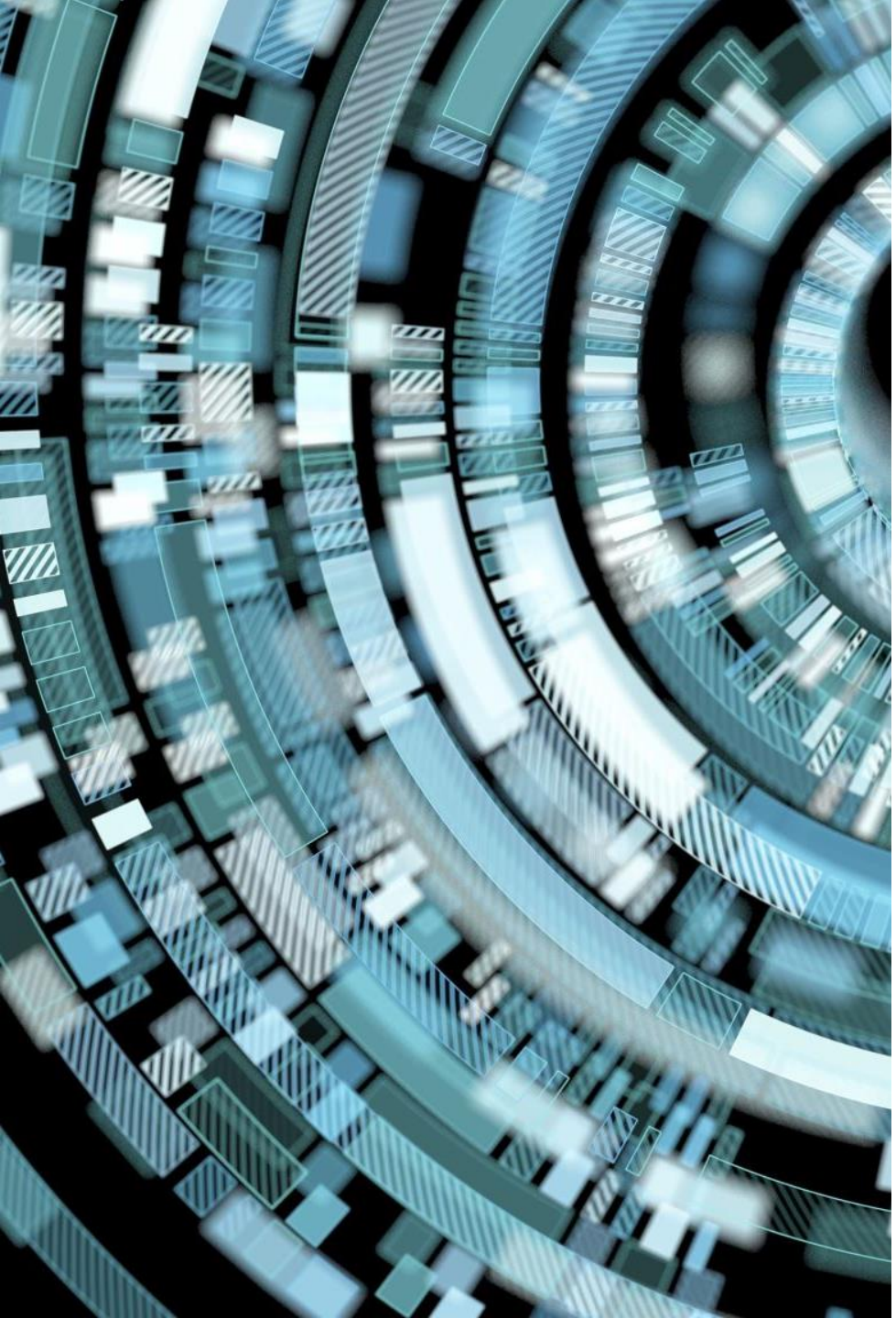
Computer System Design & Application

计算机系统设计与应用A

陶伊达 （TAO Yida）
taoyd@sustech.edu.cn



Lecture 8

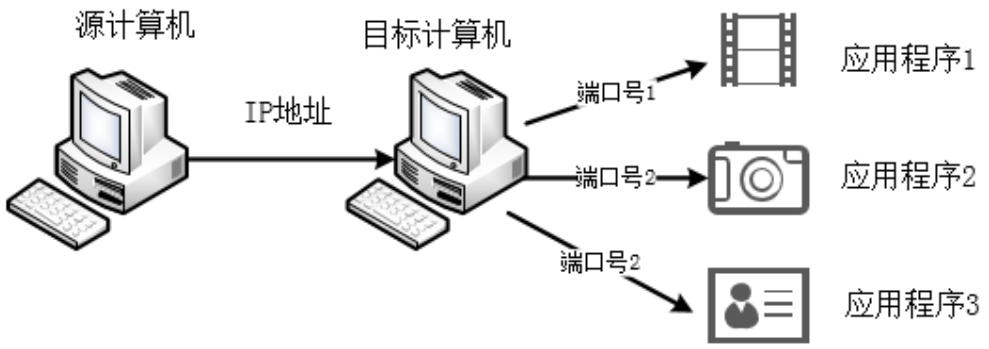
- Socket Programming
- Getting Web Data

Networking

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

TAO Yida@SUSTECH

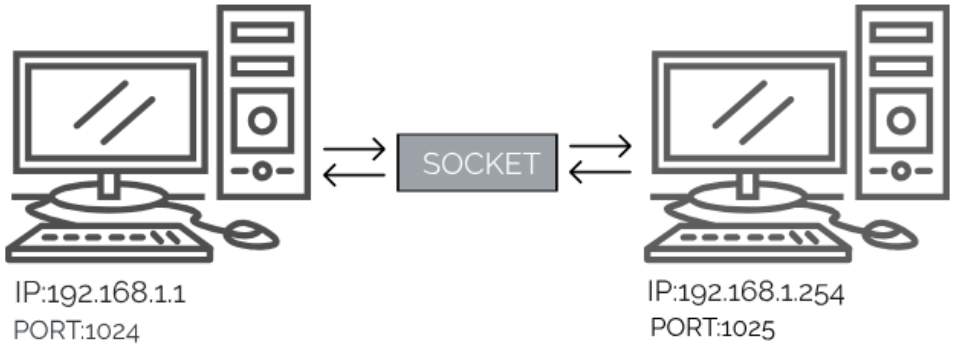
Networking Terminology



- **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

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.



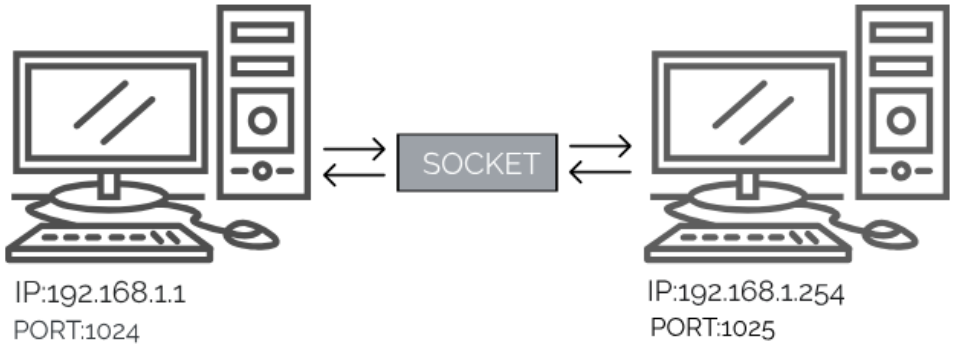
https://examradar.com/java-networking-network-basics-socket-overview/

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.

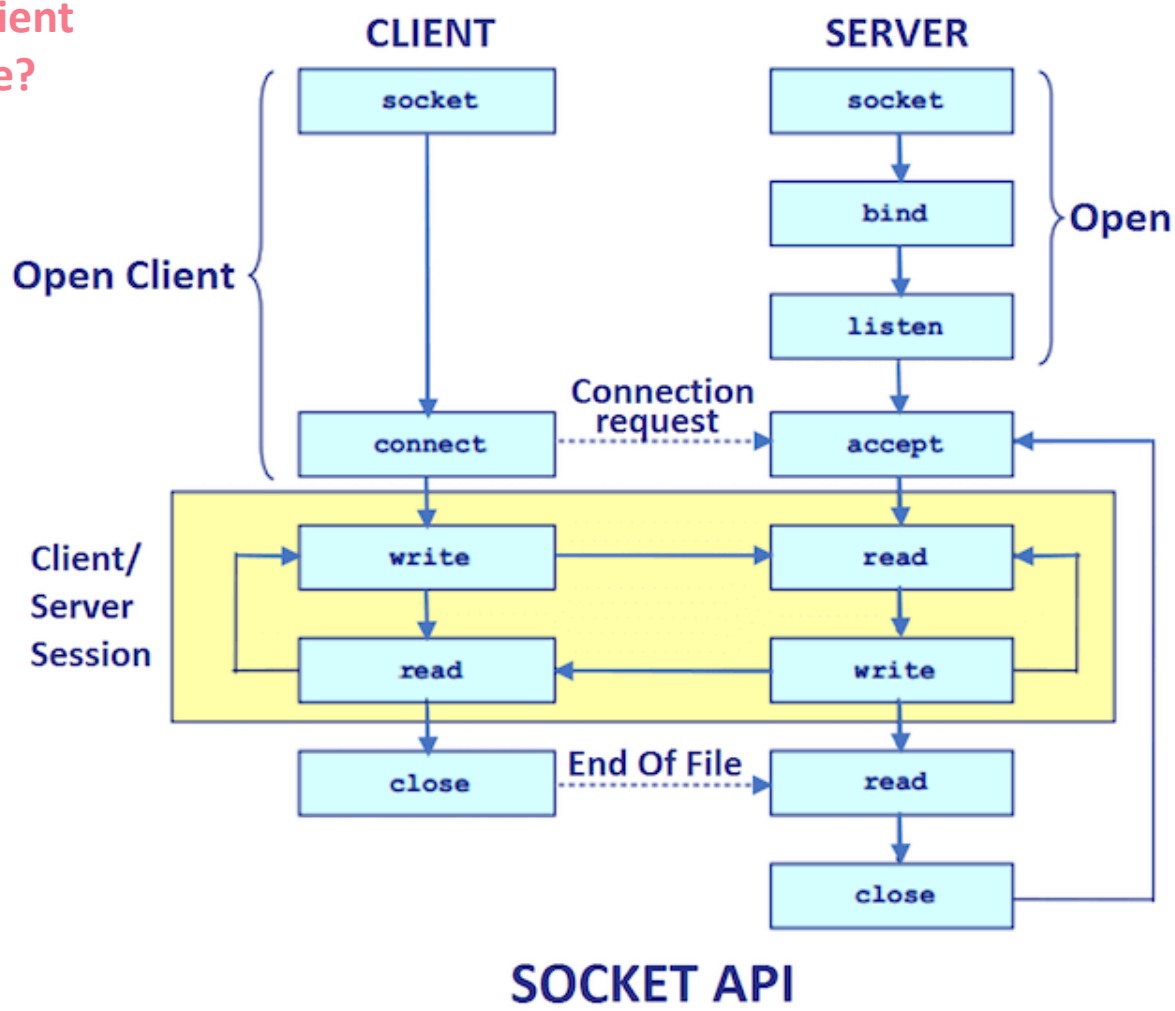


https://examradar.com/java-networking-network-basics-socket-overview/

```
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?

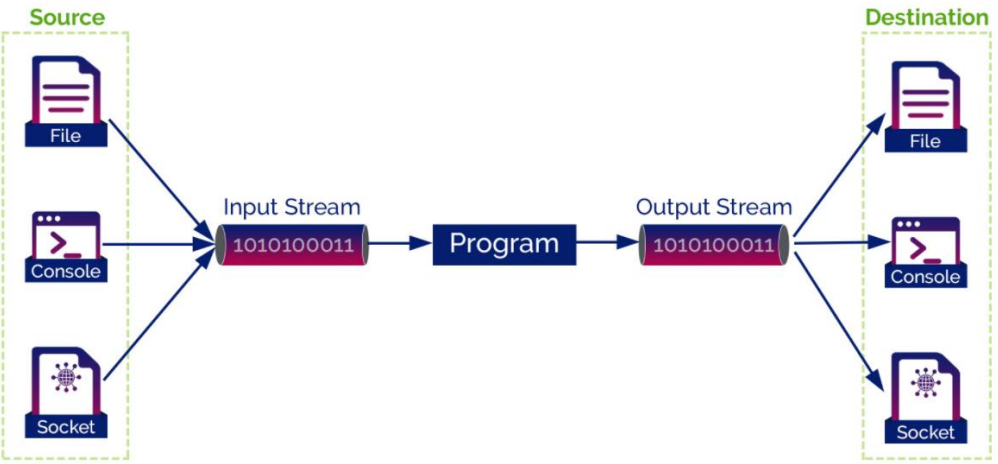


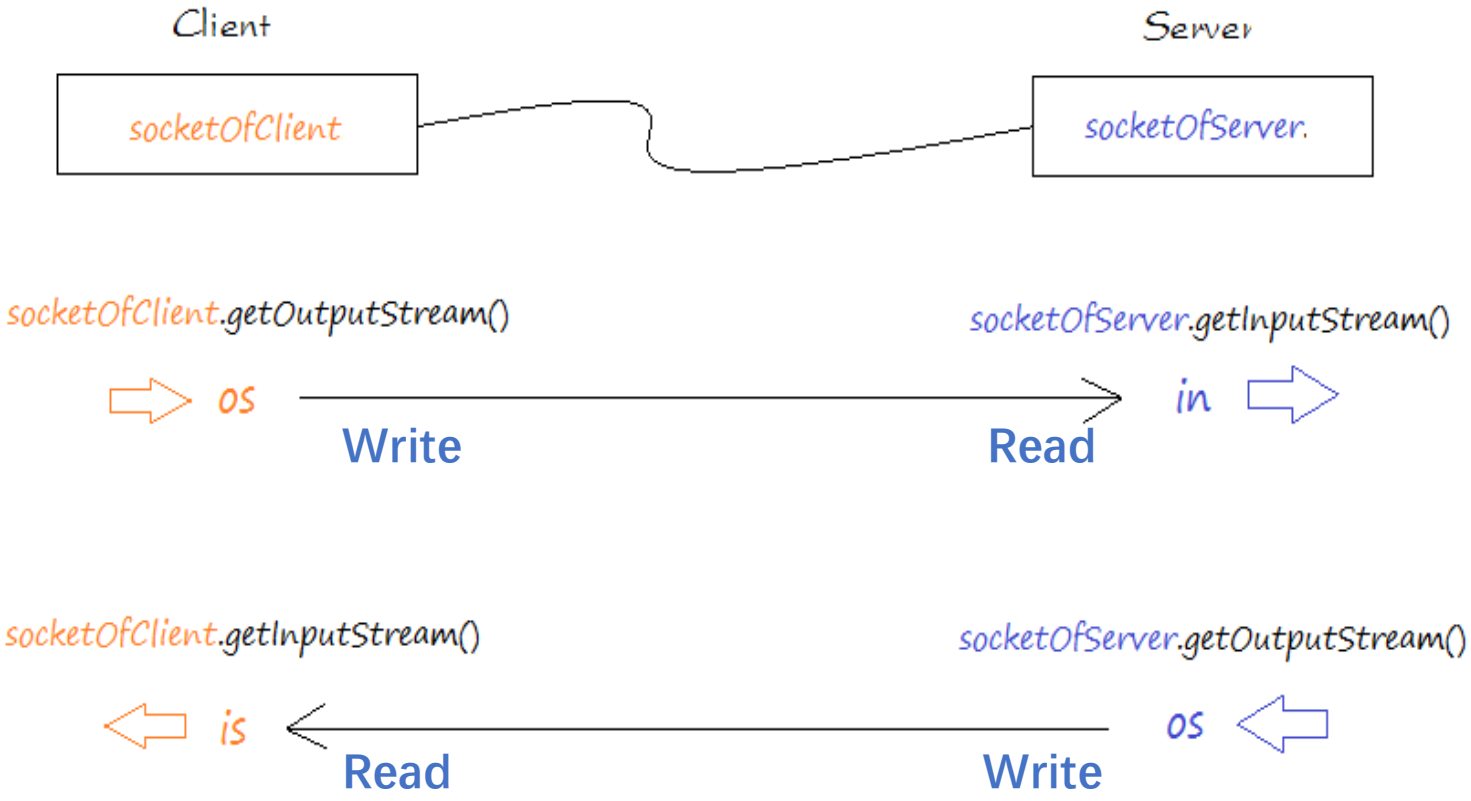
- Create an instance of Socket by passing the IP or hostname of the server and a port number
- If the connection fails, an Exception is thrown
- Otherwise, establish the connection and use Socket to read and write.

- 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.

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





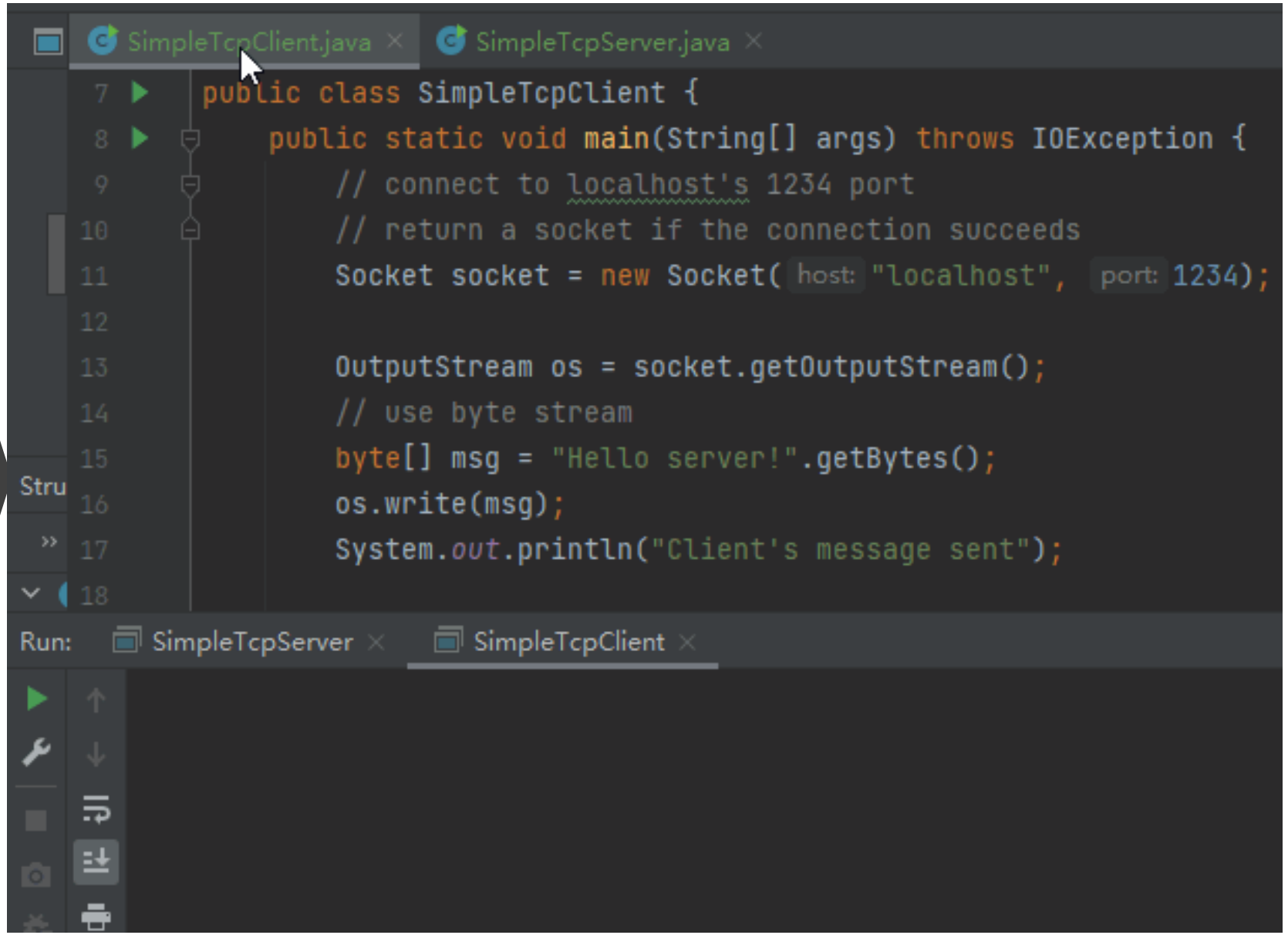
A Toy Example: Client

```
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("localhost", 1234);

        // Get OutputStream
        // and write messages as bytes
        OutputStream os = socket.getOutputStream();
        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();
    }
}
```

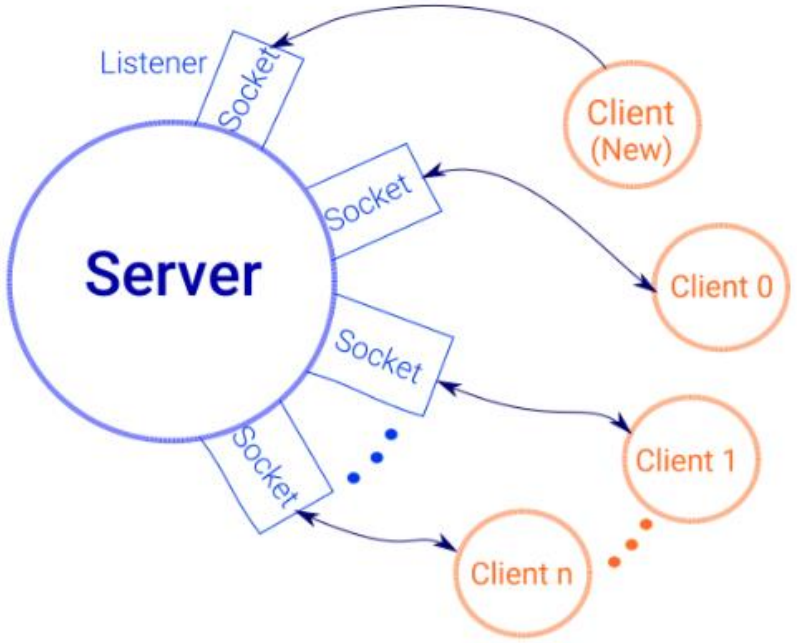
A Toy Example: Server

```java
public class SimpleTcpServer {
    public static void main(String[] args) throws IOException {
        // Listen to port 1234
        ServerSocket serverSocket = new ServerSocket(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, 0, readLen));
        }

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

A Toy Example



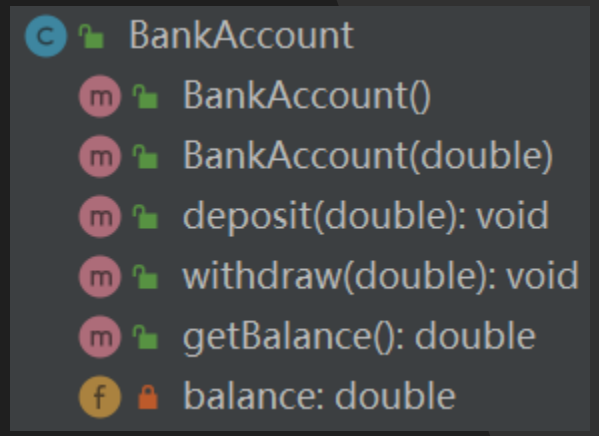
Why “Toy” Examples?



- 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: use multi-threads on server side: whenever a client request comes, a separate thread is assigned for handling each request

Case Study: Banking Service

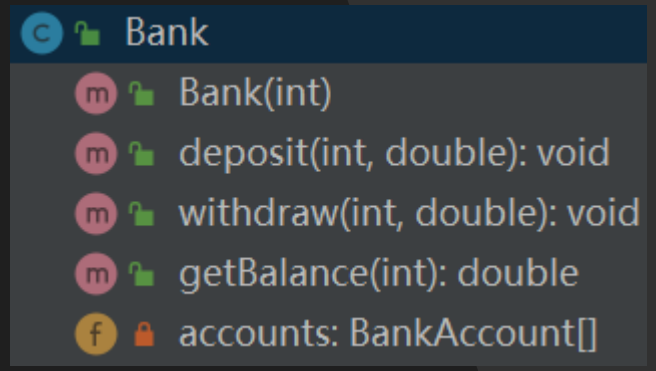


- A bank account has a balance that can be changed by deposits and withdrawals.

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

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

deposit() and withdraw() are properly synchronized

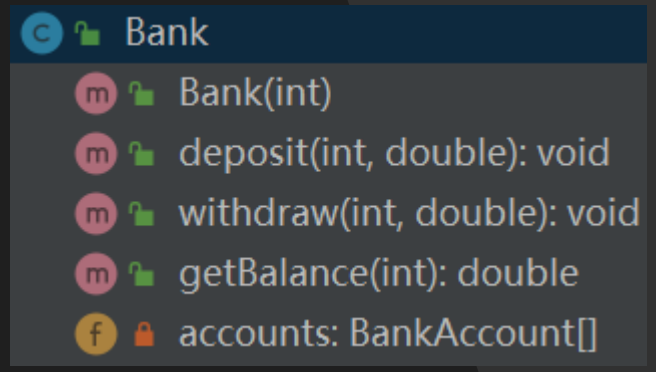
Case Study: Banking Service



A bank has multiple bank accounts

```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



A bank can withdraw from or deposit to a specific account

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

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

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

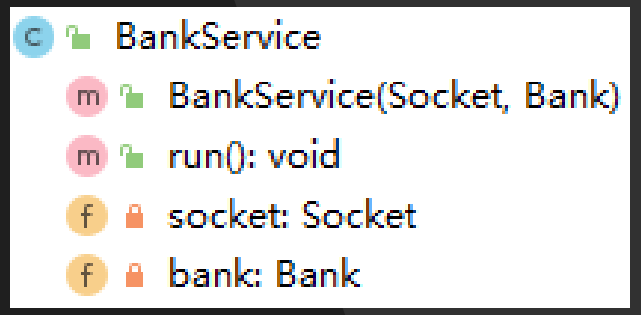
Banking Service Protocol

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 client/server application, you need to specify some application-level protocol that clients can use to interact with the server

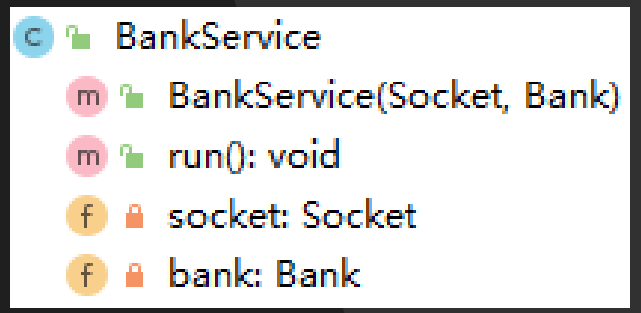
Case Study: Banking Service



A bank service executes the banking service protocol

```
public class BankService implements Runnable {
    private Socket socket;
    private Bank bank;

    public BankService (Socket socket, Bank bank) {
        this.socket = socket;
        this.bank = bank;
    }
```

Case Study: Banking Service




```
public void run() {
      Scanner in = new Scanner( socket.getInputStream());
      PrintWriter out = new PrintWriter( socket.getOutputStream());

      while (true) {
        if (!in.hasNext()) return;
        String command = in.next();
        if ("QUIT".equals(command)) return;

        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 Server

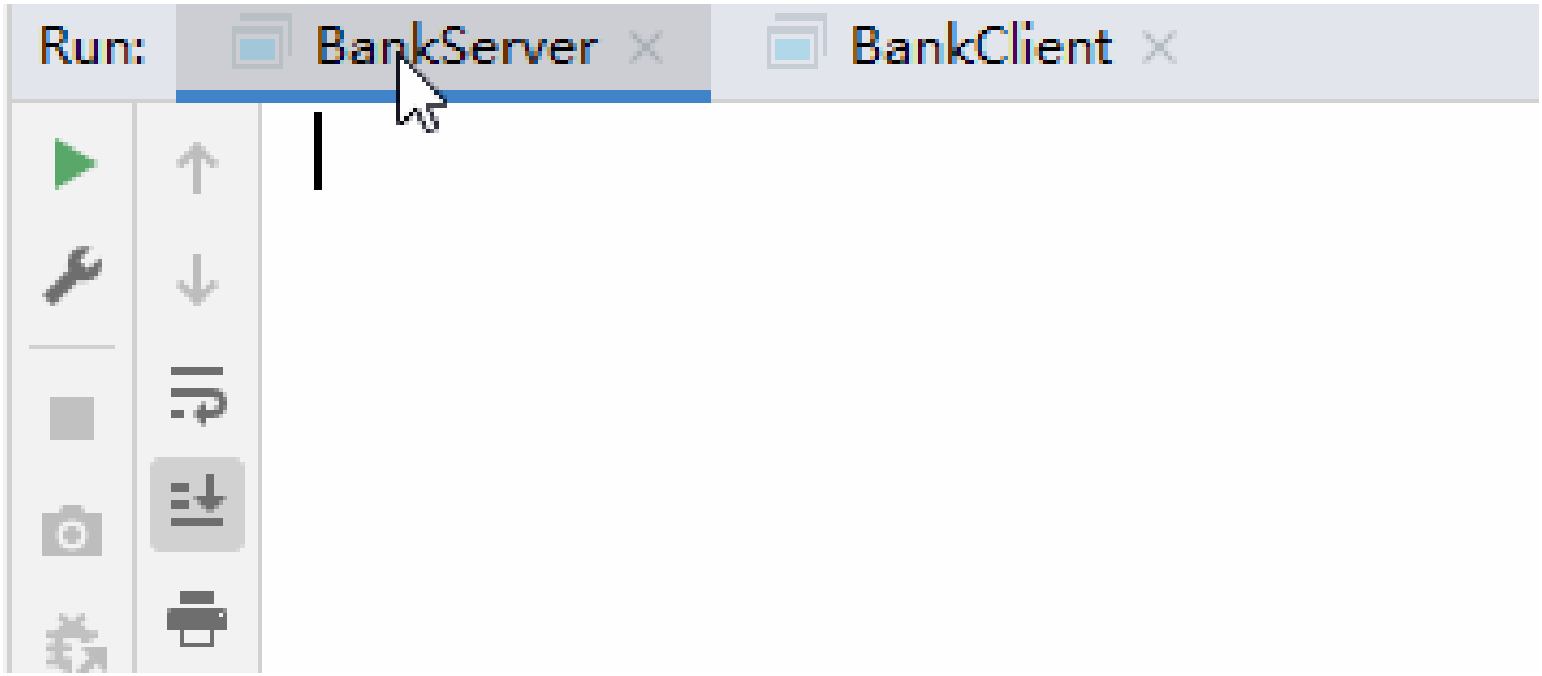
```
public class BankServer {
   public static void main (String[] args) throws IOException  {
      // initialized bank with 10 accounts
      Bank bank = new Bank(10);

      ServerSocket server = new ServerSocket(8888);

      System.out.println( "Waiting for clients to connect..." );
      // server keeps running
      while (true) {
         Socket socket = server.accept();
         System.out.println( "Client connected." );

         // start a thread for providing service to the client
         BankService service = new BankService(socket, bank);
         Thread t = new Thread(service);
         t.start();
      }
   }
}
```

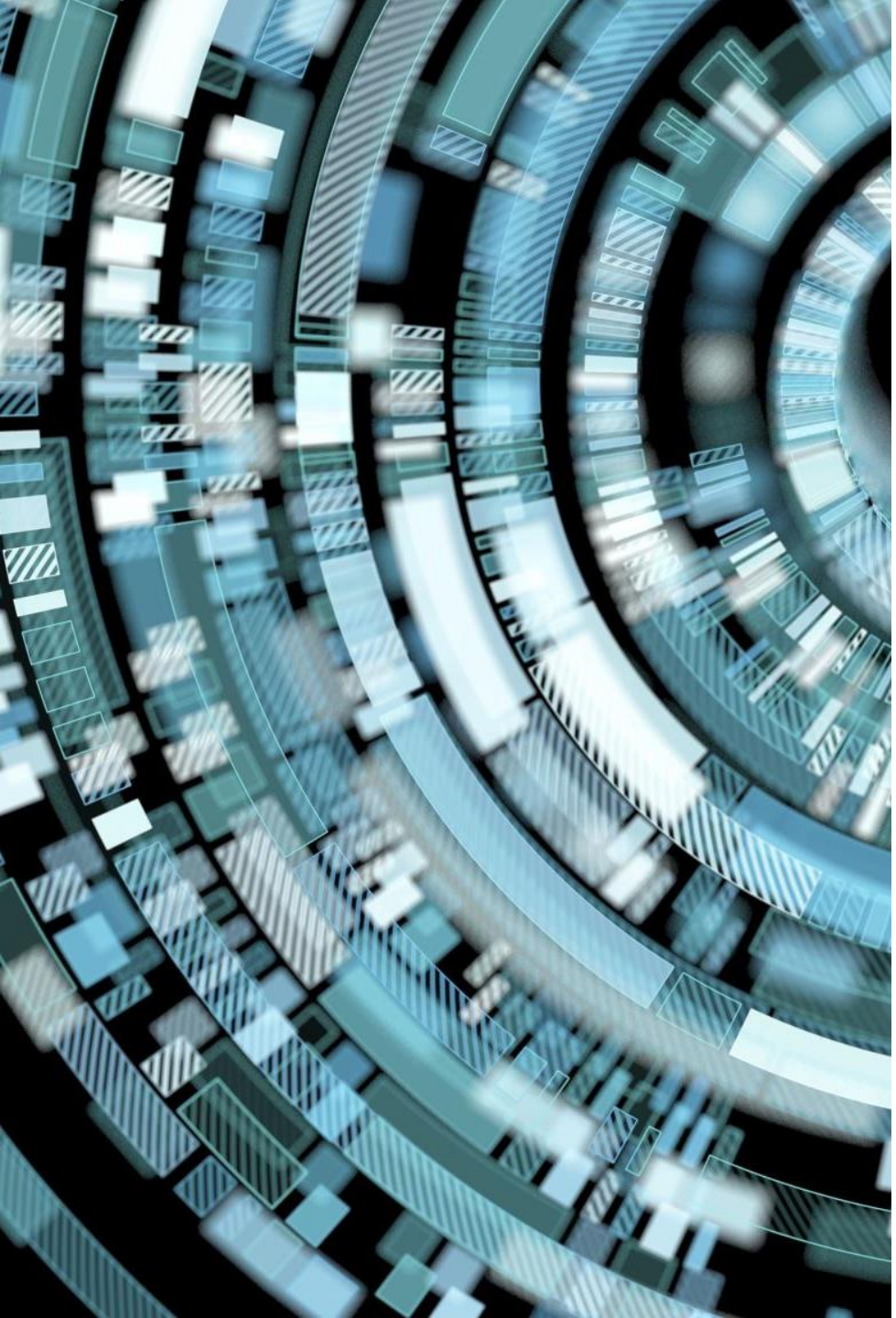
Bank Client

- To communicate with the server by sending and receiving text, you could turn the streams into scanners and writers
- Use `println` instead of `print` to mark the end of input
- The flush method empties the buffer and forwards all waiting characters to the destination.

```
public class BankClient {
   public static void main (String[] args) throws IOException {

      Socket s = new Socket("localhost", 8888);

      Scanner in = new Scanner(s.getInputStream());
      PrintWriter out = new PrintWriter(s.getOutputStream());

      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 = "QUIT";
      out.println(command);
      out.flush();

      s.close();
   }
}
```

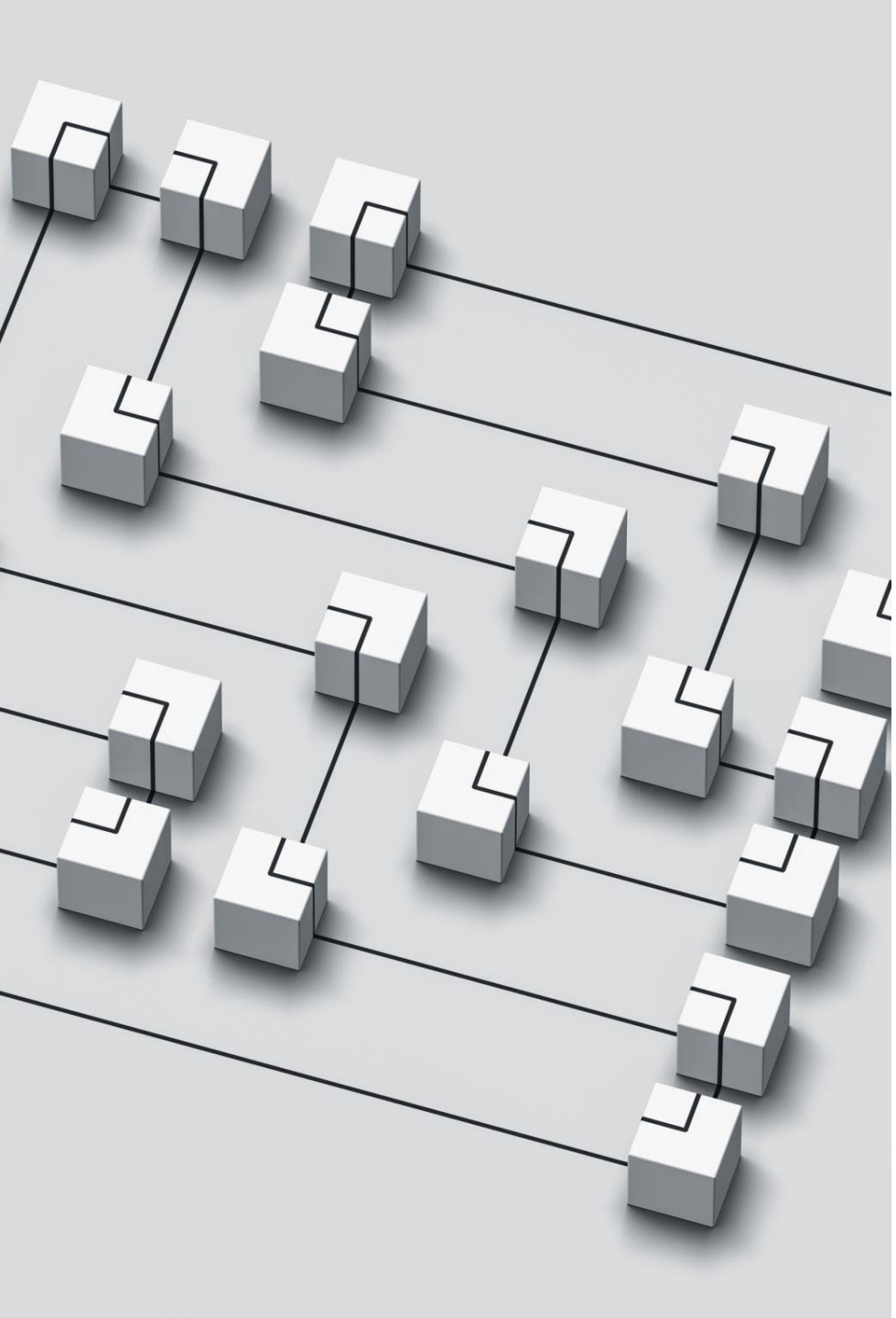
Execution





Lecture 8

- Socket Programming
- Getting Web Data
  - java.net package
  - REST API



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

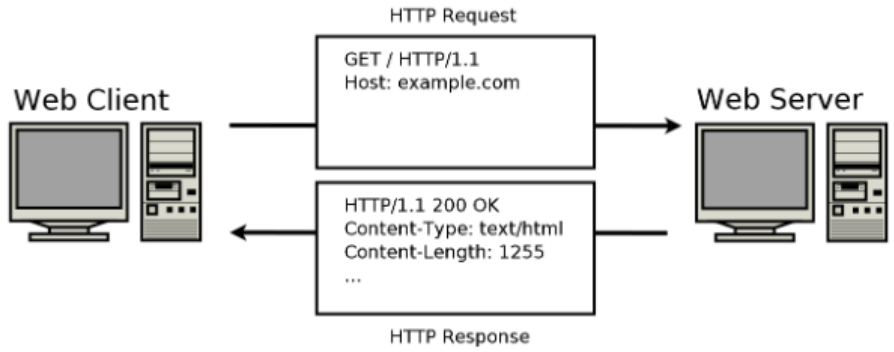
- Each Internet application has a different application protocol, which describes how the data for that particular application are transmitted.
- A port number helps a computer decide which application should receive an incoming piece of data

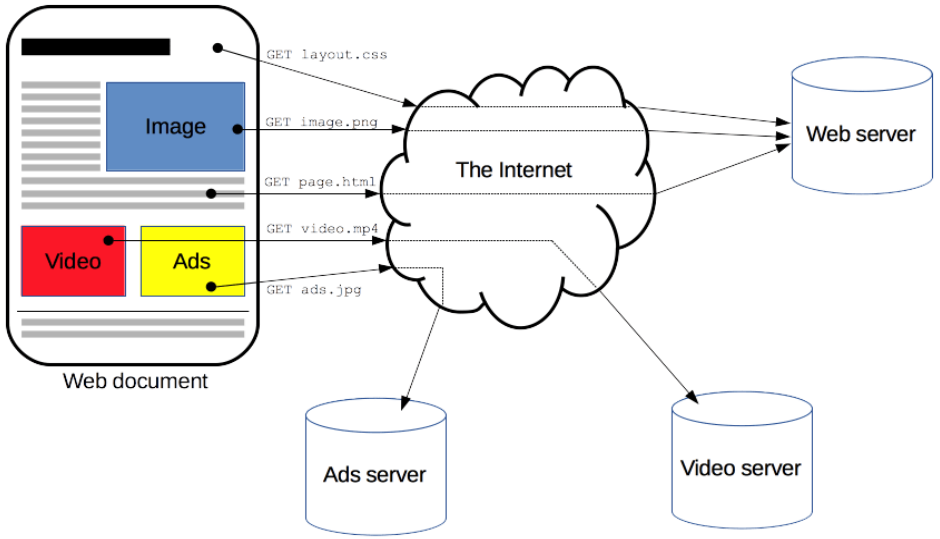
Well-known port numbers are reserved and we can no longer use them for other purposes

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





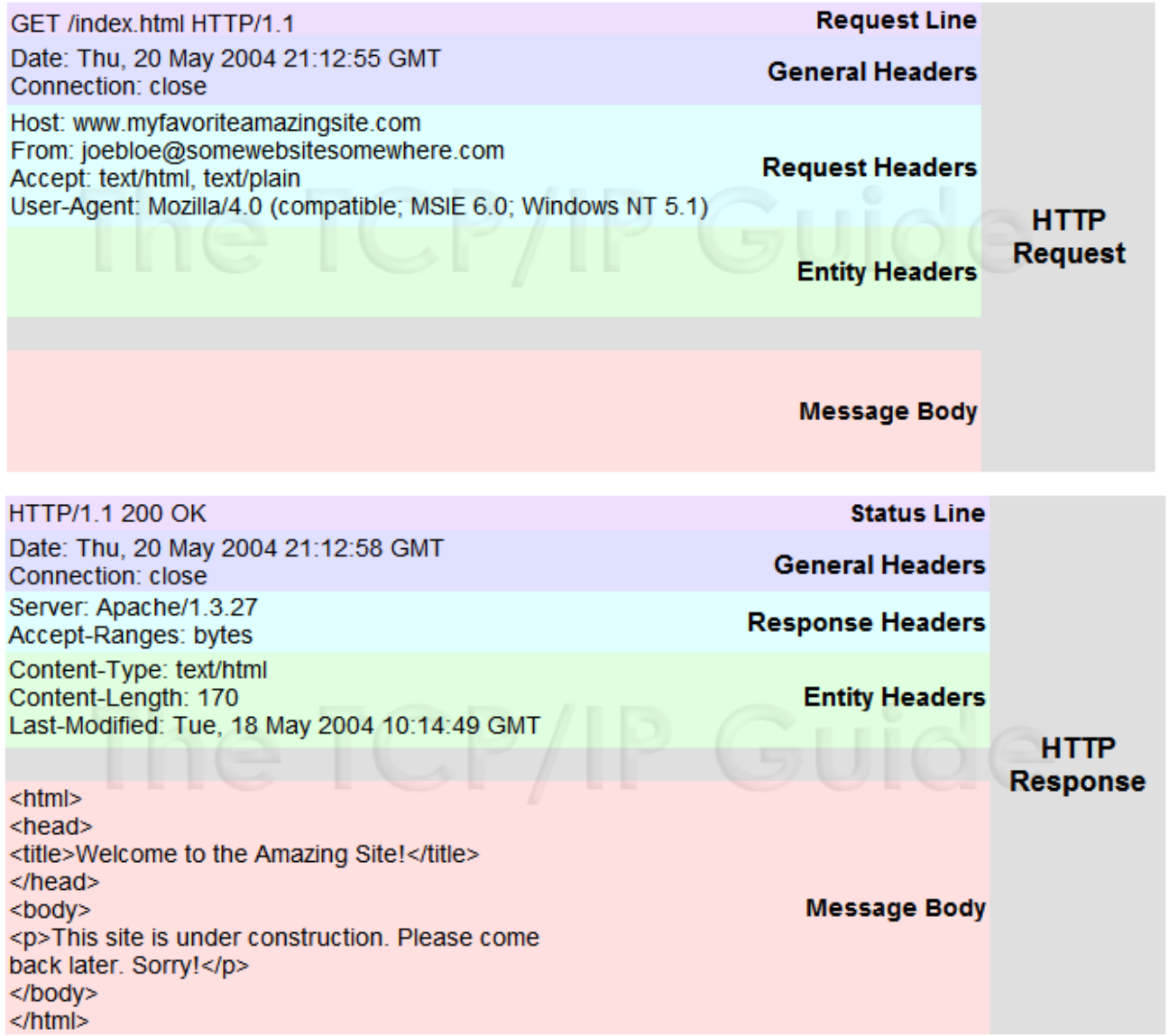
https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

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



HTTP Request/Response Message Format

Fetching a web page with socket

```
public static void readBySocket(String host) throws IOException {
    try(Socket s = new Socket(host, 80)){
        InputStream instream = s.getInputStream();
        OutputStream outstream = s.getOutputStream();

        // working with text
        Scanner in = new Scanner(instream);
        PrintWriter out = new PrintWriter(outstream);

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

        // Read server response
        while(in.hasNextLine()){
            String input = in.nextLine();
            System.out.println(input);
        }
    }
}
```

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

```
public static void readBySocket(String host) throws IOException {
    try(Socket s = new Socket(host, 80)){
        InputStream instream = s.getInputStream();
        OutputStream outstream = s.getOutputStream();

        // working with text
        Scanner in = new Scanner(instream);
        PrintWriter out = new PrintWriter(outstream);

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

        // Read server response
        while(in.hasNextLine()){
            String input = in.nextLine();
            System.out.println(input);
        }
    }
}
```

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

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 6526
Content-Type: text/html; charset=utf-8
P3P: CP="NON UNI COM NAV STA LOC CURa DEVa PSAa PSDa OUR IND"
Set-Cookie: MUID=342CEEBFAABE64703824FD7BABD865EC; domain=.bing.com; expires=Tue, 03-D
Set-Cookie: MUIDB=342CEEBFAABE64703824FD7BABD865EC; expires=Tue, 03-Dec-2024 02:55:50
Set-Cookie: _EDGE_S=F=1&SID=1DA351BD365D6B6B0B634279373B6A27; domain=.bing.com; path=/
Set-Cookie: _EDGE_V=1; domain=.bing.com; expires=Tue, 03-Dec-2024 02:55:50 GMT; path=/
Set-Cookie: SRCHD=AF=NOFORM; domain=.bing.com; expires=Sun, 09-Nov-2025 02:55:50 GMT;
Set-Cookie: SRCHUID=V=2&GUID=91210C057210470392FD49A1D90FDFDA&dmnchg=1; domain=.bing.c
Set-Cookie: SRCHUSR=DOB=20231109; domain=.bing.com; expires=Sun, 09-Nov-2025 02:55:50
Set-Cookie: SRCHHPGUSR=SRCHLANG=zh-Hans; domain=.bing.com; expires=Sun, 09-Nov-2025 02
Set-Cookie: _SS=SID=1DA351BD365D6B6B0B634279373B6A27; domain=.bing.com; path=/
Set-Cookie: ULC=; domain=.bing.com; expires=Wed, 08-Nov-2023 02:55:50 GMT; path=/
Set-Cookie: _HPVN=CS=eyJQbiI6eyJDbiI6MSwiU3QiOjAsIlFzIjowLCJQcm9kIjoiUCJ9LCJTYyI6eyJDb
X-EventID: 654c4a3623a744babae08b1c6e40a52d
UserAgentReductionOptOut: A7kgTC5xdZ2WIVGZEfb1hUoNuvjzOZX3VIV/BA6C18kQOOF50Q0D3oWoAm49
X-Cache: CONFIG_NOCACHE
X-MSEdge-Ref: Ref A: BE44B4ADF98A45F49C530032339CFD7E Ref B: BJ1EDGE1116 Ref C: 2023-1
Date: Thu, 09 Nov 2023 02:55:49 GMT

<!doctype html><html lang="zh" dir="ltr"><head><meta name="theme-color" content="#4F4F
                    placeholder="" type="search" maxlength="1000" autocapitalize="off"
                    title="输入搜索词" autofocus="autofocus" ) /><input id="sb_form_go" t
    title="搜索" name="search" value=""
```

Fetching a web page with socket

```
public static void readBySocket(String host) throws IOException {
    try(Socket s = new Socket(host, 80)){
        InputStream instream = s.getInputStream();
        OutputStream outstream = s.getOutputStream();

        // working with text
        Scanner in = new Scanner(instream);
        PrintWriter out = new PrintWriter(outstream);

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

        // Read server response
        while(in.hasNextLine()){
            String input = in.nextLine();
            System.out.println(input);
        }
    }
}
```

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

```
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://cn.bing.com/";

```
200  OK
<!doctype html><html lang="zh" dir="ltr"><head><meta name="theme-color
 style="position:relative;vertical-align:top;margin-right:-16px;right:
微软必应手机版</span><span class="id_qrcode_subtitle">全球资源，有求必应</sp
var preloadBg = document.getElementById('preloadBg'); if (preloadBg) {
//]]></script><script type="text/javascript" crossorigin="anonymous" s
0;function getBrowserWidth_Desk(){var t=_d.documentElement,n=Math.roun
//]]></script><script type="text/javascript" crossorigin="anonymous" s
sa_config={"f":"sb_form","i":"sb_form_q","c":"sw_as","zisParent":"sa_z
//]]></script><div id="aRmsDefer"><script type="text/rms">//<![CDATA[
var mcp_banner=function(n){function u(n){var t=sj_gx(),i,r;if(t.open("
//]]></script><script type="text/rms">//<![CDATA[
0;
//]]></script><script type="text/rms">//<![CDATA[
```

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)

```
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.

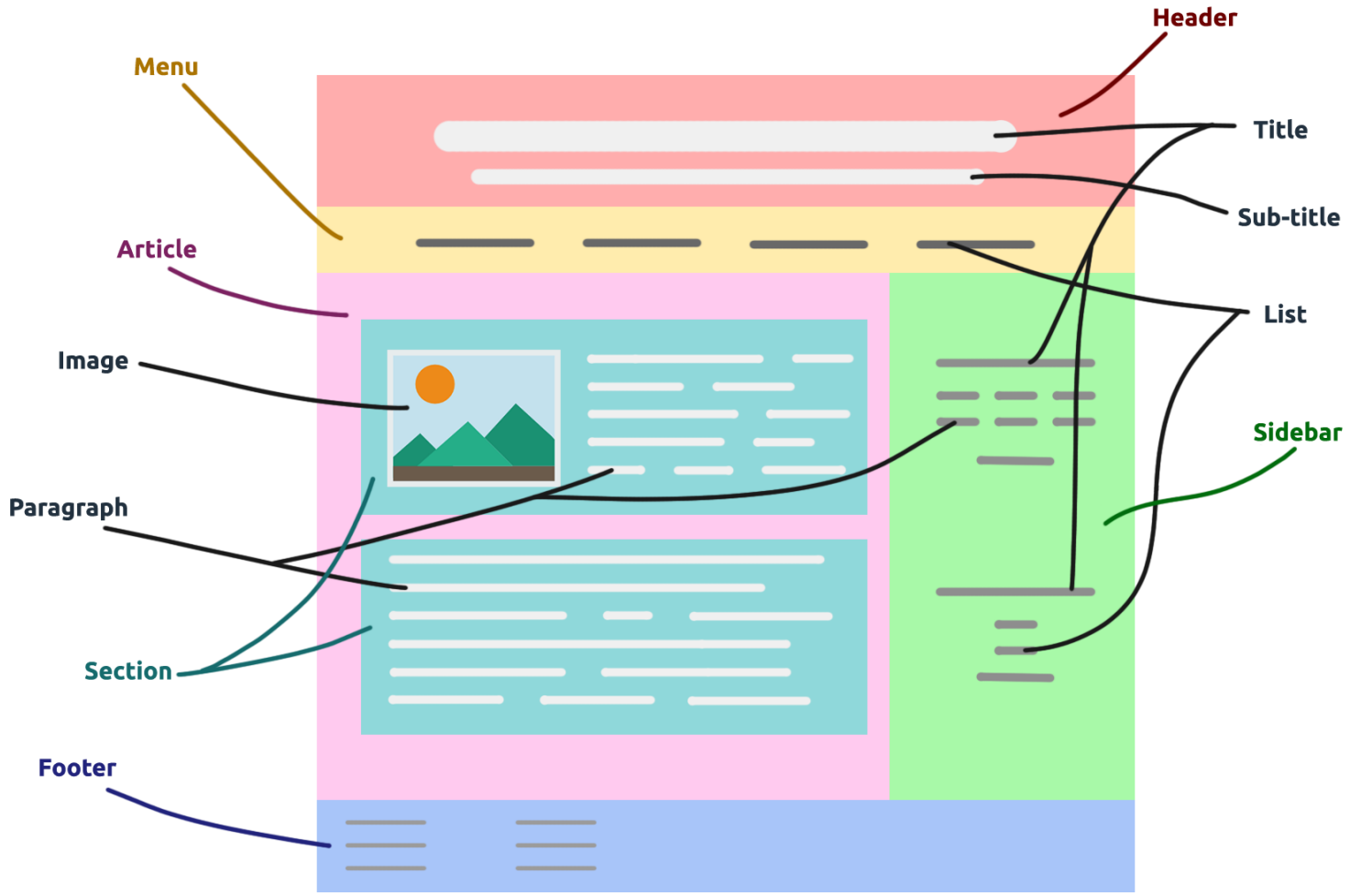
The java.net package can be roughly divided in two sections:

- *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*.

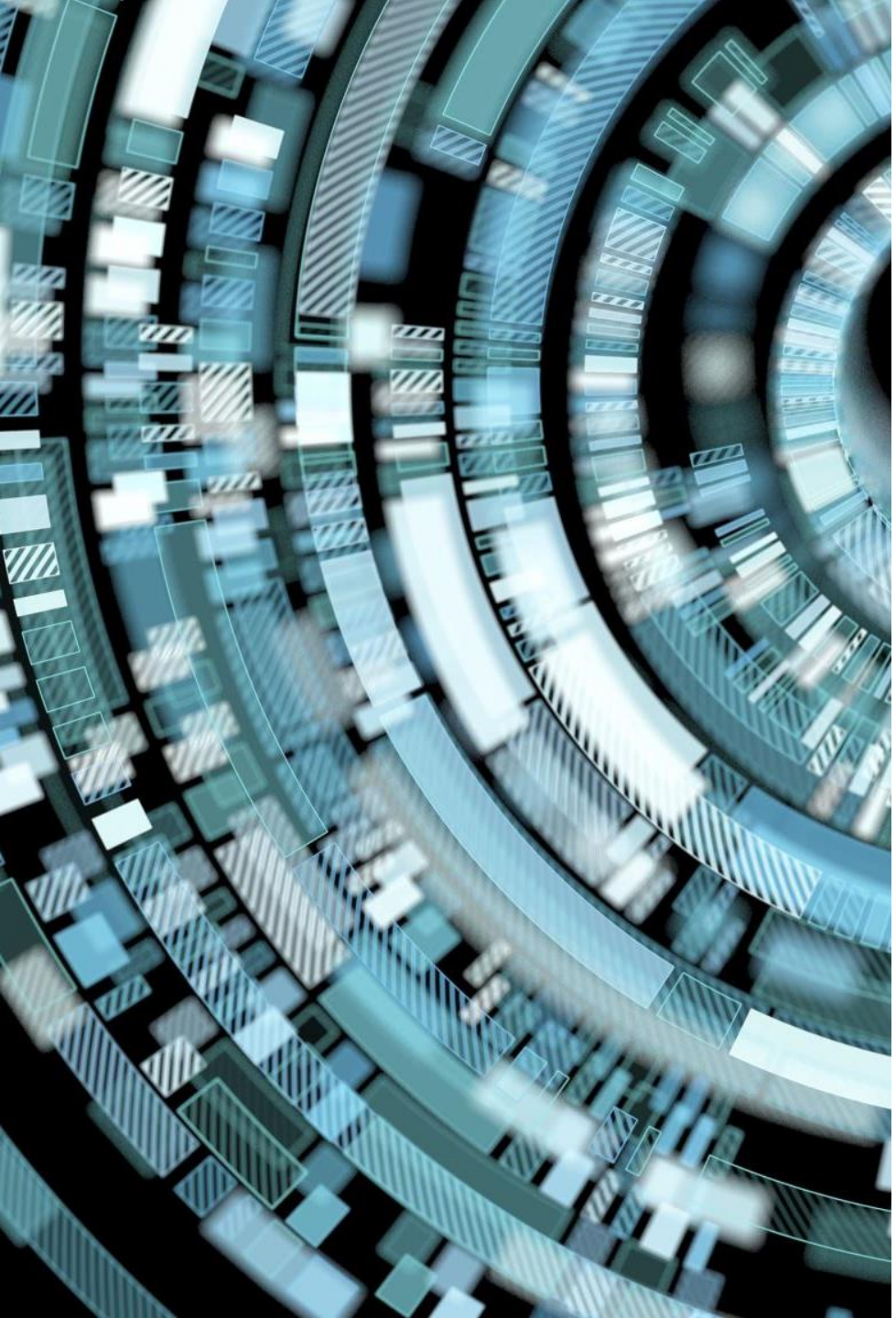
https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html#package_description

Collecting Web Data

- Web data in HTML format can be inconsistent or complex, as it is designed for human viewers, not machine processing.
- If the website structure changes, our socket application or crawlers may break, requiring frequent maintenance.



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



Lecture 8

- Socket Programming
- Getting Web Data
  - java.net package
  - 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`)
- **RESTful API**
  - A REST APIs is an API conforms to the constraints of REST architectural style
  - RESTful APIs are widely used in industry for communicating between clients and servers
- **REST**
  - **RE**presentational **S**tate **T**ransfer
  - REST is a software 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

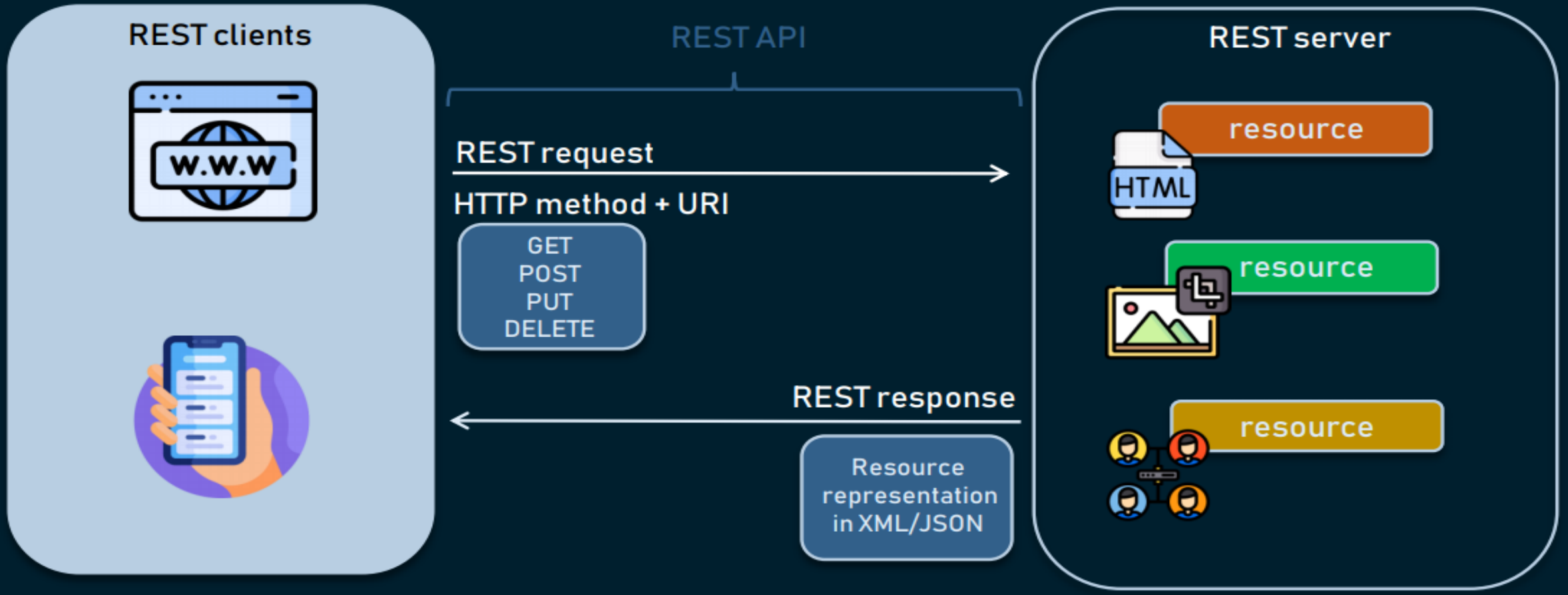


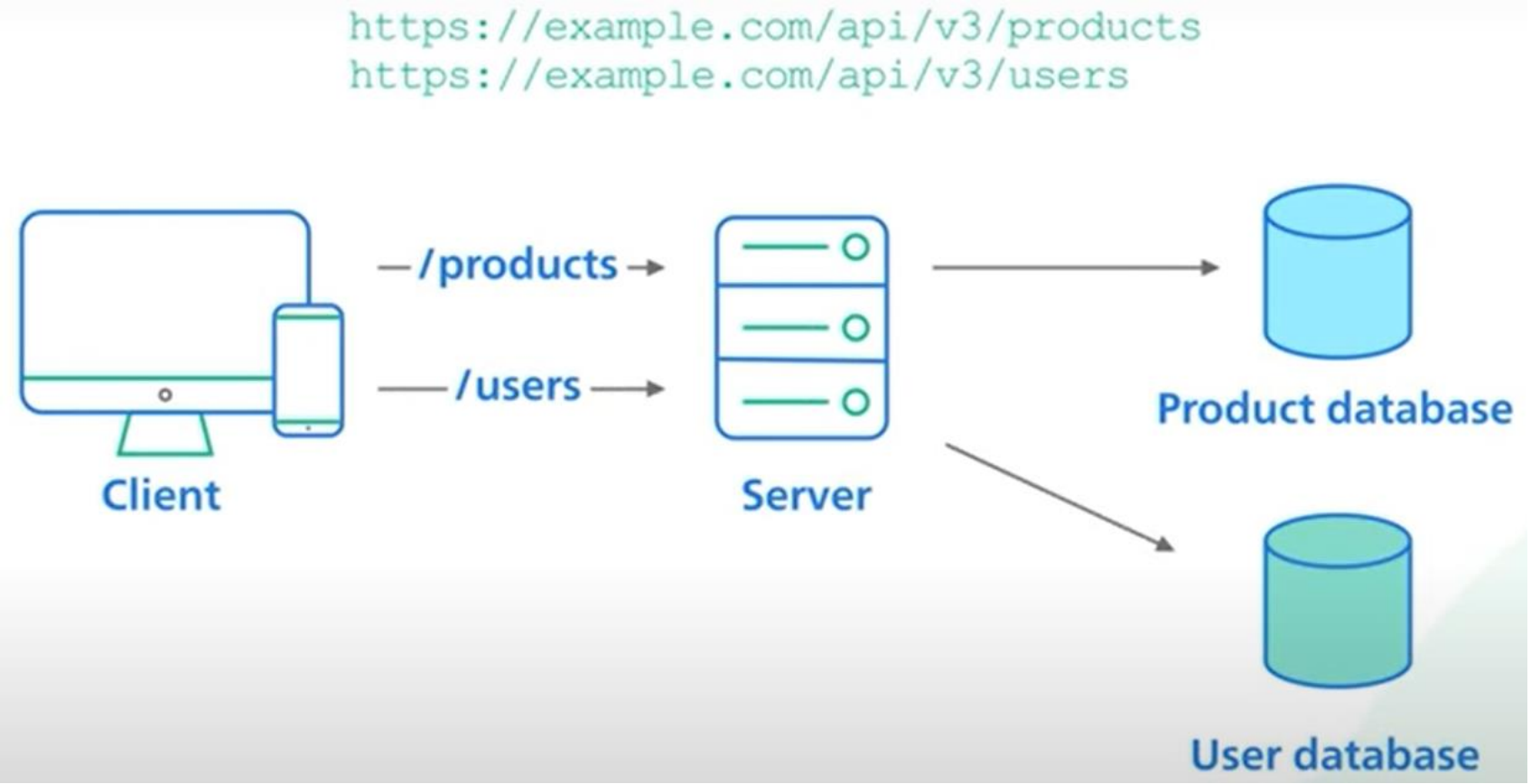
Image source: https://www.altexsoft.com/blog/rest-api-design/

RESTful API Request Design

Request = Verb + Object

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

RESTful API Request Design



Using **nouns** to differentiate different resources

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
https://docs.github.com/en/rest/repos/repos#get-a-repository

`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}`

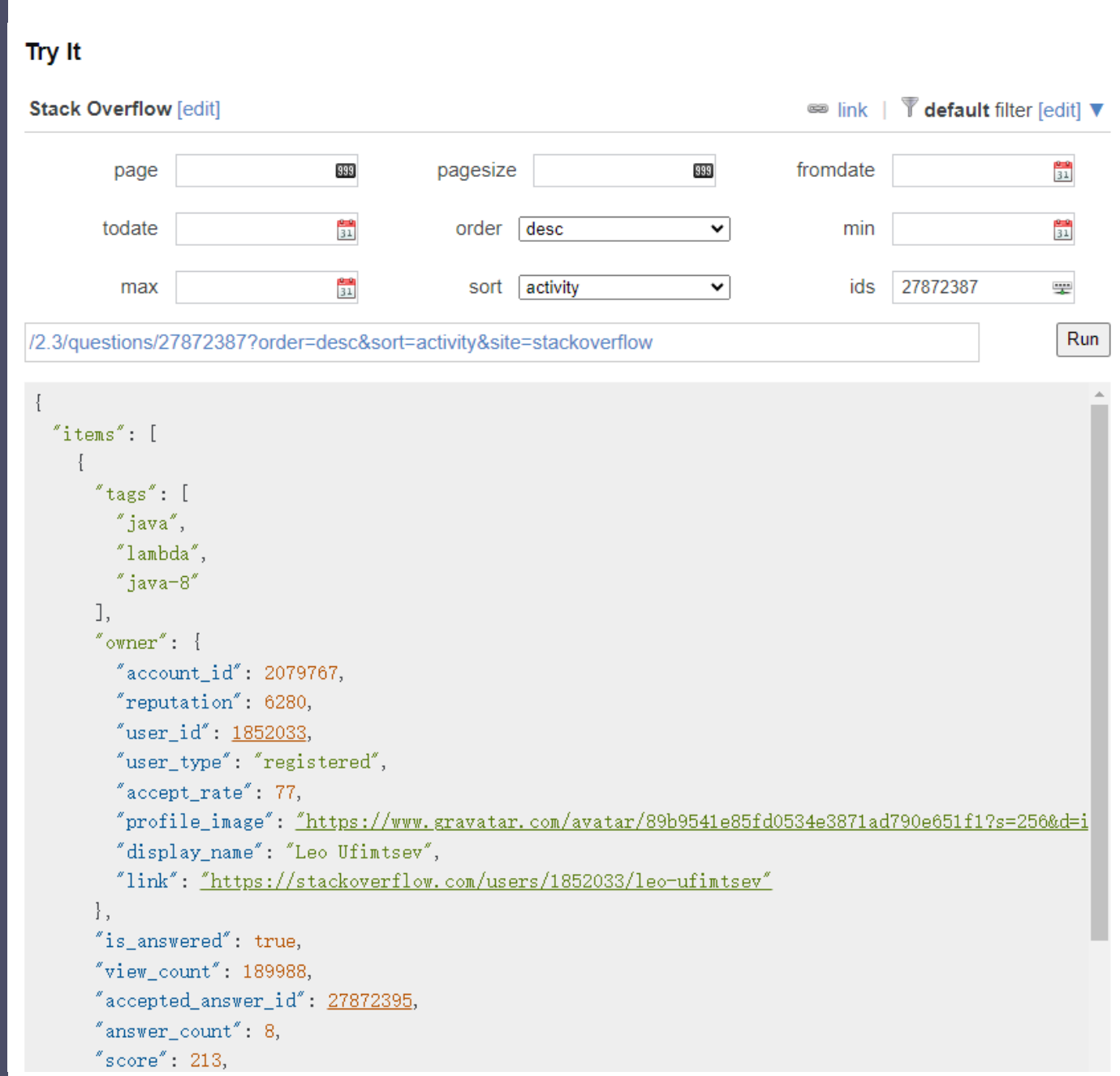
Update a release (must have push access to this repo)

`GET` `/search/topics`

Search for topics (should specify the topic using parameters)

Stack Overflow REST API

https://api.stackexchange.com/docs



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();
```

java.net package

Request verb

200
OK
gzip

Request Response

HTTP Status Code

`GET` `/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}",
```

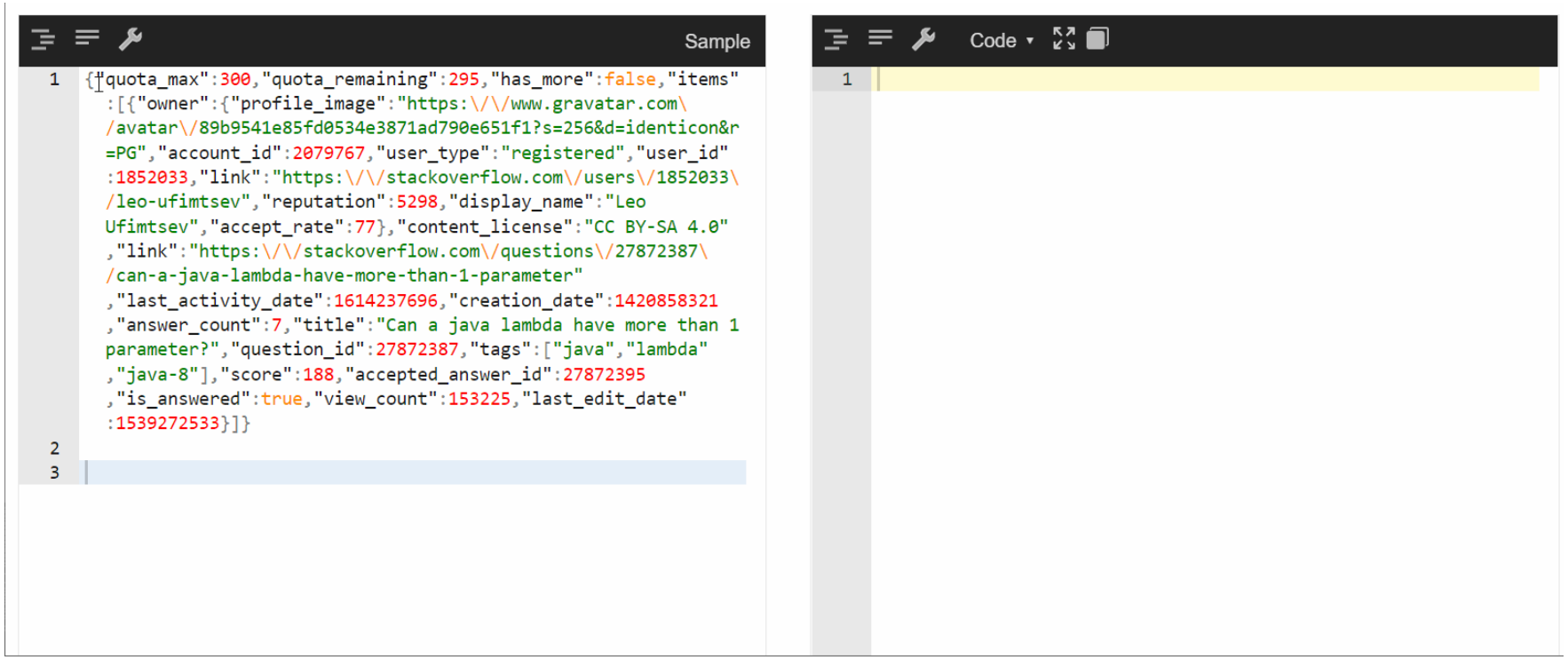

JSON format

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)



Type https://api.github.com/repos/spring-projects/spring-boot in your browser and see what happens

Next Lecture

- GUI Intro
- JavaFX