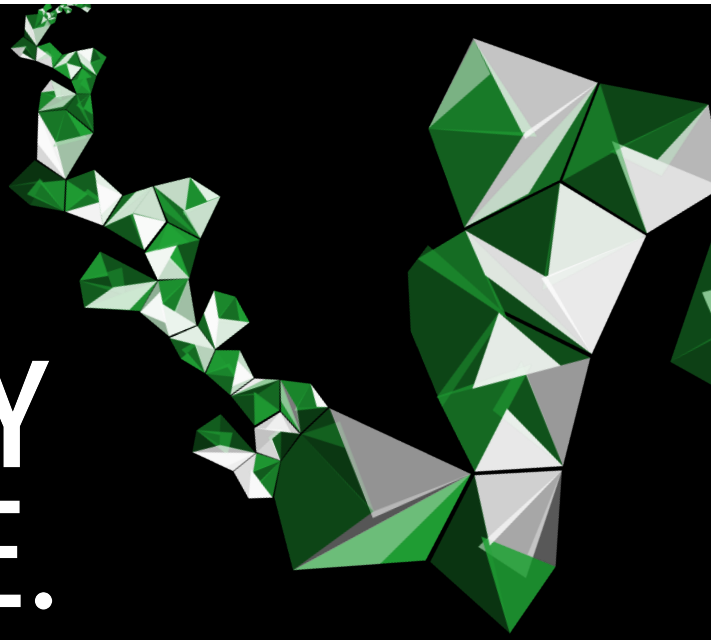
FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS AND COMPUTER SCIENCE (EEMCS)



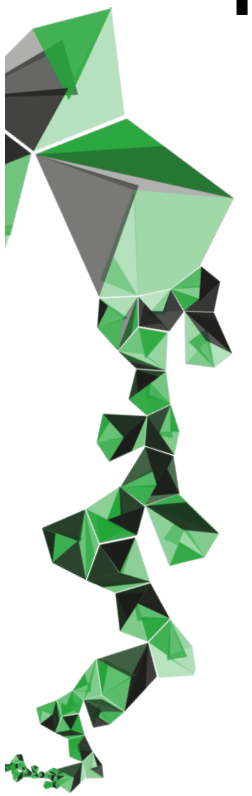
UNIVERSITY OF TWENTE.

P7.1: BASIC NETWORKING

PROGRAMMING (202001066) | MODULE 2: SOFTWARE DEVELOPMENT (202001064)

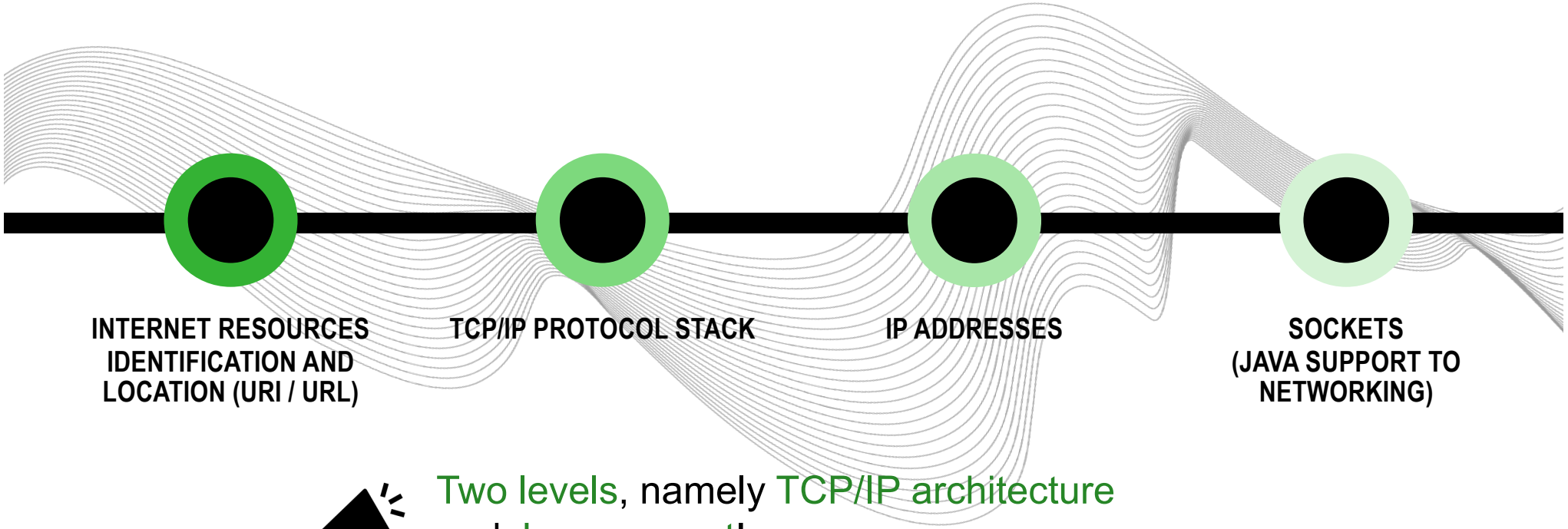
4 JANUARY 2021

PROGRAMMING LINE OVERVIEW



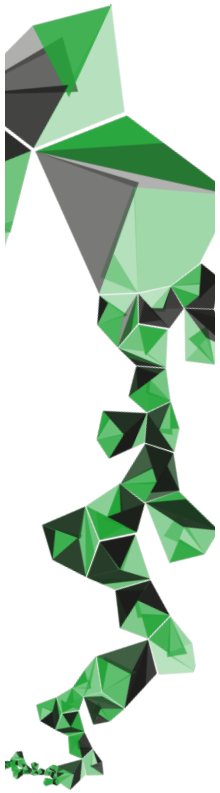
Week 1 Values and variables Control flow	**Week 2** Classes and objects Testing	**Week 3** Interfaces and Inheritance Subtyping Security 1
Week 4 Arrays and Lists List implementations Collections	**Week 5** Exceptions Stream I/O and MVC Security 2	**Week 6** Concurrency Project kick-off IDE Tips & Tricks
Week 7 **Basic Networking** Networking and Multithreading GUIs	**Week 8/9** Advanced Java facilities Test	**Week 10** Project Test resit

IN THIS PRESENTATION:



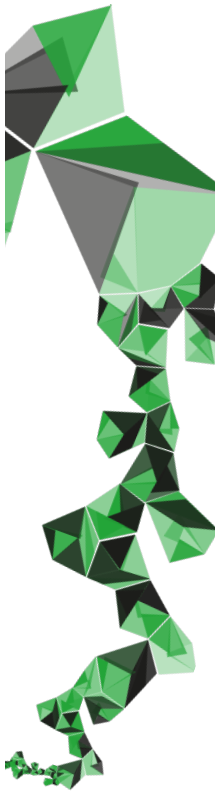
Two levels, namely TCP/IP architecture and Java support!

UNIVERSITY OF TWENTE.



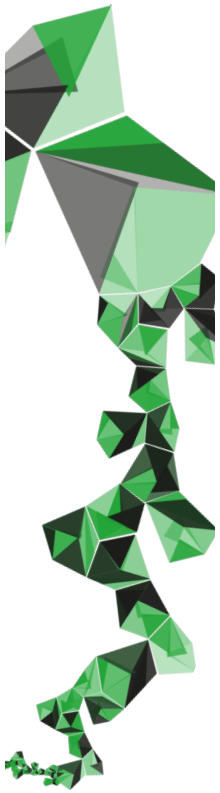
UNIFORM RESOURCE IDENTIFIERS AND LOCATORS

- URI: Uniform Resource Identifier
  - Standard syntactical construct that identifies a resource (file, book, person) unambiguously (see https://tools.ietf.org/html/rfc3986)
  - URN: Uniform Resource Name
    - URI defined to name a resource
  - URL: Uniform Resource Locator
    - Reference to a web resource that specifies its location on a computer network and a mechanism for retrieving this resource



IDENTIFIERS VERSUS LOCATORS
EXAMPLES

- Cars
  - Identifier: Nationality + number plate according to national system
  - Locator: Number plate + address of current owner
- Books
  - Identifier: ISBN
  - Locator: library catalogue number???
    An ISBN identifies a book type with many instances!



URI/URL SYNTAX
EXAMPLES

`mailto:l.ferreirapires@utwente.nl` (officially not a URL)

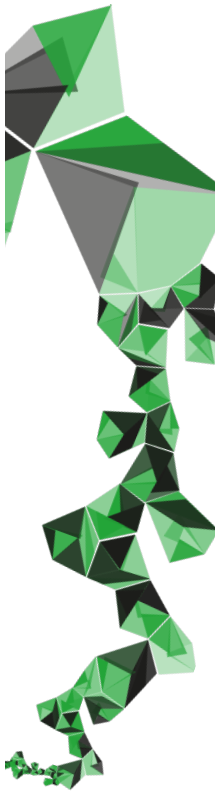
`http://www.utwente.nl/`

`file:///Macintosh%20HD/Java/Docs/api/java.net.InetAddress.html#_top_`

`http://www.macintouch.com:80/newsrecent.shtml`

`ftp://ftp.info.apple.com/pub/`

`http://mail-sendgrid.wikia.com/wf/click?upn=SGmz-2F-2FMHF1sUr6O3N1vB`

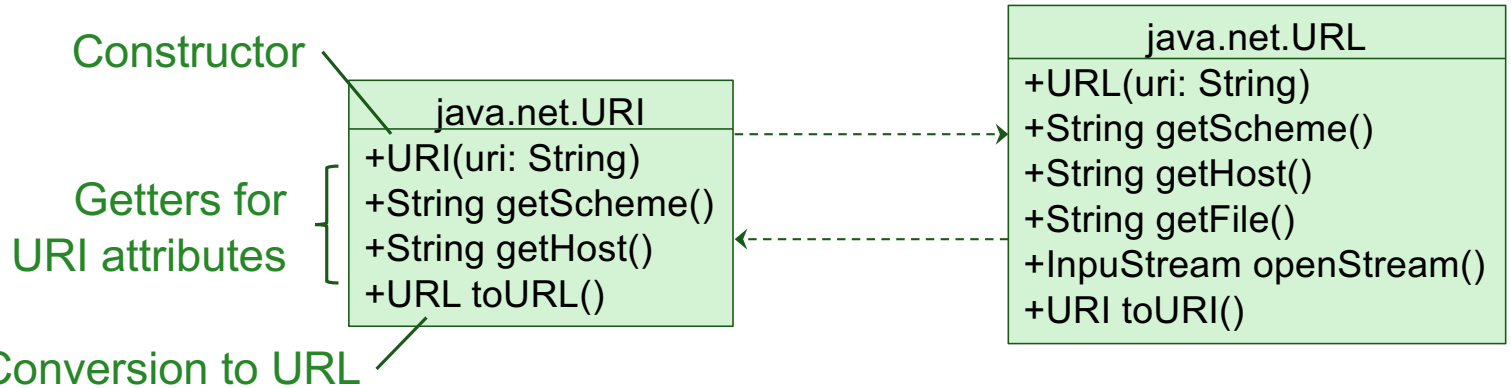


URI/URL SYNTAX
GENERIC SYNTAX

`scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]`

- `scheme`: defines kind of URI (`mailto`, `http`, `https`, `...`)
- `host`: host name (e.g., `utwente.nl`) or IPv4/IPv6 internet address (e.g., `192.168.0.1`)
- `port`: 16-bit number that points to a communication endpoint (`80`, `8080`)
- `path`: `/`-separated relative path on host (e.g., *`<directory>/<file>`*)
- `query`, `fragment`: used in Module 4

URI/URL JAVA SUPPORT



Constructors throw exceptions (respectively)

- `URISyntaxException`, subclass of `java.lang.Exception`
- `MalformedURLException`, subclass of `IOException`

EXAMPLE USAGE: READING A URL

```
public class WebCat {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            try (InputStream in = new URL(args[i]).openStream()) {
                BufferedReader br = new BufferedReader(
                        new InputStreamReader(in));
                String theLine;
                while ((theLine = br.readLine()) != null) {
                    System.out.println(theLine);
                }
            } catch (IOException e) {
                System.err.println(e);
            }
        }
    }
}
```

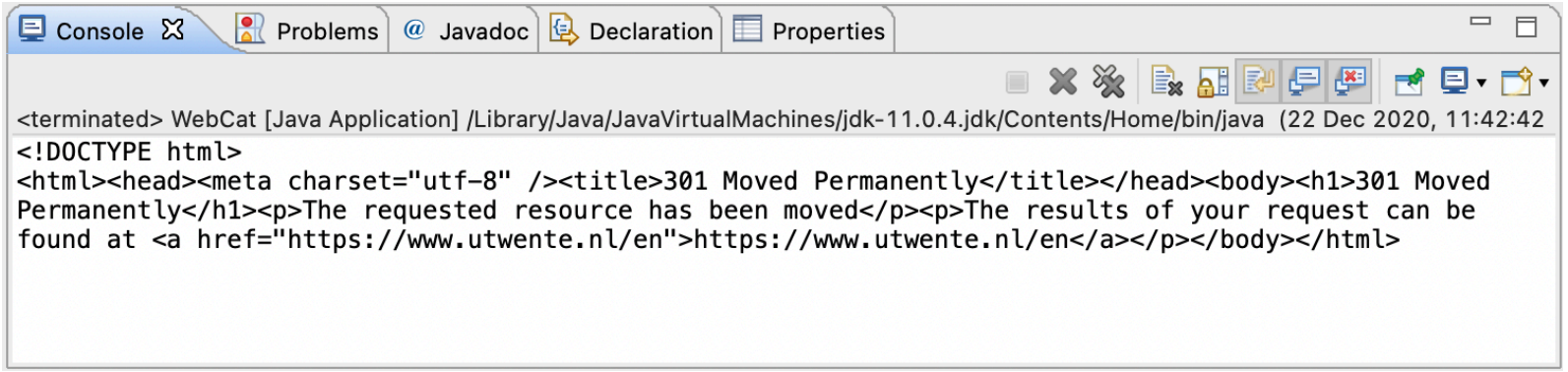
Create URL and query resource

Receive response

Handle I/O exceptions (including `MalformedURLException`)

RESULT

- `WebCat` call with `http://www.utwente.nl/en`

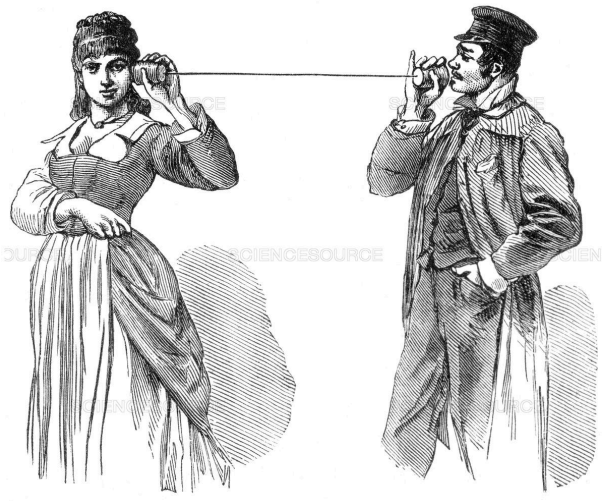


UNIVERSITY OF TWENTE.

NETWORK COMMUNICATION

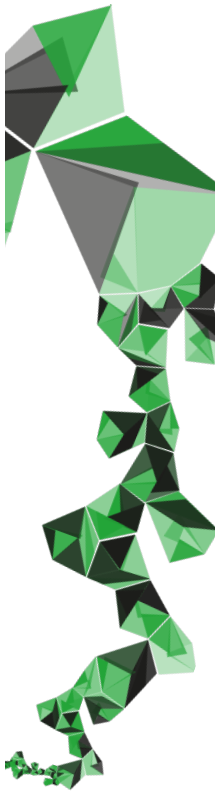
Discussion

- What is necessary for two (or more) parties to meaningfully communicate?



- Physical connection
- Communication rules
- Message agreements

NETWORK PROTOCOL

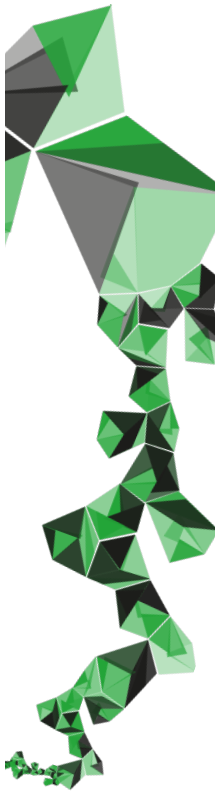


Protocol

- Given that a physical connection exists!
- Agreement on data exchange between two or more parties

Protocol elements

- Data format, order of data items
- Interpretation: how processes must behave (react to received data)
  - Can be defined in, e.g., a statechart or sequence diagram



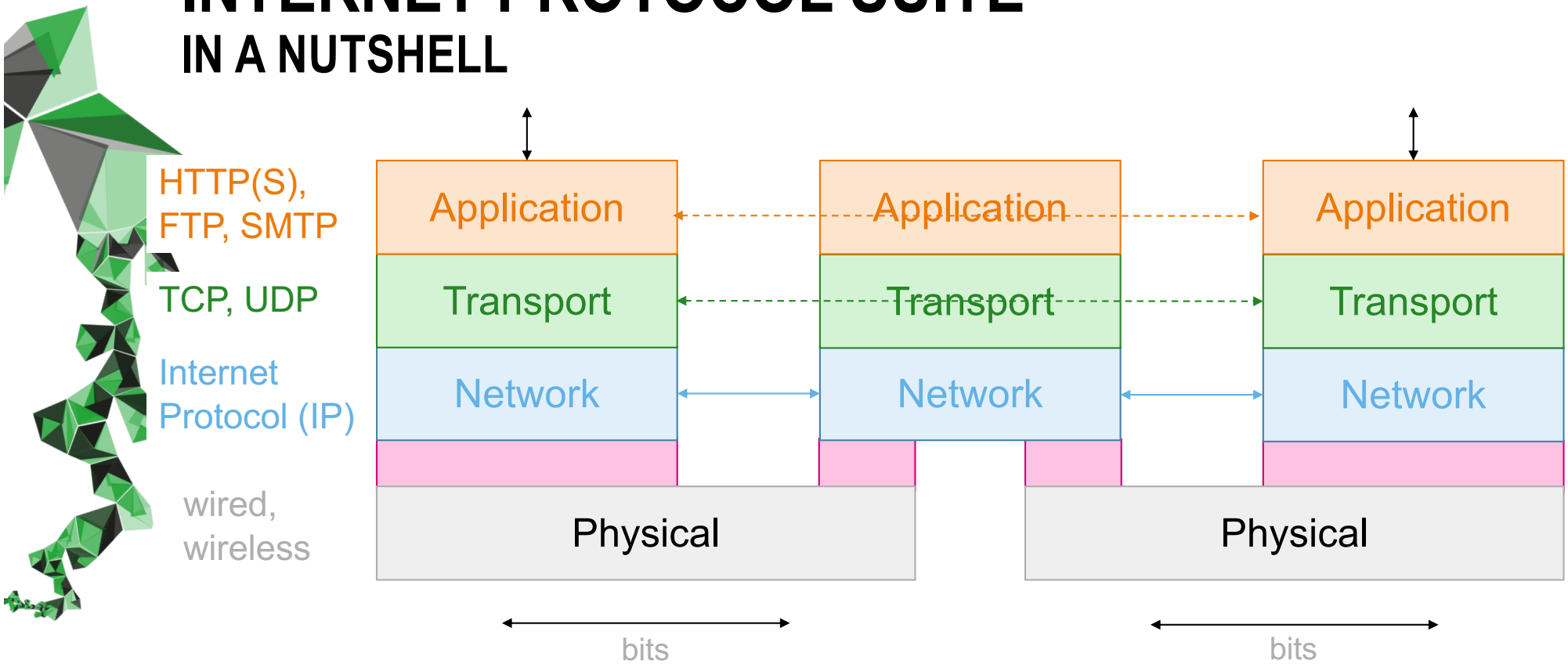
PROTOCOLS
EXAMPLES

- Real life: starting/ending phone conversation
- Internet Protocol (IP): peer-to-peer communication
- Transmission Control Protocol (TCP): end-to-end communication

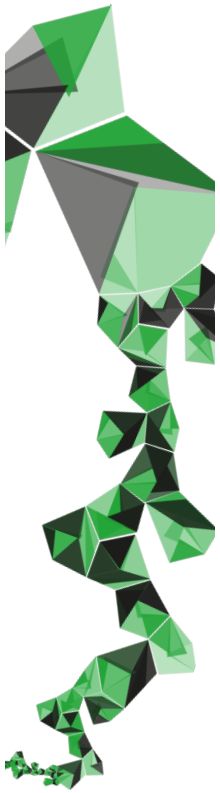
} Network protocols

- Popular paradigm: Client/Server architecture
  - Server waits for Client
  - Client queries the Server

INTERNET PROTOCOL SUITE
IN A NUTSHELL

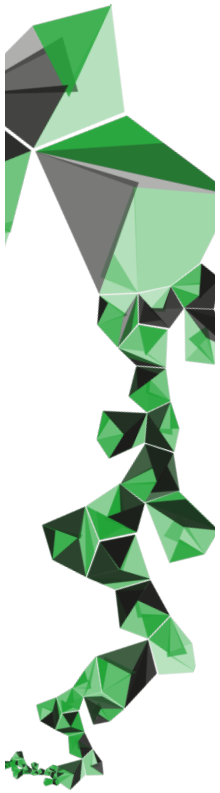


UNIVERSITY OF TWENTE.



INTERNET PROTOCOL SUITE
IN A NUTSHELL

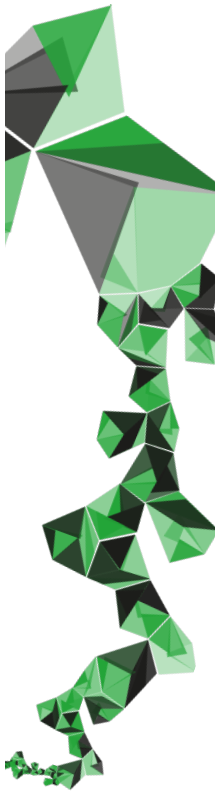
- Application Layer (HTTP, FTP, etc.): offer application to end-user
- Transport Layer: regulate end-to-end communication
  - UDP: connectionless, packets may get lost, order not guaranteed
  - TCP: connection-oriented, bytes are acknowledged, order guaranteed
- Network layer (IP): hop-to-hop transmission, packets routing
- Lower layers: sending bits from one machine to another through a physical means (ether or wire), combining bits into (byte) packets



NETWORK LAYER: IP ADDRESSES VERSION 4 AND VERSION 6

- IPv4 (first standardisation 1980): 4 x 8 bits
  - Example: `130.182.22.151`
- IPv6 (first standardisation 1998): 8 x 16 bits
  - Example: `FE80:0000:0000:0000:0202:B3FF:FE1E:8329`
- IP addresses are not intelligible → mapping to machine names is necessary
  - Domain Name Server (DNS)
  - This mapping is not part of the Internet Protocol!

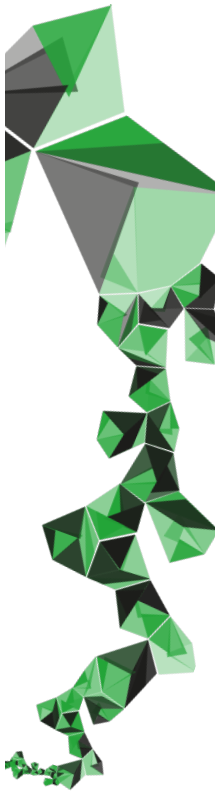
JAVA SUPPORT TO IP ADDRESSING



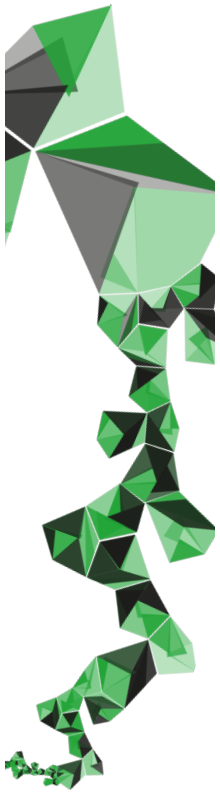
- Class `java.net.InetAddress` encapsulates an IP address
  - Supports name lookup (convert host name to IP address) and reverse lookup (convert address to host name)
- `String getHostName()` gives host name (e.g., `"www.sun.com"`)
- `String getHostAddress()` gives address (e.g., `"192.18.97.241"`)
- Factory method `InetAddress getByName(String hostName)`
  - `hostName` can be `"host.domain.com"` or `"130.95.72.134"`
- `static InetAddress getLocalHost()`

TRANSPORT LAYER: SOCKET

Originated with UNIX/C, when protocols were handled as files!

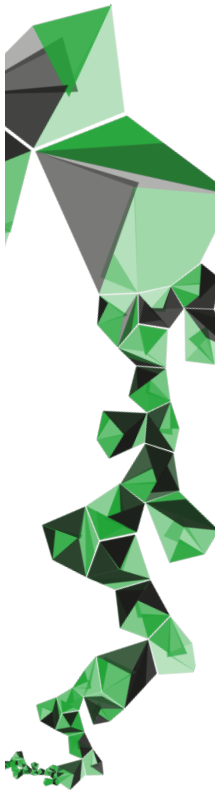


- Socket is a common abstraction of the transport layer
  → TCP or UDP
  - IP address (machine) and port number (application)
- Provides communication between program parts
  - Server waits on a port at an Internet address
  - Client tries to connect with port at an Internet address
- Java classes
  - `Socket`, `ServerSocket` (TCP)
  - `DatagramSocket` (UDP; not further covered here)



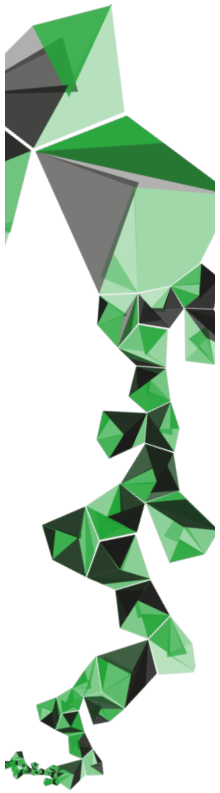
JAVA.NET.SERVERSOCKET
SOCKET TO ACCEPT CONNECTIONS

- Listens on fixed port for incoming connections
  - Creates a connection on a port for each incoming request
  - Provides a `Socket` object for each created connection
- `ServerSocket(int port)`: constructor; if port is `0` then a free port is chosen
- `Socket accept()`
  - Blocks waiting for a client's attempt to establish a connection
  - Returns a `Socket` if the attempt is successful



JAVA.NET.SERVERSOCKET (CONT.)
MORE METHODS

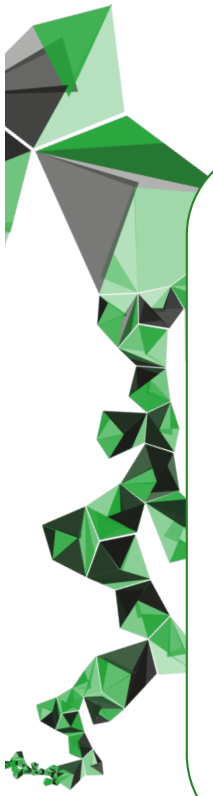
- `void close()`
  - Like for streams, deallocates resources
  - Threads waiting `accept()` calls throw a `SocketException`
- `InetAddress getInetAddress()`
  - Returns the local IP-address of this `ServerSocket`
- `int getLocalPort()`
  - Returns the port on which this `ServerSocket` listens



JAVA.NET.SOCKET
SOCKET FOR COMMUNICATION

- Provides access to TCP/IP streams
  - Bi-directional communication between sender and receiver
- `Socket(String remoteHost, int port)`
  - Constructor, starts a connection with remote host at port
- `InputStream getInputStream()`
  - Allows data from the other party to be received
- `OutputStream getOutputStream()`
  - Allows data to be sent to the other party

SERVERSOCKET AND SOCKET
ARBITRARY PORTS



Client

```
s = new Socket
  ("server", 9080);
```



```
s.getInputStream()

s.getOutputStream()
```

Example port numbers

2037 ----------→ 9080

2037 ←---------- 1583

2037 ----------→ 1583

Server "server"

```
ServerSocket ss =
  new ServerSocket(9080);
...
Socket s = ss.accept();
```



```
s.getOutputStream()

s.getInputStream()
```

DATE SERVER EXAMPLE

SERVER CODE

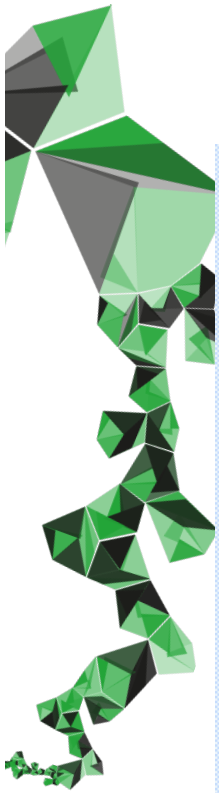
```
17  public class DateServer {
18
19      public static final int LISTENING_PORT = 32007;
20
21      public static void main(String[] args) {
22          ServerSocket listener;  // Listens for incoming connections.
23          Socket connection;      // For communication with the connecting program.
24          // Accept and process connections forever, or until some error occurs.
25          try {
26              listener = new ServerSocket(LISTENING_PORT);
27              System.out.println("Listening on port " + LISTENING_PORT);
28              while (true) {
29                      // Accept next connection request and handle it.
30                  connection = listener.accept();
31                  sendDate(connection);
32              }
33          }
34          catch (Exception e) {
35              System.out.println("Sorry, the server has shutdown.");
36              System.out.println("Error:  " + e);
37              return;
38          }
39      }
```

Listens to connection requests at port 32007

Loop: Accepts connections and sends date

DATE SERVER EXAMPLE

SEND DATE



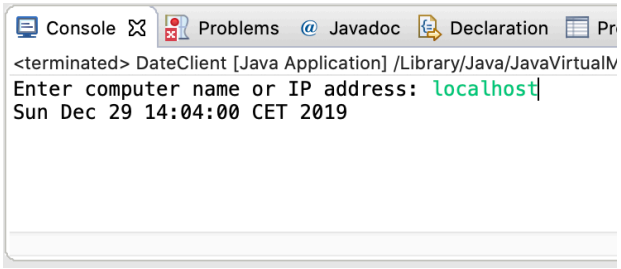
```
44      /**
45       * The parameter, client, is a socket that is already connected to another
46       * program.  Get an output stream for the connection, send the current time,
47       * and close the connection.
48       */
49      private static void sendDate(Socket client) {
50          try {
51              System.out.println("Connection from " +
52                      client.getInetAddress().toString() );
53              Date now = new Date();  // The current date and time.
54              PrintWriter outgoing;   // Stream for sending data.
55              outgoing = new PrintWriter( client.getOutputStream() );
56              outgoing.println( now.toString() );
57              outgoing.flush();  // Make sure the data is actually sent!
58              client.close();
59          }
60          catch (Exception e){
61              System.out.println("Error: " + e);
62          }
63      } // end sendDate()
```

DATE SERVER EXAMPLE

CLIENT CODE

```
20  public class DateClient {
21
22      public static final int LISTENING_PORT = 32007;
23
24      public static void main(String[] args) {
25          String hostName; // Name of the server computer to connect to.
26          Socket connection; // A socket for communicating with server.
27          BufferedReader incoming; // For reading data from the connection.
28          /* Get computer name from command line. */
29          if (args.length > 0)
30              hostName = args[0];
31          else {
32              hostName = "localhost";
33          }
34          /* Make the connection, then read and display a line of text. */
35          try {
36              connection = new Socket(hostName, LISTENING_PORT);
37              incoming = new BufferedReader(new InputStreamReader(connection.getInputStream()));
38              String lineFromServer = incoming.readLine();
39              if (lineFromServer == null) { // Null indicates end-of-stream
40                  throw new IOException("No data was sent!");
41              }
42              System.out.println(lineFromServer);
43              incoming.close();
44          } catch (Exception e) {
45              System.out.println("Error:  " + e);
46          }
47      }
48  }
```

Executing client



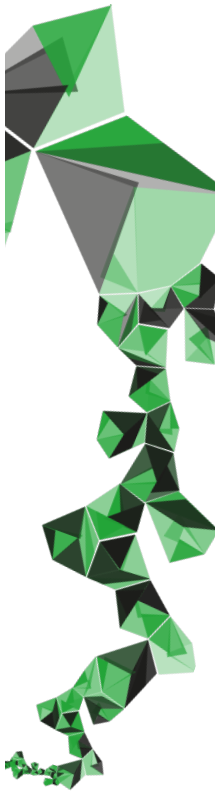
Opens a connection to host computer with port 32007

Reads line from server

Prints received line

UNIVERSITY OF TWENTE.

TAKE HOME MESSAGES



- Relevant protocols for this module
  - Network layer: Internet Protocol (IP)
  - Transport layer: Transmission Control Protocol (TCP)
  - Application layer: Hypertext Transmission Protocol (HTTP)
- Java provides support for these layers
  - Network layer: `INetAddress`
  - Transport layer: `Socket`, `ServerSocket`
  - Application layer: `URI`, `URL`