**Java URL class:**

* Represents a Uniform Resource Locator, a pointer to a resource on the www.
* A resource can be file, directory, a database or to a search engine.
* URL class provides methods to obtain details about URL.



Protocol: it is http.

Host Name or Server name: IP Address + Port Number.

Port Number: is optional, if not mentioned it return -1.

File name or directory name: it is an index file.

Constructors of URL class:

URL(String spec) – creates an instance of URL of defined String.

URL(String protocol , String host, int port, String file) - creates instance of all parameters.

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

URL(String protocol, String host, -String file) –

URL(URL context, String spec) –

URL(URL context, String spec, URLStreamHandler handler) –

**Commonly used URL methods:**

**URL url=new URL(“** <https://www.google.com/>”**);**

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.

**Java URLConnection class:**

**Represents a communication link between the URL and the application, used to read and write data to specified resource referred by the URL.**

**openConnection()** method of **URL class** establishes the connection.

public URLConnection openConnection() method

returns the object of URLConnection class

getInputStream() method

Eg:

URL u=**new** URL(**"https://www.eposnow.com/uk/careers"**);  
  
URLConnection uc=u.openConnection();  
  
InputStream st=uc.getInputStream();

**Java HttpURLConnection class**: is http specific URLConnection, it work for HTTP protocols only.

Can be used to get information of any HTTP URL, such as header info, status code response body etc.

It is a subclass of URLConnection class.

URL u=**new** URL(**"https://www.eposnow.com/uk/careers"**);  
HttpURLConnection con=(HttpURLConnection) u.openConnection();

For(int i=1;i<8;i++)

Sout(con.getHeaderFieldKey(i) + “ = ”+con.getHeaderField(i));

}

con.disconnect();

**Java InetAddress class: represents an IP address.**

**Methods of InetAddress:**

public static InetAddress getByName(String host) throws UnknownHostException

it returns the instance of InetAddress containing LocalHost IP and name.

public static InetAddress getLocalHost() throws UnknownHostException

it returns the instance of InetAdddress containing local host name and address.

public String getHostName()

it returns the host name of the IP address.

public String getHostAddress()

it returns the IP address in string format.

Eg:

InetAddress ip=InetAddress.getByName("**https://www.eposnow.com/uk/careers** ");

System.out.println("Host Name: "+ip.getHostName());

System.out.println("IP Address: "+ip.getHostAddress());

**Java Networking**: ( java.net package):

* Is a concept of connecting two or more computing devices together so that we can share resource.
* Java Socket programming provides facility to share data between different computing devices.

Java Networking Terminology:

* **IP Address:**

Is a unique number assigned to a node of a network (logical address can be changed).

* **Protocol:**

Is a set of rules that followed for communication. (TCP, FTP, HTTP, SMTP etc.)

* **Port Number:**

Is a used to uniquely identify different applications, (it acts as communication endpoint associated with IP Address for communication between two applications).

* **MAC Address:**

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

* **Connection-oriented and Connection less protocol:**

In connection-oriented protocol acknowledgement is sent by the receiver, it is reliable but slow. Eg: TCP. In connection-less protocol, acknowledgement is not sent by receiver, not reliable but fast Eg: UDP.

* **Socket:**

A socket is an endpoint between two communications.

**Java Socket Programming:**

* Is used for communication between the applications running on different JRE.
* Can be connection-oriented or connection-less

**Socket** and **ServerSocket classes** are used for **connection-oriented** socket programming**.**

**DatagramSocket** and **DatagramPacket classes** used for **connection-less** socket programming**.**

**[**Note: Socket class can be used to create a Socket, it is simply an endpoint for communication

ServerSocket object is used to establish connection Server and Client]

The client in socket programming must know two information:

* IP Address of Server and
* Port Number.

1.Simple client-server application, server receives string from client: use two command prompts.

**Creating Server:**

*//Creating a Server*ServerSocket server=**new** ServerSocket(6666);  
Socket s=server.accept();  
DataInputStream dis=**new** DataInputStream(s.getInputStream());  
String str=(String) dis.readUTF();  
System.***out***.println(**" received message = "**+str);  
s.close();

**Creating Client:**

Socket client = **new** Socket(**"localhost"**, 6666);  
DataOutputStream dout=**new** DataOutputStream(client.getOutputStream());  
dout.writeUTF(**"HELLO SERVER"**);  
dout.flush();  
dout.close();

2. Simple client-server application read and write (chatting app):

**Creating Server:**

ServerSocket srv=**new** ServerSocket(5555);  
Socket sc=srv.accept();  
DataInputStream din=**new** DataInputStream(sc.getInputStream());  
DataOutputStream dout=**new** DataOutputStream(sc.getOutputStream());  
BufferedReader reader=**new** BufferedReader(**new** InputStreamReader(System.***in***));  
String input=**""**;  
String output=**""**;  
**while** (!input.equals(**"stop"**)){  
 input=din.readUTF();  
 System.***out***.println(**"Client says : "** + input);  
 output=reader.readLine();  
 dout.writeUTF(output);  
 dout.flush();  
}  
din.close();  
sc.close();  
srv.close();

**Creating Client:**

Socket cl=**new** Socket(**"localhost"**,5555);  
DataInputStream din=**new** DataInputStream(cl.getInputStream());  
DataOutputStream dout=**new** DataOutputStream(cl.getOutputStream());  
BufferedReader reader=**new** BufferedReader(**new** InputStreamReader(System.***in***));  
String input=**""**;  
String output=**""**;  
**while** (!input.equals(**"stop"**)){  
 input=reader.readLine();  
 dout.writeUTF(input);  
 dout.flush();  
 output=din.readUTF();  
 System.***out***.println(**"Server says : "**+output);  
}  
dout.close();  
cl.close();

**DatagramSocket and DatagramPacket classes:** represents a connection less socket for sending and receiving datagram packets. (there is no guarantee of contents arrival or arrival time).

**Sending DatagramPacket by DatagramSocket:**

DatagramSocket ds = new DatagramSocket();

String str = "Welcome java";

InetAddress ip = InetAddress.getByName("127.0.0.1");

DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);

ds.send(dp);

ds.close();

**Receiving DatagramPacket by DatagramSocket:**

DatagramSocket ds = new DatagramSocket(3000);

byte[] buf = new byte[1024];

DatagramPacket dp = new DatagramPacket(buf, 1024);

ds.receive(dp);

String str = new String(dp.getData(), 0, dp.getLength());

System.out.println(str);

ds.close();