

# Network Programming

## SOCKET

### Networking with Java

#### Chapter Focus

- Basics of Networking
- **java.net** package
- **Socket** and **ServerSocket** classes
- **Objective**=>to write client-server internet/intranet programs

#### Basics of Networking

- How data is transmitted across the Internet?
  - Computers running on internet communicate to each other using **Transmission Control Protocol (TCP)** or **User Datagram Protocol (UDP)**.
- What is protocol?
  - Protocol is the special set of rules that endpoints in a telecommunication connection use when they communicate.

#### TCP

- Like a *telephone call*
- a **connection-based protocol**, guarantees that data sent from one end of the connection actually gets to the other end in the same order it was sent.
- works together with **InternetProtocol(IP)**.
- **TCP/IP** makes a reliable communication channel.
- Additional internet protocols are
  - HTTP(HypertextTransferProtocol)
  - FTP(FileTransferProtocol)

#### UDP

- Like a *postal service*.
- not a **connection-based protocol**.
- It sends independent packets of data, called **datagrams**, from one application to another.
- The order of delivery is not important and is not guaranteed, and each message is independent of any other.

#### Internet Addressing

- Every computer on the internet has an address, called IP Address.
- This address is a number that uniquely identifies each computer on the net.
- Example addresses and their name:

Name	Address
www.google.com	216.239.57.99
www.amazon.com	207.171.166.48

DNS – Domain Name Server

Name to Address Resolution is done by DNS Server.

## IP Address

- An **Internet Protocol address (IP address)** is a numerical label assigned to each device (e.g., computer, printer) participating in a computer network.
- **IP addresses** are *binary numbers*, but they are usually stored in text files and displayed in *human-readable notations*, such as 172.16.254.1 (for IPv4), and 2001:db8:0:1234:0:567:8:1 (for IPv6).

## Port

- A computer **port** is a type of electronic, software-or programming-related docking point through which information flows from a program on your computer or to your computer from the Internet or another computer in a network.
- Ports are numbered for consistency and programming.
- Ports are represented by a 16-bit number (0-65535), commonly known as the **port number**.

## Client-Server in Networking

- Involves two types of programs: *client* and *server*.
- **Server**: a program that provides services to one or more clients.
- **Client**: a program that makes a service request from server.
- Example → WorldWideWeb
  - *Server*: WebServer
  - *Client*: WebBrowsers

## Java Net API

**Question:** “Does Java support the functionality of networks?”

**Answer:** “Java Makes it easy to connect to other computers using classes from the **java.net** package.”

## Some classes in java.net package

- InetAddress
- Socket
- ServerSocket
- URI
- URL
- Datagram Packet
- Datagram Socket

## The InetAddress Class

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name for example www.javatpoint.com, www.google.com, www.facebook.com etc.

The java.net.InetAddress class is Java’s encapsulation of an IP address. It is used by most of the other networking classes, including Socket, ServerSocket, URL, DatagramSocket, DatagramPacket, and more.

This class represents an Internet address as two fields: hostName (a String) and address (an int). hostName contains the name of the host; for example, www.oreilly.com. address contains the 32-bit IP address. These fields are not public, so you can’t access them directly. It will probably be necessary to change this representation to a byte array when 16-byte IPv6 addresses come into use. However, if you always use the InetAddress class to represent addresses, the changeover should not affect you; the class shields you from the details of how addresses are implemented.

There are three types of Ethernet addresses:

- **unicast addresses** – represent a single LAN interface. A unicast frame will be sent to a specific device, not to a group of devices on the LAN.
- **multicast addresses** – represent a group of devices in a LAN. A frame sent to a multicast address will be forwarded to a group of devices on the LAN.
- **broadcast addresses** – represent all device on the LAN. Frames sent to a broadcast address will be delivered to all devices on the LAN.

The unicast address will have the value of the MAC address of the destination device.

Multicast frames have a value of 1 in the least-significant bit of the first octet of the destination address. This helps a network switch to distinguish between unicast and multicast addresses. One example of an Ethernet multicast address would be **01:00:0C:CC:CC:CC**, which is an address used by **CDP (Cisco Discovery Protocol)**.

The broadcast address has the value of **FFFF.FFFF.FFFF** (all binary ones). The switch will flood broadcast frames out all ports except the port that it was received on.

### Some methods of *InetAddress* class

- String getAddress()
- String getHostName()
- static InetAddress getLocalHost()
- static InetAddress getByName(String host)
- static InetAddress[] getAllByName(String host)
- static InetAddress getByAddress(String host, byte[] addr)
- boolean isLoopbackAddress()
- boolean isMulticastAddress()

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

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

```
            InetAddress ip=InetAddress.getByName("www.javatpoint.com");
```

```
            System.out.println("Host Name: "+ip.getHostName());
            System.out.println("IP Address: "+ip.getHostAddress());
```

```
        } catch (Exception e) {System.out.println(e);}
    }
}
```

```
Host Name: www.javatpoint.com
IP Address: 195.201.10.8
```

## Socket class

- Communication over the internet involves **sockets** for creating connections.
- Sockets connect to numbered communication ports. (The bottom 1024 ports are reserved for system use.)
- Java supports sockets with the **Socket** class.
  - `java.lang.Object`
  - `java.net.Socket`
- **java.net.Socket** is an abstraction of a bi-directional communication channel between hosts
- Send and receive data using streams



## Some Constructors of the *Socket* class

- **Socket()**
- **Socket(InetAddress address, int port)**
- **Socket(String host, int port)**

## Some methods of *Socket* class

- **void connect(SocketAddress endpoint)**
- **void bind(SocketAddress bindpoint)**
- **void close()**
- **boolean isBound()**
- **boolean isConnected()**
- **boolean isClosed()**
- **int getPort()**

## ServerSocket class

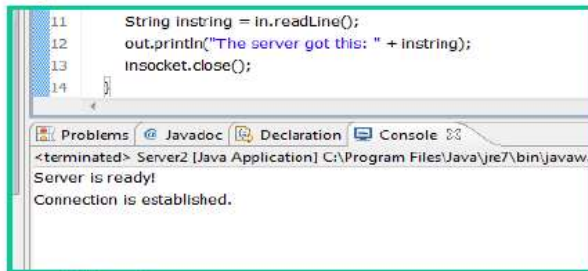
- used for creating server applications.
- create servers that listen for either local or remote client programs to connect to them on published ports.
- Constructors
  - **ServerSocket()**
  - **ServerSocket(int port)**
  - **ServerSocket(int port, int maxQueue)**
  - **ServerSocket(int port, int maxQueue, InetAddress localAddress)**

## Some methods of *ServerSocket* class

- **Socket accept()**,
  - which acts as a blocking call and waits for a client to initiate communication, and then return with a normal **Socket** that is then used for communication with client.
- **void bind(SocketAddress endpoint)**
- **void close()**
- **InetAddress getInetAddress()**
- **int getLocalPort()**

## Example

### 1. Running Server



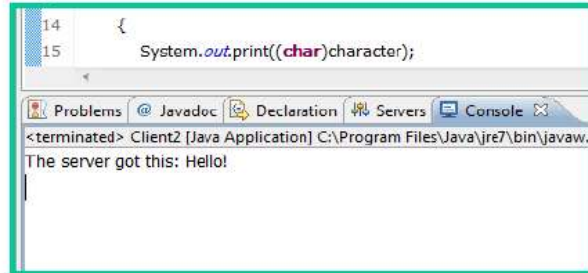
The screenshot shows an IDE with a code editor and a console window. The code editor displays the following Java code:

```
11 String instring = in.readLine();
12 out.println("The server got this: " + instring);
13 insocket.close();
14 }
```

The console window shows the following output:

```
<terminated> Server2 [Java Application] C:\Program Files\Java\jre7\bin\javaw.
Server is ready!
Connection is established.
```

### 2. Running Client



The screenshot shows an IDE with a code editor and a console window. The code editor displays the following Java code:

```
14 {
15     System.out.print((char)character);
}
```

The console window shows the following output:

```
<terminated> Client2 [Java Application] C:\Program Files\Java\jre7\bin\javaw.
The server got this: Hello!
```

## Example 1: Creating TCP/IP Server

```
import java.net.*;
import java.io.*;
public class Server2 {
    public static void main(String[] args) throws Exception {
        ServerSocket socket = new ServerSocket(8765);
        System.out.println("Server is ready!");
        Socket insocket = socket.accept();
        //blocks this invoking till connection is established
        System.out.println("Connection is established.");
        BufferedReader in = new BufferedReader(new InputStreamReader(insocket.getInputStream()));
        PrintWriter out = new PrintWriter (insocket.getOutputStream(), true);
        String instring = in.readLine();
        out.println("The server got this: " + instring);
        insocket.close();
    }
}
```

## Example 1: Creating TCP/IP Client

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

```

public static void main(String[] args) throws Exception {

    int character;

    Socket socket = new Socket("127.0.0.1", 8765); // server-address (or) name, port
    InputStream in = socket.getInputStream();
    OutputStream out = socket.getOutputStream();

    String string="Hello!\n";

    byte buffer[]=string.getBytes();

    out.write(buffer);

    while((character = in.read()) != -1)

        System.out.print((char)character);

    socket.close();

}

```

### } TCP/IP Client Sockets

- Client and Server computers establish their connection by means of a special object called a **socket**.
- How to open a socket:

```

Socket myClient;
try{
    myClient = new Socket("Server Name", PortNumber);
}catch(IOException e)
{System.out.println(e);}

```

- A **socket** can be used to connect *Java's I/O* system to other program that may reside either on the local machine or on any other machine on the Internet.
- To receive response from the server: Some useful classes to create inputstream
- InputStream- supports basic **read()** methods
- DataInputStream-supports **read(), readChar(), readInt(), readDouble(), etc..** methods
- InputStreamReader-supports **readLine()** method
- To send information to the server:
  - Some useful classes to create outputstream
    - OutputStream- supports basic **write()** methods
    - DataOutputStream- supports **print()** methods to write Java primitive data types
    - PrintStream- supports **print()** methods for displaying Java primitive data types
    - PrintWriter- supports **print()** and **println()** methods

### Matched Input/Output Objects

- InputStream and OutputStream
- DataInputStream and DataOutputStream
- PrintWriter and BufferedReader

### Example 2: Calculator Program

#### Server

- Acts as a calculator
- Accepts command and inputs, and calculates them
- Returns result

#### Client

- Interacts with user to get inputs
- Request the server giving command with data
- Receive the result

```

import java.io.*;
import java.net.*;
public class CalculatorServer{
    ServerSocket server;
    Socket socket;
    DataInputStream in;
    DataOutputStream out;
    public CalculatorServer()
    {
        String command="";
        int num1=0;
        int num2=0;
        int result=0;
        boolean flag = true;
        try{
            server = new ServerSocket(7000);
            System.out.println("Server is ready!");
            socket = server.accept();
            System.out.println("Connection is established.");
            while(flag)
            {
                System.out.println("\nWaiting request.....");
                in = new DataInputStream(socket.getInputStream());
                command = in.readUTF();
                num1 = in.readInt();
                num2 = in.readInt();
                result=0;
                System.out.println("command = " + command);
                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);
                if (command.toUpperCase().equals("ADD"))
                    result = calculateAdd(num1, num2);
                else if (command.toUpperCase().equals("SUB"))
                    result = calculateSub(num1, num2);
                else if (command.toUpperCase().equals("MULT"))
                    result = calculateMult(num1, num2);
                else
                    result = calculateDiv(num1, num2);
                out = new DataOutputStream(socket.getOutputStream());
                out.writeInt(result);
            }
            catch (Exception e){
                try { socket.close();
                    System.out.println("The connection is closed.");
                } catch (IOException e1)
                {
                    e1.printStackTrace();
                }
            }
        }
    }
    private int calculateAdd(int n1, int n2) {
        return (n1 + n2);
    }
    private int calculateSub(int n1, int n2) {
        return (n1 -n2);
    }
    private int calculateMult(int n1, int n2) {
        return (n1 * n2);
    }
    private int calculateDiv(int n1, int n2) {
        return (n1 / n2);
    }
}

```

```

    public static void main(String[] args) throws Exception {
        new CalculatorServer();
    }
}
import java.io.*;
import java.net.*;
public class CalculatorClient{
    Socket socket;
    DataInputStream in;
    DataOutputStream out;
    public CalculatorClient() throws Exception
    {
        String command="";
        int num1=0;
        int num2=0;
        int result=0;
        char ch=0;
        socket = new Socket("localhost", 7000);
        System.out.println("The client gets connection with server.");

        do{// interacts with user to get inputs
            BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
            System.out.print("Type a command (ADD, SUB, MULT, DIV) : ");
            command = br.readLine();
            System.out.print("Enter number 1 : ");
            num1 = Integer.parseInt(br.readLine());
            System.out.print("Enter number 2 : ");
            num2 = Integer.parseInt(br.readLine());
            out = new DataOutputStream(socket.getOutputStream());
            out.writeUTF(command);
            out.writeInt(num1);
            out.writeInt(num2);

            in = new DataInputStream(socket.getInputStream());
            result=in.readInt();
            System.out.println("result = " + result);
            System.out.print("\nTryit more (y/n)");
            ch=br.readLine().toLowerCase().charAt(0);
        } while (ch=='y');
        socket.close();
        System.out.println("disconnected with server!");}

    public static void main(String[] args) throws Exception
    {
        new CalculatorClient();
    }
}

```

Server is ready!  
Connection is established.

Waitingrequest.....  
command = ADD  
num1 = 20  
num2 = 30

Waitingrequest.....  
command = SUB  
num1 = 50  
num2 = 30

Waitingrequest.....  
The connection is closed.

The client gets connection with server.  
Type a command (ADD, SUB, MULT, DIV) : ADD  
Enter number 1 : 20  
Enter number 2 : 30  
result = 50

Tryit more (y/n)y  
Type a command (ADD, SUB, MULT, DIV) : SUB  
Enter number 1 : 50  
Enter number 2 : 30  
result = 20

Tryit more (y/n)n  
disconnected with server!



## Prime

```
package mysocket;
import java.io.*;
import java.net.*;
public class PrimeServer{
    ServerSocket server;
    Socket socket;
    DataInputStream in;
    DataOutputStream out;
    public PrimeServer()
    {
        int num=0;
        boolean result;

        boolean flag = true;
        try{
            server = new ServerSocket(7000);
            System.out.println("Server is ready!");
            socket = server.accept();
            System.out.println("Connection is established.");
            while(flag)
            {
                System.out.println("\nWaiting request....");
                in = new DataInputStream(socket.getInputStream());

                num = in.readInt();
                result = isPrime(num);
                out = new DataOutputStream(socket.getOutputStream());
                out.writeBoolean(result);
            }
            catch(Exception e){
                try { socket.close();
                    System.out.println("The connection is closed.");
                } catch (IOException e1)
                {
                    e1.printStackTrace();
                }
            }
        }
    }
    private boolean isPrime(int n1) {
        int div=2;
        while(n1%div!=0)
        {
            div++;
        }
        if(div==n1)
            return true;
        else
            return false;
    }
    public static void main(String[] args) throws Exception {
        new PrimeServer();
    }
}

import java.io.*;
import java.net.*;
public class PrimeClient{
    Socket socket;
    DataInputStream in;
    DataOutputStream out;
    public PrimeClient() throws Exception
    {
```

```

int num=0;
boolean result;
char ch=0;
socket = new Socket("localhost", 7000);
System.out.println("The client gets connection with server.");

do{// interacts with user to get inputs
    BufferedReader br= new BufferedReader(new InputStreamReader(System.in));

    System.out.print("Enter number: ");
    num= Integer.parseInt(br.readLine());

    out = new DataOutputStream(socket.getOutputStream());
    out.writeInt(num);

    in = new DataInputStream(socket.getInputStream());
    result=in.readBoolean();
    if(result)
    {
        System.out.println("It is prime.");
    }
    else
    {
        System.out.println("It is not prime.");
    }

    System.out.print("\nTryit more (y/n)");
    ch=br.readLine().toLowerCase().charAt(0);
} while (ch=='y');
socket.close();
System.out.println("disconnected with server!");}
public static void main(String[] args) throws Exception
{
    new PrimeClient();
}
}

```

Let's do exercises!!!!



# SOCKET

## ASSIGNMENT

1. Write a java network program that includes two java files, one for server and one is client. In client site, ask item name and quantity from user and submit to server.

Book=250  
Pencil=150  
Eraser=100  
Pen=150  
Ruler=200

The interaction should be as follow:

<p>Server is ready! Connection is established.</p> <p>Waitingrequest..... Item Name = book Quantity = 10</p> <p>Waitingrequest..... Item Name = pencil Quantity = 5</p> <p>Waitingrequest..... Item Name = noitem Quantity = 0</p> <p>Waitingrequest..... The connection is closed.</p>	<p>The client gets connection with server. Enter item you want to buy[book, pencil, eraser, pen, or ruler]book Enter quantity : 10 book 10 2500</p> <p>Any more (y/n)y Enter item you want to buy[book, pencil, eraser, pen, or ruler]pencil Enter quantity : 5 pencil 5 750</p> <p>Any more (y/n)n Total=3250 Thank you so much. Please come again</p>
---	---

2. Write a java network program that includes two java files, one for server and one is client. At client site, ask rank and working hour from user and submit to server.

The salary is basically defined as follows:

Rank	Salary
Engineer	180000(200 hours)
Technician	120000(200 hours)
Staff	80000(200 hours)

Salary is calculated as follows:

salary = basic Salary + OTCharge(if OT)

OTRate=2

EngineerHourlyPay=900

TechnicianHourlyPay=600

StaffHourlyPay=400

OTCharge=(HoursWorked-200)\*HourlyPay\*OTRate

<p>Server is ready! Connection is established.</p> <p>Waitingrequest..... Rank = Engineer Working Hour = 300</p> <p>Waitingrequest..... Rank = staff Working Hour = 180</p> <p>Waitingrequest..... The connection is closed.</p>	<p>The client gets connection with server. Enter rankEngineer Enter working hour : 300 Salary= 360000</p> <p>Any more (y/n)y Enter rankstaff Enter working hour : 180 Salary= 80000</p> <p>Any more (y/n)n</p>
--	--