CIE-356T

Advanced Java Programming

Network Programming in Java

B.Tech (CSE)
January, 2025

Nihar Ranjan Roy
Associate Professor,
VIPS School of Engineering & Technology
nihar.roy@vips.edu

VIPS
योगः कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

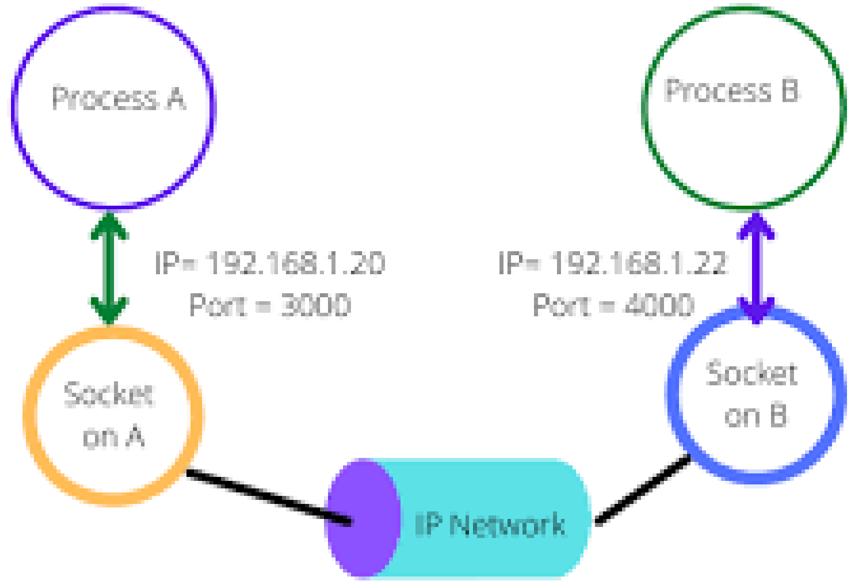
Networking protocols

What is a protocol?

- TCP/IP
- UDP

TCP/IP

- Connection oriented protocol
- Sockets are used for TCP/IP communicat
- What is socket?



Classes used

- URL
- URLConnection
- Socket
- ServerSocket

UDP

- Sends independent packets of data called datagrams from one computer to another
- Connectionless protocol
- No guarantees of arrival of data
- Similar to sending a letter without acknowledgement through postal service

Classes used in UDP comm

- DatagramPacket
- DatagramSocket
- MulticasteSockets

Difference between TCP and UDP

TCP	UDP
reliable	unreliable
Data to be sent is in continuous order, point to point eg telephone call	Packets are sent as datagrams in random order
Over head involved is more	Less overhead
Slower	Faster
Data must be received in order in which it was sent	Each message is independent of the other
Connection oriented	Connection less

How to identify computers on network?

IP Address

A series of four 8 bit numbers

Example 172.16.2.201

Max rage for each part is 0-255 as 8 bits is the limitations

Network Programming in Java

Java networking revolves around:

- **Sockets & ServerSockets** → For communication between two systems
- **URLs & HTTP Connections** → For interacting with web servers
- **Datagrams (UDP Sockets)** → For connectionless communication

Establish http url connection

Making a URL Connection

- Import
  - You need `java.net.*` for network operations and `java.io.*` for reading the response.
- Create a URL Object

```
URL url = new URL("http://example.com");
```

- Open a Connection

```
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
```

- Set Request Properties (Optional)

```
connection.setRequestMethod("GET");   // Use GET, POST, etc.
connection.setConnectTimeout(5000);   // Timeout in milliseconds
connection.setReadTimeout(5000);
connection.setRequestProperty("User-Agent", "Mozilla/5.0");
```

- Connect to the Server
  - `connection.connect();`

Making a URL Connection…

- Get the Response Code (Optional)

```
int responseCode = connection.getResponseCode();
if (responseCode == HttpURLConnection.HTTP_OK) { // 200 OK
    System.out.println("Connected Successfully!");
}
```

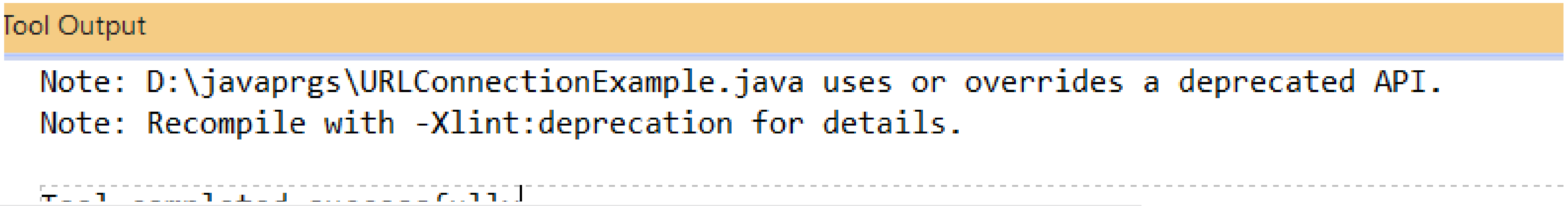
- Read the Response Data

```
BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
String line;
while ((line = reader.readLine()) != null) {
    System.out.println(line); // Print the response
}
reader.close();
```

Example-1

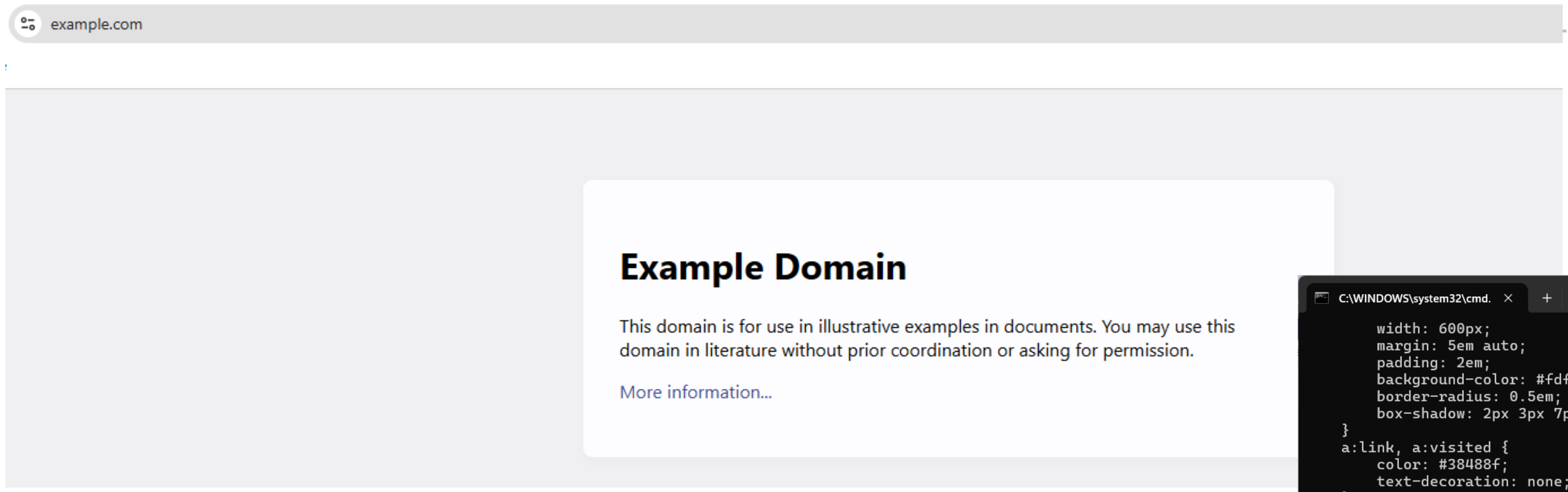
```
import java.io.*;
import java.net.*;
public class URLConnectionExample {
    public static void main(String[] args) throws Exception {
        URL url = new URL("https://www.example.com/");
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        BufferedReader in = new BufferedReader(new
        InputStreamReader(conn.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            System.out.println(inputLine);
        }
        in.close();
    }
}
```

issue



Tool Output

```
Note: D:\javaprgs\URLConnectionExample.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

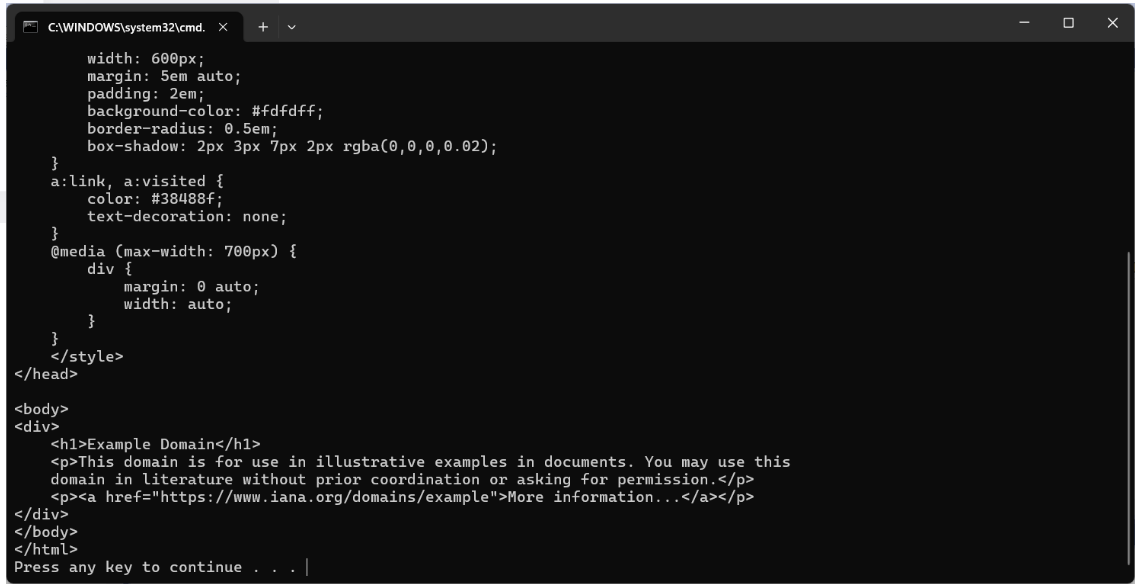


example.com

Example Domain

This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.

More information...



```
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
Press any key to continue . . .
```

Http status code

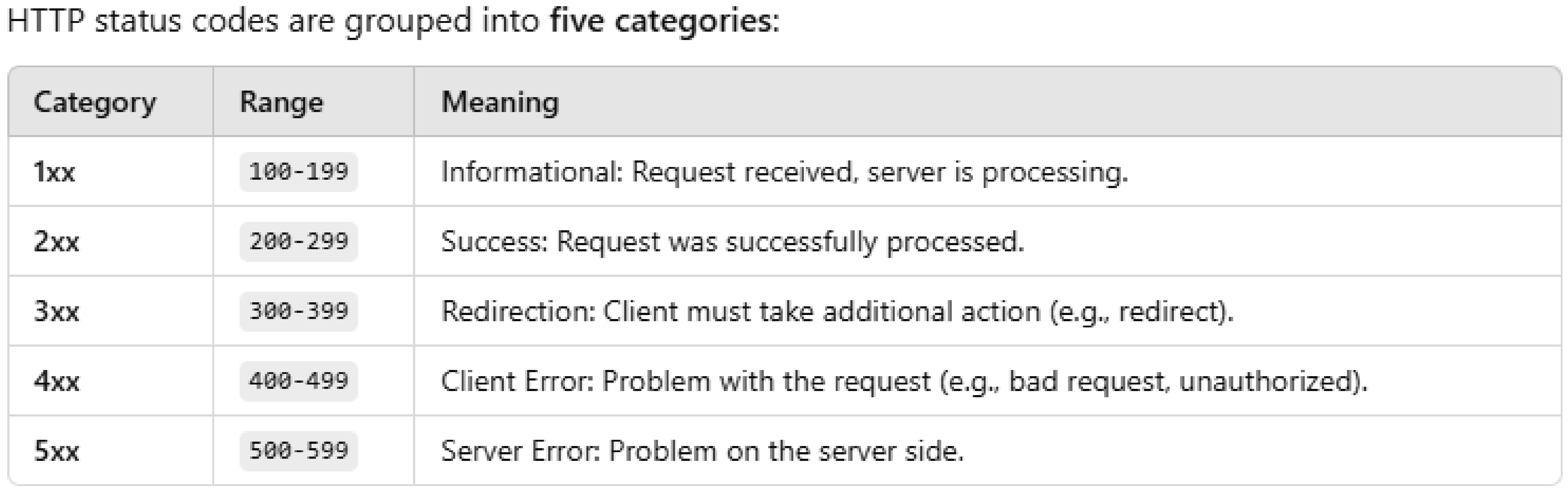
```
URL url = new URL("http://example.com");
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.setRequestMethod("GET");

int statusCode = connection.getResponseCode();  // Get HTTP Status Code
System.out.println("Status Code: " + statusCode);
```

Status Code

What is a Status Code in HTTP?

- An **HTTP status code** is a **three-digit number** returned by a server in response to a client request. It indicates whether the request was successful, failed, or requires further action.



HTTP status codes are grouped into **five categories:**

Category	Range	Meaning
1xx	`100-199`	Informational: Request received, server is processing.
2xx	`200-299`	Success: Request was successfully processed.
3xx	`300-399`	Redirection: Client must take additional action (e.g., redirect).
4xx	`400-499`	Client Error: Problem with the request (e.g., bad request, unauthorized).
5xx	`500-599`	Server Error: Problem on the server side.

Common HTTP Status Codes

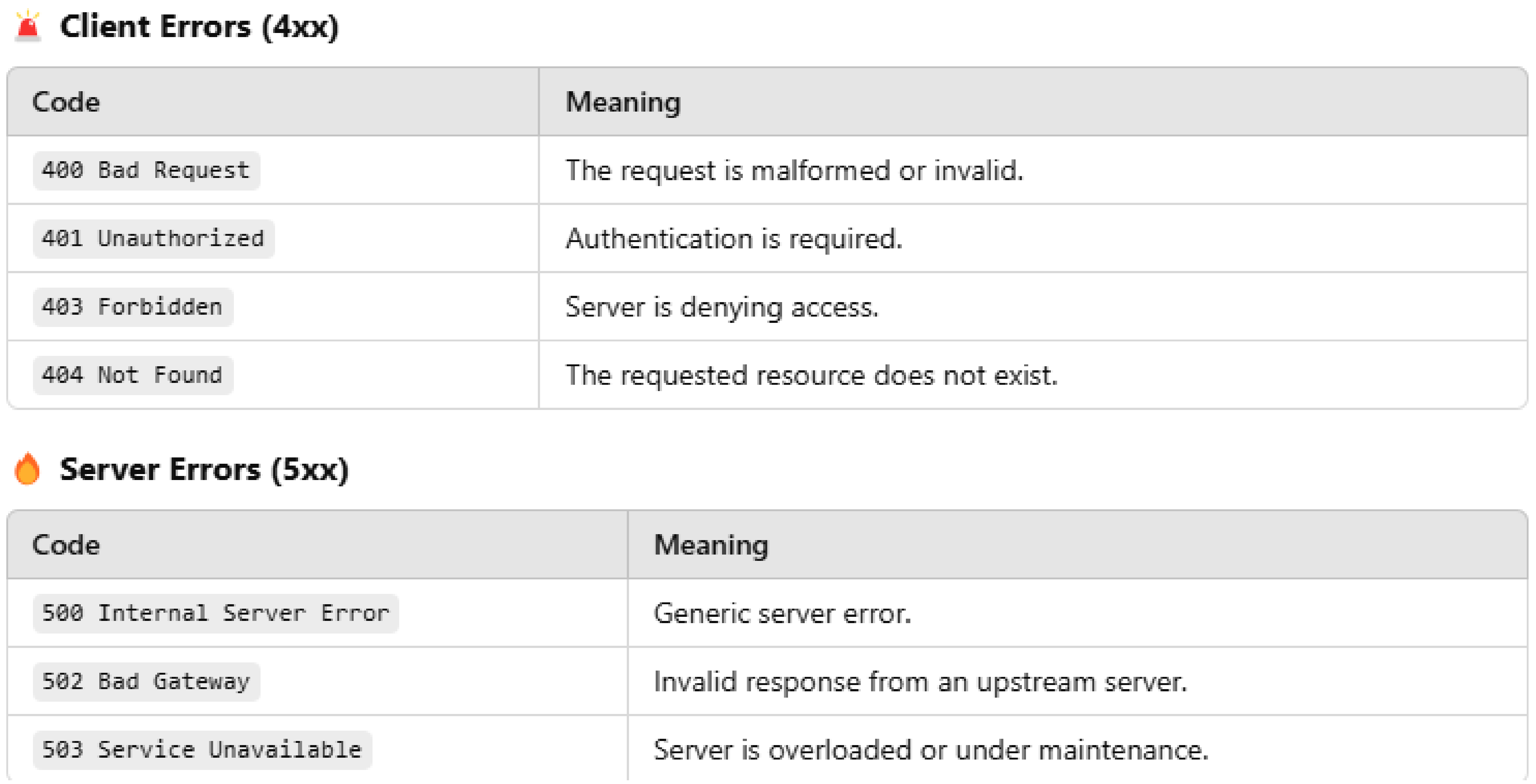
✅ Success (2xx)

Code	Meaning
`200 OK`	Request was successful, and response is returned.
`201 Created`	Request succeeded, and a new resource was created.
`204 No Content`	Request was successful, but there is no response body.

🔄 Redirection (3xx)

Code	Meaning
`301 Moved Permanently`	The requested URL has been moved permanently.
`302 Found`	Temporary redirection to another URL.
`304 Not Modified`	Resource hasn't changed (used for caching).

Common HTTP Status Codes…



🚨 Client Errors (4xx)

Code	Meaning
`400 Bad Request`	The request is malformed or invalid.
`401 Unauthorized`	Authentication is required.
`403 Forbidden`	Server is denying access.
`404 Not Found`	The requested resource does not exist.

🔥 Server Errors (5xx)

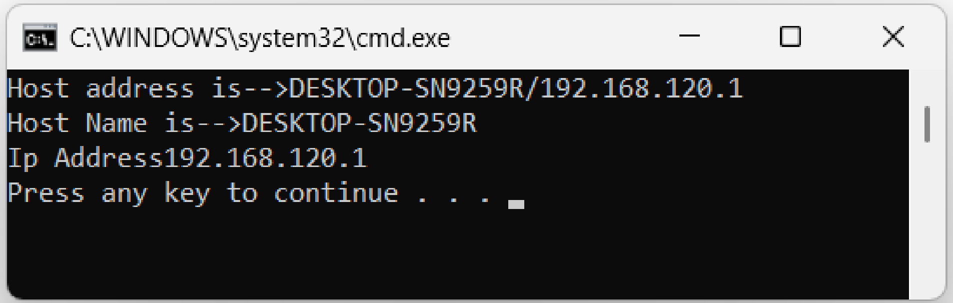
Code	Meaning
`500 Internal Server Error`	Generic server error.
`502 Bad Gateway`	Invalid response from an upstream server.
`503 Service Unavailable`	Server is overloaded or under maintenance.

Connecting to a Server (HTTP URL Connection)

```
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
public class ModernHttpClientExample {
    public static void main(String[] args) throws Exception {
        HttpClient client = HttpClient.newHttpClient();
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create("http://example.com"))
                .GET()
                .build();
        HttpResponse<String> response = client.send(request,
HttpResponse.BodyHandlers.ofString());
        System.out.println("Response Code: " + response.statusCode());
        System.out.println("Response Body: " + response.body());
    }
}
```

Getting host details

```java
import java.net.*;
 class NetGetIPAddress
 {
 public static void main(String args[])
 {InetAddress ia=null;
     try
     {
     ia=InetAddress.getLocalHost();
     }catch(UnknownHostException e){}
     System.out.println("Host address is-->"+ia);
     System.out.println("Host Name is-->"+ia.getHostName());
     String str=ia.toString();
     System.out.println("Ip Address"+str.substring(str.indexOf("/")+1));
 }
 }
```

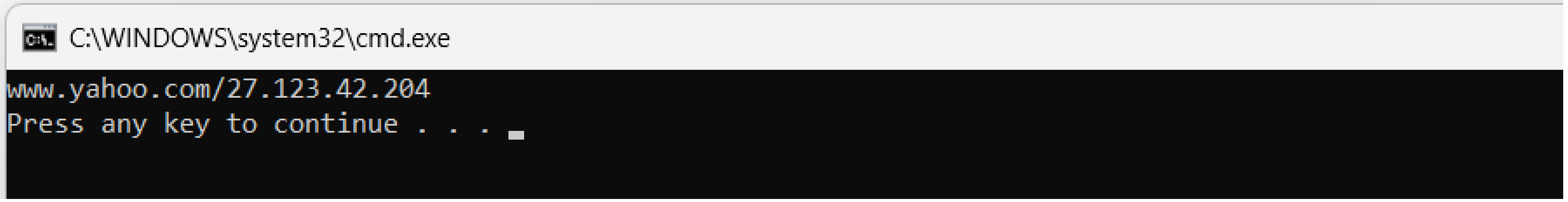


```
Host address is-->DESKTOP-SN9259R/192.168.120.1
Host Name is-->DESKTOP-SN9259R
Ip Address192.168.120.1
Press any key to continue . . .
```

Domain Name Services(DNS)

Domain Name System (DNS) is a database system that translates a computer's fully qualified domain name into an IP address.

```
import java.net.*;
 class NetDNS
 {
 public static void main(String args[]) throws Exception
 {try{
        InetAddress ia=InetAddress.getByName("www.yahoo.com");
        System.out.println(ia);
 }catch(UnknownHostException e)
{System.out.println("Unable to retreive");}
 }
 }
```



C:\WINDOWS\system32\cmd.exe

```
www.yahoo.com/27.123.42.204
Press any key to continue . . .
```

Port number

TCP/UDP both uses ports to map incoming data to a particular process running on a computer

Ports are represented by 16 bit numbers

Range ➔0-65535

Ports from 0-1023 are reserved for well known services

http➔80

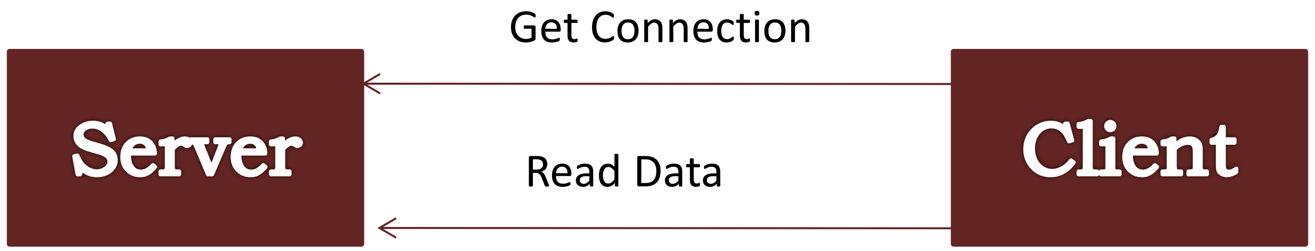
SMTP➔35

telnet➔23

Problem

- Create a chatting program using TCP/IP

How it works



What server will do in this program

1. Open Server socket for Client
2. Read User input from Keyboard
3. Pass the users message to network output stream

What client will do in this program

1. Establish connection to server
2. Read Data from the server

```
import java.net.*;         //client program
import java.io.*;
class NetClient
{ public static void main(String args[]) throws IOException
{  Socket s=null;
   BufferedReader br=null;
   try{
       s=new Socket(InetAddress.getLocalHost(),98);
       br=new BufferedReader(new InputStreamReader(s.getInputStream()));
       }catch(UnknownHostException u)
             {System.out.println("i dnot know the host");}
       String inputStr;
       while((inputStr=br.readLine())!=null)
              {
              System.out.println(inputStr);
              }
              br.close();
              s.close();
}
```

```
class NetServer                                    //server code
 { public static void main(String args[]) throws Exception
 {  ServerSocket s1=null;
    try{        s1=new ServerSocket(98);
         }catch(IOException e){}
    Socket c=null;
         try{    c=s1.accept();
                 }catch(Exception e1){}
    PrintWriter pw=new PrintWriter(c.getOutputStream(),true);
   // BufferedReader br=new BufferedReader(new InputStreamReader(c.getInputStream()));
    BufferedReader kin=new BufferedReader(new InputStreamReader(System.in));
   System.out.println("Pls enter the message now");
    String str;
                 while((str=kin.readLine())!=null)
                 { pw.println(str);     }
                 pw.close();
                 kin.close();
                 c.close();
                 s1.close();
 }
 }
```

Socket vs server Socket

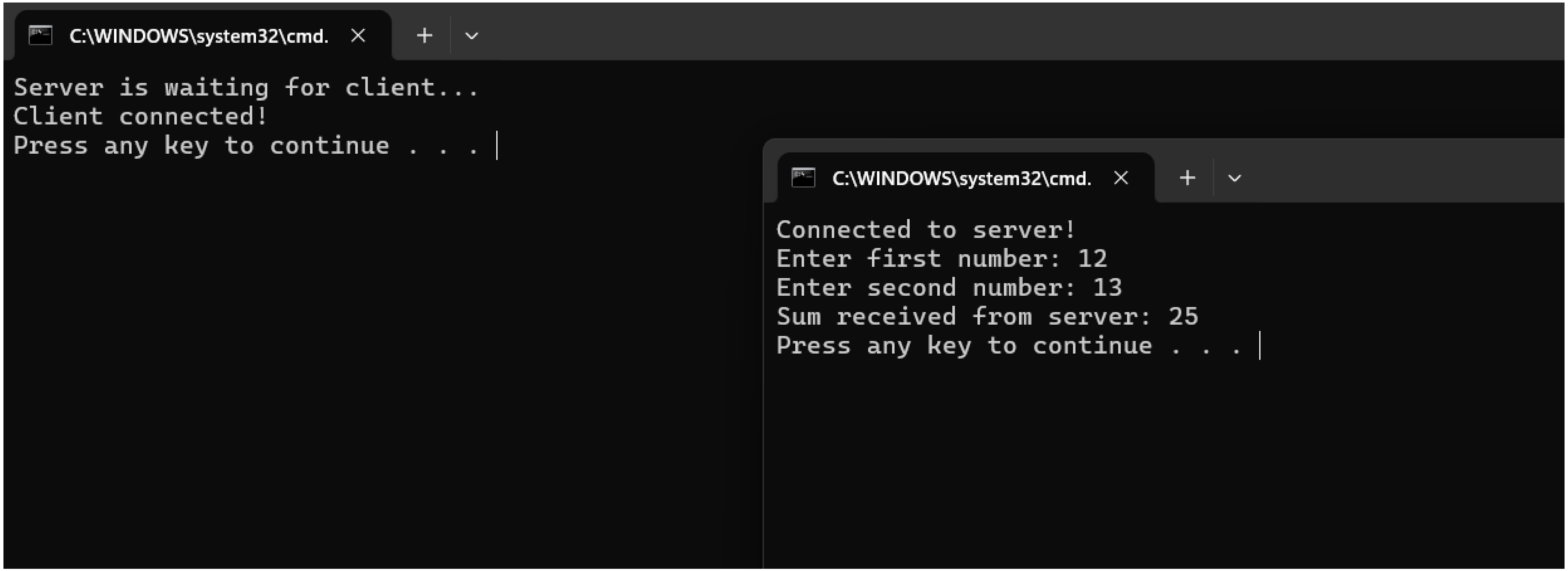
What is the difference between Socket and ServerSocket?

Problem

Write a client server based java program where server gets two numbers from client, adds them and returns to the client

```
import java.io.*;
import java.net.*;
public class AdditionServer {
    public static void main(String[] args) {
        try {
            // Create server socket on port 5000
            ServerSocket serverSocket = new ServerSocket(5000);
            System.out.println("Server is waiting for client...");
           Socket socket = serverSocket.accept();  // Accept client connection
            System.out.println("Client connected!");
            // Input and Output streams
            BufferedReader in = new BufferedReader(new
                                             InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            int num1 = Integer.parseInt(in.readLine()); // Read two numbers from client
            int num2 = Integer.parseInt(in.readLine());
            int sum = num1 + num2; // Calculate sum
            out.println(sum); // Send result back to client
            // Close resources
            in.close(); out.close(); socket.close(); serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
import java.io.*;
import java.net.*;
public class AdditionClient {
    public static void main(String[] args) {
        try {
            // Connect to server on localhost, port 5000
            Socket socket = new Socket("localhost", 5000);
            System.out.println("Connected to server!");
            // Input and Output streams
            BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new
                                                  InputStreamReader(socket.getInputStream()));
            System.out.print("Enter first number: "); // Get two numbers from user
            int num1 = Integer.parseInt(userInput.readLine());
            System.out.print("Enter second number: ");
            int num2 = Integer.parseInt(userInput.readLine());
            // Send numbers to server
            out.println(num1);          out.println(num2);
            int sum = Integer.parseInt(in.readLine()); // Receive result from server
            System.out.println("Sum received from server: " + sum);
            // Close resources
            userInput.close();          out.close();     in.close();      socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```


C:\WINDOWS\system32\cmd.

```
Server is waiting for client...
Client connected!
Press any key to continue . . .
```

C:\WINDOWS\system32\cmd.

```
Connected to server!
Enter first number: 12
Enter second number: 13
Sum received from server: 25
Press any key to continue . . .
```

Can this server accept multiple clients?

- No, the current server **only handles one client at a time** because it does not support **multithreading**.
- To make it handle **multiple clients simultaneously**, we need to create a **multithreaded server** where each client connection runs in a separate thread.

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

public class AdditionMultiClientServer {
    public static void main(String[] args) {
        try {
            // Create server socket on port 5000
            ServerSocket serverSocket = new ServerSocket(5000);
            System.out.println("Server is running and waiting for
clients...");

            // Continuously accept multiple clients
            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Client connected!");

                // Create a new thread for each client
                ClientHandler clientHandler = new
                 ClientHandler(socket);
                new Thread(clientHandler).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    } }
```

```
// Handles each client in a separate thread
class ClientHandler implements Runnable {
    private Socket socket;
    public ClientHandler(Socket socket) {
        this.socket = socket;
    }
    @Override
    public void run() {
        try {
BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
int num1 = Integer.parseInt(in.readLine());
            int num2 = Integer.parseInt(in.readLine());
int sum = num1 + num2;
out.println(sum);
in.close();          out.close();       socket.close();
            System.out.println("Client disconnected.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

UDP, Datagram packet

- Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet. Multiple packets sent from one machine to another might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

- **`DatagramPacket`**`(byte[] buf, int length)`

  Constructs a DatagramPacket for receiving packets of length length.

- **`DatagramPacket`**`(byte[] buf, int length, InetAddress address, int port)`

  Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

DataGram Socket

- **DatagramSocket** class represents a socket for sending and receiving datagram packets.
- A **datagram socket** is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.
- **`DatagramSocket`**`(int port)`

  Constructs a datagram socket and binds it to the specified port on the local host machine.
- **`DatagramSocket(int port, InetAddress laddr)`**

  Creates a datagram socket, bound to the specified local address.

```
class UDPServer
{public static DatagramSocket ds;
public static byte buffer[]=new byte[1024];
public static void myserver() throws Exception
{int pos=0;
   while(true)
   {int c=System.in.read();
   switch(c)
   {     case -1: System.out.println("Server quits");return;
         case '\r': break;
         case '\n': ds.send(new DatagramPacket(buffer,pos,InetAddress.getLocalHost(),777));
                 pos=0;
                 break;
         default:
         buffer[pos++]=(byte)c;
         }}}
   public static void main(String args[]) throws Exception
         {
                 System.out.println("Server Ready \n Type here");
                 ds=new DatagramSocket(888);
                 myserver();
                 }
}
```

```
import java.net.*;
class UDPClient
{
public static DatagramSocket ds;
public static byte buffer[]=new byte[1024];
public static void myclient() throws Exception
{
   while(true)
   {DatagramPacket p=new DatagramPacket(buffer,buffer.length);
   ds.receive(p);
   System.out.println(new String(p.getData(),0,p.getLength()));
        }
}
public static void main(String args[]) throws Exception
{
   System.out.println("Client\n press ctrl+c to quit");
   ds=new DatagramSocket(777);
   myclient();
   }
}
```

Thank You