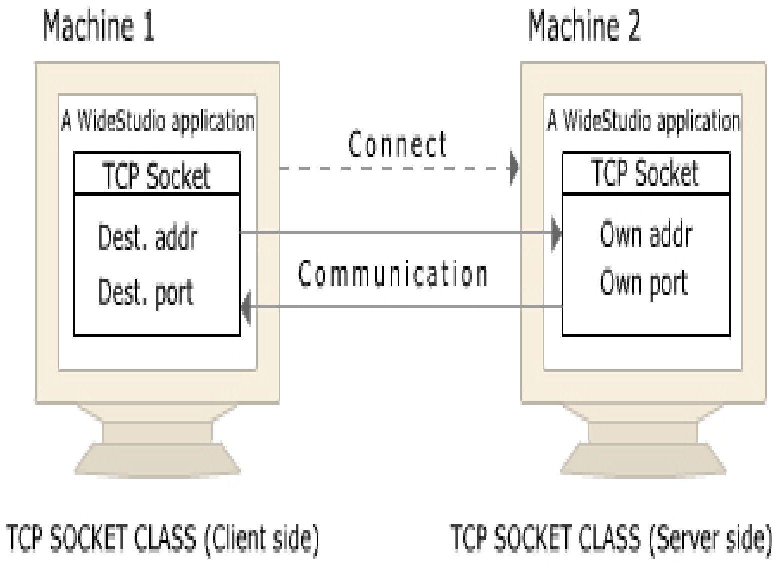
Serving Multiple clients example

```java
import java.net.ServerSocket;
import java.net.Socket;
public class MyTestServer {
    public static void main(String[] args) throws Exception{
        ServerSocket svr=new ServerSocket(9999);
        System.out.println("server waiting on port 9999...");
        while(true){
            Socket clientSocket=svr.accept();
            new Thread(new Processing(clientSocket)).start();
        }
    }
}
class Processing implements Runnable{
    private Socket clientSocket;
    public Processing(Socket clientSocket) {
    public void run() {
        //do your works
    }
}
```

For connection-based protocol - Socket class

- Represent the connection between a client program and a server program.

- Represents the client side of the connection



Implementing client

Logic diagram to implements a client

1) Create a connection to the server

2) Send request information to the server through the outgoing data stream.

3) It receives the information from the server through the incoming data stream.

4) Processing received information

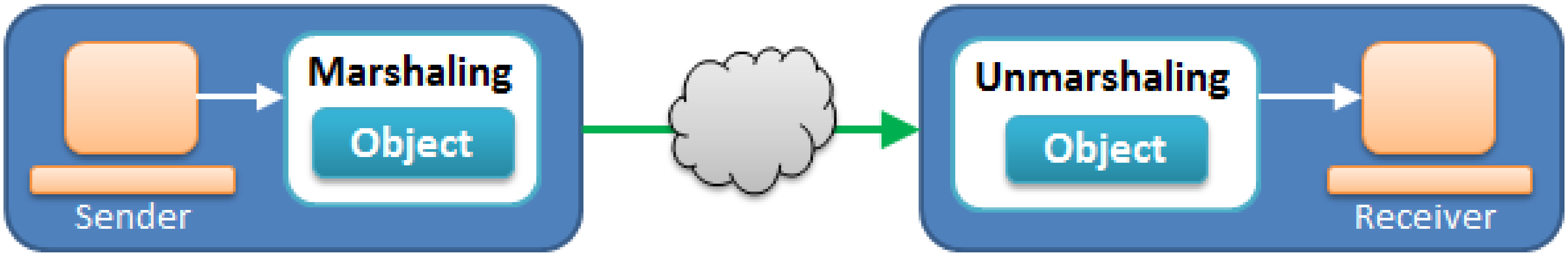
Implementing client

Client example

```java
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class MyTestClient {
    public static void main(String[] args) throws Exception{
        Socket soc=new Socket("localhost",9999);
        PrintWriter out=new PrintWriter(soc.getOutputStream(),true);
        out.println("request to server");
        Scanner in=new Scanner(soc.getInputStream());
        while(in.hasNextLine()){
            String line=in.nextLine();
            System.out.println(line);
        }
    }
}
```

Object transmission



- The processes of object transmission are:
  - Marshalling object to be transmitted
  - Send marshalled object over the network
  - Unmarshalling object and the execute
- *Note that object class must implement Serializable interface*
- To transmit *serialized objects*, we would use *ObjectInputStream* and *ObjectOutputStream* instead.

Socket programming

UDP

User Datagram Protocol (UDP)

introduction

- UDP can be used to send packets.
- The packets can be delivered in random order.
- It is up to the recipient to put the packets in order and to request retransmission of missing packets.
- UDP is most suited for applications where missing packets can be tolerated, for example, in audio or video streams...

User Datagram Protocol (UDP)

Create Datagram Socket

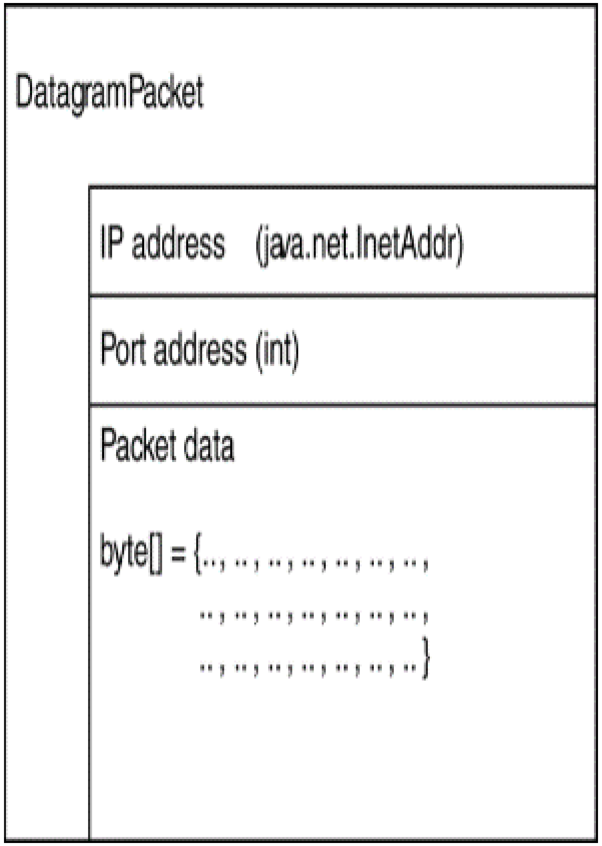
```
DatagramSocket socket = new DatagramSocket(port);
```

- The client uses a constructor that does not require a port number.
- This constructor just binds the DatagramSocket to any available local port.
- It doesn't matter what port the client is connected to because the DatagramPackets contain the addressing information.
- The server gets the port number from the DatagramPackets and send its response to that port.

User Datagram Protocol (UDP)

The Datagram packet

- An independent, self-contained message sent over the network.
- Used to employ a connectionless packet delivery service.
- Packet delivery is not guaranteed
- Datagram packet structure
  - Addressing information (IP)
  - Port
  - Sequence of bytes



User Datagram Protocol (UDP)

The DatagramPacket class

Constructor Summary
`DatagramPacket(byte[] buf, int length)` Constructs a `DatagramPacket` for receiving packets of length `length`.
`DatagramPacket(byte[] buf, int length, InetAddress address, int port)` Constructs a datagram packet for sending packets of length `length` to the specified port number on the specified host.
`DatagramPacket(byte[] buf, int offset, int length)` Constructs a `DatagramPacket` for receiving packets of length `length`, specifying an offset into the buffer.
`DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)` Constructs a datagram packet for sending packets of length `length` with offset `ioffset`to the specified port number on the specified host.
`DatagramPacket(byte[] buf, int offset, int length, SocketAddress address)` Constructs a datagram packet for sending packets of length `length` with offset `ioffset`to the specified port number on the specified host.
`DatagramPacket(byte[] buf, int length, SocketAddress address)` Constructs a datagram packet for sending packets of length `length` to the specified port number on the specified host.

User Datagram Protocol (UDP)

How to use DatagramSocket class

- Once a DatagramSocket instance is created, it can be used to send and receive data using packets.

Method Summary	
void	receive(DatagramPacket p) Receives a datagram packet from this socket.
void	send(DatagramPacket p) Sends a datagram packet from this socket.

User Datagram Protocol (UDP)

Processing logic - Client send Request to Server

```java
void clientSendRequest(DatagramSocket socket)throws Exception{
    byte[] buf = new byte[256];
    InetAddress address=InetAddress.getLocalHost();
    DatagramPacket packet = new DatagramPacket(
            buf, buf.length, address, 4445);
    socket.send(packet);
}
```

User Datagram Protocol (UDP)

Processing logic - Server receive Request from Client

```java
void serverReceiveRequest(DatagramSocket socket)
            throws Exception{
    byte[] buf = new byte[256];
    DatagramPacket packet = new DatagramPacket(buf, buf.length);
    socket.receive(packet);
    processing(packet.getData());
}
```

- The getData() method to retrieve that data from the packet.

User Datagram Protocol (UDP)

Processing logic - Server Sends Response

```
void serverSendResponse(DatagramSocket socket,
        DatagramPacket receivedPagake)throws Exception{
    byte[]buf=new byte[256];
    //fill data to buf array
    InetAddress address = receivedPagake.getAddress();
    int port = receivedPagake.getPort();
    DatagramPacket packet = new DatagramPacket(
            buf, buf.length, address, port);
    socket.send(packet);
}
```

User Datagram Protocol (UDP)

Processing logic -Client receive response

```java
void clientReceiveResponse(DatagramSocket socket)throws Exception{
    byte[]buf=new byte[256];
    DatagramPacket packet = new DatagramPacket(buf, buf.length);
    socket.receive(packet);

    processingResponse(packet.getData());
}

private void processingResponse(byte[] data) {
    //do your works
}
```

The Multicast Socket

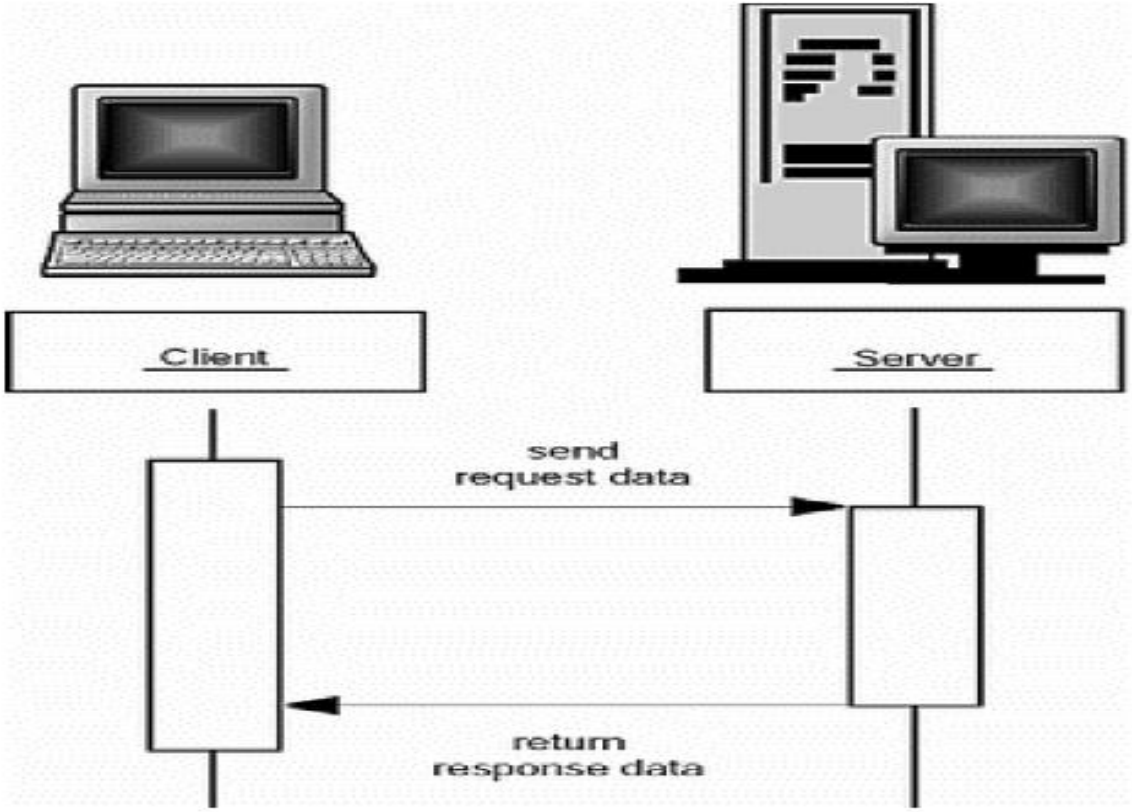
- Unicasting
  - Single receiver
- Multicasting
  - One or more receivers that have joined a multicast group
- Broadcasting
  - All nodes in the network are receivers

Read it by yourself

Remote Method Invocation

The Roles of Client and Server

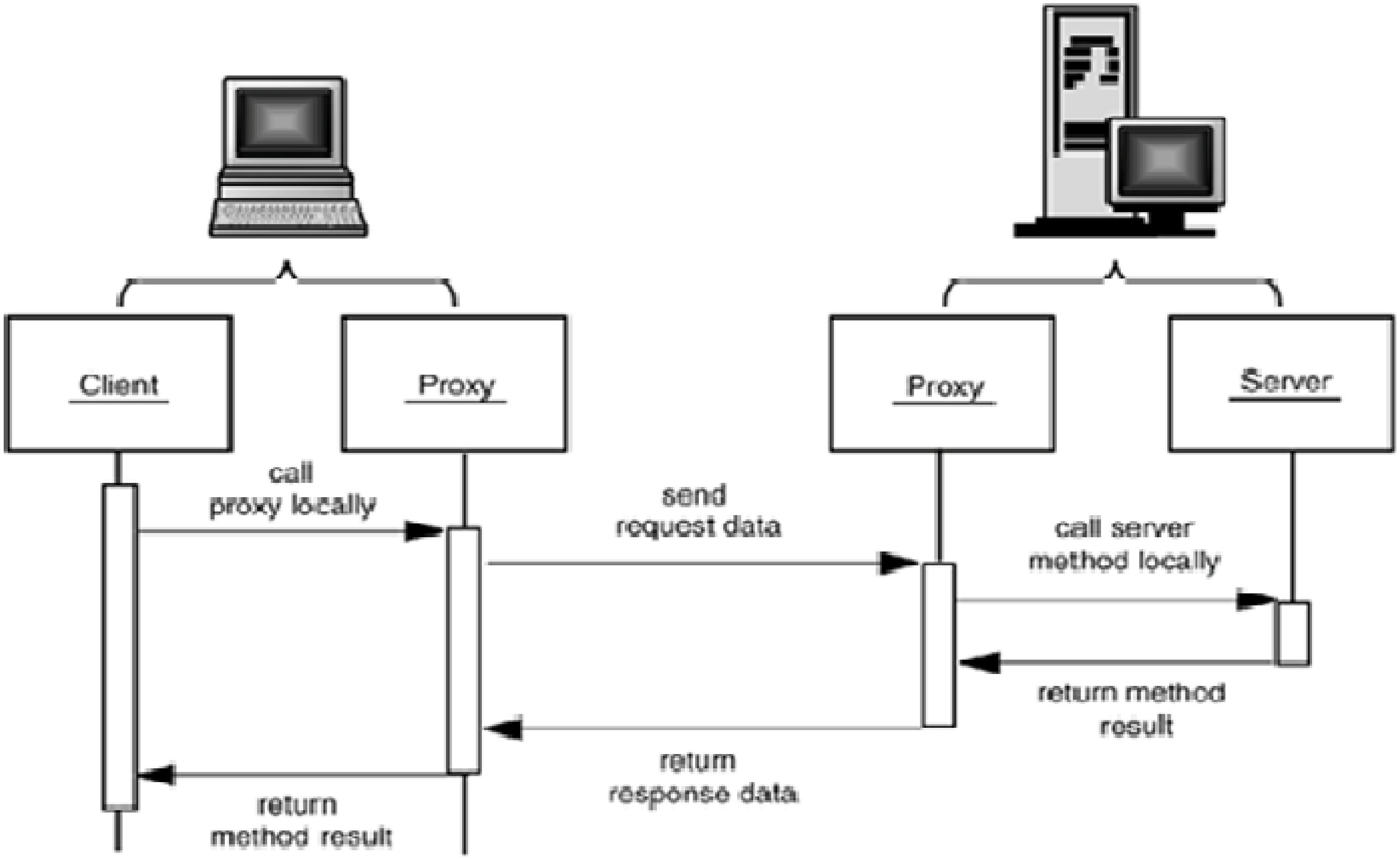
synchronized request/response model



Transmitting objects between client and server

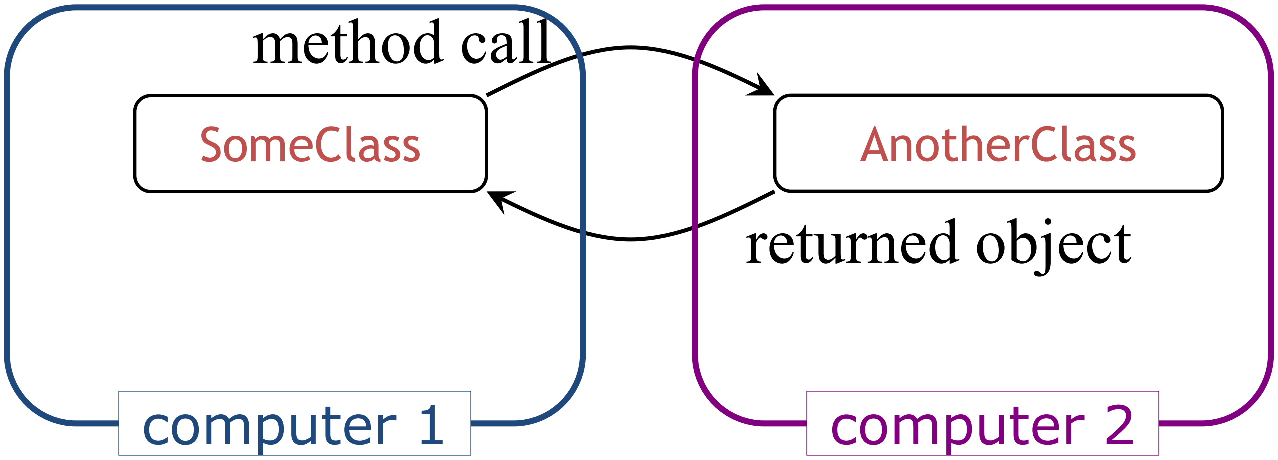
Remote Method call with proxies

new approach



“The network is the computer”*

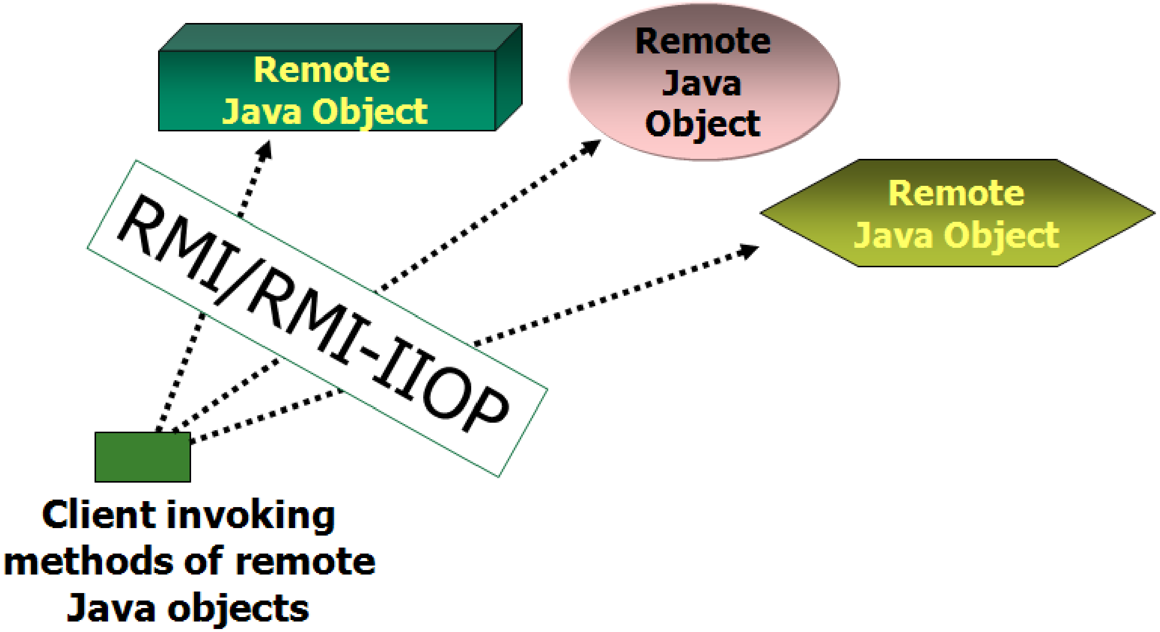
- Consider the following program organization:



- If the network *is* the computer, we ought to be able to put the two classes on different computers
- RMI is one technology that makes this possible

What is RMI?

- RMI allows objects in one JVM to invoke methods of objects in another JVM.
- All in java.rmi package



RMI Architecture

1. RMI Server provides an RMI service.

2. RMI Client invokes object methods of RMI service

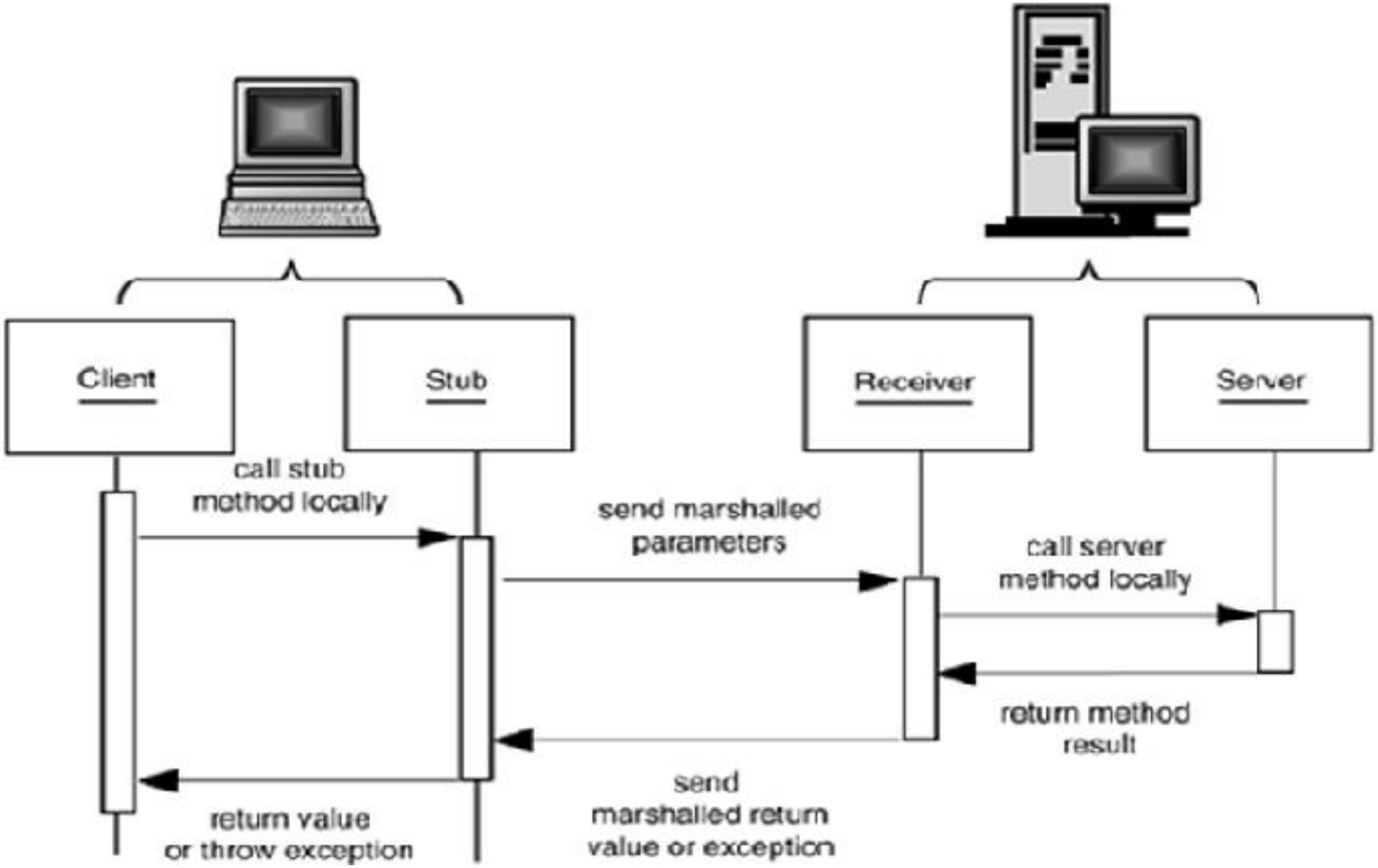
3. "rmiregistry" program
   - Runs as a separate process
   - Allows applications to register RMI services
   - Allows obtain a reference to a named service.

4. Stub object
   - Resides on the client side
   - Responsible for passing a message to a remote RMI service, waits for a response, and returns this response to the client.

Stubs

- When client code wants to invoke a remote method on a remote object, it actually calls an ordinary method of the Java programming language that is encapsulated in a surrogate object called a stub.
- The stub packages the parameters used in the remote method into a block of bytes.
- This packaging uses a device-independent encoding for each parameter.
- The process of encoding the parameters is called parameter marshalling.
- The purpose of parameter marshalling is to convert the parameters into a format suitable for transport from one virtual machine to another.

Parameters marshalling



Implementing RMI application process

1. Define RMI Service Interface

2. Implement RMI Service Interface

3. Create RMI Server program

4. Create RMI Client program

5. Running the RMI application

#1/5 Define RMI Service Interface

- Your client program needs to manipulate server objects, but it doesn't actually have copies of them.
- Their capabilities are expressed in an interface that is shared between the client and server and so resides simultaneously on both machines.
- The interface characteristics:
  - Must be public
  - Must extend the interface java.rmi.Remote
  - Every method in the interface must declare that it throws java.rmi.RemoteException (other exceptions may also be thrown)

```java
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Calc_interface extends Remote
{
    public long Add(int a,int b) throws RemoteException;
}
```

#2/5 Implementing RMI Service Interface

- On the server side, you must implement the class that actually carries out the methods advertised in the remote interface
- The class characteristics:
  - Should extend java.rmi.server.UnicastRemoteObject
  - Must implement a Remote interface
  - Must have a default(parametterless) constructor.

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class Calc_Impl extends UnicastRemoteObject implements Calc_interface
{
    public Calc_Impl()throws RemoteException{
    }
    @Override
    public long Add(int a,int b) {
        return (long)(a+b);
    }
}
```

#3/5 Creating RMI Server program

- The RMI server is responsible for:
  1. Instance object of a service implementation class
  2. Registering it with the RMI registry

```
import javax.naming.Context;
import javax.naming.InitialContext;
public class Calc_Server {
    public static void main(String[]args)throws Exception{
        Calc_interface obj=new Calc_Impl();
        Context ctx=new InitialContext();
        ctx.bind("rmi://localhost/calc_Server",obj);
        System.out.println("Server bound in Registry");
    }
}
```

#4/5 Creating RMI Client program

- The client obtains an object reference to the remote interface by making a lookup in the RMI registry
- Then invoking methods of the service interface.

```java
import javax.naming.Context;
import javax.naming.InitialContext;

public class Calc_Client{
    public static void main(String[]args) throws Exception{
        Context ctx=new InitialContext();
        Calc_interface calc=
                (Calc_interface)ctx.lookup("rmi://localhost/calc_Server");
        long x=calc.Add(100,40);
        System.out.println ("result: "+x);
    }
}
```

#5/5 Running the RMI System

- Start Server
  - Compile all java file as normal.

    cmd: `javac *.java`
  - Start rmiregistry program.

    cmd: `start rmiregistry`
  - Run server program.

    cmd: `start java Calc_Server`
- Start Client
  - Create policy file

    cmd: `client.policy`
  - Run client

    cmd: `java -Djava.security.policy=client.policy Calc_Client`

Security

- By default, the RMISecurityManager restricts all code in the program from establishing network connections.
- However, the program needs to make network connections
  - To reach the RMI registry; and
  - To contact the server objects
- To allow the client to connect to the RMI registry and the server object, you supply a policy file.
- Here is a policy file that allows an application to make any network connection to a port with port number of at least 1024:

```
grant{
        permission java.net.SocketPermission
                        "*:1024-65535", "connect,listen,resolve,accept";
};
```

Others

- Listing all bounded remote objects in registry

```
void listingAllObjects()throws Exception{
    Context ctx = new InitialContext();
    NamingEnumeration<NameClassPair> lst=ctx.list("rmi:");
    while (lst.hasMore()){
        System.out.println(lst.next().getName());
    }
```

- **Create your own RMIRegistry**

```
java.rmi.registry.LocateRegistry.createRegistry(1099);
```

Activation object

- Betty thought that if the RMI server keeps running without any connection, it seems wasting resources.
- It would be better if a remote object is delivered, shuts down itself when necessary and is activated on demand, rather than running all the time.
- Actually, Java RMI activation daemon, **rmid** is designed to do such job. The activation daemon will listen and handle the creation of activatable object on demand.

Activatable remote object

- An activatable remote object is a remote object that starts executing when its remote methods are invoked and shuts itself down when necessary.
- How to create an activatable object:
  - Create a class extends *java.rmi.activation.Activatable* class
  - and this class has a constructor to accept *ActivationID* and *MarshalledObject* parameters.

Step to develop: create activator interface

- Create activator service interface

```java
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface CalculatorServices extends Remote {
    public long addNum(long a,long b) throws RemoteException;
}
```

Step to develop: create acivatable object

- Create activatable object

```
import java.rmi.MarshalledObject;
import java.rmi.RemoteException;
import java.rmi.activation.Activatable;
import java.rmi.activation.ActivationID;

public class CalcutatorImpl extends Activatable
        implements CalculatorServices{
    public CalcutatorImpl(ActivationID id, MarshalledObject<?> data)
            throws RemoteException {
        super(id, 0);
    }
    //business method
    public long addNum(long a, long b) throws RemoteException {
        return a+b;
    }

}
```

Step to develop: Create setup (server) program

- To make a remote object accessible via an activation identifier over time, you need to register an activation descriptor for the remote object and include a special constructor that the RMI system calls when it activates the activatable object.

- The following classes are involved with the activation process:
  - ActivationGroup class -- responsible for creating new instances of activatable objects in its group.
  - ActivationGroupDesc class -- contains the information necessary to create or re-create an activation group in which to activate objects in the same JVM.
  - ActivationGroupDesc.CommandEnvironment class -- allows overriding default system properties and specifying implementation-defined options for an ActivationGroup.
  - ActivationGroupID class -- identifies the group uniquely within the activation system and contains a reference to the group's activation system.
    - getSystem()-- returns an ActivationSystem interface implementation class.
  - MarshalledObject class -- a container for an object that allows that object to be passed as a paramter in an RMI call.

Create a set-up program

1. Install security manager for the *ActivationGroup* VM.
2. Set security policy
3. Create an instance of *ActivationGroupDesc* class
4. Register the instance and get an *ActivationGroupID.*
5. Create an instance of *ActivationDesc.*
6. Register the instance with rmid.
7. Bind or rebind the remote object instance with its name
8. Exit the system.

```java
//step 1: Install security manager for the ActivationGroup VM.
System.setSecurityManager(new SecurityManager());
//step 2: Set security policy
Properties pros = new Properties();
pros.put("java.security.policy","rmi.policy");
//step 3: Create an instance of ActivationGroupDesc class
ActivationGroupDesc.CommandEnvironment ae = null;
ActivationGroupDesc group = new ActivationGroupDesc(pros, ae);
//step 4:Register the instance and get an ActivationGroupID.
ActivationGroupID groupId = ActivationGroup.getSystem().registerGroup(group);
//step 5:Create an instance of ActivationDesc.
String location = "file:.\\";
MarshalledObject<Object> data = null;
ActivationDesc desc = new ActivationDesc(groupId,"CalcutatorImpl",location,data);
//step 6:Register the instance with rmid.
CalculatorServices calcu = (CalculatorServices) Activatable.register(desc);
//step 7:Bind or rebind the remote object instance with its name
Naming.rebind("rmi://localhost:1099/CalculatorServices",calcu);
System.out.print("Object is hosting");
//step 8:Exit the system.
System.exit(0);
```

Step to develop: client

```java
import java.rmi.Naming;

public class Client {
    public static void main(String[] args) throws Exception{
        Object obj=Naming.lookup(
                "rmi://localhost:1099/CalculatorServices");
        CalculatorServices cal = (CalculatorServices)obj;
        long kq = cal.addNum(1, 3);
        System.out.println("Result : "+kq);
    }
}
```

Step to develop: policy file

- In order to run the code successfully, we need a policy file.
- The following policy file is for all permissions.

```
grant {
    permission java.security.AllPermission;
};
```

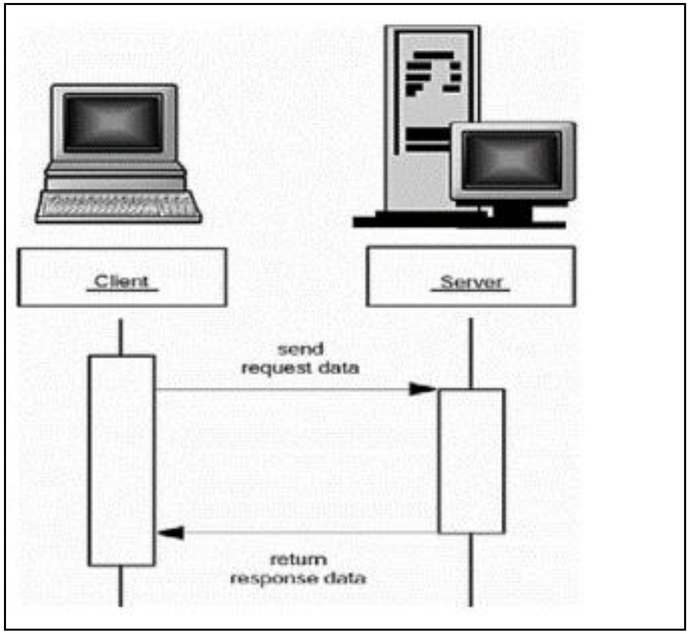
- For specific permission, you need to consult related documentation.

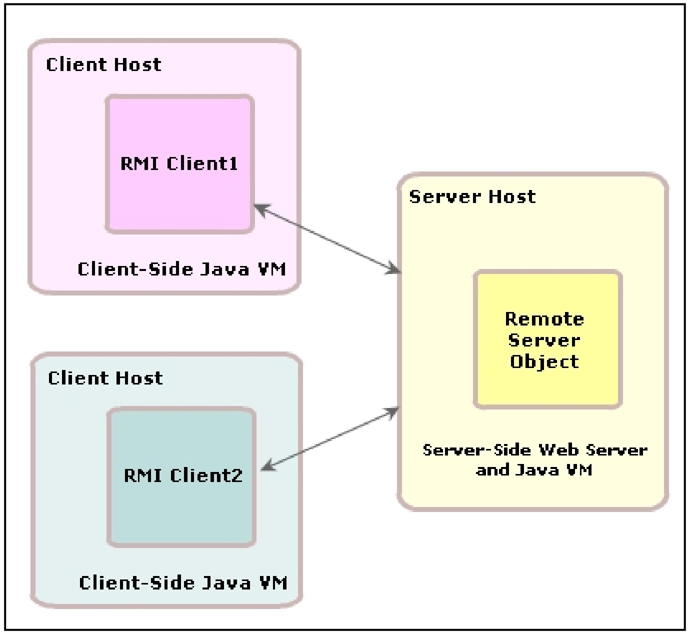
Step to develop: running

- Compile 4 classes
  - Using ***javac*** tool
- Start the rmiregistry
  - Using command: ***start rmiregistry***
- Start the activation daemon, rmid
  - Using command: ***start rmid -J-Djava.security.policy=rmi.policy***
- Run the setup program
  - Using tool: ***java -Djava.security.policy=rmi.policy*** YourServer
- Run the client
  - Uisng tool: ***java -Djava.security.policy=rmi.policy YourClient***

Summary

- Data exchange through network using Socket, UDP
- Remote method invocation





FAQ

That's all for this session!

Thank you all for your attention and patient !