

Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides a facility to share data between different computing devices.

Advantage of Java Networking

1. Sharing resources
2. Centralize software management

Do You Know ?

- How to perform connection-oriented Socket Programming in networking ?
- How to display the data of any online web page ?
- How to get the IP address of any host name e.g. www.google.com ?
- How to perform connection-less socket programming in networking ?

The `java.net` package supports two protocols,

1. **TCP:** Transmission Control Protocol provides reliable communication between the sender and receiver. TCP is used along with the Internet Protocol referred as TCP/IP.
2. **UDP:** User Datagram Protocol provides a connection-less protocol service by allowing packet of data to be transferred along two or more nodes

Java Networking Terminology

The widely used Java networking terminologies are given below:

1. IP Address
2. Protocol

3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet
- SMTP
- POP etc.

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC address.

For example, an ethernet card may have a **MAC** address of 00:0d:83::b1:c0:8e.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

A socket is an endpoint between two way communications.

Visit next page for Java socket programming.

java.net package

The java.net package can be divided into two sections:

1. **A Low-Level API:** It deals with the abstractions of addresses i.e. networking identifiers, Sockets i.e. bidirectional data communication mechanism and Interfaces i.e. network interfaces.
2. **A High Level API:** It deals with the abstraction of URIs i.e. Universal Resource Identifier, URLs i.e. Universal Resource Locator, and Connections i.e. connections to the resource pointed by URLs.

The java.net package provides many classes to deal with networking applications in Java. A list of these classes is given below:

- Authenticator
- CacheRequest
- CacheResponse
- ContentHandler
- CookieHandler
- CookieManager

- DatagramPacket
- DatagramSocket
- DatagramSocketImpl
- InterfaceAddress
- JarURLConnection
- MulticastSocket
- InetSocketAddress
- InetAddress
- Inet4Address
- Inet6Address
- IDN
- HttpURLConnection
- HttpCookie
- NetPermission
- NetworkInterface
- PasswordAuthentication
- Proxy
- ProxySelector
- ResponseCache
- SecureCacheResponse
- ServerSocket
- Socket
- SocketAddress
- SocketImpl
- SocketPermission
- StandardSocketOptions

- URI
- URL
- URLClassLoader
- URLConnection
- URLDecoder
- URLEncoder
- URLStreamHandler

List of interfaces available in java.net package:

- ContentHandlerFactory
- CookiePolicy
- CookieStore
- DatagramSocketImplFactory
- FileNameMap
- SocketOption<T>
- SocketOptions
- SocketImplFactory
- URLStreamHandlerFactory
- ProtocolFamily

What we will learn in Networking Tutorial

- Networking and Networking Terminology
- Socket Programming (Connection-oriented)
- URL class
- Displaying data of a webpage by URLConnection class
- InetAddress class

- DatagramSocket and DatagramPacket (Connection-less)

Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE.

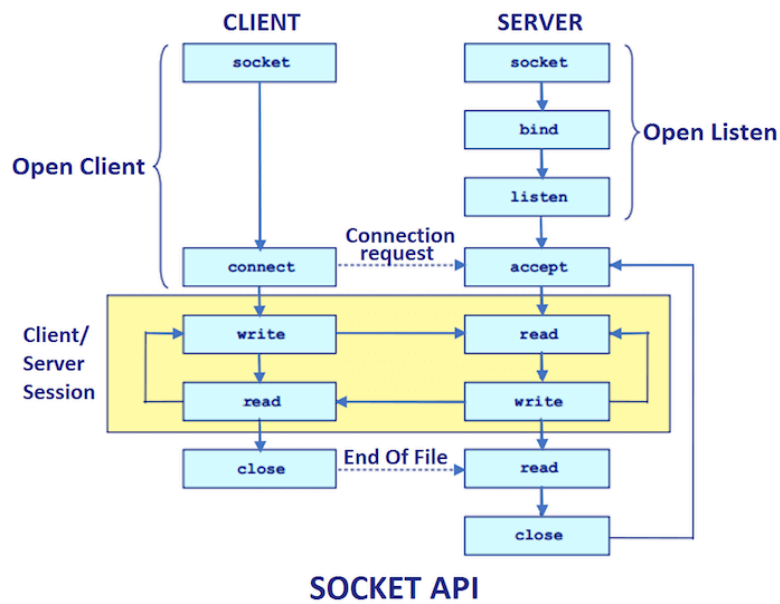
Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.



Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Important methods

| Method | Description |
|---|---|
| 1) public InputStream getInputStream() | returns the InputStream attached with this socket. |
| 2) public OutputStream getOutputStream() | returns the OutputStream attached with this socket. |
| 3) public synchronized void close() | closes this socket |

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

| Method | Description |
|-------------------------------------|--|
| 1) public Socket accept() | returns the socket and establish a connection between server and client. |
| 2) public synchronized void close() | closes the server socket. |

Example of Java Socket Programming

Creating Server:

To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

1. ServerSocket ss=**new** ServerSocket(**6666**);
2. Socket s=ss.accept();//**establishes connection and waits for the client**

Creating Client:

To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.

1. Socket s=**new** Socket(**"localhost"**,**6666**);

Let's see a simple of Java socket programming where client sends a text and server receives and prints it.

File: MyServer.java

```
1. import java.io.*;
2. import java.net.*;
3. public class MyServer {
4. public static void main(String[] args){
5. try{
6. ServerSocket ss=new ServerSocket(6666);
7. Socket s=ss.accept();//establishes connection
8. DataInputStream dis=new DataInputStream(s.getInputStream());
9. String str=(String)dis.readUTF();
10. System.out.println("message= "+str);
11. ss.close();
12. }catch(Exception e){System.out.println(e);}
13. }
14. }
```

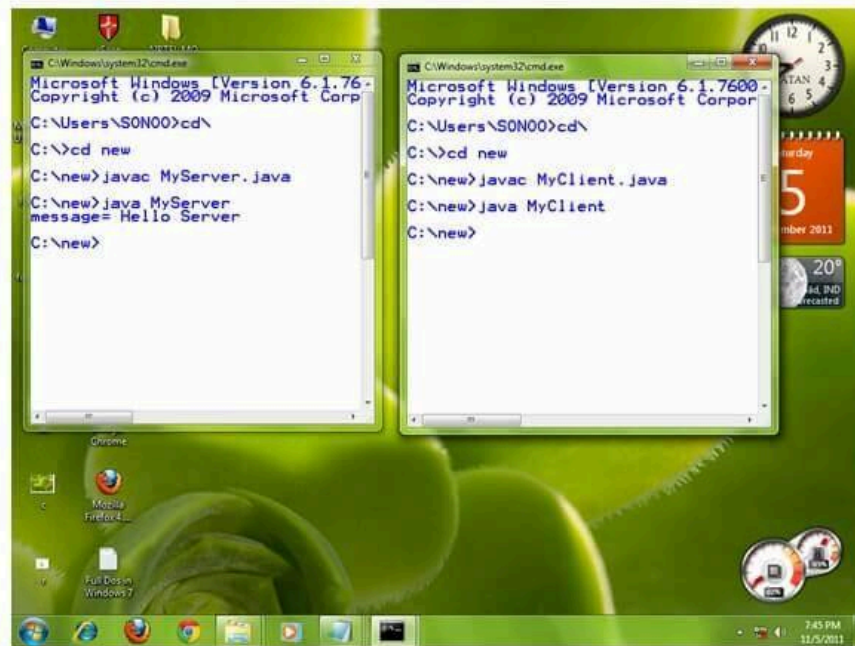
File: MyClient.java

```
1. import java.io.*;
2. import java.net.*;
3. public class MyClient {
4. public static void main(String[] args) {
5. try{
6. Socket s=new Socket("localhost",6666);
7. DataOutputStream dout=new DataOutputStream(s.getOutputStream());
8. dout.writeUTF("Hello Server");
9. dout.flush();
10. dout.close();
11. s.close();
12. }catch(Exception e){System.out.println(e);}
13. }
14. }
```

[download this example](#)

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure.

After running the client application, a message will be displayed on the server console.



Example of Java Socket Programming (Read-Write both side)

In this example, client will write first to the server then server will receive and print the text. Then server will write to the client and client will receive and print the text. The step goes on.

File: *MyServer.java*

1. **import** java.net.*;
2. **import** java.io.*;
3. **class** MyServer{
4. **public static void** main(String args[])**throws** Exception{

```

5. ServerSocket ss=new ServerSocket(3333);
6. Socket s=ss.accept();
7. DataInputStream din=new DataInputStream(s.getInputStream());
8. DataOutputStream dout=new DataOutputStream(s.getOutputStream());
9. BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
10.
11. String str="",str2="";
12. while(!str.equals("stop")){
13. str=din.readUTF();
14. System.out.println("client says: "+str);
15. str2=br.readLine();
16. dout.writeUTF(str2);
17. dout.flush();
18. }
19. din.close();
20. s.close();
21. ss.close();
22. }}

```

File: MyClient.java

```

1. import java.net.*;
2. import java.io.*;
3. class MyClient{
4. public static void main(String args[])throws Exception{
5. Socket s=new Socket("localhost",3333);
6. DataInputStream din=new DataInputStream(s.getInputStream());
7. DataOutputStream dout=new DataOutputStream(s.getOutputStream());
8. BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
9.
10. String str="",str2="";
11. while(!str.equals("stop")){

```

```
12. str=br.readLine();
13. dout.writeUTF(str);
14. dout.flush();
15. str2=din.readUTF();
16. System.out.println("Server says: "+str2);
17.}
18.
19. dout.close();
20. s.close();
21.}}
```

Java URL

The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:

1. <https://www.javatpoint.com/java-tutorial>



A URL contains many information:

1. **Protocol:** In this case, http is the protocol.
 2. **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
 3. **Port Number:** It is an optional attribute. If we write `http://www.javatpoint.com:80/sonoojaiswal/` , 80 is the port number. If port number is not mentioned in the URL, it returns -1.
 4. **File Name or directory name:** In this case, index.jsp is the file name.
-

Constructors of Java URL class

URL(String spec)

Creates an instance of a URL from the String representation.

URL(String protocol, String host, int port, String file)

Creates an instance of a URL from the given protocol, host, port number, and file.

URL(String protocol, String host, int port, String file, URLStreamHandler handler)

Creates an instance of a URL from the given protocol, host, port number, file, and handler.

URL(String protocol, String host, String file)

Creates an instance of a URL from the given protocol name, host name, and file name.

URL(URL context, String spec)

Creates an instance of a URL by parsing the given spec within a specified context.

URL(URL context, String spec, URLStreamHandler handler)

Creates an instance of a URL by parsing the given spec with the specified handler within a given context.

Commonly used methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class are given below.

| Method | Description |
|-----------------------------|--|
| public String getProtocol() | it returns the protocol of the URL. |
| public String getHost() | it returns the host name of the URL. |
| public String getPort() | it returns the Port Number of the URL. |

| | |
|--|---|
| public String getFile() | it returns the file name of the URL. |
| public String getAuthority() | it returns the authority of the URL. |
| public String toString() | it returns the string representation of the URL. |
| public String getQuery() | it returns the query string of the URL. |
| public String getDefaultPort() | it returns the default port of the URL. |
| public URLConnection openConnection() | it returns the instance of URLConnection i.e. associated with this URL. |
| public boolean equals(Object obj) | it compares the URL with the given object. |
| public Object getContent() | it returns the content of the URL. |
| public String getRef() | it returns the anchor or reference of the URL. |
| public URI toURI() | it returns a URI of the URL. |

Example of Java URL class

1. `//URLDemo.java`
2. `import java.net.*;`
3. `public class URLDemo{`
4. `public static void main(String[] args){`
5. `try{`
6. `URL url=new URL("http://www.javatpoint.com/java-tutorial");`
- 7.
8. `System.out.println("Protocol: "+url.getProtocol());`
9. `System.out.println("Host Name: "+url.getHost());`
10. `System.out.println("Port Number: "+url.getPort());`

```
11. System.out.println("File Name: "+url.getFile());
12.
13. }catch(Exception e){System.out.println(e);}
14. }
15. }
```

Test it Now

Output:

```
Protocol: http
Host Name: www.javatpoint.com
Port Number: -1
File Name: /java-tutorial
```

Let us see another example URL class in Java.

```
1. //URLDemo.java
2. import java.net.*;
3. public class URLDemo{
4.     public static void main(String[] args){
5.     try{
6.         URL url=new
           URL("https://www.google.com/search?q=javatpoint&oq=javatpoint&sourceid=
           chrome&ie=UTF-8");
7.
8.         System.out.println("Protocol: "+url.getProtocol());
9.         System.out.println("Host Name: "+url.getHost());
10.        System.out.println("Port Number: "+url.getPort());
11.        System.out.println("Default Port Number: "+url.getDefaultPort());
12.        System.out.println("Query String: "+url.getQuery());
13.        System.out.println("Path: "+url.getPath());
14.        System.out.println("File: "+url.getFile());
15.    }
```

```
16. }catch(Exception e){System.out.println(e);}
17. }
18. }
```

Output:

Protocol: https

Host Name: www.google.com

Port Number: -1

Default Port Number: 443

Query String: q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8

Path: /search

File: /search?q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8

Java URLConnection Class

The **Java URLConnection** class represents a communication link between the URL and the application. It can be used to read and write data to the specified resource referred by the URL.

What is the URL?

- URL is an abbreviation for Uniform Resource Locator. An URL is a form of string that helps to find a resource on the World Wide Web (WWW).
- URL has two components:
 1. The protocol required to access the resource.
 2. The location of the resource.

Features of URLConnection class

1. URLConnection is an abstract class. The two subclasses HttpURLConnection and JarURLConnection makes the connection between the client Java program and URL resource on the internet.

2. With the help of `URLConnection` class, a user can read and write to and from any resource referenced by an `URL` object.
3. Once a connection is established and the Java program has an `URLConnection` object, we can use it to read or write or get further information like content length, etc.

Constructors

| Constructor | Description |
|--|---|
| 1) protected <code>URLConnection(URL url)</code> | It constructs a <code>URL</code> connection to the specified <code>URL</code> . |

URLConnection Class Methods

| Method | Description |
|--|---|
| <code>void addRequestProperty(String key, String value)</code> | It adds a general request property specified by a key-value pair |
| <code>void connect()</code> | It opens a communications link to the resource referenced by this <code>URL</code> , if such a connection has not already been established. |
| <code>boolean getAllowUserInteraction()</code> | It returns the value of the <code>allowUserInteraction</code> field for the object. |
| <code>int getConnectionTimeout()</code> | It returns setting for connect timeout. |
| <code>Object getContent()</code> | It retrieves the contents of the <code>URL</code> connection. |

| | |
|--|---|
| Object getContent(Class[] classes) | It retrieves the contents of the URL connection. |
| String getContentEncoding() | It returns the value of the content-encoding header field. |
| int getContentLength() | It returns the value of the content-length header field. |
| long getContentLengthLong() | It returns the value of the content-length header field as long. |
| String getContentType() | It returns the value of the date header field. |
| long getDate() | It returns the value of the date header field. |
| static boolean getDefaultAllowUserInteraction() | It returns the default value of the allowUserInteraction field. |
| boolean getDefaultUseCaches() | It returns the default value of an URLConnetion's useCaches flag. |
| boolean getDoInput() | It returns the value of the URLConnection's doInput flag. |
| boolean getDoOutput() | It returns the value of the URLConnection's doOutput flag. |
| long getExpiration() | It returns the value of the expires header files. |

| | |
|--|--|
| <code>static FileNameMap getFilenameMap()</code> | It loads the filename map from a data file. |
| <code>String getHeaderField(int n)</code> | It returns the value of n th header field |
| <code>String getHeaderField(String name)</code> | It returns the value of the named header field. |
| <code>long getHeaderFieldDate(String name, long Default)</code> | It returns the value of the named field parsed as a number. |
| <code>int getHeaderFieldInt(String name, int Default)</code> | It returns the value of the named field parsed as a number. |
| <code>String getHeaderFieldKey(int n)</code> | It returns the key for the n th header field. |
| <code>long getHeaderFieldLong(String name, long Default)</code> | It returns the value of the named field parsed as a number. |
| <code>Map<String, List<String>> getHeaderFields()</code> | It returns the unmodifiable Map of the header field. |
| <code>long getIfModifiedSince()</code> | It returns the value of the object's ifModifiedSince field. |
| <code>InputStream getInputStream()</code> | It returns an input stream that reads from the open condition. |
| <code>long getLastModified()</code> | It returns the value of the last-modified header field. |

| | |
|---|--|
| OutputStream getOutputStream() | It returns an output stream that writes to the connection. |
| Permission getPermission() | It returns a permission object representing the permission necessary to make the connection represented by the object. |
| int getReadTimeout() | It returns setting for read timeout. |
| Map<String, List<String>> getRequestProperties() | It returns the value of the named general request property for the connection. |
| URL getURL() | It returns the value of the URLConnection's URL field. |
| boolean getUseCaches() | It returns the value of the URLConnection's useCaches field. |
| Static String guessContentTypeFromName(String fname) | It tries to determine the content type of an object, based on the specified file component of a URL. |
| static String guessContentTypeFromStream(InputStream is) | It tries to determine the type of an input stream based on the characters at the beginning of the input stream. |
| void setAllowUserInteraction(boolean allowuserinteraction) | It sets the value of the allowUserInteraction field of this URLConnection. |

| | |
|--|--|
| static void setContentHandlerFactory(ContentHandlerFactory fac) | It sets the ContentHandlerFactory of an application. |
| static void setDefaultAllowUserInteraction(boolean defaultallowuserinteraction) | It sets the default value of the allowUserInteraction field for all future URLConnection objects to the specified value. |
| void setDefaultUseCaches(boolean defaultusecaches) | It sets the default value of the useCaches field to the specified value. |
| void setDoInput(boolean doinput) | It sets the value of the doInput field for this URLConnection to the specified value. |
| void setDoOutput(boolean dooutput) | It sets the value of the doOutput field for the URLConnection to the specified value. |

How to get the object of URLConnection Class

The openConnection() method of the URL class returns the object of URLConnection class.

Syntax:

1. **public** URLConnection openConnection()**throws** IOException{

Displaying Source Code of a Webpage by URLConnecton Class

The `URLConnection` class provides many methods. We can display all the data of a webpage by using the `getInputStream()` method. It returns all the data of the specified URL in the stream that can be read and displayed.

Example of Java `URLConnection` Class

```
1. import java.io.*;
2. import java.net.*;
3. public class URLConnectionExample {
4. public static void main(String[] args){
5. try{
6. URL url=new URL("http://www.javatpoint.com/java-tutorial");
7. URLConnection urlcon=url.openConnection();
8. InputStream stream=urlcon.getInputStream();
9. int i;
10. while((i=stream.read())!=-1){
11. System.out.print((char)i);
12.}
13.}catch(Exception e){System.out.println(e);}
14.}
15.}
```

Java `HttpURLConnection` class

The **Java `HttpURLConnection`** class is http specific `URLConnection`. It works for HTTP protocol only.

By the help of `HttpURLConnection` class, you can retrieve information of any HTTP URL such as header information, status code, response code etc.

The `java.net.HttpURLConnection` is subclass of `URLConnection` class.

`HttpURLConnection` Class Constructor

| Constructor | Description |
|-----------------------------------|---|
| protected URLConnection(URL u) | It constructs the instance of URLConnection class. |

Java HttpURLConnection Methods

| Method | Description |
|---|---|
| void disconnect() | It shows that other requests from the server are unlikely in the near future. |
| InputStream getErrorStream() | It returns the error stream if the connection failed but the server sent useful data. |
| Static boolean getFollowRedirects() | It returns a boolean value to check whether or not HTTP redirects should be automatically followed. |
| String getHeaderField(int n) | It returns the value of nth header file. |
| long getHeaderFieldDate(String name, long Default) | It returns the value of the named field parsed as a date. |

| | |
|--|--|
| String getHeaderFieldKey(int n) | It returns the key for the nth header file. |
| boolean getInstanceFollowRedirects() | It returns the value of HttpURLConnection's instance FollowRedirects field. |
| Permission getPermission() | It returns the SocketPermission object representing the permission to connect to the destination host and port. |
| String getRequestMethod() | It gets the request method. |
| int getResponseCode() | It gets the response code from an HTTP response message. |
| String getResponseMessage() | It gets the response message sent along with the response code from a server. |
| void setChunkedStreamingMode(int chunklen) | The method is used to enable streaming of a HTTP request body without internal buffering, when the content length is not known in advance. |

| | |
|---|--|
| void setFixedLengthStreamingMode(int contentlength) | The method is used to enable streaming of a HTTP request body without internal buffering, when the content length is known in advance. |
| void setFixedLengthStreamingMode(long contentlength) | The method is used to enable streaming of a HTTP request body without internal buffering, when the content length is not known in advance. |
| static void setFollowRedirects(boolean set) | It sets whether HTTP redirects (requests with response code) should be automatically followed by HttpURLConnection class. |
| void setInstanceFollowRedirects(boolean followRedirects) | It sets whether HTTP redirects (requests with response code) should be automatically followed by instance of HttpURLConnection class. |
| void setRequestMethod(String method) | Sets the method for the URL request, one of: GET POST HEAD OPTIONS PUT DELETE TRACE are legal, subject to protocol restrictions. |
| abstract boolean usingProxy() | It shows if the connection is going through a proxy. |

How to get the object of HttpURLConnection class

The `openConnection()` method of `URL` class returns the object of `URLConnection` class.

Syntax:

1. **public** `URLConnection` `openConnection()`**throws** `IOException`{

You can typecast it to `HttpURLConnection` type as given below.

1. `URL url=new URL("http://www.javatpoint.com/java-tutorial");`
2. `HttpURLConnection huc=(HttpURLConnection)url.openConnection();`

Java HttpURLConnection Example

1. **import** `java.io.*`;
2. **import** `java.net.*`;
3. **public class** `HttpURLConnectionDemo`{
4. **public static void** `main(String[] args){`
5. **try**{
6. `URL url=new URL("http://www.javatpoint.com/java-tutorial");`
7. `HttpURLConnection huc=(HttpURLConnection)url.openConnection();`
8. **for**(**int** `i=1`; `i<=8`; `i++`){
9. `System.out.println(huc.getHeaderFieldKey(i)+ " = "+huc.getHeaderField(i));`
- 10.}
11. `huc.disconnect();`
12. **catch**(`Exception e`){`System.out.println(e);`}
- 13.}
- 14.}

Test it Now

Output:

Date = Thu, 22 Jul 2021 18:08:17 GMT

Server = Apache

Location = <https://www.javatpoint.com/java-tutorial>

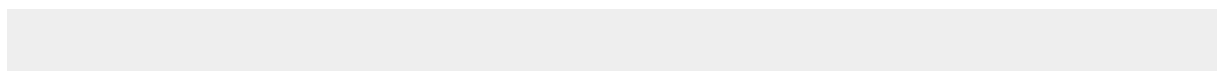
Cache-Control = max-age=2592000

Expires = Sat, 21 Aug 2021 18:08:17 GMT

Content-Length = 248

Keep-Alive = timeout=5, max=1500

Connection = Keep-Alive



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\WIN 8.1>cd Desktop
C:\Users\WIN 8.1\Desktop>javac HttpURLConnectionDemo.java
C:\Users\WIN 8.1\Desktop>java HttpURLConnectionDemo
Date = Thu, 22 Jul 2021 18:08:17 GMT
Server = Apache
Location = https://www.javatpoint.com/java-tutorial
Cache-Control = max-age=2592000
Expires = Sat, 21 Aug 2021 18:08:17 GMT
Content-Length = 248
Keep-Alive = timeout=5, max=1500
Connection = Keep-Alive

C:\Users\WIN 8.1\Desktop>
```

Java 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.

An IP address is represented by 32-bit or 128-bit unsigned number. An instance of InetAddress represents the IP address with its corresponding host name. There are two types of addresses: Unicast and Multicast. The Unicast is an identifier for a single interface whereas Multicast is an identifier for a set of interfaces.

Moreover, InetAddress has a cache mechanism to store successful and unsuccessful host name resolutions.

IP Address

- An IP address helps to identify a specific resource on the network using a numerical representation.
- Most networks combine IP with TCP (Transmission Control Protocol). It builds a virtual bridge among the destination and the source.

There are two versions of IP address:

1. IPv4

IPv4 is the primary Internet protocol. It is the first version of IP deployed for production in the ARPANET in 1983. It is a widely used IP version to differentiate devices on network using an addressing scheme. A 32-bit addressing scheme is used to store 2^{32} addresses that is more than 4 million addresses.

Features of IPv4:

- It is a connectionless protocol.
- It utilizes less memory and the addresses can be remembered easily with the class based addressing scheme.
- It also offers video conferencing and libraries.

2. IPv6

IPv6 is the latest version of Internet protocol. It aims at fulfilling the need of more internet addresses. It provides solutions for the problems present in IPv4. It provides 128-bit address space that can be used to form a network of 340 undecillion unique IP addresses. IPv6 is also identified with a name IPng (Internet Protocol next generation).

Features of IPv6:

- It has a stateful and stateless both configurations.
- It provides support for quality of service (QoS).
- It has a hierarchical addressing and routing infrastructure.

TCP/IP Protocol

- TCP/IP is a communication protocol model used connect devices over a network via internet.
- TCP/IP helps in the process of addressing, transmitting, routing and receiving the data packets over the internet.
- The two main protocols used in this communication model are:
 1. TCP i.e. Transmission Control Protocol. TCP provides the way to create a communication channel across the network. It also helps in transmission of packets at sender end as well as receiver end.
 2. IP i.e. Internet Protocol. IP provides the address to the nodes connected on the internet. It uses a gateway computer to check whether the IP address is correct and the message is forwarded correctly or not.

Java InetAddress Class Methods

| Method | Description |
|---|--|
| <pre> public static InetAddress getByName(String host) throws UnknownHostException </pre> | It returns the instance of InetAddress containing LocalHost IP and name. |
| <pre> public static InetAddress getLocalHost() throws UnknownHostException </pre> | It returns the instance of InetAddress containing local host name and address. |
| <pre> public String getHostName() </pre> | It returns the host name of the IP address. |
| <pre> public String getHostAddress() </pre> | It returns the IP address in string format. |

Example of Java InetAddress Class

Let's see a simple example of InetAddress class to get ip address of www.javatpoint.com website.

InetDemo.java

1. **import** java.io.*;
2. **import** java.net.*;
3. **public class** InetDemo{
4. **public static void** main(String[] args){
5. **try**{

```
6. InetAddress ip=InetAddress.getByName("www.javatpoint.com");
7.
8. System.out.println("Host Name: "+ip.getHostName());
9. System.out.println("IP Address: "+ip.getHostAddress());
10.}catch(Exception e){System.out.println(e);}
11.}
12.}
```

Test it Now

Output:

Host Name: www.javatpoint.com

IP Address: 172.67.196.82

Program to demonstrate methods of InetAddress class

InetDemo2.java

```
1. import java.net.Inet4Address;
2. import java.util.Arrays;
3. import java.net.InetAddress;
4. public class InetDemo2
5. {
6.     public static void main(String[] arg) throws Exception
7.     {
8.         InetAddress ip = Inet4Address.getByName("www.javatpoint.com");
9.         InetAddress ip1[] = InetAddress.getAllByName("www.javatpoint.com");
10.        byte addr[]={72, 3, 2, 12};
11.        System.out.println("ip : "+ip);
12.        System.out.print("\nip1 : "+ip1);
13.        InetAddress ip2 = InetAddress.getByAddress(addr);
```

```

14. System.out.print("\nip2 : "+ip2);
15. System.out.print("\nAddress : " +Arrays.toString(ip.getAddress()));
16. System.out.print("\nHost Address : " +ip.getHostAddress());
17. System.out.print("\nisAnyLocalAddress : " +ip.isAnyLocalAddress());
18. System.out.print("\nisLinkLocalAddress : " +ip.isLinkLocalAddress());
19. System.out.print("\nisLoopbackAddress : " +ip.isLoopbackAddress());
20. System.out.print("\nisMCGlobal : " +ip.isMCGlobal());
21. System.out.print("\nisMCLinkLocal : " +ip.isMCLinkLocal());
22. System.out.print("\nisMCNodeLocal : " +ip.isMCNodeLocal());
23. System.out.print("\nisMCOrgLocal : " +ip.isMCOrgLocal());
24. System.out.print("\nisMCSiteLocal : " +ip.isMCSiteLocal());
25. System.out.print("\nisMulticastAddress : " +ip.isMulticastAddress());
26. System.out.print("\nisSiteLocalAddress : " +ip.isSiteLocalAddress());
27. System.out.print("\nhashCode : " +ip.hashCode());
28. System.out.print("\n Is ip1 == ip2 : " +ip.equals(ip2));
29.}
30.}

```

Output:

```

C:\Users\WIN 8.1>cd Desktop
C:\Users\WIN 8.1\Desktop>javac InetDemo2.java
C:\Users\WIN 8.1\Desktop>java InetDemo2
ip : www.javatpoint.com/172.67.196.82
ip1 : [Ljava.net.InetAddress;@7852e922
ip2 : /72.3.2.12
Address : [-84, 67, -60, 82]
Host Address : 172.67.196.82
isAnyLocalAddress : false
isLinkLocalAddress : false
isLoopbackAddress : false
isMCGlobal : false
isMCLinkLocal : false
isMCNodeLocal : false
isMCOrgLocal : false
isMCSiteLocal : false
isMulticastAddress : false
isSiteLocalAddress : false
hashCode : -1404844974
Is ip1 == ip2 : false
C:\Users\WIN 8.1\Desktop>

```


In the above Java code, the various boolean methods of InetAddress class are demonstrated.

Java DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming using the UDP instead of TCP.

Datagram

Datagrams are collection of information sent from one device to another device via the established network. When the datagram is sent to the targeted device, there is no assurance that it will reach to the target device safely and completely. It may get damaged or lost in between. Likewise, the receiving device also never know if the datagram received is damaged or not. The UDP protocol is used to implement the datagrams in Java.

Java DatagramSocket class

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets. It is a mechanism used for transmitting datagram packets over network.

A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class

- **DatagramSocket() throws SocketEeption:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port) throws SocketEeption:** it creates a datagram socket and binds it with the given Port Number.

- **DatagramSocket(int port, InetAddress address) throws SocketException:** it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramSocket Class

| Method | Description |
|---|---|
| void bind(SocketAddress addr) | It binds the DatagramSocket to a specific address and port. |
| void close() | It closes the datagram socket. |
| void connect(InetAddress address, int port) | It connects the socket to a remote address for the socket. |
| void disconnect() | It disconnects the socket. |
| boolean getBroadcast() | It tests if SO_BROADCAST is enabled. |

| | |
|--|---|
| DatagramChannel getChannel() | It returns the unique DatagramChannel object associated with the datagram socket. |
| InetAddress getInetAddress() | It returns the address to where the socket is connected. |
| InetAddress getLocalAddress() | It gets the local address to which the socket is connected. |
| int getLocalPort() | It returns the port number on the local host to which the socket is bound. |
| SocketAddress getLocalSocketAddresses() | It returns the address of the endpoint the socket is bound to. |
| int getPort() | It returns the port number to which the socket is connected. |
| int getReceiverBufferSize()) | It gets the value of the SO_RCVBUF option for this DatagramSocket that is the buffer size used by the platform for input on the DatagramSocket. |

| | |
|--|---|
| <code>boolean isClosed()</code> | It returns the status of socket i.e. closed or not. |
| <code>boolean isConnected()</code> | It returns the connection state of the socket. |
| <code>void send(DatagramPacket p)</code> | It sends the datagram packet from the socket. |
| <code>void receive(DatagramPack et p)</code> | It receives the datagram packet from the socket. |

Java DatagramPacket Class

Java DatagramPacket is a message that can be sent or received. It is a data container. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

- **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] barr, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

Java DatagramPacket Class Methods

| Method | Description |
|--|---|
| 1) InetAddress getAddress() | It returns the IP address of the machine to which the datagram is being sent or from which the datagram was received. |
| 2) byte[] getData() | It returns the data buffer. |
| 3) int getLength() | It returns the length of the data to be sent or the length of the data received. |
| 4) int getOffset() | It returns the offset of the data to be sent or the offset of the data received. |
| 5) int getPort() | It returns the port number on the remote host to which the datagram is being sent or from which the datagram was received. |
| 6) SocketAddress getSocketAddress() | It gets the SocketAddress (IP address + port number) of the remote host that the packet is being sent to or is coming from. |

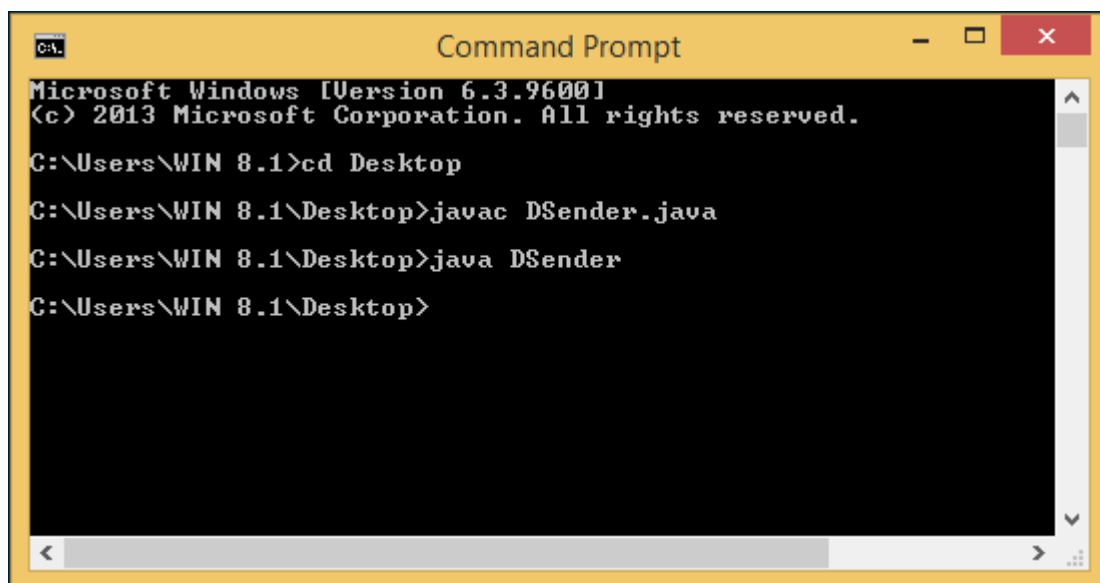
| | |
|---|--|
| 7) void setAddress(InetAddress iaddr) | It sets the IP address of the machine to which the datagram is being sent. |
| 8) void setData(byte[] buff) | It sets the data buffer for the packet. |
| 9) void setLength(int length) | It sets the length of the packet. |
| 10) void setPort(int iport) | It sets the port number on the remote host to which the datagram is being sent. |
| 11) void setSocketAddress(SocketAddress addr) | It sets the SocketAddress (IP address + port number) of the remote host to which the datagram is being sent. |

Example of Sending DatagramPacket by DatagramSocket

1. `//DSender.java`
2. `import java.net.*;`
3. `public class DSender{`
4. `public static void main(String[] args) throws Exception {`
5. `DatagramSocket ds = new DatagramSocket();`
6. `String str = "Welcome java";`
7. `InetAddress ip = InetAddress.getByName("127.0.0.1");`

- 8.
9. DatagramPacket dp = **new** DatagramPacket(str.getBytes(), str.length(), ip, 3000);
10. ds.send(dp);
11. ds.close();
12. }
13. }

Output:



```
C:\>
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

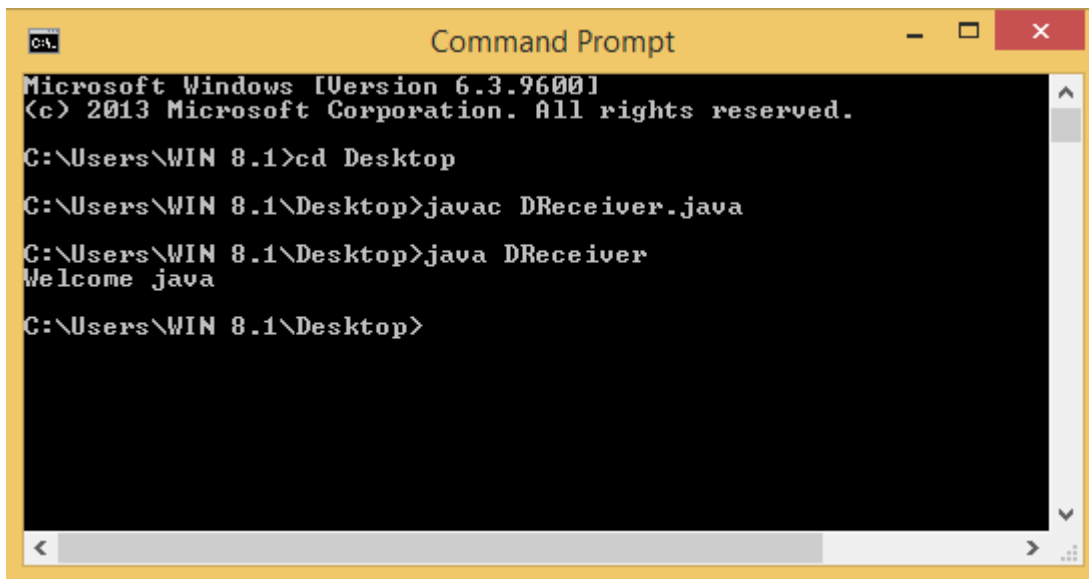
C:\Users\WIN 8.1>cd Desktop
C:\Users\WIN 8.1\Desktop>javac DSender.java
C:\Users\WIN 8.1\Desktop>java DSender
C:\Users\WIN 8.1\Desktop>
```

Example of Receiving DatagramPacket by DatagramSocket

1. //DReceiver.java
2. **import** java.net.*;
3. **public class** DReceiver{
4. **public static void** main(String[] args) **throws** Exception {
5. DatagramSocket ds = **new** DatagramSocket(3000);
6. **byte**[] buf = **new byte**[1024];
7. DatagramPacket dp = **new** DatagramPacket(buf, 1024);
8. ds.receive(dp);

```
9.   String str = new String(dp.getData(), 0, dp.getLength());
10.  System.out.println(str);
11.  ds.close();
12. }
13. }
```

Output:



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\WIN 8.1>cd Desktop
C:\Users\WIN 8.1\Desktop>javac DReceiver.java
C:\Users\WIN 8.1\Desktop>java DReceiver
Welcome java
C:\Users\WIN 8.1\Desktop>
```