Network Programming

Java provides easy-to-use classes for network programming. These classes integrate nicely with the rest of the language.

Acronyms

IP	= Internet Protocol
TCP	= Transport Control Protocol
UDP	= User Datagram Protocol
URL	= Uniform Resource Locator
HTTP	= Hypertext Transfer Protocol
FTP	= File Transfer Protocol
SMTP	= Simple Mail Transfer Protocol
POP	= Post Office Protocol

Connections

A connection with a computer on the Internet is described by

1. A host or domain name (blue.weeg.uiowa.edu)

   or

   An IP address, a 32-bit integer written as four bytes (128.255.56.2), called dotted quad format.

2. A port number, a 16-bit integer (1≤port≤65535).

The Domain Naming Service (DNS) translates host names into IP addresses. DNS consists of a set of servers on the Internet that translate host names.

Ports

The ports from 1 to 1023 are reserved for well-known network services, for example,

7	echo	37	time
9	discard	43	whois
13	daytime	79	finger
21	ftp	80	http
25	smtp	110	pop3

Networking in Java

Java provides classes for programming network applications at a relatively high level.

We consider two groups of classes:

1. IP address class
2. TCP classes

InetAddress Class

Each object of class InetAddress encapsulates one IP address.

- InetAddress has no public constructors.
- Objects are created using class methods, called factory methods.

```
InetAddress InetAddress.getByName(String hostname)
                              throws UnknownHostException
```

Also

```
    InetAddress [] InetAddress.getAllByName(String hostname)
    InetAddress InetAddress.getLocalHost()
```

Example (in a **try** block)

```
InetAddress cs = InetAddress.getByName("john.cs.uiowa.edu");
```

Accessors

Instance methods:

cs.getHostName()	returns a String
cs.getAddress()	returns a byte array

Example: Find IP Addresses

List the IP addresses for cnn.com or some other host.

Recall that the bytes between 128 and 255 are negative integers as twos complement numbers.

```
import java.net.*;

public class Inet
{
    public static void main(String [] args)
                                throws UnknownHostException
    {
        InetAddress [] ia;
        if (args.length == 1)
            ia = InetAddress.getAllByName(args[0]);
        else
            ia = InetAddress.getAllByName("cnn.com");

        for (int k = 0; k < ia.length; k++)
        {
            System.out.println(ia[k]);
            System.out.println(ia[k].getHostName());
            byte [] bytes = ia[k].getAddress();
            for (int n=0; n<bytes.length; n++)
            {   int b = bytes[n];
                if (b<0) b = b+256;
                if (n>0) System.out.print(".");
                System.out.print(b);
            }
            System.out.println();
        }
    }
}
```

Negative bytes need to be massaged to put them into the range 0 to 255.

Example: Start with the byte -52

```
byte                                                  1100 1100
int      1111 1111 1111 1111 1111 1111 1100 1100
+256     0000 0000 0000 0000 0000 0001 0000 0000
-----------------------------------------------------------------
         0000 0000 0000 0000 0000 0000 1100 1100
```

which is the unsigned byte 204 as an int.

Note that the sign is extended when a byte is changed into an int.

Output

```
cnn.com/207.25.71.25            cnn.com/64.236.16.84
cnn.com                         cnn.com
207.25.71.25                    64.236.16.84
cnn.com/207.25.71.29            cnn.com/64.236.16.116
cnn.com                         cnn.com
207.25.71.29                    64.236.16.116
cnn.com/64.236.16.20            cnn.com/207.25.71.5
cnn.com                         cnn.com
64.236.16.20                    207.25.71.5
cnn.com/64.236.16.52            cnn.com/207.25.71.20
cnn.com                         cnn.com
64.236.16.52                    207.25.71.20
```

TCP/IP: Connection-Oriented Protocol

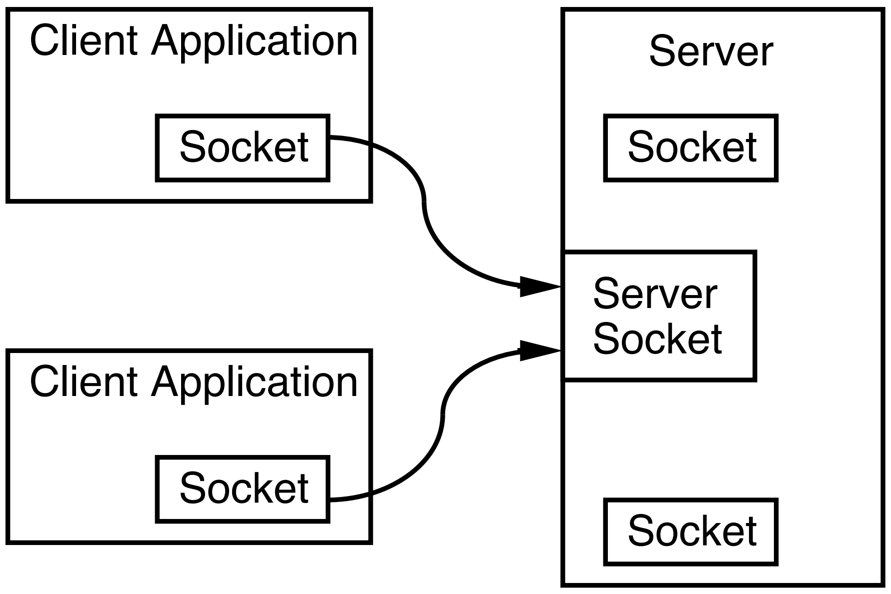
Similar to a telephone conversation.

Server

Listens at a particular port for incoming conversations.

Client

Initiates a conversation with a server by naming the server and the port at which the server is listening.



Sockets

A socket is an abstraction that treats a network connection as a stream of bytes that can be read or written.

A socket is constructed by a client explicitly when it connects with a server.

Socket Class

Constructors

Socket sock = **new** Socket("rome.divms.uiowa.edu", 7);

If we already have an InetAddress, say *inad*, use

Socket sk = **new** Socket(inad, 7);

Accessors

sock.getInetAddress()	returns remote host
sock.getPort()	returns remote port
sock.getLocalPort()	returns local port

Creating Streams

InputStream is = sock.getInputStream();

OutputStream os = sock.getOutputStream();

Closing a Socket

sock.close();

Example: Echo Client

Connect to the echo port on a unix machine.

```
import java.net.*;
import java.io.*;
public class EchoClient
{
   public static void main(String [] args)
   {
      String hostname;

      if (args.length > 0)
            hostname = args[0];
      else
            hostname = "localhost";

      try
      {
         Socket sock = new Socket(hostname, 7);

         System.out.println("Connected to " + sock.getInetAddress());

         BufferedReader inStream = new BufferedReader(
                  new InputStreamReader(sock.getInputStream()));

         boolean autoFlush = true;
         PrintWriter outStream = new PrintWriter(
                  new OutputStreamWriter(
                           sock.getOutputStream()), autoFlush);

         BufferedReader userInput = new BufferedReader(
                     new InputStreamReader(System.in));
```

```
        System.out.println("Enter text with . at end");
        String aLine = userInput.readLine();
        while (!aLine.equals("."))
        {
           System.out.println("User: " + aLine);
           outStream.println(aLine);
           System.out.println("Echo: " + inStream.readLine());
           aLine = userInput.readLine();
        }
        sock.close();
     }
     catch (UnknownHostException e)
     {  System.out.println(e);  }
     catch (IOException e)
     {  System.out.println(e);  }
  }
}
```

```
% java EchoClient cse.buffalo.edu
Connected to cse.buffalo.edu/128.205.32.2
Enter text with . to end
Hello buffalo.
User: Hello buffalo.
Echo: Hello buffalo.
How is Lake Erie?
User: How is Lake Erie?
Echo: How is Lake Erie?
This will be the last line.
User: This will be the last line.
Echo: This will be the last line.
.
```

Example: Look for Ports

Connect to a host and try all the ports from 1 to 1023 to see if they accept a connection. Use a command-line argument.

Try on *friendly* machines only.

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

public class Look4Ports
{
   public static void main(String [] args)
   {
      String host;
      Socket theSocket = null;

      if (args.length>0) host = args[0];
      else host = "localhost";

      try
      {
         for (int k=1; k<1024; k++)
         {
            try
            {
               theSocket = new Socket(host, k);
               System.out.println("Port " + k + " of "
                                    + host + " has a server.");
            }
            catch (ConnectException e)
            {
               /*  no server on this port:
                      throws a ConnectException  */
            }
```

```
                if (theSocket!=null)
                    theSocket.close();
            }
        }
        catch (UnknownHostException e)
        {  System.out.println(e);  }
        catch (IOException e)
        {  System.out.println(e);  }
    }
}
```

```
% java Look4Ports red.weeg.uiowa.edu
Port 21 of red.weeg.uiowa.edu has a server.
Port 22 of red.weeg.uiowa.edu has a server.
Port 23 of red.weeg.uiowa.edu has a server.
Port 25 of red.weeg.uiowa.edu has a server.
Port 106 of red.weeg.uiowa.edu has a server.
Port 110 of red.weeg.uiowa.edu has a server.
Port 111 of red.weeg.uiowa.edu has a server.
Port 113 of red.weeg.uiowa.edu has a server.
Port 512 of red.weeg.uiowa.edu has a server.
Port 513 of red.weeg.uiowa.edu has a server.
Port 514 of red.weeg.uiowa.edu has a server.
Port 620 of red.weeg.uiowa.edu has a server.
Port 621 of red.weeg.uiowa.edu has a server.
Port 999 of red.weeg.uiowa.edu has a server.
```

Server Sockets

Constructor

```
ServerSocket ss = new ServerSocket(2233);
        Parameter is an int specifying the port.
        The port number must be greater than 1023.
```

Instance Method

```
Socket sock = ss.accept();
```

Blocks until a client requests session with this host on server port.

Returns a Socket when a connection is made.

Example: Echo Server

Create a server that echoes each string that is sent to it, terminating when BYE is the string.

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

public class EchoServer
{
  public static void main(String [] args)
  {
    try
    {
      ServerSocket ss = new ServerSocket(7007);

      System.out.println("Echo Server running on " +
                 InetAddress.getLocalHost().getHostName());
```

```
         Socket incoming = ss.accept( );

         String client = incoming.getInetAddress().getHostName();

         System.out.println("Connection made with " + client);

         BufferedReader br = new BufferedReader(
            new InputStreamReader(incoming.getInputStream()));

         PrintWriter pw = new PrintWriter(
            new OutputStreamWriter(
                                 incoming.getOutputStream()), true);

         pw.println("Hello! Enter BYE to exit.\n");

         String str = br.readLine();

         while (str != null && !str.trim().equals("BYE"))
         {
            pw.println("Echo: " + str + "\n");
            str = br.readLine();
         }
         incoming.close();
      }
      catch (IOException e)
      {
         System.out.println(e);
      }
   }
}
```

The EchoServer can be tested using telnet.

A telnet session is initiated by a unix command:

```
% telnet  remoteMachineName  portNumber
```

First Start EchoServer on a machine (l-lnx205).

```
% java EchoServer
Echo Server running on l-lnx205.divms.uiowa.edu
Connection made with r-lnx233.cs.uiowa.edu
```

Second: Start telnet on r-lnx233.cs.uiowa.edu

```
% telnet l-lnx205.divms.uiowa.edu 7007
Trying...                                       ✓
Connected to l-lnx205.divms.uiowa.edu.          ✓
Escape character is '^]'.                       ✓
Hello! Enter BYE to exit.

Hello l-lnx205.
Echo: Hello l-lnx205.

This session comes from telnet.
Echo: This session comes from telnet.

BYE
Connection closed by foreign host.              ✓
```

✓ These lines produced by telnet.

Anonymous Mail

Using the simple mail transfer protocol, SMTP, we can send mail anonymously.

First, make a connection at port 25 with the server that contains the addressee to whom you want to send the mail and then send the following linefeed-terminated strings:

Client Messages	**Responses from Server**
<connect>	220 smtp.divms.uiowa.edu ESMTP Tue, 9 Nov 2004 12:49:34 -0600 (CST)
HELO sun.com	250 serv01.divms.uiowa.edu Hello [128.255.132.56], pleased to meet you
MAIL FROM: nobody@sun.com	250 nobody@sun.com... Sender ok
RCPT TO: slonnegr	250 slonnegr...Recipient ok
DATA	354 Enter mail, end with "." on a line by itself
Hello from nobody.	
.	250 PAA24437 Message accepted for delivery

POP3 Protocol

```
% telnet mail.divms.uiowa.edu 110
Trying...
Connected to mail.divms.uiowa.edu.
Escape character is '^]'.
+OK POP3 mail.divms.uiowa.edu v2001.78 server ready
USER slonnegr
+OK User name accepted, password please
PASS *********
+OK Mailbox open, 249 messages
RETR 20
+OK 2659 octets
        :
From: "stephenmosberg" <stephenmosberg@msn.com>
To: "Ken Slonneger" <slonnegr@cs.uiowa.edu>
Subject: Thank You
Date: Mon, 6 May 2002 09:44:45 -0400
        :
Dear Professor Slonneger:
I wanted to thank you for your helpful review of the proposal.
        :
.
QUIT
+OK Sayonara
Connection closed by foreign host.
```

Protocols and Serialization

For programs on two different machines to communicate, each needs to know what the other is expecting.

For example, email is possible because of the Simple Mail Transfer Protocol (smtp).

Suppose we want to provide a new service that clients can use, namely the generation of Domino objects.

First we need to define the protocol, call it the Domino Transfer Protocol (dtp).

DominoClient DominoServer

——————————— connect ——————————>

<—— "Enter number of dominoes desired" —

———————————— an int n ————————>

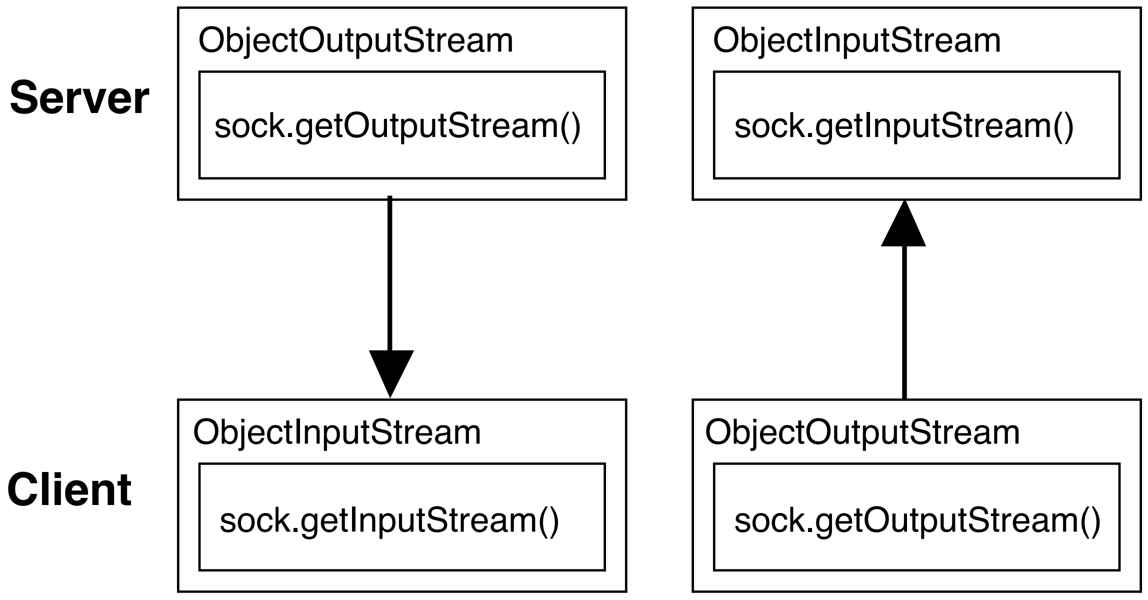
<—— n randomly generated dominoes ——

<————— terminate connection —————>

Since Domino objects need to be serialized to be sent across the network, all data will be sent through object streams.

When object streams are created between two programs, header information including magic numbers are exchanged and verified.

Therefore, the object streams must be created in compatible orders.



Domino Server

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

public class DominoServer
{
   public static void main(String[] args)
   {
      try
      {  ServerSocket ss = new ServerSocket(9876);
         System.out.println("Domino Server running.");
         System.out.println("Use cntl-c to stop server.");

         while (true)
         {
            Socket sock = ss.accept( );
            String client = sock.getInetAddress().getHostName();
            System.out.println("Connected to client " + client);
            InputStream in = sock.getInputStream();
            OutputStream out = sock.getOutputStream();
// Note order of Object stream creation.
            ObjectOutputStream outStrm =
                        new ObjectOutputStream(out);
            ObjectInputStream inStrm =
                        new ObjectInputStream(in);

            outStrm.writeUTF("Enter number of dominoes desired");
            outStrm.flush();

            int count = inStrm.readInt();

            for (int k=1; k<=count; k++)
            {
               Domino d = new Domino(true);
               outStrm.writeObject(d);
            }
            System.out.println(count + " dominoes sent to " + client);
            sock.close();              // flushes the stream
         }     // while (true)
      }
      catch (IOException e)
      {  System.out.println(e);  }
   }
}

/**************************************************/

class Domino implements Serializable
{
   :
}
```

Domino Client

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

public class DominoClient
{
   public static void main(String[] args)
   {
      String hostname;
      int port = 9876;

      if (args.length == 1)   hostname = args[0];
      else
      {  System.out.println(
                    "Usage: java DominoClient serverName");
         return;
      }

      try
      {
         Socket sock = new Socket(hostname, port);
         String server =
                    sock.getInetAddress().getHostName();
         System.out.println("Connected to server " + server);
         InputStream in = sock.getInputStream();
         OutputStream out = sock.getOutputStream();

// Note order of Object stream creation.

         ObjectInputStream inStrm =
                    new ObjectInputStream(in);

         ObjectOutputStream outStrm =
                    new ObjectOutputStream(out);

         Scanner scan = new Scanner(System.in);
         String start = inStrm.readUTF();

         System.out.print(start + ": ");

         String aLine = userInput.readLine();

         int num;
         try
         {
            num = scan.nextInt();
         }
         catch (InputMismatchException e)
         {  num = 0;  }

         outStrm.writeInt(num);
         outStrm.flush();

         for (int k=1; k<=num; k++)
         {
            Domino d = (Domino)inStrm.readObject();
            System.out.println("Domino " + k +": " + d);
         }
         sock.close();
      }
      catch (UnknownHostException e)
      {  System.out.println(e);  }
      catch (ClassNotFoundException e)
      {  System.out.println(e);  }
      catch (IOException e)
      {  System.out.println(e);  }
   }
}

/*************************************************/

class Domino implements Serializable
{
   :
}
```

Sample Server Output

```
% java DominoServer
Domino Server running on l-lnx205.divms.uiowa.edu
Use control-c to terminate server.
Connection from client: r-lnx233.cs.uiowa.edu
5 dominoes sent to r-lnx233.cs.uiowa.edu
```

Sample Client Output

```
% java DominoClient l-lnx205.divms.uiowa.edu
Connected to server l-lnx205.divms.uiowa.edu
Enter the number of dominoes desired: 5
Domino 1: <1, 1>  UP
Domino 2: <2, 3>  UP
Domino 3: <0, 0>  UP
Domino 4: <1, 4>  UP
Domino 5: <1, 1>  UP
```

Another Protocol

Most network protocols are text-based.

The client and server exchange text messages (strings).

Examples: echo, date, smtp, pop3, http

A Non-text Protocol: time

Some machines have a server running on port 37 that returns binary data.

When a client connects to such a server at port 37, the server returns a four-byte (32-bit) unsigned integer representing the number of seconds since midnight on January 1, 1900 GMT.

Since the number of seconds that have elapsed since January 1, 1900 GMT is already larger than the biggest positive **int** in Java (2,147,483,647), we must convert the four byte number to **long** to see the correct value.

Client Program

```
import java.net.*;
import java.io.*;
                                          // TCP to port 37
public class Time37
{
    public static void main(String [] args) throws IOException
    {
        String host;
        if (args.length == 1)
            host = args[0];
        else
            throw new RuntimeException(
                          "Usage: java Time37 time.nist.gov");
        Socket s = new Socket(host, 37);
        InputStream inStream = s.getInputStream();
        byte [] buffer = new byte [4];
        int count = inStream.read(buffer);
        long secs = 0;
        for (int k=0; k<4; k++)
        {                                 // use long arithmetic so that
            long b = buffer[k];           // after the shift left one byte,
            secs = secs << 8;             // the number will be positive
            secs = secs | (b & 0xff);     // place byte in right-most
        }                                 // position
```

```
            System.out.println(secs +
                                 " seconds since January 1, 1900, GMT");
        }
}
```

Sample Execution

```
  % java Time37 time.nist.gov
  3309015511 seconds since January 1, 1900, GMT
```

Exercise: Calculate the number of day, hours, minutes, and seconds that have elapsed since January 1, 1900 GMT.

Exercise: Use a DataInputStream to read the four bytes as an **int**. If the number is negative (it will be), use **long** arithmetic to make if positive by adding 2^{32}.

HTTP

The hypertext transport protocol defines the convention that allows a browser to find web pages on a server.

The basic web page fetch follows a simple protocol.

Protocol

1. Connect to the machine where the server in running, normally at port 80.

2a. Send a line of text that includes the name of the file to be returned.

```
GET  /index.html  HTTP/1.0
```

In most web directories, the file *index.html* contains the starting web page for the directory.

2b. Or omit the file name.

```
GET  /  HTTP/1.0
```

3. Send a header specifying the host (for HTTP 1.1):

   Host: *server we are connecting to*

4. Enter one entirely blank line.

At this point the web server should send back the web page as a stream of text, usually written in HTML.

The program below acts as a client that connects to a web server given by its first command-line argument, and downloads a file given by its second command-line argument.

If no command-line arguments are supplied, the program connects to *yahoo.com* and downloads *index.html*.

The program uses a Scanner object to read the lines of text sent back by the web server.

HttpClient.java

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

public class HttpClient
{
   public static void main(String [] args) throws IOException
   {
      String server="yahoo.com", file="index.html";
      if (args.length == 2) file = args[1];
      if (args.length >= 1) server = args[0];
      Socket sock = new Socket(server, 80);
      PrintWriter pw =
            new PrintWriter(sock.getOutputStream(),true);
```

```
        pw.println("GET  /" + file + "  HTTP/1.0");
        pw.println("Host: " + server);   // Required by HTTP 1.1
        pw.println();
        Scanner scan = new Scanner(sock.getInputStream());
        while (scan.hasNextLine())
        {
            String str = scan.nextLine();
            System.out.println(str);
        }
        sock.close();
    }
}
```

Execution

```
% java HttpClient
<!-- info_setup:ads,ad-location:Z,ad-spaceid:19849501 225 -->
<!-- info_insertion:Z 225 -->
<!-- spaceid:19849501 9764 -->
<html>
<head><title>Ask Yahoo!</title>
<meta http-equiv="pics-label" content='(pics-1.1
"http://www.icra.org/ratingsv02.html" l gen true for
"http://ask.yahoo.com" r ( nz 0 vz 0 lz 0 oz 0 ca 1))'>
</head>
<body bgcolor=white>
<center>
<table border=0 width=600 cellspacing=0 cellpadding=0>
  <tr>
   <td width=1%><img
src=http://us.i1.yimg.com/us.yimg.com/i/us/ask/gr/ask1.gif alt="Ask
Yahoo!" border=0 width=167 height=35></td>
   <td>
     <table border=0 cellspacing=0 cellpadding=0 width=100%>
      <tr>
        <td align=right valign=bottom><font face=arial size=-1>
```

```
<a href="http://rd.yahoo.com/ask/?http://dir.yahoo.com">Web Site
Directory</a> -
    :
```

Using Telnet

```
% telnet  java.sun.com  80
Trying 209.249.116.141...
Connected to 209.249.116.141.available (209.249.116.141).
Escape character is '^]'.
GET  /  HTTP/1.0
HTTP/1.1 200 OK
Server: Sun-ONE-Web-Server/6.1
Date: Thu, 12 May 2005 19:39:11 GMT
Content-type:  text/html;charset=ISO-8859-1
Set-Cookie: SUN_ID=128.255.44.205:255751115926751;
EXPIRES=Wednesday, 31-Dec-2025 23:59:59 GMT;
DOMAIN=.sun.com; PATH=/
Set-cookie:
JSESSIONID=514ED4E39D287F2DA0479869E05C1359;Path=/
Connection:  close

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Java Technology</title>
<meta name="keywords" content="Java, platform" />
<meta name="description" content="Java technology is a
portfolio of products that are based on the power of networks and
the idea that the same software should run on many different
kinds of systems and devices." />
   :
```