Chapter 20

NETWORKING

In this chapter, you will learn some basic concepts of networking using Java tools. Transfer of TCP/IP packets in the connection-oriented service and transfer of UDP packets in connectionless service are explained.

Java provides a package **java.net** to handle networking aspects. Most of the tools to deal with internet-related process are handled in this package. Java2 supports TCP/IP, TCP and UDP protocols. The classes in the **java.net** package are:

Authenticator	ServerSocket
ContentHandler	Socket
DatagramSocket	SocketImpl
DatagramSocketImpl	SocketPermission
HTTPURLConnection	URL
InetAddress	URLClassLoader
JarURLConnection	URLConnection
MulticastSocket	URLDecoder
NetPermission	URLEncoder
PasswordAuthentication	URLStreamHandler

The interfaces defined in **java.net** are:

ContentHandlerFactory
FileNameMap
SocketOptions
URLStreamHandlerFactory

20.1 InetAddress

Every computer connected to the internet has an address called IP(Internet Protocol) address. This address is necessary to establish a connection by a computer with another one. When messages are exchanged, the message packets contain the IP address. The IP address is a 32-bit number and is expressed as a group of four numbers separated by a dot. For example, 132.162.130.128 is an IP address. The IP address is handled in the class **InetAddress.** The methods defined in this class are :

boolean equals(Object ob)
Returns true if the object ob contains the same IP address as that of this object

byte[] getAddress()
Returns the raw IP address of this object; the result is in network byte order.

String getHostName()
Returns the host name for this IP address

String getHostAddress()
Returns the IP address in a string form

String toString()
Returns a string representation of this IP address

static InetAddress getByName(String host)
Returns the IP address for the given host; the host name can either be a machine name like “ruraluniv.org” or a string representing its IP address “205.31.45.101”. Throws UnknownHostException

static InetAddress[] getAllByName(String host)
Returns an array of all the IP addresses for the given host; the host name can either be a machine name or its IP address. Throws UnknownHostException

static InetAddress getLocalHost()
Returns the IP address of the local host; throws UnknownHostException

boolean isMulticastAddress()
Returns true, if this IP address is a multicast address, i.e., a class D address with first four bits as 1110

Remembering the IP address as a sequence of four numbers is difficult. Therefore, every IP address is associated with a string called domain name. For example, the IP address 192.18.97.71 is associated with www.java.sun.com and IP address 66.33.109.23 is associated with the name www.ruraluniv.org. A server keeps the database of IP addresses and the associated domain names. Such servers are called Domain Name System(DNS).

The program 20.1, illustrates some of the methods defined in InetAddress class:

Program 20.1

```
// This program illustrates the use of various
// methods in class InetAddress.
// somasundaramk@yahoo.com
import java.net.*;
class Inetadrs
    {
    public static void main(String args [])
        {
        InetAddress iphost, javasun, ruraluniv;
        try
           {
           javasun = InetAddress.getByName("java.sun.com");
           InetAddress [] yahoo =
                     InetAddress.getAllByName("yahoo.com");
           ruraluniv =
                   InetAddress.getByName("ruraluniv.org");
           System.out.println("IP address of java.sun.com
                                       = " + javasun);
           int yahoolen = yahoo.length;
           for (int i = 0; i < yahoolen; i++)
                System.out.println("IP address of
                     yahoo.com   = " + yahoo[i]);
           System.out.println("IP address of
                     ruraluniv.org   = " + ruraluniv);
           iphost = InetAddress.getLocalHost();
           System.out.println("Local host address = " +
                     iphost);
           System.out.println("Name of the host :" +
                     iphost.getHostName());
           }
        catch (UnknownHostException e)
            {
            System.out.println("Host address could not
                                        be found   " + e);
            }
        }
    }
```

The above program gives the following output:

```
IP address of java.sun.com      = java.sun.com/192.18.97.71
IP address of yahoo.com         = yahoo.com/66.218.71.198
IP address of ruraluniv.org     = ruraluniv.org/66.33.109.23
Local host address              = ks/202.9.169.168
Name of the host :ks
```

> ☝ The address of the computer connected to the internet is called IP (Internet Protocol) address and is 32-bit in length. This number is expressed as a group of four numbers separated by a dot. Example: 132.162.130.225.

20.2 Socket Programming

To establish a connection between one host and another in a network, the socket mechanism is used. A socket is used to connect the Java's I/O to other programs in the same machine or to another host on the network. Once a connection is established, higher-level protocols are used to do the required services. Each protocol is realized through a concept called port. A port is identified by a number that will automatically follow a predefined protocol. For example, port no. 21 is for FTP(File Transfer Protocol), 23 is for Telnet, 80 is for HTTP and so on.

> ☝ Port numbers are logical numbers that identify a particular protocol.

For the client-server system, the socket used for a server is handled by the class **ServerSocket** and that for client is handled by the class **Socket**.

20.2.1 ServerSocket(TCP/IP)

This class deals with the server sockets. A server socket keeps waiting for request calls from the network. When a request is received, it does some operation appropriate to the request and returns a reply to the caller. The request is processed by methods defined in **SocketImp1** class. The connection established is a connection–oriented and governed by TCP. It is a reliable service.

Constructors

The constructors used to create server socket are given below. All of them throw **IOException**.

ServerSocket(int port)
Creates a server socket on a specified port; a port of 0 creates a socket on any free port. The maximum queue length for incoming connection is set to 50.

ServerSocket(int port, int q)
Creates a server socket on the specified port; the maximum queue length is set to q.

ServerSocket(int port, int q, InetAddress address)
Creates a server socket on the specified port with a maximum queue length of q; in a multi-homed host, address specifies the IP address to which the socket binds.

Methods

Some of the methods defined in the **ServerSocket** are :

InetAddress getInetAddress()
Returns the local address of this server socket

int getLocalPort()
Returns the port number on which this socket is listening

Socket accept()
Listens for a connection to be made to this socket and accepts it; a new socket is created.
It throws an **IOException**.

void close()
Closes this socket; throws IOException if an I/O error occurs when closing the socket

void setSoTimeout(int timeout)
Enables/disables SO_TIMEOUT with the specified timeout in milliseconds; with a non-zero timeout, a call to accept() for this ServerSocket will block for only this much amount of time. After the expiry of the timeout, InterruptedIOException is thrown out. This method will also throw SocketException.

int getSoTimeout()
Returns the SO_TIMEOUT; a 0 value implies that the option is disabled.

20.2.2 Client Socket(TCP/IP)

The client socket is implemented in the class **Socket**. A socket is an end point for communication between two host machines. A client socket establishes a connection with the server socket. The connection is a reliable, connection-oriented TCP/IP connection. Once a connection is established, methods defined in **SocketImpl** class are used to carry out the desired task.

Constructors

Some of the constructors used to create client sockets are given below:

Socket(String host, int port)
Creates a stream socket and connects it to the specified port number on the specified host; throws an UnknownHostException and IOException

Socket(InetAddress address, int port)
Creates a stream socket and connects it to the specified port number at the specified IP address; throws an IOException

Socket(InetAddress address, int port, InetAddress localadrs, int localport)
Creates a socket and connects it to the specified remote address and remote port; the socket binds to the local address and the specified local port. Throws an IOException

Methods

Some of the methods defined in class **Socket** are given below. They are used to manage socket-related operations.

InetAddress getInetAddress()
Returns the remote IP address to which this socket is connected

InetAddress getLocalAddress()
Returns the local address to which the socket is connected

int getPort()
Returns the remote port number to which this socket is connected

int getLocalPort()
Returns the local port number to which this socket is connected

InputSream getInputStream()
Returns an input stream for reading bytes from this socket; throws an IOException

OutputStream getOutputStream()
Returns an output stream for writing bytes to this socket; throws an IOException

void close()
Closes this socket; throws an IOException.

20.2.3 Server Sending Message to Client

In the following program 20.2, a server socket is created in the local host (with IP address 127.0.0.1) with port number 95. The **accept()** method keeps listening for a client request. Once a client socket makes a request, it accepts and this process creates a socket in the server side. A connection is established with the client. After that, whatever is typed in the server side is communicated to the client socket. The program 20.3 illustrates a client socket. A client socket is created, connecting to the local host at port 95 acting as server. Once the client socket is connected to the server socket, a communication channel is created. The client side creates an input stream (output of server) and prints out whatever is available at the input. Both programs jointly create a one-way communication from server side to client side.

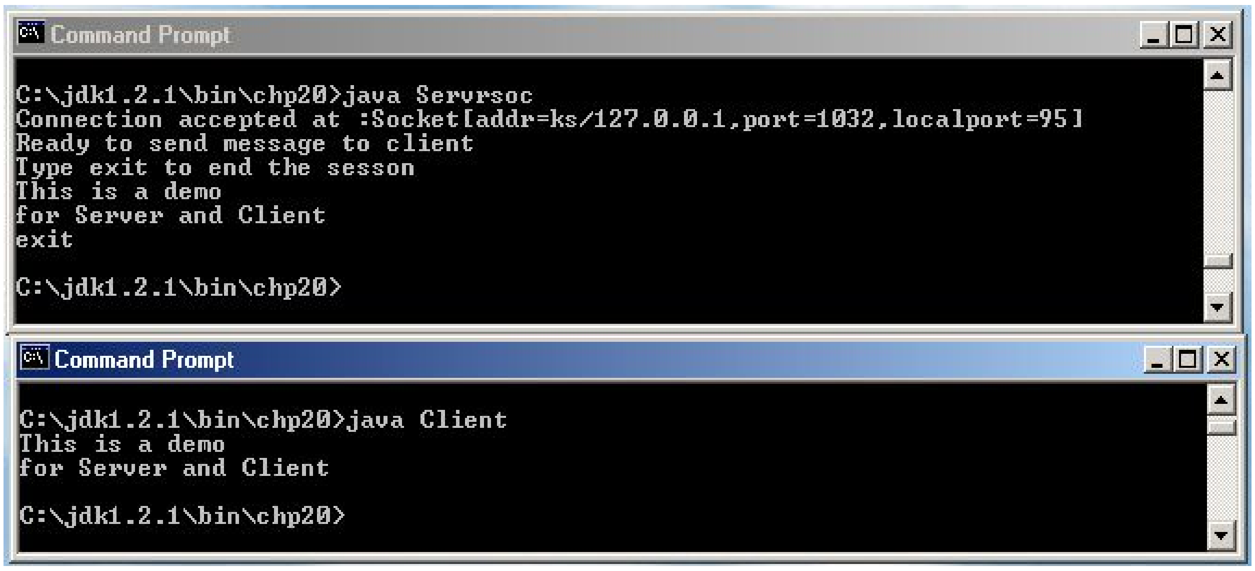
Program 20.2

```
// This program creates a server socket.
// somasundaramk@yahoo.com
import java.net.*;
import java.io.*;
class Servrsoc
    {
    public static void main(String args [])
        throws IOException
        {
        ServerSocket ss = null;
        try
            {
            ss = new ServerSocket(95);
            }
        catch (IOException ioe)
            {
            System.out.println("Error in finding the port
                                        95");
            System.exit(1);
            }
      Socket ssoc = null;
      ssoc = ss.accept();
      System.out.println("Connection accepted at :" + ssoc);
      DataOutputStream out = new
           DataOutputStream(ssoc.getOutputStream());
      BufferedReader inp = new BufferedReader(new
           InputStreamReader(ssoc.getInputStream()));
      String si, clstr;
      BufferedReader kybd = new BufferedReader(new
           InputStreamReader(System.in));
        System.out.println("Ready to send message to
             client\nType exit to end the sesson");

        // read text from keyboard
        while (!(si = kybd.readLine()).equals("exit"))
              {
              int ln = si.length();

              for (int i = 0; i < ln; i++)
             // send the text char by char to the client
                  out.write((byte)si.charAt(i));

              out.write(13);
              out.write(10);
              out.flush();
              }
          out.close();
          kybd.close();
          ss.close();
          }
      }
```

Program 20.3

```
// This program creates client socket.
// somasundaramk@yahoo.com

import java.net.*;
import java.io.*;
class Client
    {
    public static void main(String args [])
        throws IOException
      {
      Socket cls = null;
      BufferedReader br = null;
      try
           {
           cls = new Socket(InetAddress.getLocalHost(),95);
           br = new BufferedReader(new
                 InputStreamReader(cls.getInputStream()));
            }
        catch (UnknownHostException uh)
            {
             System.out.println("I dont know the host");
            System.exit(0);
            }
        String inp, si;
        while ((inp = br.readLine()) != null)
            {
            System.out.println(inp);
            }
            br.close();
            cls.close();
           }
        }
```

To execute the programs, first run the server program 20.2 in a DOS window. Then create another DOS window and run the client program 20.3. Then, whatever you type in the server side will be displayed in the client side window. The result for the above two programs executed simultaneously is given below:



```
C:\jdk1.2.1\bin\chp20>java Servrsoc
Connection accepted at :Socket[addr=ks/127.0.0.1,port=1032,localport=95]
Ready to send message to client
Type exit to end the sesson
This is a demo
for Server and Client
exit

C:\jdk1.2.1\bin\chp20>
```

```
C:\jdk1.2.1\bin\chp20>java Client
This is a demo
for Server and Client

C:\jdk1.2.1\bin\chp20>
```

Fig.20.1 Output Screens for Simultaneous Execution of Programs 20.2 and 20.3

Both programs terminate when the string exit is typed on the server side.

20.2.4 Echo-Server and Client

The program 20.4 is an echo server. It keeps waiting for a client to attach. It receives the message sent by the client and then retransmits the same message to the client and thus acts as an echo server.

Program 20.4

```
// This program creates a server socket.
// somasundaramk@yahoo.com

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

class Server2
    {
    public static void main(String args [])
        throws IOException
    {
    ServerSocket ss = null;
    try
        {
        ss = new ServerSocket(95);
        }
    catch (IOException ioe)
        {
        System.out.println("Error in finding the
                                          port 95");
        System.exit(1);
        }
    Socket ssoc = null;
    try
        {
         ssoc = ss.accept();
         System.out.println("Connection accepted
                                at :"+ssoc);
        }
     catch (IOException ioe)
        {
        System.out.println("Server failed to
            accept :");
        System.exit(1);
        }
  DataOutputStream out = new
            DataOutputStream(ssoc.getOutputStream());
  BufferedReader inp = new BufferedReader(new
       InputStreamReader(ssoc.getInputStream()));
  String si;
  System.out.println("Server waiting for message
                         from the client");
  boolean quit = false;
  // read text from the client
  do
     {
       String mesg = "";
       si = inp.readLine();
       int ln = si.length();
       if (si.equals("exit"))
            quit = true;
       for (int i = 0; i < ln; i++)
            // send the text char by char to the client
           {
            mesg = mesg + si.charAt(i);
            out.write((byte)si.charAt(i));
           }
       System.out.println("From Client :" + mesg);
       out.write(13);
       out.write(10);
```

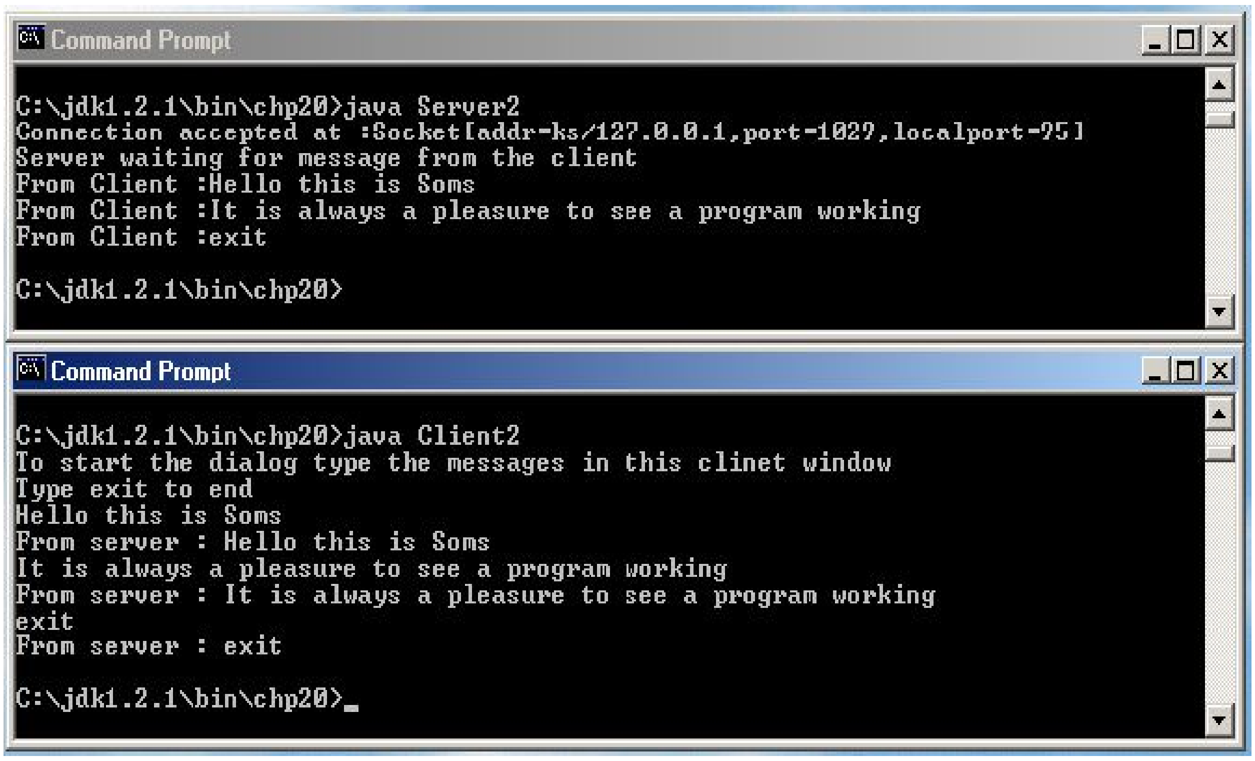
```
            out.flush();
            }while (!quit);
         out.close();
         ss.close();
        }
     }
```

Program 20.5 is a client. It connects to the echo server in the local host machine at port number 95. The client first accepts text typed through the keyboard and sends it to the echo server. It then receives the message back and prints out on the console. It again waits for keyboard input. This sending and receiving continues until the user type exit for which both server and client exit.

Program 20.5

```
/*This program creates a client socket. Whatever is typed is
sent to the server. The client then reads the message from
the server and prints the message.
 somasundaramk@yahoo.com
*/
import java.net.*;
import java.io.*;
class Client2
    {
    public static void main(String args [])
        throws IOException
       {
        Socket cls = null;
        String intext = null;
        BufferedReader input = null;
        DataOutputStream output = null;
        BufferedReader kybd = new BufferedReader(new
                     InputStreamReader(System.in));
        try
            {
             cls = new Socket(InetAddress.getLocalHost(),
                                           95);
             input = new BufferedReader(new
                InputStreamReader(cls.getInputStream()));
             output = new
                DataOutputStream(cls.getOutputStream());
            }
        catch (UnknownHostException uh)
            {
             System.out.println("Unknown host");
             System.exit(0);
            }
        System.out.println("To start the dialog type the
                messages in this client window\nType exit
                to end");
            boolean more = true;
            while (more)
                {
                //read message from the keyboard
                intext = kybd.readLine();
                //send message to server
               output.writeBytes(intext);
               output.write(13);
                output.write(10);
                output.flush();
               // receive message from server
               String inp, si;
               inp = input.readLine();
               System.out.println("From server : " + inp);
               if (inp.equals("exit"))
                    break;
               }
            input.close();
            output.close();
            cls.close();
            }
        }
```

First execute the echo server, program 20.4, in DOS window. Then execute the client program 20.5 in another DOS window. The dialog begins only from the client side. A sample output for executing both programs simultaneously is given in fig.20.2.



```
C:\jdk1.2.1\bin\chp20>java Server2
Connection accepted at :Socket[addr=ks/127.0.0.1,port=1029,localport=95]
Server waiting for message from the client
From Client :Hello this is Soms
From Client :It is always a pleasure to see a program working
From Client :exit

C:\jdk1.2.1\bin\chp20>
```

```
C:\jdk1.2.1\bin\chp20>java Client2
To start the dialog type the messages in this clinet window
Type exit to end
Hello this is Soms
From server : Hello this is Soms
It is always a pleasure to see a program working
From server : It is always a pleasure to see a program working
exit
From server : exit

C:\jdk1.2.1\bin\chp20>_
```

Fig. 20.2 Output Screens for Echo-Server (Program 20.4) and Client (Program 20.5)

20.3 Datagram

The mechanism seen in section 20.2 deals with exchange of message using connection–oriented TCP/IP protocol. Another way of sending and receiving message is by using datagrams. Datagrams are fire-and-forget type of message packets. They make use of connectionless UDP(User Datagram Protocol). The preparation of datagram packets are handled in **DatagramPacket** class and the delivery mechanism is handled by **DatagramSocket** class.

20.3.1 DatagramPacket

A message to be transmitted is broken into small message packets of a certain length. The IP address of the destination is also packed into it. The preparation of datagram packets are handled in the **Datagrampacket** class.

Constructors

Constructors in DatagramPacket are:

DatagramPacket(byte[] buf, int offset, int length)
: Constructs a datagram packet for receiving packets of byte length length, into the buffer buf starting at offeset in the array

DatagramPacket(byte[] buf, int length)
: Constructs a datagram packet for receiving packets of byte length of length

DatagramPacket(byte[] buff, int offset, int length, InetAddress address, int port)
: Constructs a datagram packet for sending packets of length, length with offset at offset, to the specified port on the specified host of address address

DatagramPacket(byte[] buff, 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 of address, address

Methods

Some of the methods defined in the **DatagramPacket** are:

InetAddress getAddress()
: Returns the IP address of the machine to which this datagram is being sent or from which the datagram is received

int getPort()
: Returns the port number on the remote host to which this datagram is being sent or from which the datagram is received

byte[] getData()
   Returns the data bytes received or the data bytes to be sent

int getLength()
   Returns the length of the data to be sent or the length of the data received

void setAddress(InetAddress address)
   Sets the IP address for this packet

void setPort(int port)
   Sets the port number for this packet

void setData(byte[] buf)
   Sets the data buffer for this packet

void setLength(int length)
   Sets the length in bytes for this packet

20.3.2 DatagramSocket

Datagram packets are transmitted or received through datagram socket. The **DatagramSocket** class implements server and client sockets using connectionless protocol UDP. The datagram packets may follow different routes to reach the destination. Multiple packets sent from a source may reach the destination in any order.

Constructors

Constructors in this class are:

DatagramSocket()
   Constructs a datagram socket and binds it to any available port on the local host machine; throws SocketException

DatagramSocket(int port)
   Constructs a datagram socket and binds it to the specified port on the local host machine; throws SocketException

DatagramSocket(int port, InetAddress ladrs)
   Creates a datagram socket and binds it to the specified local address and port

Methods

Some of the methods defined in this class for handling datagram socket are:

void connect(InetAddress adrs, int port)
  Connects the socket to a remote address and remote port; when a socket is connected to a remote address, packets may only be sent to or received from that address.

void disconnect()
  Disconnects the socket

InetAddress getInetAddress()
  Returns the address to which this socket is connected

int getPort()
  Returns the port number to which this socket is connected

void send(DatagramPacket p)
  Sends a datagram packet from this socket; throws an IOException

void receive(DatagramPacket p)
  Receives a datagram packet from this socket; when this method returns, the DatagramPacket's buffer is filled with the data received. Throws an IOException

InetAddress getLocalAddress()
  Returns the local address to which the socket is bound

int getLocalPort()
  Returns the port number on the local host to which this socket is bound

void close()
  Closes this socket

20.3.3 UDP Server-Client Conversation

UDP Server

Program 20.6 is a UDP-based server. It waits for the receipt of a UDP message packet. When it receives a message, it displays the message and waits for a keyboard input. The text typed is packed in a datagram and sent to a client in the local host with port number 3456. This process of receiving and sending continues until the message sent or received is "exit".

Program 20.6

```
/*
This program creates a datagram  socket. It makes use of
DatagramPacket and DatagramSocket classes and the methods defined
in them. This program acts like a server. It recevies message
packets from another client . The text typed in the keyboard is
```

```
sent as a reply datagram to client.
somasundaramk@yahoo.com
*/

import java.net.*;
import java.io.*;
class Dserver
    {
    public static void main(String args [])
        throws IOException
       {
        BufferedReader kybd = new BufferedReader(new
                          InputStreamReader(System.in));
        InetAddress localadrs =
                          InetAddress.getLocalHost();
        InetAddress remoteadrs = null;
        DatagramSocket dsocket = new DatagramSocket(3456);
        int buffersize = 2000;
        int remoteport;
        DatagramPacket outgram;
        System.out.println("Type replies.\nType exit to
                                    quit\n");
        boolean more = true;
        while (more)
           {
           byte [] inbuffer = new byte[buffersize];
           DatagramPacket ingram = new
                     DatagramPacket(inbuffer, buffersize);
           byte [] outbuffer = new byte[buffersize];
           dsocket.receive(ingram);
           remoteadrs = ingram.getAddress();
           remoteport = ingram.getPort();
           String data = new String(ingram.getData());
           data = data.trim();
           if (data.equals("exit"))
                break;
           System.out.println("From client : " + data);
           String reply = kybd.readLine();
           outbuffer = reply.getBytes();
           outgram = new DatagramPacket(outbuffer,
                outbuffer.length, remoteadrs, remoteport);
           dsocket.send(outgram);
           if (reply.equals("exit"))
                break;
           }
        dsocket.close();
        }
    }
```

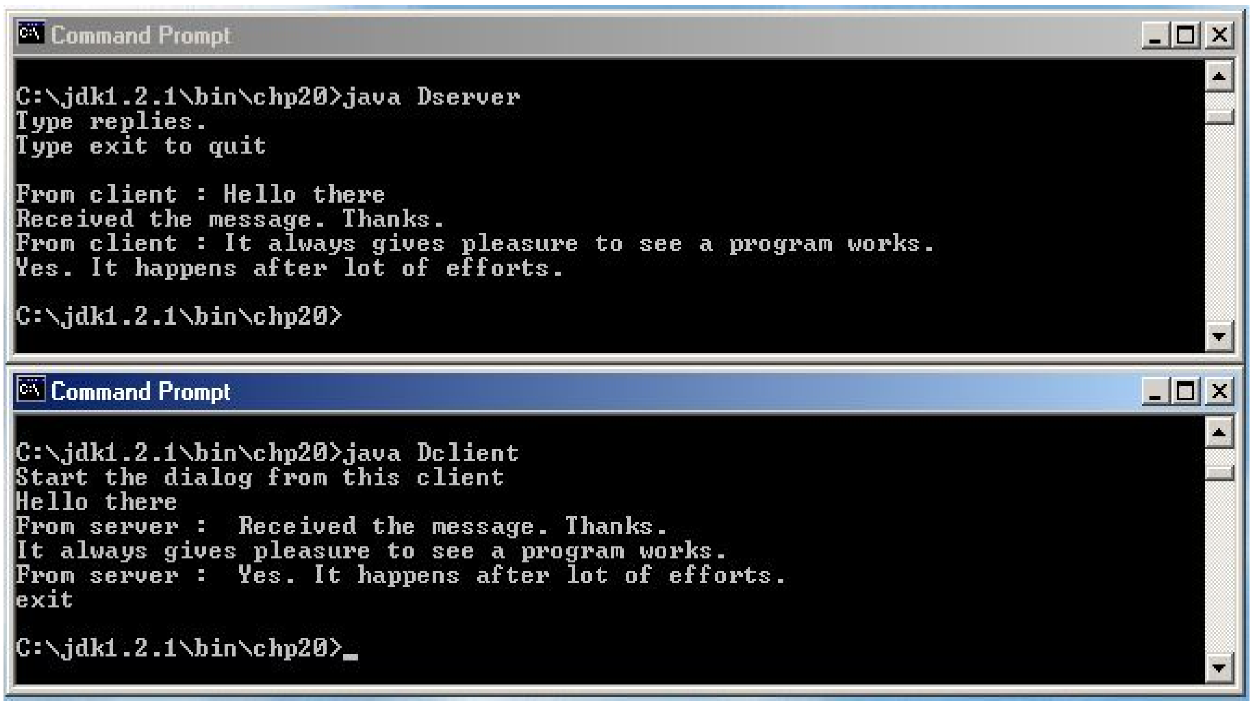
UDP Client

Program 20.7 is a UDP-based client. A socket with port number 3457 is created. It waits for the keyboard input. When a user types in a line of text, a UDP packet is prepared. The packet is then sent to the local host with port number 3456. It then waits for the arrival of a packet. When a packet arrives, the data in the packet is extracted and displayed on the monitor. This process of sending and receiving the message continues until the user types the word exit for which both the client and the server terminate.

Program 20.7

```
/*
This program creates a datagram  socket. It makes use of
DatagramPacket and DatagramSocket classes and the methods defined
in them. This program acts like a client. It sends message
packets to another server . The reply  received is printed out.
 somasundaramk@yahoo.com
*/
import java.net.*;
import java.io.*;
class Dclient
    {
    public static void main(String args [])
        throws IOException
       {
       BufferedReader kybd = new BufferedReader(new
                 InputStreamReader(System.in));
       InetAddress localadrs =
                 InetAddress.getLocalHost();
       InetAddress remoteadrs = localadrs;
       DatagramSocket dsocket = new
                              DatagramSocket(3457);
       int buffersize = 2000;
       byte [] outbuffer = new byte[buffersize];
       DatagramPacket outgram;
       System.out.println("Start the dialog from this
                              client");
       boolean more = true;
       while (more)
            {
            byte [] inbuffer = new byte[buffersize];
            DatagramPacket ingram = new
                      DatagramPacket(inbuffer, buffersize);
            String send = kybd.readLine();
            outbuffer = send.getBytes();
            outgram = new DatagramPacket(outbuffer,
                      outbuffer.length, remoteadrs, 3456);
            dsocket.send(outgram);
            if ((send.trim()).equals("exit"))
                 break;
            dsocket.receive(ingram);
            String data = new String(ingram.getData());
            data = data.trim();
            System.out.println("From server :  " + data);
            if (data.equals("exit"))
                 break;
            }
        dsocket.close();
        }
  }
```

Execute both server program 20.6 and client program 20.7 simultaneously, each on a separate DOS window. Figure 20.3 shows a sample output for executing the server and the client.



```
C:\jdk1.2.1\bin\chp20>java Dserver
Type replies.
Type exit to quit

From client : Hello there
Received the message. Thanks.
From client : It always gives pleasure to see a program works.
Yes. It happens after lot of efforts.

C:\jdk1.2.1\bin\chp20>
```

```
C:\jdk1.2.1\bin\chp20>java Dclient
Start the dialog from this client
Hello there
From server :  Received the message. Thanks.
It always gives pleasure to see a program works.
From server :  Yes. It happens after lot of efforts.
exit

C:\jdk1.2.1\bin\chp20>_
```

Fig.20.3 Sample Output for UDP-Based Server-Client Conversation (Programs 20.6 and 20.7)

20.4 URL

URL stands for Uniform Resource Locator. URL unifies many higher-level protocols and file formats into a single form called web or World Wide Web. Such a web is uniquely identified by an address specified by URL. All web browsers use URL to identify resources in the internet. Examples for URL are :

http://www.sun.java.com/
http://www.ruraluniv.org/
http://www.sun.java.com : 80/index.htm

The URL format consists of four parts. The first part indicates the protocol to be used. In the above examples http is the protocol. The protocol is delimited by the colon(:). The second part is the domain name enclosed between // and /, as in the first two examples. The third part is the port number, which by default is 80 for http and is optional. Therefore the first and the third examples have the same port number 80. The fourth part is the file name which appears after /.

20.4.1 URL Class

This class is used to create URL objects and has methods to process URL.

Constructors

Some of the commonly used constructors to create URL objects are :

URL(String spec)
: Creates a URL object from the string specification spec; throws MalformedURLException

URL(URL context, String spec)
: Creates a URL by parsing the specification spec within a specified context; if the context is not null and the spec argument is a partial URL specification, any of the string's missing components are inherited from the context argument. Throws MalformedURLException

Methods

Some of the methods defined in the URL class are:

int getPort()
: Returns port number of this URL; returns -1 if the port is not set

String getProtocol()
: Returns the protocol name of this URL

String getHost()
: Returns the host name of this URL, if applicable; for "file" protocol, this is an empty string.

String getFile()
: Returns the file name of this URL

URLConnection openConnection()
: Returns a URLConnection object that represents a connection to the remote object referred to by the URL; throws an IOException

final InputStream openStream()

Opens a connection to this URL and returns an InputStream for reading from that connection; this method represents a short-hand method for openConnection.getInputstream(). Throws an IOException

final Object getContent()

Returns the contents of this URL; this is equivalent to openConnection.getContent(). Throws an IOException

The following program 20.8 illustrates the creation of a URL and the use of some of the methods defined in URL class. The URL of www.ruraluniv.org is created.

The **openStream()** method creates an input stream. Using that input stream, the contents of the index.html file of this URL is read character by character and printed out.

Program 20.8

```
/*
This program illustrates the creation of URL. The use of a few
methods in URL is given. An input stream is opened and the
contents of a file are read from the specified URL.
somasundaramk@yahoo.com
*/

import java.net.*;
import java.io.*;
class URLuse
    {
    public static void main(String args [])
      {
      // The following is the website of Gandhigram
      // Rural University
      String gri = “http://www.ruraluniv.ac.in/index.html”;
      URL myurl = null;
      InputStream inps;
      try
           {
           myurl = new URL(gri);
           String proto = myurl.getProtocol();
           int griport = myurl.getPort();
           System.out.println(“URL used : “ +
                                myurl.toString());
           System.out.println(“Protocol :” + proto);
           System.out.println(“Port used: “ + griport);
           System.out.println(“File name: “ +
                                myurl.getFile());
```

```
            Object obj = myurl.getContent();
            System.out.println("Content :" +
                                    obj.toString());
            // reads the index.html file and prints char
            // by char
            inps = myurl.openStream();
            System.out.println("\nContent of index.html
                                    file\n");
            int c;
            while ((c = inps.read()) != -1)
                 System.out.print((char)c);
            inps.close();
            }
         catch (MalformedURLException me)
             {
             System.out.println("URL cannot be created");
             }
         catch (IOException ioe)
             {
             System.out.println("IO Problem");
             }
         }
     }
```

20.4.2 URLConnection

This is an abstract class. This class is a superclass of all classes that represents an HTTP connection between an application and a web object represented by URL. Objects of this class can be used to read from and write to the URL resource. Methods in this can be used to study about the URL.

Constructor

The constructor in this class is :

URLConnection(URL url)
  Constructs a URL connection to the specified URL; however, connection to the object referenced by the URL is not created.

Methods

Some of the methods defined in this class are :

abstract void connect()
  Opens a communication link to this URL; throws an IOException

URL getURL()
  Returns the value of this URLConnection's URL field

int getContentLength()
: Returns the content length of the resource that this connection represents or -1 if the content length is not known

String getContentType()
: Returns the content type of the resource that the URL represent or null if not known

long getDate()
: Returns the sending date of the resource that the URL references; the value returned is in milliseconds since January 1970.

long getLastModified()
: Returns the date of the resource that this connection represents was last modified or 0 if not known

Object getContent()
: Retrieves the contents of this URL connection; throws an IOException

InputStream getInputStream()
: Returns an input stream that reads from this connection; throws an IOException

OutputStream getOutputStream()
: Returns an output stream that writes to this connection; throws an IOException

---

After reading this chapter, you should have learned the following:

➲ Obtianing IP address of URLs

➲ Socket programming for server

➲ Socket programming for clients

➲ Sending and receiving UDP-based datagrams

➲ Accessing URLs and their contents

In the next chapter, you will learn how to access databases through Java clients.

Exercise-20

I. Fill in the blanks

20.1 IP stands for ________ .

20.2 IP address is a sequence of _____ numbers seperated by a dot.

20.3 Every IP address is associated with a name called _________ .

20.4 DNS is an acronym for ---------------- .

20.5 TCP is a __________ oriented service.

20.6 UDP is a __________ service.

20.7 The very important aspect of a server socket is that it _________ for a request from a client.

20.8 ________ is a number that indicates the protocol to be followed during a transaction.

20.9 Sockets are _______ points of communication.

20.10 The preparation of a datagram is dealt by __________ class.

20.11 URL is an acronym for ____________ .

20.12 The first part of an URL indicates the __________ to be used.

II. Write programs for the following problems:

20.13 Write a program to find all the IP addresses and port numbers of the following domain names:

sify.com hotmail.com

20.14 Write a server and a client (TCP/IP) program such that message typed in server is displayed on a client and vice versa.

20.15 Write a program using UDP so as to transfer a text file from one host to another.

20.16 Write a program to download the following file:
http:://www.java.sun.com/downloads/index.html

* * * * * *