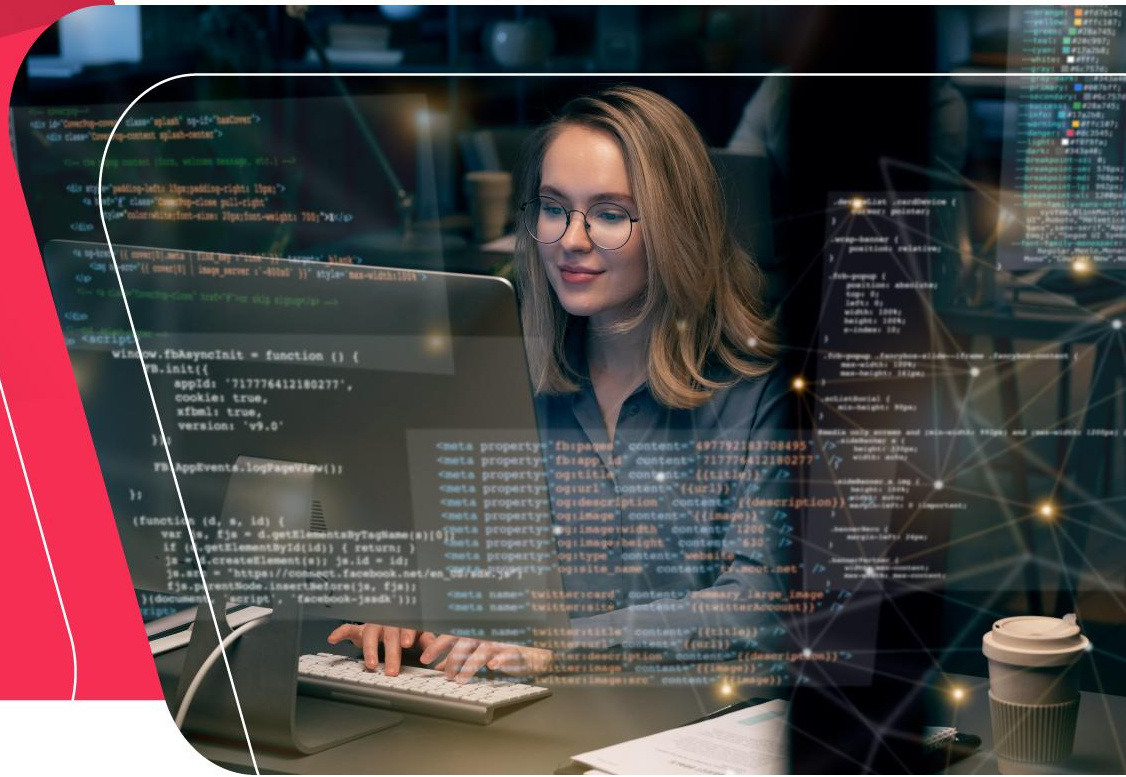
TIA-Academy

Java Bootcamp

Day 21



Technologies will be Use

- JDK 8/**11**/15
- JRE 8/**11**/15
- **Intellij IDEA Community Edition**
- **JAVA Networking (Socket, FTP, etc)**

TIA-Academy

Networking

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 facility to share data between different computing devices.
- Advantage of Java Networking
    1. sharing resources
    2. centralize software management

Java Networking

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

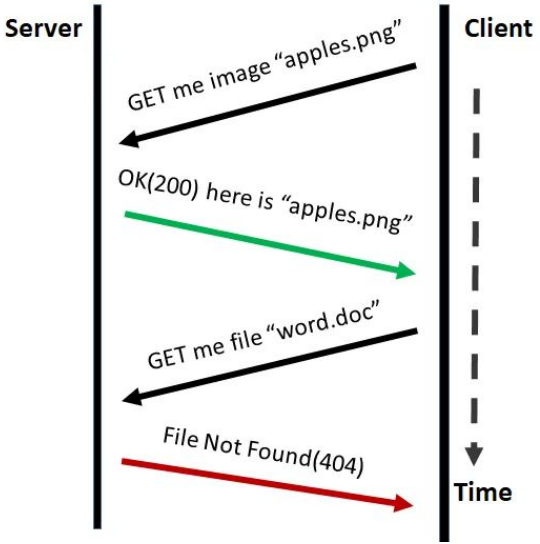
Java Networking Terminology

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.

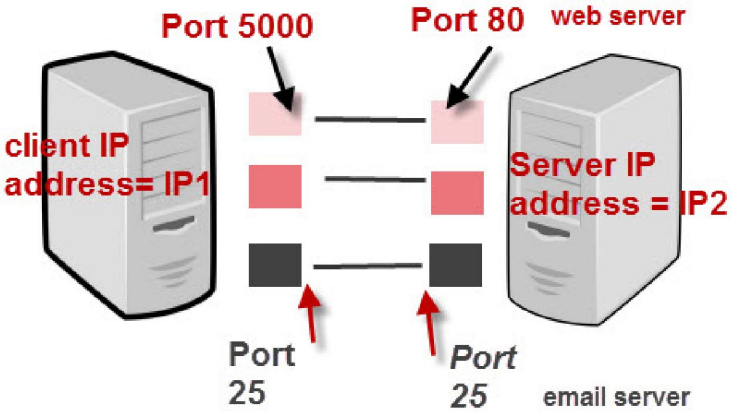
Java Networking Terminology

2. Protocol
   - A protocol is a set of rules basically that is followed for communication. For example:
     - TCP
     - FTP
     - Telnet
     - SMTP
     - POP etc.



Java Networking Terminology

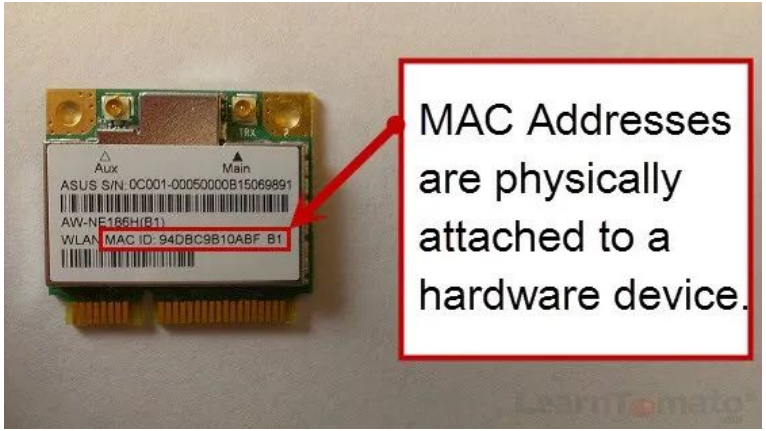
3. Port Number
   - The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.
   - The port number is associated with the IP address for communication between two applications.



Java Networking Terminology

4. MAC Address

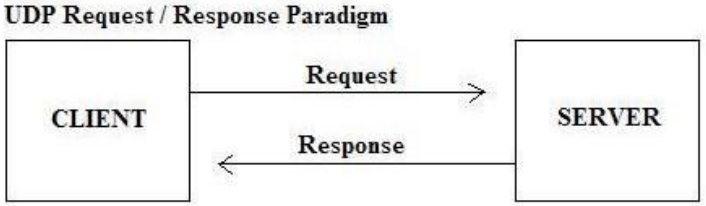
- MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

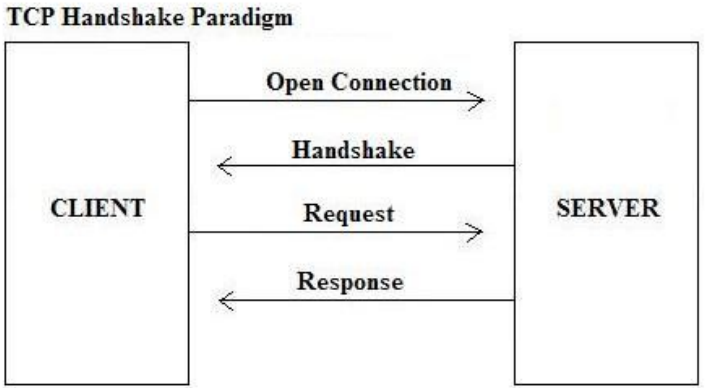


Java Networking Terminology

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.

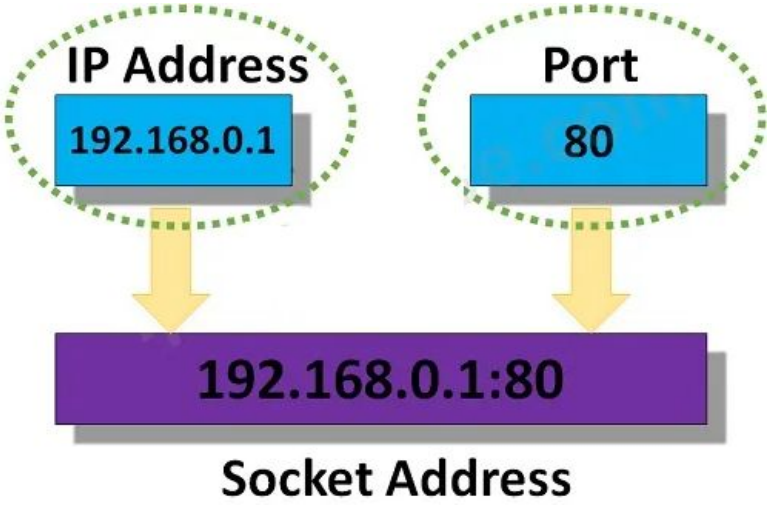




Java Networking Terminology

6. Socket

- A socket is an endpoint between two way communication..



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

Socket class

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

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.

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

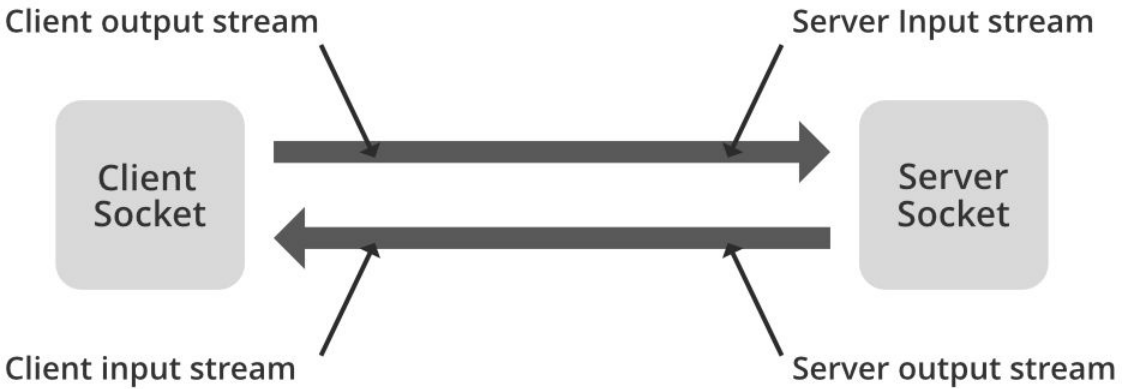
```
import java.io.*;
import java.net.*;
public class MyServer {
	public static void main(String[] args){
		try{
			ServerSocket ss=new ServerSocket(6666);
			Socket s=ss.accept();//establishes connection
			DataInputStream dis=new DataInputStream(s.getInputStream());
			String  str=(String)dis.readUTF();
			System.out.println("message= "+str);
			ss.close();
		}catch(Exception e){System.out.println(e);}
	}
}
```

Example of Java Socket Programming

```
import java.io.*;
import java.net.*;
public class MyClient {
      public static void main(String[] args) {
            try{
                  Socket s=new Socket("localhost",6666);
                  DataOutputStream dout=new DataOutputStream(s.getOutputStream());
                  dout.writeUTF("Hello Server");
                  dout.flush();
                  dout.close();
                  s.close();
            }catch(Exception e){System.out.println(e);}
      }
}
```

Java Socket Programming (Read-Write both

- 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



TIA-Academy

ASSIGNMENT 01

TIA-Academy

ASSIGNMENT 02

TIA-Academy

Networking Continued

Java FTP file upload

- To write Java code that uploads a file from local computer to a remote FTP server, the **Apache Commons Net API** is a preferred choice of developers.
- It has simple and comprehensive API that makes coding with upload files to FTP server with ease.

Apache Commons Net API for uploading files by FTP protocol

The `FTPClient` class provides six `storeXXX()` methods for transferring a local file to a remote server via FTP protocol:

- boolean **storeFile**(String remote, *InputStream* local)
- *OutputStream* **storeFileStream**(String remote)
- boolean **storeUniqueFile**(*InputStream* local)
- boolean **storeUniqueFile**(String remote, *InputStream* local)
- *OutputStream* **storeUniqueFileStream**()

Apache Commons Net API for uploading files by FTP protocol

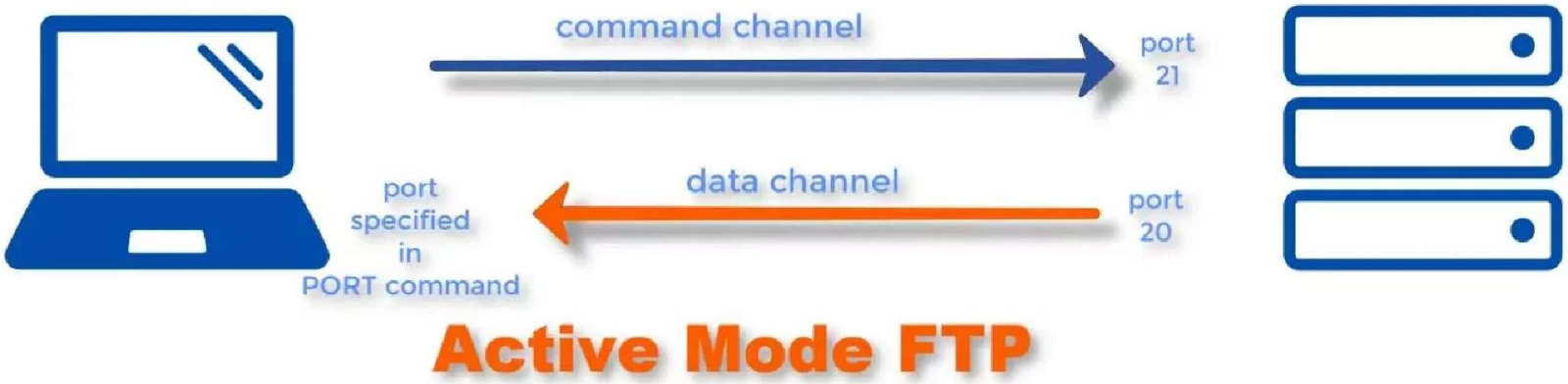
Sounds too much? What is the difference among these methods? When to use which one? Well, they can be categorized by the following means:

- **Store files by providing an InputStream of the local file** (those methods which have an InputStream as a parameter). This type of methods can be used when **we don't care how the bytes are transferred** from the local file to the remote one, just let the system done the ins and outs.

- **Store files by writing to an OutputStream of the connection** (those methods which return an OutputStream). This type of methods is needed when **we want to control how the bytes are transferred**, by writing our own code for reading bytes from the local file and write these bytes to the remote file through the OutputStream object. This can be useful if we want to show progress of the upload, by calculating how many bytes are transferred over total bytes needed.

Apache Commons Net API for uploading files by FTP protocol

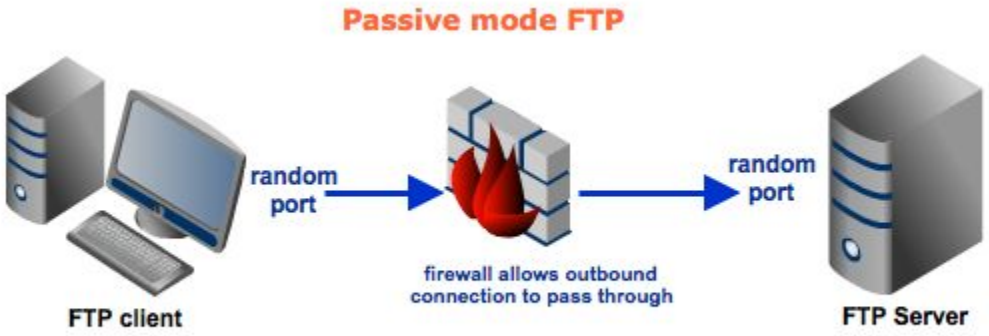
- By default, the FTP protocol establishes a data connection by opening a port on the client and allows the server connecting to this port. This is called *local active mode*, but it is usually blocked by firewall so the file transfer may not work.



Apache Commons Net API for uploading files by FTP protocol

- Fortunately, the FTP protocol has another mode, *local passive mode*, in which a data connection is made by opening a port on the server for the client to connect – and this is not blocked by firewall.

- So it is recommended to switch to *local passive mode* before transferring data, by invoking the method enterLocal



The proper steps to upload a file

To properly write code to upload files to a FTP server using Apache Commons Net API, the following steps should be followed:

1. Connect and login to the server.
2. Enter local passive mode for data connection.
3. Set file type to be transferred to binary.
4. Create an InputStream for the local file.
5. Construct path of the remote file on the server. The path can be absolute or relative to the

The proper steps to upload a file

7. Call one of the storeXXX()methods to begin file transfer. There are two scenarios:

   - Using an InputStream-based approach: this is the simplest way, since we let the system does the ins and outs. There is no additional code, just passing the InputStream object into the appropriate method, such as storeFile(String remote, InputStream local) method.

   - Using an OutputStream-based approach: this is more complex way, but more control. Typically we have to write some code that reads bytes from the InputStream of the local file and writes those bytes into the OutputStream which is returned by the storeXXX() method, such as storeFileStream(String remote) method.

The proper steps to upload a file

7. Close the opened InputStream and OutputStream.

8. Call completePendingCommand() method to complete transaction.

9. Logout and disconnect from the server.

The proper steps to upload a file

FTPUploadFileDemo.java

Java FTP file download

- With the help of **Apache Commons Net API**, it is easy to write Java code for downloading a file from a remote FTP server to local computer.
- Here, you will learn how to properly implement Java code to get files downloaded from a server via FTP protocol.

boolean retrieveFile(String remote, OutputStream local):

- This method retrieves a remote file whose path is specified by the parameter remote, and writes it to the OutputStream specified by the parameter local.
- The method returns true if operation completed successfully, or false otherwise.
- This method is suitable in case we don’t care how the file is written to disk, just let the system use the given OutputStream to write the file.
- We should close OutputStream the after the method returns.

InputStream retrieveFileStream(String remote):

- This method does not use an OutputStream, instead it returns an InputStreamwhich we can use to read bytes from the remote file.
- This method gives us more control on how to read and write the data. But there are two important points when using this method:
  - The method completePendingCommand() must be called afterward to finalize file transfer and check its return value to verify if the download is actually done successfully.
  - We must close the InputStream explicitly.

Apache Commons Net API for downloading files by FTP protocol

- The first method provides the simplest way for downloading a remote file, as just passing an OutputStream of the file will be written on disk.

- The second method requires more code to be written, as we have to create a new OutputStream for writing file's content while reading its byte arrays from the returned InputStream. This method is useful when we want to measure progress of the download, i.e. how many percentages of the file have been transferred. In addition, we have to call the completePendingCommand()to finalize the download.

- Both the methods throw an IOException exception (or one of its descendants, FTPConnectionClosedExceptionand CopyStreamException). Therefore, make sure to handle these exceptions when calling the methods.

The proper steps to download a file

1. Connect and login to the server.
2. Enter local passive mode for data connection.
3. Set file type to be transferred to binary.
4. Construct path of the remote file to be downloaded.

The proper steps to download a file

5. Create a new OutputStream for writing the file to disk.

   - If using the first method (retrieveFile):
     - Pass the remote file path and the OutputStream as arguments of the method retrieveFile().
     - Close the OutputStream.
     - Check return value of retrieveFile() to verify success.
   - If using the second method (retrieveFileStream):
     - Retrieve an InputStream returned by the method retrieveFileStream().
     - Repeatedly a byte array from the InputStream and write these bytes into the OutputStream, until the InputStream is empty.
     - Call completePendingCommand() method to complete transaction.
     - Close the opened OutputStream the InputStream.
     - Check return value of completePendingCommand() to verify success.

6. Logout and disconnect from the server.

The proper steps to download a file

FTPDownloadFileDemo.java

Java FTP list directory

- The `FTPClient` class provides the following methods for listing content of a directory on the FTP server:
  - `FTPFile[] listDirectories()`
  - `FTPFile[] listDirectories(String parent)`

Method	List files?	List directories?	List in current working directory?	List in a specified directory?
`listDirectories()`	No	**Yes**	**Yes**	No
`listDirectories(String parent)`	No	**Yes**	No	**Yes**
`listFiles()`	**Yes**	**Yes**	**Yes**	No
`listFiles(String pathname)`	**Yes**	**Yes**	No	**Yes**

Java FTP list directory

- All the above methods return an array of FTPFile objects which represent files and directories. The FTPFile class provides various methods for querying detailed information of a file or directory, to name some useful ones:
  - `String getName()`: returns name of the file or directory.
  - `long getSize()`: return size in bytes.
  - `boolean isDirectory()`: determines if the entry is a directory.
  - `boolean isFile()`: determines if the entry is a file.
  - `Calendar getTimestamp()`: returns last modified time.

Java FTP list directory

- The FTPClient class also has two simple methods for listing files and directories:
  - `String[] listNames()`
  - `String[] listNames(String pathname)`
- Unlike the `listFiles()` and `listDirectories()` methods, the `listNames()` methods simply return an array of String represents file/directory names; and they list both files and directories.

TIA-Academy

ASSIGNMENT 03

TIA-Academy

ASSIGNMENT 04
(HOME ASSIGNMENT)

TIA-Academy

Thank You