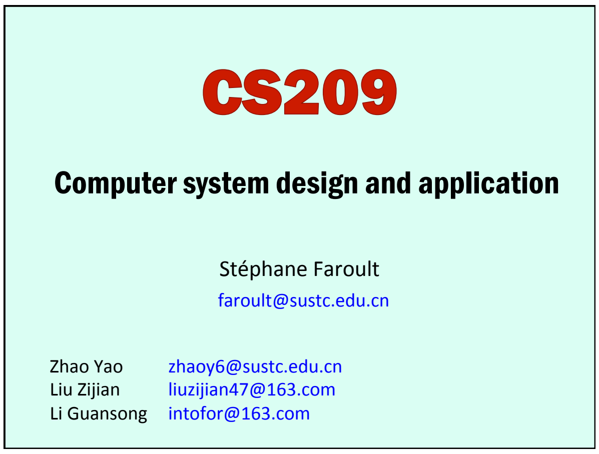
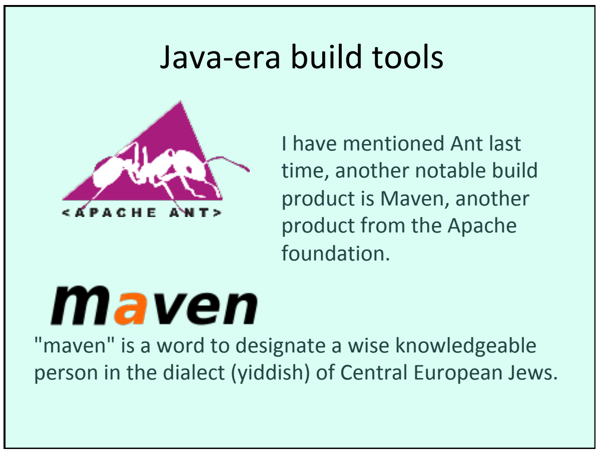


CS209

Computer system design and application

Stéphane Faroult
faroult@sustc.edu.cn

Zhao Yao	zhaoy6@sustc.edu.cn
Liu Zijian	liuzijian47@163.com
Li Guansong	intofor@163.com



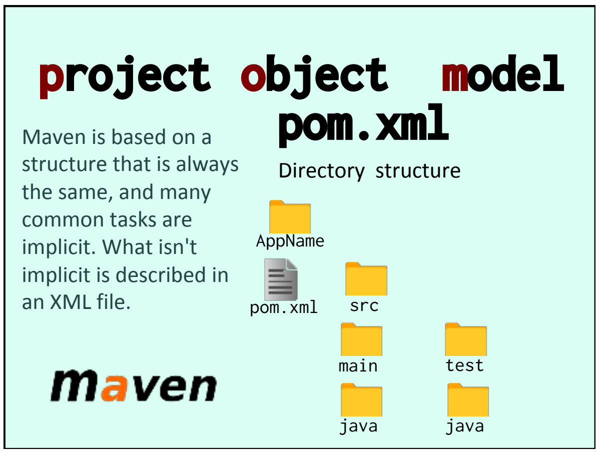
Java-era build tools

<APACHE ANT>

I have mentioned Ant last time, another notable build product is Maven, another product from the Apache foundation.

maven

"maven" is a word to designate a wise knowledgeable person in the dialect (yiddish) of Central European Jews.



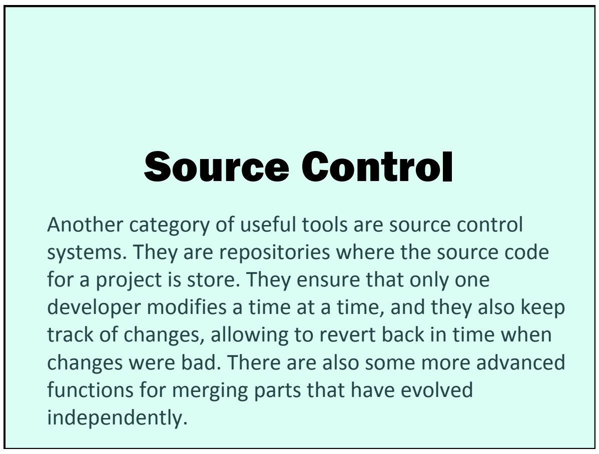
project object model
pom.xml

Maven is based on a structure that is always the same, and many common tasks are implicit. What isn't implicit is described in an XML file.

Directory structure

```
AppName
  pom.xml
  src
    main
      java
    test
      java
```

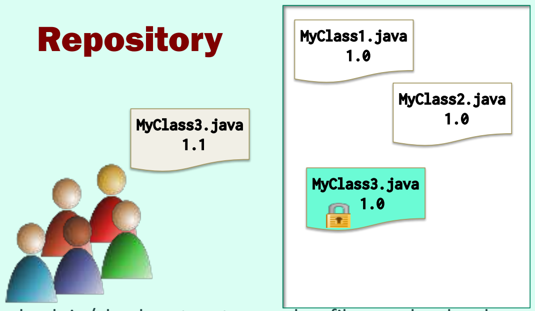
maven



Source Control

Another category of useful tools are source control systems. They are repositories where the source code for a project is store. They ensure that only one developer modifies a time at a time, and they also keep track of changes, allowing to revert back in time when changes were bad. There are also some more advanced functions for merging parts that have evolved independently.

Source Control Systems



A check-in/check-out system makes files read-only when someone is modifying them.

Source Control Systems

SCCS Bell Laboratories First one, 1970s

SUBVERSION

git

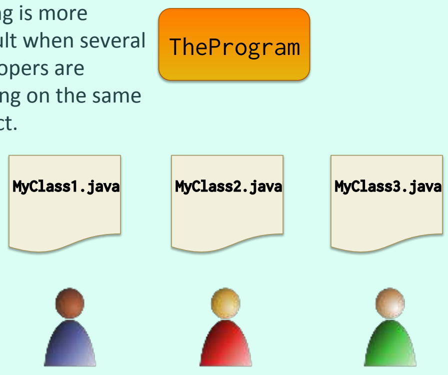
mercurial

There are many systems, the first ones were local to one computer, today they can be local or use a web server.

Testing

A very important phase in the life of a developer is testing, which is both kind of boring and difficult to do properly. Testing is a task that has to be done repeatedly, because very often a change (new feature, bug fix) breaks something that used to work. For big projects, you have "test suites" that just run the software through a lot of controls and checks that everyone of them is passed.

Testing is more difficult when several developers are working on the same project.

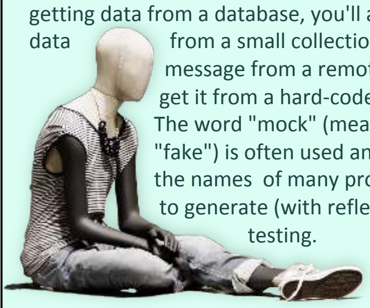


The fastest developer cannot wait on the slowest one to test the code!

You usually want to test your code as early as possible, even if you are using objects and methods currently being developed by someone else (remember that object-oriented programming is mostly objects exchanging messages by calling methods)

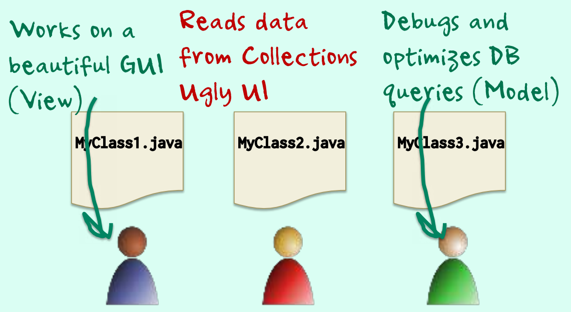
Write dummy classes and methods for testing

The tactic is usually to create very simple objects and methods that simply simulate the real thing. Instead of getting data from a database, you'll always return the same data from a small collection. Instead of getting a message from a remote server, you'll get it from a hard-coded method. The word "mock" (meaning "imitation" or "fake") is often used and can be found in the names of many products helping to generate (with reflection...) code for testing.



MOCK

Write dummy classes and methods for testing


Works on a beautiful GUI (View)
Reads data from Collections Ugly UI
Debugs and optimizes DB queries (Model)
MyClass1.java
MyClass2.java
MyClass3.java

This allows developers to test their code without having to wait for others – or without having written all their methods.

DEPENDENCY INJECTION

An important idea for testing is that when an object depends on another object from a different class, it should not create the other object, but should get a reference to it. The dependency is "injected" (passed). This makes testing far easier, because you don't have to worry about what the constructor should look like and what arguments it takes. Dependency injection is central to some development frameworks such as Spring and is considered a good development practice.

Several aspects to testing

Testing covers many fields – included the behaviour when something that wasn't expected happens.

Expected result?

New change doesn't break something?
(Non regression)

What the user wanted?
(User Acceptance Test)

Correct performance?
(Load testing)

JUNIT junit.org

also TestNG

Some tests are usually carried out by support teams, not by developers themselves. For the testing part that directly regards developers, some tools exist that are based on annotations.

```
import static org.junit.Assert.*;
import org.junit.*
```

```
class MyClass                  class MyClassTest

public int method1 {           @Test
}                              public void testmethod1 {
                               }
public int method2 {           @Test
}                              public void testmethod2 {
                               }
```

The idea is to mirror a class with a test class that checks, in a test method, a method from the original class.

Test methods?

A test method, annotated as such, use an assert*xxx*() function to compare the result of a method to test to an expected result.

```
@Test
public void testmethod1 {
   // Create object,
   // initialize parameters ...
   assertEquals(expected_result,
                obj.method1(...));
}
```

```
assertEquals("message",A,B);
assertTrue(A);
assertFalse(A);
assertNotNull(A);
...
```

There are many assert*xxx* methods, that can optionally take a messsage as parameter.

```
@Test
@Before
@After
@Test(expected = Exception.class)
@Test(timeout = 100)
```

Annotations allow to define "before" and "after" operations, and even to check that we are getting the proper exception.

Ideally test only one class

As many test methods as you want

"Test suites"

Normally you are supposed to test one class at once, and you can have multiple test methods to test different aspects (there is NOT a one-to-one correspondance between methods being tested and test methods).

Tests can be run from multiple environments.

Running JUnit Tests

IDE eclipse

Build tool maven

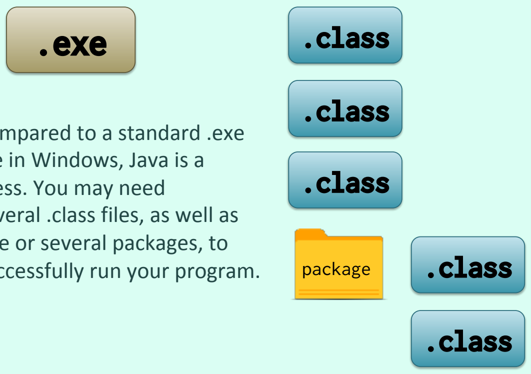
Command line

```
java org.junit.runner.JUnitCore TestClass1 [...other test classes...]
```

Deployment

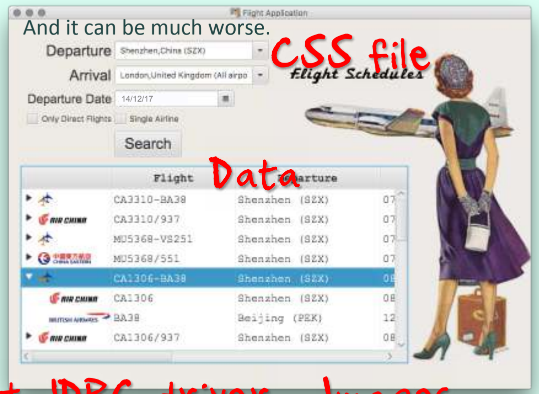
Distributing the program

The last aspect isn't the list important. How are you going to distribute your progam? Cases when the program is a single .class are very rare. Usually you need quite a number of files to successfully run a program.



Compared to a standard .exe file in Windows, Java is a mess. You may need several .class files, as well as one or several packages, to successfully run your program.

And it can be much worse.



Departure: Shenzhen,China (SZX)
Arrival: London,United Kingdom (All airpo
Departure Date: 14/12/17
☐ Only Direct Flights ☐ Single Airline

Flight	Departure
CA3310-BA38	Shenzhen (SZX)
CA3310/937	Shenzhen (SZX)
MU5368-VS251	Shenzhen (SZX)
MU5368/551	Shenzhen (SZX)
CA1306-BA38	Shenzhen (SZX)
CA1306	Shenzhen (SZX)
BA38	Beijing (PEK)
CA1306/937	Shenzhen (SZX)

When you need to send by mail several files, you often zip them. You do the same in Java in a .jar file.
The JVM knows how to read and execute a .jar file without having to unzip it first.

single file

Jar files

Either a complementary library
(such as JDBC drivers)

```
java -cp somefile.jar myprog
```

Or the main program

```
java -jar myprog.jar
```

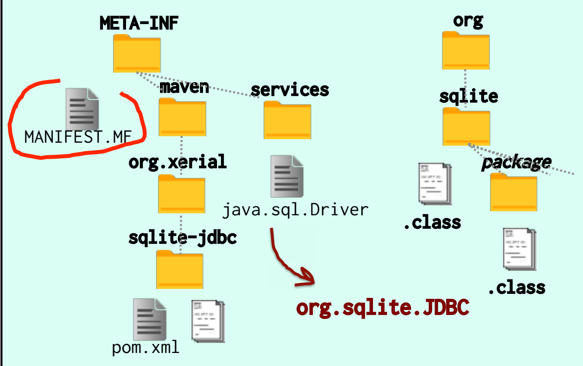
"jar" means "java archive", it's inspired by "tar" (tape archive), an old Unix command. It's also a pun, as a jar is usually a glass or earthenware container with a wide opening.

Jar files

Java archive

Technically, a **.jar** file is a compressed (zip) file.

It practice, it IS a zip file, and you can apply unzip to a .jar. Which I have done for the SQLite driver.



This is what you find, with obvious traces of Maven. The manifest describes the contents of the .jar

```
Manifest-Version: 1.0
Archiver-Version: Plexus Archiver
Created-By: Apache Maven Bundle Plugin
Built-By: leo
Build-Jdk: 1.8.0_74
Bnd-LastModified: 1484116742984
Bundle-Description: SQLite JDBC library
Bundle-License: http://www.apache.org/licenses/LICENSE-2.0.txt
Bundle-ManifestVersion: 2
Bundle-Name: SQLite JDBC
Bundle-SymbolicName: org.xerial.sqlite-jdbc;singleton:=true
Bundle-Version: 3.16.1
Export-Package: org.sqlite;version="3.16.1.SNAPSHOT";uses:="javax.sql,
 org.sqlite.core,org.sqlite.jdbc4",org.sqlite.core;version="3.16.1.SNA
 PSHOT";uses:="org.sqlite,org.sqlite.date,org.sqlite.jdbc4",org.sqlite
 .date;version="3.16.1.SNAPSHOT",org.sqlite.javax;version="3.16.1.SNAP
 SHOT";uses:="javax.sql,org.sqlite,org.sqlite.jdbc4",org.sqlite.jdbc3;
 version="3.16.1.SNAPSHOT";uses:="org.sqlite,org.sqlite.core",org.sqli
 te.jdbc4;version="3.16.1.SNAPSHOT";uses:="javax.sql,org.sqlite,org.sq
 lite.core,org.sqlite.jdbc3",org.sqlite.util;version="3.16.1.SNAPSHOT"
Import-Package: javax.sql;resolution:=optional
Originally-Created-By: Apache Maven Bundle PluginTool: Bnd-2.1.0.20130426-122213
```

This is the content of the Manifest file.

Note that references to files in a .jar are supposed to be in the .jar, unless they are prefixed by **file:**

References to files

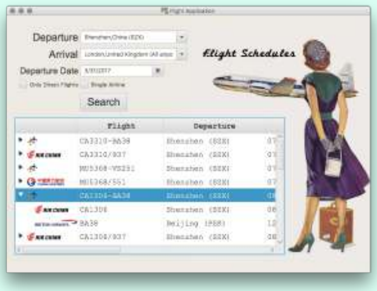
file:*path* Operating System

.jar file

look in CLASSPATH

```
this.getClass()
    .getClassLoader()
    .getResource("images/image.png")
```

images must be in CLASSPATH



Before packaging my application into a .jar file, I must have clean directories.

```
FlightController  FlightModel  FlightView  CustomSQLFunc  resources
.class            .class       .class      .class         images  data
FlightApp.class
```

main() is here

```
$ jar cvf FlightApp.jar CustomSQLFunc FlightController
FlightModel FlightView resources
added manifest
adding: CustomSQLFunc/(in = 0) (out= 0)(stored 0%)
adding: CustomSQLFunc/ConvertMinToHour.class(in = 1223)
(out= 734)(deflated 39%)
adding: CustomSQLFunc/GetArrivalTime.class(in = 2280)
(out= 1226)(deflated 46%)
adding: CustomSQLFunc/GetDayOfWeek.class(in = 1374)
...
```

A default manifest is created.

```
FlightController  FlightModel  FlightView  CustomSQLFunc  resources
.class            .class       .class      .class         images  data
FlightApp.class
```

Create file manifest.txt

```
Main-Class:FlightController.FlightApp
Class-Path:resources sqlite-jdbc-x.x.x.jar
```

I can customize the Manifest.xml file by creatin a manifest.txt saying first where is the main my default class path

Carriage return required!

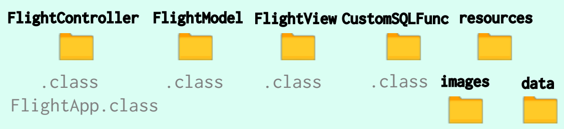
```
FlightController  FlightModel  FlightView  CustomSQLFunc  resources
.class            .class       .class      .class         images  data
FlightApp.class
```

Create file manifest.txt

```
Main-Class:FlightController.FlightApp
Class-Path:resources sqlite-jdbc-x.x.x.jar
```

u means "update"

```
$ jar ufm FlightApp.jar manifest.txt
CustomSQLFunc FlightController FlightModel
FlightView resources
```



An IDE (or ant or maven) can also prepare the .jar file.

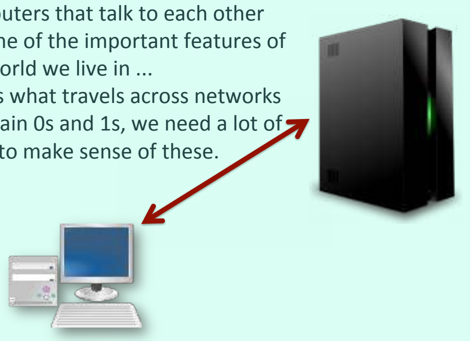
... but it's always good to be able to do it by hand.

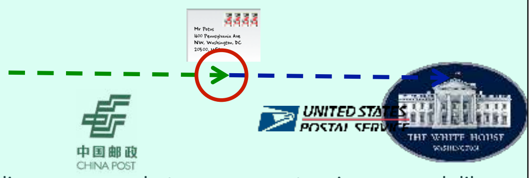
Network Programming

(short overview)

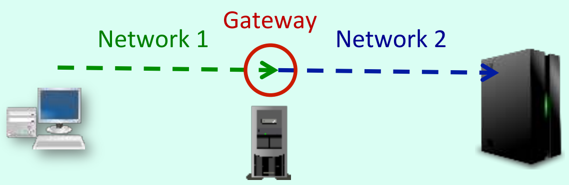
Change of topic, to switch to something that is in no way specific to Java, but that Java can ALSO do.

Computers that talk to each other are one of the important features of the world we live in ...
But as what travels across networks are plain 0s and 1s, we need a lot of rules to make sense of these.





Sending a message between computers is very much like sending a letter, say to the White House. You may drop it in a China Post mailbox, but it will not be delivered by China Post. China Post will take it by plane to the US, perhaps to Los Angeles, where the letter will be handled to another network (USPS, United States Postal Services). Additionally, none of them cares about what you wrote.

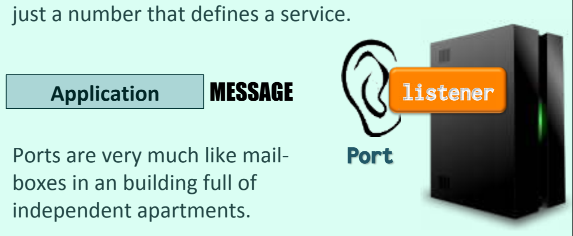


It's very much the same with computers. Your machine is on a network, and what you send must reach a special computer called a gateway that has two network cards connected to different networks and will send your "message" to another network (and possibly many gateways) until it reaches the target computer.

Need for PROTOCOLS (RULES)

In the same way that there are rules for sending a letter (address on the front, sender address on the back, country at the bottom when sending abroad, stamps...) there are rules for sending messages on a computer network. The word used isn't "rules" but "protocols", which basically means the same ("behavior rules").

To be able to send a message, some program must be there to receive it. Some programs, collectively known as listeners or servers, do this. There may be several ones on one computer, they are listening for a "port" which is just a number that defines a service.



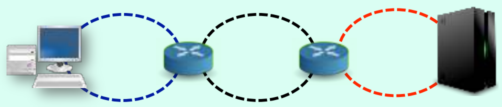
Ports are very much like mail-boxes in an building full of independent apartments.

IP address

To send your message you must provide the port, but also the address or name of the computer (a name will be converted to an address). You may have seen IP address, they look like 192.254.23.127. Your message will be packaged with this information according to a set of rules known as TCP

Transmission Control Protocol

Application
Host-to-Host



TCP relies on another set of rules that ensure that your message travels from your network to the target network:

Application
Host-to-Host
Internet

IP **Internet Protocol**

"Internet" simply means "between networks"

All this can only work if you can reach the network and the gateway, where machines know as "routers" will direct your message.

Application
Host-to-Host
Internet
Network Access

We have seen URI/URL (more or less the same already) a lot of operations in which networks are involved are transparent, and done by methods in libraries. However, if you have special needs, you may want to code a network application of your own.

How does it work at a low level?

In Java (as in C) network communications are based on a "socket". In Java it's a class, and you use it like a file.

Based on a SOCKET

Server socket

●   ●

Client socket

stream, like a file

talks to one server

```
import java.net.*;

        class Socket
        class ServerSocket
```

can serve several clients

```
Socket s = new Socket(hostname,
                      port_number);

in = new BufferedReader(new
       InputStreamReader(s.getInputStream()));
out = new BufferedWriter(new
       OutputStreamWriter(s.getOutputStream()));
```

What travels along a network is nothin more than a byte stream. As a socket works in both directions, you have both an input and an output stream.

```
while ((fromServer = in.readLine()) != null) {
      // Analyze fromServer message
      // Prepare fromUser message
      out.write(fromUser);
}

in.close();
out.close();
s.close();
```

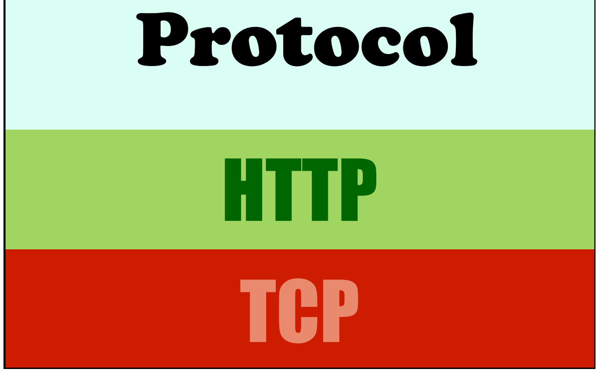
It's just a matter of reading and writing. All the lower-level protocols are managed inside the socket object (much easier than in C, for those who have taken or are taking CS205). However YOUR application needs a protocol: messages and answers need a meaning!

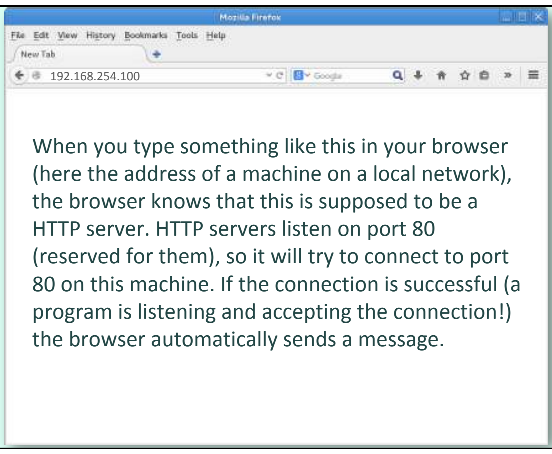
Exemple:
Get the home page of a website

A simple example is getting the home page of any website. Your browser uses a protocol the name of which must be familiar to you: HTTP, or Hyper-Text Transfer Protocol. It sends messages that must be understandable to a server, and the server sends back "web pages", using a protocol that must be understandable by the server. As this protocol is publicly specified, anybody can write an HTTP server or a browser as long as you respect the rules.

In the case of HTTP, I have represented the server by an Apache maiden ("Apache" is a very popular web server). The program that runs is called "httpd", the "**d**" is often used in Unix system to indicate that a program runs in the background without interacting with the screen nor the keyboard (these programs are called "**d**aemons" in Unix systems)

The HTTP protocol is an application protocon on top of TCP: just text with special keywords. Let's take a look at it.





When you type something like this in your browser (here the address of a machine on a local network), the browser knows that this is supposed to be a HTTP server. HTTP servers listen on port 80 (reserved for them), so it will try to connect to port 80 on this machine. If the connection is successful (a program is listening and accepting the connection!) the browser automatically sends a message.

```
GET / HTTP/1.1
Host: 192.168.254.100
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:2.0.1) [...]
Accept: text/html,application/xhtml+xml,application/xml;[...]
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Cache-Control: max-age=0
[empty line]
```

This is what is sent; only the first two lines and the empty line at the end are mandatory.

The first line says "send me your home page, I'm talking this version of HTTP". The second line repeats the server address for control.

var
www
htdocs

.htaccess ?

index.html

An Apache server will look into some special directories, will check for a .htaccess file that may require a password, and if everything is OK will send back file index.html.

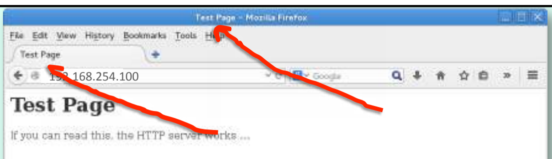
index.html may look like this. What the browser will display is between **<body>** and **</body>**

```
<html>
   <head>
      <title>Test Page</title>
   </head>
   <body>
      <h1>Test Page</h1>
       <p>If you can read this, the HTTP server
works ...</p>
   </body>
</html>
```

index.html

```
HTTP/1.1 200 OK
Date: [...]
Server: Apache/2/2/15 (Linux/SUSE)
Last-Modified: [date]
ETag: "2d7d-b7-4a3c52ef29046"
Accept-Ranges: bytes
Content-Length: 175
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
[empty line]
<html>
  <head>
   <title>Test Page</title>
  </head>
  <body>
    <h1>Test Page</h1>
    <p>If you can read this, the HTTP server works ...</p>
  </body>
</html>
```

Before the page, Apache sends a "header", which is read by the browser

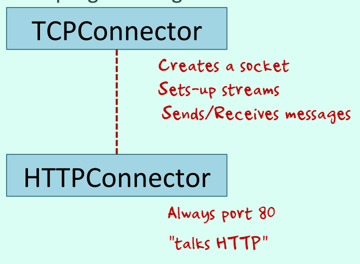

Test Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Test Page
192.168.254.100
Google

Test Page

If you can read this, the HTTP server works ...

Information in the **<head> </head>** section of the HTML page is used by the browser for setting window/tab titles.

Borrowing from something done in C++ by a British consultant called Vic Hargrave, I have created two classes to make programming easier.



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

class TCPConnector {
    private String         hostName;
    private String         hostAddr;
    private int            port;
    private Socket         s = null;
    private BufferedReader in = null;
    private BufferedWriter out = null;
```

www.sample.com
123.123.123.123

The TCPConnector class sets up a socket for communicating with a server.

```
public TCPConnector(String host, int portNum)
                    throws IOException {
    hostName = host;
    port = portNum;
    s = new Socket(host, portNum);
    s.setSoTimeout(1000);
    hostAddr = s.getInetAddress().toString().split("/")[1];
    in = new BufferedReader(new
                InputStreamReader(s.getInputStream()));
    out = new BufferedWriter(new
                OutputStreamWriter(s.getOutputStream()));
}
```

A simple constructor. Method getInetAddress() returns, when transformed into a string, a website name (possibly empty) followed by "/" and an IP address.

Closing connections properly is important ...

```
public void close() throws IOException {
  try {
    in.close();
    out.close();
    s.close();
  } catch (java.net.SocketException e) {
    // Do nothing
  }
}

public String getHostAddr() {
    return hostAddr;
}
```

And other than this two simple methods to receive and send a message. You need to flush the message when sending, otherwise it won't go before buffers are full...

```
  public void send(String msg) throws IOException {
      out.write(msg);
      out.flush();   // IMPORTANT
  }

  public String receive() throws IOException {
      try {
        String s = in.readLine();
        return s;
      } catch (SocketTimeoutException e) {
        return null;
      }
  }
}
```

This one is simplistic and should be far more complicated.

```
class HTTPConnector extends TCPConnector {
    private StringBuffer header;
    private StringBuffer body;

    public HTTPConnector(String host) throws IOException {
        super(host, 80);
        header = new StringBuffer();
        body = new StringBuffer();
    }

    public String get(String pagename) throws IOException {
       String msg;
       header.delete(0, header.length());
       body.delete(0, body.length());
```

The get function handles separately header and body. The goal is to be able to store if needed into the header the length of the body, so that the other end can check that the full message was received.

```
        header.append("GET " + pagename + " HTTP/1.1\n");
        header.append("Host: " + getHostAddr() + "\n");
        header.append("User-Agent: Java test program\n");
        header.append("\n");
        send(header.toString());
        header.delete(0, header.length());
        boolean reading_header = true;
        int     bytesToRead = -1;
        int     read = 0;
        while ((bytesToRead != 0)
                && ((msg = receive()) != null)) {
           if (reading_header) {
             if (msg.trim().isEmpty()) {
                reading_header = false;
              }
```

We send the basic message, then wait for the answer. We first read the header (over when we read an empty line)

```
          if (reading_header) {
            if (msg.trim().isEmpty()) {
               reading_header = false;
             } else {
               if (msg.toLowerCase()
                        .startsWith("content-length")) {
                  bytesToRead = Integer
                                 .parseInt(msg.split(":")[1]
                                 .trim());
               }
               header.append(msg);
            }
          }
```

When reading the header, if the conten length is given then we read it.

```
                header.append(msg);
            }
          } else {
            read = 1 + msg.length();
            body.append(msg);
            body.append("\n");
            bytesToRead -= read;
          }
        }
        return body.toString();
    }
}
```

And we finally return the body (and only the body) that we have read.

```
public class GetHomePage {

    public static void main(String[] args)
                   throws IOException {
       if (args.length > 0) {
          HTTPConnector h = new HTTPConnector(args[0]);
          System.out.println(h.get("/"));
          h.close();
       }
    }

}
```

HTTPConnector can get any page, we are asking for the home-page specifically.

All this will actually be automatically done for you if you pass to a method that takes an URI or URL as parameter anything that starts with "http:"

Many packages in Java have built-in networking capabilities

```
java.net.URI

URI resourceName = new URI("...");
  "file:...    "
  "http:...    "
```

http://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml

Writing Your Own Server

Writing a server is hardly more difficult than writing a client (actually, it's easier, because a server doesn't need a nice interface).

Define your top-level protocol!

What are the messages that the server understands?

The first, and probably more important step, is to define the application protocol. You are going to send some commands to the server. What does it understand? How does it reply? What does it say when it receives a wrong command? what are the parameters associated with a command?

Define your top-level protocol!

Title
Director
Actors/Actresses
Year
Country

For instance I have written a server that queries a database for films that match some conditions.

Define your top-level protocol!

KEYWORD cond

```
ACTRESS Audrey Hepburn
```

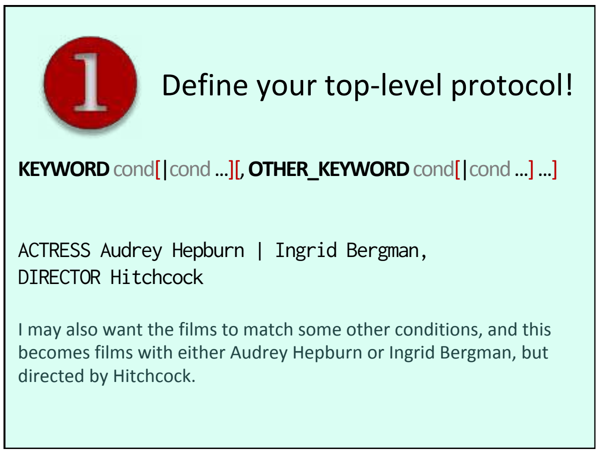
My protocol is a keyword followed by a condition. This message would ask for the list of films in which Audrey Hepburn played.

Define your top-level protocol!

KEYWORD cond[|cond ...]

```
ACTRESS Audrey Hepburn | Ingrid Bergman
```

I may want to implement "or". This would return films with either Audrey Hepburn or Ingrid Bergman.

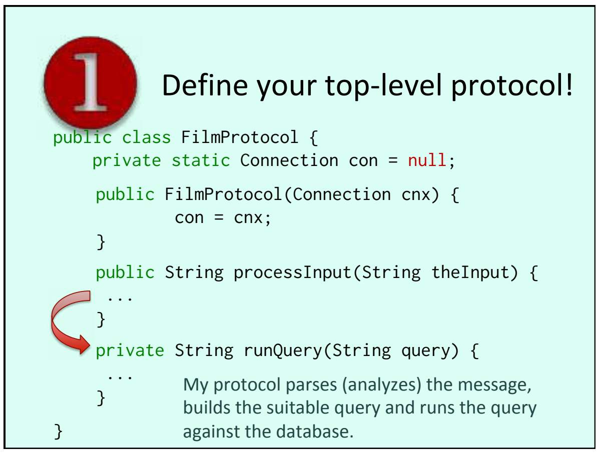


1 Define your top-level protocol!

KEYWORD cond[|cond ...][, **OTHER_KEYWORD** cond[|cond ...] ...]

```
ACTRESS Audrey Hepburn | Ingrid Bergman,
DIRECTOR Hitchcock
```

I may also want the films to match some other conditions, and this becomes films with either Audrey Hepburn or Ingrid Bergman, but directed by Hitchcock.

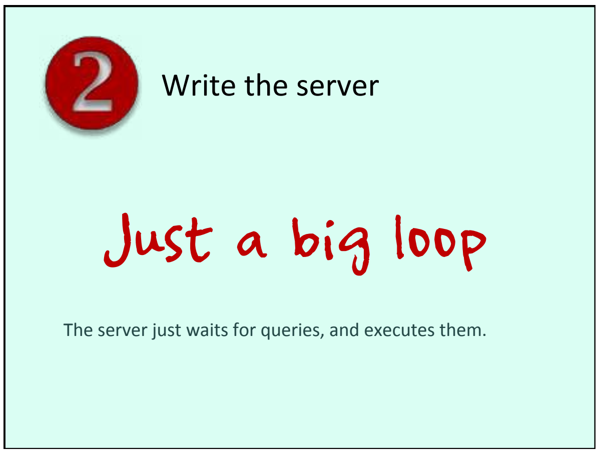


1 Define your top-level protocol!

```
public class FilmProtocol {
    private static Connection con = null;

    public FilmProtocol(Connection cnx) {
          con = cnx;
    }
    public String processInput(String theInput) {
     ...
    }
    private String runQuery(String query) {
     ...
    }
}
```

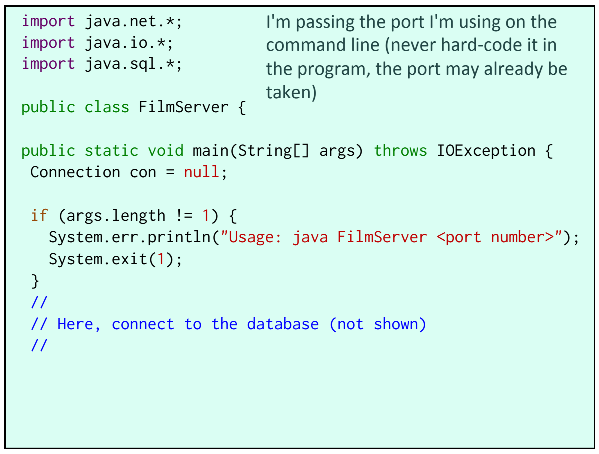
My protocol parses (analyzes) the message, builds the suitable query and runs the query against the database.



2 Write the server

Just a big loop

The server just waits for queries, and executes them.



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

public class FilmServer {

public static void main(String[] args) throws IOException {
 Connection con = null;

 if (args.length != 1) {
   System.err.println("Usage: java FilmServer <port number>");
   System.exit(1);
 }
 //
 // Here, connect to the database (not shown)
 //
```

I'm passing the port I'm using on the command line (never hard-code it in the program, the port may already be taken)

```
FilmProtocol   filmP = new FilmProtocol(con);
int            portNumber = Integer.parseInt(args[0]);
String         inputLine, outputLine;
PrintWriter    out = null;
BufferedReader in = null;
ServerSocket   serverSocket = null;
Socket         clientSocket = null;

try {
  serverSocket = new ServerSocket(portNumber);
  System.err.println("Film server started on port "
                                   + args[0]);
  while (true) {
    clientSocket = serverSocket.accept();
    System.err.println("Accepted connection");
    out =
       new PrintWriter(clientSocket.getOutputStream(), true);
    in = new BufferedReader(
       new InputStreamReader(clientSocket.getInputStream()));
```

We create a FilmProtocol (that does all the job) then create a socket and loop forever (we should have a way to stop it cleanly in real life)

Autoflush

As said earlier, it's really the FilmProtocol Object that does the tough bits ...

```
    // Wait for input
    if ((inputLine = in.readLine()) != null) {
      outputLine = filmP.processInput(inputLine);
      out.println(outputLine);
    }
    clientSocket.close();
  }
} catch (...) {
   ...
} finally {
  ...
 }
}
}
```

3 Write a client

This one is very basic

Once the server is ready and that we know the protocol, time to write a (command-line here) client. It won't be a sexy program, but it will be functional.

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

public class FilmClient {

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

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

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

The client must be told where to connect.

```java
while (loop) {
  try (
    Socket          sock = new Socket(hostName, portNumber);
    PrintWriter     out =
       new PrintWriter(sock.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
       new InputStreamReader(sock.getInputStream()));
  ) {
    BufferedReader stdIn =
      new BufferedReader(new InputStreamReader(System.in));
    String fromServer;
    String fromUser;
```

As any query is independent (there is no "session") I'm creating (and closing) one connection for each query. Better suited to my server in that case.

```java
    System.out.print("Query> ");
    // read from input
    fromUser = stdIn.readLine();
    // send to server
    out.println(fromUser);
    // read from server
    while ((fromServer = in.readLine()) != null) {
      if (fromServer.length() > 0) {
        System.out.println(fromServer);
      }
      if (fromServer.equals("Goodbye")) {
        loop = false;
        break;
      }
    }
```

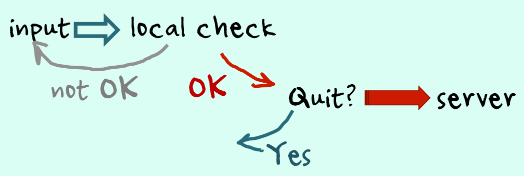
When I exit the server sends an acknowledgement. Note that I just expect raw data from the server (an error message would be raw data)

```java
  } 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 to " + hostName);
    System.exit(1);
  }
 }
}
}
```

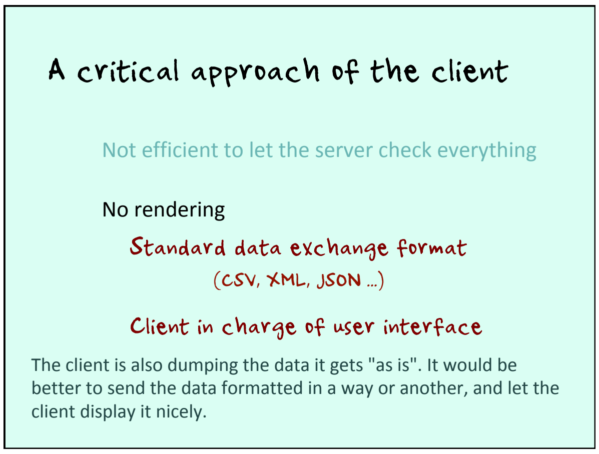
Handling here connection errors.

A critical approach of the client

Not efficient to let the server check everything



The client is very dumb. It would be better if it knew the protocol and could check before sending if the message is correct. It would give less work to the server and use a little less bandwidth.



A critical approach of the client

Not efficient to let the server check everything

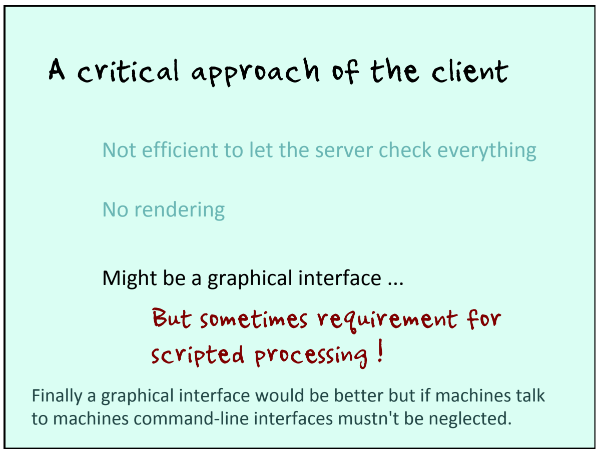
No rendering

Standard data exchange format

(CSV, XML, JSON ...)

Client in charge of user interface

The client is also dumping the data it gets "as is". It would be better to send the data formatted in a way or another, and let the client display it nicely.



A critical approach of the client

Not efficient to let the server check everything

No rendering

Might be a graphical interface ...

But sometimes requirement for scripted processing !

Finally a graphical interface would be better but if machines talk to machines command-line interfaces mustn't be neglected.