Lecture 11

Multi Tasking

Multi-tasking

Need to coordinate

Need to communicate

Not very easy with processes if you think about it...

Java Threads are lightweight processes!

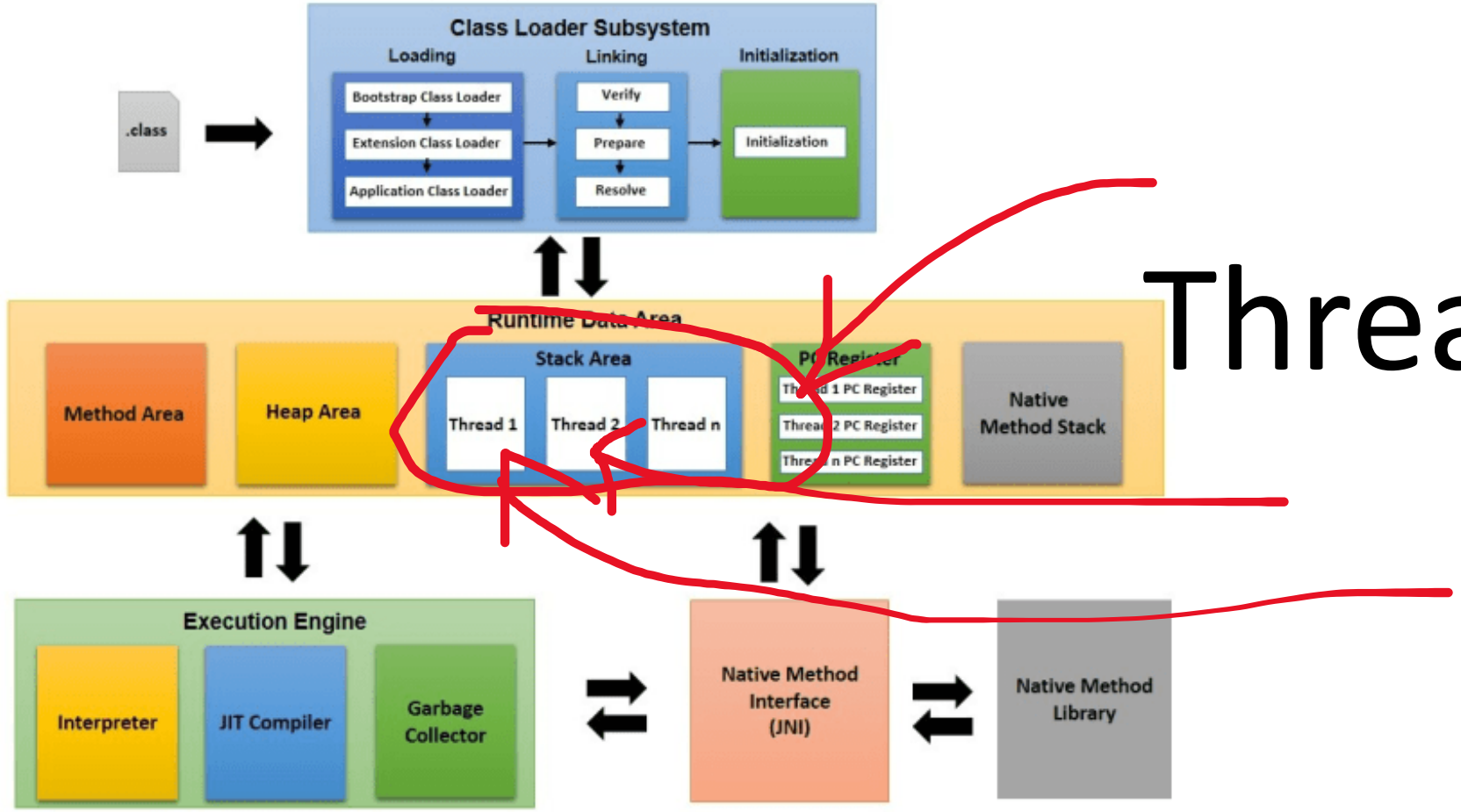
Process

Runs a program

Thread

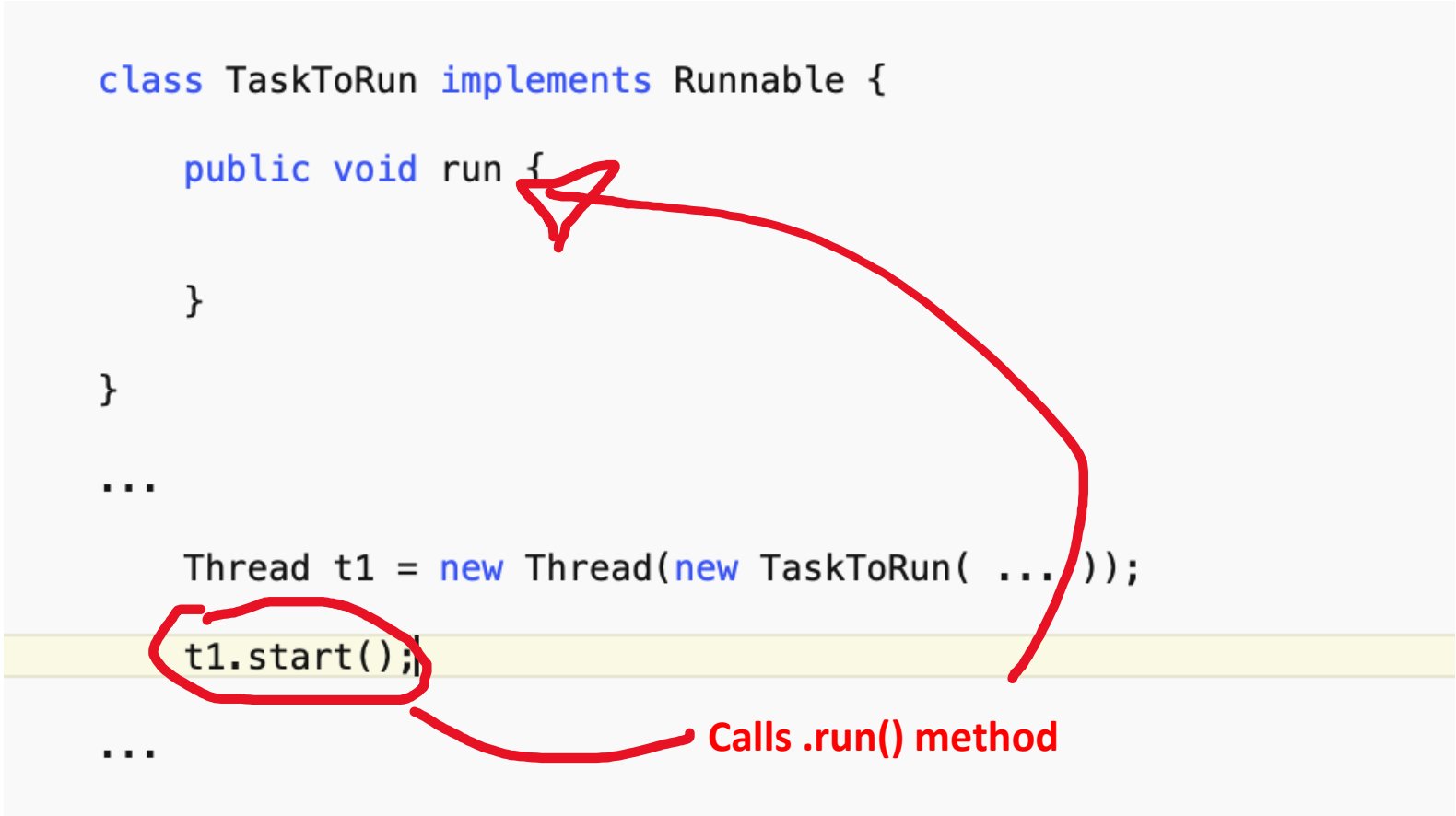
Runs independently in
a method in a program

If you remember the JVM, it can support multiple threads. Not three or four like the diagram could suggest: it can support thousands of threads.



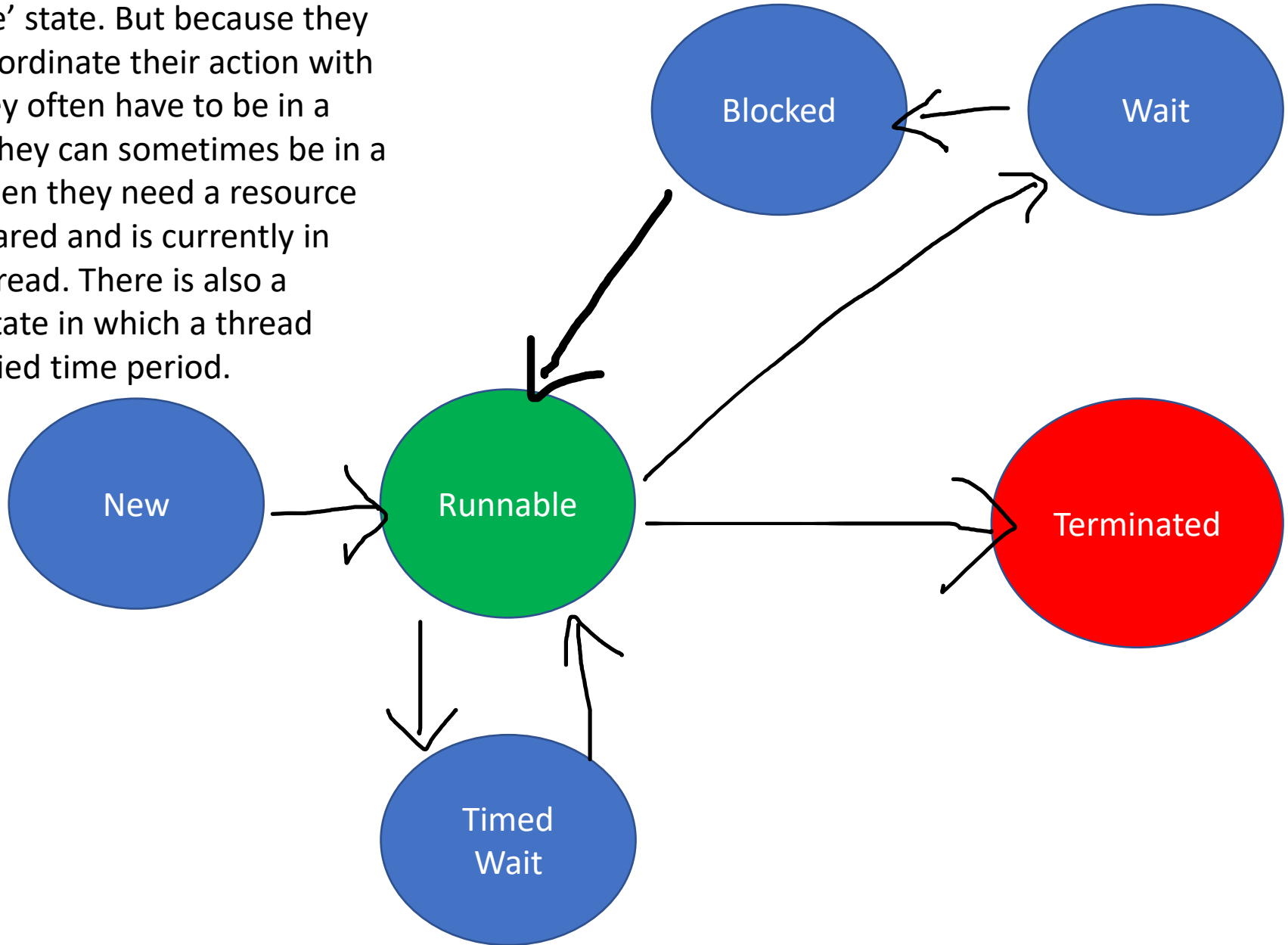
The Thread Class

java.util.concurrent



- Threads are java objects that can call the run() method of an object that implements Runnable.
- Once “run” the object works independently as a separate procedure.

Threads can be in multiple states. When you create them, they are in a ‘New’ state. When you call their start() method, they get into the ‘Runnable’ state. But because they usually have to coordinate their action with other threads, they often have to be in a ‘Wait’ state, and they can sometimes be in a ‘Blocked’ state when they need a resource that cannot be shared and is currently in use by another thread. There is also a “timed waiting” state in which a thread sleeps for a specified time period.



Thread States

- New
- Runnable
- Blocked
- Waiting
- Time waiting
- Terminated

- We will see a possible sequence of passing through the states in the server example from the previous presentation

There are many cases where you want independent tasks to be performed within the same program.

Network Listener (server)

Let’s take the example of a network listener. If it processes a request (which may be something such as building a webpage from the result of a database query), it will be unable to handle requests arriving meanwhile.

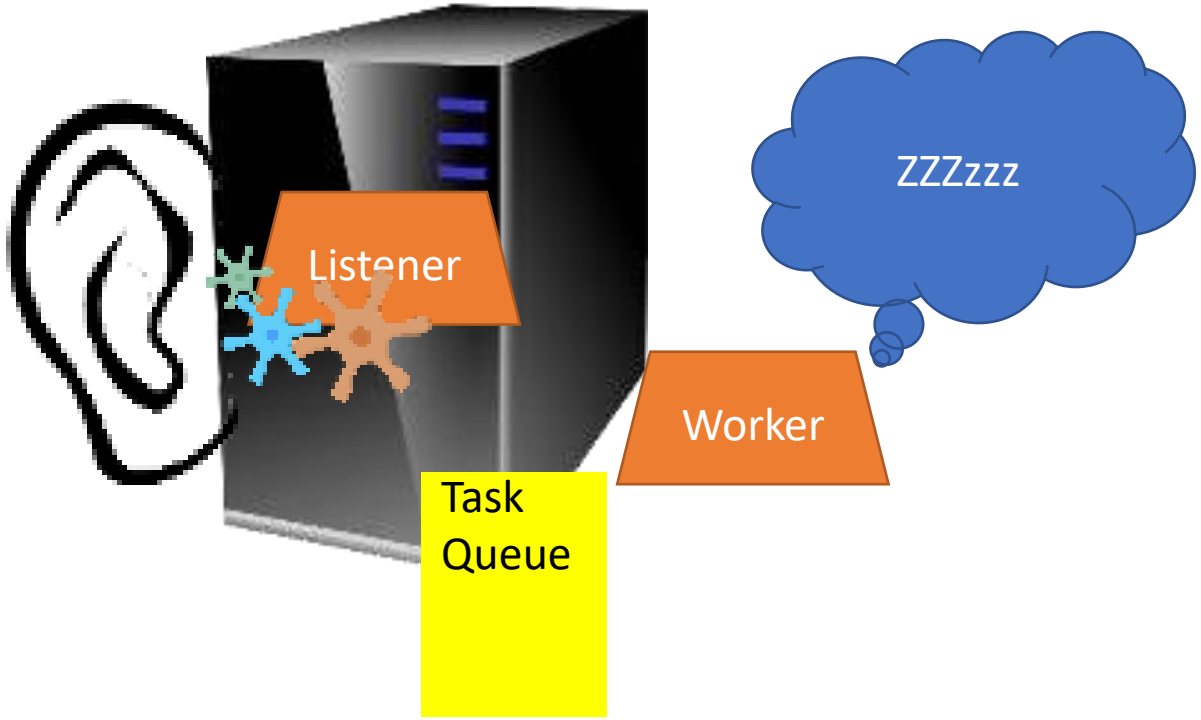
These requests will probably time out if they don’t get a response from the busy listener within a short timeframe

Listener and worker have nothing to do until a request arrives. They basically have two ways of waiting, calling a wait() method that makes them wait until something happens, or sleep for a fixed amount of time.

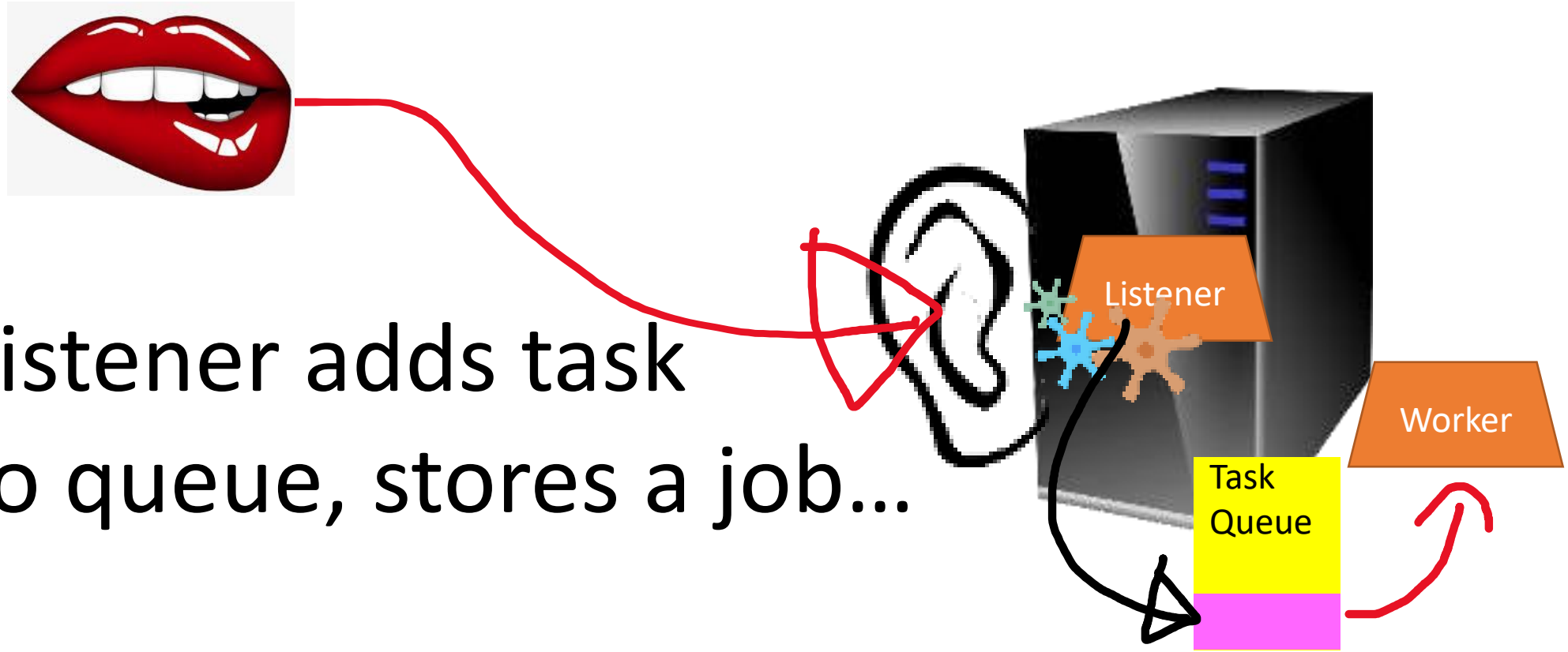
TimedWaiting

Runnable

Waiting



When the listener has received and stored something, then the worker thread must get to work.



Coordination patterns

Listener

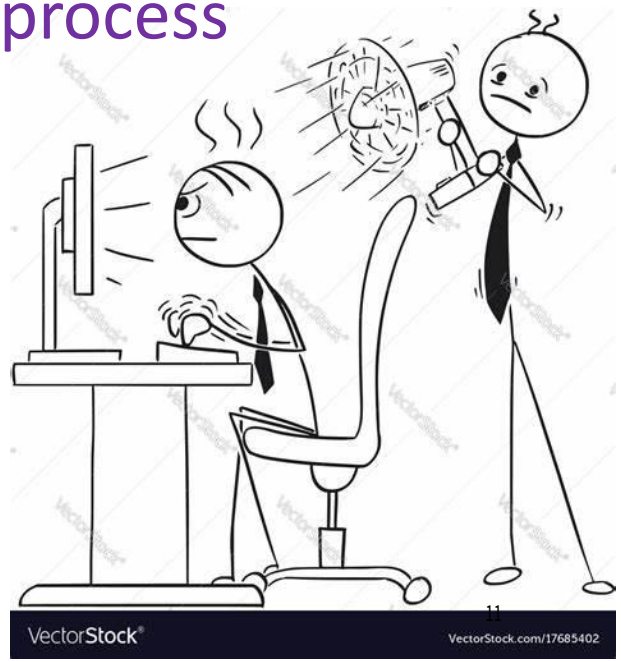
call worker.notify()

Worker

Calls listener.wait()

// do process

…



There are a few different ways for coordinating the processes.

- One is to make the Worker call the wait() method of the Listener. Then the worker will be made to wake up when the Listener calls its notify() method.

Coordination patterns

Listener

```
worker.interrupt();
```

Worker

```
Thread.sleep(3600000);

catch (InterruptedException e) {
        // Do Process
}
while (this.interrupted()) {
        // Do Process
}
```

There are a few different ways for coordinating the processes.

- One is to make the Worker call the wait() method of the Listener. Then the worker will be made to wake up when the Listener calls its notify() method.
- Or the worker can sleep (timed waiting) until the listener can throw a special exception to the worker thread and wake it up (interrupt). Because several interruptions could be generated, the “interrupted” state should be checked in a loop.

Coordination patterns

Listener

There are a few different ways for coordinating the processes.

- One is to make the Worker call the wait() method of the Listener. Then the worker will be made to wake up when the Listener calls its notify() method.
- Or the worker can sleep (timed waiting) until the listener can throw a special exception to the worker thread and wake it up (interrupt). Because several interruptions could be generated, the “interrupted” state should be checked in a loop.
- Or the Worker may ignore the Listener, check by itself, and keep on checking until it’s interrupted – which means no more work to expect.
- This is called “Polling”.

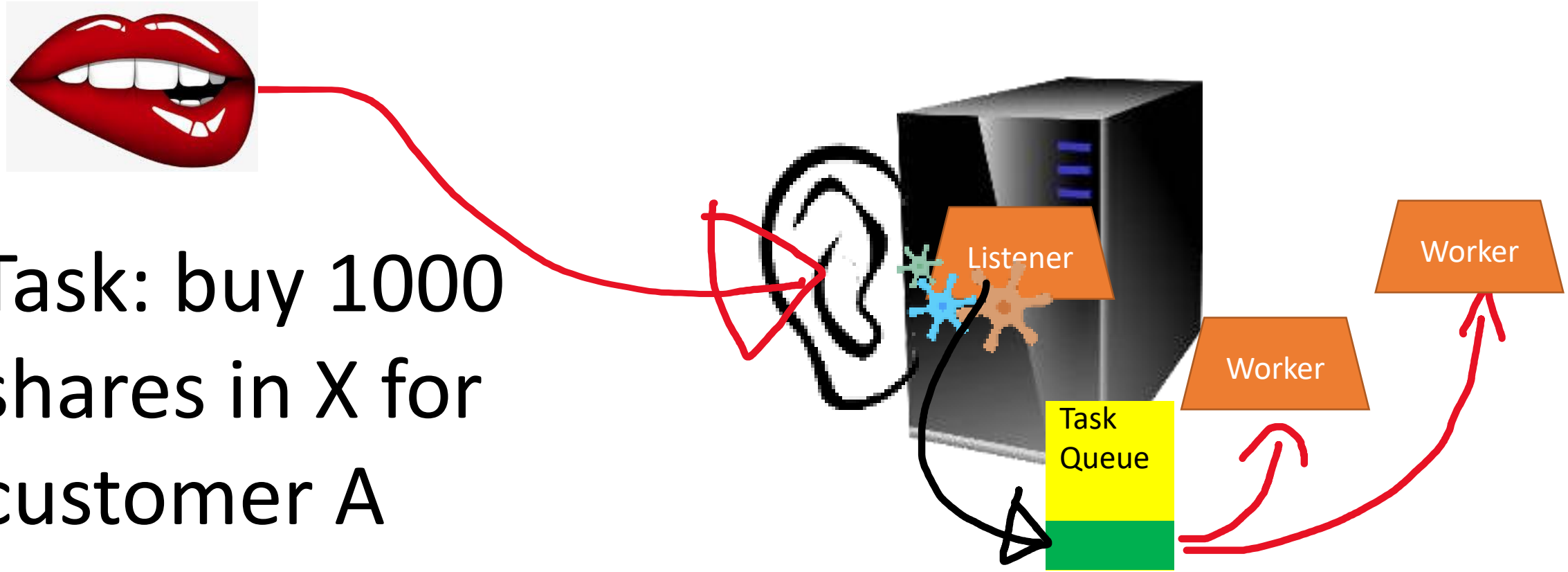
Worker

```
while (true) {
    try {
        Thread.sleep(1000);
        if ( /*something to do */ )
            // Just Do It
        }
    } catch (InterruptedException e) {
        break;
    }

}
```

`interrupt()`

- This method is used to get the attention of another thread
- Very useful! You can interrupt a thread that works for far to long
- Usually the way exceptions work is that some method calls another one and then “throw” sends an exception to the caller.
- Here it is an exception that is “thrown” to a different thread - it comes from the outside (not from the call hierarchy)

However, coordination is a bit more complicated: we have a shared resource, which is the queue. You want every task to be processed once, not several times by Workers ignorant of each other.



```
public static void main (String [] args) {
    for (int i = 1; i<11; i++) {
        tasks.add("task-" + i);
    }

    int numT = Integer.valueOf(args[0]);
    for (int i = 1; i <= numT ; i++) {
        Thread t = new Thread(new Worker ("T-"+i));
        t.start();
    }


}
```

Each thread is now running separately, it will choose a task from tasks to process

Race Conditions

If we are not careful two or more workers may check the queue at the same time, see the same task and process it. This is called a race condition.

$ javac NaïveQueue
$ java NaiveQueue

```
T-3 processing task-1
T-2 processing task-1
T-4 processing task-10
T-5 processing task-2
T-1 processing task-3
T-4 processing task-4
T-3 processing task-6
T-5 processing task-5
T-2 processing task-4
T-1 processing task-7
T-4 processing task-8
T-2 processing task-9
T-3 processing task-9
T-5 processing task-8
```

Synchronized

It works!

```
static PriorityQueue<String> tasks = new PriorityQueue<String>();


public static synchronized String checkTask() {
    String task = null;

    try {
        task = tasks.remove();
    } catch (Exception e) {
        task = null;
    }
    return task;
}
```

```
T-3 processing task-1
T-1 processing task-4
T-4 processing task-10
T-2 processing task-3
T-5 processing task-2
T-5 processing task-5
T-1 processing task-9
T-3 processing task-8
T-2 processing task-7
T-4 processing task-6
```

By making the checkTask method synchronized only one thread at a time can enter this piece of code (that is the method checkTask)

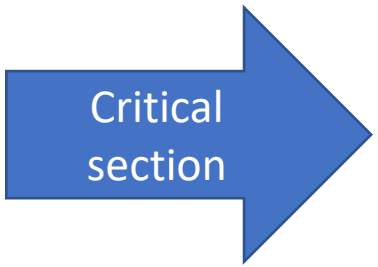
```
static class Worker implements Runnable{

    String myName;
    Random rand;
    public Worker  (String name) {
        myName = name;
        rand = new Random(28);
    }

    public void run () {

        String task;
        while (true) {
            try {
                // sleep for 1/2 a second
                Thread.sleep(500);
                task = Synchro.checkTask();
                if(task != null) {
                    System.out.println(myName + " processing " + task);
                    Thread.sleep(rand.nextInt(1000));
                }

            } catch (InterruptedException e) {
                break;
            }
        }
        System.err.println(myName + " stopping");
    }
}
```

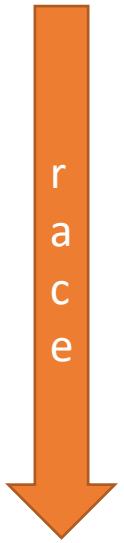
The threads sleep and periodically “wake up” to check if there are new tasks in tasks. They may be interrupted in which case they will break out of the sleep wak cycle and print a message to System.err.

Critical sections

- Critical sections are any pieces of code that accesses shared resources
- As a result of sharing, the result of the code depends on the order that threads gained access to the resources
- Critical sections are handled in java by using the synchronized key word
- Critical sections can be handled as blocks of code {…}, methods or even classes

```
public class CriticalSectionDemo{
        private int i = 0;

        public int incrementValue() {
                System.out.println("Current Thread " + Thread.currentThread().getName());
                try {
                        Thread.sleep(300);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                return this.i++;
        }
        public static void main(String[] args) {
                CriticalSectionDemo demo = new CriticalSectionDemo();
                new Thread(() -> demo.incrementValue()).start();
                new Thread(() -> demo.incrementValue()).start();
                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                System.out.println(demo.i);
        }
}
```





We don’t know which thread reaches the line this.i++ first

```
C:\Users\adamg\Desktop\Java Systems 2021>java CriticalSectionDemo
Current Thread Thread-0
Current Thread Thread-1
2

C:\Users\adamg\Desktop\Java Systems 2021>java CriticalSectionDemo
Current Thread Thread-0
Current Thread Thread-1
2

C:\Users\adamg\Desktop\Java Systems 2021>java CriticalSectionDemo
Current Thread Thread-0
Current Thread Thread-1
2

C:\Users\adamg\Desktop\Java Systems 2021>java CriticalSectionDemo
Current Thread Thread-0
Current Thread Thread-1
2

C:\Users\adamg\Desktop\Java Systems 2021>java CriticalSectionDemo
Current Thread Thread-1
Current Thread Thread-0
2
```

```
private int i = 0;

        public int incrementValue() {
                System.out.println("Current Thread " + Thread.currentThread().getName());
                try {
                        Thread.sleep(300);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                return this.i++;
        }
```

Critical section of the code where the sequence of execution by different threads will change program behaviour

```
public synchronized int incrementValue() {
                System.out.println("Current Thread " + Thread.currentThread().getName());
                try {
                        Thread.sleep(300);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                return this.i++;
        }
```

Class level locks are applicable for static synchronized methods and blocks.

Now the second thread has to wait for the first to finish working in the method increment value…

Thread-0 always finishes first.

```
C:\Users\adamg\Desktop\Java Systems 2021>javac CriticalSectionDemo.java

C:\Users\adamg\Desktop\Java Systems 2021>java CriticalSectionDemo
Current Thread Thread-0
Current Thread Thread-1
2

C:\Users\adamg\Desktop\Java Systems 2021>
C:\Users\adamg\Desktop\Java Systems 2021>java CriticalSectionDemo
Current Thread Thread-0
Current Thread Thread-1
2

C:\Users\adamg\Desktop\Java Systems 2021>java CriticalSectionDemo
Current Thread Thread-0
Current Thread Thread-1
2

C:\Users\adamg\Desktop\Java Systems 2021>java CriticalSectionDemo
Current Thread Thread-0
Current Thread Thread-1
2

C:\Users\adamg\Desktop\Java Systems 2021>java CriticalSectionDemo
Current Thread Thread-0
Current Thread Thread-1
2

C:\Users\adamg\Desktop\Java Systems 2021>java CriticalSectionDemo
Current Thread Thread-0
Current Thread Thread-1
2

C:\Users\adamg\Desktop\Java Systems 2021>java CriticalSectionDemo
Current Thread Thread-0
Current Thread Thread-1
2

C:\Users\adamg\Desktop\Java Systems 2021>
```

```
public class CriticalSectionDemo{
        private int i = 0;

        public synchronized int incrementValue() {
                System.out.println("Current Thread " + Thread.currentThread().getName());
                try {
                        Thread.sleep(300);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                return this.i++;
        }
        public static void main(String[] args) {
                CriticalSectionDemo demo = new CriticalSectionDemo();
                new Thread(() -> demo.incrementValue()).start();
                new Thread(() -> demo.incrementValue()).start();
                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                System.out.println(demo.i);
        }
}
```

```
public int incrementValue() {
        System.out.println("Current Thread " + Thread.currentThread().getName());
        try {
                Thread.sleep(300);
        } catch (InterruptedException e) {
                e.printStackTrace();
        }
        synchronized (this) {
                return this.i++;
        }
}
```

- We can also make just a part of a method synchronized depending on the situation
- This will not stop the race condition here but will prevent the line from being executed simultaneously

The first arrived must **block** the others

Make it Synchronized

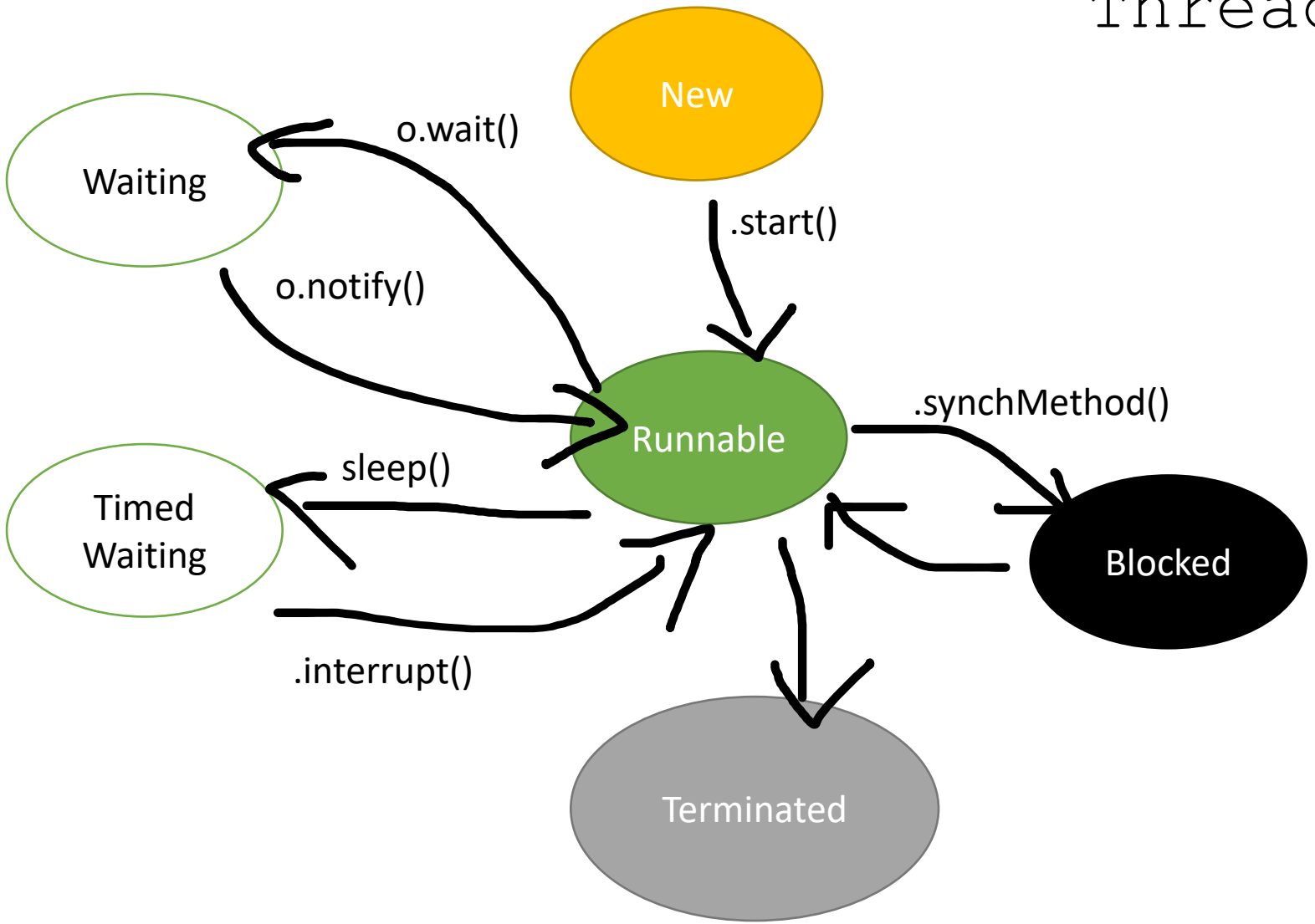
If we want to avoid this we want to make sure that on only one Worker reads from the queue at the same time (and in fact we should check that it’s not reading at exactly the same time as the Listener is writing, which could be messy too). In Java there is a mechanism called synchronization that guarantees that only one thread can access a synchronized resource at the same time.

The first arrived must **block** the others

synchronized *type* methodName(...) { }

You just declare the method to be synchronized. It means that the first thread to execute it will lock it. Other threads will block, and (each in turn) will be able to execute the method when the first thread returns from it. Needless to say, it’s a bad idea to synchronize methods that could take hours to run. You should only synchronize pieces of code that should run fast.

Thread State Summary



Built-in Support for Concurrency

- You can also use synchronized collections in which the main methods (e.g. remove) are synchronized already. A similar concept to the observable lists we used with JavaFX.
- Java has built-in mechanisims for collections
  - Special wrappers designed for concurrency which you can call from the Collections class:
    - EG: List list = Collections.synchronizedList(new ArrayList());
  - OR you can use purpose built collections:
    - EG a FIFO queue: java.util.concurrent.ArrayBlockingQueue<E>

Built-in Support for Concurrency

- Using a synchronized collection means that YOU don’t need to write a synchronized method. There are however still plenty of cases where you might want to – think of a counter of tasks completed increased by one by every Worker that is done with a task ...
- You may also have synchronization or not with Strings, using either a StringBuilder (not synchronized) or a StringBuffer (synchronized) object. Avoid a synchronized version (slower) when not needed.

Work Safety

WARNING : Deadlocks

- The fact that a thread can be blocked by another one is good, as long as one of the threads c or less long time, complete what it has to do, then release the locks and unblock any other thread waiting. But you can have problems, because of multiple locks th eld at the same moment by different thread.
- Because locks are associated with objects, **different objects** can call the same synchronized method at the same time!
- There would be no problem with a synchronized static method, because then the (single) lock class would be held by only one thread. But when the methods are attached to different objects, the code can be executed at the same time and the locks held by two distinct threads.

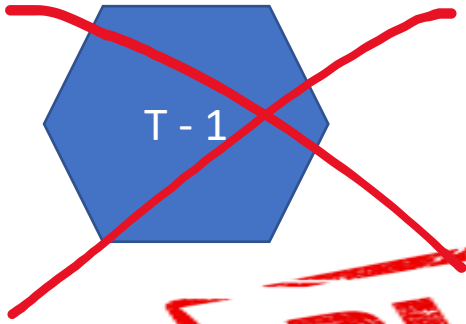
Deadlocks

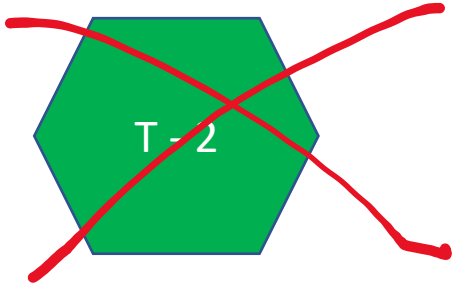
Gets the lock of o1

Gets the lock of o2

```
call synchronized o1.m1() {
   call synchronized o1.m2();
}
```

```
call synchronized o2.m1() {
   call synchronized o2.m2();
}
```





Example: Two people try to transfer money from their account to the other's account at the same time…

```
class Account {
    private int accountId;
    private float balance;

    public Account(int id, float amount) {
        accountId = id;
        balance = amount;
    }

    public synchronized void credit(float amount) {
        System.out.println("Thread "
                + Thread.currentThread().getName()
                + " crediting account #"
                + Integer.toString(accountId)
                + " of " + Float.toString(amount));
            balance += amount;
    }

}
```

I’m synchronizing every sensitive method.

The bank doesn’t want to see an account credited twice.

```
    }

    public synchronized boolean debit(float amount) {

        System.out.println("Thread "
        + Thread.currentThread().getName()
        + " debiting account #"
        + Integer.toString(accountId)
        + " of " + Float.toString(amount));

        if (amount < balance) {
            balance -= amount;
            return true;
        } else {
            return false;
        }
    }

}
```

I’m synchronizing every sensitive method.

The bank also doesn’t want to see an account debited twice.

When I’m transferring money from the current account to another account, I’m checking I still have enough on the account, then subtracting the amount from the current balance before crediting the other account with the amount. As I’m already in a synchronized method, there is no need to call this.debit(amount).

```
public synchronized void transfer(Account toAccount,float amount) {
    if (balance >= amount) {
        balance -= amount;
        toAccount.credit(amount);
    }
}
```

Here is my basic money transfer task which is run in a thread.

```
class MoneyTransfer implements Runnable {
    private Account fromAccount;
    private Account toAccount;
    private float moneyToTransfer;

    public MoneyTransfer(Account fromAcnt, Account toAcnt, float amount) {
        fromAccount = fromAcnt;
        toAccount = toAcnt;
        moneyToTransfer = amount;
    }

    public void run() {
        fromAccount.transfer(toAccount, moneyToTransfer);
    }
}
```

```
public class BankingTransaction {
    private final static Account account1 = new Account(1, 1000);
    private final static Account account2 = new Account(2, 500);

    public static void main(String[] args) {
        Thread t1 = new Thread(new MoneyTransfer(account1, account2, (float)500));
        Thread t2 = new Thread(new MoneyTransfer(account2, account1, (float)250));
        t1.start();
        t2.start();
    }
}
```

Now if you run this program, sometimes it will work fine, sometimes it will remain stuck. Random problem, and deadlocks can be hard to debug.

You normally avoid deadlocks by always locking resources in the same order. Here it's a bit different. You might imagine having a third thread watching and interrupting a stuck task ...

.isAlive() and .isJoin()

- Check for thread termination:

  thr.isAlive()

- Wait for thread termination:

  thr.join()

Sometimes the main thread (that started all the other ones running) must gather work done by the others and kind of finalize operations. It can check if a thread is still running, and if this is the case call its .join() method that blocks until the thread has terminated.

```
for (int i = 0; i < threadCount; i++) {
    thr = new Thread(...):
    threadList.add(thr);
    thr.start();

    Iterator iter = threadList.iterator();

    while (iter.hasNext()) {
         thr = iter.next();
    if (thr.isAlive()) {
         thr.join();
    }
}
```

// Now use what threads have done (combining their work)

Here is a simple example of a master thread starting children threads and waiting for the termination of each of them.

starvation

You can set thread priority but it is something to be done with caution.

If a thread is starved of CPU time it is not locked but it also cannot make progress.

Safe Threads

- It is better to use immutable objects
  - All attributes private and final
  - No setters
  - class final (means it cannot be extended)

- To avoid problems it is recommended that you use immutable objects in threads

Networking Intro

Client – Server Programming

Network Programming

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.

Client Socket ← → Server Socket

Stream = like a file

- java.net.*
- class Socket
- class ServerSocket

Class Socket

Talk to one server

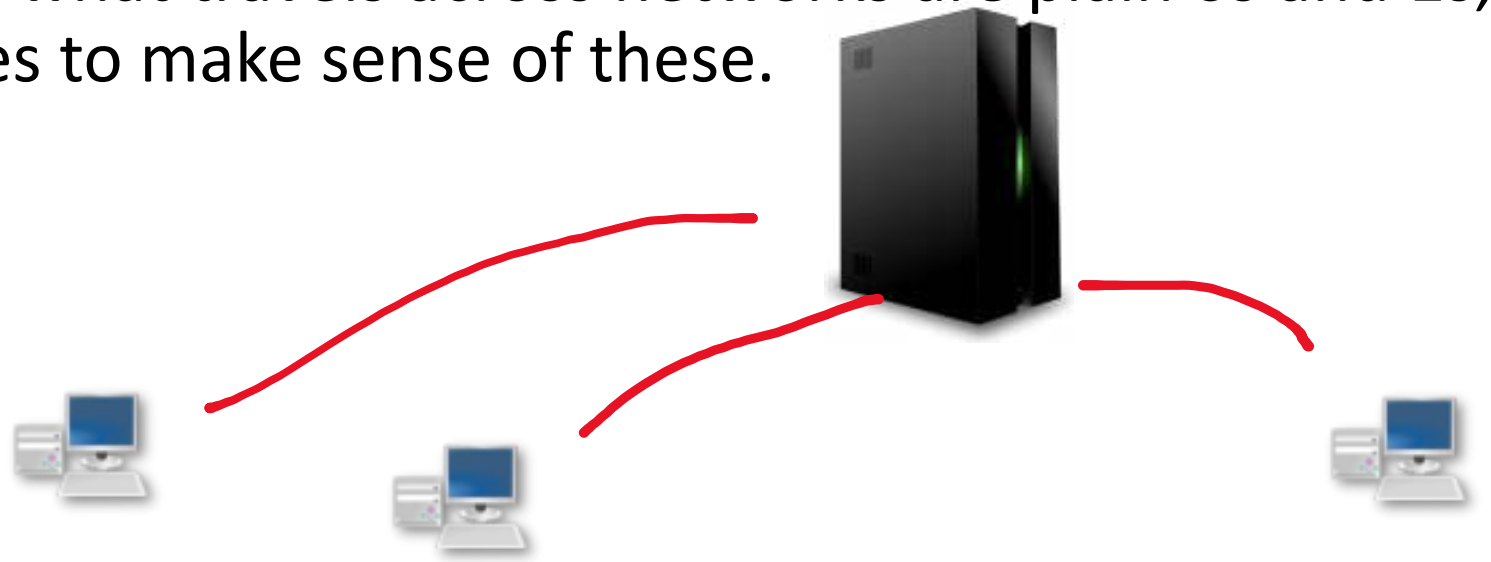
Class Server Socket

Talk to multiple servers – or used to serve multiple clients

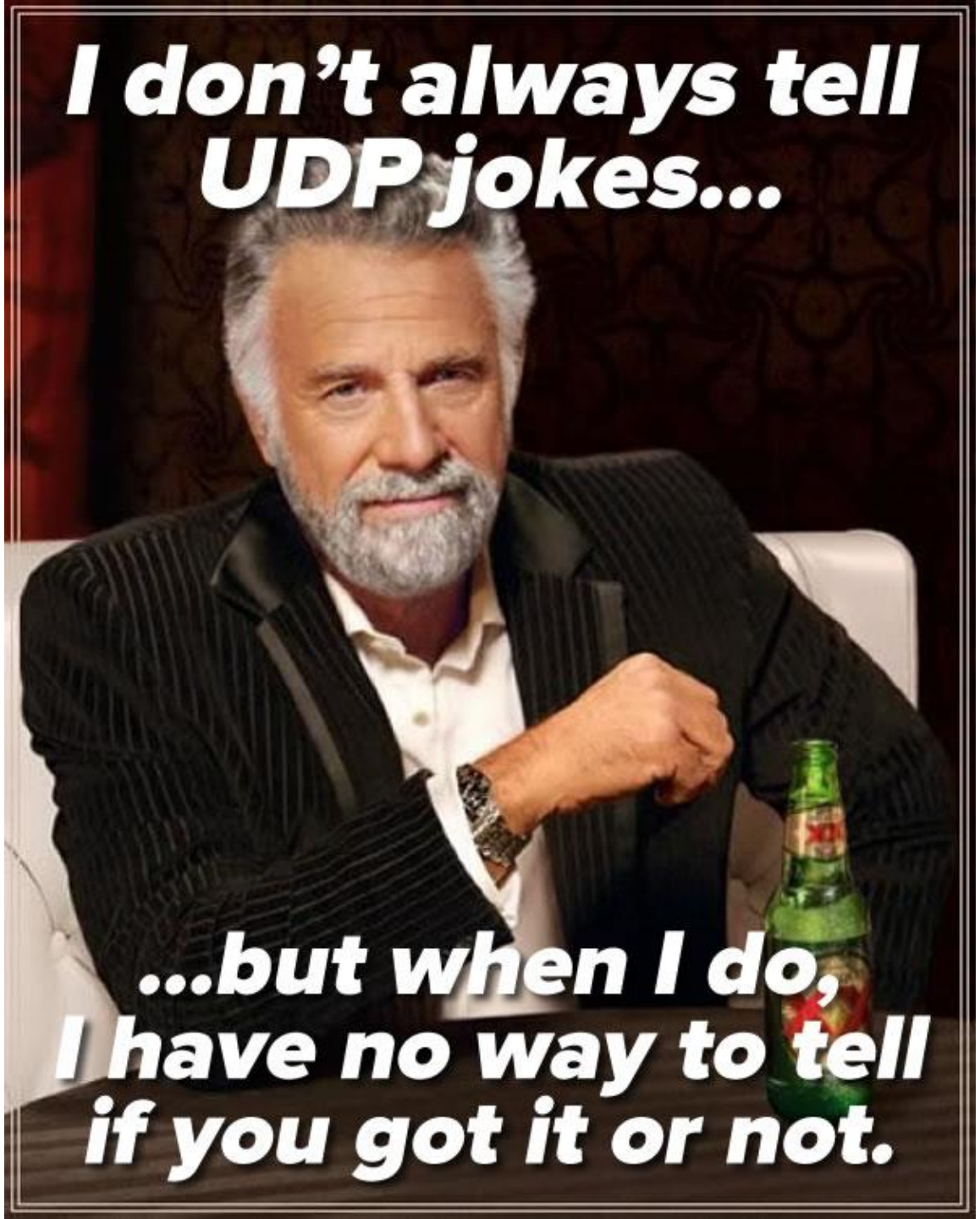
Network Programming

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.



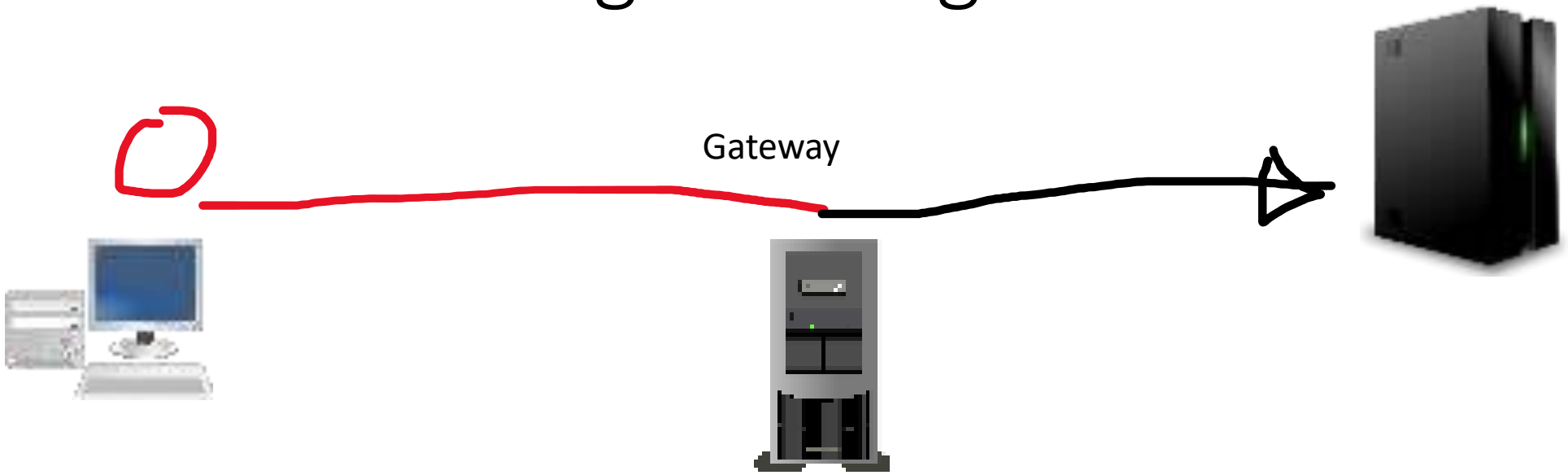
Protocols


I don't always tell UDP jokes...
...but when I do, I have no way to tell if you got it or not.

Network Programming

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.

Network Programming



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.

Rules = PROTOCOLS

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").

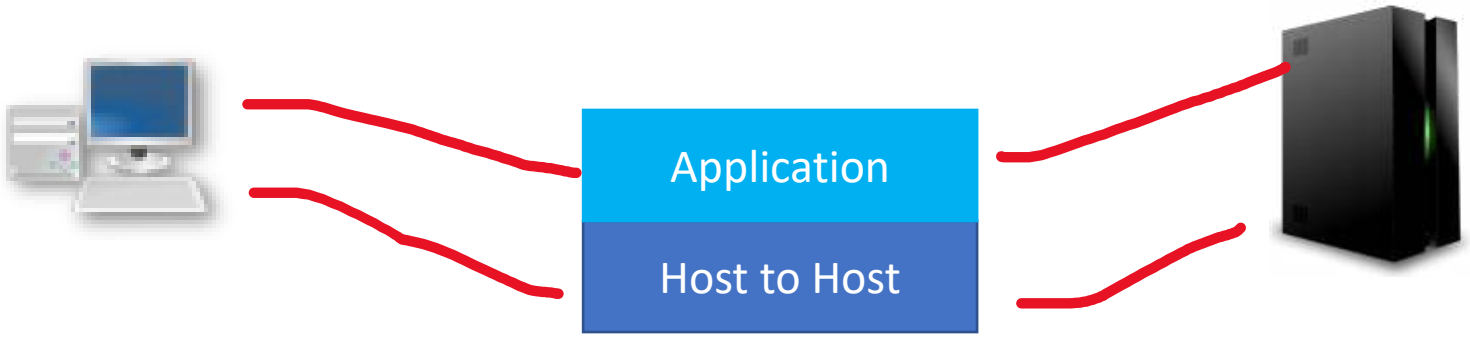
Rules = PROTOCOLS

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").

Ports are like mail boxes in a building full of independent apartments…

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.

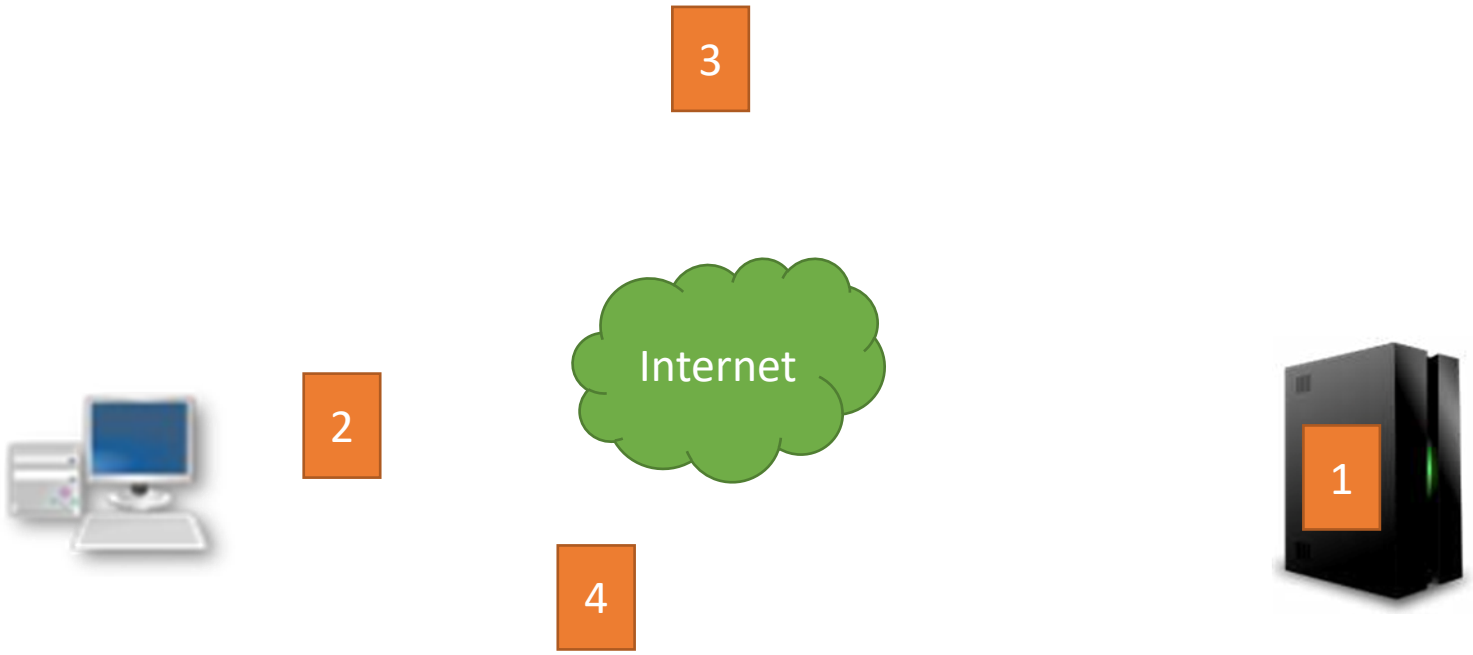
TCP



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 the TCP (= Transmission Control Protocol).

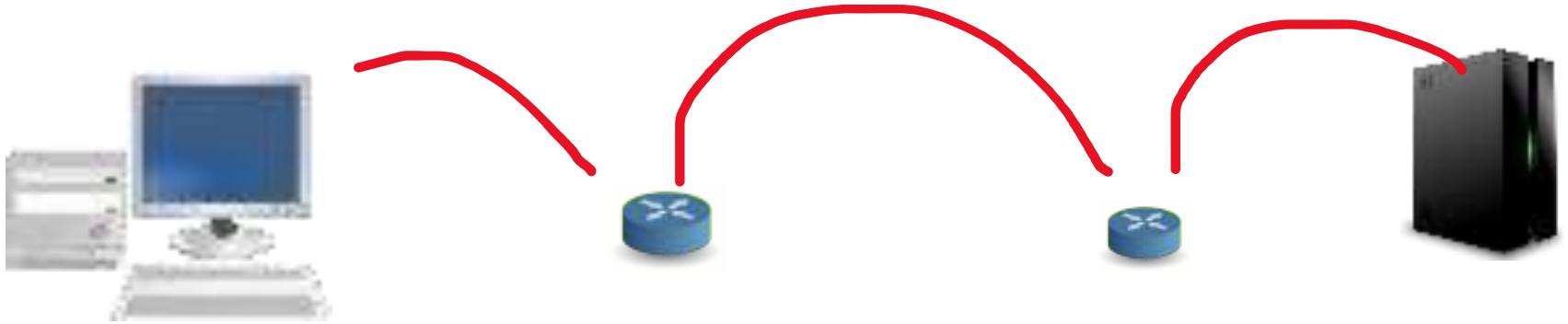
In TCP there is a connection line between the server and the computer (messages are guaranteed to arrive in sequence).

UDP



- UDP (User Datagram Protocol) is another protocol
- It has less overhead and packets (parts of messages) are sent without considering order (even if a packet is sent first it may arrive last depending on the network

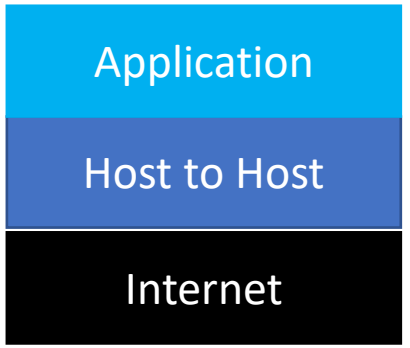
IP



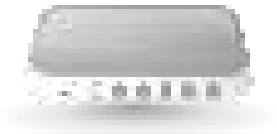
TCP relies on another set of rules that ensure that your message travels from your network to the target network:

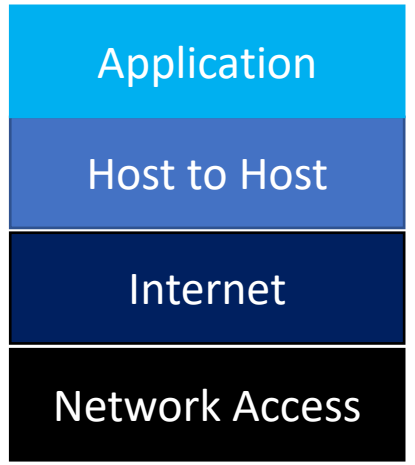
This is the IP or Internet Protocol

Internet just means between networks



Network Access





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

How does it work?

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.

Socket

```
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 nothing more than a byte stream. As a socket works in both directions, you have both an input and an output stream.

Sending and Listening for Messages

```
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). However YOUR application needs a “protocol": messages and answers probably should have a meaning!

More Examples from JavaNotes 8.1

Read section 11.4 Networking:
http://math.hws.edu/eck/cs124/javanotes8/c11/s4.html

Example of a trivial client server
http://math.hws.edu/eck/cs124/javanotes8/c11/s4.html#IO.4.4

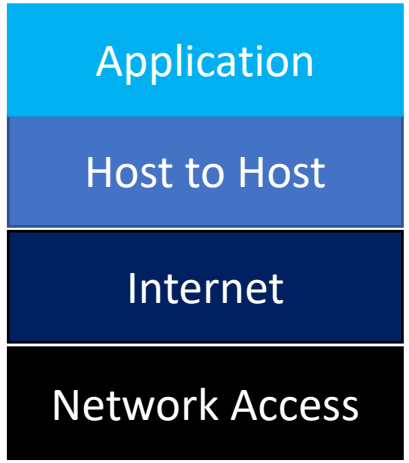
Example a simple network chat (exchanging messages between a client and a server):

http://math.hws.edu/eck/cs124/javanotes8/c11/s4.html#IO.4.5

Network Programming

So far…

- Computers that can “talk” to one another
- Stack of protocols (rules)



- Programming with Sockets

Sockets…

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.

client socket

server socket

import java.net.*

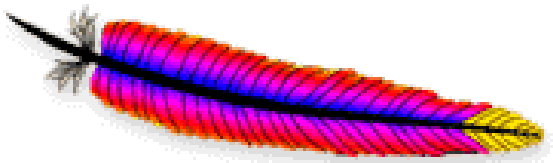
class Socket — talks to ONE server

class ServerSocket — talks to multiple clients

Example: Get the Home Page of a Website

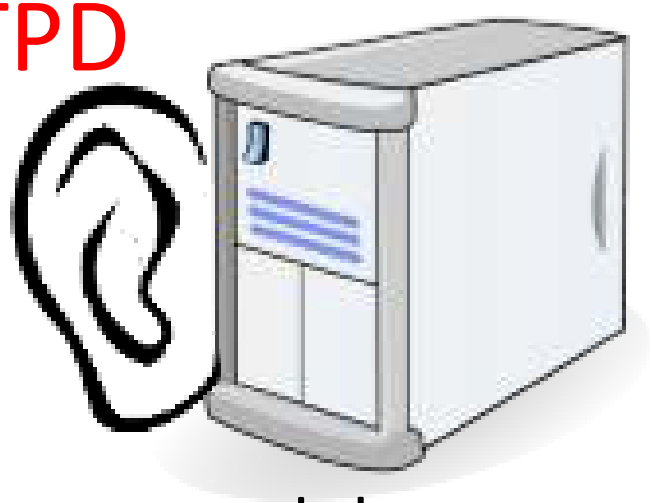
A simple example is getting the home page of any website. Your browser uses a protocol the name of which must be familiar to you: HTTP, or Hyper-Text Transfer Protocol. It sends messages that must be understandable to a server, and the server sends back "web pages", using a protocol that must be understandable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

Server Example with Apache Daemon

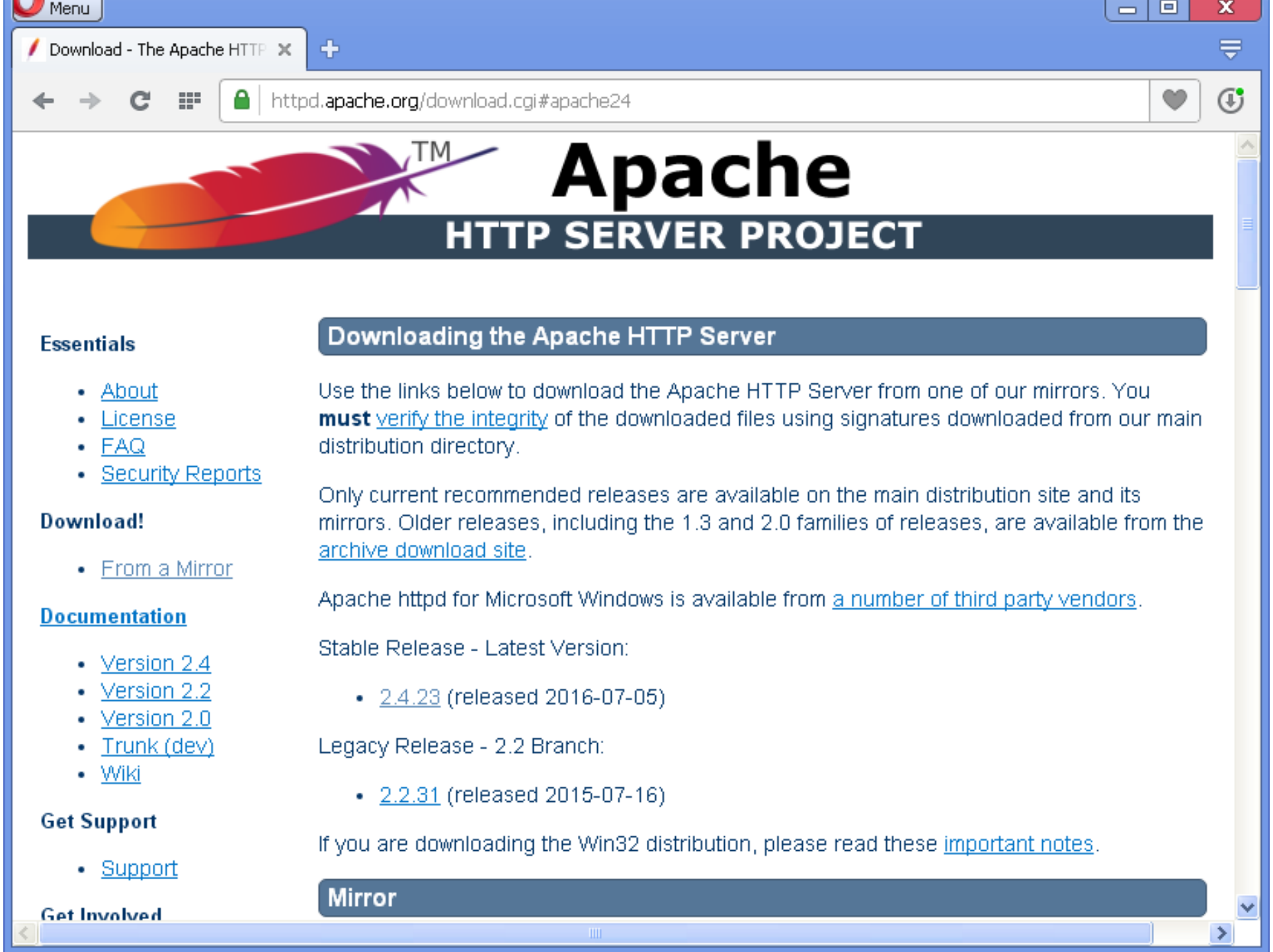


Apache Commons™
http://commons.apache.org/

HTTPD

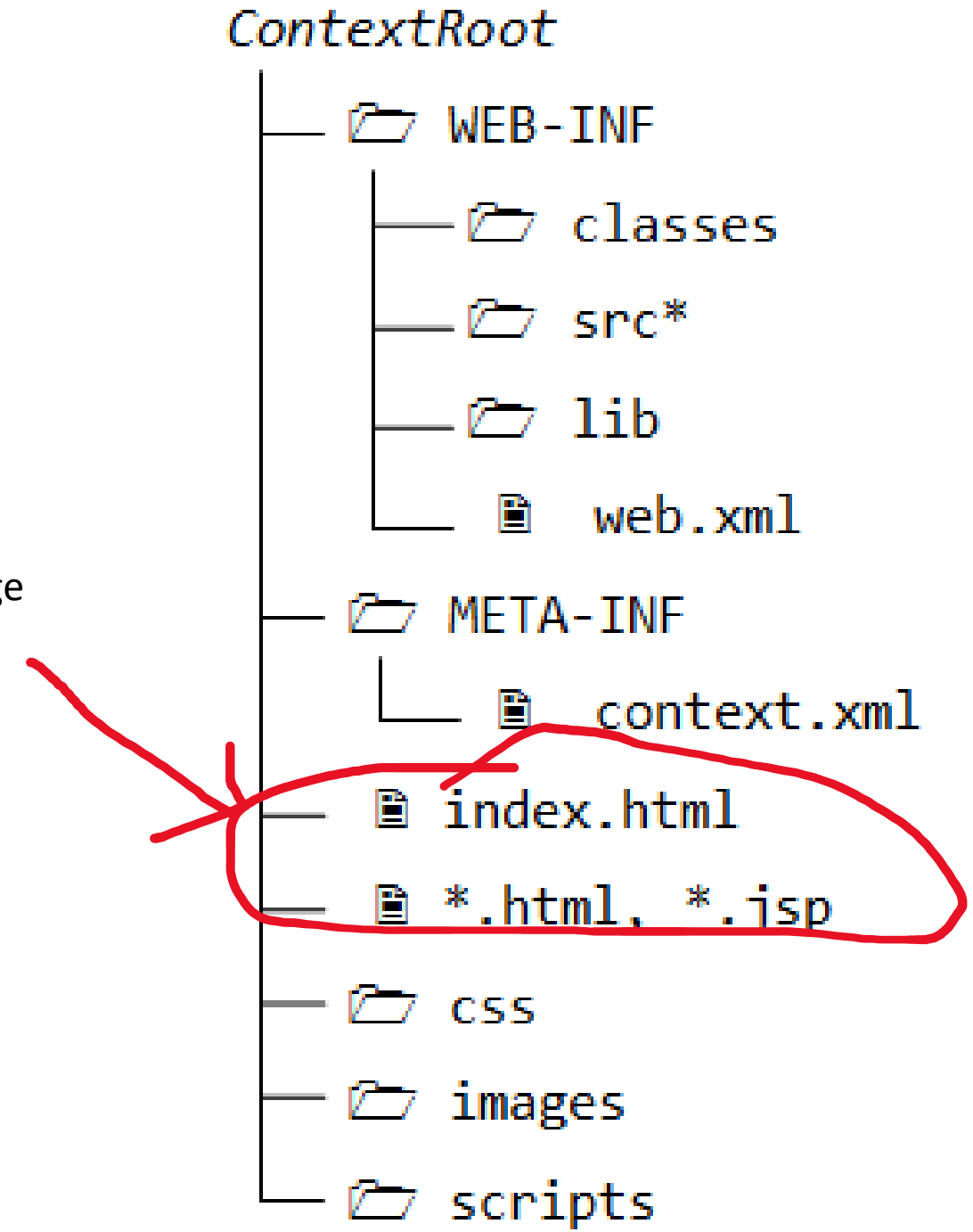


In the case of HTTP, I have represented the server by an Apache server ("Apache" is a very popular web server). The program that runs is called "httpd", the "d" is often used in Unix systems to indicate that a program runs in the background without interacting with the screen nor the keyboard (these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


Menu
Download - The Apache HTTP
httpd.apache.org/download.cgi#apache24
Apache
HTTP SERVER PROJECT
Essentials
About
License
FAQ
Security Reports
Download!
From a Mirror
Documentation
Version 2.4
Version 2.2
Version 2.0
Trunk (dev)
Wiki
Get Support
Support
Get Involved
Downloading the Apache HTTP Server
Use the links below to download the Apache HTTP Server from one of our mirrors. You must verify the integrity of the downloaded files using signatures downloaded from our main distribution directory.
Only current recommended releases are available on the main distribution site and its mirrors. Older releases, including the 1.3 and 2.0 families of releases, are available from the archive download site.
Apache httpd for Microsoft Windows is available from a number of third party vendors.
Stable Release - Latest Version:
2.4.23 (released 2016-07-05)
Legacy Release - 2.2 Branch:
2.2.31 (released 2015-07-16)
If you are downloading the Win32 distribution, please read these important notes.
Mirror

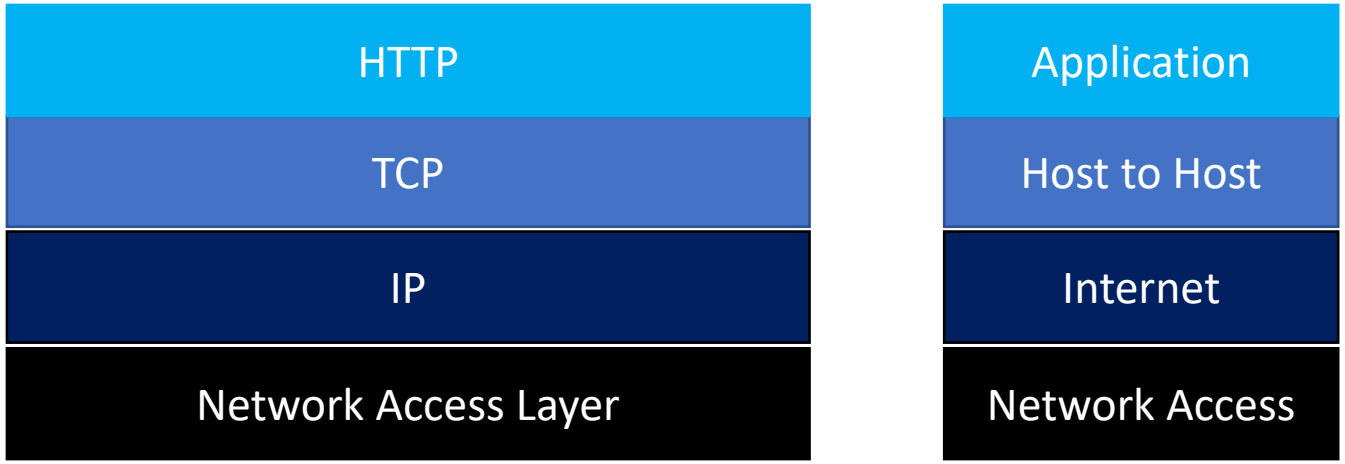
Apache Tomcat

Home Page

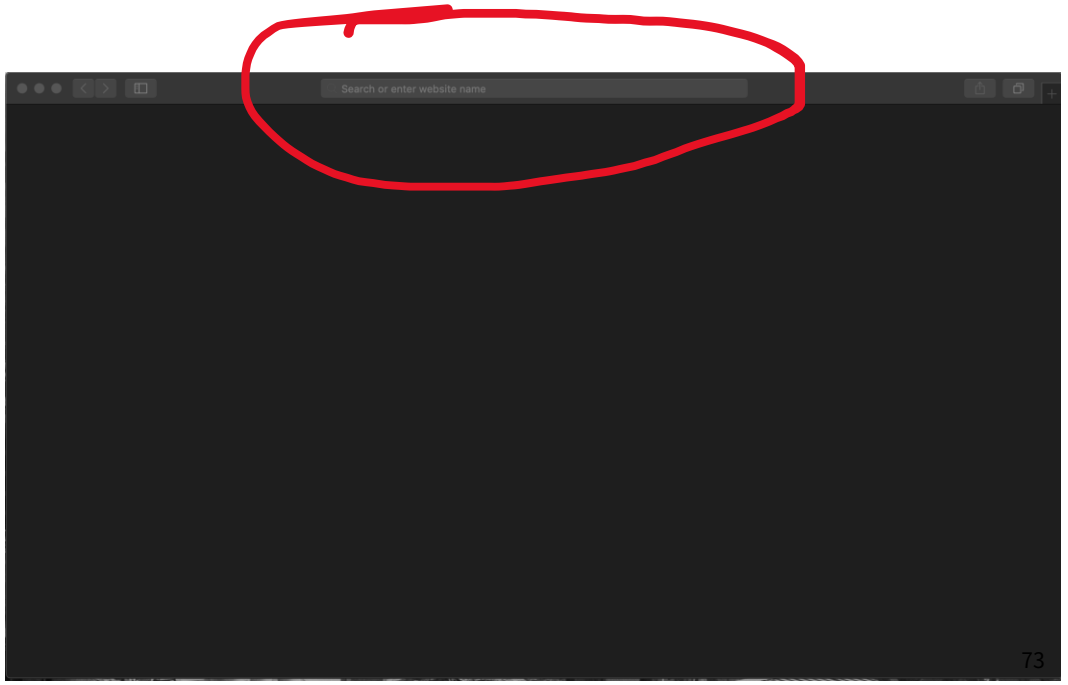


HTTP

HTTP is an application protocol on top of TCP: it consists of just text with special character sequences.



When you type an address in your browser, e.g. the local network address 192.168.254.100 ,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.



A GET Request

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

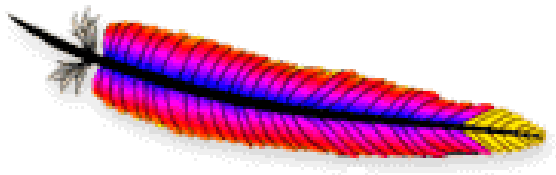
[empty line]

- 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.

Apache Server

- 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.

- Directory structure:
  - var
    - www
      - htdocs
        - .htaccess
        - index.html



Apache Commons™
http://commons.apache.org/

permissions are set using the .htaccess

index.html

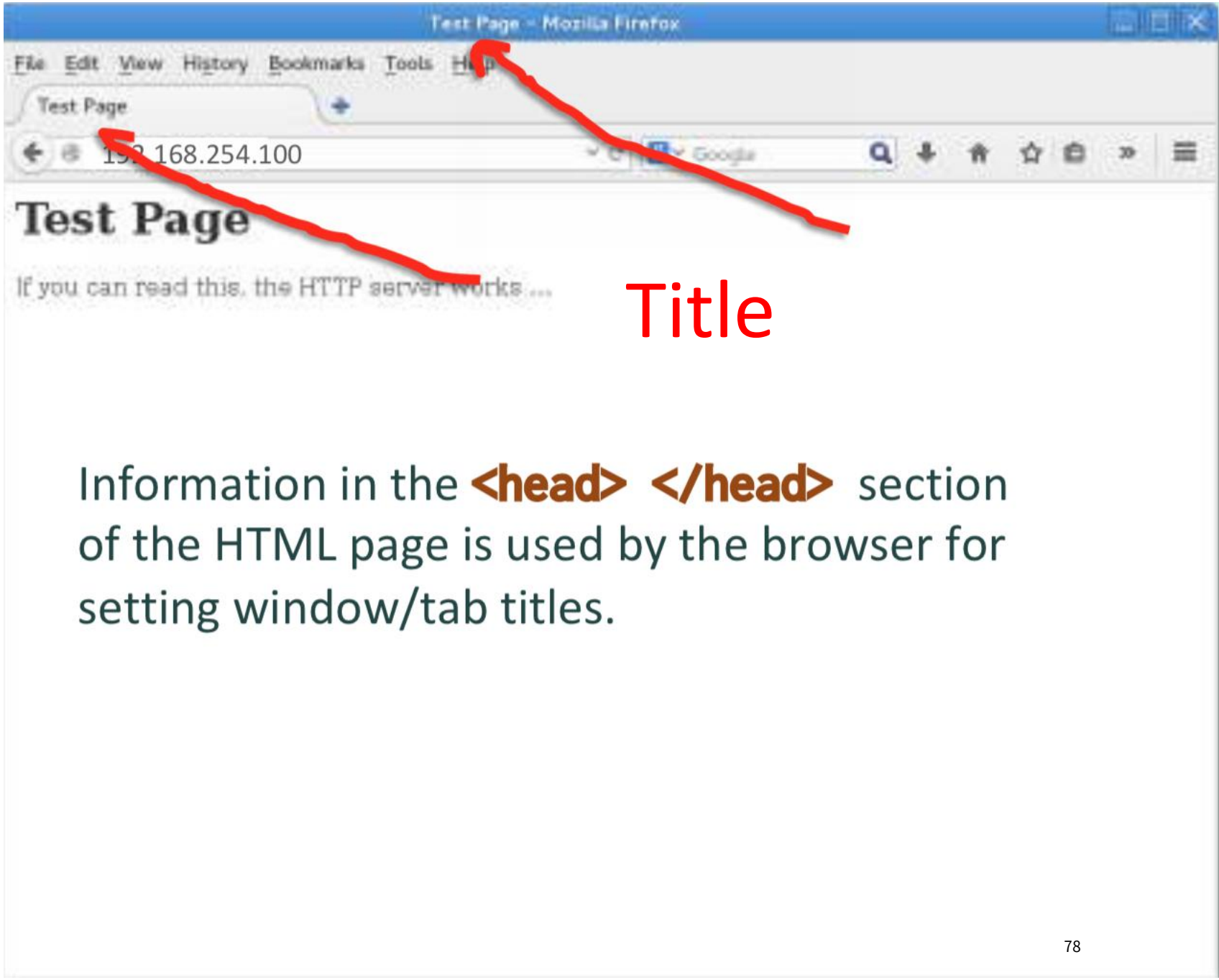
- index.html may look like this.
- What the browser will display is between <body> and </body>

```
<!doctype html>
<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>
```

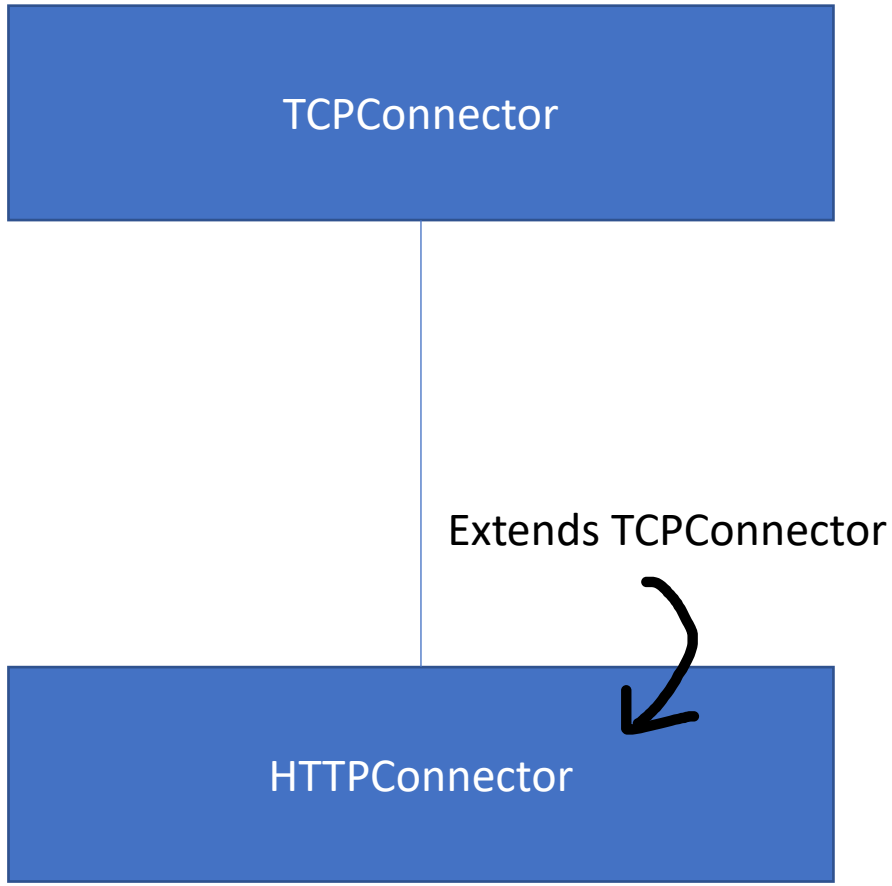
The Rendered (Generated) Page



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

Design*

- 2 Classes
- TCPConnector
  - Creates a socket
  - Set up streams
  - Send and receive messages
- HTTPConnector
  - Communicate HTTP
  - Always use port 80



(*credit Vic Hargrave)

TCPConnector Class

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

Here are the fields defined in the class…

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

class TCPConnector {

private hostName;      www.example.com
private hostAddr;      123.123.123.123
private int port;

private Socket s = null;
private BufferedReader in = null;
private BufferedWriter out = null;
```

TCPConnector

- A constructor.
- getInetAddress() returns, when transformed into a string, a website name (possibly empty) followed by "/" and an IP address…
- Another method getHostAddress() returns the text resolution of the domain name (e.g. www.example.com).

```
1  import java.net.Socket;
2  import java.net.SocketTimeoutException;
3  import java.io.*;
4
5  class TCPConnector {
6
7  private hostName;
8  private hostAddr;
9  private int port;
10
11 private Socket s = null;
12 private BufferedReader in = null;
13 private BufferedWriter out = null;
14
15 public TCPConnector(String host, int portNum) throws IOException
16     hostName = host;
17     port = portNum;
18
19     s = new Socket(host, portNum);
20     s.setSoTimeout(1000);
21     hostAddr = s.getInetAddress().toString().split("/")[1];
22
23     in = new BufferedReader(new
24     InputStreamReader(s.getInputStream()));
25
26     out = new BufferedWriter(new
27     OutputStreamWriter(s.getOutputStream()));
28 }
```

```
28      }
29
30      public void close() throws IOException {
31          try {
32              in.close();
33              out.close();
34              s.close();
35          } catch (java.net.SocketException e) {
36              // Do nothing
37          }
38      }
39
40      public String getHostAddr() {
41          return hostAddr;
42      }
```

Closing connections properly is important ...

TCPConnector

- Finally another 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...

```
42  }
43
44  public void send(String msg) throws IOException {
45      out.write(msg);
46      out.flush(); // IMPORTANT
47  }
48
49  public String receive() throws IOException {
50      try {
51          String s = in.readLine();
52          return s;
53      } catch (SocketTimeoutException e) {
54          return null;
55      }
56  }
```

```
public 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());

       header.append("GET " + pagename + " HTTP/1.1\n");
```

- The HTTPConnector class extends the TCP class just defined. HTTP is an application protocol that operates ”on top” of TCP and consists of text including headers, body tags etc, it uses TCP to transfer the information across a network in packets (more in networking class)
- The get function handles header and body separately. The goal is to be able to store the length of the body in the header so that the other end can check that the full message was received.

HTTPConnector

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

    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 with a GET request header and then wait for the answer.
- We first read the header (stopping when hit an empty line)

HTTPConnector

condition: “bytesToRead” in body and message is not null

```
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;
        } else {
            if (msg.toLowerCase()
                                .startsWith("content-length")) {

                bytesToRead = Integer
                                    .parseInt(msg.split(":")[1]
                                    .trim());
            }
            header.append(msg);
        }
    }
```

When reading the header, if the content length is given then we read this length (and assign to bytesToRead which then allows the loop to continue until all data is read)

```
                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 when there are no more bytes to read.

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

    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;
            } else {
                if (msg.toLowerCase()
                                    .startsWith("content-length")) {

                    bytesToRead = Integer
                                        .parseInt(msg.split(":")[1]
                                        .trim());
                }
                header.append(msg);
            }
        } else{
            read = 1 + msg.length();
            body.append(msg);
            body.append("\n");
            bytesToRead -= read;
        }
    }
return body.toString();
```

```
lic class GetHomePage {

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

The class HTTPConnector can be used get any web page, to get the home page we are asking for the home-page specifically at the root “/” directory.

Network Features in Java Packages

- Many packages in Java have built-in networking capabilities

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

URI Class: see java.net.URI
(https://docs.oracle.com/javase/7/docs/api/java/net/URI.html)

The URI class represents a Uniform Resource Identifier (URI) reference.

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

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

Designing a Client Server Application: Server

Designing a 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).

1. Define a protocol for (client-server) communications
2. Server
3. Client

Protocol: *"what messages does the server understand?"*

- To write a client/server application the first thing to define was the protocol, which is basically defining the list of commands recognized by the server and how it should respond to each of them.
- 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?
- Of course, all error cases should also be thought of! You can have several cases of errors, client side errors (the client sent either a command you couldn't make sense of, or misused a command), or server side errors (... for instance for some obscure reason you cannot connect to the database you need).

Example: Protocol for Queries of a Film Database

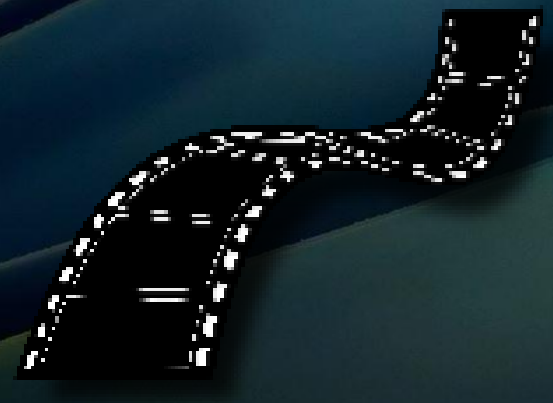
- Query the database for films that match certain conditions
  - Title
  - Director
  - Actors/Actresses
  - Year
  - Country

see www.imdb.com for instance for a database of films with a search feature.

Example: Protocol for Queries of a Film Database

Application Protocol (messages that the applications uses in queries for films)

It will run on top of other network protocols such as TCP or UDP



Such a protocol could be a keyword followed by a condition:

- **KEYWORD** cond.

This message would ask for the list of films in which Audrey Hepburn played:

- EG **ACTRESS** Audrey Hepburn

Example: Protocol for Queries of a Film Database

- I may want to implement "or"

KEYWORD cond[**|**cond ...]

- This would return films with either Audrey Hepburn or Ingrid Bergman:

ACTRESS Audrey Hepburn **|** Ingrid Bergman

Example: Protocol for Queries of a Film Database

- I may also want the films to match some other conditions

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

- This becomes films with either Audrey Hepburn or Ingrid Bergman, but directed by Hitchcock.

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

Example: Protocol for Queries of a Film Database

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

    public FilmProtocol(Connection cnx) {
        con = cnx;
    }

    public String processInput(String theInput) {
        ...
    }

    private String runQuery(String query) {
        ...
    }
}
```

calls a method to query a database

Code Outline for such a protocol:

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

Server

Just a **big** loop

The server just waits for queries, and executes them.

Film Server Example

Notice passing the port we are using on the command line (never hard-code it in the program, the port may already be taken)

```
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)
    //
```

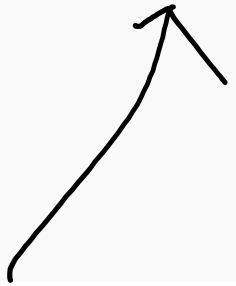
Film Server Example

We create a FilmProtocol (that does basically all the hard bits in the job) then create a socket and loop forever (should in a real app have a way to stop it nicely).

```
//
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()));
```

autoflush = true

```
    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()));

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

}
```



the FilmProtocol Object that does the tough bits (parsing the input line and generating the required SQL query)

Designing a Client Server Application: Client

Client

- Can be very Basic:
  - command line tool
- Or much more complex:
  - e.g. a website or a GUI
- Don't underestimate ugly console applications – they are easier to integrate with other processes.

- Standard data exchange formats: JSON, XML, CSV

- The client might also use a bit of its processing power to try to present the result better. It would make the job easier if data returned by the server were better formatted.

Basic Film Client Example

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.

Basic Film Client Example

```
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 has to be told where to connect (host and port)…

Basic Film Client Example

```
boolean loop = true;
while (loop) {
    try ( // try with resources
        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 every query is independent (there is no "session") here we create (and close) one connection for each query.

Basic Film Client Example

```
String fromUser;
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;
    }
}
```

On exit the server will send an acknowledgement (“Goodbye”).

Any error messages could also be received from the server here and displayed by the client as unprocessed text data from the server in this example.

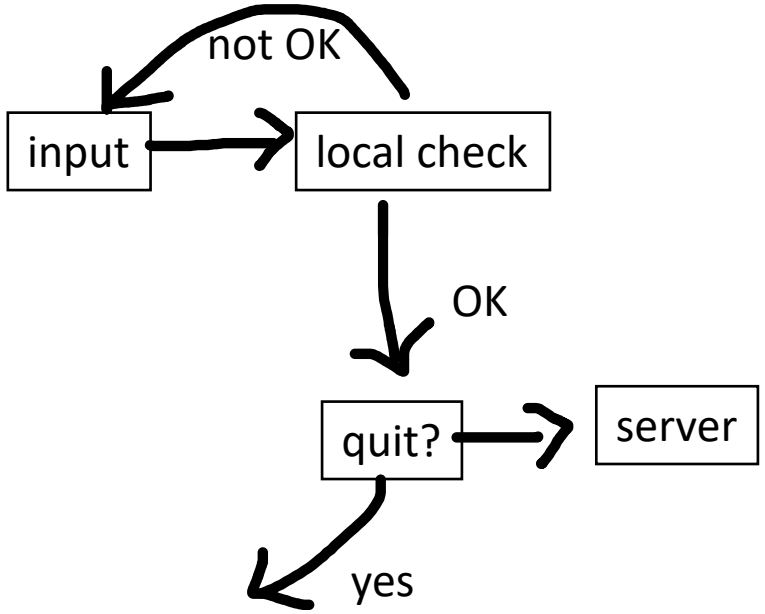
Basic Film Client Example

```
            break;
            }
        }
    } 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 connection errors in case client is unable to contact the server at all.

Client Design

- This client is quite simple and generally it is not efficient to let the server check everything
- 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 require less network bandwidth.



Client Design

- Further potential improvements to client are in rendering and presenting the data
- Standard data exchange format such as CSV, XML, JSON ...
- The client might also use a bit of its processing power to try to present the result better. It would make the job easier if data returned by the server were better formatted.

Limitation: One Client at a Time

- The main weakness of previous examples we have seen of servers is that when it processes a query, it cannot check if there is another request. This was OK because there were not hundreds of requests arriving at the same time and queries execute fast. But think of the traffic on a popular website, for instance an ecommerce one, where a lot of users are asking for pages that are built on the fly ...

- Solution: multithreading.

Software System Design: Modular Systems

Software is Complex

Complex ≠ complicated

Complex = composed of many simple parts

related to one another

Complicated = not well understood, or explained

Software Engineering is Complex and Complicated

- Large scale software systems have many components and this tends to lead to chaos
- It is difficult to define requirements as users cannot know what the precisely want or need and they evolve over time, the creeping change of requirements is a primary reason for software project failure (relationship of system function – behavior – structure)
- Also different designs can achieve similar functions and it is difficult to judge whether a design is good (Conway ’s law: The structure of software tends to be similar to the organizational structure of the development team)

Function-Behaviour-Structure Ontology

https://en.wikipedia.org/wiki/Function-Behaviour-Structure_ontology

Gero J.S. (1990) "Design prototypes: a knowledge representation schema for design", AI Magazine, 11(4), pp. 26–36.

- Function (F): the teleology (purpose) of the design object.
- Behavior (B): the attributes that can be derived from the design object's structure.
- Structure (S): the components of the design object and their relationships.
- The three ontological categories are interconnected:

Function is connected with behavior, and behavior is connected with structure. There is no connection between function and structure!

Conway’s Law

Conway's Law:

The rule that the organization of the software and the organization of the software team will be congruent; commonly stated as “If you have four groups working on a compiler, you'll get a 4-pass compiler”.

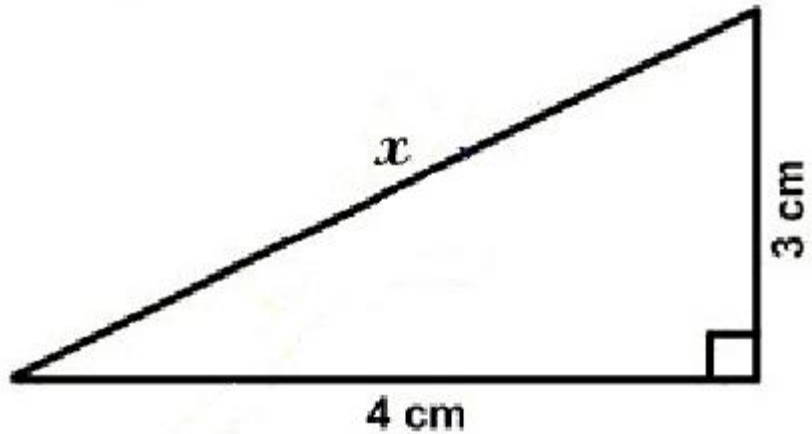
The original statement was more general, “Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations.” This first appeared in the April 1968 issue of *Datamation*.

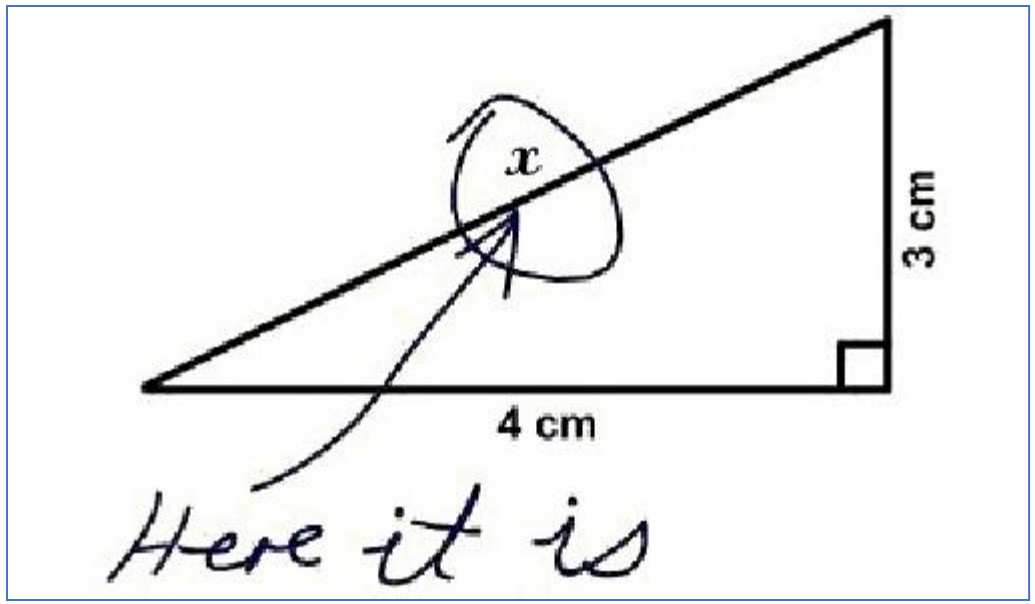
http://www.catb.org/~esr/jargon/html/C/Conways-Law.html

One of the Reason to explain Why Software Development is difficult:

How to judge whether a design is good or not is a big challenge, that is largely based on experience.

3. Find x.



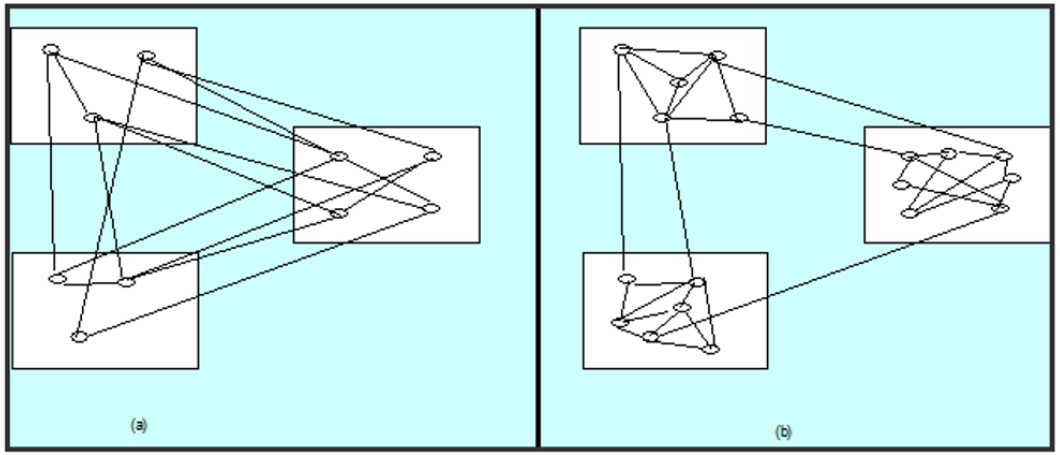


Decomposing Systems into Modules

- High cohesion within modules
- Loose coupling between modules

Parnas, D.L. (Dec. 1972). "On the Criteria To Be Used in Decomposing Systems into Modules". Communications of the ACM. **15** (12): 1053–58.

A visual representation



high coupling (low cohesion)

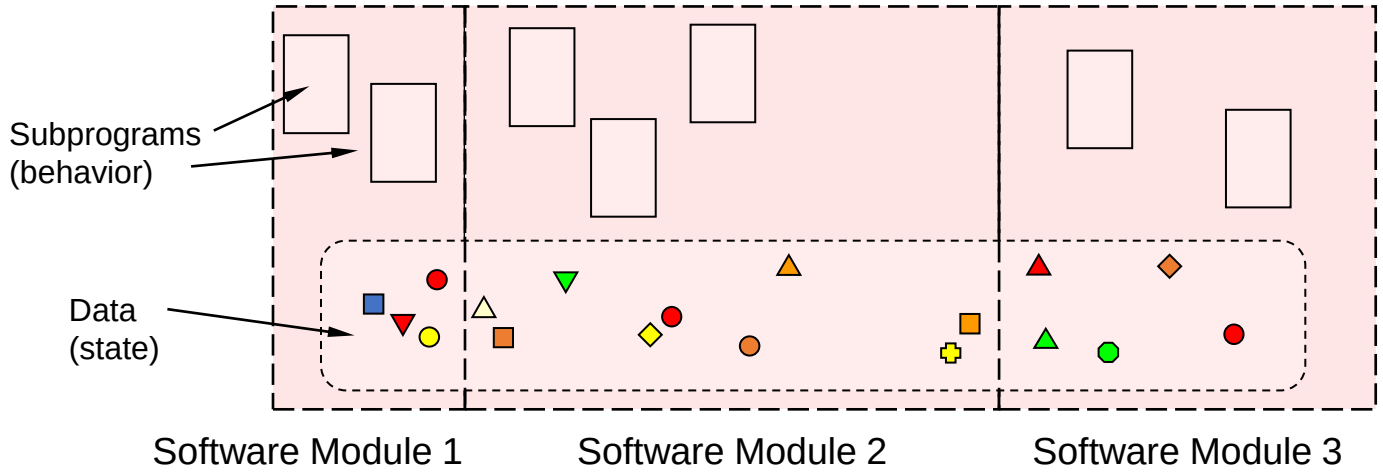
low coupling (high cohesion)

Modularity, Cohesion and Coupling

- A complex system may be divided into simpler pieces called *modules*
- A system that is composed of modules is called *modular*
- Supports application of separation of concerns
  - when dealing with a module we can ignore details of other modules
- Each module should be *highly cohesive*
  - module understandable as a meaningful unit
  - Components of a module are closely related to one another
- Modules should exhibit *low coupling*
  - modules have low interactions with others
  - understandable separately

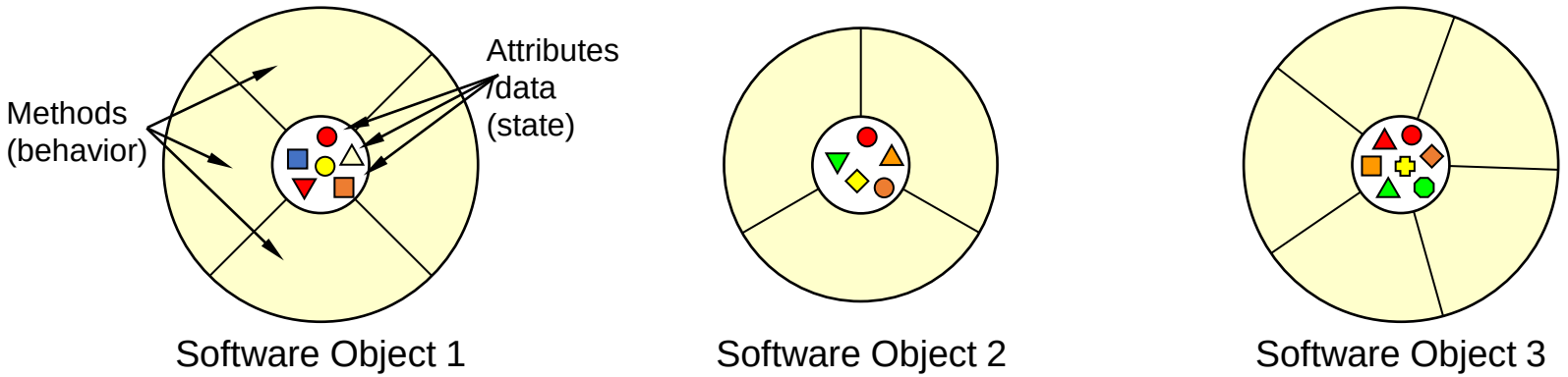
Modules versus Objects

Modules are loose groupings of subprograms and data



"Promiscuous" access to data often results in misuse

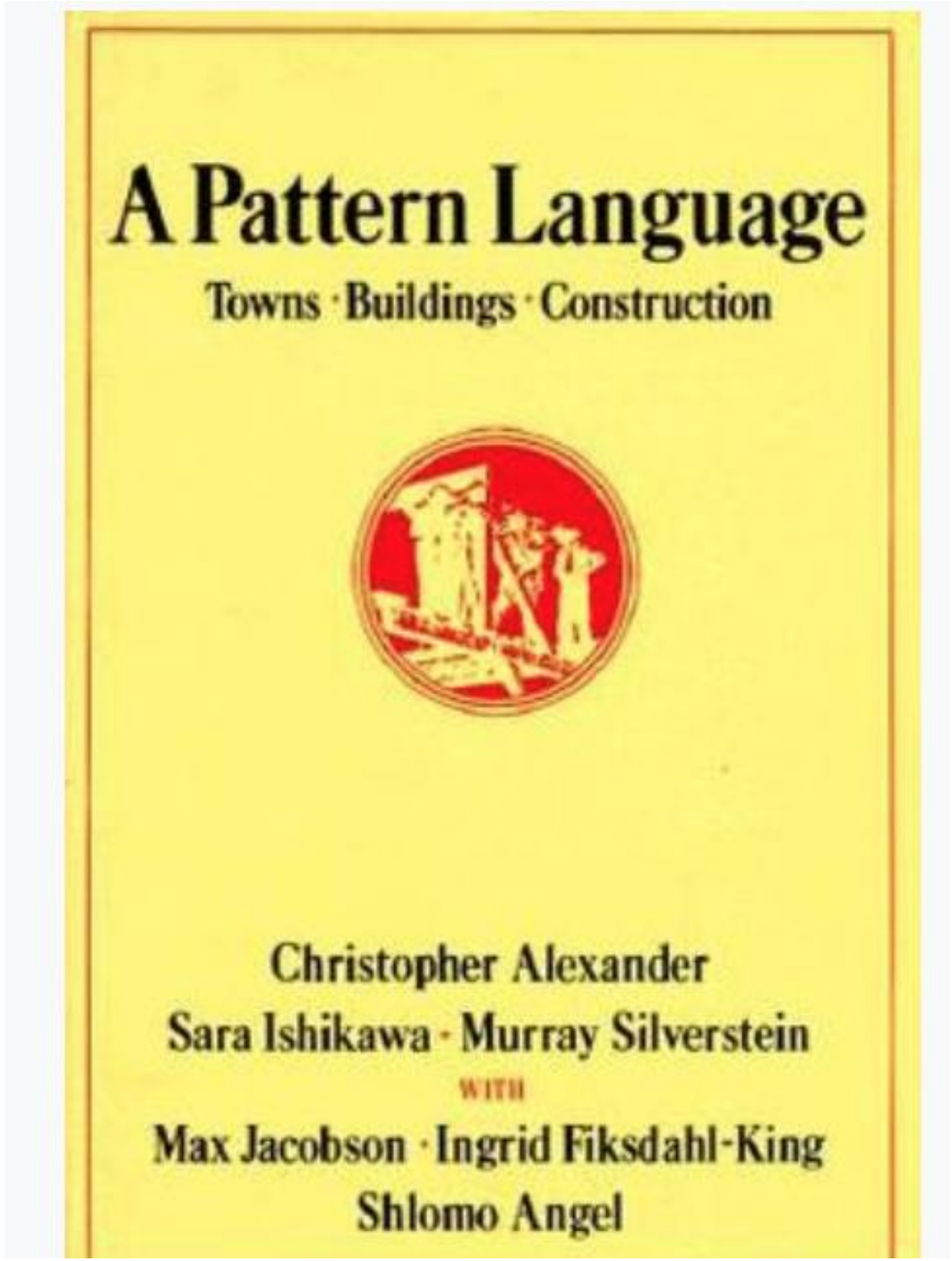
Objects ***encapsulate*** data

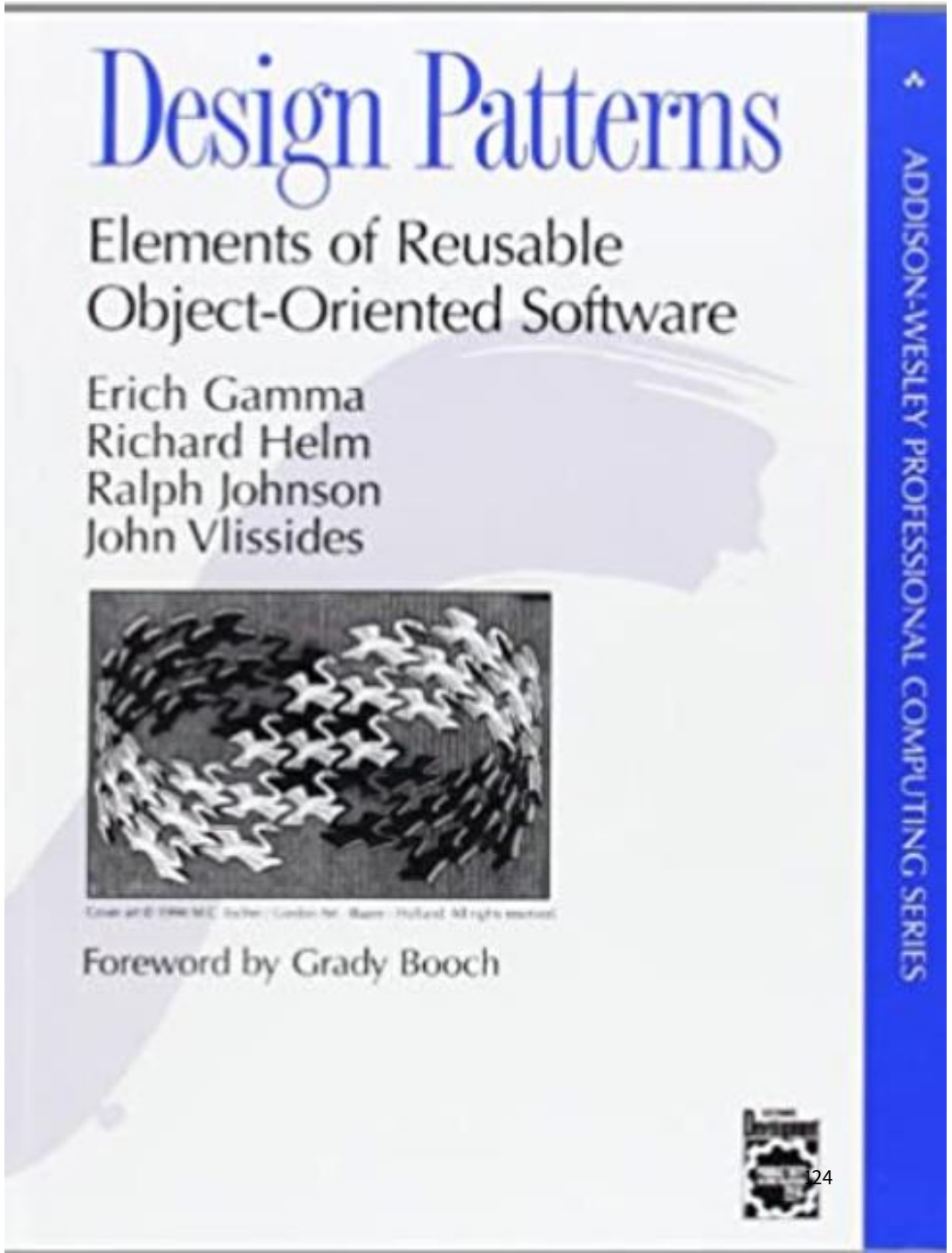


Software System Design: Patterns

The Pattern Concept

- History: Architectural Patterns
- Christopher Alexander
- Each pattern has
  - a short *name*
  - a brief description of the *context*
  - a lengthy description of the *problem*
  - a prescription for the *solution*


A Pattern Language
Towns · Buildings · Construction
Christopher Alexander
Sara Ishikawa · Murray Silverstein
WITH
Max Jacobson · Ingrid Fiksdahl-King
Shlomo Angel


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Short Passages Pattern

Short Passages Pattern

Context

"...Long, sterile corridors set the scene for everything bad about modern architecture..."

Problem

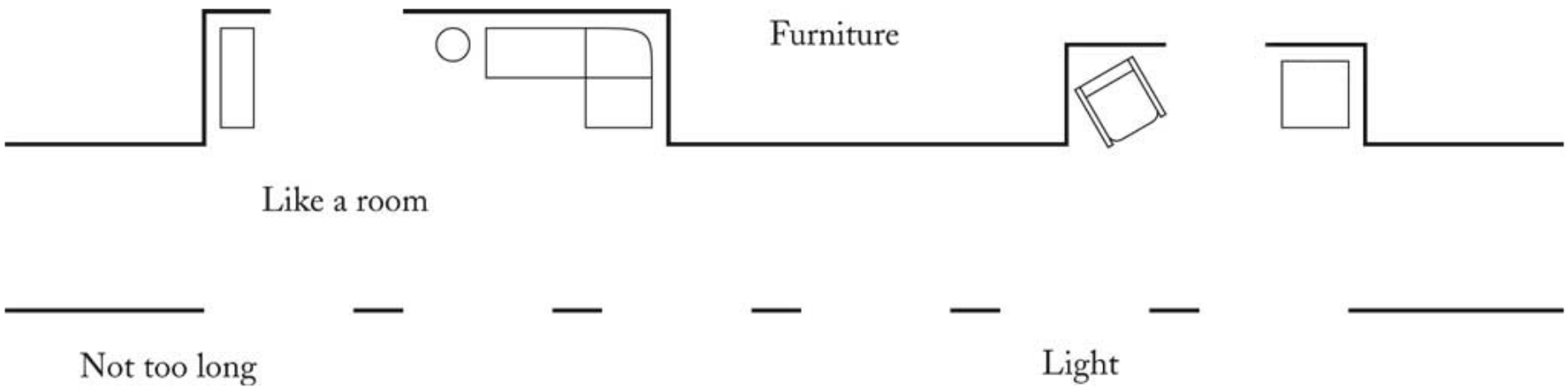
a lengthy description of the problem, including

- a depressing picture
- issues of light and furniture
- research about patient anxiety in hospitals
- research that suggests that corridors over 50 ft are considered uncomfortable

Short Passages Pattern

Solution

Keep passages short. Make them as much like rooms as possible, with carpets or wood on the floor, furniture, bookshelves, beautiful windows. Make them generous in shape and always give them plenty of light; the best corridors and passages of all are those which have windows along an entire wall.



Iterator Pattern

Context

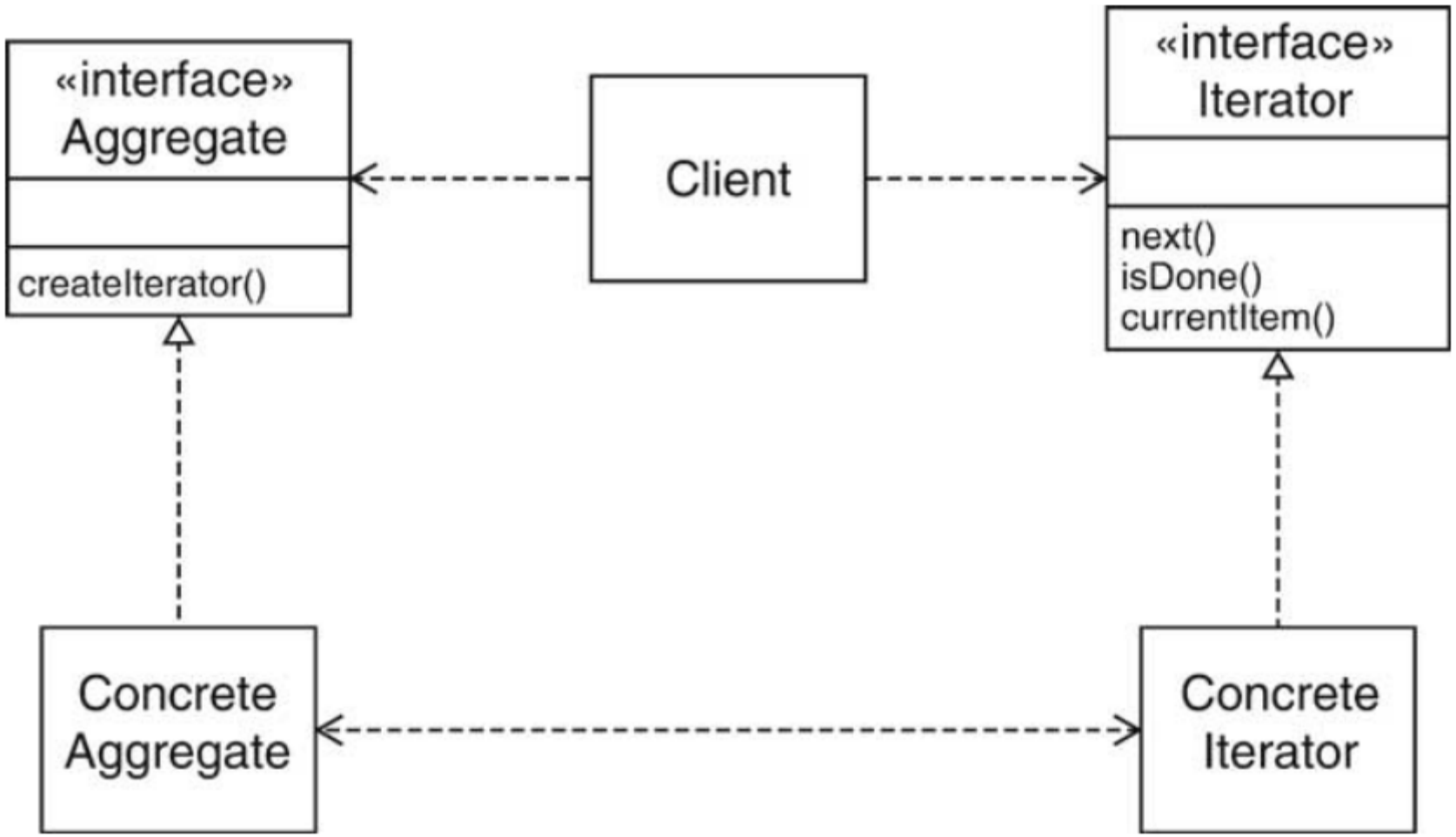
1. An aggregate object contains element objects
2. Clients need access to the element objects
3. The aggregate object should not expose its internal structure
4. Multiple clients may want independent access

Iterator Pattern

Solution

1. Define an iterator that fetches one element at a time
2. Each iterator object keeps track of the position of the next element
3. If there are several aggregate/iterator variations, it is best if the aggregate and iterator classes realize common interface types.

Iterator Pattern



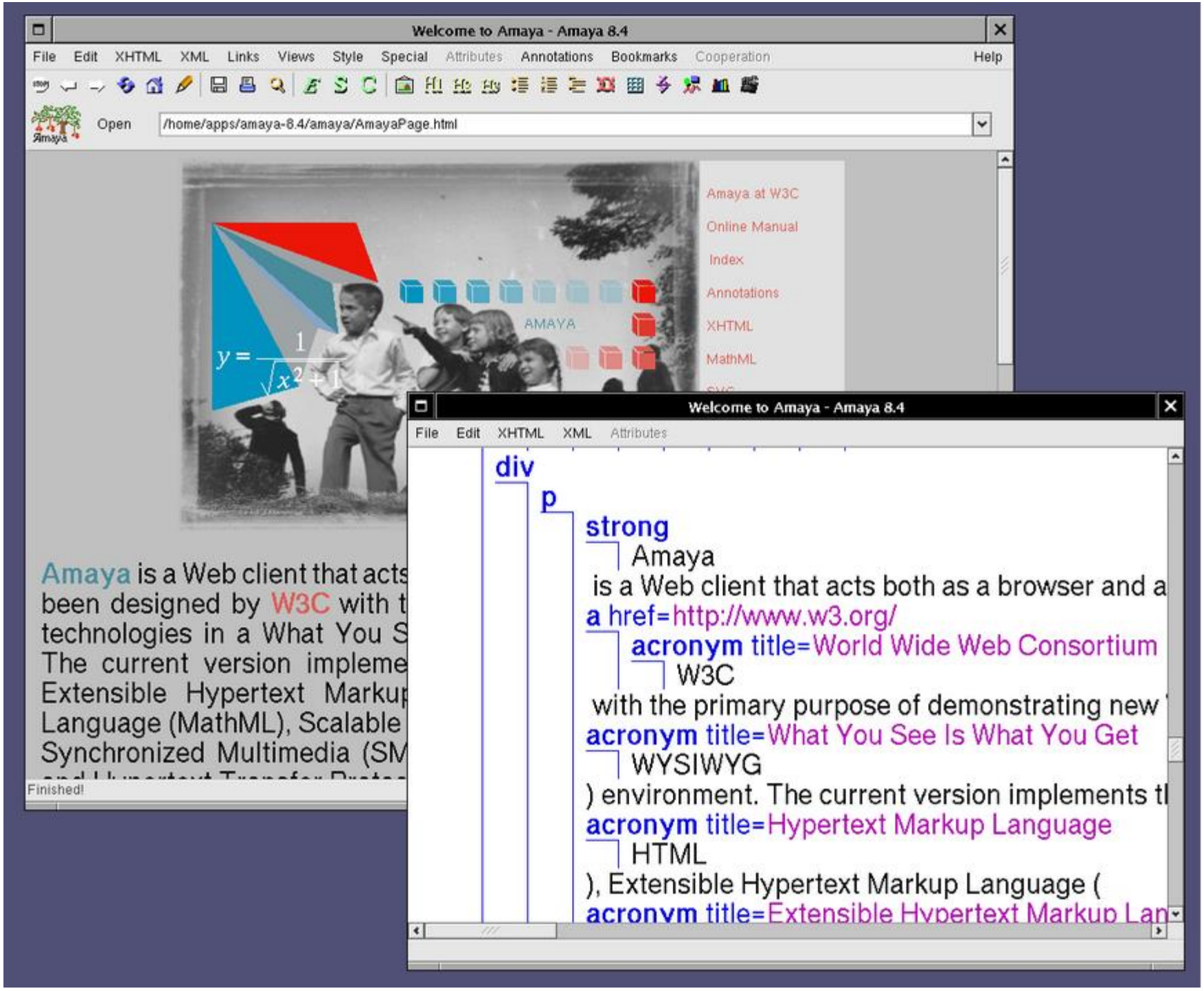
Iterator Pattern

- Names in pattern are *examples*
- Names differ in each occurrence of pattern

Name in Design Pattern	Actual Name (linked lists)
`Aggregate`	`List`
`ConcreteAggregate`	`LinkedList`
`Iterator`	`ListIterator`
`ConcreteIterator`	anonymous class implementing `ListIterator`
`createIterator()`	`listIterator()`
`next()`	`next()`
`isDone()`	opposite of `hasNext()`
`currentItem()`	return value of `next()`

Model/View/Controller

- Some programs have multiple editable views
- Example: HTML Editor
  - WYSIWYG view
  - structure view
  - source view
- Editing one view updates the other
- Updates seem instantaneous

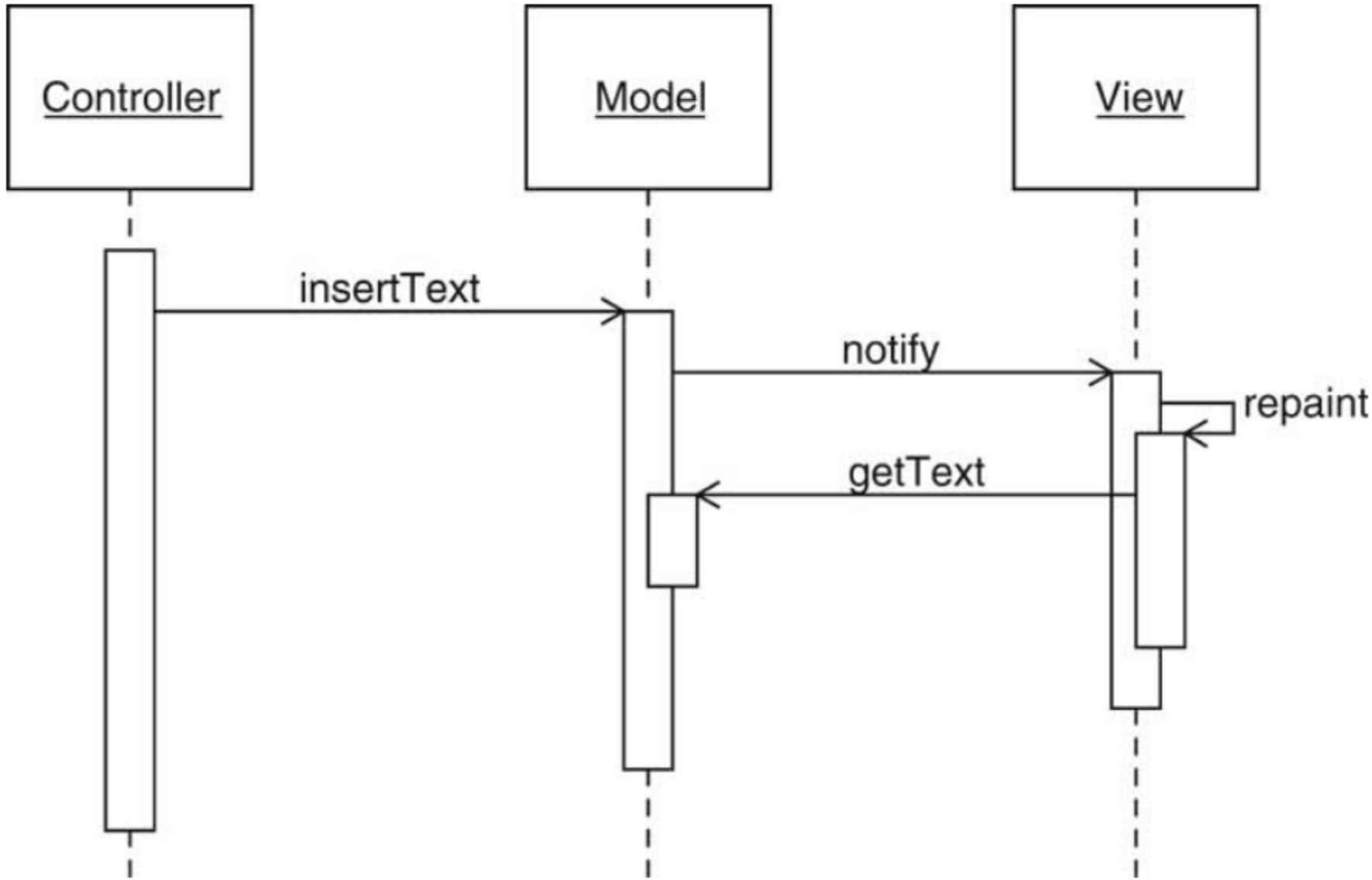

Welcome to Amaya - Amaya 8.4
File Edit XHTML XML Links Views Style Special Attributes Annotations Bookmarks Cooperation Help
Open
/home/apps/amaya-8.4/amaya/AmayaPage.html
Amaya at W3C
Online Manual
Index
Annotations
XHTML
MathML
AMAYA
Amaya is a Web client that acts
been designed by W3C with t
technologies in a What You S
The current version impleme
Extensible Hypertext Markup
Language (MathML), Scalable
Synchronized Multimedia (SM
Finished!
Welcome to Amaya - Amaya 8.4
File Edit XHTML XML Attributes
div
p
strong
Amaya
is a Web client that acts both as a browser and a
a href=http://www.w3.org/
acronym title=World Wide Web Consortium
W3C
with the primary purpose of demonstrating new
acronym title=What You See Is What You Get
WYSIWYG
) environment. The current version implements t
acronym title=Hypertext Markup Language
HTML
), Extensible Hypertext Markup Language (
acronym title=Extensible Hypertext Markup Lan

Model/View/Controller

- Model: data structure, no visual representation
- Views: visual representations
- Controllers: user interaction

- Views/controllers update model
- Model tells views that data has changed
- Views redraw themselves

Model/View/Controller



Software Development Process

Writing code is one thing. But…

- **PROJECTS REQUIRE MORE**
  - Documentation
  - Organization
    - Code
    - Build
    - Integration
  - Testing
  - Deployment

In general, and with any language, a project is more than writing one program and there is a whole eco-system around it. This will be a brief overview, mostly to provide ideas and topics to search on the web.

Documentation

Everyone wants documentation

Everyone hates writing it

You are probably already somewhat familiar with javadoc and were asked to use it in your project… Uses specially formatted comments in the code which can then be used to generate a document describing the elements of the software system (classes, methods, packages and so forth).

See: https://www.oracle.com/java/technologies/javase/javadoc-tool.html#javadocdocuments

Software Organization

Organizing Code

In industry projects where there is a large code base, and perhaps the development team may change, there is a need for a lot of organization and methods.

Side notes:

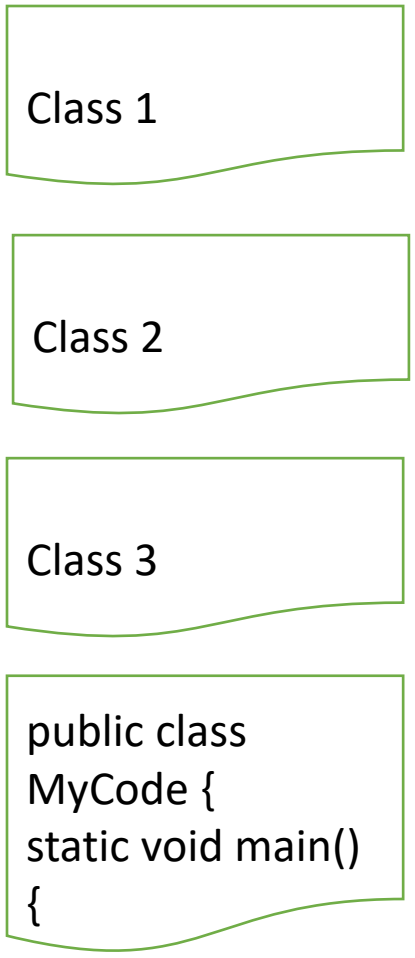
- if you are interested in software projects, there is a famous book called "The Mythical Man-Month" by Fred Brooks. It's based on the experience on a project 50 years ago but, it's still quite relevant today.
- Another famous book is “Implementing Lean Software Development: From Concept to Cash” by Mary and Tom Poppendieck, which has a lot to say about software development strategies particularly in focusing on aspects that improve efficiency and avoid creating unnecessary work.

Organizing Code

```
public class
MyCode {

            …
  Lots of methods
            …

static void main()
{

}
```

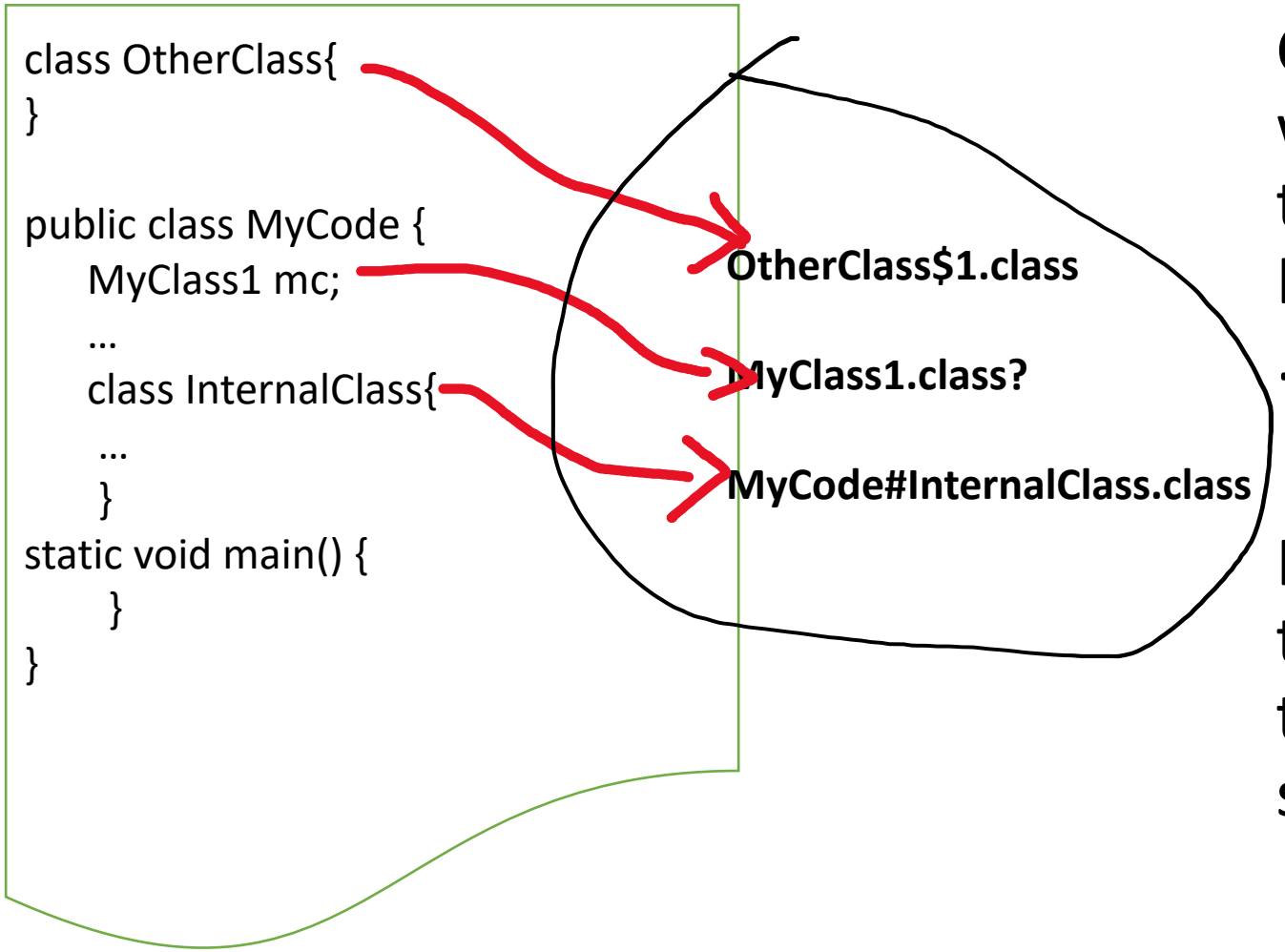
The "one big file with lots of methods" approach cannot work with several people working on the same project. You have to break everything into smaller classes with their own methods, calling methods from other classes.

Organizing Code



- Only one class contains a public static void main() methods. Other classes are public classes with constructors and methods, but no main().
- It's interesting to see what javac generates when applied to a single .java file. It won't always be a single .class file.

Javac



Classes outside the main one, as well as internal classes, will be turned into separate .class files. In fact (surprise) you have one .class per class.

If you are using another class, the JVM loader expects to find the corresponding .class somewhere

Loading Classes

The loader will look in what is supplied with the Java runtime environment, by default it will look into the current directory or any directory or .jar file (more about .jar files later) that you specify.

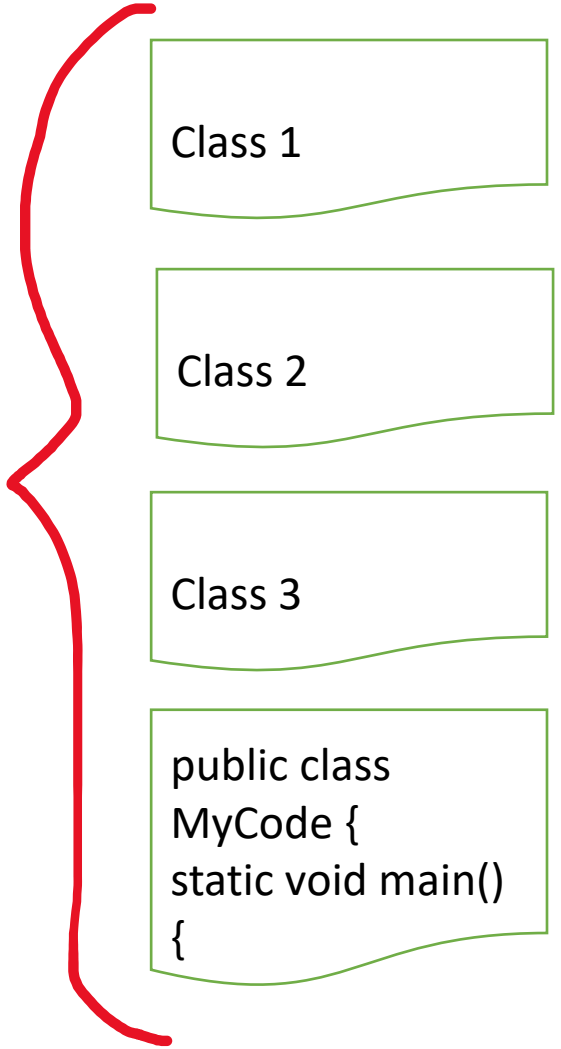
- Looks for .class files in the current directory (folder)
- Or in directories specified in CLASSPATH
- Or with -cp
  - *directory1*:*directory2*:...:...
  - *directory1*;*directory2*;...;...
- Special Directories:
  - "." = current
  - ".." = parent

Packages

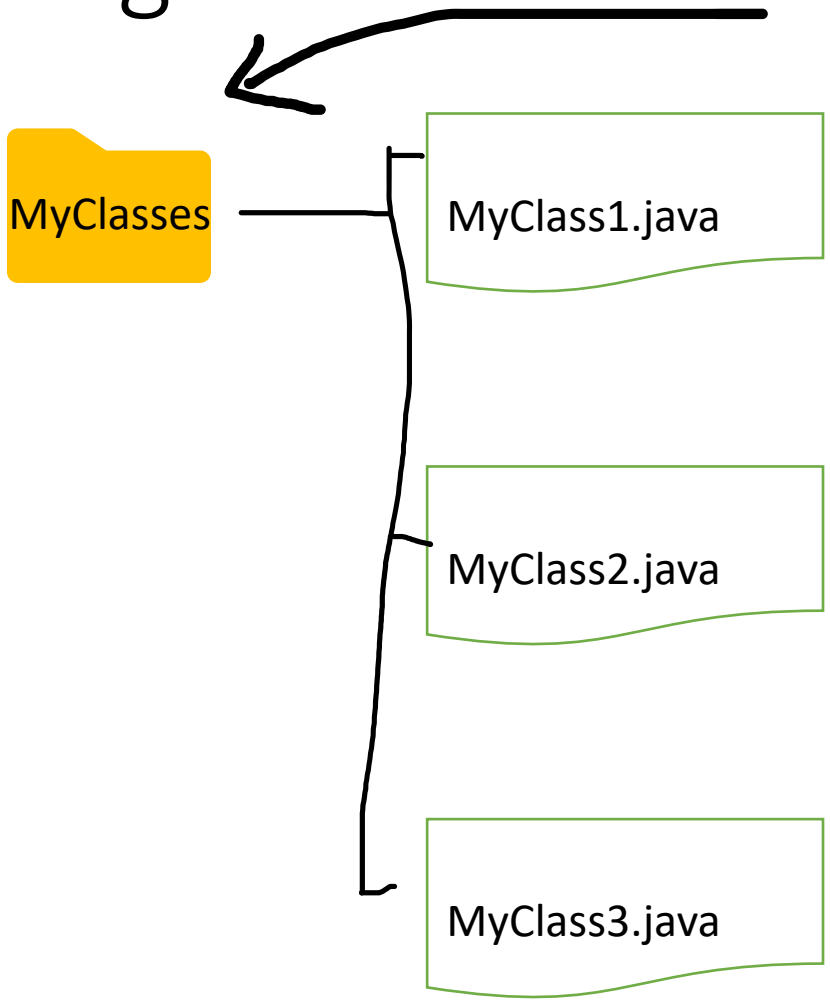
"Packages", as the name suggests, as for storing related software classes together.

Packages

You want to use packages especially when you feel that the classes could be reused in a different software project.



Packages



In practice a package is just a directory. You must name the package at the beginning of the files in the package.

```
import MyClasses.*;

public class MyCode {
static void main() {
```

Packages and naming

There is always a problem with naming: the possibility of a name conflict or giving something a a name that already exists

To avoid name conflict, companies build package directories that look like their reversed website name, followed by whatever they want. By giving the full package name, you know exactly what you are talking about.

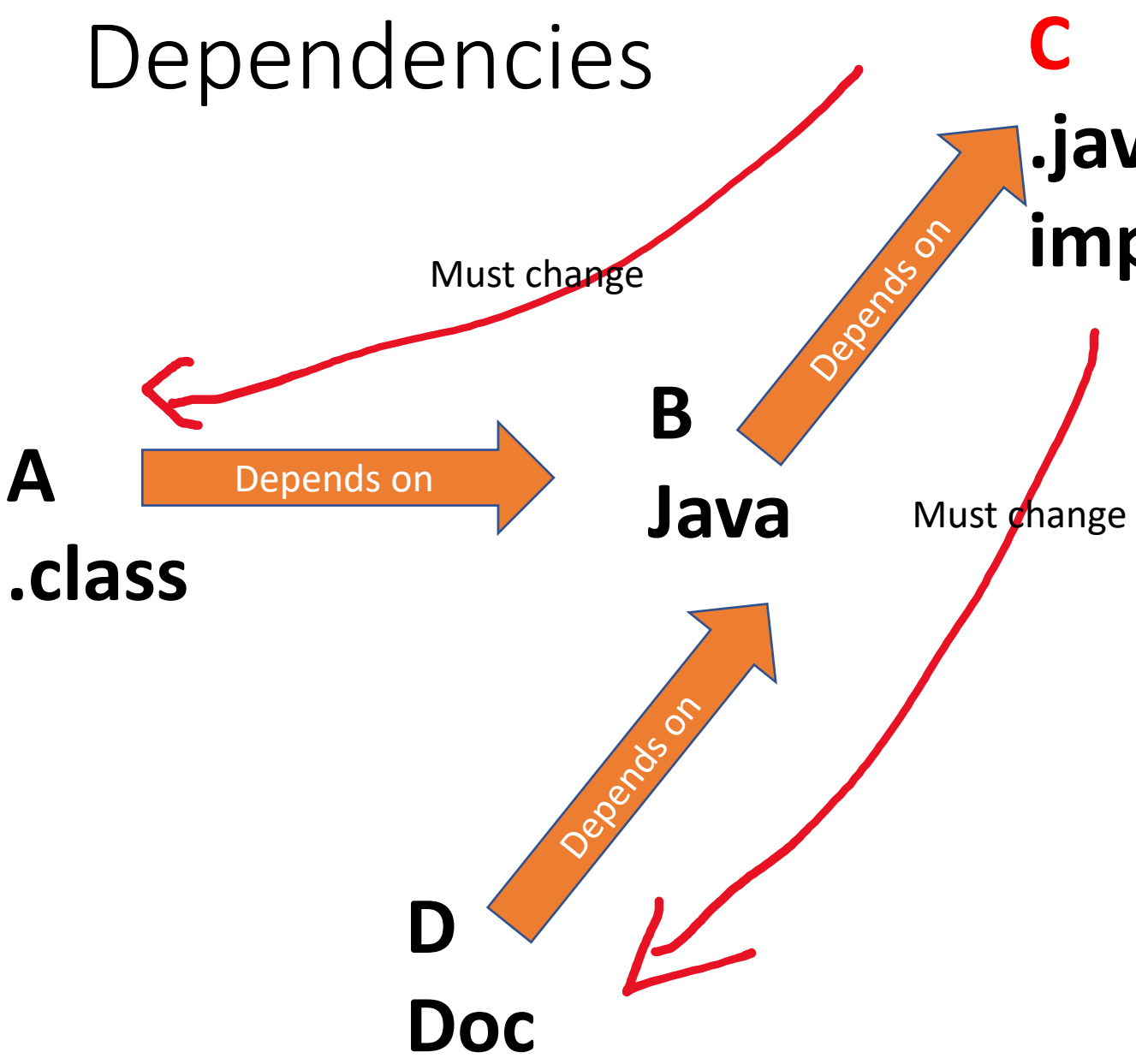
Software Build Tools

Build tools

Bigger project = Many components Dependencies...

Once again, running javac isn't the only thing you have to do. Before you compile the code, you must often extract it from a repository (more about this later). If javac is successful, you need to generate the doc, (with Javadoc tool is similar to compiling code - or in eclipse there is a wizard). You need to run (automated) tests, you need to prepare software for deployment... A lot of small, often boring tasks, depending on each other, that developers have tried to automate as much as possible.

Dependencies



When you change something somewhere, there are usually a number of tasks that must be run again on whatever depends on this component.

Build Tools: Make

- Based on file modification time
- The grand-dad of build tools is make, still very much in use and very popular among C developers. You can use it with Java as well!

But other build tools appeared that were more specifically targeted at Java development. As they were created when XML was all the rage, they usually rely heavily on XML configuration files saying what depends on what and what to do when something changes.

Apache Ant : Build.XML


```xml
<?xml version="1.0"?>
<project name="Hello" default="compile">
    <target name="clean" description="remove intermediate files">
        <delete dir="classes"/>
    </target>
    <target name="clobber" depends="clean" description="remove all artifact files">
        <delete file="hello.jar"/>
    </target>
    <target name="compile" description="compile the Java source code to class files">
        <mkdir dir="classes"/>
        <javac srcdir="." destdir="classes"/>
    </target>
    <target name="jar" depends="compile" description="create a Jar file for the application">
        <jar destfile="hello.jar">
            <fileset dir="classes" includes="**/*.class"/>
            <manifest>
                <attribute name="Main-Class" value="HelloProgram"/>
            </manifest>
        </jar>
    </target>
</project>
```

(from Wikipedia)

Apache Ant

Describes dependencies and tasks:

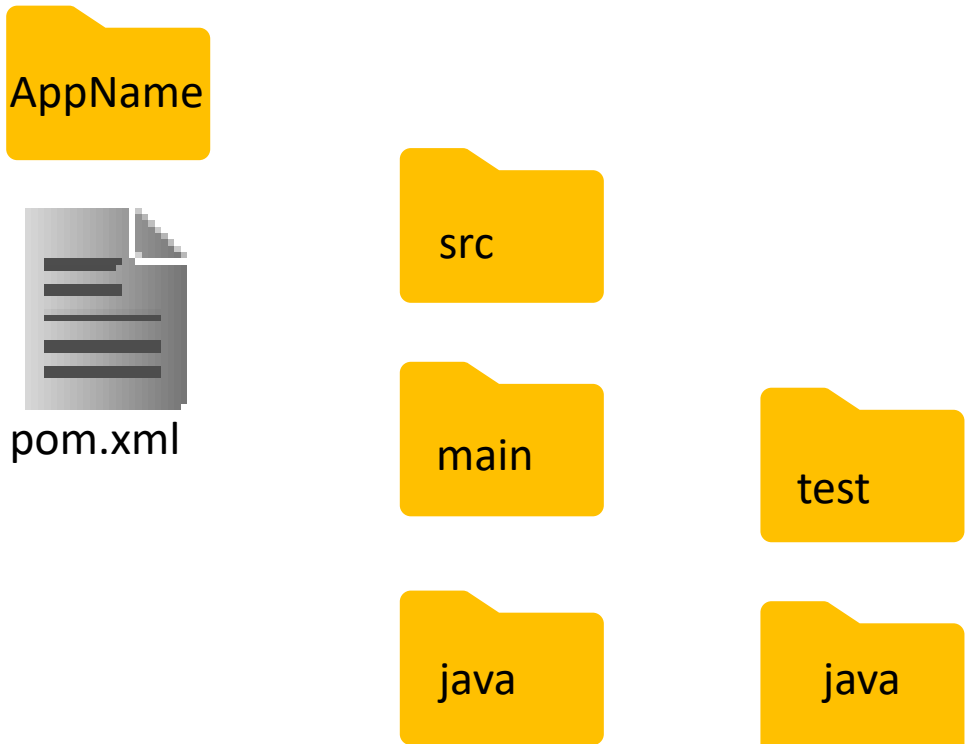
- Get source code out of source control
- Compile
- Run test
- Copy file
- And so on…

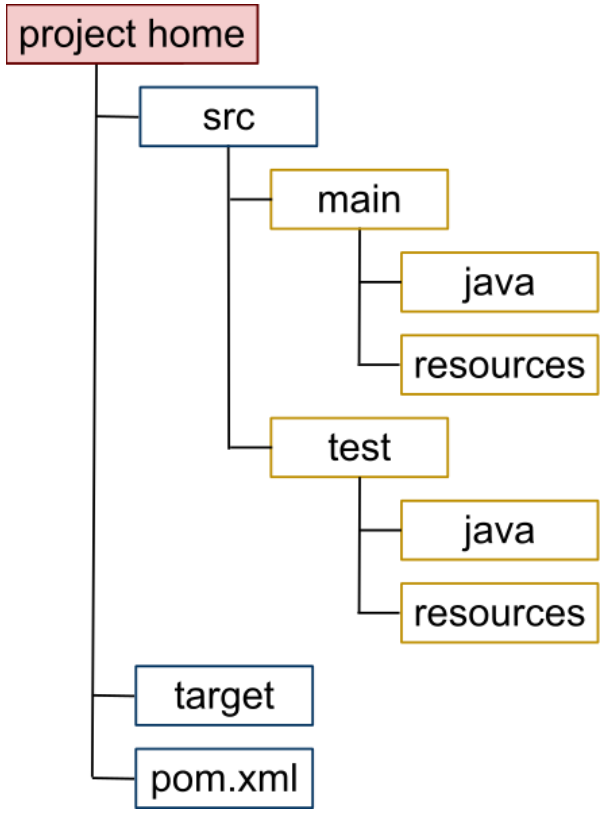
All common tasks correspond to an XML tags, with the detail provided as XML attributes.

Software Build Tools

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. “Convention over Configuration”…





- Maven handles dependencies and compilation/tool execution
- It is designed to be easy to use and provides pre-defined targets for things like compilation, packaging, deployment
- Unlike Ant it uses conventions which if followed don’t need to be explicitly specified
  - For example: build order, running unit tests, and much more
- Only things that are different from the standard conventions need to be specified in the pom.xml file

Build Lifecycles

Most of Maven's functionality is in plug-ins. A plugin provides a set of goals that can be executed using the command mvn [plugin-name]:[goal-name]. For example, a Java project can be compiled with the compiler-plugin's compile-goal by running mvn compiler:compile.

The build lifecycle is a list of named *phases* that can be used to give order to goal execution. One of Maven's standard lifecycles is the *default lifecycle*, which includes the following phases:

- validate
- generate-sources
- process-sources
- generate-resources
- process-resources
- compile
- process-test-sources
- process-test-resources
- test-compile
- test
- package
- install
- deploy

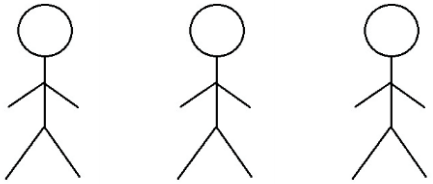
Source Control Systems

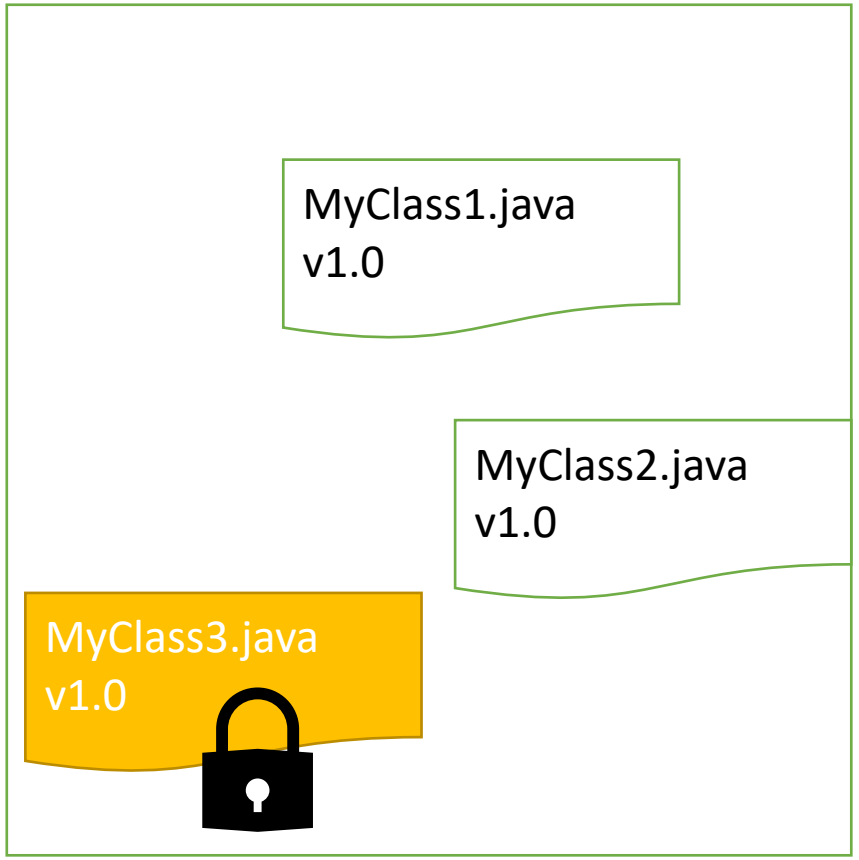
Source Control Systems

Another category of useful tools are source control systems. They are repositories where the source code for a project is stored. They ensure that only one developer modifies a file 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

Repository (“repo”)

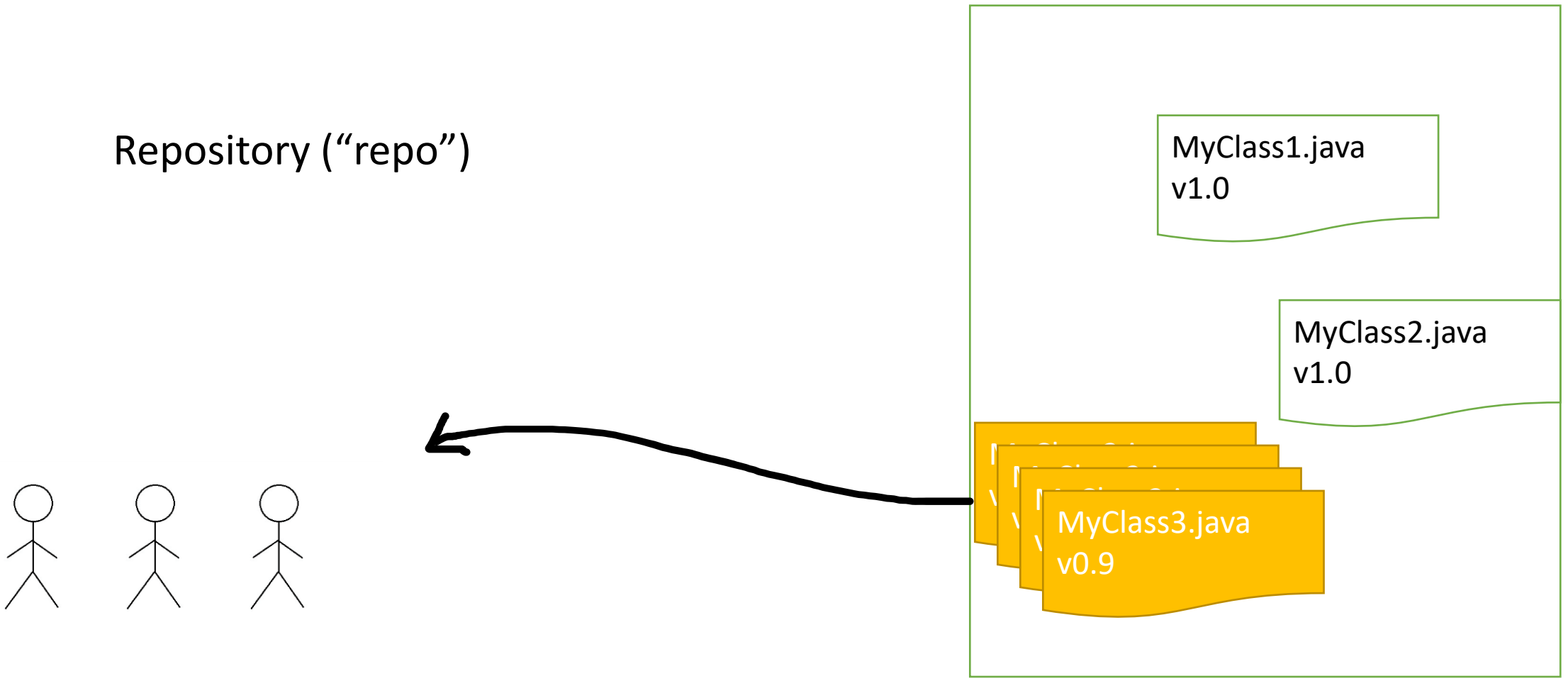




A check-in/check-out system makes files read-only when someone is modifying them.

Source Control Systems

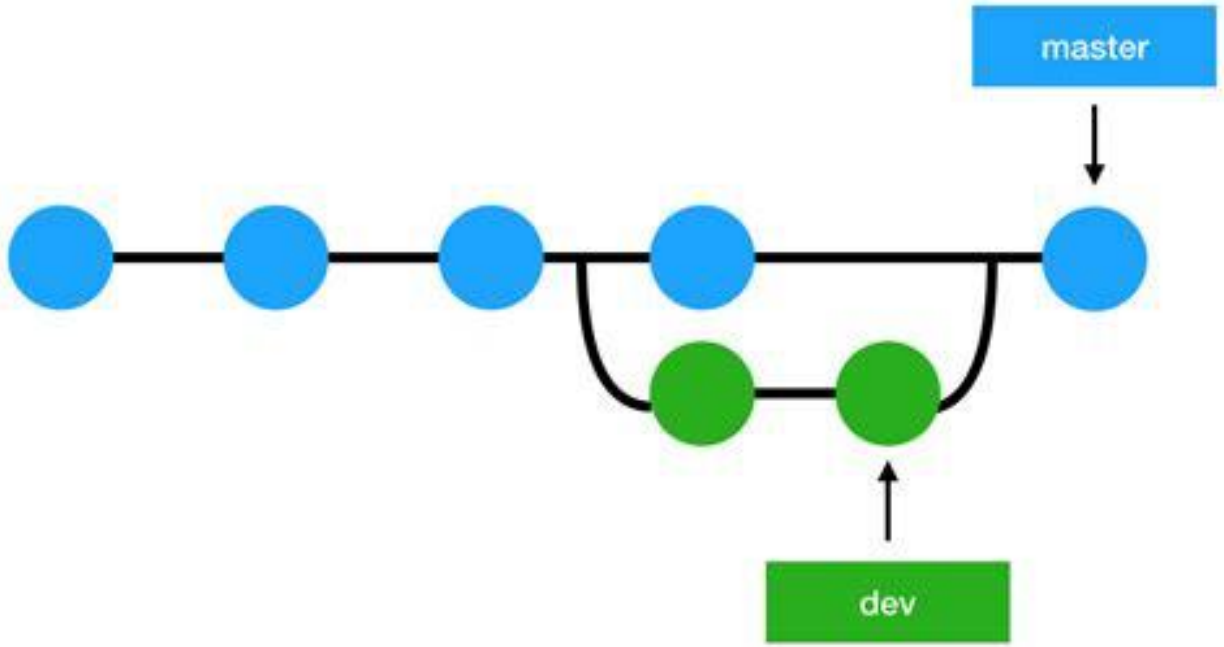
Repository (“repo”)

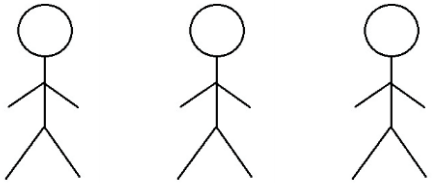


Source control systems keep a history with modifications of each file over time enabling rollback.

Source Control Systems

Branches





Source control systems have features for merging parts that have evolved independently.

Source Control Systems

- SCCS: 

- First one – 1970s

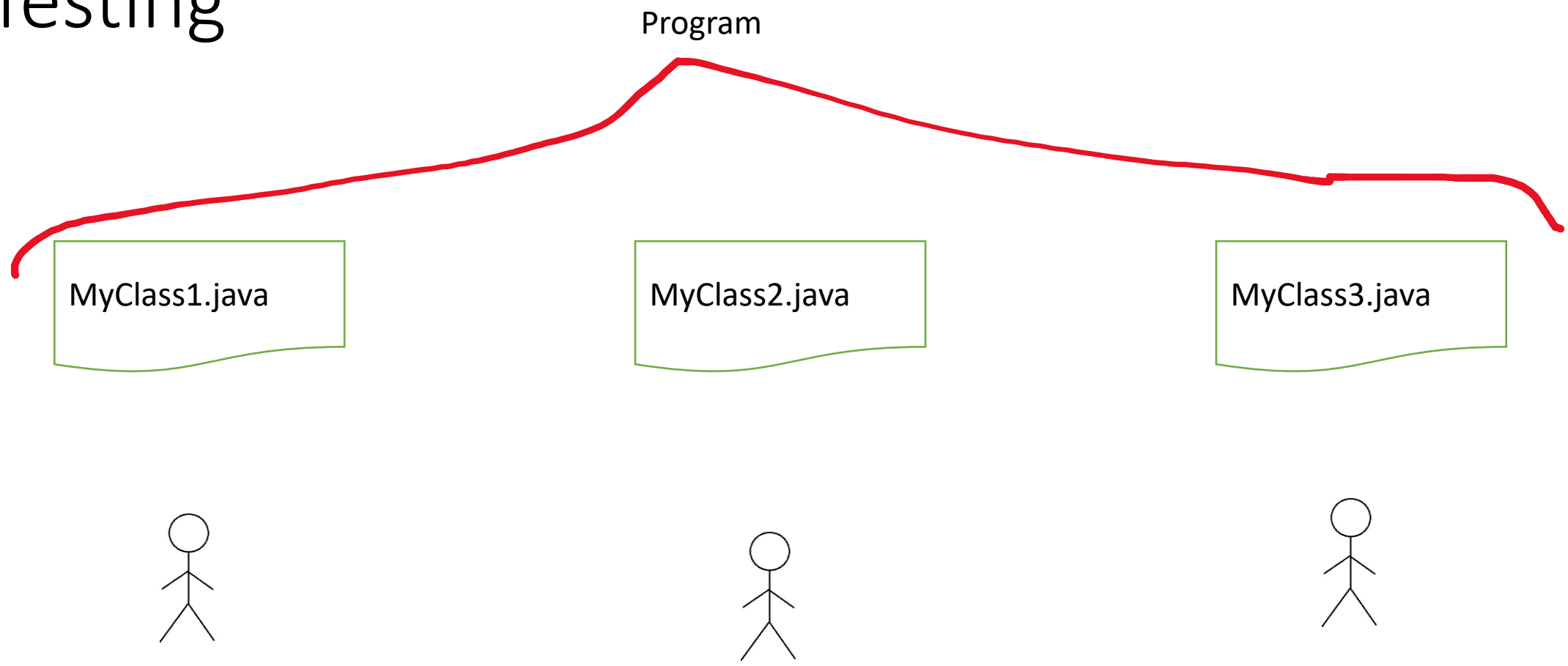
There are many systems, the first ones were local to one computer, today they can be local or use a web server.

Testing

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



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

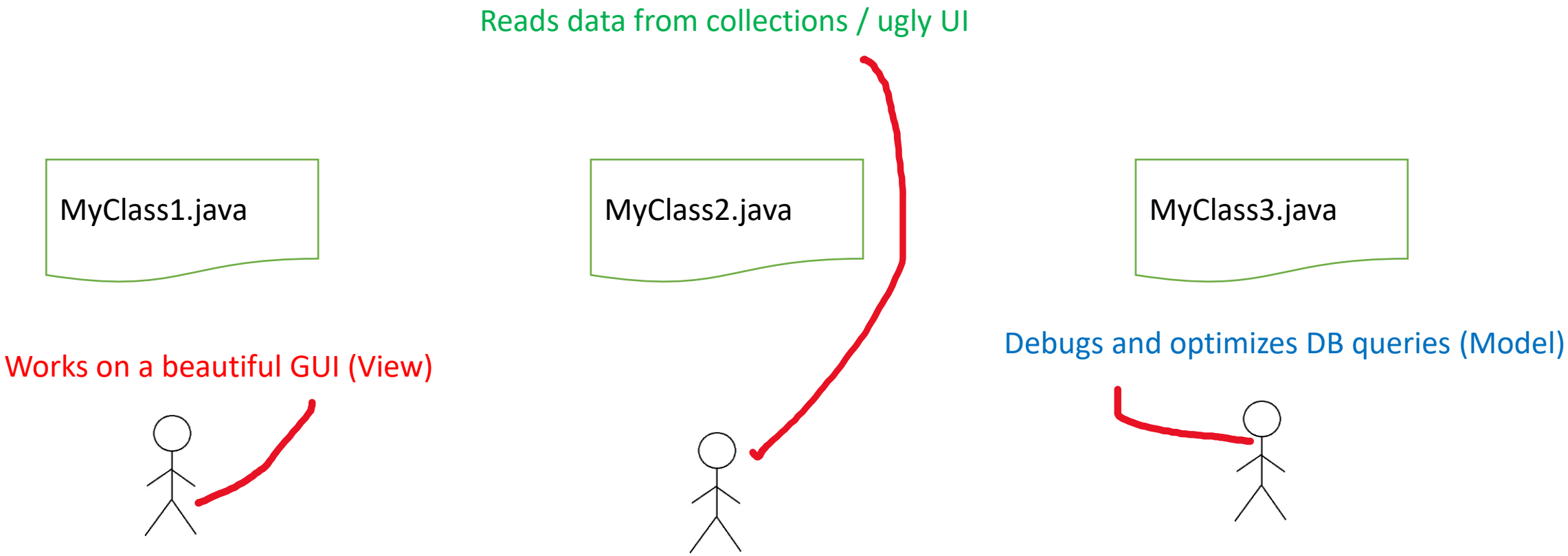
The fastest developer cannot wait on the slowest one to test the code!

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 solution is to write very simple objects and methods just for simulating the real thing (that is not ready or in another component). EG instead of actually getting data from a DB you can return the same data from a small hard-coded or configured collection. Instead of getting a real message from a remote server you can make a fake one. The word ”mock” meaning imitation or fake is often used, it can be found in the names of products that are useful for generating (with reflection) code that is used in testing.

Testing



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

Testing: 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.

Aspects of Testing

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

- Expected outcome/ result?
- New change doesn’t break something? (regression)
  - Potentially a very large set of small tests that are run to make sure when a new piece of code is added to a large system it does not cause it to regress to be less useful (that is to reduce the current functions it is expected to have)
- What the user wanted?
  - User Acceptance Test
- Correct performance
  - EG load testing

Testing Tools

Also see 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.

JUnit

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

class MyClass

```
public int method1{
}

public int method2{
}
```

```
@test
public int method1{
}

@test
public int method2{
}
```

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, uses 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(...));
}
```

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

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

Junit Annotations

@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.

Testing Setup

- **"Test suites"**
- **Ideally test only one class**
- **As many test methods as you need**

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 (Tests can be run from multiple environments).
  - IDE
  - Build tool
  - Command line: $ java org.junit.runner.JUnitCore TestClass1 [...other test classes...]

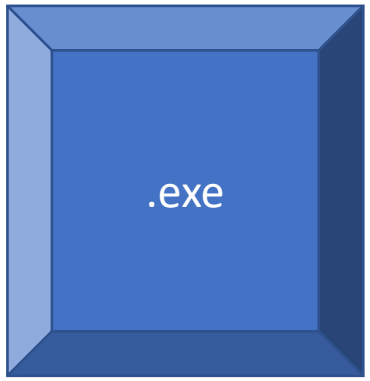
Deployment

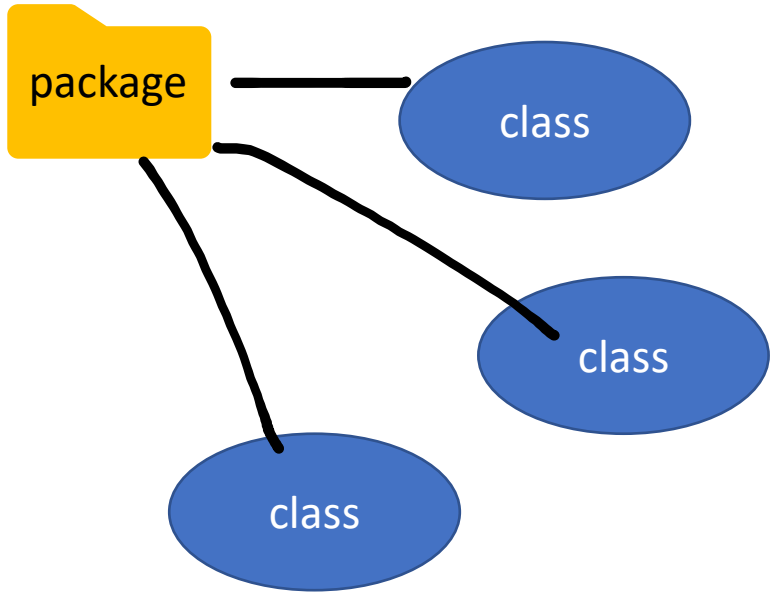
Deployment: *Distributing the program*

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

Deployment: Executable Files

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.





Deployment: Jar Files

- When you need to send files by email you sometimes zip them into an archive.
- You do the same thing in java with a .jar file
- The JVM knows how to read and execute a .jar file without having to unzip it first.

- Either a complementary library (e.g. JDBC drivers)

  java –cp somefile.jar

- Or the main program

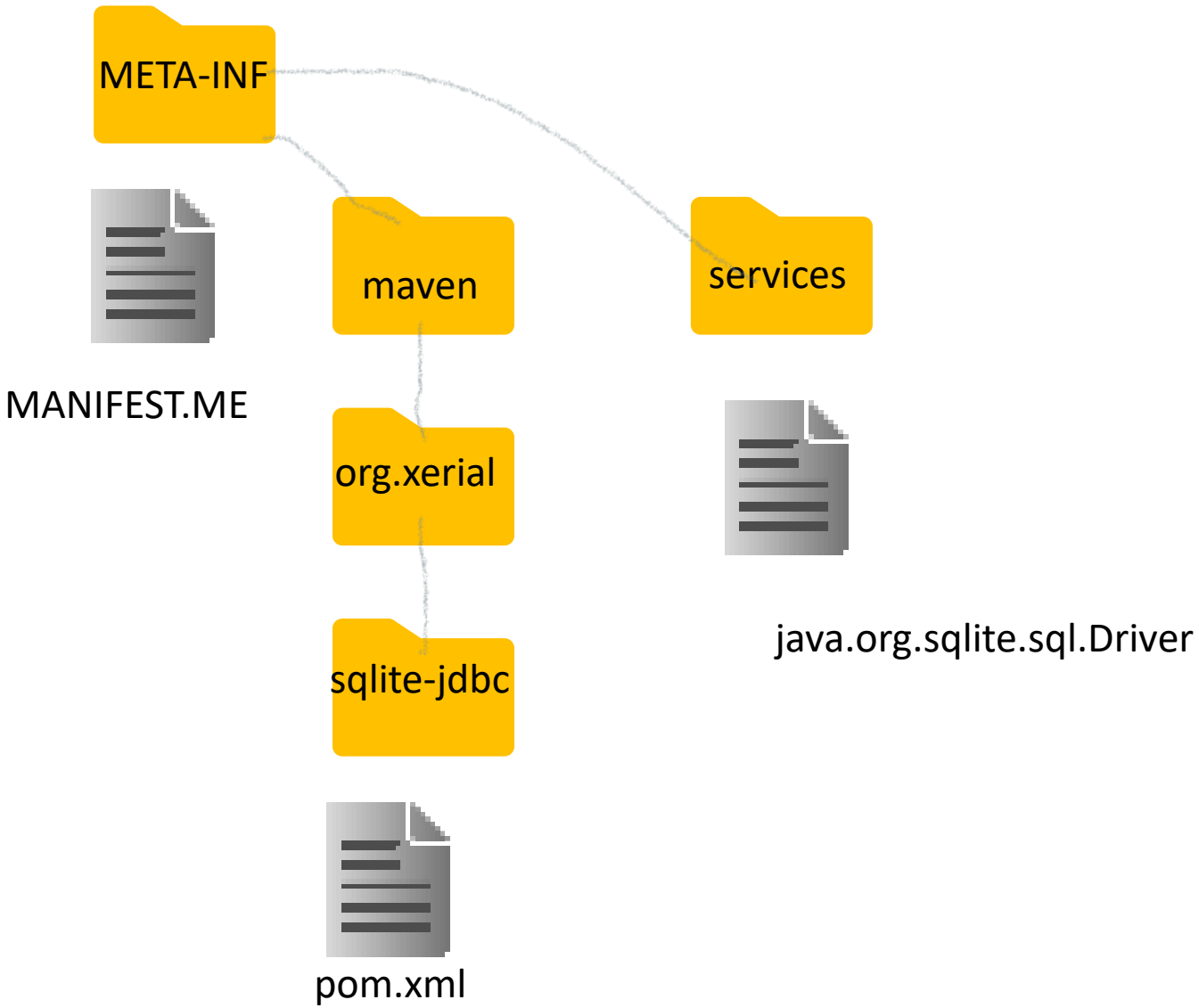
  java –jar myprogram.jar

JAR files: Java Archive

"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.

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 we will do for the SQLite driver as an exercise.



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_192
Bnd-LastModified: 1577134259111
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: 3.30.1
Export-Package: org.sqlite;version="3.30.1.SNAPSHOT";uses:="javax.sql,
 org.sqlite.core,org.sqlite.date",org.sqlite.core;version="3.30.1.SNAP
 SHOT";uses:="org.sqlite,org.sqlite.jdbc4",org.sqlite.util;version="3.
 30.1.SNAPSHOT",org.sqlite.date;version="3.30.1.SNAPSHOT",org.sqlite.j
 avax;version="3.30.1.SNAPSHOT";uses:="javax.sql,org.sqlite,org.sqlite
 .jdbc4",org.sqlite.jdbc4;version="3.30.1.SNAPSHOT";uses:="javax.sql,o
 rg.sqlite,org.sqlite.core,org.sqlite.jdbc3",org.sqlite.jdbc3;version=
 "3.30.1.SNAPSHOT";uses:="org.sqlite,org.sqlite.core"
Import-Package: javax.sql;resolution:=optional
Originally-Created-By: Apache Maven Bundle Plugin
Tool: Bnd-2.1.0.20130426-122213
```

This is the content of the Manifest file.

Jar Files

References to files in a .jar are supposed to be in the .jar, unless they are prefixed by “file:”

<u>References to files:</u>

- file:path
- Operating System

look in the CLASSPATH variable

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

images must be in CLASSPATH