# Unit 5:
# Network Programming

**Java Networking :**

- Java Networking is a concept of connecting two or more computing devices together so that we can share resources.
- The application layer of a Java program communicates with the network.
- The java.net package contains all of the Java networking classes and interfaces.

**Common Network Protocols:**

The java.net package supports 2 protocols. These are their names:

**1. The Transmission Control Protocol (TCP):**

❖ TCP enables secure communication between two applications. It is commonly applied together with the Internet Protocol, which is referred to as TCP/IP.

**2. User Datagram Protocol (UDP):**

❖ UDP is a connectionless protocol allowing data packets to be sent between applications.

## 1. TCP (Transmission Control Protocol)

- TCP (Transmission Control Protocol) is a connection-based protocol that provides a reliable flow of data between two computers. When two applications want to communicate to each other reliably, they establish a connection and send data back and forth over that connection.

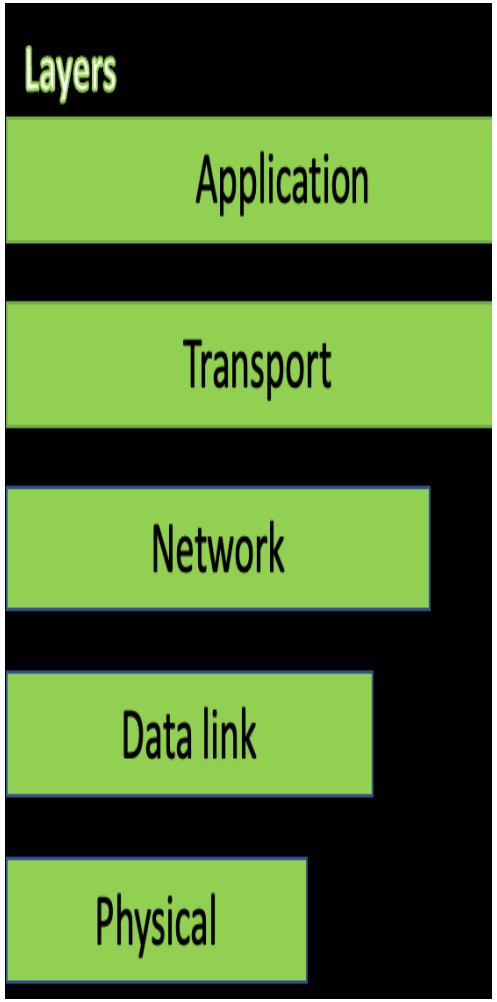
- This connection establishment is similar to making a telephone call. Like a phone call, data is sent back and forth over the established connection.

- TCP guarantees that data sent from one end of the connection actually gets to the other end and in the same order it was sent. Otherwise, an error is reported.

- TCP provides a point-to-point channel for applications requiring reliable communications.

- Examples of such applications include Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Telnet.

**TCP Layers:**




**Application Layer — Human Interaction**

- The Application Layer is where applications and users interact with the network.
- It includes protocols like HTTP for browsing the web, SMTP for sending emails, and FTP for sharing files.
- This layer enables different programs and services to communicate over the network.

**Transport Layer — Transmission Protocols**

- The Transport Layer enables communication between devices.
- It establishes connections between them and makes sure data is delivered reliably and efficiently.
- The main protocol in this layer is TCP, which ensures data is sent accurately and in the right order.

**Network Layer — Physical Path**

- The Network Layer is responsible for addressing, routing, and logical organization of data packets for transmission across different networks.
- It uses IP addresses to uniquely identify devices on the network and determines the most optimal path for data to reach its intended destination.
- This layer handles tasks such as packet fragmentation and reassembly to accommodate different network sizes and technologies.

**TCP Layers:**

**Data Link Layer — Format of Data**

- The Data Link Layer provides reliable and error-free data transfer between directly connected devices on the same network.
- It takes the data from the Network Layer and organizes it into manageable units called frames.
- This layer also handles tasks such as error detection and correction, flow control, and medium access control.
- It ensures that the data is efficiently and accurately transmitted over the physical link.

**Physical Layer — Transmits Raw Bit**

- The Physical Layer is responsible for the actual transmission of data over the network.
- It deals with the physical aspects of communication, including the hardware components such as cables, connectors, and network interface cards.
- This layer ensures that the signals representing the data can be accurately transmitted and received between devices.

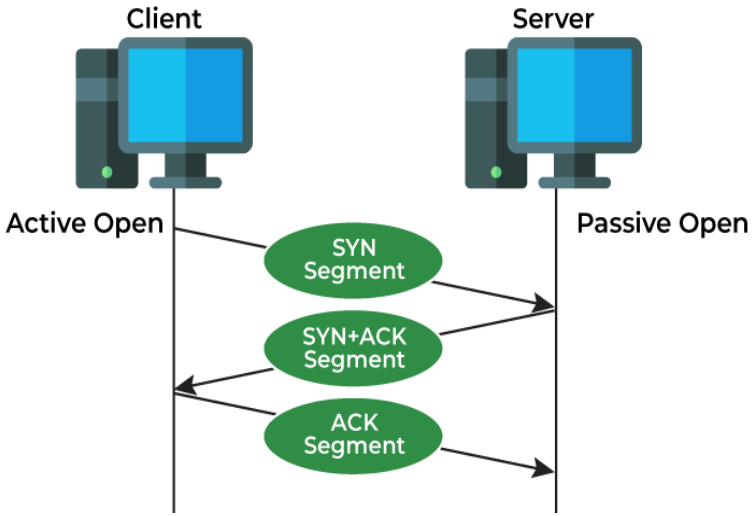


**TCP performs the following actions:**

- Determines how to break application data into packets that networks can deliver.
- Sends packets to, and accepts packets from, the network layer.
- Manages flow control.
- Handles retransmission of dropped or garbled packets, as it's meant to provide error-free data transmission; and acknowledges all packets that arrive.

**Advantages of TCP**

- It is reliable for maintaining a connection between Sender and Receiver.
- It is responsible for sending data in a particular sequence.
- Its operations are not dependent on OS.
- It allows and supports many routing protocols.
- It can reduce the speed of data based on the speed of the receiver.

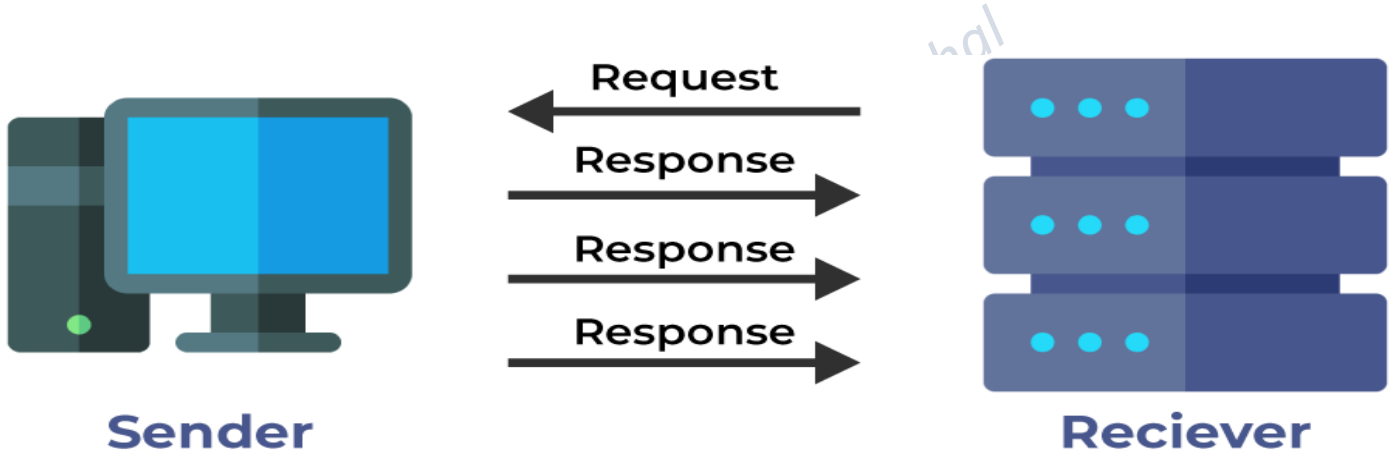



**Disadvantages of TCP**

- It is slower than UDP and it takes more bandwidth.
- Slower upon starting of transfer of a file.
- Not suitable for LAN and PAN Networks.
- It does not have a multicast or broadcast category.
- It does not load the whole page if a single data of the page is missing.

**2. User Datagram Protocol (UDP) :**

- It is a communications protocol for time-sensitive applications like gaming, playing videos, or Domain Name System (DNS) lookups.
- UDP results in speedier communication because it does not spend time forming a firm connection with the destination before transferring the data. Because establishing the connection takes time, eliminating this step results in faster data transfer speeds.
- However, UDP can also cause data packets to get lost as they go from the source to the destination. It can also make it relatively easy for a hacker to execute a distributed denial-of-service (DDoS) attack.



**Advantages of UDP:**

- It does not require any connection for sending or receiving data.
- Broadcast and Multicast are available in UDP.
- UDP can operate on a large range of networks.
- UDP has live and real-time data.
- UDP can deliver data if all the components of the data are not complete.

**Disadvantages of UDP :**

- We can not have any way to acknowledge the successful transfer of data.
- UDP cannot have the mechanism to track the sequence of data.
- UDP is connectionless, and due to this, it is unreliable to transfer data.
- In case of a Collision, UDP packets are dropped by Routers in comparison to TCP.
- UDP can drop packets in case of detection of errors.

**Which Protocol is Better: TCP or UDP?**

- It is difficult because it totally depends on what work we are doing and what type of data is being delivered.
- UDP is better in the case of online gaming as it allows us to work lag-free.
- TCP is better if we are transferring data like photos, videos, etc. because it ensures that data must be correct has to be sent.

**Where TCP is Used?**

- Sending Emails
- Transferring Files
- Web Browsing

**Where UDP is Used?**

- Gaming
- Video Streaming
- Online Video Chats

| Basis | Transmission Control Protocol (TCP) | User Datagram Protocol (UDP) |
|---|---|---|
| Type of Service | TCP is a connection-oriented protocol. Connection orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data. | UDP is the Datagram-oriented protocol. This is because there is no overhead for opening a connection, maintaining a connection, or terminating a connection. UDP is efficient for broadcast and multicast types of network transmission. |
| Reliability | TCP is reliable as it guarantees the delivery of data to the destination router. | The delivery of data to the destination cannot be guaranteed in UDP. |
| Error checking mechanism | TCP provides extensive error-checking mechanisms. It is because it provides flow control and acknowledgment of data. | UDP has only the basic error-checking mechanism using checksums. |
| Acknowledgment | An acknowledgment segment is present. | No acknowledgment segment. |
| Sequence | Sequencing of data is a feature of Transmission Control Protocol (TCP). this means that packets arrive in order at the receiver. | There is no sequencing of data in UDP. If the order is required, it has to be managed by the application layer. |
| Speed | TCP is comparatively slower than UDP. | UDP is faster, simpler, and more efficient than TCP. |
| Retransmission | Retransmission of lost packets is possible in TCP, but not in UDP. | There is no retransmission of lost packets in the User Datagram Protocol (UDP). |
| Header Length | TCP has a (20-60) bytes variable length header. | UDP has an 8 bytes fixed-length header. |
| Weight | TCP is heavy-weight. | UDP is lightweight. |
| Handshaking Techniques | Uses handshakes such as SYN, ACK, SYN-ACK | It’s a connectionless protocol i.e. No handshake |
| Broadcasting | TCP doesn’t support Broadcasting. | UDP supports Broadcasting. |
| Protocols | TCP is used by HTTP, HTTPs, FTP, SMTP and Telnet. | UDP is used by DNS, DHCP, TFTP, SNMP, RIP, and VoIP. |
| Stream Type | The TCP connection is a byte stream. | UDP connection is a message stream. |
| Applications | This protocol is primarily utilized in situations when a safe and trustworthy communication procedure is necessary, such as in email, on the web surfing, and in military services. | This protocol is used in situations where quick communication is necessary but where dependability is not a concern, such as VoIP, game streaming, video, and music streaming, etc. |

**Java Networking Terminology**

**1. IP Address:**

- An IP address is a unique address that distinguishes a device on the internet or a local network.
- IP stands for “Internet Protocol.”
- It comprises a set of rules governing the format of data sent via the internet or local network.
- IP Address is referred to as a logical address that can be modified.
- It is composed of octets.
- The range of each octet varies from 0 to 255.
- Range of the IP Address – 0.0.0.0 to 255.255.255.255
- For Example – 192.168.0.1

**2. Port Number :**

- A port number is a method to recognize a particular process connecting internet or other network information when it reaches a server.
- The port number is used to identify different applications uniquely.
- The port number behaves as a communication endpoint among applications.
- The port number is correlated with the IP address for transmission and communication among two applications.
- There are 65,535 port numbers, but not all are used every day.

**3. Socket:**

- A socket in computer network programming is an endpoint for sending or receiving data across a computer network. It provides an interface for programs to communicate with each other over a network, using various network protocols like TCP (Transmission Control Protocol) or UDP (User Datagram Protocol).

**Socket Types:**

**1. Stream Sockets (TCP Sockets):**

- Stream sockets are used for reliable, connection-oriented communication. They are associated with the TCP protocol (Transmission Control Protocol).

**2. Datagram Sockets (UDP Sockets):**

- UDP sockets provide a connectionless, message-based communication model, where each message is sent as an independent packet, and there is no guarantee of delivery, order, or error checking.

**Socket programming**

- Socket programming in Java facilitates establishing network connections, sending and receiving data over a network, and communication between devices or processes.
- It is a fundamental building block that allows seamless data exchange and interaction between devices or processes in a networked environment.
- Socket programming in Java is an essential element of network programming that empowers developers to create applications that are capable of communicating with other systems through the Internet or a local network.

Socket programming process involves the following steps:

**1. Server Setup:**

- The server application creates a ServerSocket and binds it to a specific port on the machine. The server socket listens for incoming client connections.

**2. Data Transfer:**

- Once the connection is established, data can be sent or received between the client and the server using input and output streams associated with the socket.

**3. Client Connection:**

- The client application creates a Socket and specifies the server's IP address and port number.

**4. Connection Termination:**

- After the data exchange is complete or when either the client or the server decides to end the connection, the sockets are closed, terminating the connection.

**5. Establishing Connection:**

- The client sends a connection request to the server; if the server is listening and can accept the connection, a connection is established between the client and the server.

**Socket programming using TCP:**

- Socket programming using TCP (Transmission Control Protocol) in Java involves creating a client-server application where two Java programs communicate with each other over a network using sockets.

**Steps to create Server Program:**

**1. Server Setup:**

```
ServerSocket serverSocket = new ServerSocket(port);
```

**2. Accepting Client Connection**

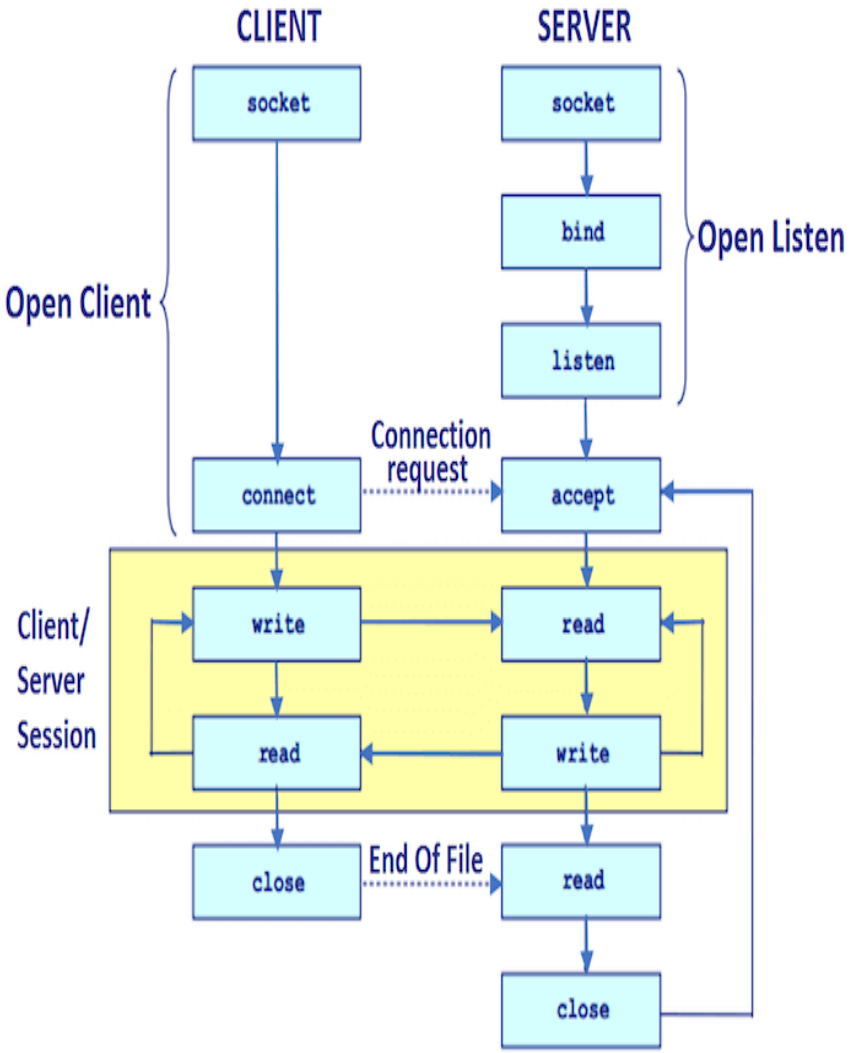
```
Socket clientSocket = serverSocket.accept();
```

**3. Input and Output Streams**

```
BufferedReader clientIn = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

BufferedWriter clientOut = new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream()));

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



**4. Message Exchange Loop**

```
String clientMessage;
String serverMessage;

while (true) {
clientMessage = clientIn.readLine();
System.out.println("Client says: " + clientMessage);
serverMessage = serverIn.readLine();
```

**5. Sending a Response**

```
clientOut.write(serverMessage + "\n");
clientOut.flush();
}
```

**Steps to create Client Program:**

**1. Client Setup**

```
Socket clientSocket = new Socket(serverAddress, serverPort);
```

**2. Input and Output Streams**

```
BufferedReader serverIn = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
BufferedWriter serverOut = new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));
BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));
```

**3. Message Exchange Loop**

```
while(true) {
     System.out.println("Client Request : ");
     clientMessage = clientIn.readLine();
     serverOut.write(clientMessage);
     serverOut.flush();
```

**4. Displaying Server Responses**

```
     serverMessage =  serverIn.readLine();
     System.out.println("Server Response: "+serverMessage);
}
```

**Socket programming using UDP:**

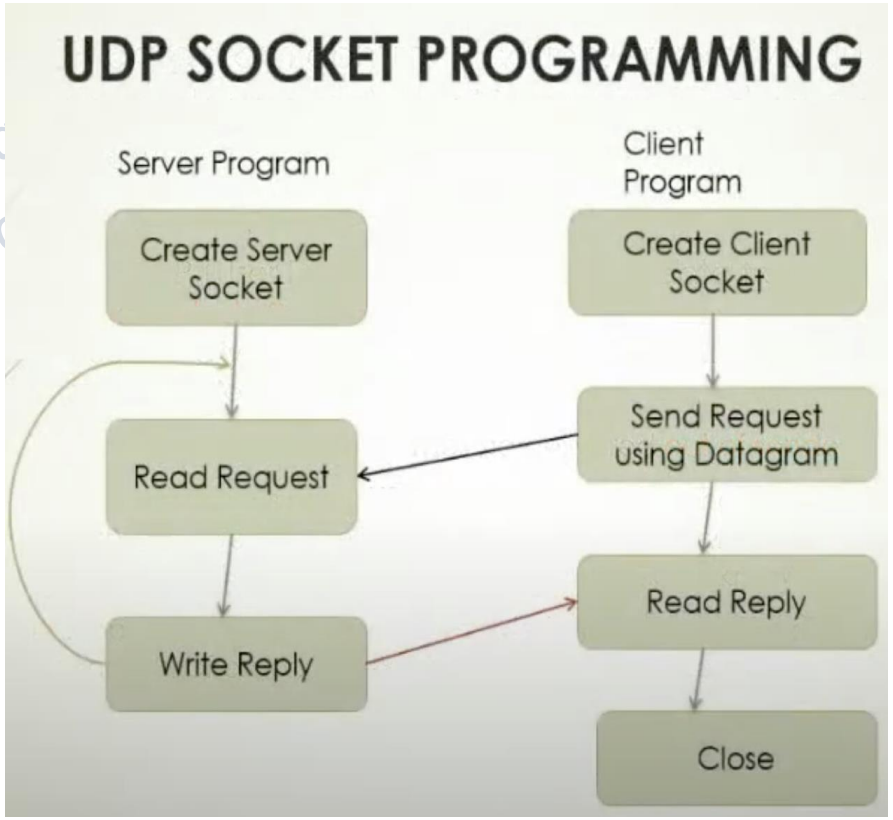
- Socket programming using UDP (User Datagram Protocol) in Java involves creating a client and server to exchange data over a network.
- UDP is a connectionless, low-level protocol that allows for fast data transmission but does not guarantee the delivery of data or its order.
- Java supports datagram communication through the following classes:
  - DatagramPacket
  - DatagramSocket

**DatagramPacket:**

- It represent Datagram.
- Datagram is an independent, self-contained message sent over the network whose arrival time and content are not guaranteed.

- **DatagramSocket:**
  - It is used to send and receive DatagramPackets.
  - It has two implement methods to Read and Send packets.
- Recive
- Send





**Steps to create Server Program:**

**1. Create a DatagramSocket :**

```
DatagramSocket serverSocket = new DatagramSocket(port);
```

**2. Read a request from socket :**

```
byte[] messsgae = new byte[20000];
DatagramPacket  datagramPacket = new DatagramPacket(messsgae, messsgae.length);
serverSocket.receive(datagramPacket);

String result =  new String(datagramPacket.getData());
System.out.println("Message :  "+result);
```

**3. Write a replay message with client host address and port number**

```
String reply = "";
byte[] replys =  reply.getBytes();
InetAddress inetAddress = InetAddress.getLocalHost();
DatagramPacket replyPacket = new DatagramPacket(replys, replys.length, inetAddress, datagramPacket.getPort());
serverSocket.send(replyPacket);
```

**4. Continue to read request**

**Steps to create Client Program:**

1. Create a DatagramSocket for the client

```
DatagramSocket clinetSocket = new DatagramSocket();
```

**2. Send request from socket**

```
String message = "";
byte[]  messages = message.getBytes();
InetAddress inetAddress = InetAddress.getByName("localhost");
DatagramPacket datagramPacket  = new DatagramPacket(messages, messages.length, inetAddress,port);
clinetSocket.send(datagramPacket);
```

**3. Read reply message:**

```
byte[] reMessage = new byte[20000];
DatagramPacket  rePacket = new DatagramPacket(reMessage, reMessage.length);
clinetSocket.receive(rePacket);
String result =  new String(rePacket.getData());
System.out.println("Reply from Server :  "+result);
```



**InetAddress:**

- The InetAddress is Java's representation of an IP address. Instance of this class are used together with UDP DatagramSockets and normal. Socket's and ServerSocket's.
- InetAddress has no public constructor.

**URL Class in Java:**

- URL known as Uniform Resource Locator is simply a string of text that identifies all the resources on the Internet, telling us the address of the resource, how to communicate with it, and retrieve something from it.

- URL consists of the Base URL, Resource, Query, or Path Parameters.


## Constructors of the URL class:

**1. URL(String address) throws MalformedURLException:** It creates a URL object from the specified String.

**2. URL(String protocol, String host, String file):** Creates a URL object from the specified protocol, host, and file name.

**3. URL(String protocol, String host, int port, String file):** Creates a URL object from protocol, host, port, and file name.

**4. URL(URL context, String spec):** Creates a URL object by parsing the given spec in the given context.

**5. URL(String protocol, String host, int port, String file, URLStreamHandler handler):**
Creates a URL object from the specified protocol, host, port number, file, and handler.

**6. URL(URL context, String spec, URLStreamHandler handler):**
Creates a URL by parsing the given spec with the specified handler within a specified 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. |

**Java URLConnection**

- 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.

Commonly used methods of the URLConnection class:

1. getInputStream(): Returns an input stream from the connection to read data from the resource.
2. getContentEncoding(): Returns the value of the content-encoding header field.
3. setAllowUserInteraction(boolean value): Set the value of the allowUserInteraction field of this URLConnection.
4. setDoInput(boolean doinput): Specifies whether this URLConnection allows input. The default is true.
5. setDoOutput(boolean dooutput): Specifies whether this URLConnection allows output. The default is false.

**e.g.**

```
URL url = new URL("http://google.com");
URLConnection urlConnection = url.openConnection();
System.out.println(urlConnection.getContentEncoding());
urlConnection.setAllowUserInteraction(true);
BufferedReader br = new BufferedReader(new InputStreamReader(urlConnection.getInputStream()));
String response;
while ((response = br.readLine()) != null) {
System.out.println(response);
}
br.close();
```

**Java Mail API:**

- The **JavaMail** is an API that is used to compose, write and read electronic messages (emails).
- The JavaMail API provides protocol-independent and plateform-independent framework for sending and receiving mails.
- The **javax.mail** and **javax.mail.activation** packages contains the core classes of JavaMail API.

**Steps to send email using JavaMail API**

There are following three steps to send email using JavaMail. They are as follows:

1. Get the session object that stores all the information of host like host name, username, password etc.
2. Compose the message
3. Send the message

For sending the email using JavaMail API, load the two jar files:

1. mail.jar
2. activation.jar



**Maven:**

```
<dependency>
   <groupId>com.sun.mail</groupId>
   <artifactId>javax.mail</artifactId>
   <version>1.6.2</version>
</dependency>
```

**Step 1: Get the session object**

The javax.mail.Session class provides two methods to get the object of session, Session.getDefaultInstance() method and Session.getInstance() method. You can use any method to get the session object.

```
 Properties props = new Properties();
props.put("mail.smtp.host", "smtp.gmail.com");
props.put("mail.smtp.socketFactory.port", "465");
props.put("mail.smtp.socketFactory.class",  "javax.net.ssl.SSLSocketFactory");
props.put("mail.smtp.auth", "true");
props.put("mail.smtp.port", "465");

  Session session = Session.getDefaultInstance(props,  new javax.mail.Authenticator() {
    protected PasswordAuthentication getPasswordAuthentication() {
   return new PasswordAuthentication(user,password);
    }
   });
```



**Step 2 : Compose the message**

To create the message, you need to pass session object in MimeMessage class constructor.

```
MimeMessage message = new MimeMessage(session);
message.setFrom(from);
message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
message.setSubject(subject);
message.setText(bodyMessage);
```

**Step 3: Send the message:**
The javax.mail.Transport class provides method to send the message.

Transport.send(message);

**Receiving email in Java:**

For receiving email Store and Folder classes are used in collaboration with MimeMessage, Session and Transport classes.

There are 5 steps to receive the email using JavaMail API. They are:

**1.Get the session object:**
Properties properties = new Properties();
properties.put("mail.pop3.host", pop3Host);
Session emailSession = Session.getDefaultInstance(properties);



**2. create the POP3 store object and connect with the pop server**
Store store = session.getStore("pop3");
store.connect(host, username, password);

**3. create the folder object and open it**
Folder inbox = store.getFolder("INBOX");
inbox.open(Folder.READ_ONLY);

**4. retrieve the messages from the folder in an array and print it**

```
Message[] messages = emailFolder.getMessages();
for (int i = 0; i < messages.length; i++) {
Message message = messages[i];
System.out.println("---------------------------------");
System.out.println("Email Number " + (i + 1));
System.out.println("Subject: " + message.getSubject());
System.out.println("From: " + message.getFrom()[0]);
System.out.println("Text: " + message.getContent().toString());
}
```

**5. close the store and folder objects**

```
emailFolder.close(false);
emailStore.close();
```