Using Java for Network Programming Practice Course 1998

TEMPUS_SJEP-12495-97

József Vörös
Kando Polytechnic of Technology
Institute of Informatics

Contents

Preface

Course Description

The primary objective of this practise course is to teach students to develop simple networking applications and applets using Java. After this course students will be able to develop client-server networking applications using TCP/IP and distributed systems using Java RMI. The examples use Java 1.1 APIs.

Prerequisites

Potential students should meet the following prerequisites:

- Java programming fundamentals such as syntax, structure, java.lang and java.util package knowledge.
- Optional Java development environment knowledge (Java Development Kit is recommended).
- Object-oriented programming terminology and concepts such as objects, properties, and methods.
- Essential networking terminology and concepts such as TCP/IP networking, client-server theory and distributed computing fundamentals.

Objectives

At the end of this course, the student will be able to:

- Describe the fundamental concepts of TCP/IP networking in Java.
- Programming sockets and streams in Java.
- Develop simple TCP/IP server applications.
- Programming concurrent TCP/IP server applications.
- Develop client-server networking applications.
- Describe the fundamental concepts of Java RMI.
- Develop distributed applications using Java RMI.

Classroom Requirements

This course requires a classroom with a minimum of one computer for the instructor and one computer for each student. You can use any kind of computer with any kind of operating system on which the Java Development Kit runs. Typically used platform, for example, Windows 95/NT, Solaris.
Before class begins, install and configure JDK 1.1 and your favour word processor on all computers by using installation instruction in README of related product. You can use other JVM 1.1 compatible Java Development Environment.

Related books

[1.] Sridharan, Prashant: Advanced Java Networking, Prentice-Hall, 1997
[2.] Courtois, Todd : Java Networking and Communications, Prentice-Hall, 1998
[3.] Friedrichs, Jurgen: Java Thin-Client Programming for the Network Computing Environment, Prentice-Hall, 1998
[4.] Hughes, Merlin: JAVA Network Programming, Prentice-Hall, 1998
[5.] Umar, Amjad: Object-Oriented Client/Server Internet Environments, Prentice-Hall, 1998
[6.] Jim Farley: Java Distributed Computing, O'Reilly, 1998
[7.] Elliotte R. Harold: Java Network Programming, O'Reilly, 1997
[8.] David Flanagan, et al.: Java in a Nutshell, O'Reilly, 1997
[9.] Deitel, Harvey: Java How to Program, Prentice-Hall, 1998

Lab 1: Working with Internet addresses and DNS names

In this lab you will write code to processing Internet addresses and DNS names.
Estimated time to complete this lab: 30 minutes.

Objectives

After completing this lab, you will be able to:

- process Internet addresses

Prerequisites

Before working this lab, you should be familiar with DNS operational mechanism.

Tools in Java for processing Internet addresses

In Java, you can use java.net.Inetaddress class to represent Internet addresses. This class provides methods to get information related to Internet addresses.

Java's java.net.Inetaddress class:

```
public final class InetAddress extends Object implements Serializable {
        public boolean isMulticastAddress();
        public String getHostName();
        public byte[] getAddress();
        public String getHostAddress();
        public int hashCode();
        public boolean equals(Object obj);
        public String toString();
        public static InetAddress getByName(String host) throws UnknownHostException;
        public static InetAddress[] getAllByName(String host) throws
        UnknownHostException;
        public static InetAddress getLocalHost() throws UnknownHostException;
}
```

Applications should use the methods getLocalHost, getByName, or getAllByName to create a new InetAddress instance. Functions of methods are the following:
public boolean isMulticastAddress(): Utility routine to check if the InetAddress is a IP multicast address. IP multicast address is a Class D address i.e first four bits of the address are 1110.
public String getHostName(): Returns the hostname for this address. If the host is equal to null, then this address refers to any of the local machine's available network addresses.
public byte[] getAddress(): Returns the raw IP address of this InetAddress object.
public String getHostAddress(): Returns the IP address string "%d.%d.%d.%d".
public int hashCode(): Returns a hashcode for this IP address.

public boolean equals(Object obj): Compares this object against the specified object. The result is true if and only if the argument is not null and it represents the same IP address as this object.
public String toString(): Converts this IP address to a String.
public static InetAddress getByName(String host) throws UnknownHostException: Determines the IP address of a host, given the host's name. The host name can either be a machine name, such as "www.kando.hu", or a string representing its IP address, such as "193.224.40.2".
public static InetAddress[] getAllByName(String host) throws UnknownHostException: Determines all the IP addresses of a host, given the host's name. The host name can either be a machine name, such as "www.kando.hu", or a string representing its IP address, such as "193.224.40.2".
public static InetAddress getLocalHost() throws UnknownHostException: Returns the local host.

Common programming errors

- The getByName(), getAllByName() and getLocalHost() methods thrown UnknownHostException if no IP address for the host could be found.

Exercises

Exercise 1 Find IP address and hostname of the local machine.

Write an application which prints to standard output the IP address and hostname of the local machine. If any error occurs it prints the proper error message.

Solution:
myAddress.java

Exercise 2 Find IP address or hostname like nslookup

Write an application which reads from standard input IP addresses or hostnames. If the given input is IP address it prints to standard output the hostname of given address. If the given input is hostname it prints to standard output the IP address of given hostname. If any error is occurred it prints the proper error message. The program reads input by line and it is stopped, if user type „exit”.

Solution:
nslookup.java

Lab 2: Writing connection-oriented clients

In this lab you will write connection-oriented clients and using a low-level network programming interface for communicating with TCP/IP over the network..
Estimated time to complete this lab: 60 minutes.

Objectives

After completing this lab, you will be able to:

- make a client side TCP/IP network connection.
- send and receive data over the connection.
- make simple client programs.

Prerequisites

Before working this lab, you should be familiar with IP addressing protocol (hostname, IP address, communication port, socket).

Tools in Java for connection-oriented communication

The client side communication socket is supported by Socket class in java.net package. To communicate with server application we must know the following things:

- hostname or IP adress of the machine or machines which provides the service
- port number of the service on the host

The Java's java.net.Socket class

```
public class Socket extends Object {
        protected Socket()
        protected Socket(SocketImpl impl) throws SocketException
        public Socket(String host, int port) throws UnknownHostException, IOException
        public Socket(InetAddress address, int port) throws IOException
        public Socket(String host, int port, InetAddress localAddr, int localPort) throws
        IOException
        public Socket(InetAddress address, int port, InetAddress localAddr, int localPort)
        throws IOException
        public Socket(String host, int port, boolean stream) throws IOException
        public Socket(InetAddress host, int port, boolean stream) throws IOException
        public InetAddress getInetAddress()
        public InetAddress getLocalAddress()
        public int getPort()
        public int getLocalPort()
        public InputStream getInputStream() throws IOException
        public OutputStream getOutputStream() throws IOException
        public void setTcpNoDelay(boolean on) throws SocketException
        public void setSoLinger(boolean on, int val) throws SocketException
```

```
    public int getSoLinger() throws SocketException
    public synchronized void setSoTimeout(int timeout) throws SocketException
    public synchronized int getSoTimeout() throws SocketException
    public synchronized void close() throws IOException
    public String toString()
    public static synchronized void setSocketImplFactory(SocketImplFactory fac) throws
    IOException
}
```

Common programming errors

Any method in Java throws exception if an error is occurs. Let us see the general exceptions what may are thrown when we try to make a client side socket connection:

- java.lang.SecurityException: if the program don't have a permission to make connection. Typically an applet try to communicate with a remote computer, but It is only allowed to make network connection with local machine.
- java.net.BindException: if the operating system couldn't reserved the specified TCP port. It is already in use or we specified an invalid Internet address.
- java.net.SocketException: the operating system couldn't process the specified operation on the specified port.
- java.lang.IllegalArgumentException: if the given TCP port is out of range 0-65535.
- java.io.IOException: if any other communication error occurs.
- java.net.ProtocolException: if any protocol error occurs.
- java.net.UnkownHostException: if it couldn't find the specified hostname.
- java.net.NoRouteException: if it couldn't make contact with remote computer. Probably, a firewall filters our IP packets or a router out of order in the network.
- java.net.ConnectException: if service isn't available on specified TCP port.

During transmitting data, java.io.IOException or java.io.InterruptedException will be thrown if any communication error occurs.

A typical client does the following steps:

1. It reserves an unused, free TCP communication port.
2. It makes a networking connection between itself and the server.
3. It makes stream over networking connection to provide data transmission.
4. It communicates with server, sending or receiving data.
5. After communications, It closes the network connection and releases reserved communication port.

Exercises

Exercise 1 Command line Daytime client

Write a client application which returns the time string of a given host. The application uses a standard daytime service which available in TCP port 13. The arguments are hostname and port number. The default value of port number is 13. If the program is ran without arguments it prints day time of localhost. Program uses standard output to print information.

Step by step solution:
1. handling arguments
2. creating network connection
3. creating data stream
4. reading a time string and printing to standard output
5. closing network connection

Solution:
daytimeClient.java

Exercise 2 Command line DateAthost client

Write a client application which sets machine time to a given host after that it prints new value of date. The application uses a standard date at host service which available in TCP port 37. The arguments are hostname and port number. The default value of port number is 37. If the program is ran without arguments it communicates with localhost. Program uses standard output to print information.

Important note:
dateAtHost service gives a signed 32 bit integer with number of seconds since 1st January, 1990.

Step by step solution:
1. handling arguments
2. creating network connection
3. creating datastream
4. reading an integer
5. closing network connection
6. setting the time and printing a new value.

Solution:
dateAtHostClient.java

Exercise 3 Command line echo client

Write a client application which connects to echo server the specified TCP port on the specified host. After that it reads line from standard input to a string. If the string equals to '.' it closes connection and stops, otherwise it sends the string to the server. After sending, it reads reply from the server to a string and printing the read string to standard output. The arguments are hostname and port number. The hostname is required, the default value of port number is 7. If the program is ran without arguments it prints usage help.

Step by step solution:
1. handling arguments
2. creating network connection
3. creating datastream
4. reading line from standard input and sending to the server if string doesn't equal to '.'. Otherwise it closes network connection and stops.
5. reading string from the server and printing to standard output

6. go to step 4.

Solution:
echoClient.java

Exercise 4 Command line finger client

Write a client application which queries the specified user on the specified host. The client connects to a specified host on default TCP port of finger service (Default port is 37.). After the connection it sends the specified user name to the server. Finally, it reads reply of the server and printing the information to standard output. The program has one required argument which consists of user name and hostname separated by '@'. If the program is ran without arguments it prints usage help.

Step by step solution:
1. handling arguments
2. creating network connection
3. creating data stream
4. sending user name
5. reading user information and printing to standard output
6. closing network connection

Solution:
fingerClient.java

Lab 3: Writing connection-oriented servers

In this lab you will write connection-oriented servers and using a low-level network programming interface for communicating with TCP/IP over the network..
Estimated time to complete this lab: 60 minutes.

Objectives

After completing this lab, you will be able to:

- make a server side TCP/IP network connection.
- send and receive data over the connection.
- make iterative servers.
- make concurrent servers.

Prerequisites

Before working this lab, you should be familiar with IP addressing protocol (hostname, IP address, communication port, socket).

Tools in Java for connection-oriented communication

The server side communication socket is supported by ServerSocket class in java.net package. After connection is established it uses Socket class to communicate with client. Server applications have two type:

- iterative server
  It serves just one client at same time, another clients have to wait for a service in a queue. The client requests are processed in arrival order.
- concurrent server
  It can serve more client in parallel at same time.

The Java's java.net.ServerSocket class

```
Public class ServerSocket extends Object {
        public ServerSocket(int port) throws IOException
        public ServerSocket(int port, int backlog) throws IOException
        public ServerSocket(int port, int backlog, InetAddress bindAddr) throws IOException
        public InetAddress getInetAddress()
        public int getLocalPort()
        public Socket accept() throws IOException
        protected final void implAccept(Socket s) throws IOException
        public void close() throws IOException
        public synchronized void setSoTimeout(int timeout) throws SocketException
        public synchronized int getSoTimeout() throws IOException
        public String toString()
        public static synchronized void setSocketFactory(SocketImplFactory fac) throws
        IOException
```

}

Common programming errors

Any method in Java throws exception if an error occurs. Let us see the general exceptions what may are thrown when we try to make a server side socket connection:

- java.lang.SecurityException: the program don't have a permission to make connection. Typically an applet program try to create server socket, but only application programs are allowed to create server socket.
- java.net.BindException: if the operating system couldn't reserved the specified TCP port. It is already in use or we specified an invalid Internet address.
- java.net.SocketException: the operating system couldn't process the specified operation on the specified port.
- java.lang.IllegalArgumentException: if the given TCP port is out of range 0-65535.
- java.io.IOException: if any other communication error occurs.

A typical iterative server does the following steps:

1. It reserves a free TCP communication port and assigning with the name of service.
2. It waits for a client request.
3. It makes a connection between itself and the client.
4. It serves the client. After that, it closes the connection.
5. It goes to step 2.

A typical concurrent server does the following steps:

1. It reserves a free TCP communication port and assigns with the name of service.
2. It always waits for a client request. If client request arrived it makes a thread to parallel serve the client. While It's serving the client like iterative server in step 3 and step 4, it's waiting for another client request. After client is served it stops the serving thread.

Exercises

Exercise 1 DayTime server

Write a server application which listens on given TCP port and it sends date in string format to clients like standard daytime service. The program argument is the port number. The default value of port number is 13. Program uses standard output to print information.

Step by step solution:

1. handling arguments
2. creating server socket
3. waiting client request
4. creating network connection with client
5. creating data stream
6. sending time string to client
7. closing client network connection

8. go to step 3

Solution:
daytimeServer.java

Exercise 2 DateAtHost server

Write a server application which listens on given TCP port and it sends date in 32 bit integer format to clients like standard date at host service. The program argument is the port number. The default value of port number is 7. Program uses standard output to print information.
Important note:
dateAtHost service gives a signed 32 bit integer with number of seconds since 1st January, 1990.

Step by step solution:
1. handling arguments
2. creating server socket
3. waiting client request
4. creating network connection with client
5. creating data stream
6. sending date to the client
7. closing client network connection
8. go to step 3

Solution:
dateAtHostServer.java

Exercise 3 Echo server

Write a server application which listens on given TCP port. It reads string from client and sending back the same string to client like standard echo service. The program argument is the port number. The default value of port number is 7. Program uses standard output to print information.

Step by step solution:
1. handling arguments
2. creating server socket
3. waiting client request
4. creating network connection with client
5. creating data stream
6. reading string from client. After that sending back the string to the client
7. closing client network connection
8. go to step 3

Solution:
echoServer.java

Exercise 4 Simple finger server

Write a UNIX based server application which listens on given TCP port. First, it reads a user name to string. After that it seeks for the user in /etc/passwd file. If user is founded, the server sends user login name and full name to client in string format. If no such as user on the machine, the server sends „Sorry, no match to your query!”. The program argument is the port number. The default value of port number is 79. Program uses standard output to print information.

Note: You must open /etc/passwd file in Read Only mode. The file structure is the following:
login:x:UID:GID:Full name:Home directory:Shell
For example:
guest:x:600:600:Guest account:/home/guest:/bin/bash

Step by step solution:

1. handling arguments
2. creating server socket
3. waiting client request
4. creating network connection with client
5. creating data stream
6. reading user name
7. searching in passwd file
8. sending the result
9. closing client network connection
10. go to step 3

Exercise 5 Simple HTTP server

Write a server application which listens a given TCP port and it provides a simple http services. It’s only capability is to pass back a requested file in the server's directory or sub-directory. The request from a browser should be in the form: "GET /path/filename.xyz". If no file name or only „/” in the request, the server passes the default index.html file from server’s directory. If requested file is not found, it passes „404 Object not found”. If requested file is found but it don’t have a permission to read, the servers passes „403 Forbidden”. If any other error is occurred, the server passes „400 Bad request”. The program argument is the port number. The default value of port number is 1234. Program uses standard output to print information.

Step by step solution:

1. handling arguments
2. creating server socket
3. waiting client request
4. creating network connection with client in a thread
5. creating data stream
6. reading and processing request
7. sending the result
8. closing client network connection
9. go to step 3

Exercise 6 Further exercises

- Add a Frame to the http server application. Have it display the incoming requests in a TextField.
- Add domain restriction capabilities to the server. The restriction.cfg text file consists a list of restricted html pages and the list of domains which can access these pages. Each page has a different line which consists the name of page and the list of permitted domains separated by space. For example, the example.html is accessible from only obuda.kando.hu and jozsef.kando.hu domain, so restriction.cfg has the following line: example.html obuda.kando.hu jozsef.kando.hu
  If a client request arrives, first the server queries the client's hostname and find out from restriction.cfg that the requested file is restricted or not. If the requested file is not restricted, the server passes the file to the client. If the requested file is restricted, the server checks the client's domain. If domain is allowed to access file, the server passes it otherwise the server passes error403.html and prints to display „403 Forbidden".

Lab 4: Writing connection-oriented client-server application

In this lab you will write connection-oriented client-server application and using a low-level network programming interface for communicating with TCP/IP over the network..
Estimated time to complete this lab: 120 minutes.

Objectives

After completing this lab, you will be able to:

- make a client-server application over TCP/IP network connection.

Prerequisites

Before working this lab,

- you should be familiar with IP addressing protocol (hostname, IP address, communication port, socket).
- you should be familiar with writing connection-oriented clients
- you should be familiar with writing connection-oriented servers

Exercises

Exercise 1 NetToe game

Write a client-server NetToe game which provides TicTacToe game for two player against each other over TCP/IP network connection. The program has two main part, the server and the client. The server handles the game. It controls moves and events and checks moves and result.
The server is available trough a specified TCP port on specified machine by TicTacToe client. The server communicates the clients through socket connection and prints information such as player connected, player move and so on, to standard output. Only two player can connect to server at same time and they can play with each other. If any additional player try to connect to server, he will be refused. After the game each player going to be disconnected and they have to connect again to play another game.
The first connected player's mark is 'X' and the second one's is 'O'. The game starts when both player ready for game, therefore both is connected. The game goes on 10x10 grid board and each player put down own mark to one free square in each round. The winner is who has five mark side by side down, across or diagonal. Player 'X' begins game after that they take turns at moving until someone win the game or the board is full. In second case the game is dawn.
Players use the NetToe Client to play game. It has a Graphical User Interface to handle user interactions. The user interface is a simple window and the window has four part. First is title field which shows the player's mark. Second is the connect field, where the user can give the machine name and TCP port number to access NetToe server. Third one is the board to show current score on board and it handles user's moves which are given by mouse. Fourth one is the information panel to show events and related information.

Step by step solution

define object and class hierarchy

NetToeServer

execute()	run the game
validMove()	check player's move
gameOver()	check the winner. It's true if someone win the game
isOccupied	check board's sqaure. It's true if square isn't empty.

Player

otherPlayerMoved()	send message
otherPlayerWin()	send message
run()	handle player's move

NetToeClient

execute()	run the game
processMessage()	process received messages
handle user input	

Solution
NetToeServer.java
NetToeClient.java

Exercise 2 Torpedo game

Write a client-server Torpedo game which provides Torpedo game for two player against each other over TCP/IP network connection. The program has two main part, the server and the client. The server handles the game. It controls moves and events and checks moves and result. The server is available trough a specified TCP port on specified machine by Torpedo client. The server communicates the clients through socket connection and prints information such as player connected, player move and so on, to standard output. Only two player can connect to server at same time and they can play with each other. If any additional player try to connect to server, he will be refused. After the game each player going to be disconnected and they have to connect again to play another game.
The game starts when both player ready for game, therefore both is connected and both put down all of their own ship. The game goes on two 10x10 grid board. One is the first player's

board, another is the second player's board. Each player put down ships to own board. The players have the following ships:

amount of ship	size of ship*
1	5
2	4
3	3
3	2
2	1

* the unit of size is one square.

Players use the Torpedo Client to play game. It has a Graphical User Interface to handle user interactions. The user interface is a simple window and the window has four part. First one is the connect field, where the user can give the machine name and TCP port number to access Torpedo server. Second one is the player's own board to show current score and the shots of another player. Third one is the another player's board to show current score and it handles user's moves which are given by mouse. Fourth one is the information panel to show events and related information.

Exercise 3 Snail game

Write a client-server Snail game which provides Snail game for two player against each other over TCP/IP network connection. The program has two main part, the server and the client. The server handles the game. It controls moves and events and checks moves and result.
The server is available trough a specified TCP port on specified machine by Snail client. The server communicates the clients through socket connection and prints information such as player connected, player move and so on, to standard output. Only two player can connect to server at the same time and they can play with each other. If any additional player try to connect to server, he will be refused. After the game each player going to be disconnected and they have to connect again to play another game.
The game starts when both player ready for game, therefore both is connected. The game is very simple. Each player has a snail which is always in move. The snails are in same pitch and they can move only in four direction: left, right, up, down. The players have to control their own snail and avoid collision with each other and the edge of the pitch. In addition, the snails are increase their speed and their length in stated intervals. The winner is who doesn't collide.
Players use the Snail Client to play game. It has a Graphical User Interface to handle user interactions. The user interface is a simple window and the window has three part. First one the connect field, where the user can give the machine name and TCP port number to access Snail server. Second one is the pitch to show snails and it handles user's controls which are given by keyboard. Fourth one is the information panel to show events and related information.

Lab 5: Connectionless communication in Java

In this lab you will write connectionless networking application and using a low-level network programming interface for communicating with UDP/IP over the network..
Estimated time to complete this lab: 60 minutes.

Objectives

After completing this lab, you will be able to:

- make UDP network connection.
- send and receive data over the UDP connection.

Prerequisites

Before working this lab,

- you should be familiar with IP addressing protocol (hostname, IP address, communication port, socket).
- you should be familiar with UDP operational mechanism

Tools in Java for connectionless communication

Connectionless communication is supported by DatagramSocket and DatagramPacket classes in java.net package. DatagramSocket class provides communication with connectionless socket and DatagramPacket provides datapacket abstraction. These classes use the Unreliable Datagram Protocol (UDP) that makes no guarantees about delivery of packets or the order of delivered packets. The sender transmits UDP packets either and either it makes to receiver or it doesn't. UDP sockets are typically used in bandwidth-limited applications, where the overhead associated with resending packets is not tolerable.

Java's java.net.DatagramSocket class

```
Public class DatagramSocket extends Object {
        public DatagramSocket() throws SocketException;
        public DatagramSocket(int port) throws SocketException;
        public DatagramSocket(int port, InetAddress laddr) throws SocketException;
        public void send(DatagramPacket p) throws IOException;
        public synchronized void receive(DatagramPacket p) throws IOException;
        public InetAddress getLocalAddress();
        public int getLocalPort();
        public synchronized void setSoTimeout(int timeout) throws SocketException;
        public synchronized int getSoTimeout() throws SocketException;
        public void close();
}
```

Common programming errors

Any method in Java throws exception if an error occurs. Let us see the general exceptions what may are thrown when we try to create a datagram socket connection:

- java.lang.SecurityException: the program don't have a permission to create communication point.
- java.lang.IllegalArgumentException: the specified port must be between 0 and 65535 inclusive.
- java.net.SocketException: if the socket could not be opened, or the socket could not bind to the specified local port.

On created socket you can send and receive data with send() and receive() methods. These methods transmit a datagrampackets between communicating agents. If the received message is longer than the buffer length, the message is truncated.

Let us see the typical exceptions during communication:

- java.lang.SecurityException: the program don't have a permission to communicate.
- java.lang.IOException: any other error occurred during data transmission.
- java.lang.InterruptedException: the timeout of receiving expired. The value of timeout can be set by setSoTimeout() method.

Java's java.net.DatagramPacket class

```
public final class DatagramPacket extends Object {
        public DatagramPacket(byte ibuf[], int ilength);
        public DatagramPacket(byte ibuf[], int ilength, InetAddress iaddr, int iport);
        public synchronized InetAddress getAddress();
        public synchronized int getPort();
        public synchronized byte[] getData();
        public synchronized int getLength();
        public synchronized void setAddress(InetAddress iaddr);
        public synchronized void setPort(int iport);
        public synchronized void setData(byte ibuf[]);
        public synchronized void setLength(int ilength);
}
```

Exercises

Exercise 1 HeartBeat program

Write a simple client-server program which demonstrate the usage of datagram sockets. The client application sends message in datagram packet to the server when it starts and stops. The server application listens on UDP port 5000. If the server receives a datagram packet, it prints the message in the datagram packet to display. The programs don't have arguments.

Solution

HearBeat.java - the client applet
Pulse.java - the server application

Exercise 2 Echoing program

Write a client-server program which does the following. The program uses datagrams to send packets of information between a client application and server application. On the client

application, the user types a message into a textfield and press Enter. The message sent to the server in datagram packet. The server receives the packet and displays the information in the packet, then echoes the packet back to the client. When the client receives the packet, the client displays the information in the packet.

Exercise 3 Reliable communicator

Write Java class which provide a reliable, one-way communication over UDP. Therefore, it has a method for sending datagram packet which can realise that the packet is lost and it sends lost packet again. Otherwise it sends packets in order. It has a method for receiving datagram packet in order which sends acknowledgement to the sender and filtering the wrongly resent packets.

Lab 6: Using Java Remote Method Invocation

In this lab you will use Java RMI package for handling distributed objects. You will write distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts.

Estimated time to complete this lab: 60 minutes.

Objectives

After completing this lab, you will be able to:

- write interface for remote objects
- implement server application
- registering and using remote object
- develop Java-to-Java distributed applications

Prerequisites

Before working this lab,

- you should be familiar with RMI operational mechanism.
- you should be familiar with collaborative systems

Tools for Java RMI

The Remote Method Invocation is provided by java.rmi package, which includes three major sub-packages: java.rmi.dgc, java.rmi.registry, and java.rmi.server.
The java.rmi package contains the Remote interface as well as the Naming class and the RMISecurityManager class and number of basic RMI exception. These interfaces are used by both RMI clients and servers to define remote interfaces.
The java.rmi.registry package contains classes that provide an interface and implementation for the various elements of the RMI object registry.
The java.rmi.server package contains the classes used in server implementations of remote objects. The RemoteServer class acts as the base class for all RMI server objects. UnicastRemoteObject, the single subclass of RemoteServer, implements a non-persistent, point-to-point object communication scheme. In addition, the java.rmi.server package contains several Exception relevant to the server implementation of remote object.

Make an RMI application step by step

1. Define interface of remote object. The interface has to be written as extending of the java.rmi.Remote interface and all methods in the interface must be declared as throwing the java.rmi.RemoteException. In addition, the interface must be public. A remote object passed as an argument or return value (either directly or embedded within a local object) must be declared as the remote interface, not the implementation class.
2. Write the server implementation of the interface. The implementation object of interface extends the java.rmi.server.UnicastRemoteObject class.

3. Start RMI registry. The registry serves the role of the Object Manager and Naming service for distributed object system. You can start an RMI registry on a host by running the rmiregistry command. By default, the registry listens to port 1099 on the local host for connections, but you can specify any port for the registry.
4. Create client sub and server skeleton for object. First the interface and the server implementation are compiled into bytecodes using the javac compiler, like normal classes. Next, we have to generate the linkage from the client to the object implementation through the RMI registry. This is done using RMI stub compiler, rmic.
5. Finally, register an instance of our implementation on a remote server, and then look up the object on a client.

Exercises

Exercise 1 Distributed Hello World program

Write a distributed Hello World program which has two part, a RMI server application and a client applet. The applet make a remote method call to the server from which it was downloaded to retrieve the message "Hello World!". When the applet runs and the server is accessible, the message is displayed on the client.

Step by step solution:

1. Define a Remote Interface:

```
public interface Hello extends java.rmi.Remote {
        String sayHello() throws java.rmi.RemoteException;
}
```

2. Write a Server Implementation Class

To write a remote object, you write a class that implements one or more remote interfaces. The implementation class needs to:

- Specify the remote interface(s) being implemented.
- Define the constructor for the remote object.
- Provide implementations for the methods that can be invoked remotely.
- Create and install a security manager.
- Create one or more instances of a remote object.
- Register at least one of the remote objects with the RMI remote object registry, for bootstrapping purposes.

Source:

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

public class HelloImpl extends UnicastRemoteObject implements Hello
{
        private String name;
```

```
        public HelloImpl(String s) throws RemoteException {
                super();
                name = s;
        }

        public String sayHello() throws RemoteException {
                return  "Hello World!";
        }

        public static void main(String args[])
        {
                // Create and install a security manager
                System.setSecurityManager(new RMISecurityManager());
                try {
                        HelloImpl obj = new HelloImpl("HelloServer");
                        // Register object as HelloServer
                        Naming.rebind("HelloServer", obj);
                        System.out.println("HelloServer bound in registry");
                } catch (Exception e) {
                        System.out.println("HelloImpl err: " + e.getMessage());
                        e.printStackTrace();
                }
        }
}
```

3. Write a Client Applet that Uses the Remote Service

The applet part of the distributed Hello World example remotely invokes the HelloServer's sayHello method in order to get the string "Hello World!", which is displayed.

Source:

```
import java.awt.*;
import java.rmi.*;

public class HelloApplet extends java.applet.Applet {
        String message = "";
        public void init() {
                try {
                        Hello obj = (Hello)Naming.lookup("//" +getCodeBase().getHost() +
                        "/HelloServer");
                        message = obj.sayHello();
                } catch (Exception e) {
                System.out.println("HelloApplet exception: " + e.getMessage());
                e.printStackTrace();
                }
        }
```

```
        public void paint(Graphics g) {
                g.drawString(message, 25, 50);
        }
}
```

1.The applet first gets a reference to the "HelloServer" from the server's registry, constructing the URL by using the getCodeBase method in conjunction with the getHost method.
2.The applet remotely invokes the sayHello method of the HelloServer remote object and stores the return value from the call (the string "Hello World!") in a variable named message.
3.The applet invokes the paint method to draw the applet on the display, causing the string "Hello World!" to be displayed.

Exercise 2 Distributed event logger

Write a distributed Event Logger program which has two part, a RMI server application and a client applet. The applet make a remote method call to the server to note an event. The method has one String type argument which is the description of event. Otherwise, the client can make a remote method call to the server to messages on list of the server. The applet has two part. First one is used for making logs and second one for getting list from.

Step by step solution:

1. Define a Remote Interface:

```
public interface eEventLogger extends java.rmi.Remote {
        void makeNote(String note) throws java.rmi.RemoteException;
        String getList() throws java.rmi.RemoteException;
}
```

2. Write a Server Implementation Class

To write a remote object, you write a class that implements one or more remote interfaces.

Source:

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

public class EventLoggerImpl extends UnicastRemoteObject implements EventLogger
{
        private String name;

        public HelloImpl(String s) throws RemoteException {
                super();
                name = s;
        }


        public void makeNote(String note) throws RemoteException {
```

```
        }

        public String getList() throws RemoteException {
        }

        public static void main(String args[])
        {
                // Create and install a security manager
                System.setSecurityManager(new RMISecurityManager());
                try {
                        EventLoggerImpl obj = new EventLoggerImpl("EventLoggerServer");
                        // Register object as EventLoggerServer
                        Naming.rebind("EventLoggerServer", obj);
                        System.out.println("EventLoggerServer bound in registry");
                } catch (Exception e) {
                        System.out.println("EventLoggerImpl err: " + e.getMessage());
                        e.printStackTrace();
                }
        }
}
```

3. Write a Client Applet that Uses the Remote Service

The applet remotely invokes the EventLoggerServer's getList() and makeNote() methods.

Source:

```
import java.awt.*;
import java.rmi.*;

public class HelloApplet extends java.applet.Applet {
        String message = "";
        public void init() {
                try {
                        Hello obj = (Hello)Naming.lookup("//" +getCodeBase().getHost() +
                        "/HelloServer");
                        message = obj.sayHello();
                } catch (Exception e) {
                System.out.println("HelloApplet exception: " + e.getMessage());
                e.printStackTrace();
                }
        }

        public void paint(Graphics g) {
                g.drawString(message, 25, 50);
        }
}
```

Exercise 3 Chat system

Write a Java RMI based chat system. The users chatting with their chat client, which is receiving messages from remote chat clients and displaying them in a text window next to their name.

Exercise 4 Distributed white board

Write a Java RMI based distributed white board where users can draw in same board in the same time over network connection.

Appendixes

Find IP address and hostname of the local machine

myaddress.java

```
import java.net.*;

class myAddress {

  public static void main (String args[]) {

    try {
      InetAddress address = InetAddress.getLocalHost();
      System.out.println("Hello. My name is " +  address.getHostName() +
                         " and my IP adress is " + address.getHostAddress());
    }
    catch (UnknownHostException e) {
      System.out.println("I'm sorry. I don't know my own name and address.");
    }

  }
}
```

Find IP address or hostname like nslookup

nslookup.java

```
import java.net.*;
import java.io.*;

public class nslookup {


        public static void main (String args[]) {

        BufferedReader myInputStream =
        new BufferedReader(new InputStreamReader(System.in));
                System.out.println("Name Service lookup utility\r\n"+
                        "Enter names or IP addresses. Enter \"exit\" to quit.");

                while (true) {
                        String inputString;
                        try {
                                System.out.print("> ");
```

```
                        inputString = myInputStream.readLine();
                    }
                    catch (IOException e) {
                        break;
                    }
                    if (inputString.equals("exit")) break;
                    lookup(inputString);
                }

        } /* end main */


        private static void lookup(String s) {

        InetAddress address;

  // get the bytes of the IP address
  try {
    address = InetAddress.getByName(s);
  }
  catch (UnknownHostException ue) {
     System.out.println("** Can't find host " + s);
     return;
  }

  if (isHostname(s)) {
            System.out.println(" Address: "+address.getHostAddress());
  }
  else {  // this is an IP address
            System.out.println(" Name: "+address.getHostName());
  }

 }  // end lookup

        private static boolean isHostname(String s) {

            char[] ca = s.toCharArray();
            // if we see a character that is neither a digit nor a period
            // then s is probably a hostname
            for (int i = 0; i < ca.length; i++) {
                if (!Character.isDigit(ca[i]) &&
                    ca[i] != '.') return true;
            }

            // Everything was either a digit or a period
            // so s looks like an IP address in dotted quad format
            return false;

        }  // end isHostName
```

```
} // end nslookup
```

Command line Daytime client

daytimeClient.java

```
// java daytimeClient [hostname] [TCPport]
import java.net.*;
import java.io.*;


public class daytimeClient {
	static final int defaultPort = 13;

	public static void main(String[] args) {

	int portNumber;

	Socket clientSocket;
	BufferedReader timeStream;
	String hostName;

	switch(args.length) {
		case 1:	hostName = args[0];
			portNumber = defaultPort;//daytimePort;
		break;
		case 2:	hostName = args[0];
			portNumber = new Integer(args[1]).intValue();
		break;
		default:
			hostName = "localhost";
			portNumber = defaultPort;//daytimePort;
		}

		try {
			// make a connection with the server
			clientSocket = new Socket(hostName,portNumber);
			// make a datastream from the connection
			timeStream = new BufferedReader(new
			InputStreamReader(clientSocket.getInputStream()));
			// get data from the server
			String timeString = timeStream.readLine();
			System.out.println("It is " + timeString + " at " + hostName);
			// close datastream and network connection
			timeStream.close();
			clientSocket.close();
		}
```

```
                catch (UnknownHostException e) {
                        // unknown host error message
                    System.err.println(" Unknown host error");
                }
                catch (ConnectException e) {
                    System.out.println(" Service unavailable on port "+portNumber+"
of host
                                                "+hostName);
                }
                catch (IOException e) {
                    // IO error message
                System.err.println(" Communication error occured\r\n "+e);
                }

        }
}
```

Command line DateAthost client

dayAtHostClient.java

```
// java dateAtHostClient [hostname] [TCPport]
import java.net.*;
import java.io.*;
import java.util.*;


public class dateAtHostClient  {
        static final int defaultPort = 37;
        static final long offset = 2208988800L;
        // number of seconds since 1st January, 1970
        public static void main(String[] args) {

        int portNumber;

        Socket clientSocket;
        DataInputStream timeStream;
        String hostName;

        switch(args.length) {
                case 1:         hostName = args[0];
                                portNumber = defaultPort;//dateAtHost Port;
                break;
                case 2:         hostName = args[0];
                                portNumber = new Integer(args[1]).intValue();
                break;
                default:
                                hostName = "localhost";
```

```
                        portNumber = defaultPort;//dateAtHost Port;
                }

                try {
                        // make a connection with the server
                        clientSocket = new Socket(hostName,portNumber);
                        // make a datastream from the connection
                        timeStream = new
DataInputStream(clientSocket.getInputStream());
                        // get data from the server
                        int dateAtHost = timeStream.readInt() + (int)(1L<<32);
                        // readInt() reads unsigned int, that's why we have to add sign to
the value
                        new Date().setTime((dateAtHost-offset)*1000);
                        System.out.println("It is " + new Date().toString() + " at " +
hostName);
                        // close datastream and network connection
                        timeStream.close();
                        clientSocket.close();
                }
                catch (UnknownHostException e) {
                        // unknown host error message
                        System.err.println(" Unknown host error");
                }
                catch (ConnectException e) {
                        System.out.println(" Service unavailable on port "+portNumber+"
of host "+hostName);
                }
                catch (IOException e) {
                        // IO error message
                System.err.println(" Communication error occured\r\n "+e);
                }

        }
}
```

Command line echo client

echoClient.java

```
import java.net.*;
import java.io.*;

public class echoClient {
        final static int defaultPort=7;

        public static void main(String[] args) {
```

```
        Socket clientSocket;
        String hostName;
        int portNumber;
        BufferedReader theInput;
        PrintWriter theOutput;
        BufferedReader userInput;
        String inputString;

        switch(args.length) {
                case 1:         hostName = args[0];
                        portNumber = defaultPort;//echo Port;
                break;
                case 2:         hostName = args[0];
                                portNumber = new Integer(args[1]).intValue();
                break;
                default:
                        hostName = "localhost";
                        portNumber = defaultPort;//echo Port;
        }


        try {
                System.out.println("Client side Echo utility");
                clientSocket = new Socket(hostName,portNumber);
                theInput = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                theOutput = new PrintWriter(clientSocket.getOutputStream());
                userInput = new BufferedReader(new
InputStreamReader(System.in));
                System.out.println("Enter your text or put only \'.\' to quit.");
                while (true) {
                        inputString = userInput.readLine();
                        if (inputString.equals(".")) break;
                        theOutput.println(inputString);
                        System.out.println(theInput.readLine());
                }
        }  // end try
        catch (UnknownHostException e) {
                // unknown host error message
                System.err.println(" Unknown host error");
        }
        catch (ConnectException e) {
                System.out.println(" Service unavailable on port "+portNumber+"
of host "+hostName);
        }
        catch (IOException e) {
                // IO error message
        System.err.println(" Communication error occured\r\n "+e);
        }
```

```
  }

}
```

Command line finger client

fingerClient.java

```
import java.net.*;
import java.io.*;
import java.util.*;

public class fingerClient {

  static final int defaultPort = 79;

  public static void main(String[] args) {

    String hostName = "localhost";
        String userName = null;
        int portNumber = defaultPort;
    Socket clientSocket;
    BufferedReader fingerInput;
    PrintWriter fingerOutput;

    switch(args.length) {
                case 2:
                                portNumber = new Integer(args[1]).intValue();
            case 1:
                                StringTokenizer theQuery = new
StringTokenizer(args[0],"@",true);
                                String token;
                                switch(theQuery.countTokens()) {
                                        case 1:
                                                token = theQuery.nextToken();
                                                if(token.equals("@"))
                                                        userName = " ";
                                                else
                                                        userName = token;
                                                hostName = "localhost";
                                        break;
                                        case 2:
                                                token = theQuery.nextToken();
                                                if(token.equals("@")) {
                                                        userName = " ";
                                                        hostName = theQuery.nextToken();
                                                }
```

```
                                        else {
                                                userName = token;
                                                hostName = "localhost";
                                        }
                                break;
                                case 3:
                                        token = theQuery.nextToken();
                                        if(token.equals("@")) {
                                                System.out.println(" Sorry, I don't
understand your query.\r\n Please use \'user@hostname\' format.");
                                                System.exit(0);
                                        }
                                        else {
                                                userName = token;
                                                theQuery.nextToken();
                                                hostName = theQuery.nextToken();
                                        }
                                break;
                                default:
                                        System.out.println(" Sorry, I don't understand
your query.\r\n Please use \'user@hostname\' format.");
                                        System.exit(0);
                        }

                break;
                default:
                        hostName = "localhost";
                        userName = null;
                        portNumber = defaultPort; //fingerPort;
                }

   try {
     clientSocket = new Socket(hostName, portNumber);
     fingerOutput = new PrintWriter(clientSocket.getOutputStream(),true);
     fingerOutput.println(userName);
     fingerInput = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
      String s;
     while ((s = fingerInput.readLine()) != null) {
       System.out.println(s);
     }
   }
        catch (UnknownHostException e) {
                // unknown host error message
                System.err.println(" Unknown host error");
        }
        catch (ConnectException e) {
                System.out.println(" Service unavailable on port "+portNumber+" of
host "+hostName);
```

```
        }
        catch (IOException e) {
                // IO error message
        System.err.println(" Communication error occured\r\n "+e);
        }

  }

}
```

DayTime server

daytimeServer.java

```
import java.net.*;
import java.io.*;
import java.util.Date;

public class daytimeServer {

  public final static int daytimePort = 13;

  public static void main(String[] args) {

   ServerSocket theServerSocket;
   Socket theConnectionSocket;
   PrintWriter out;

   try {
     theServerSocket = new ServerSocket(daytimePort);
     System.out.println("Timeserver ready at port "+daytimePort);
          try {
       while (true) {
         theConnectionSocket = theServerSocket.accept();
             System.out.println("Request arrived!");
         out = new PrintWriter(theConnectionSocket.getOutputStream(),true);
         out.println(new Date());
         theConnectionSocket.close();
       }
     }
     catch (IOException e) {
       theServerSocket.close();
       System.err.println(e);
     }

   }
   catch (IOException e) {
     System.err.println(e);
```

```
  }

 }

}
```

DateAtHost server

dateAtHost.java

```
import java.net.*;
import java.io.*;
import java.util.Date;

public class dateAtHostServer {

        public final static int defaultPort = 37;
        static final long offset = 2208988800L;
        // number of seconds since 1st January, 1970


 public static void main(String[] args) {

  ServerSocket theServerSocket;
  Socket theConnectionSocket;
  DataOutputStream out;
  int dateInt;

  try {
    theServerSocket = new ServerSocket(defaultPort);
    System.out.println("dateAtHost server ready at port "+defaultPort);
        try {
     while (true) {
       theConnectionSocket = theServerSocket.accept();
                System.out.println("Request arrived!");
       out = new DataOutputStream(theConnectionSocket.getOutputStream());
       dateInt = (int)new Date().getTime() + (int)offset + (int)(1L<<32);
                out.write(dateInt);
      // theConnectionSocket.close();
     }
    }
    catch (IOException e) {
     theServerSocket.close();
     System.err.println(e);
    }

  }
  catch (IOException e) {
```

```
    System.err.println(e);
   }

  }

}
```

Echo server

echoServer.java

```
import java.net.*;
import java.io.*;


public class echoServer {

  public final static int echoPort = 7;

  public static void main(String[] args) {

   ServerSocket theServerSocket;
   Socket theConnectionSocket;
   BufferedReader in;
   PrintWriter out;

   try {
    theServerSocket = new ServerSocket(echoPort);
    System.out.println("EchoServer ready at port "+ echoPort);

      while (true) {


                                        theConnectionSocket = theServerSocket.accept();
                        try {
                                        System.out.println("Request arrived!");
                                        in  = new BufferedReader(new
InputStreamReader(theConnectionSocket.getInputStream()));
                                        out = new
PrintWriter(theConnectionSocket.getOutputStream(),true);
                                        while(true) {
                                                String readText = in.readLine();
                                                out.println(readText);
                                        }
                        }
                        catch (IOException e) {
                                theConnectionSocket.close();
                        }
```

```
      }



   }
   catch (IOException e) {
     System.err.println(e);
   }

  }

}
```

NetToe game

NetToeServer.java

```
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class NetToeClient extends Frame implements
MouseListener,WindowListener, ActionListener {
        private final int boardSize = 10;
        private Square board[][];
        private Panel boardPanel,mainPanel;
        private TextArea display;
        private TextField id;
        private Socket clientsock;
        private BufferedReader input;
        private PrintWriter output;
        char myMark;
        private Square currentSquare;
        boolean isMyMoveNext=false;
        Panel enterPanel;
        Panel headPanel;
        Label enterLabel;
        TextField enter;
        private String hostName;
        private final int portNumber=5000;
        private boolean hostIsKnown;

        public NetToeClient () {
                super("Net-Toe client");
                headPanel = new Panel();
                headPanel.setLayout(new BorderLayout());
```

```
                id = new TextField();
                id.setEditable(false);
                headPanel.add("North",id);

                enterPanel = new Panel();
                enterLabel = new Label("Enter server:");
                enter = new TextField(20);
                enterPanel.add(enterLabel);
                enterPanel.add(enter);
                headPanel.add("South",enterPanel);
                add("North",headPanel);
                display = new TextArea(4,30);
                display.setEditable(false);
                add("South",display);

                boardPanel = new Panel();
                boardPanel.setLayout(new GridLayout(boardSize,boardSize,0,0));
                board = new Square[boardSize][boardSize];
                for(int row=0;row<board.length;row++)
                        for(int col=0;col<board.length;col++) {
                                board[row][col]=new Square();
                                board[row][col].addMouseListener(this);
                                boardPanel.add(board[row][col]);
                        }


                mainPanel = new Panel();
                mainPanel.add(boardPanel);
                add("Center",mainPanel);
                setSize((boardSize+2)*30,(boardSize+2)*30+140);
                setResizable(false);
                addWindowListener(this);
                setVisible(true);
                enter.addActionListener(this);

                }

        public void execute() {
                //create connection
                if(hostName == null)
                        display.append("Please enter server's host...\n");
                while(hostName == null);
                display.append("Connect to server in progress...\n");
                while(clientsock == null) {
                        try {
                                clientsock = new Socket(hostName,portNumber);
                                input = new BufferedReader(new
InputStreamReader(clientsock.getInputStream()));
```

```
                        output = new
PrintWriter(clientsock.getOutputStream(),true);
                    }catch(ConnectException e) {
                        display.append("Server unavailable.\n");
                        try {
                            Thread.sleep(3000);
                        }
                        catch (InterruptedException ie) { };
                        display.append("Trying to connect again.\n");
                    }
                    catch(IOException e) {
                        System.out.println(e);
                    }
                }
                //first get myMark
                try {
                    myMark = input.readLine().charAt(0);
                    id.setText("You are player \""+myMark+"\"");
                }catch(IOException e) {
                    System.out.println(e);
                }
                while(true) {
                    try {
                        String s = input.readLine();
                        processMessage(s);
                    } catch(IOException e) {}

                }
        }


        public void processMessage(String s) {
                if(s.equals("Valid move.")) {
                    display.append("Valid move, please wait.\n");
                    currentSquare.setMark(myMark);
                    currentSquare.repaint();
                    isMyMoveNext=false;
                } else if(s.equals("Invalid move, try again.")) {
                    display.append(s+"\n");
                } else if(s.equals("Oponent moved.")) {
                    try {
                        StringTokenizer move = new
StringTokenizer(input.readLine());
                        int row = Integer.parseInt(move.nextToken());
                        int col = Integer.parseInt(move.nextToken());
                        board[row][col].setMark((myMark=='X' ? 'O':'X'));
                        board[row][col].repaint();
                        display.append("Opponent moved. Your turn.");
                        isMyMoveNext = true;
```

```
                    } catch(IOException e) { }
                } else if(s.equals("Other player connected. Your move.")) {
                        isMyMoveNext = true;
                        display.append(s+"\n");
                }else if(s.equals("Other player win.")) {
                        display.append(s+"\n");
                        isMyMoveNext=false;
                } else if(s.equals("You win. Congratulations.")) {
                        display.append(s+"\n");
                        isMyMoveNext=false;
                }

        }

        public static void main(String args[]) {
                NetToeClient game = new NetToeClient();
                game.execute();
        }
        //**------------------------
        public void mouseClicked(MouseEvent e) {
                if(isMyMoveNext) {
                        for(int row=0;row<board.length;row++)
                                for(int col=0;col<board.length;col++) {
                                        if(e.getSource()==board[row][col]) {
                                                currentSquare = board[row][col];
                                                output.println(String.valueOf(row)+"
"+String.valueOf(col));
                                        }
                                }
                } else
                        display.append("Please wait with your move.\n");
        }

        public void mousePressed(MouseEvent e) {
        }

        public void mouseReleased(MouseEvent e) {
        }

        public void mouseEntered(MouseEvent e) {
        }

        public void mouseExited(MouseEvent e) {
        }

//**---------------------------------------------
        public void windowOpened(WindowEvent e) {

        }
```

```
        public void windowClosing(WindowEvent e) {
                setVisible(false);
                dispose();
                System.exit(0);
        }
        public void windowClosed(WindowEvent e) {
        }
        public void windowDeiconified(WindowEvent e) {
        }
        public void windowIconified(WindowEvent e) {
        }
        public void windowActivated(WindowEvent e) {
        }
        public void windowDeactivated(WindowEvent e) {
        }

        //---------------------------------------
        public void actionPerformed(ActionEvent e) {
                hostName = enter.getText();
        }

        //**************************
        class Square extends Canvas {
                char mark=' ';
                public Square() {
                        setSize(30,30);
                }
                public void setMark(char c) {
                        mark = c;
                }
                public void paint(Graphics g) {
                        g.drawRect(0,0,29,29);
                        g.drawString(String.valueOf(mark),11,20);
                }

        }

}
```

NetToeClient.java

```
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;


public class NetToeServer extends Frame implements WindowListener {
```

```
private final int boardSize = 10;
private final int maxPlayers = 2;
private char board[][];
private Player players[];
private int numberOfPlayers;
private int currentPlayer;
private ServerSocket serversock;
private TextArea output;
final int portNumber=5000;


//---------------------------------------
public NetToeServer () {
        super(" Net-Toe Server ");
        board = new char[boardSize][boardSize];
        players = new Player[maxPlayers];
        numberOfPlayers = 0;
        currentPlayer = 0;

        //create a server socket
        try {
                serversock = new ServerSocket(portNumber,maxPlayers);
        }
        catch(SocketException se) {
                System.out.println(se);
                System.exit(1);
        }
        catch(IOException ie) {
                System.out.println(ie);
                System.exit(1);
        }

        // create GUI
        output = new TextArea();
        add("Center",output);
        setSize(300,300);
        addWindowListener(this);
        setVisible(true);

}

//---------------------------------------
public void execute() {
        for(int i = 0;i<maxPlayers;i++) {
                try {
                        Socket myClient = serversock.accept();
                        players[numberOfPlayers]= new Player(myClient,this,i);
                        players[numberOfPlayers].start();
                        numberOfPlayers++;
```

```
                    }
                    catch(IOException ie)
                    {
                        System.out.println(ie);
                        System.exit(1);
                    }
                }
            }

        //---------------------------------------
        public void display(String s) {
                output.append(s+"\n");
            }

            //---------------------------------------
            public synchronized boolean validMove(int row, int col, int player) {

                while ( player != currentPlayer) {
                    try {
                        wait();
                    }
                    catch(InterruptedException e) {
                    }
                }

                if(! isOccupied(row,col)) {
                    board[row][col]=(currentPlayer==0 ? 'X' : 'O');
                    currentPlayer = ++currentPlayer % maxPlayers;
                    players[currentPlayer].otherPlayerMoved(row,col);
                    notify();
                    return true;
                }
                else
                    return false;
            }

            //-----------------------------------------
            public int getNumberOfPlayers() {
                return numberOfPlayers;
            }
            //---------------------------------------
            public boolean isOccupied(int row,int col) {
                if(board[row][col]=='X' || board[row][col]=='O')
                    return true;
                else
                    return false;
            }

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

```
public boolean gameOver(int row, int col,char mark) {
        int firstrow, firstcol,lastrow,lastcol;
        int hornum = 0;
        int vernum = 0;
        int dianum1 = 0;
        int dianum2 = 0;
        firstrow = row<5 ? 0 : row-5;
        firstcol = col<5 ? 0 : col-5;
        lastrow = boardSize>row+5 ? row+5 : boardSize-1;
        lastcol = boardSize>col+5 ? col+5 : boardSize-1;
        for(int i=firstrow;i<lastrow;i++) {
                if(board[i][col]==mark) vernum++;
                else if(vernum<5) vernum=0;
        }
        for(int i=firstcol;i<lastcol;i++) {
                if(board[row][i]==mark) hornum++;
                else if(hornum<5) hornum=0;
        }

        for(int i=firstcol,j=firstrow;i<lastcol && j<lastrow;i++,j++) {
                if(board[j][i]==mark) dianum1++;
                else if(dianum1<5) dianum1=0;
        }
        for(int i=lastcol,j=lastrow;i<firstcol && j<firstrow;i--,j--) {
                        if(board[j][i]==mark) dianum2++;
                        else if(dianum2<5)dianum2=0;
        }
        if( hornum>=5 || vernum>=5 || dianum1>=5 || dianum2>=5) {
                players[currentPlayer].otherPlayerWin();
                return true;
        }
        else
                return false;
}

//----------------------------------------
public static void main(String args[]) {

        NetToeServer game = new NetToeServer();
        game.execute();
}

//**-----------------------------------------------
public void windowOpened(WindowEvent e) {

}
public void windowClosing(WindowEvent e) {
        setVisible(false);
        dispose();
```

```
                System.exit(0);
        }
        public void windowClosed(WindowEvent e) {
        }
        public void windowDeiconified(WindowEvent e) {
        }
        public void windowIconified(WindowEvent e) {
        }
        public void windowActivated(WindowEvent e) {
        }
        public void windowDeactivated(WindowEvent e) {
        }

}

//*************************************************
class Player extends Thread {
        final int maxPlayers = 2;
        Socket clientsock;
        BufferedReader input;
        PrintWriter output;
        NetToeServer server;
        char mark;
        int number;

        //-----------------------------------------
        public Player(Socket s, NetToeServer n, int num) {
                mark = (num==0 ? 'X' : 'O');
                clientsock = s;
                server = n;
                number = num;
                try {
                        input = new BufferedReader(new
InputStreamReader(clientsock.getInputStream()));
                        output = new PrintWriter(clientsock.getOutputStream(),true);
                }
                catch (UnknownHostException e) {
                System.err.println(e);
                }
                catch(IOException e)
                {
                        System.out.println(e);
                }
        }

        //-----------------------------------------
        public void otherPlayerMoved(int row, int col) {
                        output.println("Oponent moved.");
                        output.println(String.valueOf(row)+" "+String.valueOf(col));
```

```
        }

        //-----------------------------------------
        public void otherPlayerWin() {
                        output.println("Other player win.");
        }

        //-----------------------------------------
        public void run() {
                boolean gameEnded = false;

                        server.display("Player "+mark+" connected.");
                        output.println(String.valueOf(mark));
                        output.println("Player "+(number == 0 ? "X connected":"O
connected, please wait"));

                        if(server.getNumberOfPlayers()<maxPlayers) {
                                output.println("Waiting for another player");
                                while(server.getNumberOfPlayers()<maxPlayers);
                  output.println("Other player connected. Your move.");
                        }

                // play game
                while (!gameEnded){
                        try {
                                StringTokenizer move = new
StringTokenizer(input.readLine());
                                int row = Integer.parseInt(move.nextToken());
                                int col = Integer.parseInt(move.nextToken());
                                if(server.validMove(row,col,number)) {
                                        server.display(mark+" move: "+row+","+col);
                                        output.println("Valid move.");
                                        if(server.gameOver(row,col,mark)) {
                                                gameEnded=true;
                                                server.display(mark+" player win");
                                                output.println("You win. Congratulations.");

                                        }

                                } else
                                        output.println("Invalid move, try again.");
                        } catch(IOException e) {
                                //server.display("What's up!");
                        }
                }
                try {
                clientsock.close();
                } catch(IOException e) { }
```

```
        }


}
```

HeartBeat program

HeartBeat.java

```
import java.net.*;
import java.io.*;
public class HeartBeat extends java.applet.Applet {
String myHost;
  public void init() {
    myHost = getCodeBase().getHost();
  }


  public void sendMessage( String message ) {
                try {
                  byte[] data = new byte[ message.length() ];
                  message.getBytes( 0, data.length, data, 0 );
                  InetAddress addr = InetAddress.getByName(myHost);
                  DatagramPacket pack = new DatagramPacket(data, data.length, addr,
1234);
                  DatagramSocket ds = new DatagramSocket();
                  ds.send( pack );
                  ds.close();
                }
                catch (IOException e) {
                        System.out.println( e );
                }
  }

  public void start() {
        sendMessage( "I have been started!" );
  }

  public void stop() {
        sendMessage( "I have been stopped!" );
  }
}
```

Pulse.java

```
import java.net.*;
import java.io.*;
public class Pulse {

        public static void main( String argv[] ) throws IOException {
                DatagramSocket s = new DatagramSocket( 1234 );
                System.out.println(" Waiting for heartbeats..");
                while (true) {
                        DatagramPacket packet = new DatagramPacket( new
byte[1024], 1024 );
                        s.receive( packet );
                        String message = new String(packet.getData(), 0, 0,
packet.getLength());
                        System.out.println( "Message from "+
                                packet.getAddress().getHostName()+" - "+message );
                }
        }
}
```

Echoing program

UDPClient.java

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;

public class UDPClient extends Frame implements WindowListener, ActionListener {
        TextField enter;
        TextArea display;
        Panel enterPanel;
        Label enterLabel;

        DatagramPacket sendPacket, receivePacket;
        DatagramSocket sendSocket, receiveSocket;

        public UDPClient() {
                super("UDPClient");
                enterPanel = new Panel();
                enterLabel = new Label("Enter message:");
                enter = new TextField(20);
                enterPanel.add(enterLabel);
                enterPanel.add(enter);
                add("North",enterPanel);
                display = new TextArea(20,10);
                add("Center",display);
```

```
            setSize(400,300);
            setVisible(true);

            try {
            sendSocket = new DatagramSocket();
                  receiveSocket = new DatagramSocket(5001);
            }
            catch(SocketException se) {
                  se.printStackTrace();
                  System.exit(1);
            }

            addWindowListener(this);
            enter.addActionListener(this);
      }

   public void waitForPackets() {

            while(true) {
                  try {
                  //set up packet with maximum possible size
                  byte array[] = new byte[100];
                        receivePacket = new DatagramPacket(array,array.length);

                        // wait for packet
                        receiveSocket.receive(receivePacket);

                        //process packet
                        display.append("\nPacket received:"+
                              "\nFrom host: "+receivePacket.getAddress() +
                              "\nHost port: "+receivePacket.getPort() +
                              "\nLength: "+receivePacket.getLength()+
                              "\nContaining:    ");
                  byte data[] = receivePacket.getData();
                        String received = new String(data);
                        display.append(" \n "+received);

                  }
                  catch (IOException ex) {
                  display.append("\n" + ex.toString()+"\n");
                        ex.printStackTrace();
                  }

            }
      }


//**-------------------------------------------
      public void windowOpened(WindowEvent e) {
```

```
        }
        public void windowClosing(WindowEvent e) {
                setVisible(false);
                dispose();
                System.exit(0);
        }
        public void windowClosed(WindowEvent e) {
        }
        public void windowDeiconified(WindowEvent e) {
        }
        public void windowIconified(WindowEvent e) {
        }
        public void windowActivated(WindowEvent e) {
        }
        public void windowDeactivated(WindowEvent e) {
        }

        /*-----*/
        public void actionPerformed(ActionEvent e) {
                try {
                        display.append("\n\nSending packet containing:\n"+
                                                enter.getText()+"\r\n");
                        String s = enter.getText();
                        byte data[] = new byte[100];
                        s.getBytes(0,s.length(),data,0);
                        sendPacket = new
DatagramPacket(data,s.length(),InetAddress.getLocalHost(),5000);
                        sendSocket.send(sendPacket);
                        display.append("\nPacket sent");
                }
                catch(IOException ex) {
                        display.append(ex.toString()+"\n");
                        ex.printStackTrace();
                }
        }

        public static void main(String Args[]){
                UDPClient c = new UDPClient();
                c.waitForPackets();
        }




}
```

UDPServer.java

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;

public class UDPServer extends Frame implements WindowListener {

        DatagramPacket sendPacket, receivePacket;
        DatagramSocket sendSocket, receiveSocket;

        TextArea display;

        public UDPServer() {
        super("UDPServer");
                display = new TextArea(20,10);
                add("Center",display);
                setSize(400,300);
                addWindowListener(this);
                setVisible(true);

                try {
                sendSocket = new DatagramSocket();
                        receiveSocket = new DatagramSocket(5000);
                }
                catch(SocketException se) {
                se.printStackTrace();
                        System.exit(1);
                }
        }

      public void waitForPackets() {

                while(true) {
                        try {
                        //set up packet with maximum possible size
                        byte array[] = new byte[100];
                                receivePacket = new DatagramPacket(array,array.length);

                                // wait for packet
                                receiveSocket.receive(receivePacket);

                                //process packet
                                display.append("\nPacket received:"+
                                        "\nFrom host: "+receivePacket.getAddress() +
                                        "\nHost port: "+receivePacket.getPort() +
                                        "\nLength: "+receivePacket.getLength()+
                                        "\nContaining:   ");
                        byte data[] = receivePacket.getData();
                                String received = new String(data);
```

```
                                display.append(" \n "+received);

                                // echo information from packet back to client
                                display.append("\n\nEcho data to client... ");
                                sendPacket=new
DatagramPacket(data,data.length,receivePacket.getAddress(),5001);
                                sendSocket.send(sendPacket);
                                display.append("\nPacket sent");

                        }
                  catch (IOException ex) {
                        display.append(ex.toString()+"\n");
                                ex.printStackTrace();
                        }

                }

        }
        //**-----------------------------------------------
        public void windowOpened(WindowEvent e) {

        }
        public void windowClosing(WindowEvent e) {
                setVisible(false);
                dispose();
                System.exit(0);
        }
        public void windowClosed(WindowEvent e) {
        }
        public void windowDeiconified(WindowEvent e) {
        }
        public void windowIconified(WindowEvent e) {
        }
        public void windowActivated(WindowEvent e) {
        }
        public void windowDeactivated(WindowEvent e) {
        }

        public static void main(String Args[]){
                UDPServer s = new UDPServer();
                s.waitForPackets();
        }
}
```

Distributed Hello World program

Hello.java

```
package rmiexample;
public interface Hello extends java.rmi.Remote {
   String sayHello() throws java.rmi.RemoteException;
}
```

HelloApplet.java

```
package rmiexample;
import java.awt.*;
import java.rmi.*;

public class HelloApplet extends java.applet.Applet {

   String message = "";

   public void init() {

   try {
        Hello obj = (Hello)
           Naming.lookup("//" + getCodeBase().getHost() + "/HelloServer");
        message = obj.sayHello();

        } catch (Exception e) {
           System.out.println("HelloApplet: an exception occurred:");
           e.printStackTrace();
        }
   }

   public void paint(Graphics g) {
        g.drawString(message, 25, 50);
   }

}
```

HelloImpl.java

```
package rmiexample;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl
        extends UnicastRemoteObject
        implements Hello
{
   private String name;

   public HelloImpl(String s) throws java.rmi.RemoteException {
        super();
        name = s;
```

```
    }

    public String sayHello() throws RemoteException {
        return  "Hello World!";
    }

    public static void main(String args[])
    {
        // Create and install the security manager
        System.setSecurityManager(new RMISecurityManager());

        try {
            HelloImpl obj = new HelloImpl("HelloServer");
            Naming.rebind("HelloServer", obj);
            System.out.println("HelloImpl created and bound in the registry to the name
HelloServer");

        } catch (Exception e) {
            System.out.println("HelloImpl.main: an exception occurred:");
            e.printStackTrace();
        }
    }
}
```