Java - Lesson 4

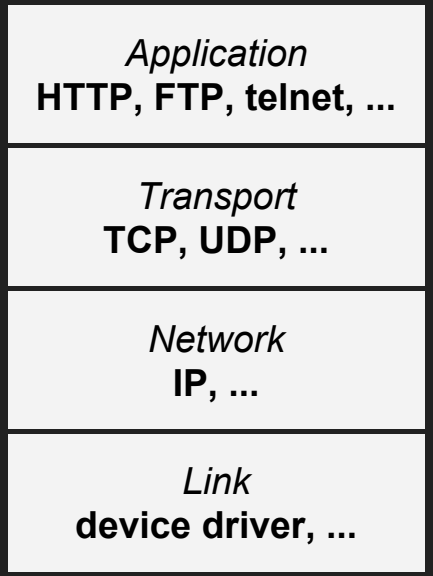
IO and Networking

Author: Kirill Volkov

vulkovk@gmail.com

Network protocol layers

- Most developers work with Application and Transport layers



URL’s

- URL is an acronym for Uniform Resource Locator and is an address to a resource on the Internet.
- A URL has two main components:
  - Protocol identifier: For the URL *http://example.com*, the **protocol** identifier is ***http***.
  - Resource name: For the URL *http://example.com*, the **resource** name is ***example.com***.
- Host Name
  - The name of the machine on which the resource lives.
- Filename
  - The pathname to the file on the machine.
- Port Number
  - The port number to which to connect (typically optional).
- Reference
  - A reference to a named anchor within a resource that usually identifies a specific location within a file (typically optional).

Creating URL’s

http://example.com/pages/page1.html
http://example.com/pages/page2.html

```
URL myURL = new URL("http://example.com/pages/");
URL page1URL = new URL(myURL, "page1.html");
URL page2URL = new URL(myURL, "page2.html");
```

For encoding conversions use URI

http://example.com/hello world/

```
// need to use special characters
URL url = new URL("http://example.com/hello%20world");

// or simply use URI
URI uri = new URI("http", "example.com", "/hello world/", "");
URL url = uri.toURL();
```

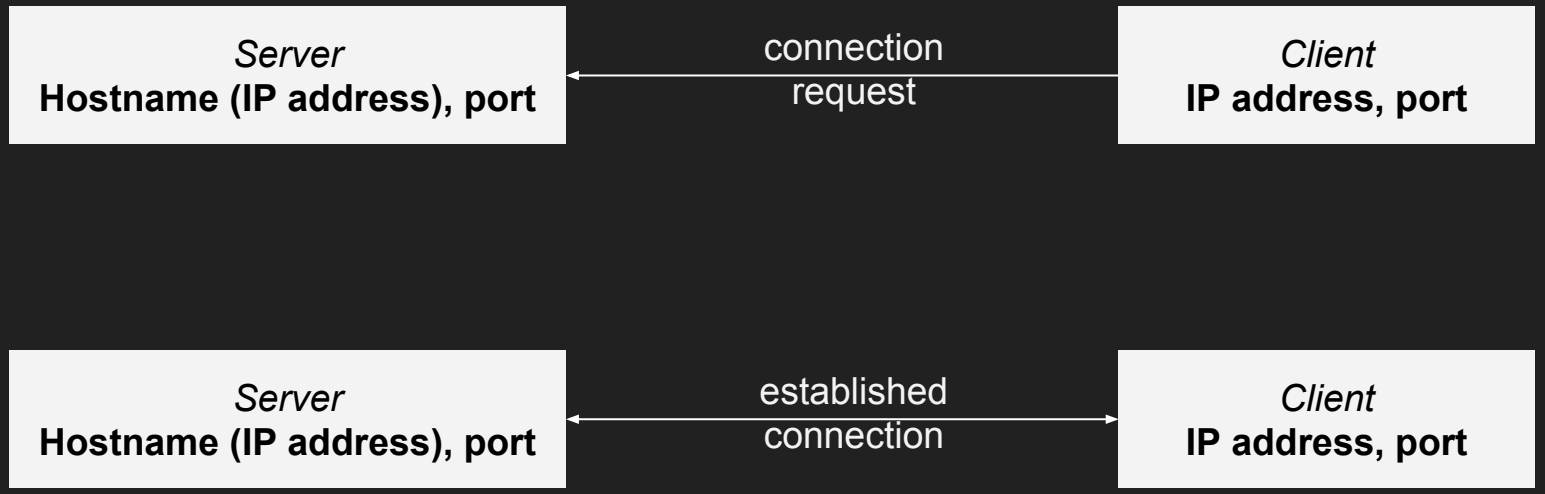
Reading from URL's

Url has an openStream() method that will contain data aquired from remote.

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

public class URLReader {
    public static void main(String[] args) {

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

        BufferedReader in;
        try {
            in = new BufferedReader(new InputStreamReader(url.openStream()));

            String inputLine;
            while ((inputLine = in.readLine()) != null)
                System.out.println(inputLine);

        } catch(Exception e) {
            // catch exception
        } finally {
            in.close();      // Always close streams to avoid memory leaks
        }
    }
}
```

Sockets: TCP socket

- Socket is a two-way connection link
- TCP Sockets are a safe way to send data over network. Delivery is guaranteed



Sockets: server example

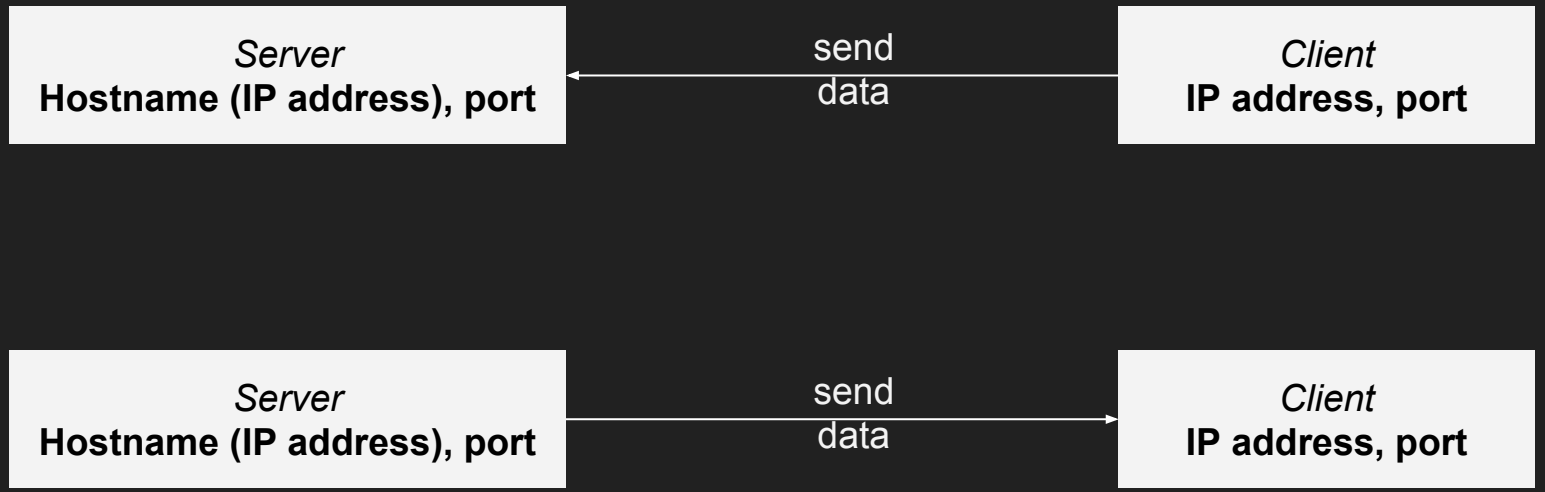
```
public class EchoServer {
    public static void main(String[] args) throws IOException {

        if (args.length != 1) {
            System.err.println("Usage: java EchoServer <port number>");
            System.exit(1);
        }

        int portNumber = Integer.parseInt(args[0]);

        try (
            ServerSocket serverSocket = new ServerSocket(Integer.parseInt(args[0]));
            Socket clientSocket = serverSocket.accept();

            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        ) {
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                out.println(inputLine);
            }
        } catch (IOException e) {
            System.out.println("Exception caught when trying to listen on port "
                + portNumber + " or listening for a connection");
            System.out.println(e.getMessage());
        }
    }
}
```

Sockets: client example

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

        if (args.length != 2) {
            System.err.println(
                "Usage: java EchoClient <host name> <port number>");
            System.exit(1);
        }

        String hostName = args[0];
        int portNumber = Integer.parseInt(args[1]);

        try (
            Socket echoSocket = new Socket(hostName, portNumber);
            PrintWriter out =
                new PrintWriter(echoSocket.getOutputStream(), true);
            BufferedReader in =
                new BufferedReader(
                    new InputStreamReader(echoSocket.getInputStream()));
            BufferedReader stdIn =
                new BufferedReader(
                    new InputStreamReader(System.in))
        ) {
            String userInput;
            while ((userInput = stdIn.readLine()) != null) {
                out.println(userInput);
                System.out.println("echo: " + in.readLine());
            }
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host " + hostName);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to " +
```

Sockets: UDP socket

- UDP socket aka Datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.



Sockets: UDP server example

```
public class QuoteServerThread extends Thread {

    protected DatagramSocket socket = null;
    protected BufferedReader in = null;
    protected boolean moreQuotes = true;

    public QuoteServerThread() throws IOException {
        this("QuoteServerThread");
    }

    public QuoteServerThread(String name) throws IOException {
        super(name);
        socket = new DatagramSocket(4445);

        try {
            in = new BufferedReader(new
FileReader("one-liners.txt"));
        } catch (FileNotFoundException e) {
            System.err.println("Could not open quote file.
                                    Serving time instead.");
        }
    }

    public void run() {

        while (moreQuotes) {
            try {
                byte[] buf = new byte[256];

                // receive request
                DatagramPacket packet = new DatagramPacket(buf,

buf.length);
                socket.receive(packet);

                // figure out response
                if (in == null) dString = new Date().toString();
                else            dString = getNextQuote();

                buf = dString.getBytes();

                // send the response to the client at "address"
                // and "port"
                InetAddress address = packet.getAddress();
                int port = packet.getPort();
                packet = new DatagramPacket(buf, buf.length,
                                            address, port);
                socket.send(packet);
            } catch (IOException e) {
               e.printStackTrace();
               moreQuotes = false;
            }
        }
        socket.close();
    }

    protected String getNextQuote() {
        String returnValue = null;
        try {
            if ((returnValue =  in.readLine()) == null) {
                in.close();
                moreQuotes = false;
                returnValue = "No more quotes. Goodbye.";
            }
        } catch (IOException e) {
            returnValue = "IOException occurred in server.";
        }
        return returnValue;
    }
}
```

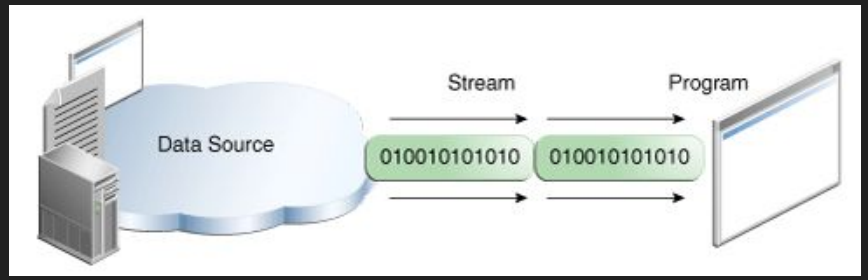
Sockets: UDP client example

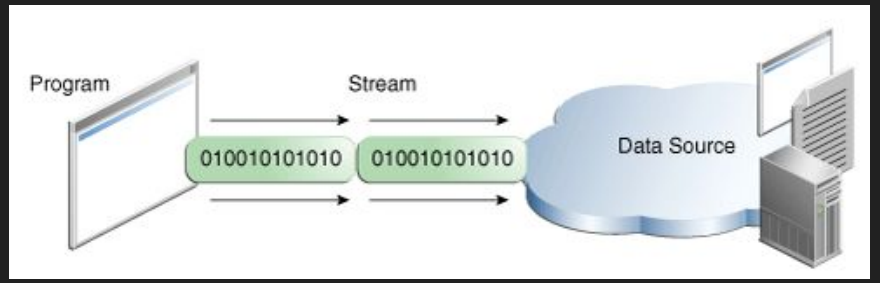
```
public class QuoteClient {

    public static void main(String[] args) throws IOException {
        if (args.length != 1) {
             System.out.println("Usage: java QuoteClient <hostname>");
             return;
        }

        // get a datagram socket
        DatagramSocket socket = new DatagramSocket();

        // send request
        byte[] buf = new byte[256];
        InetAddress address = InetAddress.getByName(args[0]);
        DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 4445);
        socket.send(packet);

        // get response
        packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);

        // display response
        String received = new String(packet.getData(), 0, packet.getLength());
        System.out.println("Quote of the Moment: " + received);

        socket.close();
    }
}
```

```
public class QuoteServer {
    public static void main(String[] args) throws IOException {
        new QuoteServerThread().start();
    }
}
```

IO: streams

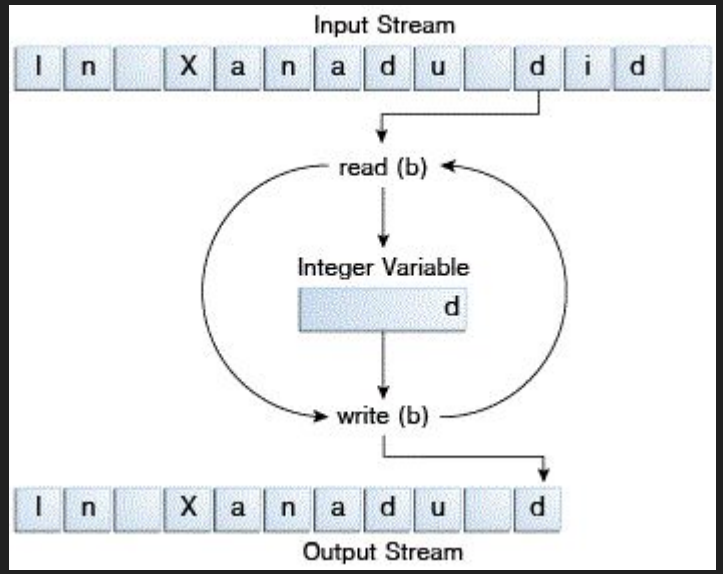
- An IO Stream represents an input source or an output destination
  - disk files
  - devices
  - other programs
  - memory arrays.





IO: byte stream

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

        FileInputStream in = null;
        FileOutputStream out = null;

        try {
            in = new FileInputStream("xanadu.txt");
            out = new FileOutputStream("outagain.txt");
            int c;

            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```



IO: character stream, buffered stream

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

        FileReader inputStream = null;
        FileWriter outputStream = null;

        try {
            inputStream = new FileReader("xanadu.txt");
            outputStream = new FileWriter("characteroutput.txt");

            int c;
            while ((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

```
inputStream = new BufferedReader(new FileReader("xanadu.txt"));
outputStream = new BufferedWriter(new FileWriter("characteroutput.txt"));
```

IO: data and object streams

- DataInputStream and DataOutputStream support reading and writing of
  - byte, unsigned byte
  - short, unsigned short
  - int
  - long
  - float
  - double
  - boolean
  - String
- ObjectInputStream and ObjectOutputStream support reading and writing of Objects, that implement Serializable interface

```
Object ob = new Object();
out.writeObject(ob);
out.writeObject(ob);

Object ob1 = in.readObject();
Object ob2 = in.readObject();
```