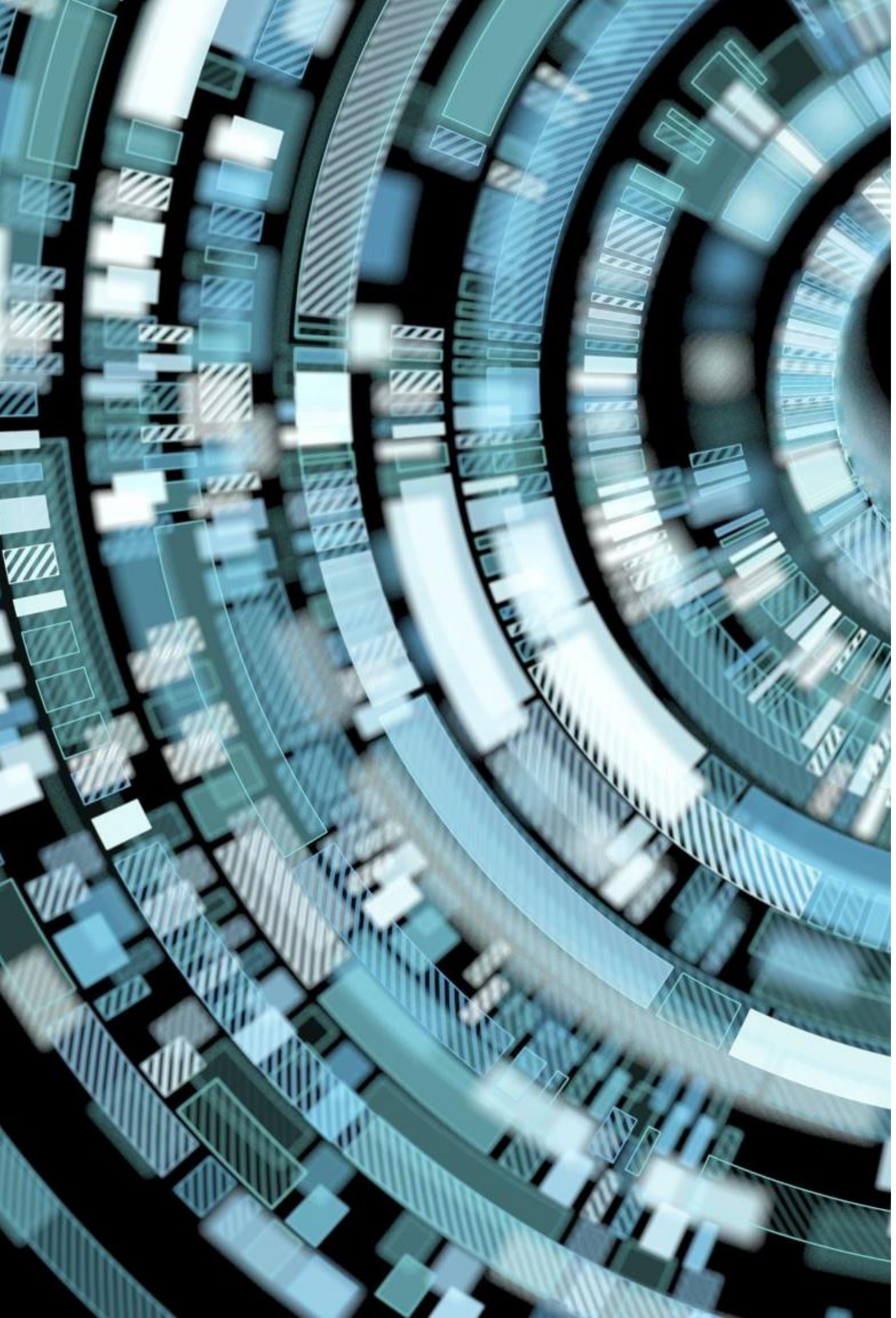
Computer System Design & Application
计算机系统设计与应用A

陶伊达 （TAO Yida）
taoyd@sustech.edu.cn



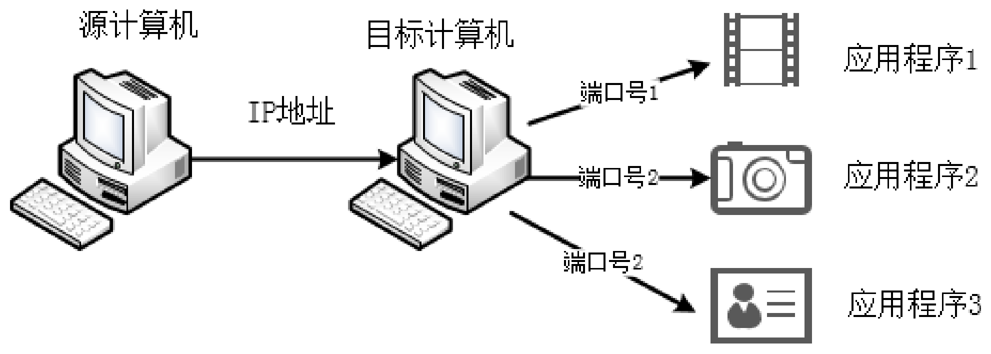
Lecture 8

- Network Basics
- Network Protocols
- Socket Programming
- Getting Web Data

Networking

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

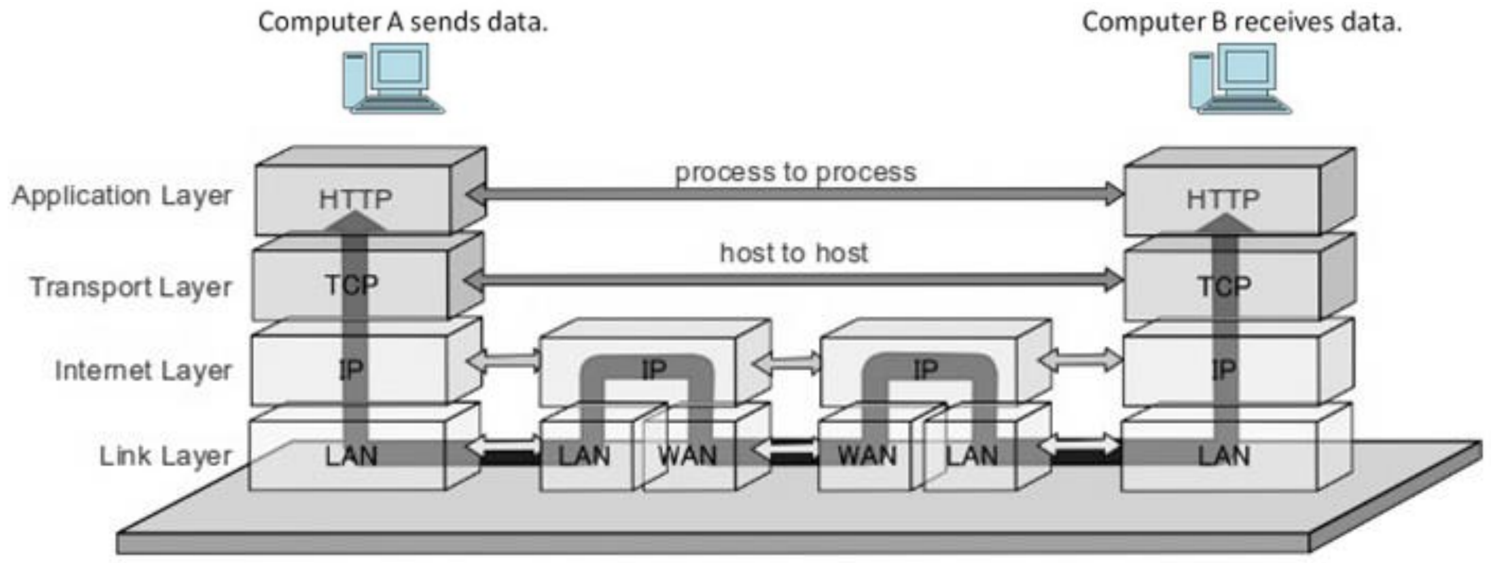
Networking Terminology



- **IP address**: a unique address that distinguishes a device on the internet or a local network
- **Domain name**: a human-friendly version of an IP address that you enter in browser (translated by DNS)
- **Port number**: a number used to identify different applications/processes uniquely

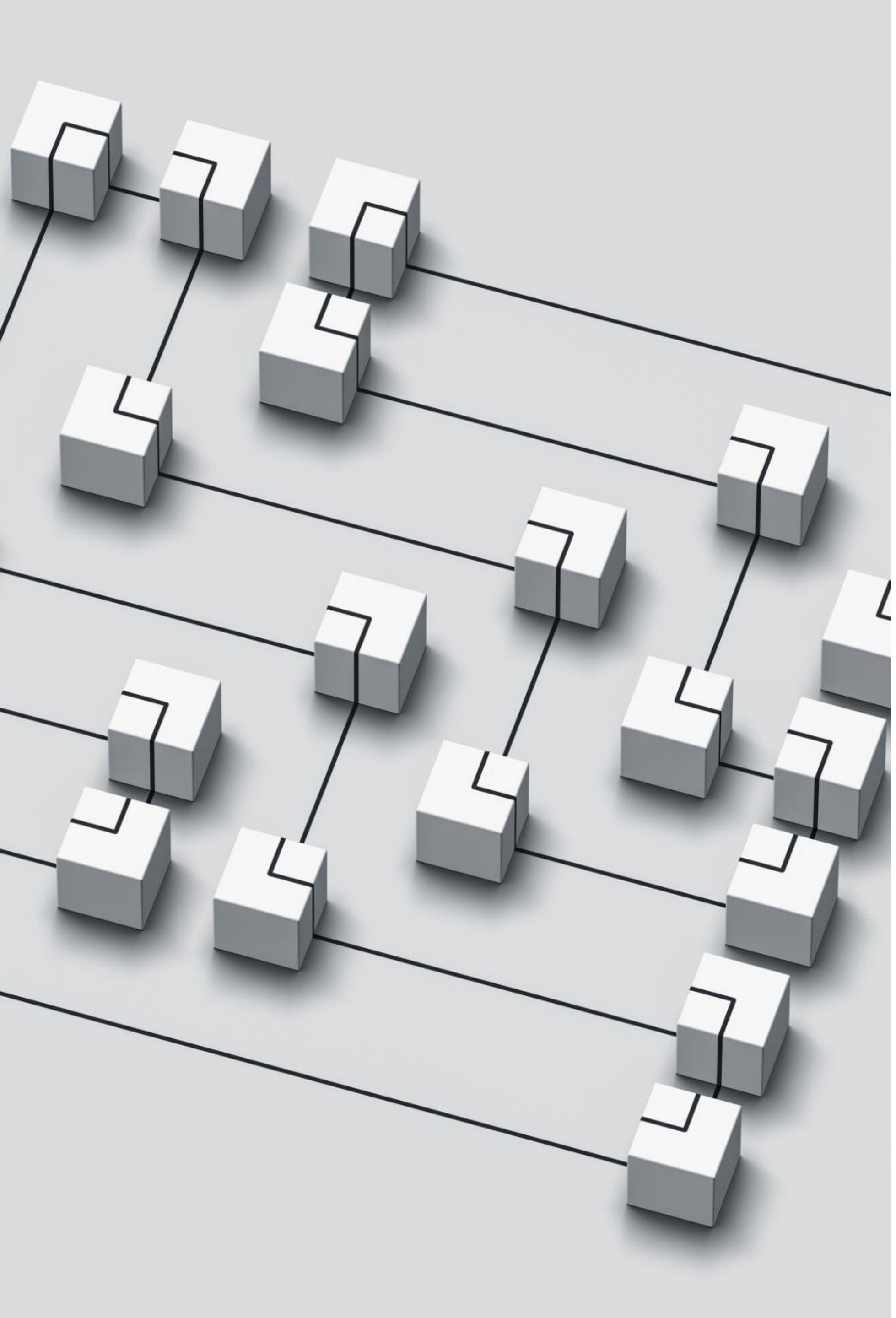
Network Architecture

- Network architecture refers to a network's structural and logical layout. It describes how the network devices are connected and the rules that govern data transfer between them



https://www.elprocus.com/tcp-ip-protocol-architecture-and-its-layers/

How do devices communicate with each other?



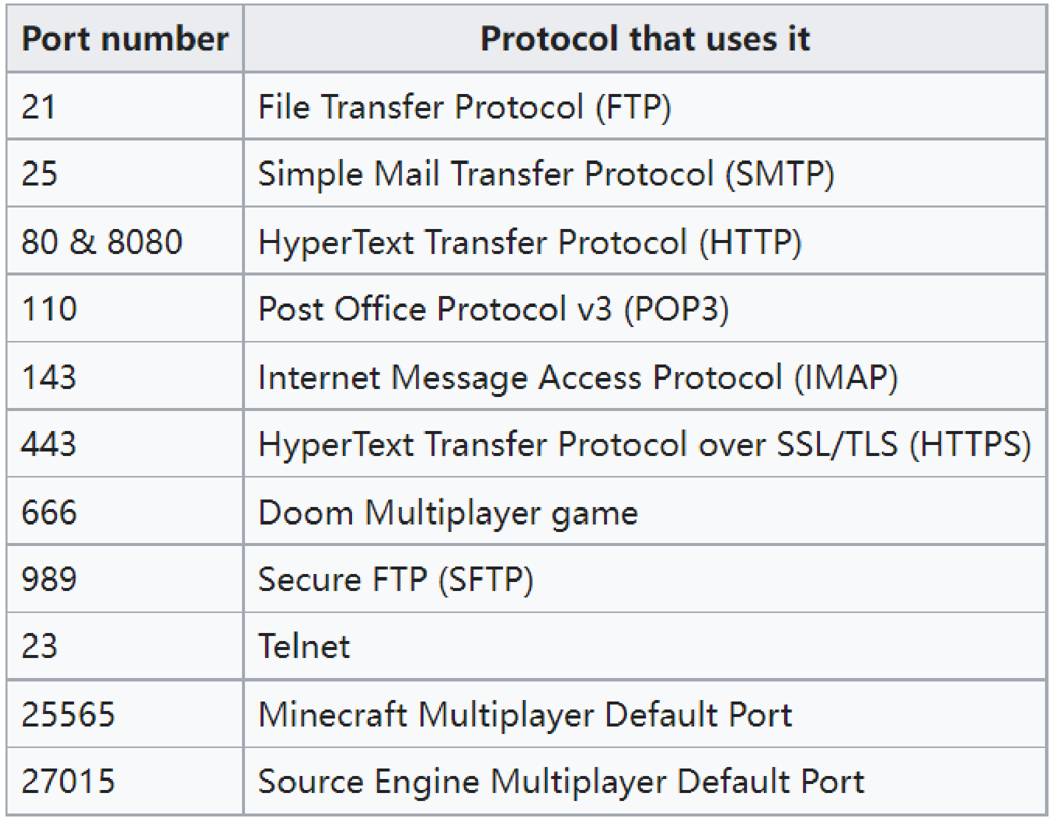
Network Protocols

- A network protocol (网络协议) is a set of established rules that dictate how to format, transmit and receive data so that computer network devices can communicate, regardless of the differences in their underlying infrastructures, designs or standards.
- To successfully send and receive information, devices on both sides of a communication exchange must accept and follow protocol conventions
- Without computing protocols, computers and other devices would not know how to engage with each other.

Application Layer Protocols

- Each Internet application has a different application protocol, which describes how the data for that particular application are transmitted.
- A port number helps a computer decide which application should receive an incoming piece of data

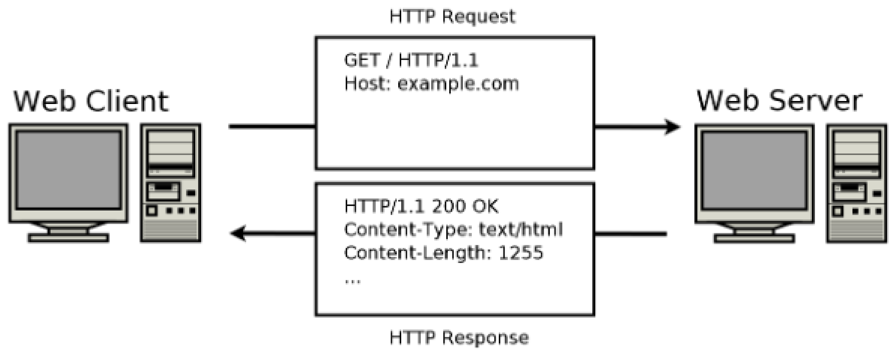
Well-known port numbers are reserved and we can no longer use them for other purposes



Port number	Protocol that uses it
21	File Transfer Protocol (FTP)
25	Simple Mail Transfer Protocol (SMTP)
80 & 8080	HyperText Transfer Protocol (HTTP)
110	Post Office Protocol v3 (POP3)
143	Internet Message Access Protocol (IMAP)
443	HyperText Transfer Protocol over SSL/TLS (HTTPS)
666	Doom Multiplayer game
989	Secure FTP (SFTP)
23	Telnet
25565	Minecraft Multiplayer Default Port
27015	Source Engine Multiplayer Default Port

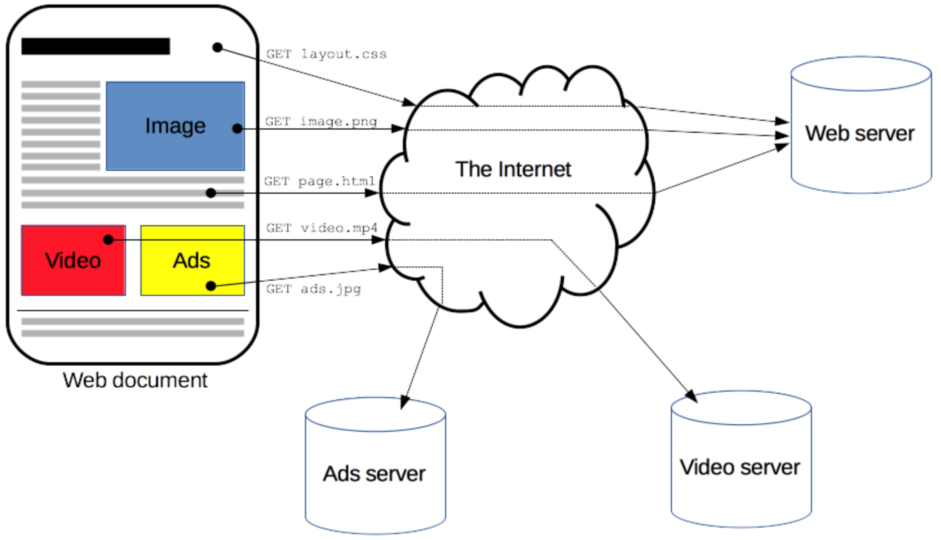
HTTP (Hypertext Transfer Protocol)

- HTTP is a protocol for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web
- It is a client-server protocol, which means requests are initiated by the client, usually the web browser.
- Web server responds with an HTTP response



```
GET / HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1255
...
```



https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

HTTP Request Commands

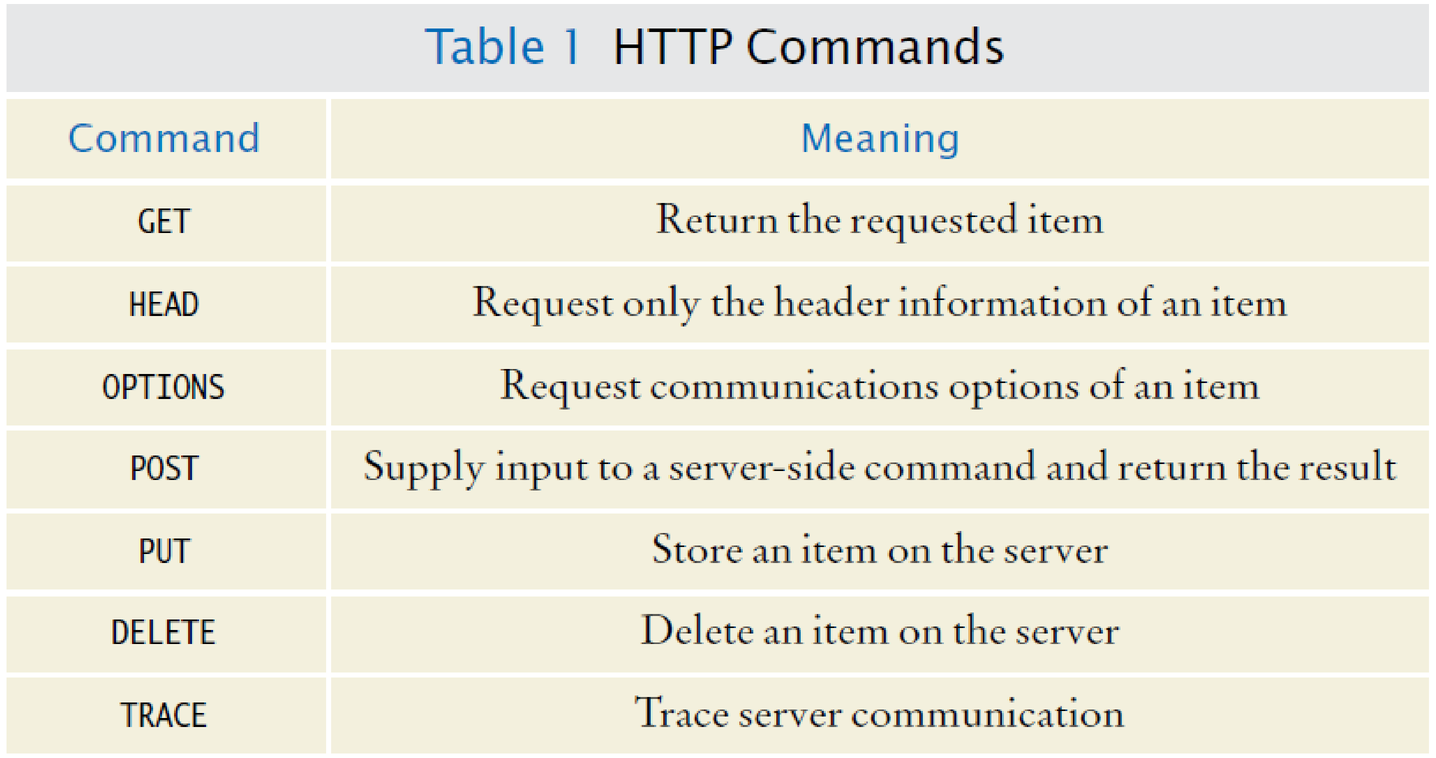
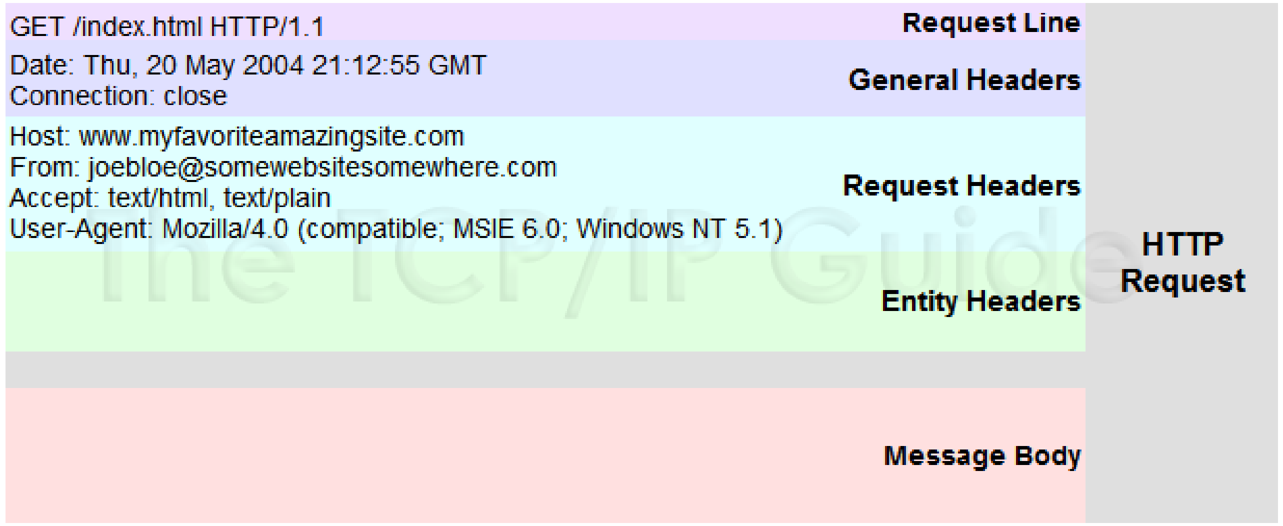
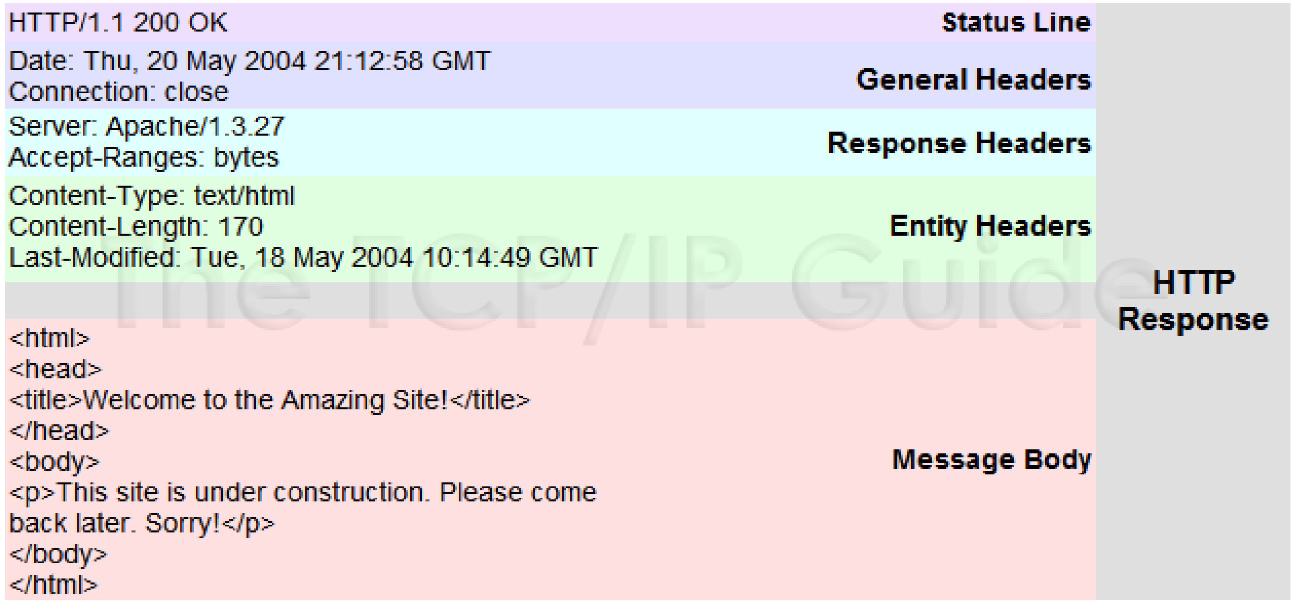


Table 1 HTTP Commands

Command	Meaning
`GET`	Return the requested item
`HEAD`	Request only the header information of an item
`OPTIONS`	Request communications options of an item
`POST`	Supply input to a server-side command and return the result
`PUT`	Store an item on the server
`DELETE`	Delete an item on the server
`TRACE`	Trace server communication

http://www.tcpipguide.com/free/t_HTTPResponseMessageFormat.htm

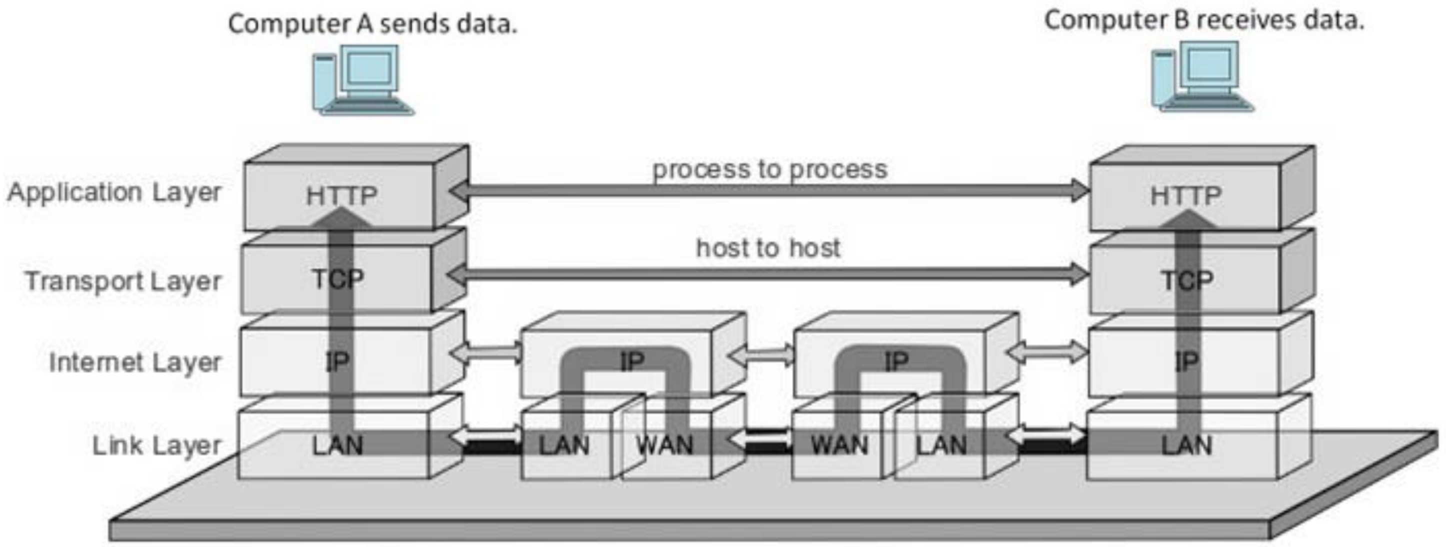

```
GET /index.html HTTP/1.1
Date: Thu, 20 May 2004 21:12:55 GMT
Connection: close
Host: www.myfavoriteamazingsite.com
From: joebloe@somewebsitesomewhere.com
Accept: text/html, text/plain
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
```


```
HTTP/1.1 200 OK
Date: Thu, 20 May 2004 21:12:58 GMT
Connection: close
Server: Apache/1.3.27
Accept-Ranges: bytes
Content-Type: text/html
Content-Length: 170
Last-Modified: Tue, 18 May 2004 10:14:49 GMT

<html>
<head>
<title>Welcome to the Amazing Site!</title>
</head>
<body>
<p>This site is under construction. Please come
back later. Sorry!</p>
</body>
</html>
```

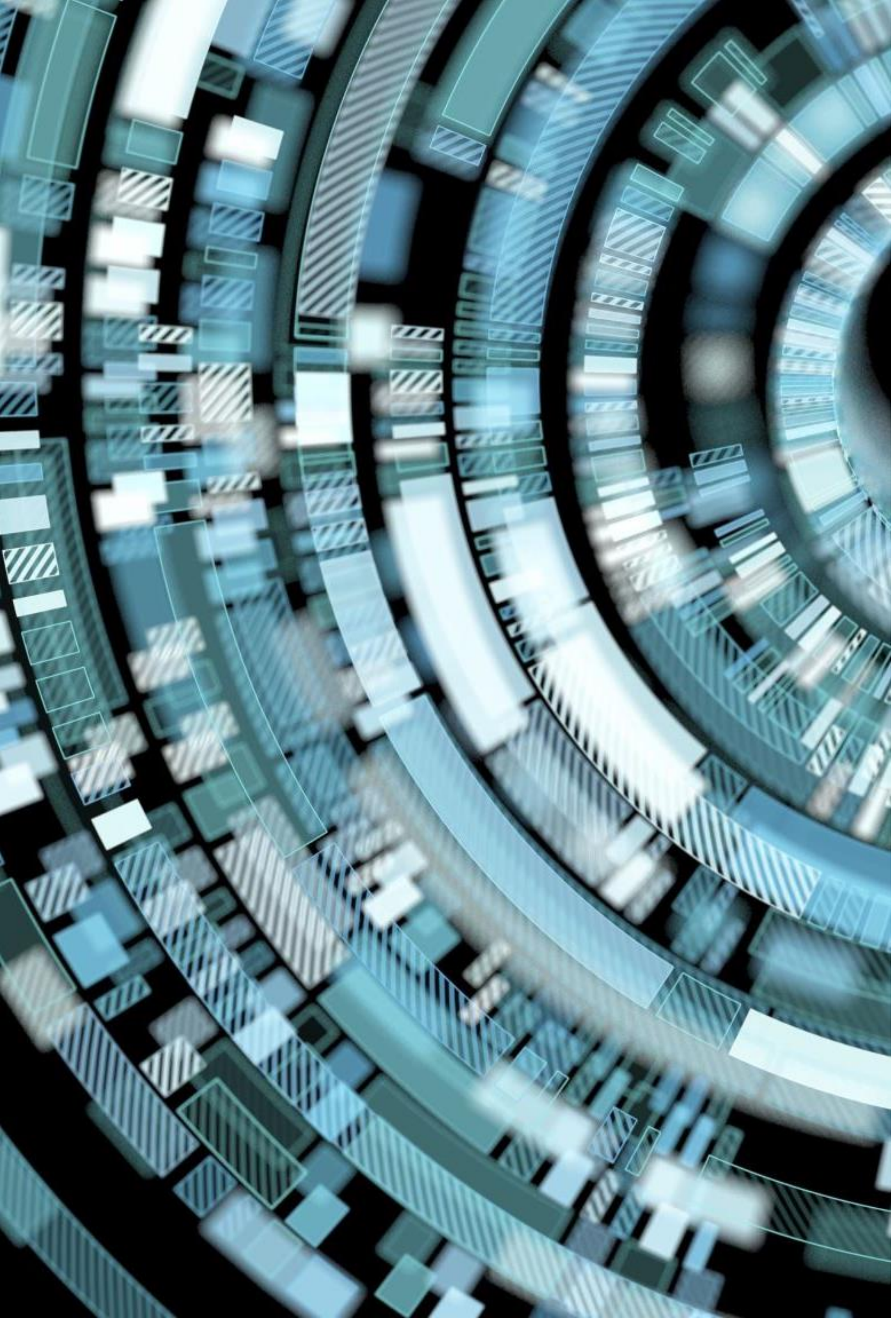
HTTP Request/Response Message Format

Transport Layer Protocols

- TCP (Transmission Control Protocol)
  - TCP provides a reliable, point-to-point communication channel for clients and servers to communicate over the Internet
  - TCP is the protocol used most on top of IP, we often referred to as TCP/IP
- UDP (User Datagram Protocol)
  - contains a minimum amount of communication mechanisms (no acknowledgement, unreliable)



https://www.elprocus.com/tcp-ip-protocol-architecture-and-its-layers/

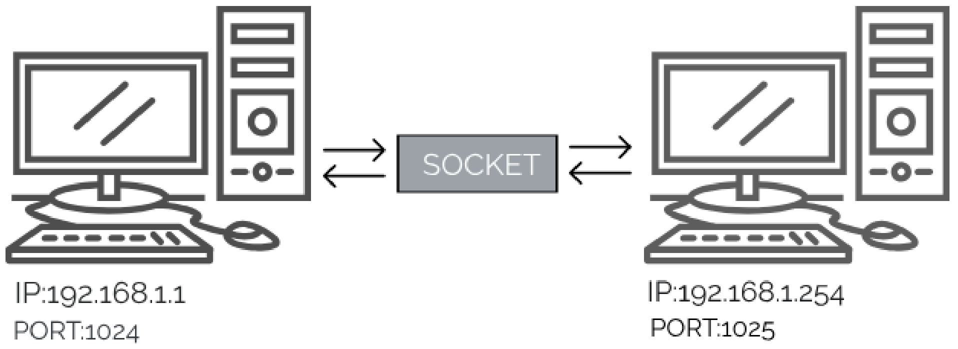


Lecture 8

- Network Basics
- Network Protocols
- Socket Programming
- Getting Web Data

Socket

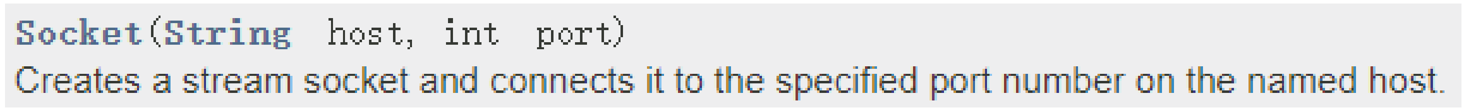
- To communicate, a client program and a server program establish a connection to one another
- Each program binds a socket to its end of the connection
- A socket is one endpoint of a two-way communication link between two programs running on the network.
  - Endpoint: IP address + Port number
- To communicate, the client and the server each reads from and writes to the socket bound to the connection.



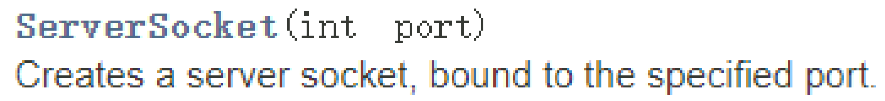
https://examradar.com/java-networking-network-basics-socket-overview/

Socket

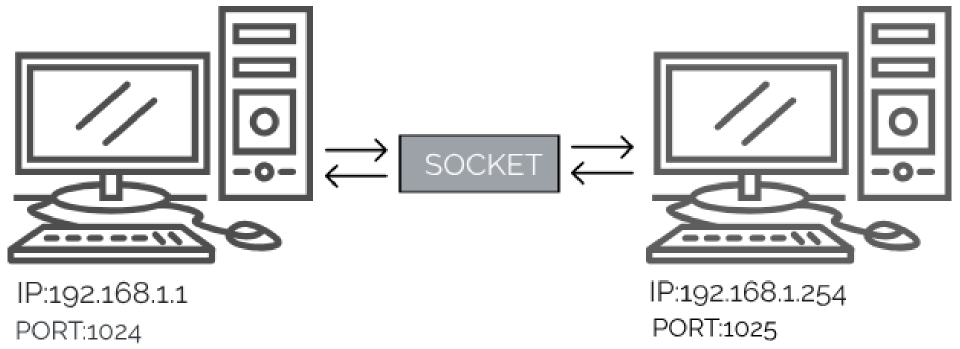
- The `java.net` package provides a powerful and flexible infrastructure for networking, providing various classes and interfaces that execute the low-level communication features



```
Socket(String  host, int  port)
```
Creates a stream socket and connects it to the specified port number on the named host.



```
ServerSocket(int  port)
```
Creates a server socket, bound to the specified port.



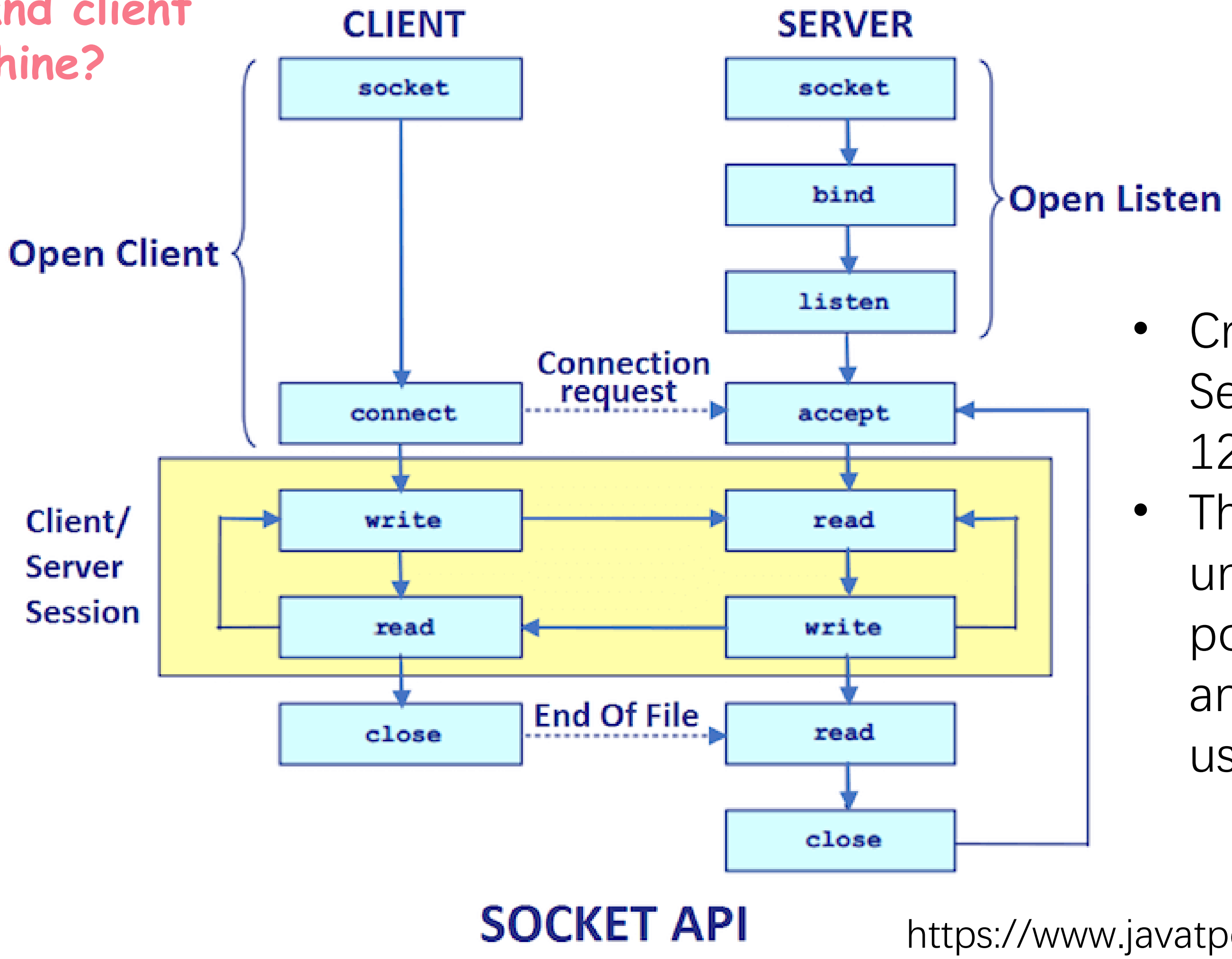
https://examradar.com/java-networking-network-basics-socket-overview/

```
Socket s = new Socket("www.serverip.com", 1234);
```

```
ServerSocket ss = new ServerSocket(1234);
Socket s = ss.accept();
```

What if the server and client run on the same machine?

- Create an instance of Socket by passing the IP or hostname of the server and a port number
- If the connection fails, an Exception is thrown
- Otherwise, establish the connection and use Socket s to read and write.

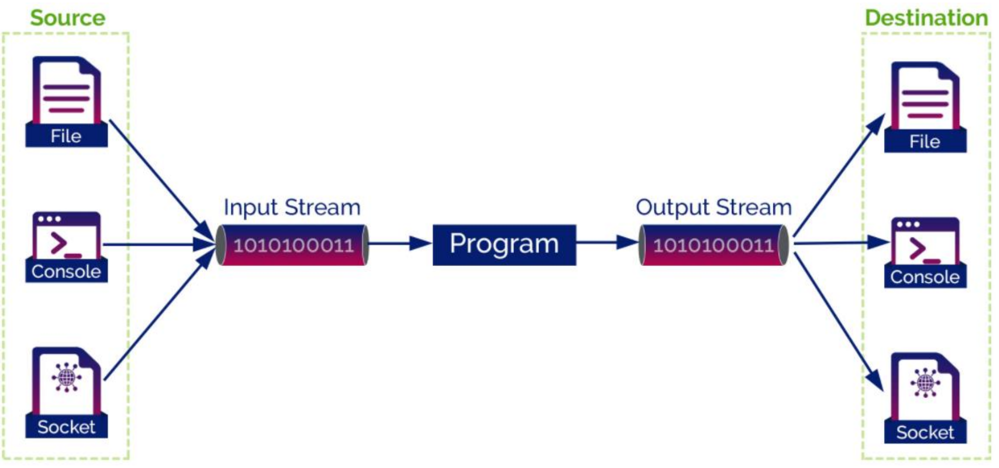


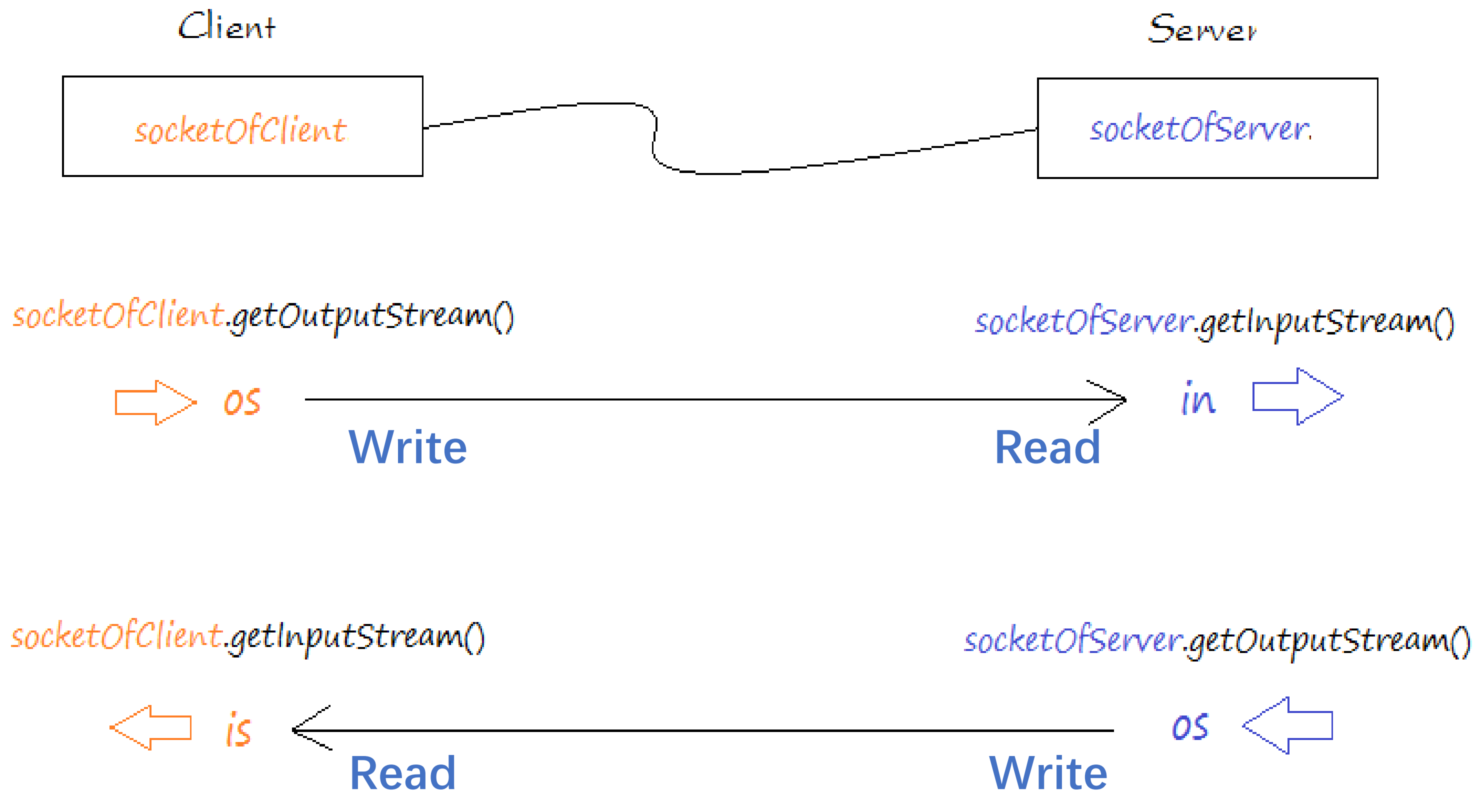
- Create an instance of ServerSocket by binding to 1234 port number
- The accept() method waits until a client connects to port 1234, and if so, return an instance of Socket that is used for reading and writing.

https://www.javatpoint.com/socket-programming

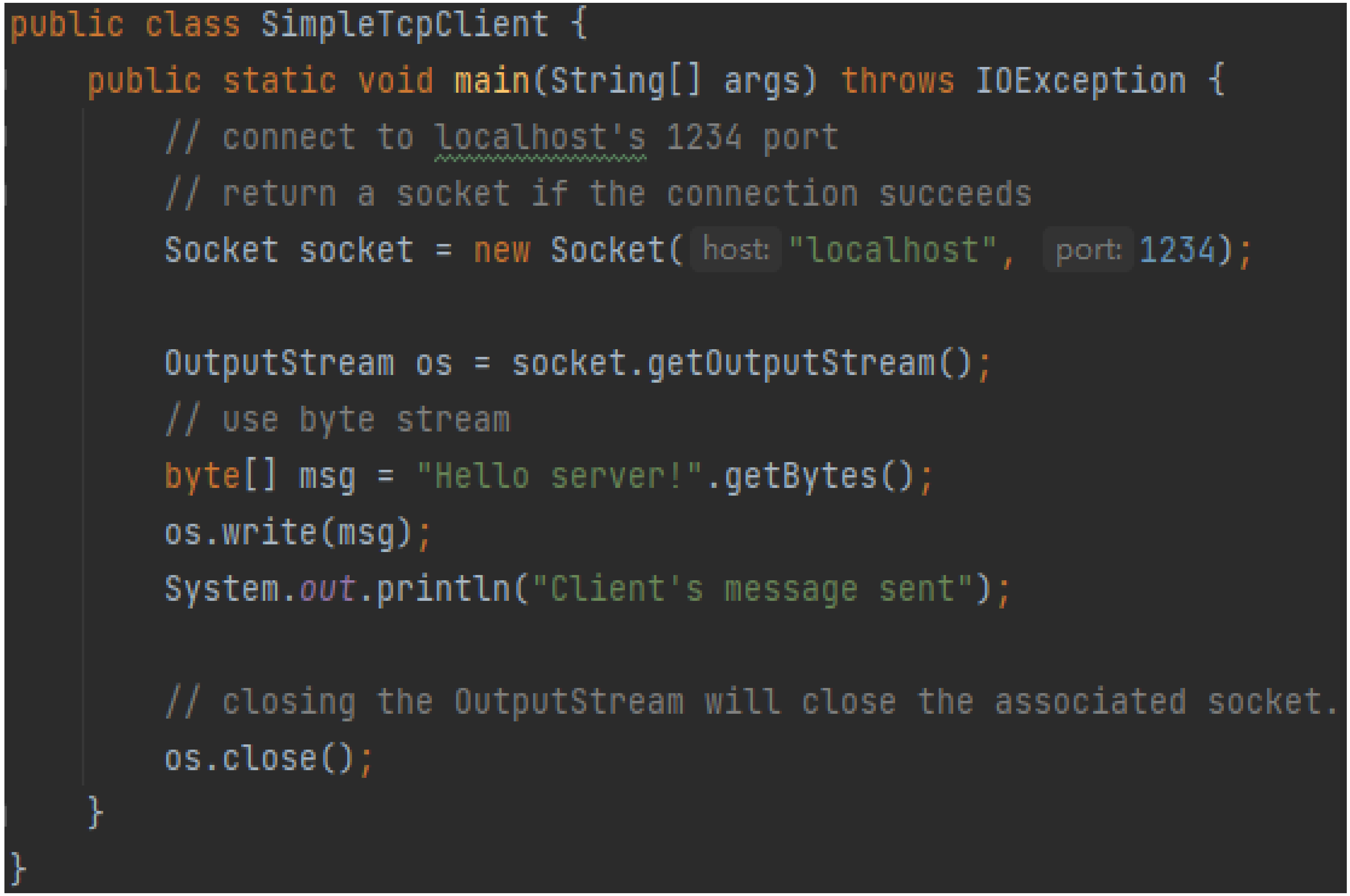
Reading from and Writing to a Socket

- After establishing the connection, we can use socket.getInputStream() and socket.getOutputStream() for both the client and the server



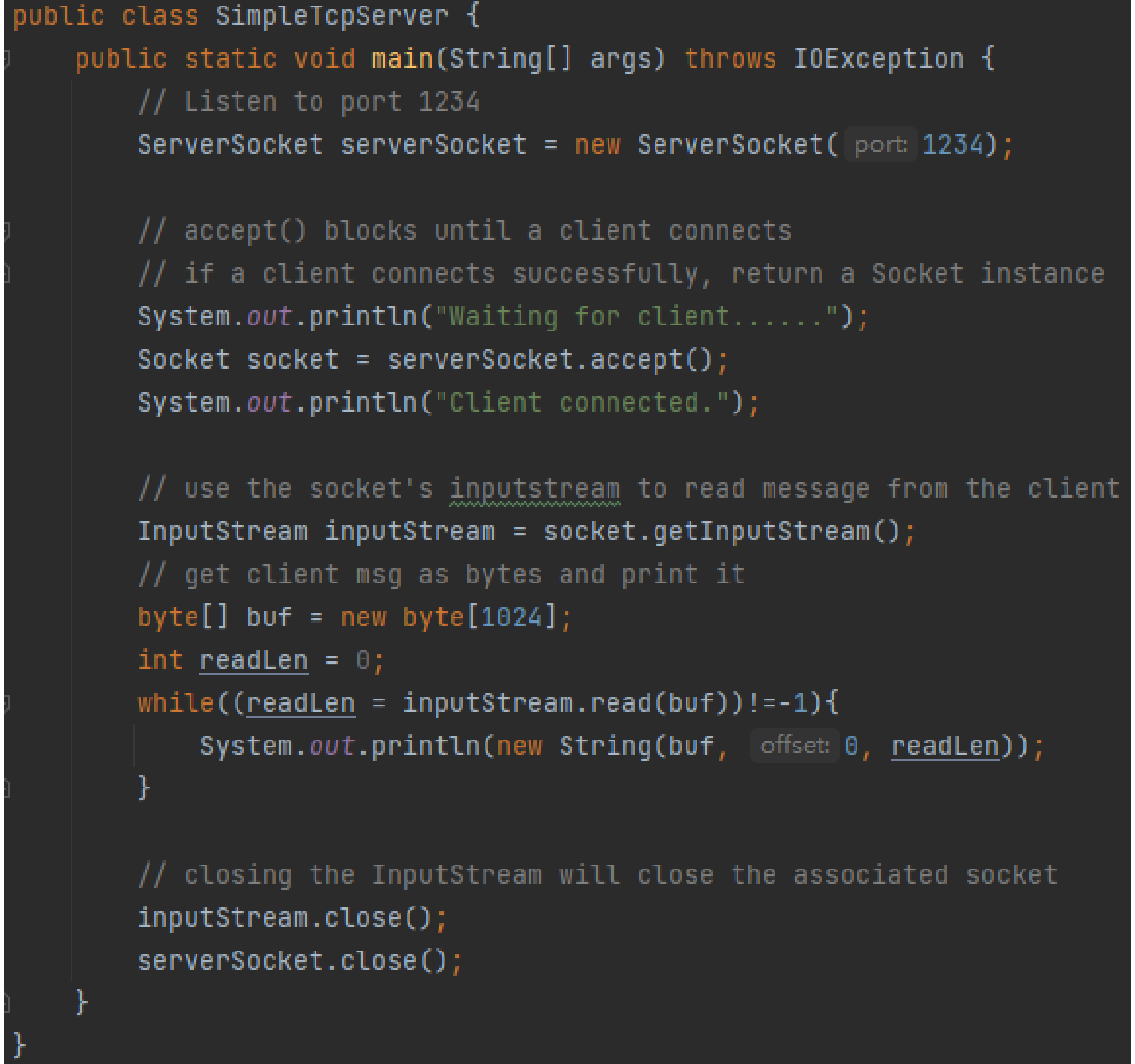

Client
Server
socketOfClient
socketOfServer.
socketOfClient.getOutputStream()
socketOfServer.getInputStream()
os
Write
Read
in
socketOfClient.getInputStream()
socketOfServer.getOutputStream()
is
Read
Write
os

A Toy Example: Client



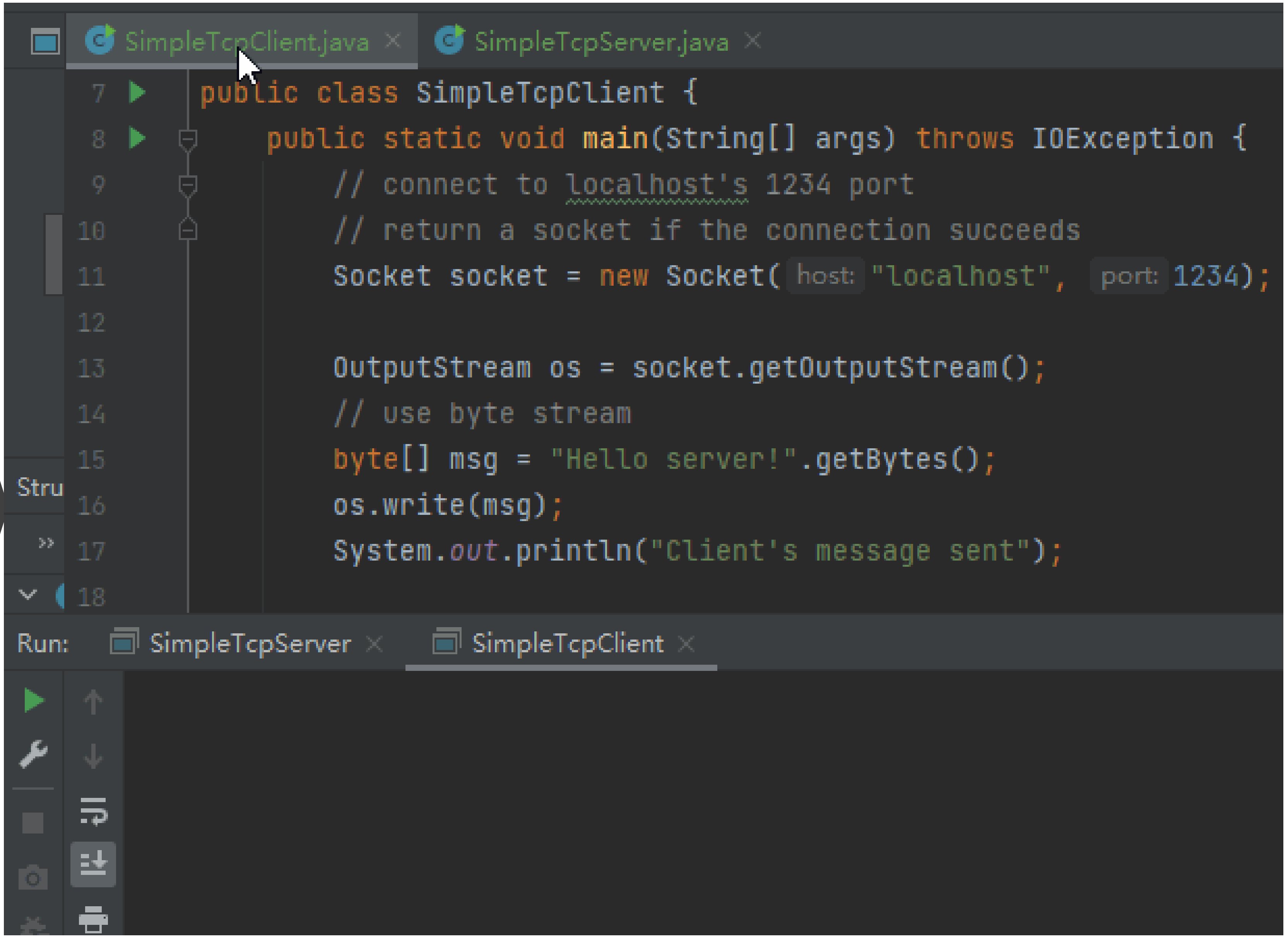
```
public class SimpleTcpClient {
    public static void main(String[] args) throws IOException {
        // connect to localhost's 1234 port
        // return a socket if the connection succeeds
        Socket socket = new Socket( host: "localhost",  port: 1234);

        OutputStream os = socket.getOutputStream();
        // use byte stream
        byte[] msg = "Hello server!".getBytes();
        os.write(msg);
        System.out.println("Client's message sent");

        // closing the OutputStream will close the associated socket.
        os.close();
    }
}
```

A Toy Example: Server



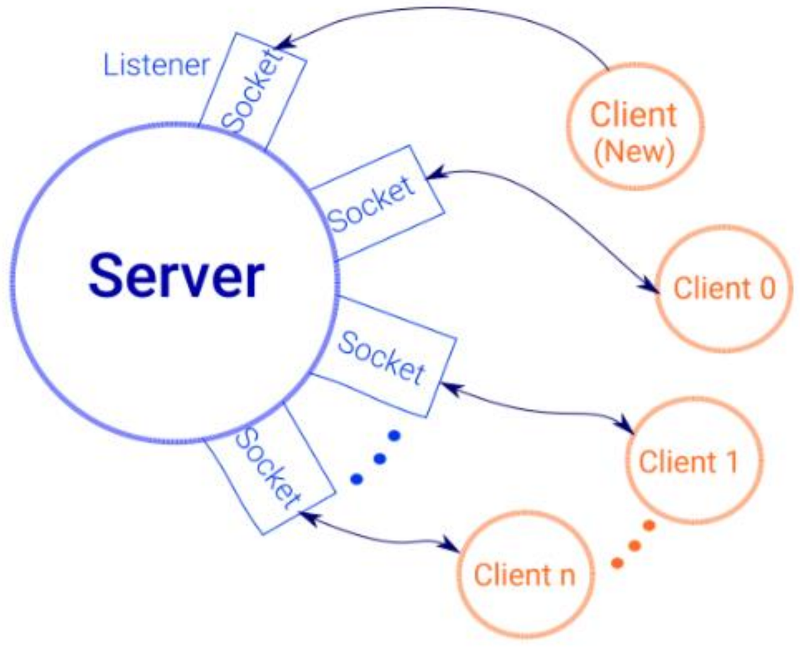
```java
public class SimpleTcpServer {
    public static void main(String[] args) throws IOException {
        // Listen to port 1234
        ServerSocket serverSocket = new ServerSocket( port: 1234);

        // accept() blocks until a client connects
        // if a client connects successfully, return a Socket instance
        System.out.println("Waiting for client......");
        Socket socket = serverSocket.accept();
        System.out.println("Client connected.");

        // use the socket's inputstream to read message from the client
        InputStream inputStream = socket.getInputStream();
        // get client msg as bytes and print it
        byte[] buf = new byte[1024];
        int readLen = 0;
        while((readLen = inputStream.read(buf))!=-1){
            System.out.println(new String(buf,  offset: 0, readLen));
        }

        // closing the InputStream will close the associated socket
        inputStream.close();
        serverSocket.close();
    }
}
```

A Toy Example



```java
public class SimpleTcpClient {
    public static void main(String[] args) throws IOException {
        // connect to localhost's 1234 port
        // return a socket if the connection succeeds
        Socket socket = new Socket( host: "localhost",  port: 1234);

        OutputStream os = socket.getOutputStream();
        // use byte stream
        byte[] msg = "Hello server!".getBytes();
        os.write(msg);
        System.out.println("Client's message sent");
```

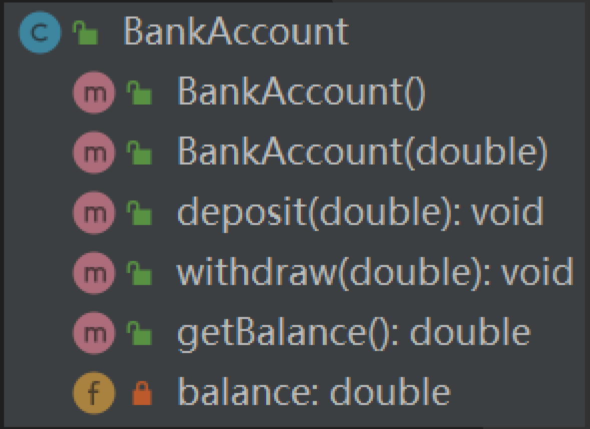
Why "Toy" Examples?



- The toy server reads only 1 message then exits; In practice, server keeps running
- The toy client/server handles byte directly, which is cumbersome
- In practice, servers need to support multiple clients at the same time

More practical: We could use threads on server side: whenever a client request comes, a separate thread is assigned for handling each request

Case Study: Banking Service



BankAccount
- BankAccount()
- BankAccount(double)
- deposit(double): void
- withdraw(double): void
- getBalance(): double
- balance: double

- A bank account has a balance that can be changed by deposits and withdrawals.

```java
public synchronized void deposit (double amount) {
    balance = balance + amount;
    notifyAll();
}
```

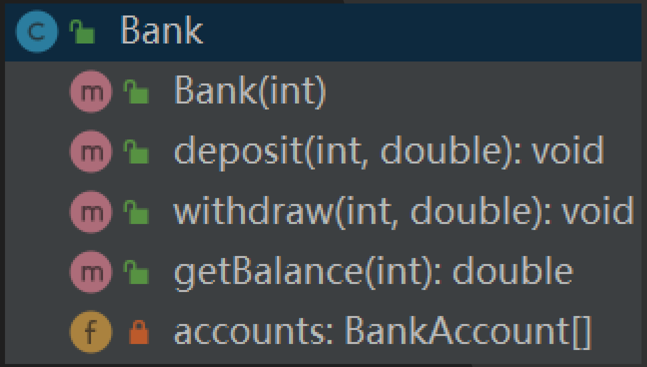
Wakes up all threads that are waiting on this object's monitor.

```java
public synchronized void withdraw (double amount) {
    try {
        while (balance < amount) wait();
        balance = balance - amount;
    } catch (InterruptedException e) {}
}
```

Causes the current thread to wait until another thread invokes the **notify()** method or the **notifyAll()** method for this object.

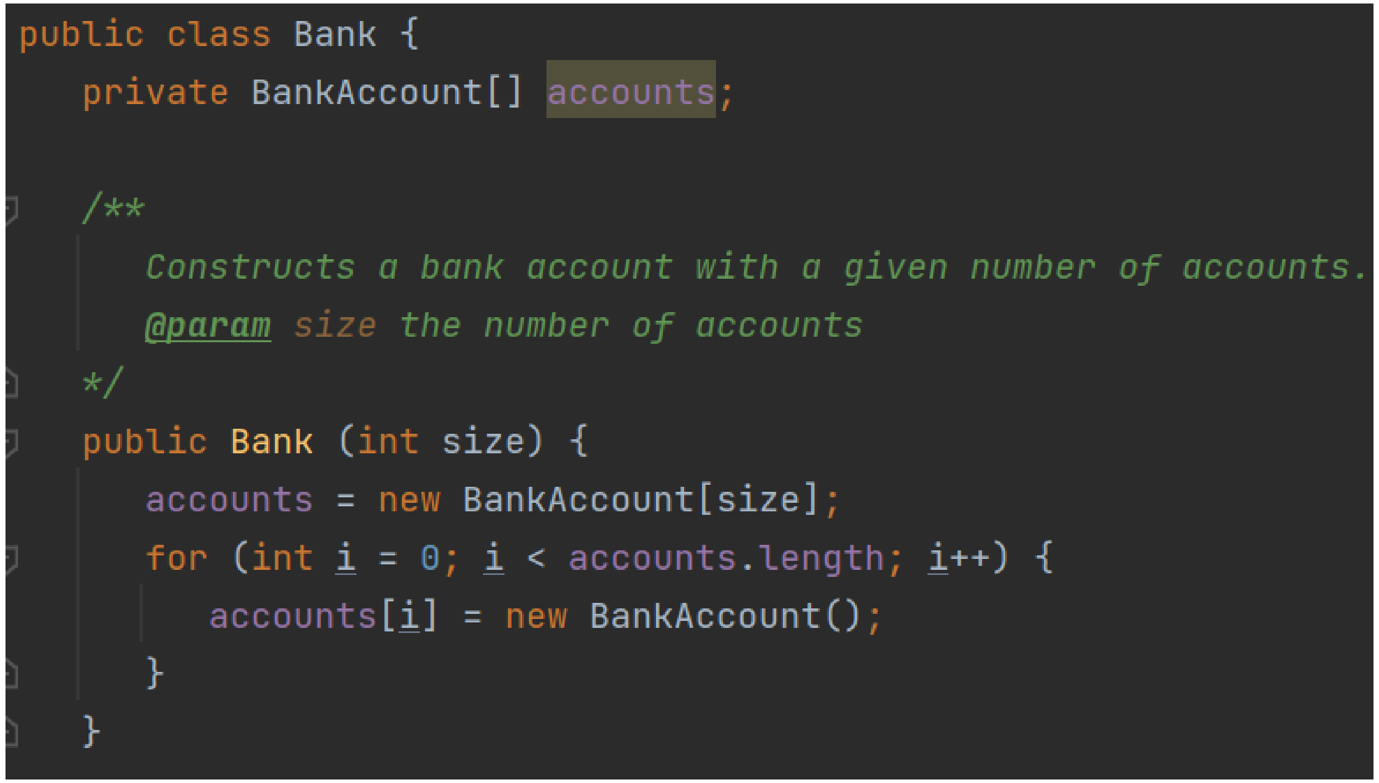
deposit() and withdraw() are properly synchronized

Case Study: Banking Service



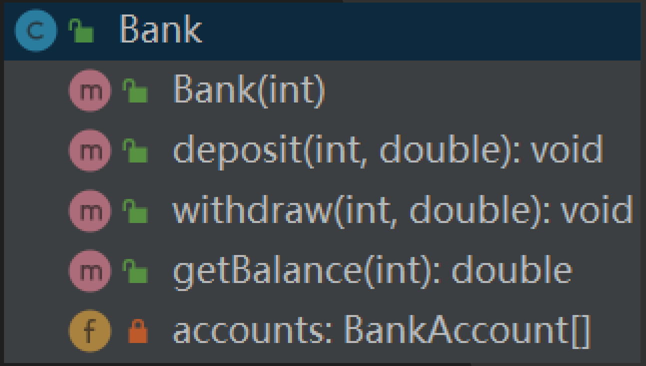
```
Bank
    Bank(int)
    deposit(int, double): void
    withdraw(int, double): void
    getBalance(int): double
    accounts: BankAccount[]
```

- A bank has multiple bank accounts
- A bank can withdraw from or deposit to a certain account



```java
public class Bank {
    private BankAccount[] accounts;

    /**
        Constructs a bank account with a given number of accounts.
        @param size the number of accounts
    */
    public Bank (int size) {
        accounts = new BankAccount[size];
        for (int i = 0; i < accounts.length; i++) {
            accounts[i] = new BankAccount();
        }
    }
```

Case Study: Banking Service



```
Bank
    Bank(int)
    deposit(int, double): void
    withdraw(int, double): void
    getBalance(int): double
    accounts: BankAccount[]
```

- A bank has multiple bank accounts
- A bank can withdraw from or deposit to a certain account

```java
public void deposit (int accountNumber, double amount) {
    BankAccount account = accounts[accountNumber];
    account.deposit( amount);
}
```

```java
public void withdraw (int accountNumber, double amount) {
    BankAccount account = accounts[accountNumber];
    account.withdraw( amount);
}
```

```java
public double getBalance (int accountNumber) {
    BankAccount account = accounts[accountNumber];
    return account.getBalance();
}
```

Banking Service Protocol

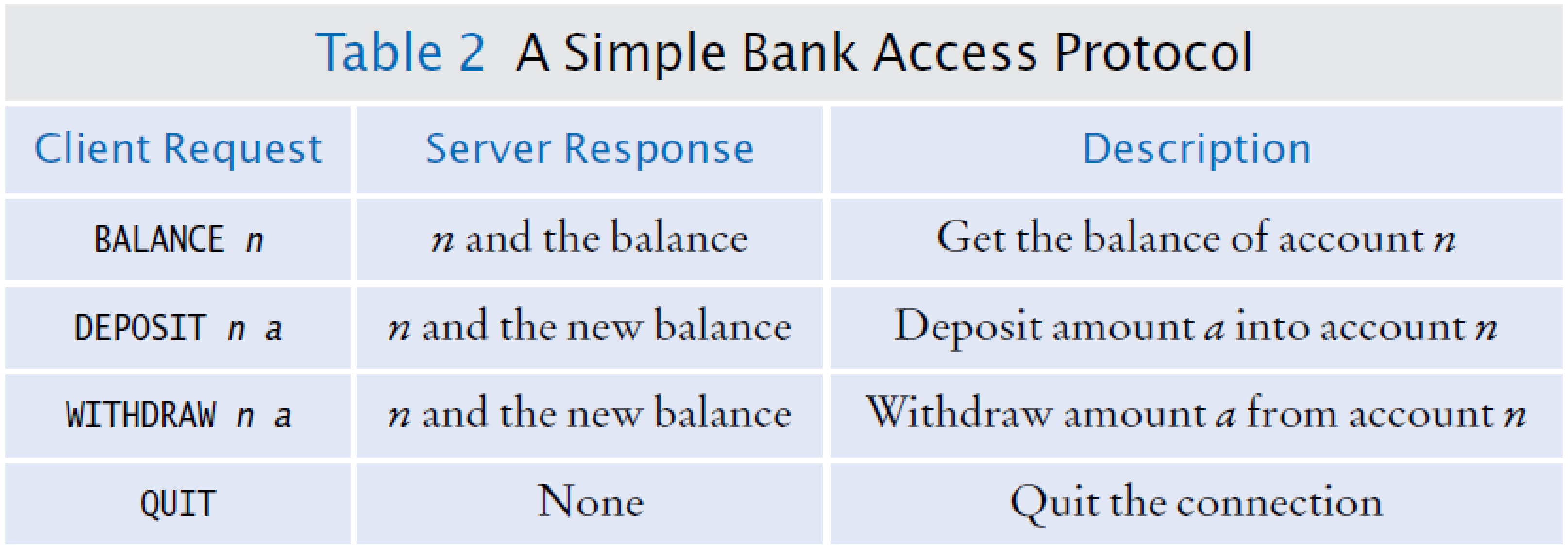
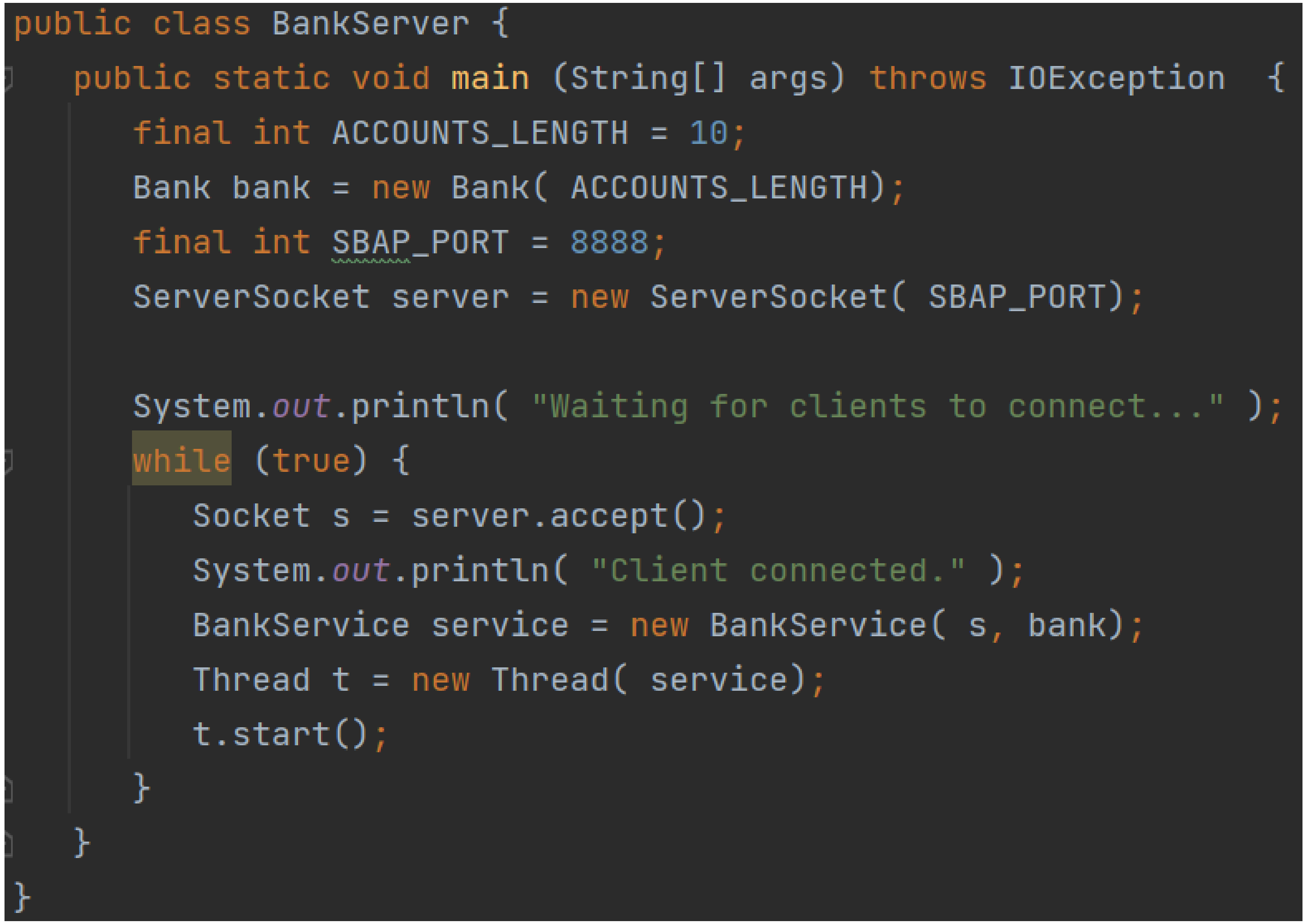


Table 2 A Simple Bank Access Protocol

Client Request	Server Response	Description
`BALANCE n`	*n* and the balance	Get the balance of account *n*
`DEPOSIT n a`	*n* and the new balance	Deposit amount *a* into account *n*
`WITHDRAW n a`	*n* and the new balance	Withdraw amount *a* from account *n*
`QUIT`	None	Quit the connection

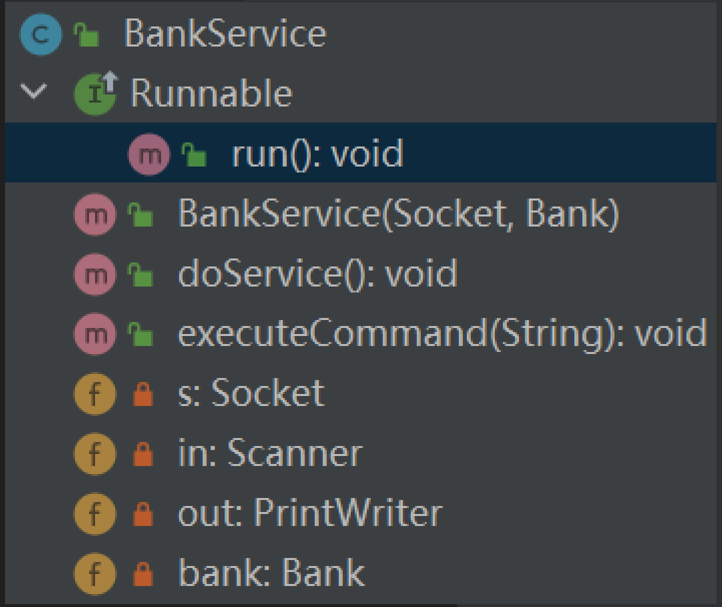
Whenever you develop a server application, you need to specify some application-level protocol that clients can use to interact with the server

Bank Server



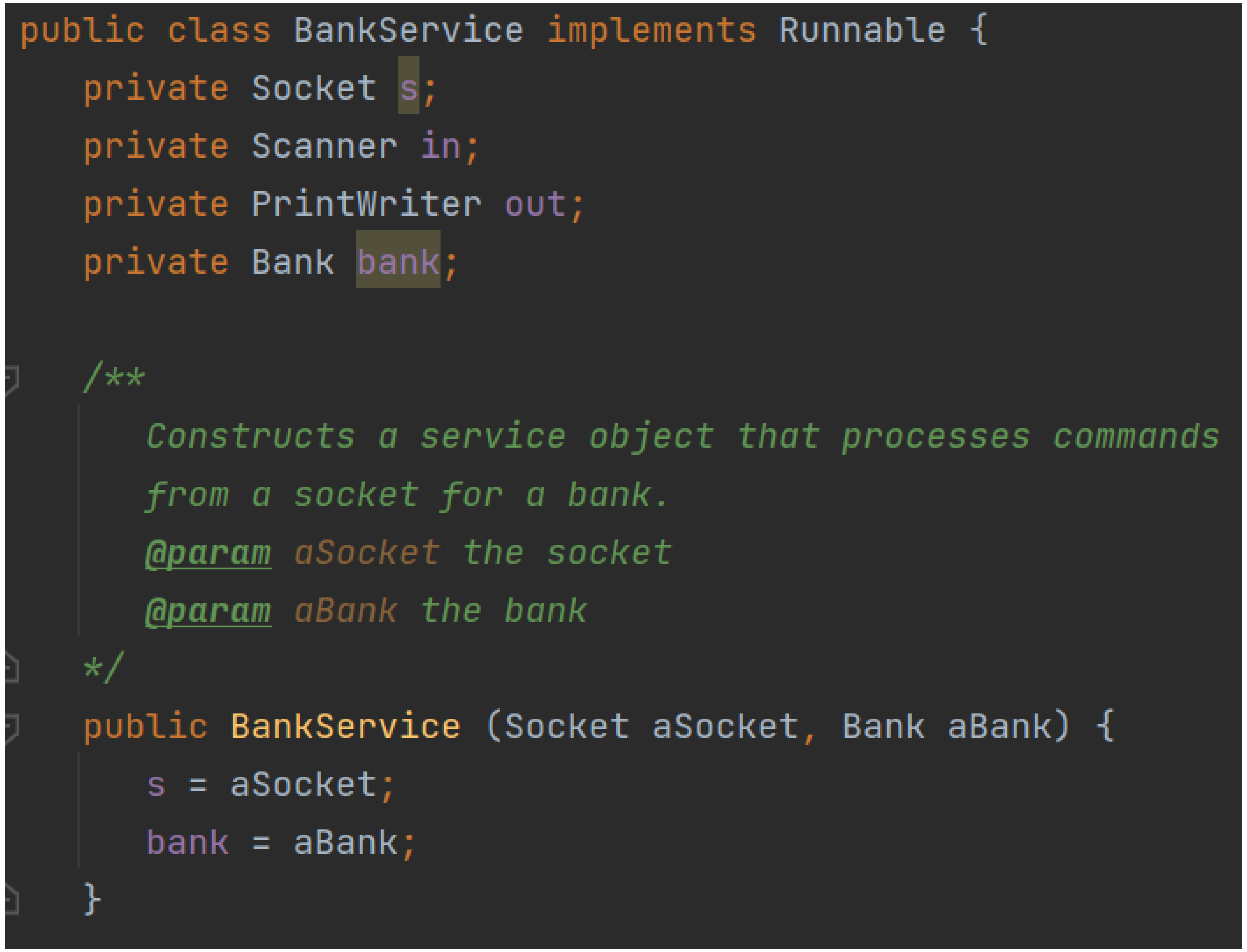
```
public class BankServer {
    public static void main (String[] args) throws IOException  {
        final int ACCOUNTS_LENGTH = 10;
        Bank bank = new Bank( ACCOUNTS_LENGTH);
        final int SBAP_PORT = 8888;
        ServerSocket server = new ServerSocket( SBAP_PORT);

        System.out.println( "Waiting for clients to connect..." );
        while (true) {
            Socket s = server.accept();
            System.out.println( "Client connected." );
            BankService service = new BankService( s, bank);
            Thread t = new Thread( service);
            t.start();
        }
    }
}
```

Case Study: Banking Service



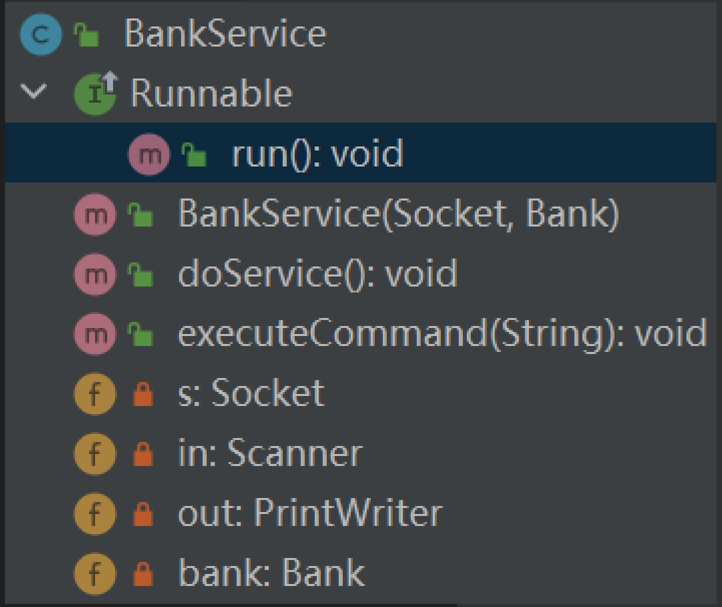
```
BankService
  Runnable
    run(): void
  BankService(Socket, Bank)
  doService(): void
  executeCommand(String): void
  s: Socket
  in: Scanner
  out: PrintWriter
  bank: Bank
```

- A bank service executes the banking service protocol

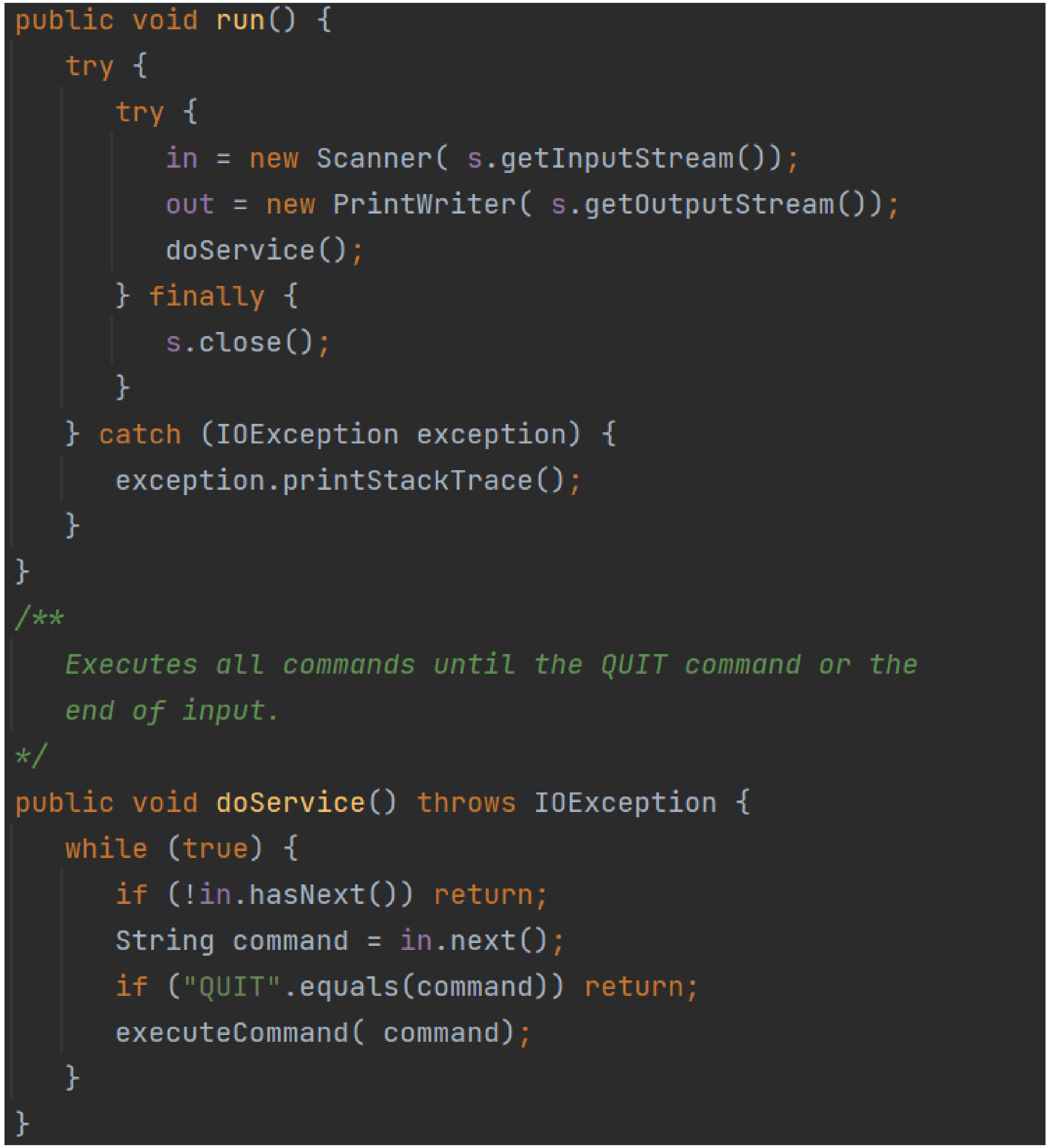


```java
public class BankService implements Runnable {
    private Socket s;
    private Scanner in;
    private PrintWriter out;
    private Bank bank;

    /**
        Constructs a service object that processes commands
        from a socket for a bank.
        @param aSocket the socket
        @param aBank the bank
    */
    public BankService (Socket aSocket, Bank aBank) {
        s = aSocket;
        bank = aBank;
    }
```

Case Study: Banking Service

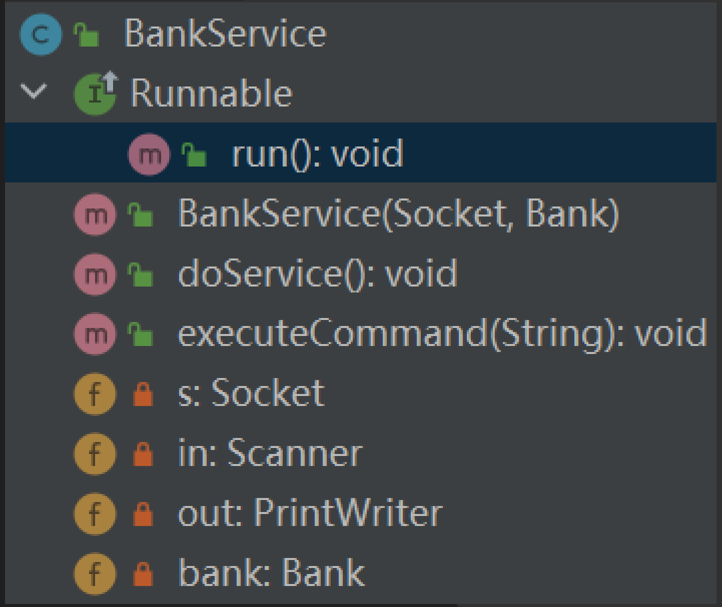


```
BankService
  Runnable
    run(): void
  BankService(Socket, Bank)
  doService(): void
  executeCommand(String): void
  s: Socket
  in: Scanner
  out: PrintWriter
  bank: Bank
```

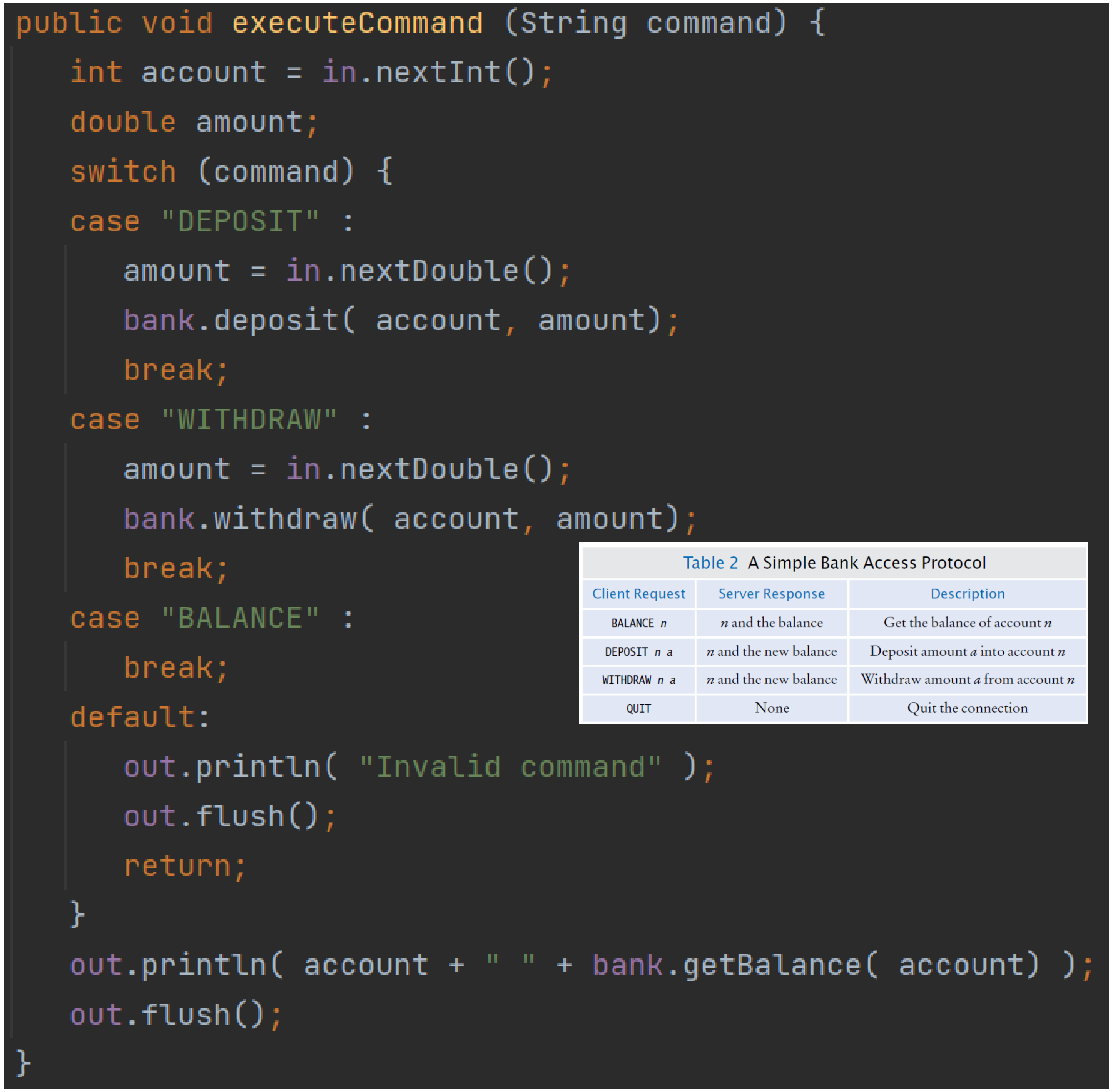


```java
public void run() {
   try {
      try {
         in = new Scanner( s.getInputStream());
         out = new PrintWriter( s.getOutputStream());
         doService();
      } finally {
         s.close();
      }
   } catch (IOException exception) {
      exception.printStackTrace();
   }
}
/**
   Executes all commands until the QUIT command or the
   end of input.
*/
public void doService() throws IOException {
   while (true) {
      if (!in.hasNext()) return;
      String command = in.next();
      if ("QUIT".equals(command)) return;
      executeCommand( command);
   }
}
```

Case Study: Banking Service



```
BankService
  Runnable
    run(): void
  BankService(Socket, Bank)
  doService(): void
  executeCommand(String): void
  s: Socket
  in: Scanner
  out: PrintWriter
  bank: Bank
```



```java
public void executeCommand (String command) {
    int account = in.nextInt();
    double amount;
    switch (command) {
    case "DEPOSIT" :
        amount = in.nextDouble();
        bank.deposit( account, amount);
        break;
    case "WITHDRAW" :
        amount = in.nextDouble();
        bank.withdraw( account, amount);
        break;
    case "BALANCE" :
        break;
    default:
        out.println( "Invalid command" );
        out.flush();
        return;
    }
    out.println( account + " " + bank.getBalance( account) );
    out.flush();
}
```

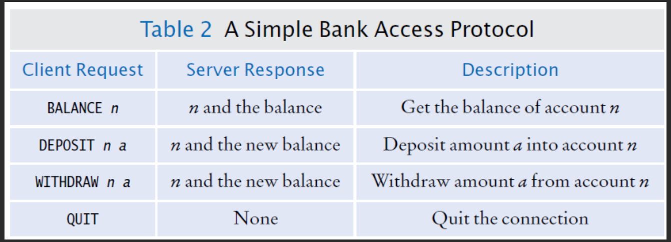
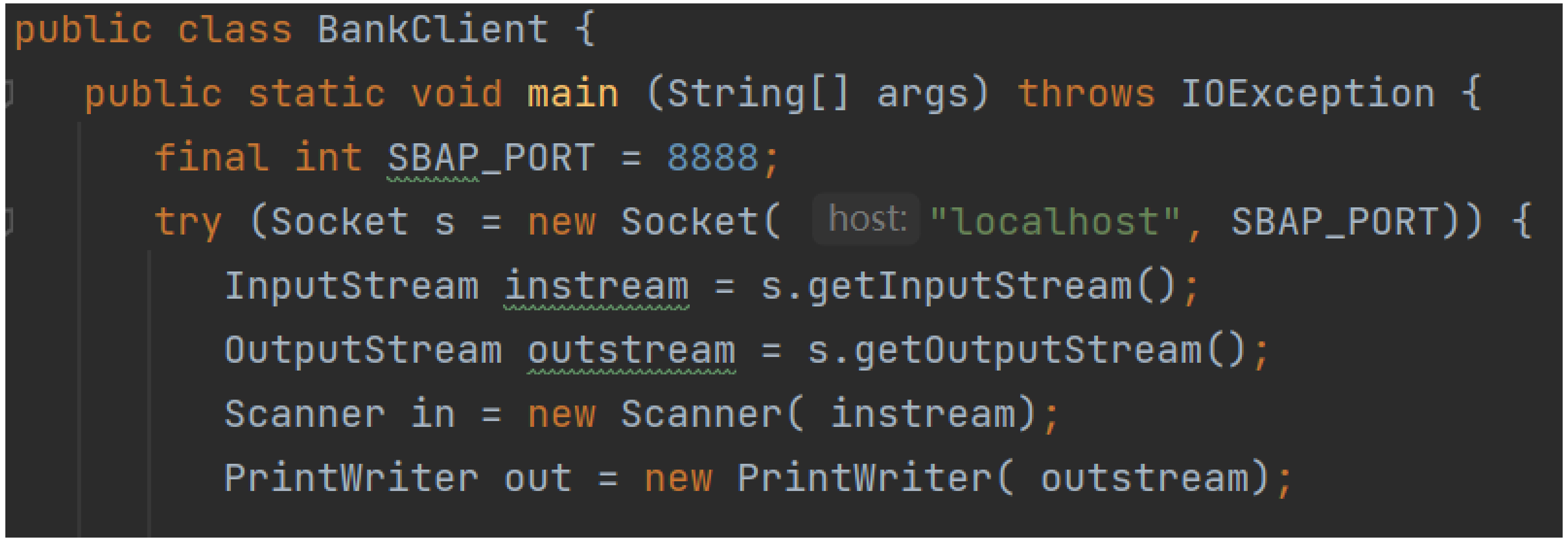


Table 2 A Simple Bank Access Protocol

Client Request	Server Response	Description
`BALANCE` *n*	*n* and the balance	Get the balance of account *n*
`DEPOSIT` *n a*	*n* and the new balance	Deposit amount *a* into account *n*
`WITHDRAW` *n a*	*n* and the new balance	Withdraw amount *a* from account *n*
`QUIT`	None	Quit the connection

Bank Client

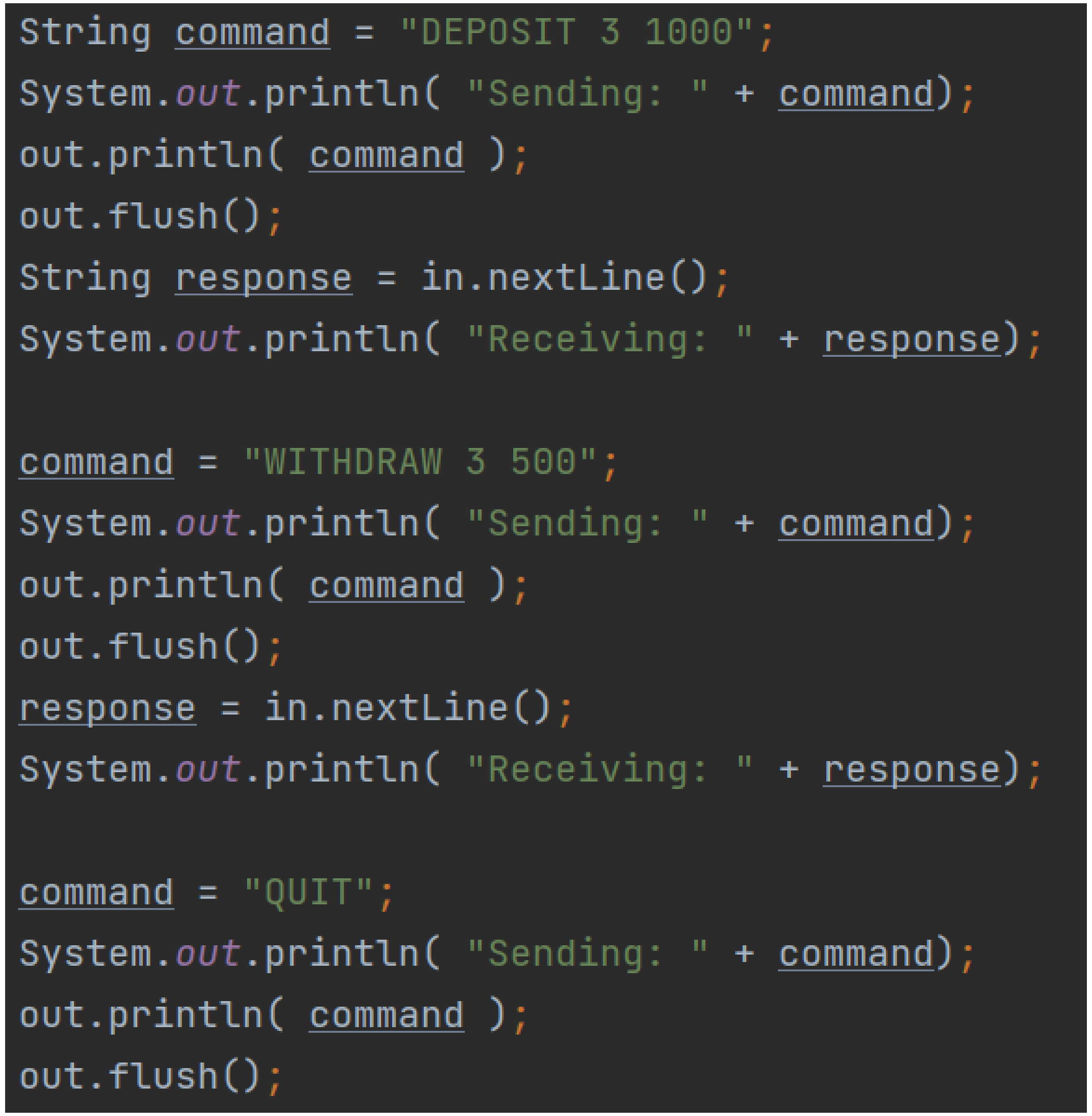


```
public class BankClient {
    public static void main (String[] args) throws IOException {
        final int SBAP_PORT = 8888;
        try (Socket s = new Socket( host: "localhost", SBAP_PORT)) {
            InputStream instream = s.getInputStream();
            OutputStream outstream = s.getOutputStream();
            Scanner in = new Scanner( instream);
            PrintWriter out = new PrintWriter( outstream);
```

To communicate with the server by sending and receiving text, you could turn the streams into scanners and writers

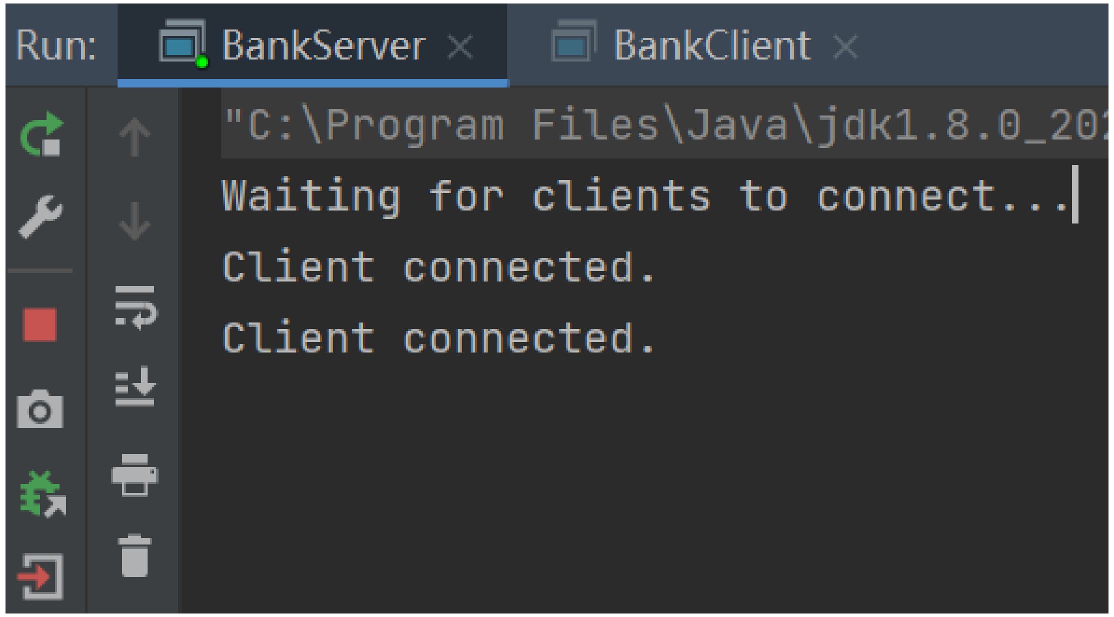
Bank Client

The flush method empties the buffer and forwards all waiting characters to the destination.



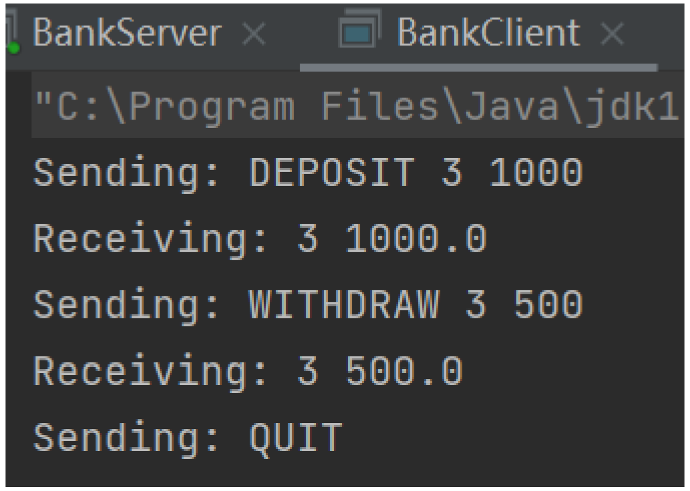
```java
String command = "DEPOSIT 3 1000";
System.out.println( "Sending: " + command);
out.println( command );
out.flush();
String response = in.nextLine();
System.out.println( "Receiving: " + response);

command = "WITHDRAW 3 500";
System.out.println( "Sending: " + command);
out.println( command );
out.flush();
response = in.nextLine();
System.out.println( "Receiving: " + response);

command = "QUIT";
System.out.println( "Sending: " + command);
out.println( command );
out.flush();
```

Case Study

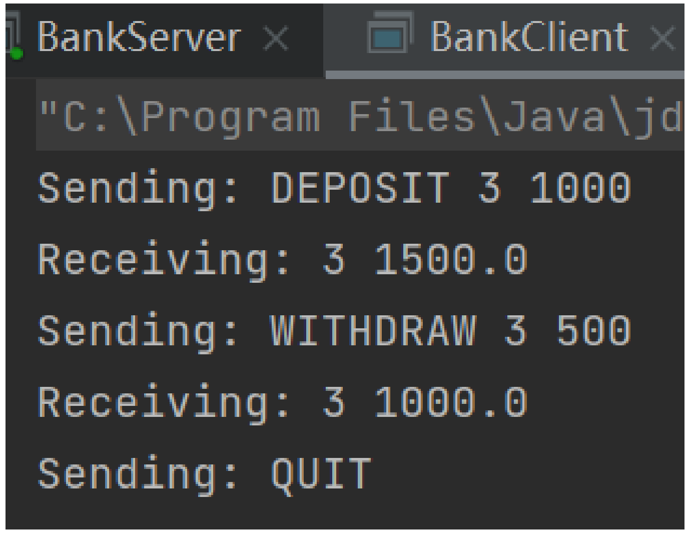


```
"C:\Program Files\Java\jdk1.8.0_20
Waiting for clients to connect...
Client connected.
Client connected.
```

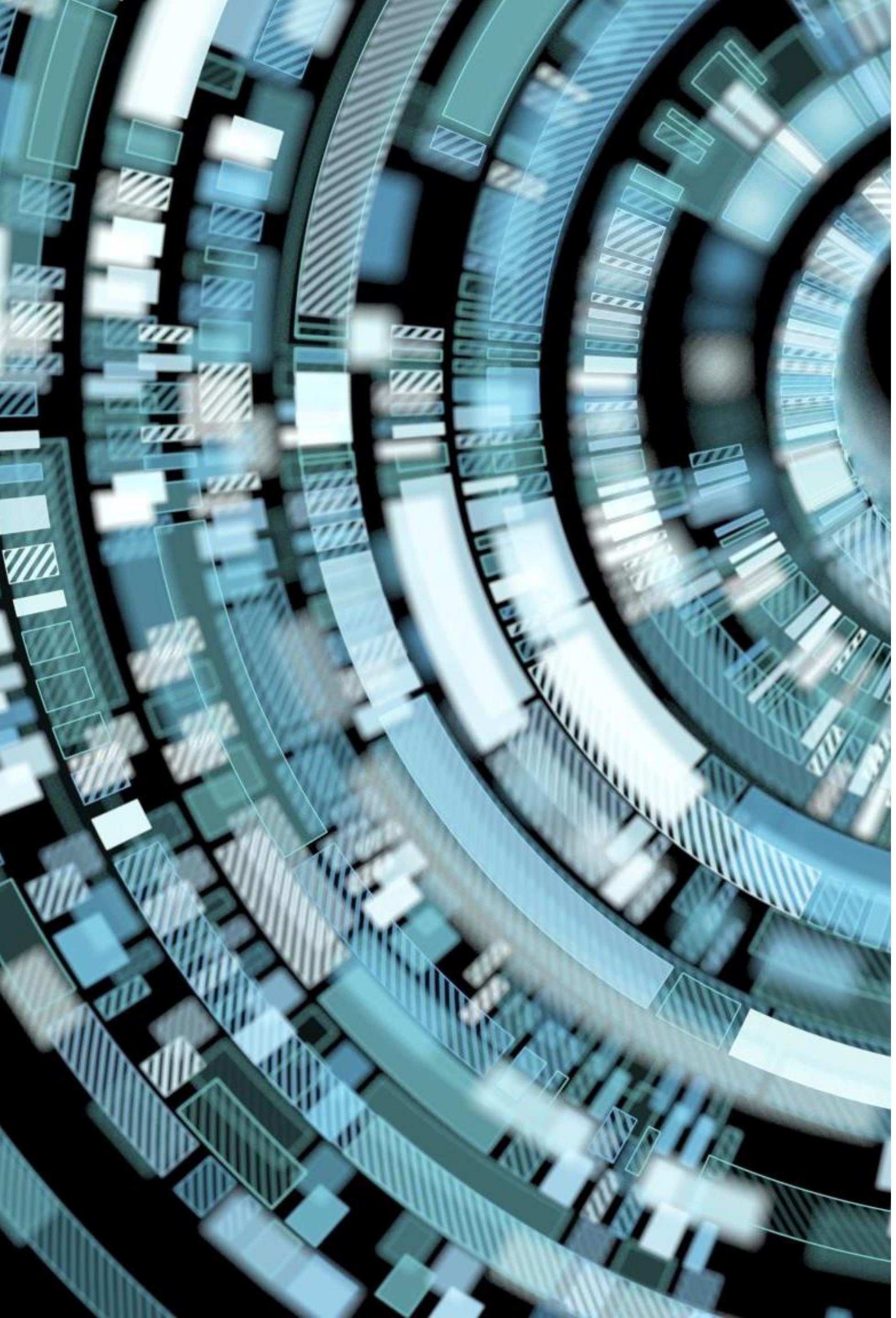
Server keeps running ……



```
"C:\Program Files\Java\jdk1
Sending: DEPOSIT 3 1000
Receiving: 3 1000.0
Sending: WITHDRAW 3 500
Receiving: 3 500.0
Sending: QUIT
```



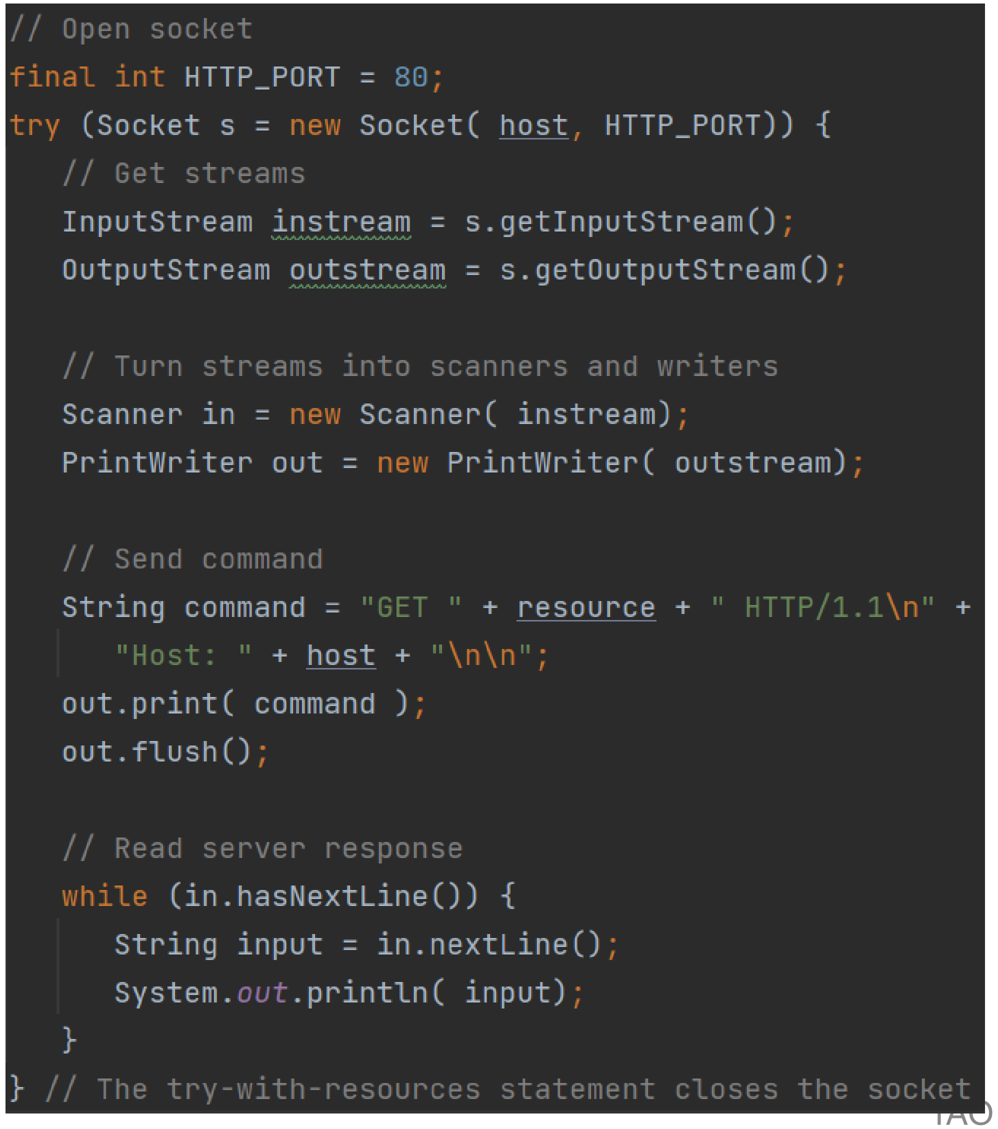
```
"C:\Program Files\Java\jd
Sending: DEPOSIT 3 1000
Receiving: 3 1500.0
Sending: WITHDRAW 3 500
Receiving: 3 1000.0
Sending: QUIT
```



Lecture 8

- Network Basics
- Network Protocols
- Socket Programming
- Getting Web Data
  - java.net package
  - Web scraping libraries
  - REST API

Fetching a web page with socket



```
// Open socket
final int HTTP_PORT = 80;
try (Socket s = new Socket( host, HTTP_PORT)) {
   // Get streams
   InputStream instream = s.getInputStream();
   OutputStream outstream = s.getOutputStream();

   // Turn streams into scanners and writers
   Scanner in = new Scanner( instream);
   PrintWriter out = new PrintWriter( outstream);

   // Send command
   String command = "GET " + resource + " HTTP/1.1\n" +
      "Host: " + host + "\n\n";
   out.print( command );
   out.flush();

   // Read server response
   while (in.hasNextLine()) {
      String input = in.nextLine();
      System.out.println( input);
   }
} // The try-with-resources statement closes the socket
```

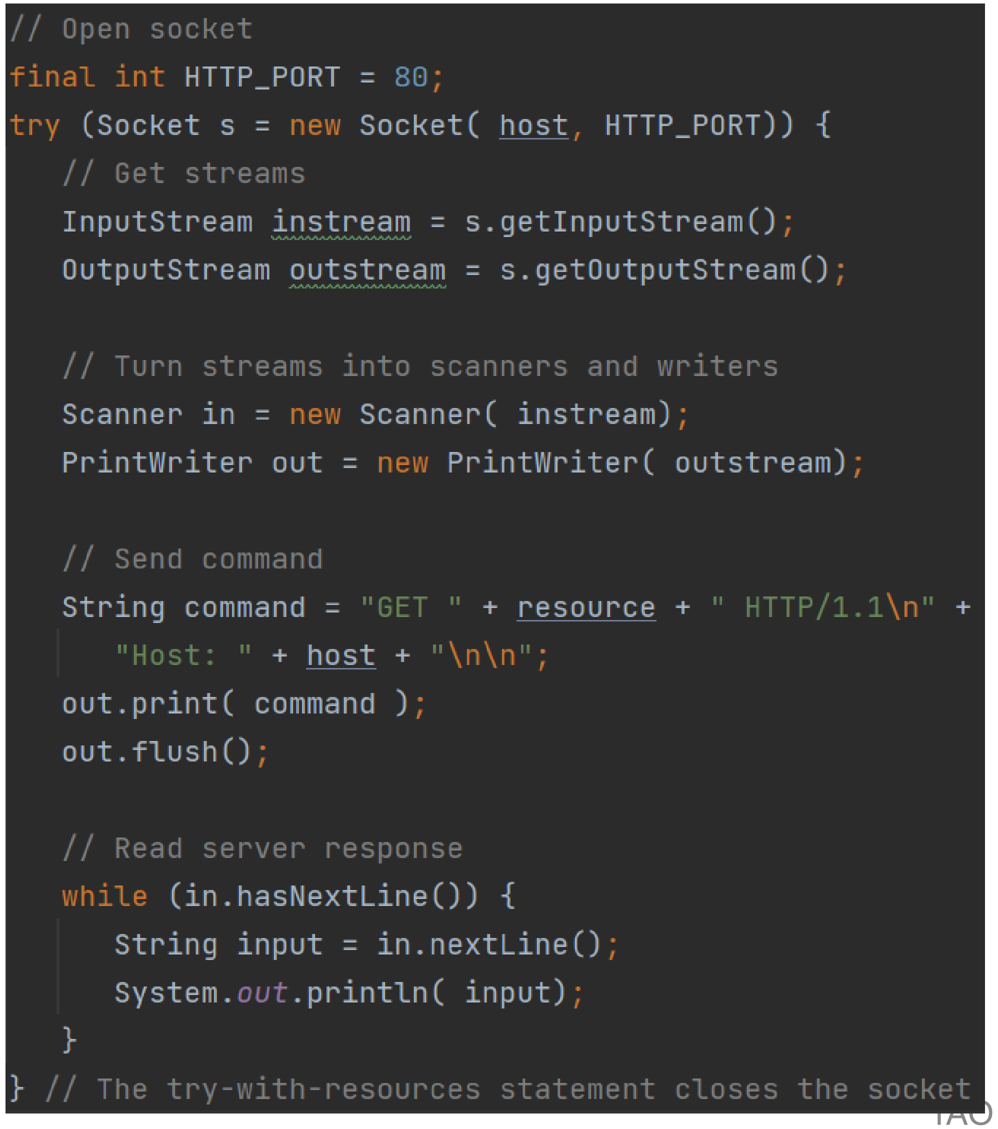
The client establish a Socket with the server. The socket constructor throws an UnknownHostException if it can't find the host.

InputStream and OutputStream classes are used for reading and writing bytes. If you want to communicate with the server by sending and receiving text, you should turn the streams into scanners and writers

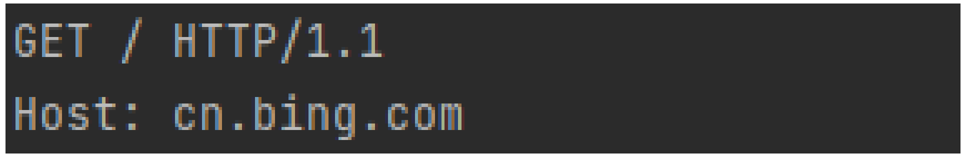
A print writer buffer characters. We need to flush the buffer manually so that the server get a complete request

Receive responses from the server

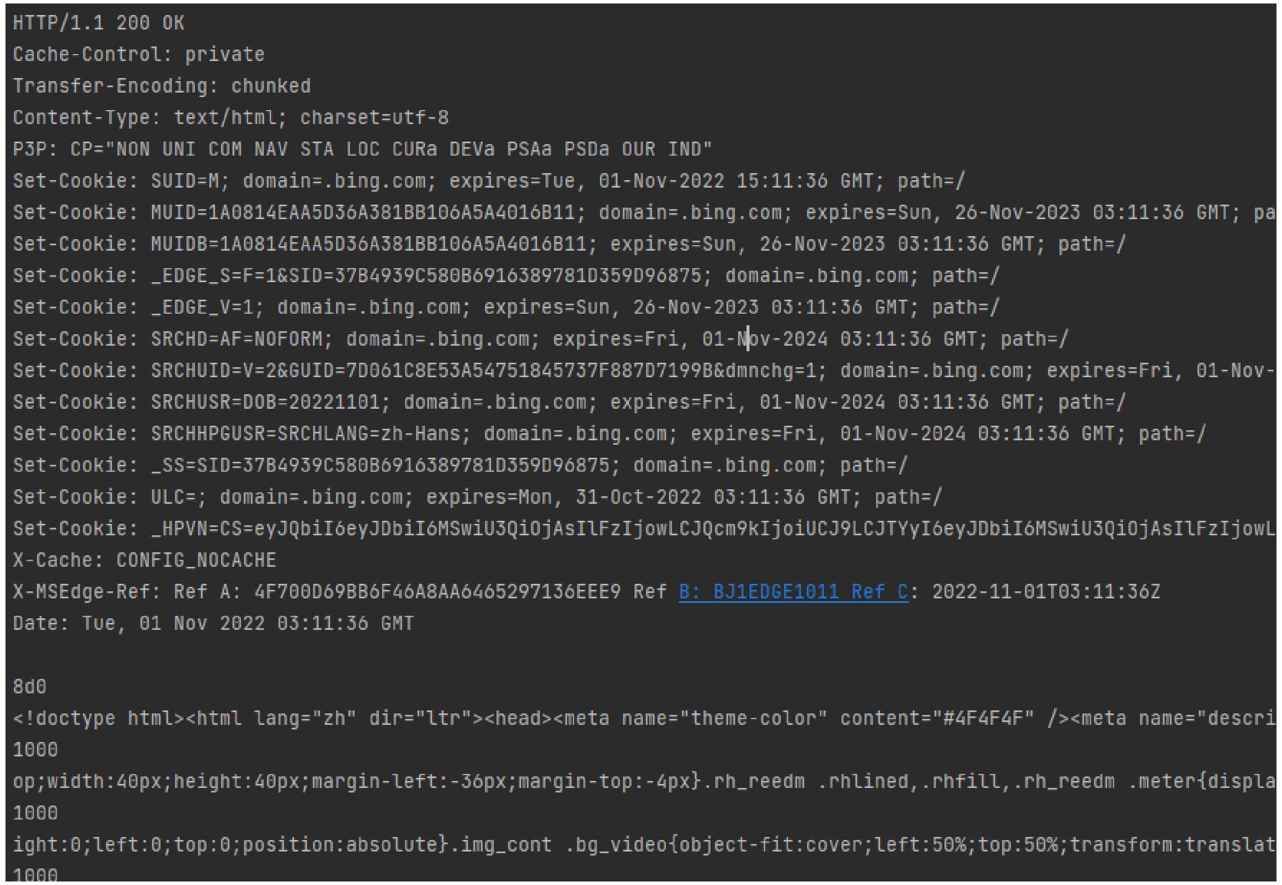
Fetching a web page with socket



```java
// Open socket
final int HTTP_PORT = 80;
try (Socket s = new Socket( host, HTTP_PORT)) {
   // Get streams
   InputStream instream = s.getInputStream();
   OutputStream outstream = s.getOutputStream();

   // Turn streams into scanners and writers
   Scanner in = new Scanner( instream);
   PrintWriter out = new PrintWriter( outstream);

   // Send command
   String command = "GET " + resource + " HTTP/1.1\n" +
      "Host: " + host + "\n\n";
   out.print( command );
   out.flush();

   // Read server response
   while (in.hasNextLine()) {
      String input = in.nextLine();
      System.out.println( input);
   }
} // The try-with-resources statement closes the socket
```

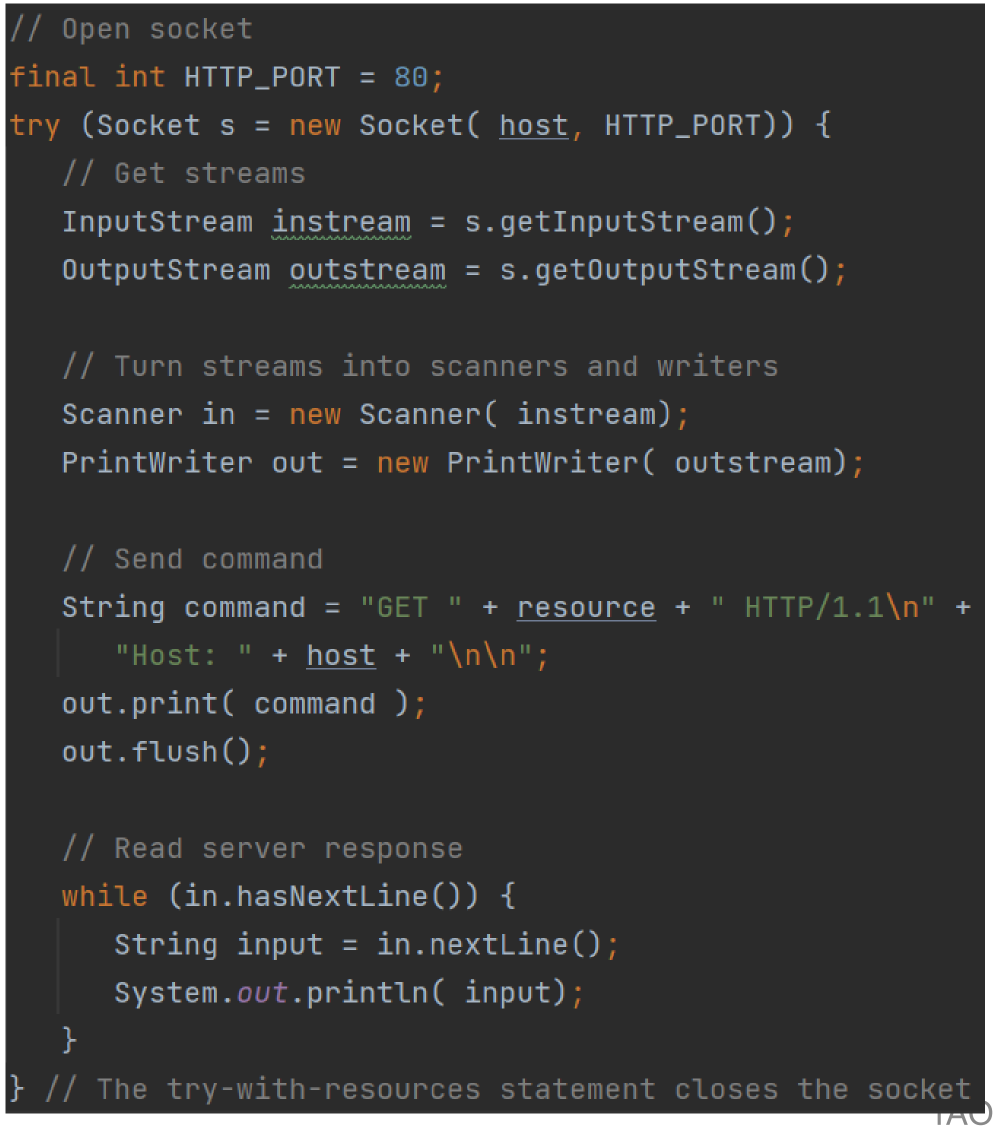


```
GET / HTTP/1.1
Host: cn.bing.com
```



```
HTTP/1.1 200 OK
Cache-Control: private
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
P3P: CP="NON UNI COM NAV STA LOC CURa DEVa PSAa PSDa OUR IND"
Set-Cookie: SUID=M; domain=.bing.com; expires=Tue, 01-Nov-2022 15:11:36 GMT; path=/
Set-Cookie: MUID=1A0814EAA5D36A381BB106A5A4016B11; domain=.bing.com; expires=Sun, 26-Nov-2023 03:11:36 GMT; pa
Set-Cookie: MUIDB=1A0814EAA5D36A381BB106A5A4016B11; expires=Sun, 26-Nov-2023 03:11:36 GMT; path=/
Set-Cookie: _EDGE_S=F=1&SID=37B4939C580B6916389781D359D96875; domain=.bing.com; path=/
Set-Cookie: _EDGE_V=1; domain=.bing.com; expires=Sun, 26-Nov-2023 03:11:36 GMT; path=/
Set-Cookie: SRCHD=AF=NOFORM; domain=.bing.com; expires=Fri, 01-Nov-2024 03:11:36 GMT; path=/
Set-Cookie: SRCHUID=V=2&GUID=7D061C8E53A54751845737F887D7199B&dmnchg=1; domain=.bing.com; expires=Fri, 01-Nov-
Set-Cookie: SRCHUSR=DOB=20221101; domain=.bing.com; expires=Fri, 01-Nov-2024 03:11:36 GMT; path=/
Set-Cookie: SRCHHPGUSR=SRCHLANG=zh-Hans; domain=.bing.com; expires=Fri, 01-Nov-2024 03:11:36 GMT; path=/
Set-Cookie: _SS=SID=37B4939C580B6916389781D359D96875; domain=.bing.com; path=/
Set-Cookie: ULC=; domain=.bing.com; expires=Mon, 31-Oct-2022 03:11:36 GMT; path=/
Set-Cookie: _HPVN=CS=eyJQbiI6eyJDbiI6MSwiU3QiOjAsIlFzIjowLCJQcm9kIjoiUCJ9LCJTYyI6eyJDbiI6MSwiU3QiOjAsIlFzIjowL
X-Cache: CONFIG_NOCACHE
X-MSEdge-Ref: Ref A: 4F700D69BB6F46A8AA6465297136EEE9 Ref B: BJ1EDGE1011 Ref C: 2022-11-01T03:11:36Z
Date: Tue, 01 Nov 2022 03:11:36 GMT

8d0
<!doctype html><html lang="zh" dir="ltr"><head><meta name="theme-color" content="#4F4F4F" /><meta name="descri
1000
op;width:40px;height:40px;margin-left:-36px;margin-top:-4px}.rh_reedm .rhlined,.rhfill,.rh_reedm .meter{displa
1000
ight:0;left:0;top:0;position:absolute}.img_cont .bg_video{object-fit:cover;left:50%;top:50%;transform:translat
1000
```

Fetching a web page with socket



```
// Open socket
final int HTTP_PORT = 80;
try (Socket s = new Socket( host, HTTP_PORT)) {
    // Get streams
    InputStream instream = s.getInputStream();
    OutputStream outstream = s.getOutputStream();

    // Turn streams into scanners and writers
    Scanner in = new Scanner( instream);
    PrintWriter out = new PrintWriter( outstream);

    // Send command
    String command = "GET " + resource + " HTTP/1.1\n" +
        "Host: " + host + "\n\n";
    out.print( command );
    out.flush();

    // Read server response
    while (in.hasNextLine()) {
        String input = in.nextLine();
        System.out.println( input);
    }
} // The try-with-resources statement closes the socket
```

Problems

- We have to handle socket connections and socket errors by ourselves
- We have to manually create HTTP requests with the correct format
- We have to manually parse HTTP responses

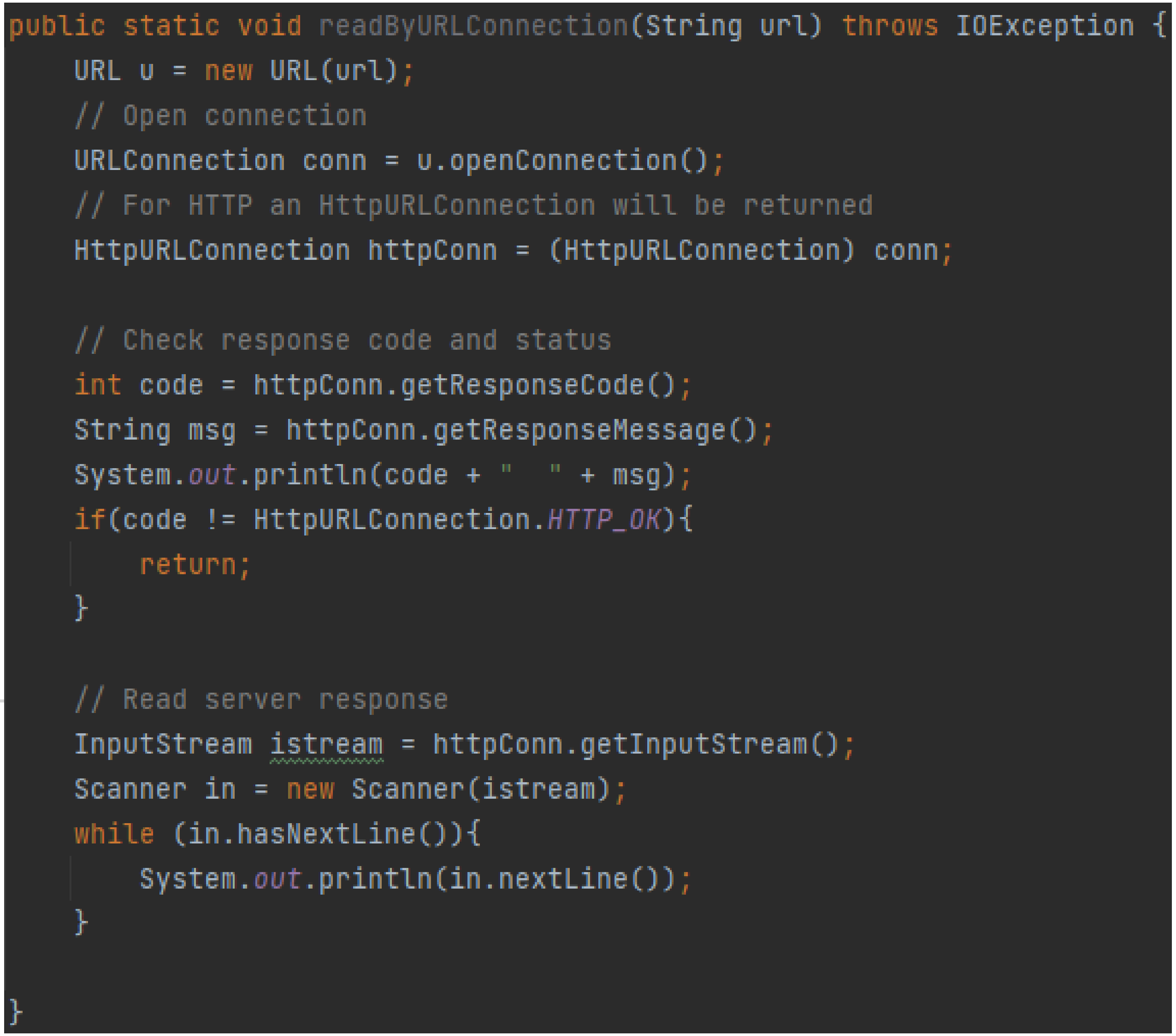
To access web servers in Java, we want to work **at a higher level** than socket connections and HTTP requests

URLConnection

- Java contains a `URLConnection` class (`java.net` package), which provides convenient support for the HTTP
- The `URLConnection` class takes care of the socket connection, so you do not have to fuss with sockets when you want to retrieve from a web server.
- As an additional benefit, the `URLConnection` class can also handle FTP, the file transfer protocol.

Fetching a web page with URLConnection

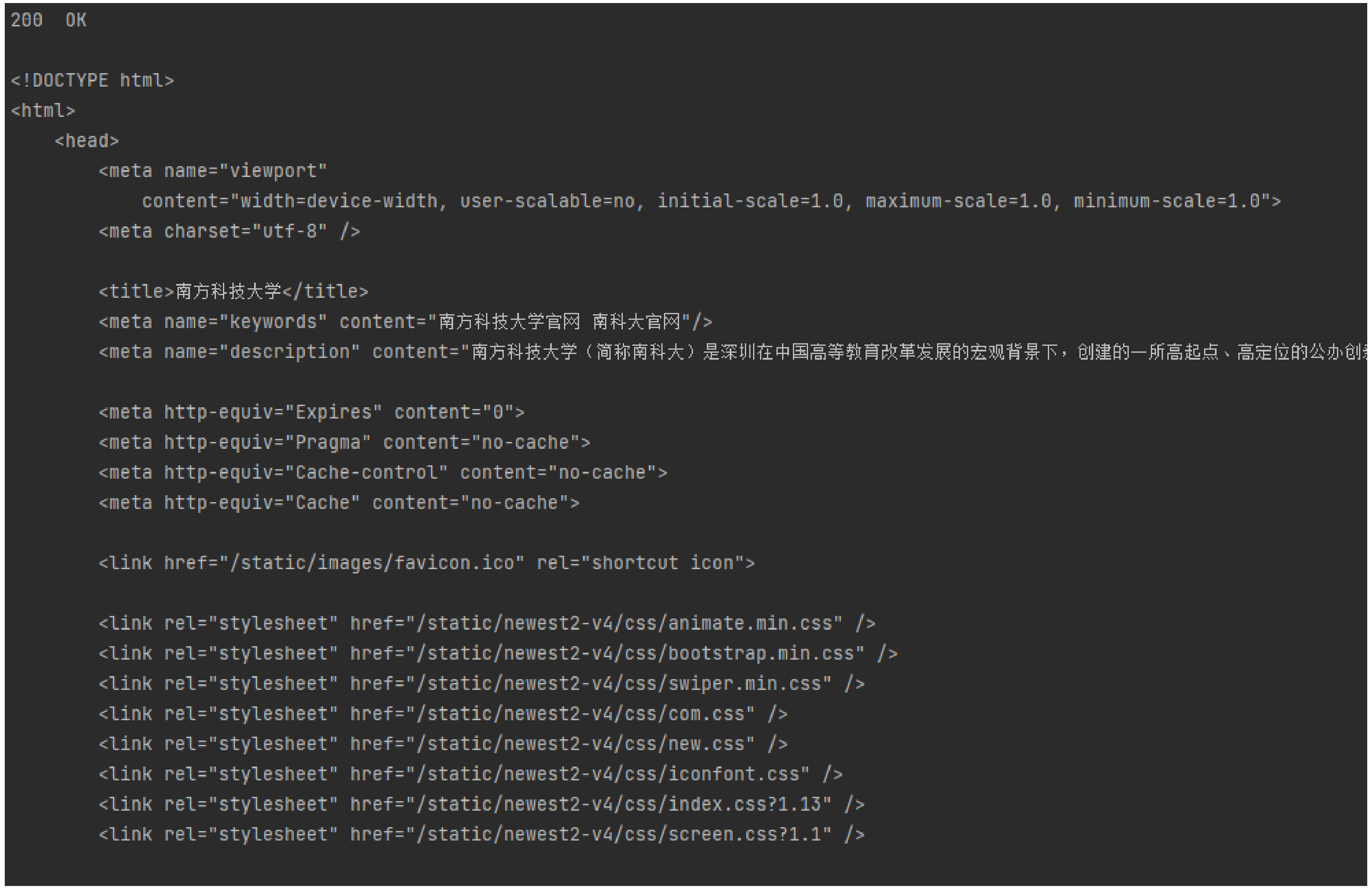
```java
String url = "https://www.sustech.edu.cn";
```



```java
public static void readByURLConnection(String url) throws IOException {
    URL u = new URL(url);
    // Open connection
    URLConnection conn = u.openConnection();
    // For HTTP an HttpURLConnection will be returned
    HttpURLConnection httpConn = (HttpURLConnection) conn;

    // Check response code and status
    int code = httpConn.getResponseCode();
    String msg = httpConn.getResponseMessage();
    System.out.println(code + "   " + msg);
    if(code != HttpURLConnection.HTTP_OK){
        return;
    }

    // Read server response
    InputStream istream = httpConn.getInputStream();
    Scanner in = new Scanner(istream);
    while (in.hasNextLine()){
        System.out.println(in.nextLine());
    }

}
```

Fetching a web page with URLConnection

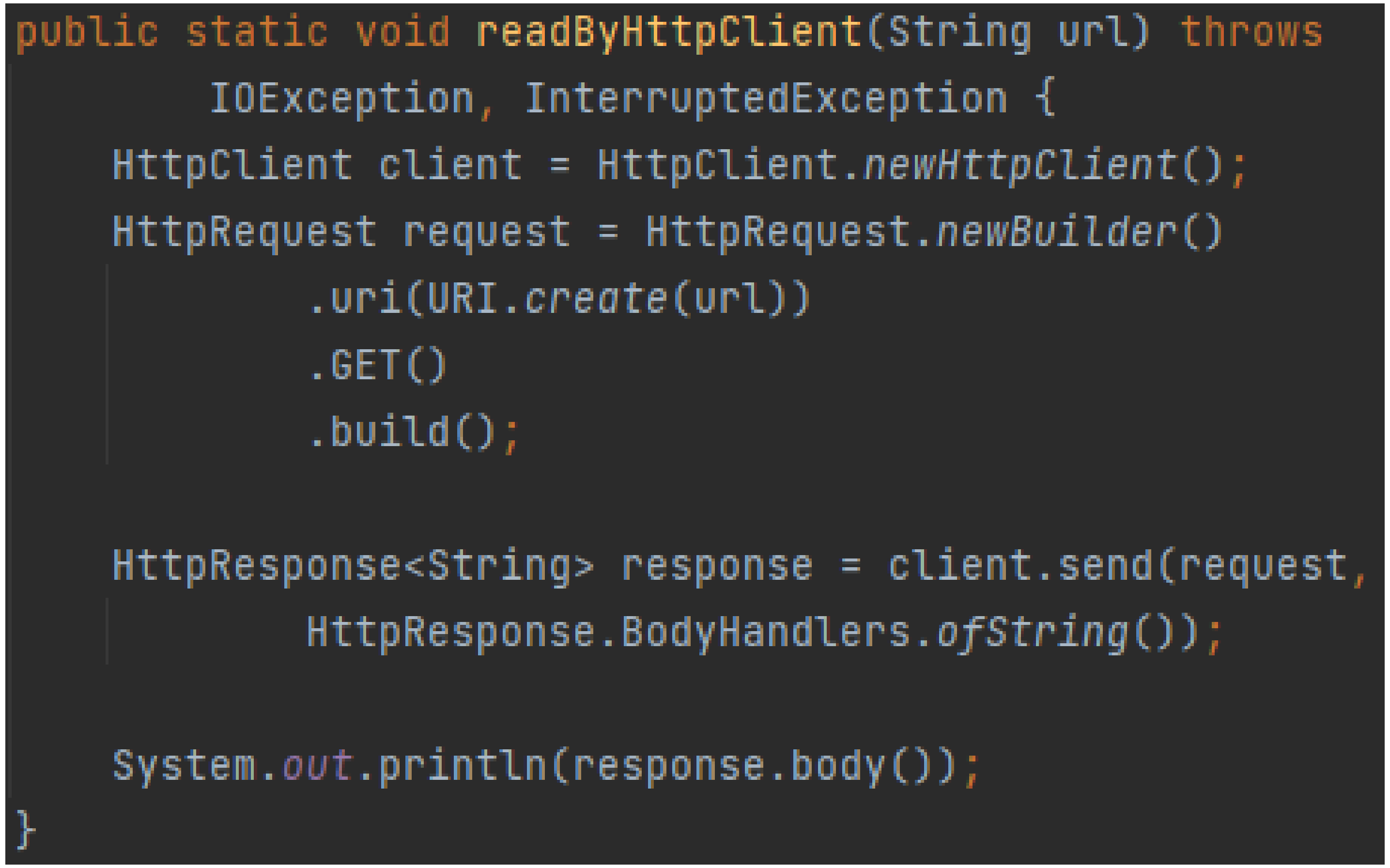
```
String url = "https://www.sustech.edu.cn";
```



```
200  OK

<!DOCTYPE html>
<html>
    <head>
        <meta name="viewport"
            content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
        <meta charset="utf-8" />

        <title>南方科技大学</title>
        <meta name="keywords" content="南方科技大学官网 南科大官网"/>
        <meta name="description" content="南方科技大学（简称南科大）是深圳在中国高等教育改革发展的宏观背景下，创建的一所高起点、高定位的公办创

        <meta http-equiv="Expires" content="0">
        <meta http-equiv="Pragma" content="no-cache">
        <meta http-equiv="Cache-control" content="no-cache">
        <meta http-equiv="Cache" content="no-cache">

        <link href="/static/images/favicon.ico" rel="shortcut icon">

        <link rel="stylesheet" href="/static/newest2-v4/css/animate.min.css" />
        <link rel="stylesheet" href="/static/newest2-v4/css/bootstrap.min.css" />
        <link rel="stylesheet" href="/static/newest2-v4/css/swiper.min.css" />
        <link rel="stylesheet" href="/static/newest2-v4/css/com.css" />
        <link rel="stylesheet" href="/static/newest2-v4/css/new.css" />
        <link rel="stylesheet" href="/static/newest2-v4/css/iconfont.css" />
        <link rel="stylesheet" href="/static/newest2-v4/css/index.css?1.13" />
        <link rel="stylesheet" href="/static/newest2-v4/css/screen.css?1.1" />
```

Fetching a web page with HttpClient

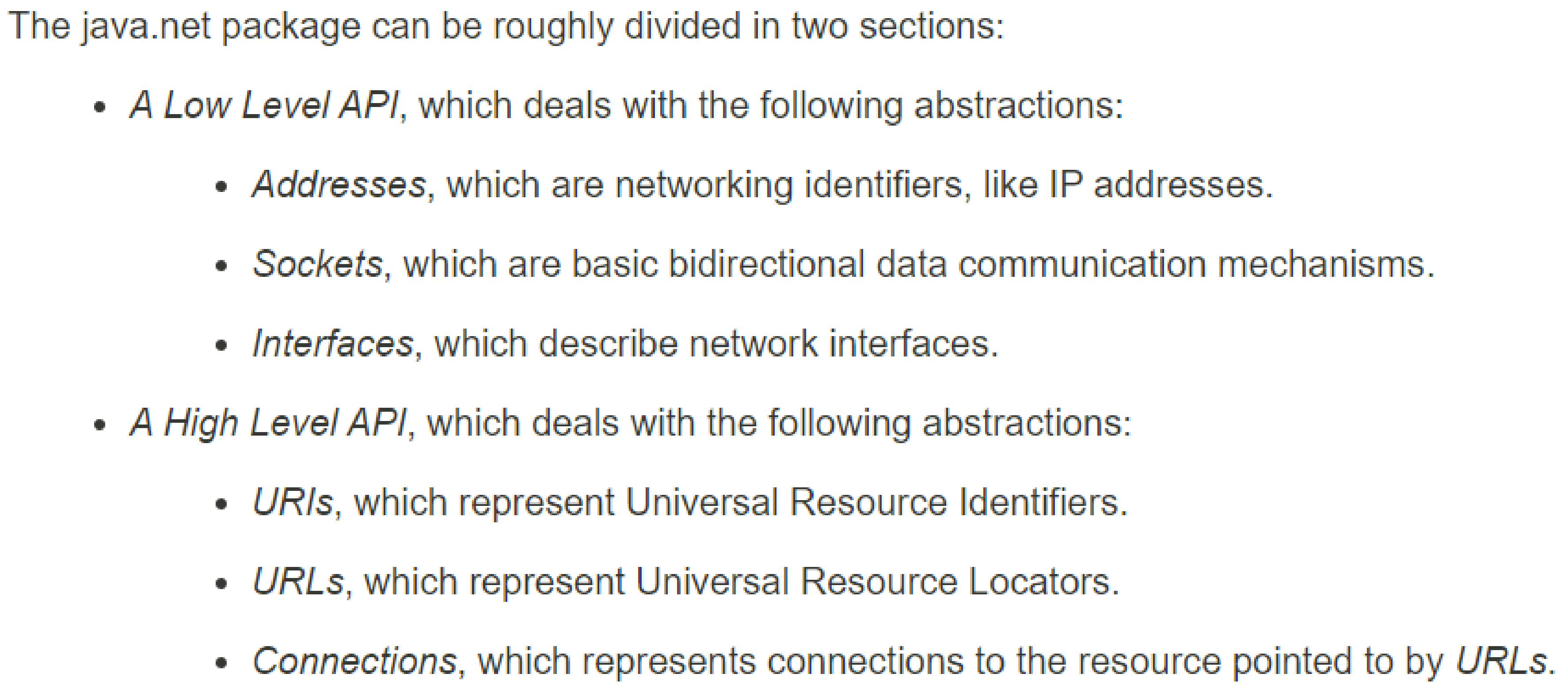
The `java.net.http.HttpClient` API provides an even simpler way to connect to a web server (Java 11)



```java
public static void readByHttpClient(String url) throws
        IOException, InterruptedException {
    HttpClient client = HttpClient.newHttpClient();
    HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(url))
            .GET()
            .build();

    HttpResponse<String> response = client.send(request,
            HttpResponse.BodyHandlers.ofString());

    System.out.println(response.body());
}
```

java.net package

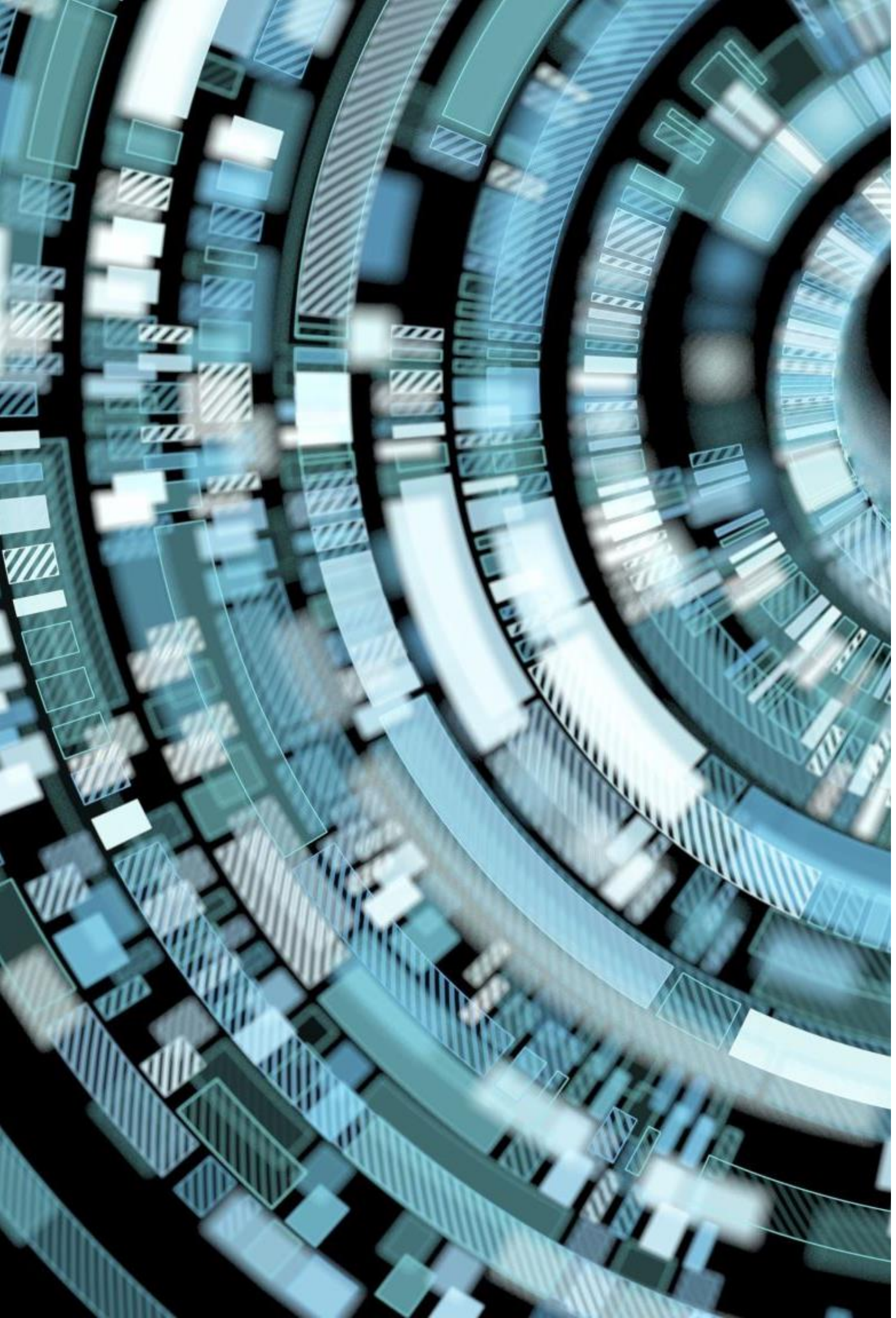
Provides the classes for implementing networking applications.



The java.net package can be roughly divided in two sections:

- *A Low Level API*, which deals with the following abstractions:
    - *Addresses*, which are networking identifiers, like IP addresses.
    - *Sockets*, which are basic bidirectional data communication mechanisms.
    - *Interfaces*, which describe network interfaces.
- *A High Level API*, which deals with the following abstractions:
    - *URIs*, which represent Universal Resource Identifiers.
    - *URLs*, which represent Universal Resource Locators.
    - *Connections*, which represents connections to the resource pointed to by *URLs*.

https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html#package_description

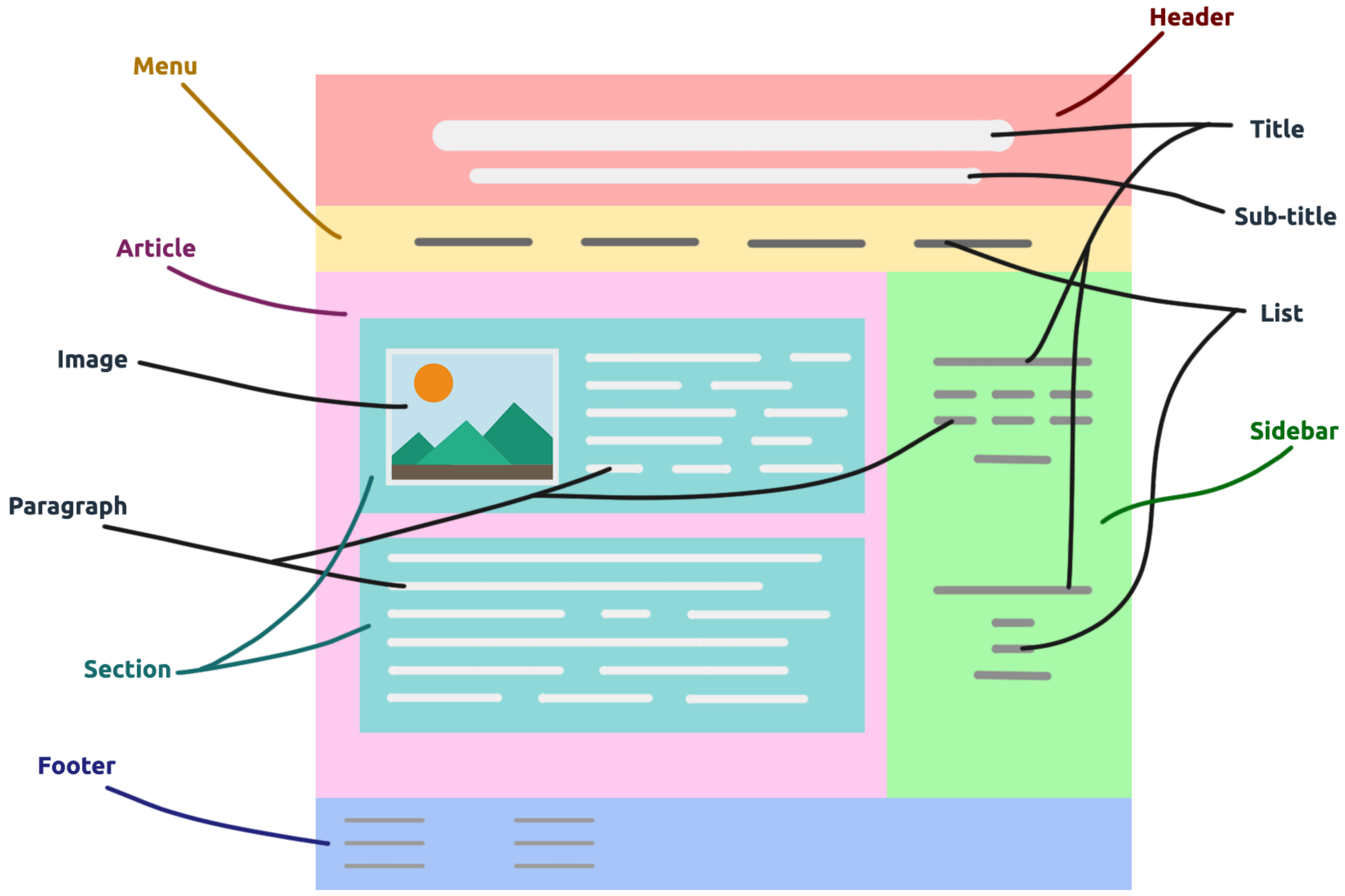


Lecture 8

- Network Basics
- Network Protocols
- Socket Programming
- Getting Web Data
  - java.net package
  - Web scraping libraries
  - REST API

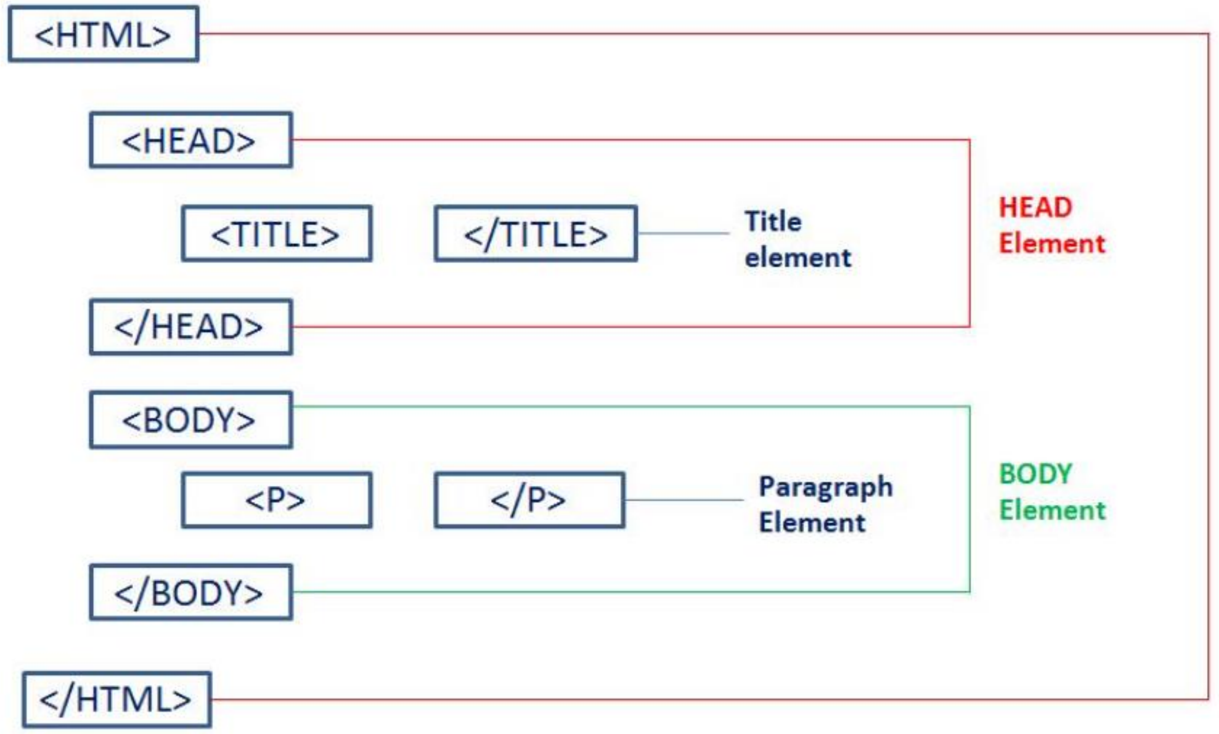
Web Scraping

- Web scraping refers to the process of extracting of data from websites.
- Typically using bots/spiders to automatically navigate through web pages and extract data



https://www.development-tutorial.com/basic-structure-html-page/

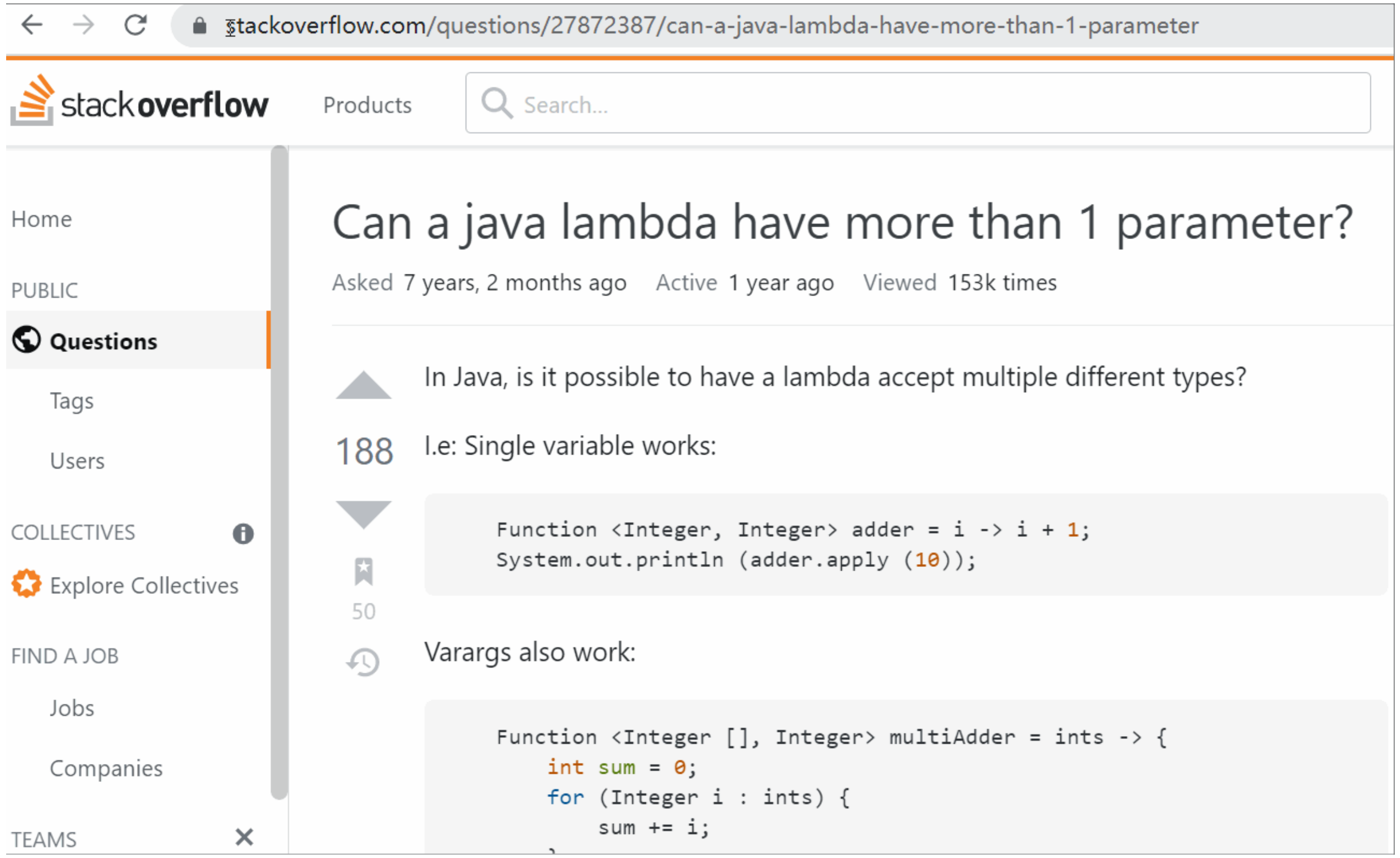
How are web pages created?



https://www.etutorialspoint.com/index.php/basic-html/html-elements

- HTML (Hypertext Markup Language): a hypertext markup language for creating web pages
- HTML uses tags for titles, headings, paragraphs, lists, tables, embedded images, etc., to describe the structure of a web page

Inspecting the HTML for an element



Can a java lambda have more than 1 parameter?

Asked 7 years, 2 months ago Active 1 year ago Viewed 153k times

In Java, is it possible to have a lambda accept multiple different types?

I.e: Single variable works:

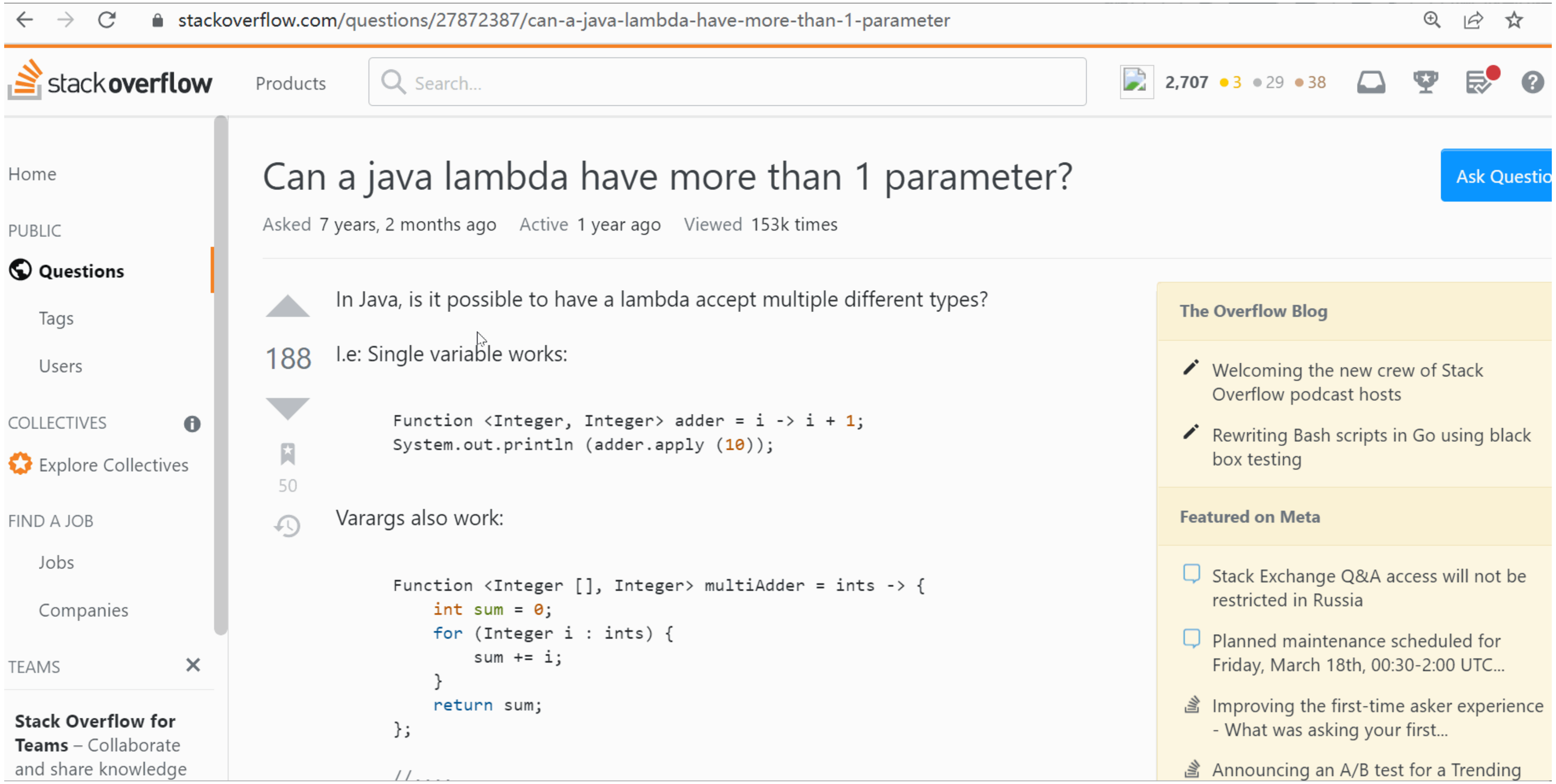
```
Function <Integer, Integer> adder = i -> i + 1;
System.out.println (adder.apply (10));
```

Varargs also work:

```
Function <Integer [], Integer> multiAdder = ints -> {
    int sum = 0;
    for (Integer i : ints) {
        sum += i;
```

What if we want to find the html element for a specific part?

Inspecting the HTML for an element



Stack Overflow for Teams – Collaborate and share knowledge

Can a java lambda have more than 1 parameter?

Asked 7 years, 2 months ago Active 1 year ago Viewed 153k times

In Java, is it possible to have a lambda accept multiple different types?

I.e: Single variable works:

```
Function <Integer, Integer> adder = i -> i + 1;
System.out.println (adder.apply (10));
```

Varargs also work:

```
Function <Integer [], Integer> multiAdder = ints -> {
    int sum = 0;
    for (Integer i : ints) {
        sum += i;
    }
    return sum;
};

//....
```

The Overflow Blog

- Welcoming the new crew of Stack Overflow podcast hosts
- Rewriting Bash scripts in Go using black box testing

Featured on Meta

- Stack Exchange Q&A access will not be restricted in Russia
- Planned maintenance scheduled for Friday, March 18th, 00:30-2:00 UTC...
- Improving the first-time asker experience - What was asking your first...
- Announcing an A/B test for a Trending

Static vs Dynamic Web Pages

- **Static web pages**
  - Server-side rendered HTML: web page is delivered to the user exactly as stored in the server
  - HTML is fixed
- **Dynamic web pages**
  - JavaScript rendered HTML: web page content is created dynamically using JS
  - HTML is changing (e.g., scrolling down a web page to get the news feed)
  - Needs other advanced scraping strategy/libraries

Java Libraries for Web Scraping

1

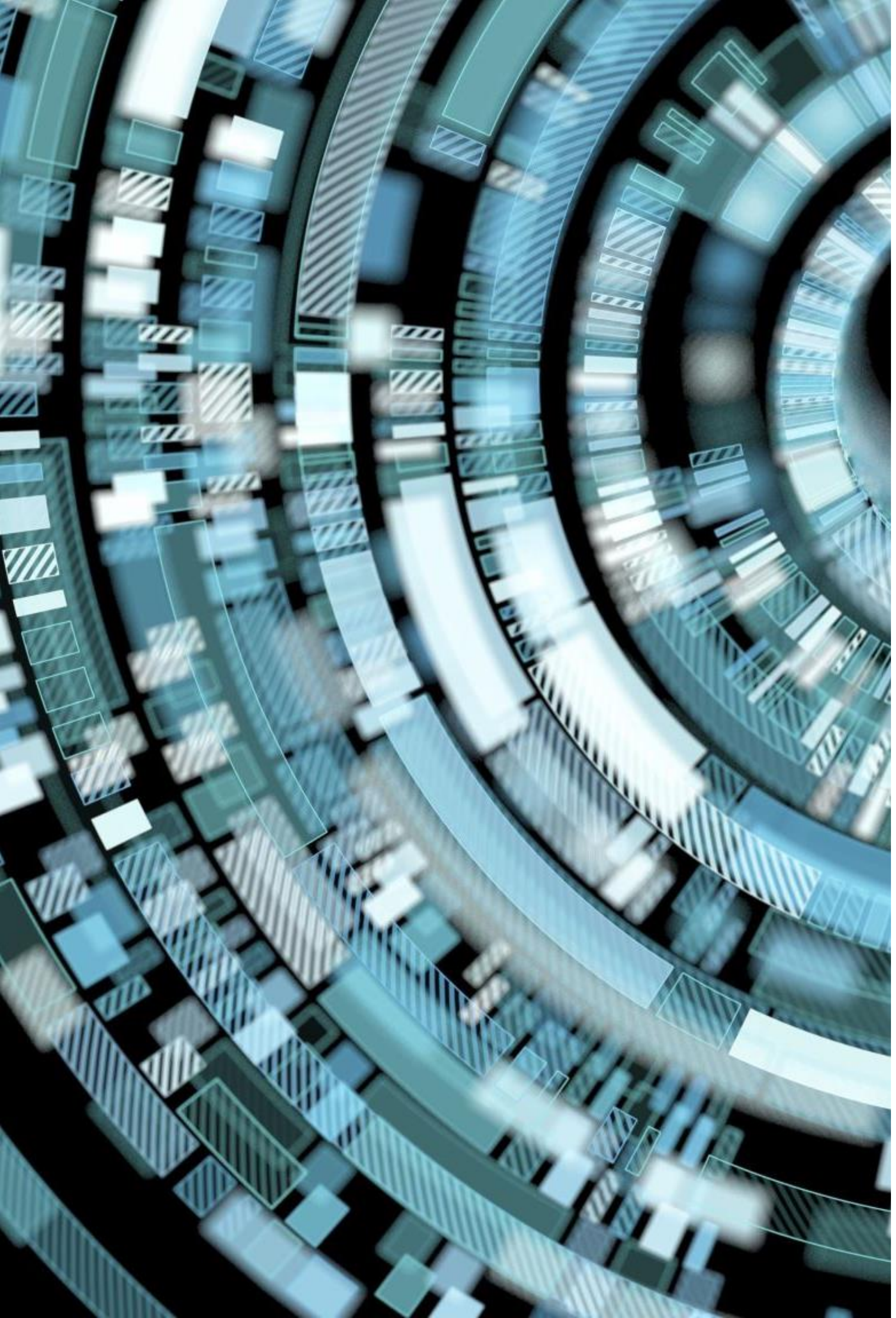
Jsoup: this is a simple open-source library that provides very convenient functionality for extracting and manipulating data by using DOM traversal or CSS selectors to find data. It is beginner friendly.

2

HTMLUnit: is a more powerful framework that can allow you to simulate browser events such as clicking and forms submission when scraping and it also has JavaScript support.

3

Jaunt: can be used to extract data from HTML pages or JSON data payloads by using a headless browser. It has recently been updated to include JavaScript support.



Lecture 8

- Network Basics
- Network Protocols
- Socket Programming
- Getting Web Data
  - java.net package
  - Web scraping libraries
  - REST API

What is REST API?

- **API**
  - An interface for multiple programs to communicate with each other (e.g., public class and methods in `java.net`)
- **REST**
  - **RE**presentational **S**tate **T**ransfer
  - REST is a software architectural style
- **REST API**
  - A REST API is an API conforms to the constraints of REST architectural style

What are the constraints of REST style?

REST Constraints

- Client-server: A client-server architecture made up of clients, servers, and resources (info like text, image, video)
- Resources could be accessed using URL
- Stateless: Resource requests should be made independently of one another
- Requests are made using HTTP protocol
    - GET: get resources
    - POST: create resources
    - PUT/PATCH: update resources
    - DELETE: delete resources

REST API IN ACTION

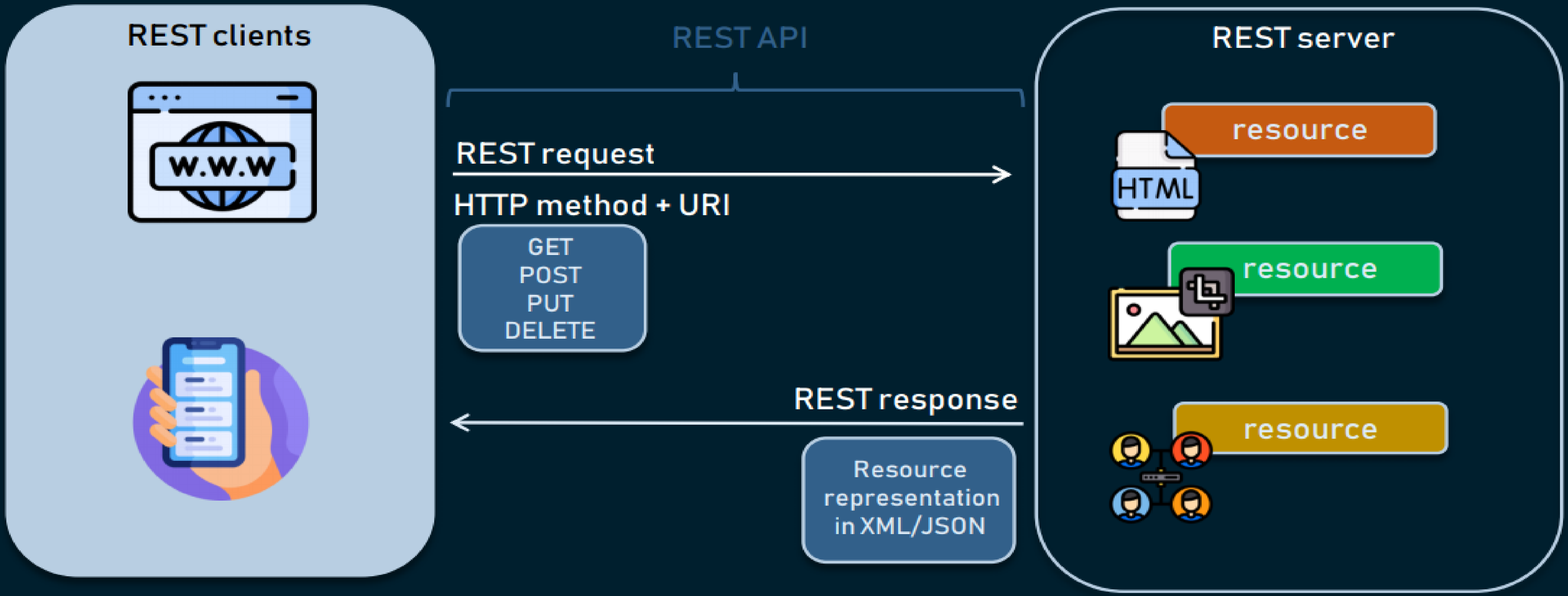


Image source: https://www.altexsoft.com/blog/rest-api-design/

REST API Request Design

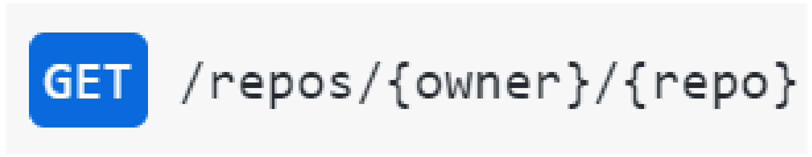
Request = Verb + Object

GET
PUT
PATCH
POST
DELETE

- Typically use noun in plural form, e.g., `questions`
- Allow parameters for filtering, e.g., `?limit=10`

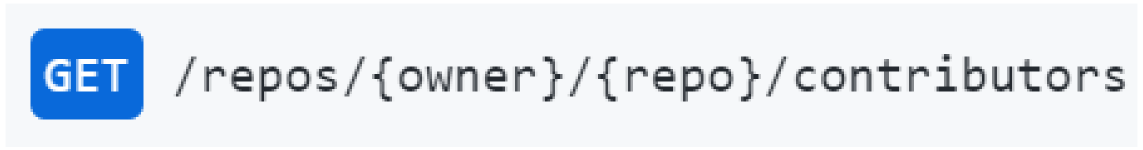
GitHub REST API

URL: https://api.github.com/
Documentation: https://docs.github.com/en/rest



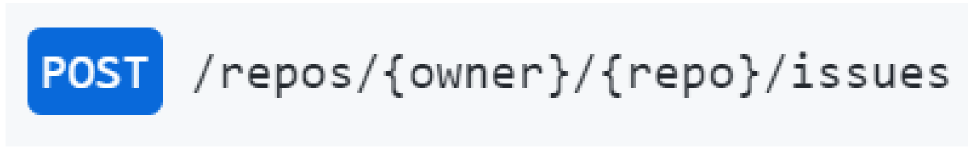
`GET /repos/{owner}/{repo}`

Get a repository info by its owner and repo name



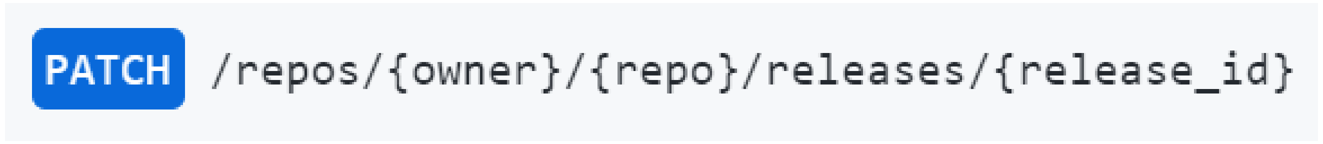
`GET /repos/{owner}/{repo}/contributors`

List repository contributors



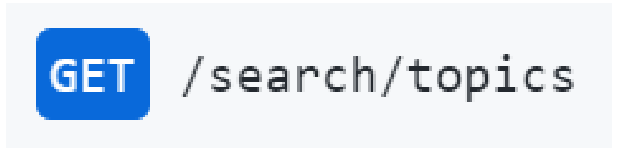
`POST /repos/{owner}/{repo}/issues`

Create an issue (must have pull access to this repo)



`PATCH /repos/{owner}/{repo}/releases/{release_id}`

Update a release (must have push access to this repo)

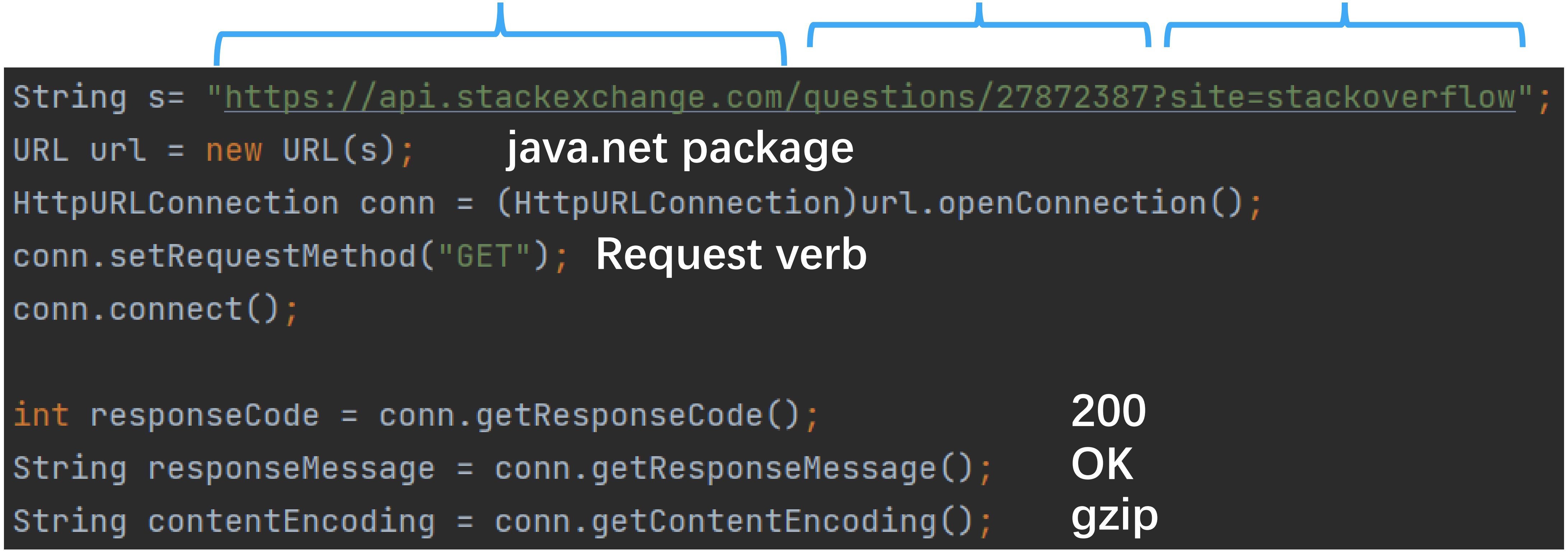


`GET /search/topics`

Search for topics (should specify the topic using parameters)

Stack Overflow REST API

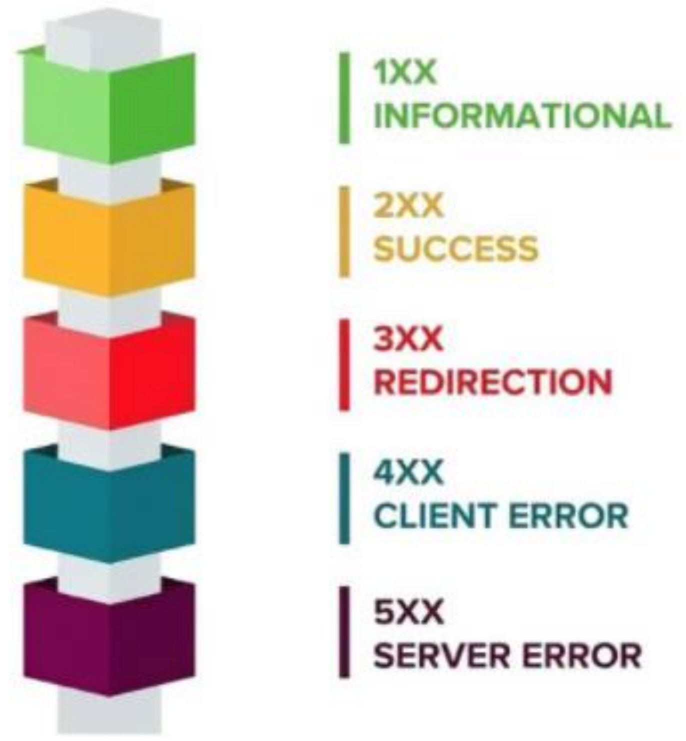
REST Service URL   Requested resource   Parameter

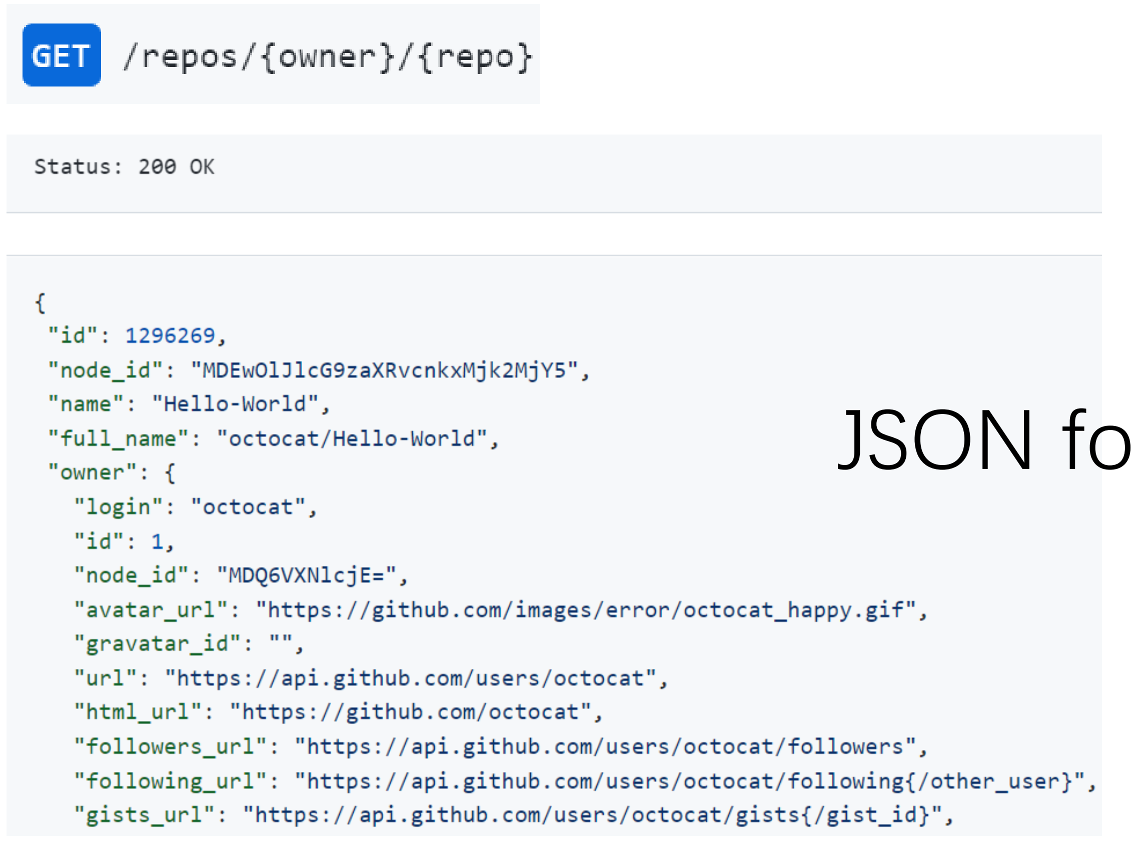


```
String s= "https://api.stackexchange.com/questions/27872387?site=stackoverflow";
URL url = new URL(s);
HttpURLConnection conn = (HttpURLConnection)url.openConnection();
conn.setRequestMethod("GET");
conn.connect();

int responseCode = conn.getResponseCode();
String responseMessage = conn.getResponseMessage();
String contentEncoding = conn.getContentEncoding();
```

Request Response

HTTP Status Code



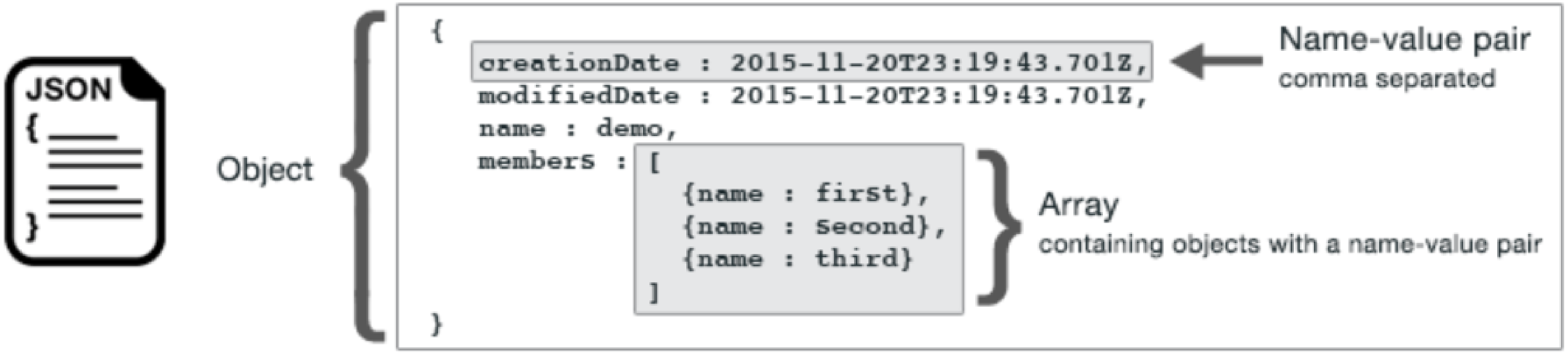


```
/repos/{owner}/{repo}
```

```
Status: 200 OK
```

```
{
 "id": 1296269,
 "node_id": "MDEwOlJlcG9zaXRvcnkxMjk2MjY5",
 "name": "Hello-World",
 "full_name": "octocat/Hello-World",
 "owner": {
   "login": "octocat",
   "id": 1,
   "node_id": "MDQ6VXNlcjE=",
   "avatar_url": "https://github.com/images/error/octocat_happy.gif",
   "gravatar_id": "",
   "url": "https://api.github.com/users/octocat",
   "html_url": "https://github.com/octocat",
   "followers_url": "https://api.github.com/users/octocat/followers",
   "following_url": "https://api.github.com/users/octocat/following{/other_user}",
   "gists_url": "https://api.github.com/users/octocat/gists{/gist_id}",
```

JSON format



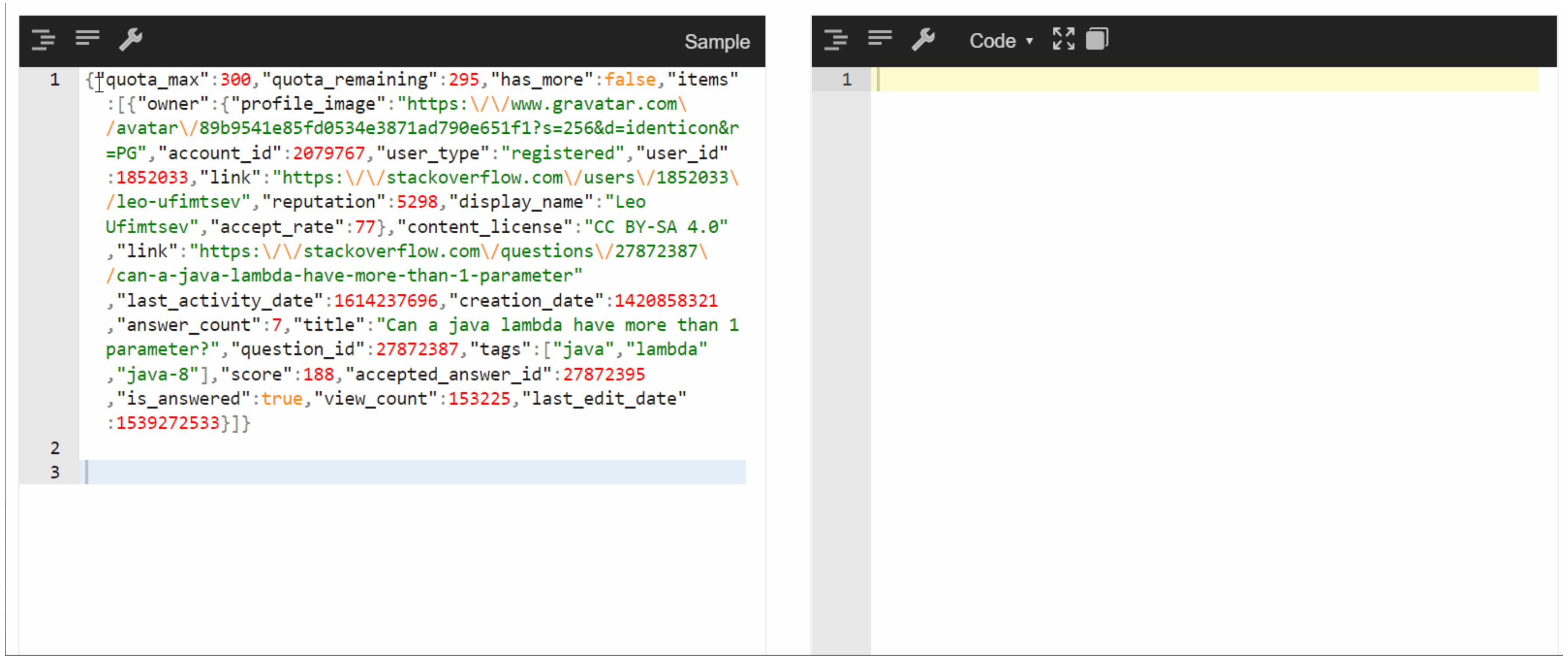
```
{
    creationDate : 2015-11-20T23:19:43.701Z,
    modifiedDate : 2015-11-20T23:19:43.701Z,
    name : demo,
    members : [
                {name : first},
                {name : second},
                {name : third}
              ]
}
```

JSON

- JavaScript Object Notation
- An open data interchange format that is both human and machine-readable
- Independent of any programming language

JSON Helper Tools

- Java Libraries for parsing and creating JSON string: JSON-simple, GSON, Jackson, etc.
- JSON viewers (help formatting the JSON string)



```
{"quota_max":300,"quota_remaining":295,"has_more":false,"items":[{"owner":{"profile_image":"https:\/\/www.gravatar.com\/avatar\/89b9541e85fd0534e3871ad790e651f1?s=256&d=identicon&r=PG","account_id":2079767,"user_type":"registered","user_id":1852033,"link":"https:\/\/stackoverflow.com\/users\/1852033\/leo-ufimtsev","reputation":5298,"display_name":"Leo Ufimtsev","accept_rate":77},"content_license":"CC BY-SA 4.0","link":"https:\/\/stackoverflow.com\/questions\/27872387\/can-a-java-lambda-have-more-than-1-parameter","last_activity_date":1614237696,"creation_date":1420858321,"answer_count":7,"title":"Can a java lambda have more than 1 parameter?","question_id":27872387,"tags":["java","lambda","java-8"],"score":188,"accepted_answer_id":27872395,"is_answered":true,"view_count":153225,"last_edit_date":1539272533}]}
```

Next Lecture

- GUI Intro
- JavaFX