

---

# NSY102

## Conception de logiciels Intranet

TCP java.net, Intro Bluetooth (JSR82)  
MVC

Cnam Paris  
jean-michel Douin, douin au cnam point fr  
version du 7 Avril 2016

**Notes de cours**

---

# Sommaire

---

- **TCP/IP** (*TCP uniquement, UDP : autre support*)
  - Serveur et clients, `java.net.ServerSocket`, `java.net.Socket`
    - Architectures respectives
  - Protocole « java », (`Serializable`)
  - Protocole « maison »
  - Protocole HTTP
- **Serveur Web**
  - Usage du patron « `ThreadPool` »
  - Applette et serveur
- **Serveur et clients Bluetooth**
- **MVC distribué ?**
  - une esquisse
- **Annexes**
  - Patrons Reactor & Acceptor
    - `java.nio.ServerSocketChannel`, `java.nio.SocketChannel`
  - Le Patron HeartBeat
  - Java Web Start
  - `junithttp`

# Bibliographie utilisée

---

- Design Patterns, catalogue de modèles de conception réutilisables de Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides [Gof95]  
International thomson publishing France

## Patron Reactor

<http://www.cs.wustl.edu/~schmidt/PDF/reactor-siemens.pdf>

par Doug Lea : <http://gee.cs.oswego.edu/dl/cpjslides/nio.pdf>

l'original <http://www.laputan.org/pub/sag/reactor.pdf>

<http://jerry.cs.uiuc.edu/~plop/plop99/proceedings/Fernandez3/RACPattern.PDF>

## java.nio,

<http://javanio.info/>

<http://www.cs.brown.edu/courses/cs161/papers/j-nio-ltr.pdf>

<http://javafaq.nu/java-article1102.html> et <http://www.javafaq.nu/java-article1103.html>

## Architecture of a Highly Scalable NIO-Based Server de G.Roth

<http://today.java.net/pub/a/today/2007/02/13/architecture-of-highly-scalable-nio-server.html>

## Java et les réseaux

<http://java.sun.com/docs/books/tutorial/networking/>

<http://monge.univ-mlv.fr/~rousseau/RESEAUJAVA/>

# Pré-requis

---

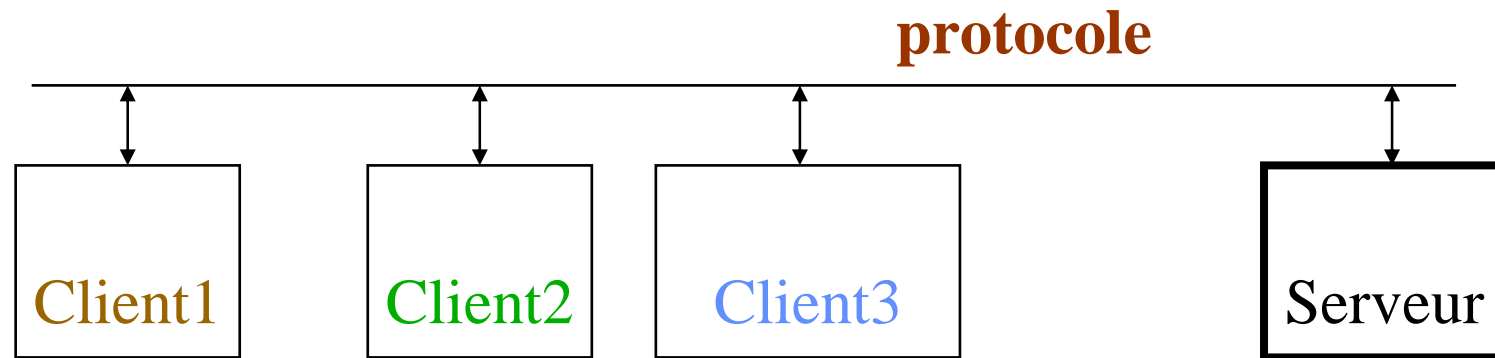
- **Notion**
  - TCP/IP
- **Notion des patrons**
  - Adaptateur
  - Procuration
  - Observateur & MVC

# Contexte

---

- **Appels distants en mode TCP/IP**
  - Point à point avec accusé de réception
  - En détail ici <http://monge.univ-mlv.fr/~rousseau/RESEAUJAVA/tcp.html>
  - telnet, ftp, http, ...
  
- **URL *Uniform Resource Locator*** *une adresse sur internet*
  - <http://jfod.cnam.fr>
  
  - http le protocole
  - [//jfod.cnam.fr](http://jfod.cnam.fr) le nom de la ressource
  
  - <http://jfod.cnam.fr:8999/ds2438/mesures.html>

# Exemples clients / serveurs



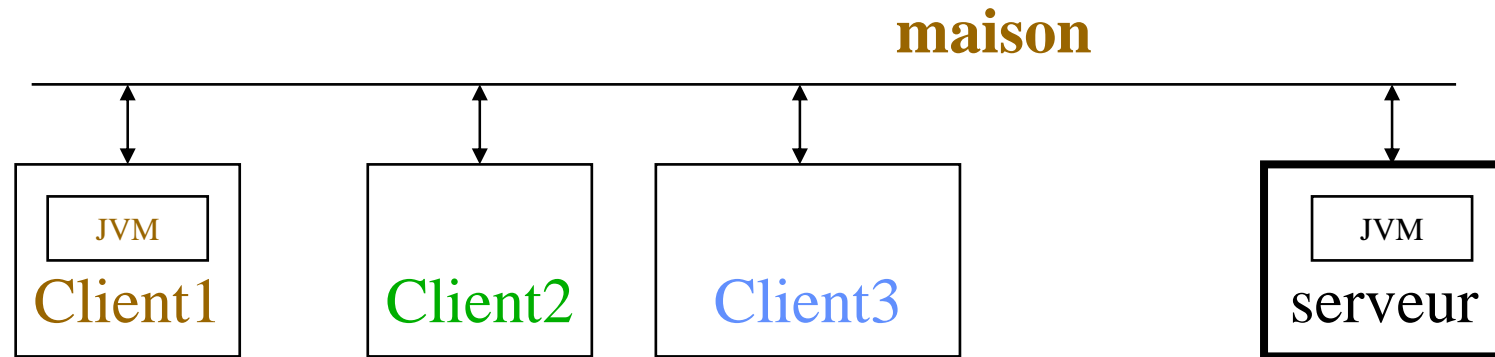
## 1. Le client s'adresse au serveur

- Établit une connexion

## 2. Le serveur satisfait ses clients

- Mode synchrone, analogue à l'appel d'une méthode locale

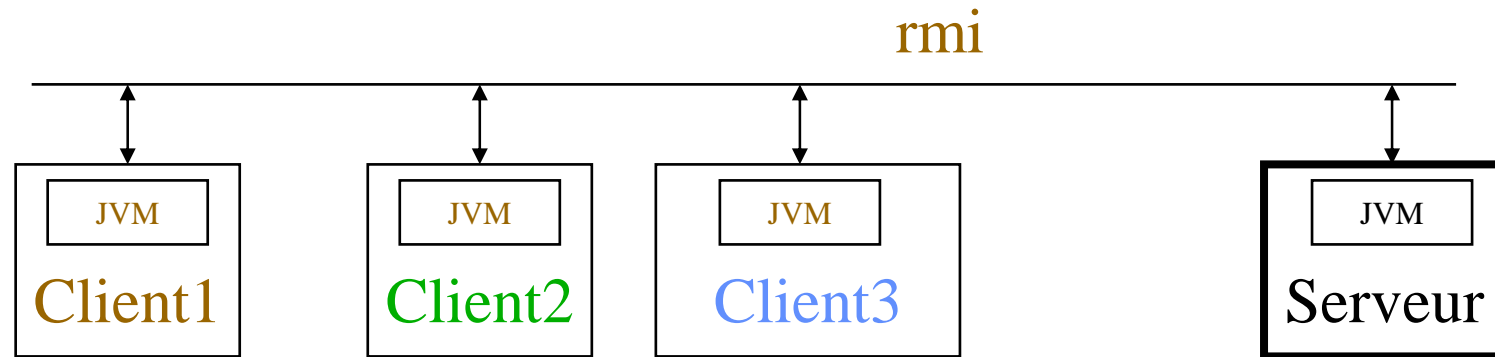
# Appels distants protocole « maison »



- **Le contexte**

- Client Java, **ou autres**
- **Serveur en java ou autre**
- **maison : //serveur/....**

# Appels distants protocole JRMP (rmi)



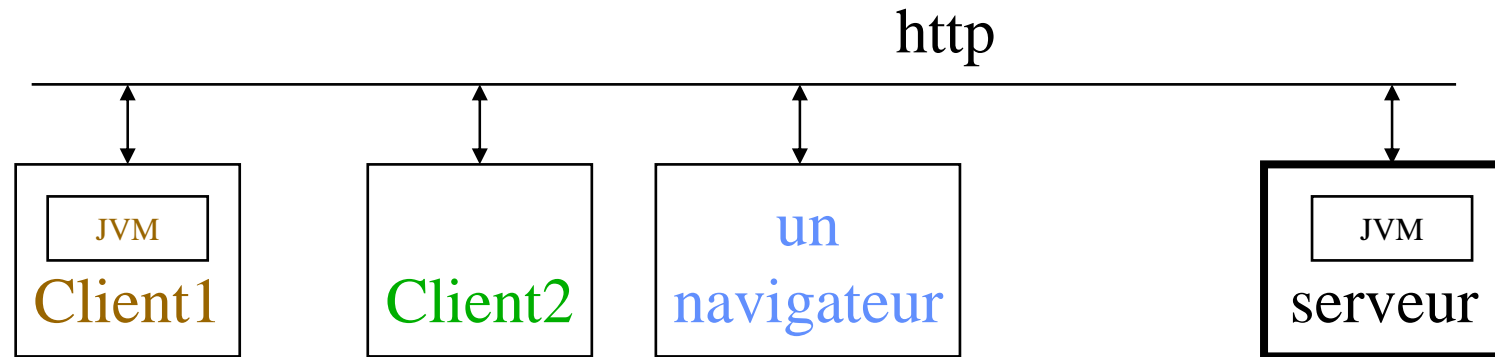
- **Le contexte**

- Clients Java uniquement
- Serveur en java
  - `rmi://serveurDeNoms/service`

- JRMP Java Remote Method Protocol



# Appels distants protocole http



- **Le contexte**

- Client Java(application comme applette), **ou autres**
- Un navigateur
  
- Serveur en java , **ou autres**
  - **http: //serveur/index.html**
  - **Standard, universel ...**

# Implémentation en Java

---

- **Paquetage java.net**

- **Principales classes**

- **ServerSocket**
    - **Socket**
    - **InetAddress**
    - **URLConnection**
    - ...

- **Quelques lignes de sources suffisent ...**

# usage de java.net TCP/IP



- **2 classes essentielles**

## Côté Serveur

- **java.net.ServerSocket**

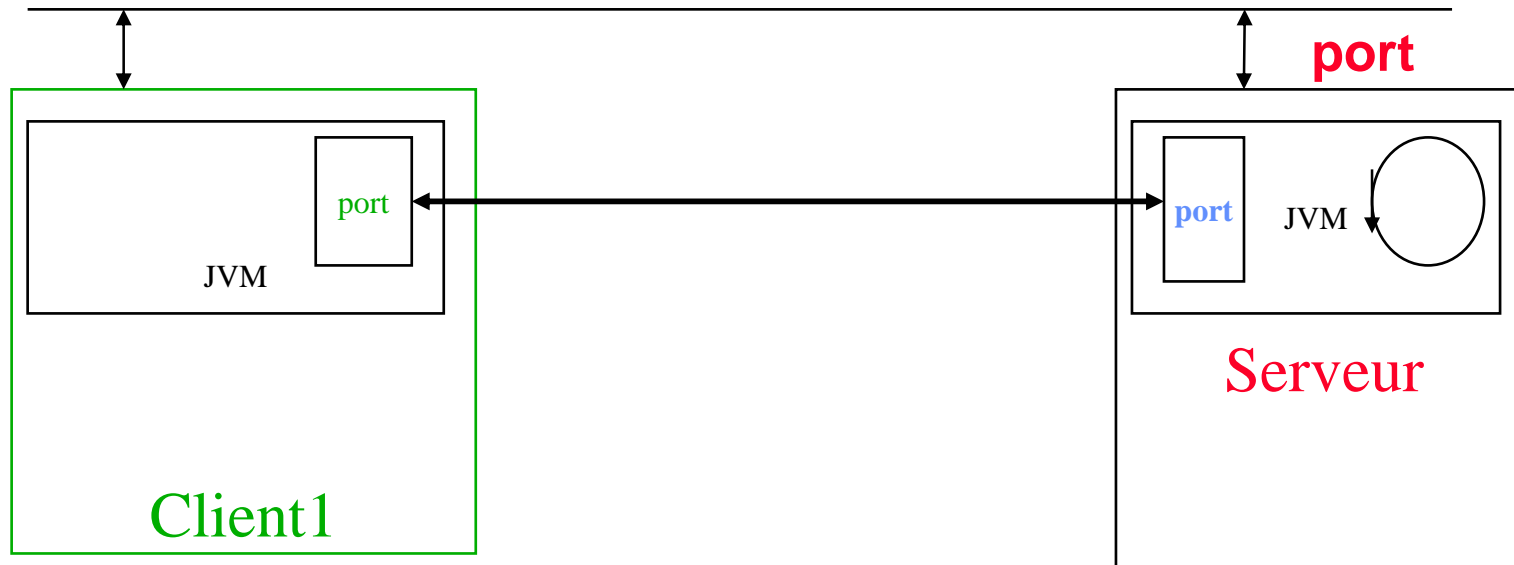
- Méthode **accept()** sur une instance de la classe **ServerSocket**

## Côté Client

- **java.net.Socket**

- Envoi sur une instance de la classe **Socket** de données

# Connexion / Principes



- **Le Serveur attend une requête sur son port**
  - `ServerSocket server = new ServerSocket(port)`
  - `Socket socket = server.accept();`
- **Dès la connexion établie,**
  - une instance de la classe `Socket` est engendrée sur un port temporaire
- **Établir une connexion par le client est effectuée par**
  - `Socket s = new Socket(Serveur, port)`

# 3 exemples

---

- **Serveur et client**

1. **Au protocole « java »**

- les instance transmises « implements » **Serializable**

2. **Au protocole « maison »**

- **Le serveur ne connaît que la commande « parle » et répond « bonjour »**
- **Tout autre commande est ignorée !**

3. **Au protocole http**

- **Seule la méthode GET /index.html HTTP1.0 est possible**
- **Un sous-ensemble donc ...**

# En TCP un serveur (jfod), protocole « java »

---

```
public class Serveur{

    public static void main(String[] args) throws Exception{
        ServerSocket serveur = new ServerSocket(5000);

        while(true) {
            Socket socket = serveur.accept(); // attente active d'un client

// requête
            ObjectInputStream ois= new ObjectInputStream(socket.getInputStream());
            Object obj = ois.readObject();

// réponse
            ObjectOutputStream oos= new ObjectOutputStream(socket.getOutputStream());
            oos.writeObject(obj.toString());

            socket.close();
        }
    }
}
```

# En TCP le Client, protocole « java » (le serveur:jfod)

---

```
public class Client{

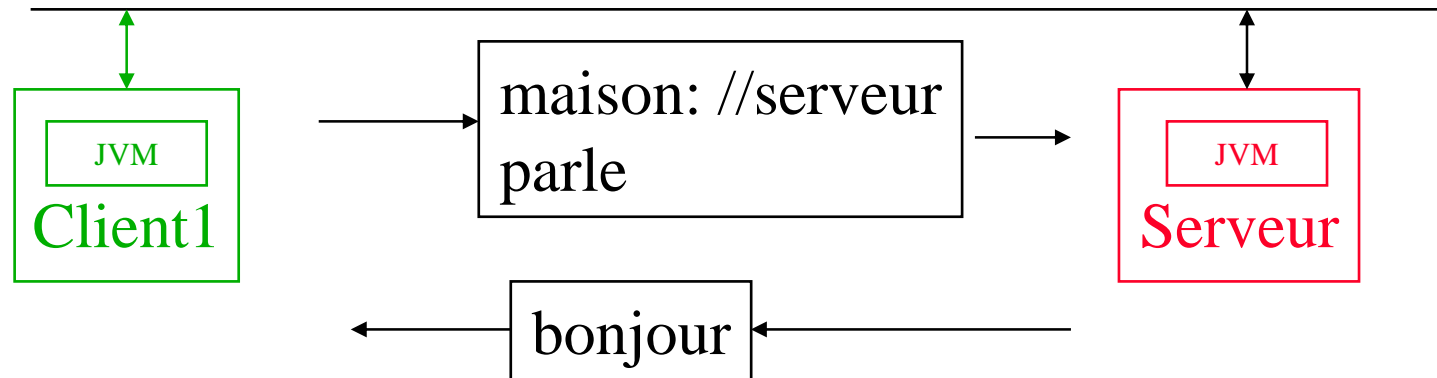
    public static void main(String[] args) throws Exception{
        // ouverture d'une connexion TCP
        Socket socket = new Socket("jfod.cnam.fr", 5000);
        ObjectOutputStream oos= new ObjectOutputStream( socket.getOutputStream());

        // envoi vers le serveur de cette « requête »
        SortedSet<String> l = new TreeSet<String>();
        l.add("TCP");l.add("essai");
        oos.writeObject( l);

        // lecture de la réponse retournée
        ObjectInputStream ois= new ObjectInputStream( socket.getInputStream());
        System.out.println("le serveur retourne : " + ois.readObject());

        socket.close();
    }
}
```

## Exemple 2



- **Au protocole « maison »**
  - **Le serveur ne connaît que la commande « parle » et répond « bonjour »**
  - **Tout autre commande est ignorée !**
- **Client java ou autre**



# Un serveur avec un protocole « maison »

---

```
public class Serveur{

    public static void main(String[] args) throws Exception{
        ServerSocket serveur = new ServerSocket(5000);
        while(true) {
            Socket socket = serveur.accept();

            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
            String cmd = in.readLine();

            // parle !!!
            DataOutputStream out = new DataOutputStream( socket.getOutputStream());
            if(cmd.equals("parle")){
                out.write("bonjour\n".getBytes());
            }else{
                out.write("commande inconnue ?\n".getBytes());
            }
            socket.close();
        }
    }
}
```

# Le client « maison »

---

```
public class Client{

    public static void main(String[] args) throws Exception{
        Socket socket = new Socket("vivaldi.cnam.fr", 5000);
        DataOutputStream out= new DataOutputStream( socket.getOutputStream());
        out.write(args[0].getBytes());
        out.write("\n".getBytes());

        BufferedReader in = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));
        System.out.println(in.readLine());

        socket.close();
    }
}
```

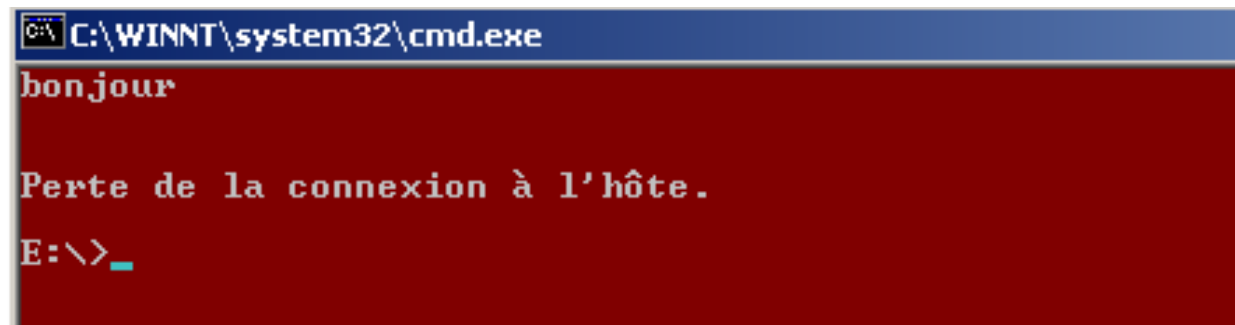
```
H:\NSY102\tp_pattern_correction>java -cp . question3.Client parle
bonjour
H:\NSY102\tp_pattern_correction>java -cp . question3.Client écris
commande inconnue ?
```

# Un client « maison », telnet

- **telnet localhost 5000**
  - **parle // frappe sans écho**



```
C:\WINNT\system32\cmd.exe - telnet localhost 5000
```



```
C:\WINNT\system32\cmd.exe
bonjour
Perte de la connexion à l'hôte.
E:\>_
```

- petit outil utile : tcpview sous windows

# Exemple 3

---

- **Le protocole HTTP**
  - Les méthodes GET, POST, ....
  
- **Mise en œuvre / démo**
  - Usage d'un client telnet sur un site existant
  
  - Une application Java cliente
  
  
  - Un serveur en java

# Protocole HTTP

---

- ***HyperText Transfer Protocol***
  - Au dessus de TCP
- **Les Méthodes**
  - GET /index.html HTTP/1.0
  - HEAD
  - POST
  
  - PUT
  - DELETE
  - TRACE
  - CONNECT
  
  - Voir <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

# Côté serveur, accept

---

```
ServerSocket listen = new ServerSocket(HTTP_PORT);

while(!stopped()){
    try{
        new Connection(listen.accept()); // création d'une instance
    }catch(Exception e){
    }
}

listen.close();
}
```

**listen.accept() est bloquant**

Chaque requête engendre la création d'une instance de la classe **Connection**  
Et chaque instance créée engendre à son tour un « **Thread** »

## Côté serveur, à chaque Connection un Thread

---

```
public class Connexion extends Thread{
...
public Connexion(Socket s){
    this.s = s; start();
}

public void run(){
    try{
        BufferedReader is = new BufferedReader(
            new InputStreamReader(s.getInputStream()));
        DataOutputStream os =
            new DataOutputStream(s.getOutputStream());

        // analyse du contenu au bon protocole HTTP

        // envoi du document
    }
}
```

## Côté serveur, accept « peut-être »

---

```
ServerSocket listen = new ServerSocket(HTTP_PORT);
listen.setSoTimeout(TIME_OUT);
while(!stopped()){
    try{
        new Connection(listen.accept());
    }catch(SocketTimeoutException e){
        // ici délai de garde échou
    }catch(Exception e){
    }
}
listen.close();
}
```

### Méthode accept avec délai de garde

exception `SocketTimeoutException` à l'échéance



# Schéma avec Un Pool de Thread

```
class WebServer { // 2004 JavaOneSM Conference | Session 1358
    Executor pool = Executors.newFixedThreadPool(7);

    public static void main(String[] args) {
        ServerSocket socket = new ServerSocket(80);
        while (true) {
            final Socket s = socket.accept();
            Runnable r = new Runnable() {
                public void run() {
                    BufferedReader is = new BufferedReader(
                        new InputStreamReader(s.getInputStream()));
                    DataOutputStream os =
                        new DataOutputStream(s.getOutputStream());
                    // analyse du contenu au bon protocole HTTP
                    // envoi du document
                }
            };
            pool.execute(r);
        }
    }
}
```

# Requête GET avec telnet

---

- Un client telnet et un site du Cnam

- telnet jfod.cnam.fr 80

- GET /index.html HTTP/1.0 (frappe sans écho, ce n'est pas un serveur telnet...)

*HTTP/1.0 200 OK*

*Last-Modified: Thu, 08 Feb 2007 14:55:29 GMT*

*Date: Thu, 08 Mar 2007 10:33:55 GMT*

*Server: Brazil/1.0*

*Content-Length: 7624*

*Content-Type: text/html*

*Connection: close*

*<HTML>*

*<HEAD>*

*<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">*

*.....*

**Le résultat est retourné, le source du fichier index.html précédé de quelques informations**

# Requête GET en Java

---

- **L'essentiel**

- Créer une URL
- Ouvrir une connexion
  - Écrire et lire sur les flots associés

- **Classe java.net.URL**

- **Classe java.net.URLConnection**

- `URL url = new URL("http://jfod.cnam.fr/index.html");`
- `URLConnection connection = url.openConnection();`

# Requête GET au complet

---

```
public void testGET()throws Exception{
    URL url = new URL("http://jfod.cnam.fr/index.html" );
    URLConnection connection = url.openConnection();

    BufferedReader in = new BufferedReader(
        new InputStreamReader(connection.getInputStream()));

    String inputLine = in.readLine();
    while(inputLine != null){
        System.out.println(inputLine);
        inputLine = in.readLine();
    }
    in.close();
}
```

# Requête GET avec paramètres

---

```
public void testGET()throws Exception{
    URL url =
        new URL("http://jfod.cnam.fr:8999/ds2438/?listAll=on" );
    URLConnection connection = url.openConnection();
    connection.setDoInput(true);

    BufferedReader in = new BufferedReader(
        new InputStreamReader(connection.getInputStream()));

    String inputLine = in.readLine();
    while(inputLine != null){
        System.out.println(inputLine);
        inputLine = in.readLine();
    }
    in.close();
}
```

# Requête POST

---

```
URL url = new URL("http://jfod.cnam.fr/index.html");
URLConnection connection = url.openConnection();
```

```
connection.setDoInput(true);
connection.setDoOutput(true);
```

```
PrintWriter out = new PrintWriter(connection.getOutputStream());
out.print("listAll=on");
```

```
out.close();
```

```
BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
String inputLine = in.readLine();
while(inputLine != null){
    System.out.println(inputLine);
    inputLine = in.readLine();
}
in.close();
```

# Requête GET ou POST avec un Thread, un schéma

```
public class Connexion extends Thread{
...
public Connexion(Socket s){
    this.s = s; result = new String(); start();
}

public void run(){
try{
    BufferedReader is = new BufferedReader( new InputStreamReader(s.getInputStream()));
    DataOutputStream os = new DataOutputStream(s.getOutputStream());
    // analyse du contenu au bon protocole HTTP
    // affectation du résultat (result)

```

- 
- public String result(){
- try{
- **this.join();**
- }catch(InterruptedException ie){}
- return result;
- }

# Requête GET ou POST, Future<T>, un schéma

```
private static class Connexion implements Callable<String>{
    ...
    private Future<String> future;

    private static ExecutorService executor;
    static{ executor = Executors.newSingleThreadExecutor();}

    public Connexion(String url, String parametres){
        this.url = url;
        this.parametres = parametres.trim();
        this.result = new String("");
        this.future = executor.submit(this);
    }

    public String result(){
        try{
            return future.get();
        }catch(Exception e){
            return "";
        }
    }

    public String call(){
        try{
            URL urlConnection = new URL(url);
            URLConnection connection = urlConnection.openConnection();
            ...
        }catch(Exception e){
            this.result = "";
        }
        return result.toString();
    }
}
```



# Quelques classes utiles

---

- **InetAddress**
  - Adresse IP en « clair »
- **URL**
  - Pour **U**niform **R**esource **L**ocator, sur le **w**ww
- **URLConnection**
  - Une classe abstraite, super classe de toutes les classes établissant un lien entre une application et une URL
  - **Sous-classes**
    - **HttpURLConnection, JarURLConnection,...**
  - Usage du Patron Fabrique afin d'écrire son propre gestionnaire de protocole
    - Voir <http://monge.univ-mlv.fr/~rousseau/RESEAUJAVA/java.url2.html>
    - Méthode `URLConnection.setContentHandlerFactory( ...);`

# java.net.InetAddress

---

- **L'adresse IP de la machine locale**
  - `InetAddress.getLocalHost().getHostAddress()`
  
- **Un client sur la même machine que le serveur**
  - `new Socket(InetAddress.getLocalHost(), 8999);`
  
- **Démonstration**
  - <http://jfod.cnam.fr/NSY102/serveurs/OneShotHttpd.java>

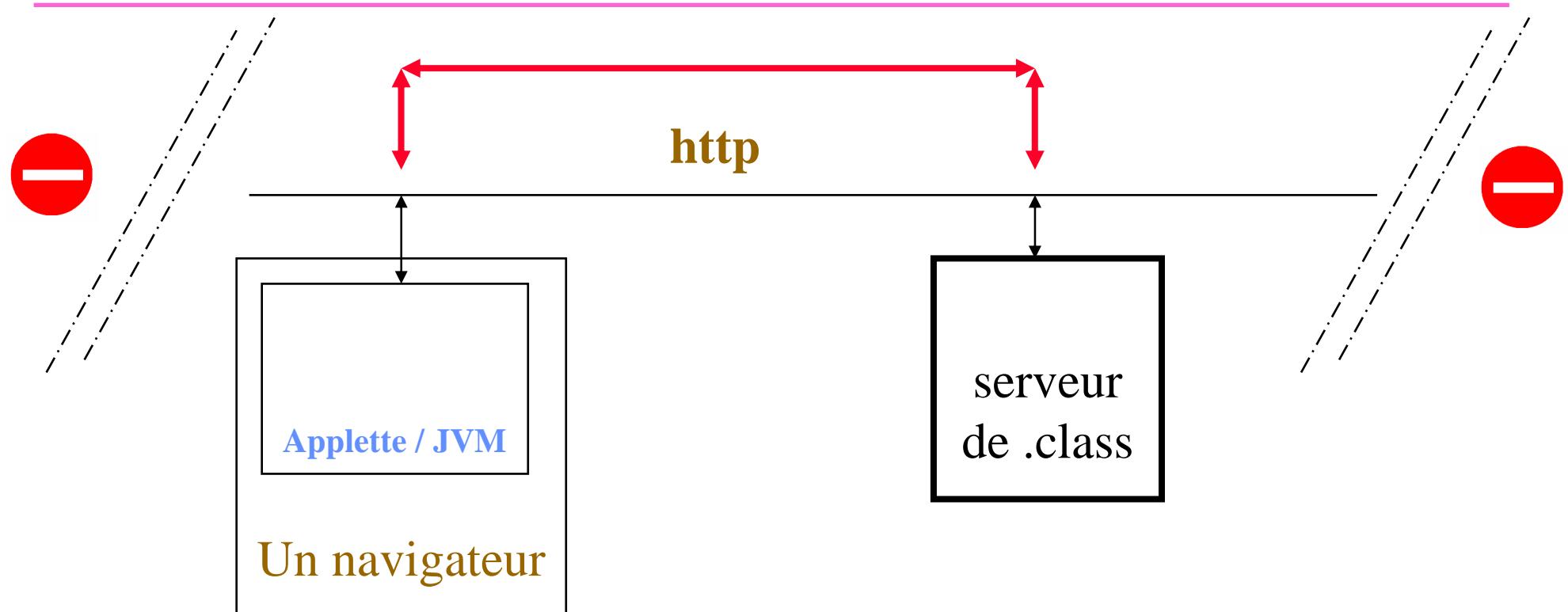
# Navigateur et serveurs

---

- **Incompatible ?**

- Une applette est une application/Java comme une autre
- Un serveur web est une application
  
- Donc
  
- Utile/inutile ?
  
- Apparenté AJAX ... Asynchronous
  - ( AJAX : un serveur java script installé côté client)

# L'Applette contient un serveur Web



1. L'applette téléchargée contient un serveur HTTP
  2. Le navigateur héberge donc un « serveur Web » (côté client)...
  3. **Attention : un seul client est possible \***
    - Le serveur depuis lequel l'applette est issue (et en localhost ...)
- \* (avec la stratégie de sécurité standard, qu'il n'est pas recommandé de modifier(rappel))

# L'applette en démo, ici port 8100

The top screenshot shows a Mozilla Firefox window titled "AppletteServeurWeb Applet - Mozilla Firefox". The address bar contains the file path: `file:///H:/NSY102/appletteServeurWeb/AppletteServeurWeb.html`. The main content area displays the text: `[127.0.0.1]_test = <<< appletteCommeServeurWeb >>>`. An arrow points from the text "Site de téléchargement" to the address bar.

The bottom screenshot shows a Mozilla Firefox window titled "Mozilla Firefox". The address bar contains the URL: `http://localhost:8100/test/?test=appletteCommeServeurWeb`. The main content area displays the text: `ok`. An arrow points from the text "requête" to the address bar, and another arrow points from the text "résultat" to the content area.

- **Cette applette Serveur Web**

- affiche la valeur du paramètre test et
- retourne au client **ok**

- Le source ici <http://jfod.cnam.fr/NSY102/appletteServeurWeb/>

- L'applette est ici <http://jfod.cnam.fr/NSY102/appletteServeurWeb/AppletteServeurWeb.html>

# Source de l'applette : un extrait

```
public class AppletServeurWeb extends JApplet implements Runnable{
    public static final int TIME_OUT = 500;

    private Thread    local;
    private String    urlPrefix  = "test";
    private int       port      = 8100;

    public void init(){
        JRootPane rootPane = this.getRootPane();
        // IHM et paramètres

        local = new Thread(this);
        local.start();
    }

    public void run(){
        ServerSocket listen = new ServerSocket(this.port);
        listen.setSoTimeout(TIME_OUT);
        while(!local.isInterrupted()){
            Socket      socket = null;
            BufferedReader in    = null;
            DataOutputStream out  = null;
            try{
                socket = listen.accept();
                in     = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                out    = new DataOutputStream( socket.getOutputStream());

                // analyse du contenu au bon protocole HTTP
                // envoi du document
            }
        }
    }
}
```

# Java Web Start & Serveur Web

---

- **Et les applications ?**
- **JavaWebStart**
  - Téléchargement d'applications Java
  - Une archive Java signée
  - Recherche automatique de la dernière version
  - Console> javaws
  - Ou depuis un navigateur
  
- Cette application pourrait être un serveur Web

# Architecture distribuée et HTTP

---

- **Hypothèses**

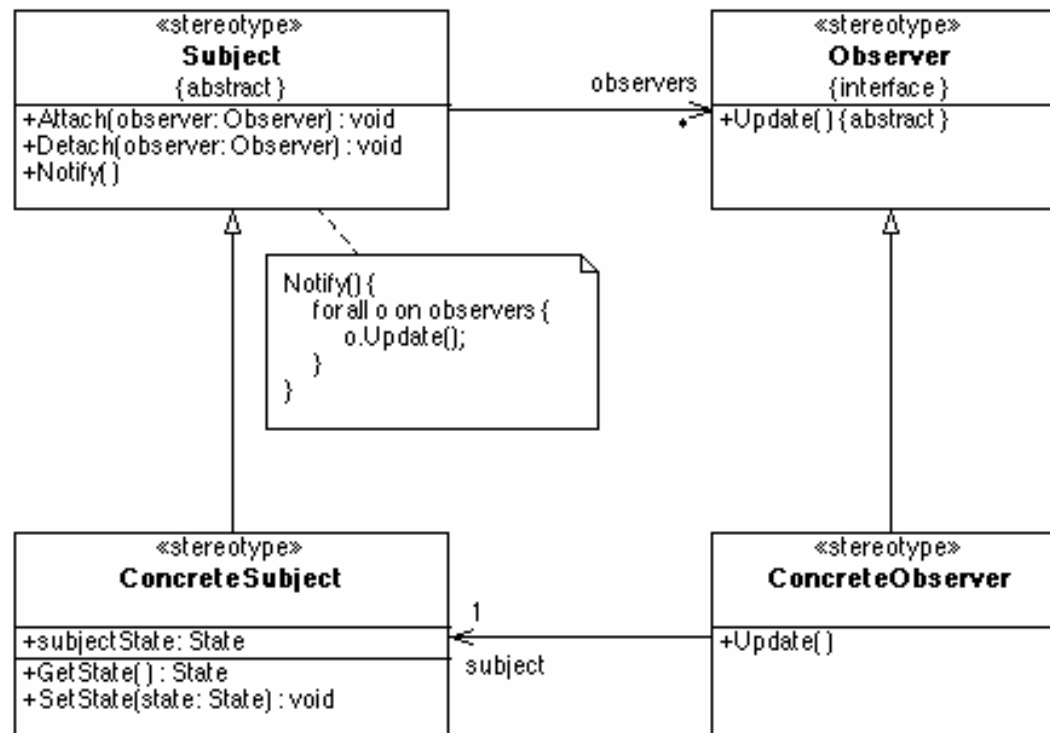
- **Toute machine connectée possède un serveur Web/HTTP**
  - **En application autonome / et ou téléchargée**
  - **Dans un navigateur comme une applette**

- **Premier essai d'architecture**

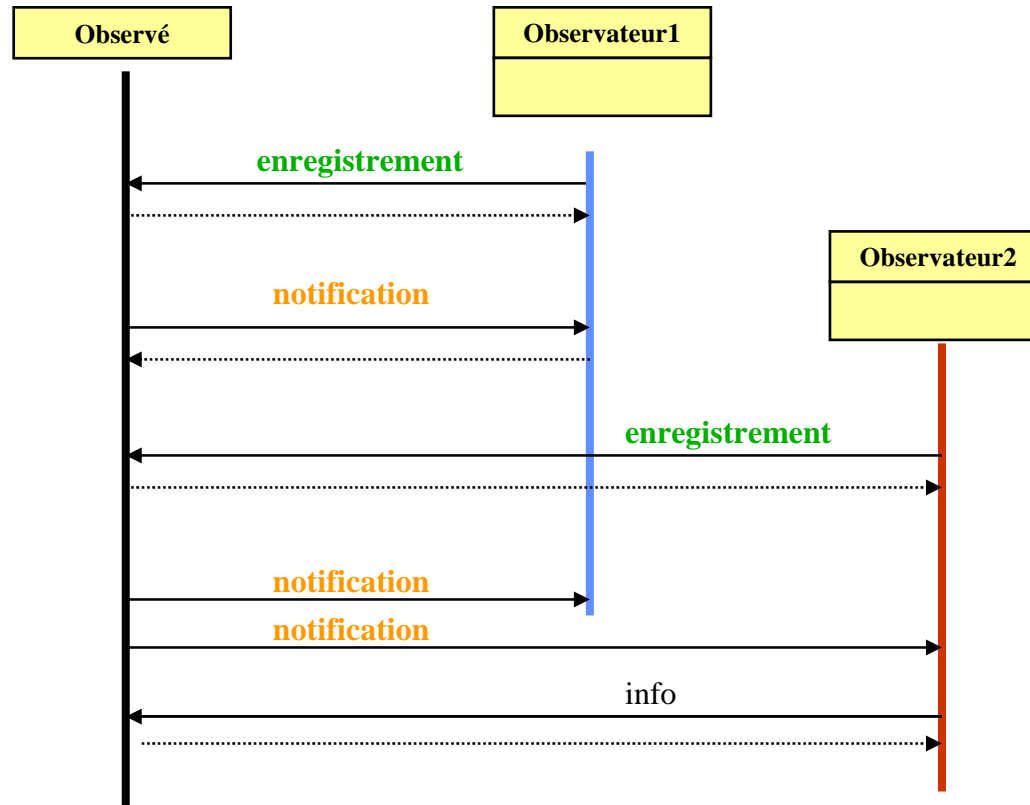
- **Patron Observateur/Observé (le retour)**
  - **Lors d'un changement d'état, notification aux observateurs inscrits**



# Observateur/Observé : l'original



# Diagramme de séquence



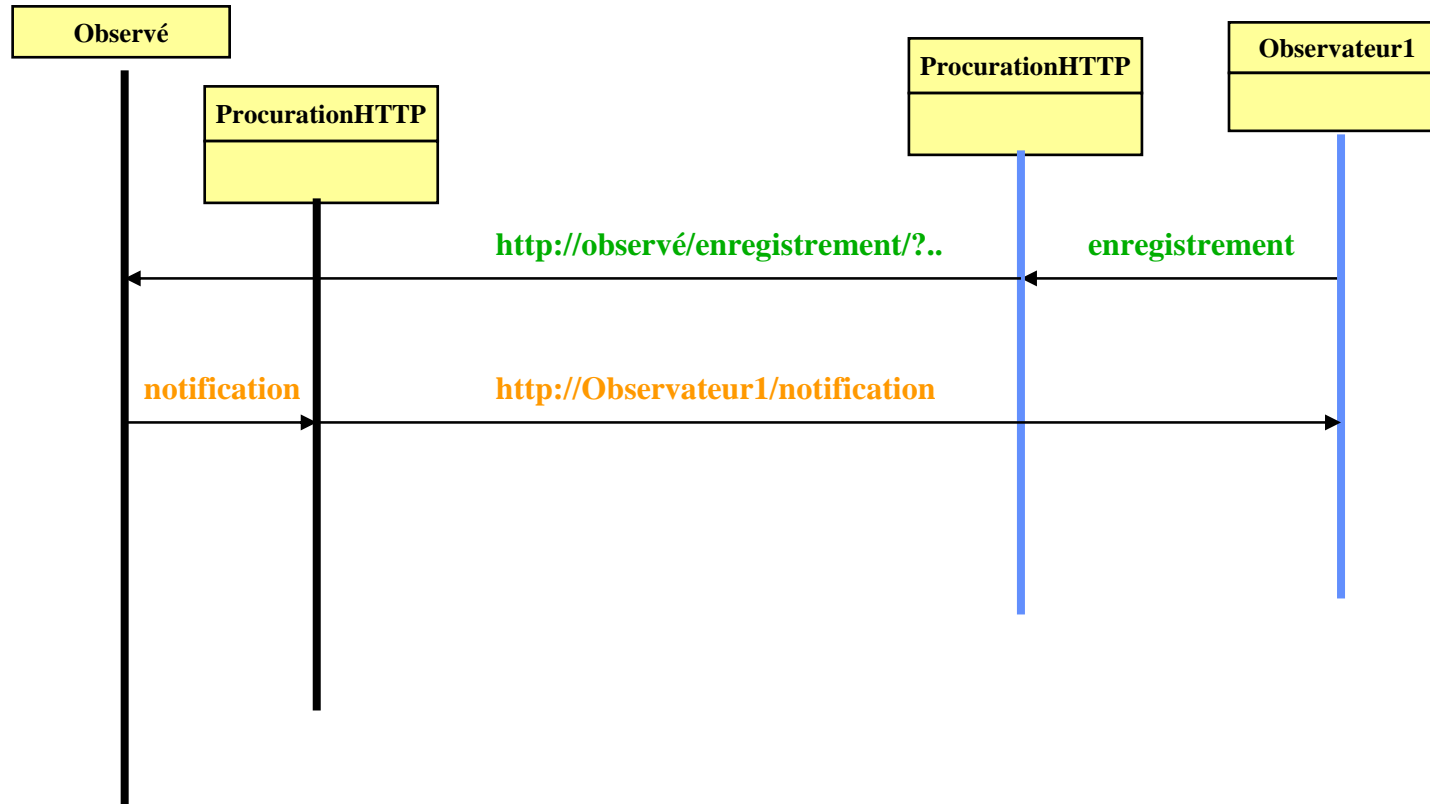
**Adéquation**  
enregistrement

<http://observé/addObserver?url=http://observateur1/update/>

notification

<http://observateur1/update/?arg=evt>

# Patron Procuration, ( s'abstraire du protocole)



- **Procuration à l'émission/réception**
  - Avons-nous une adéquation appel de méthodes/ requêtes HTTP
  - Discussion ...

# Exemple de mise en œuvre méthode addObserver

---

```
public class ProcurationHTTP extends Observable{
    private String urlString;
    private final ExecutorService executor;

    public HttpObservable(String urlString){
        super();
        this.urlString = urlString;
        this.executor = Executors.newCachedThreadPool();
    }

    public void addObserver(Observer obs){
        super.addObserver(obs);
        Future<String> res = executor.submit(new RequeteHTTP());
    }

    ...
}
```

# RequêteHTTP en Java (Future et Callable)

---

```
private class RequeteHTTP implements Callable<String>{
    private String result = "";

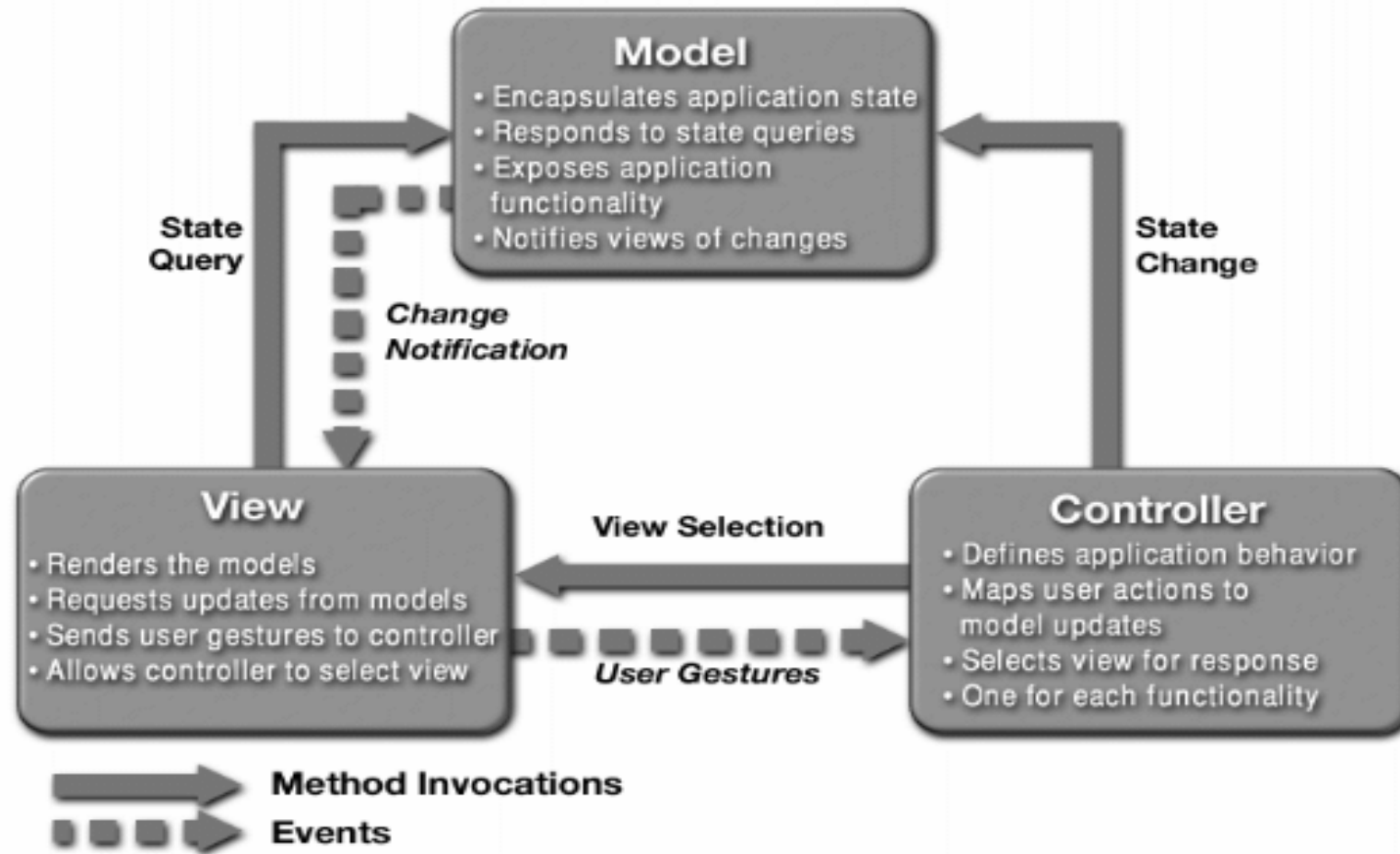
    public String call(){
        try{
            URL url = new URL(urlString + paramètres);
            URLConnection connection = url.openConnection();

            connection.setDoInput(true);

            BufferedReader in = new BufferedReader(
                new InputStreamReader(connection.getInputStream()));

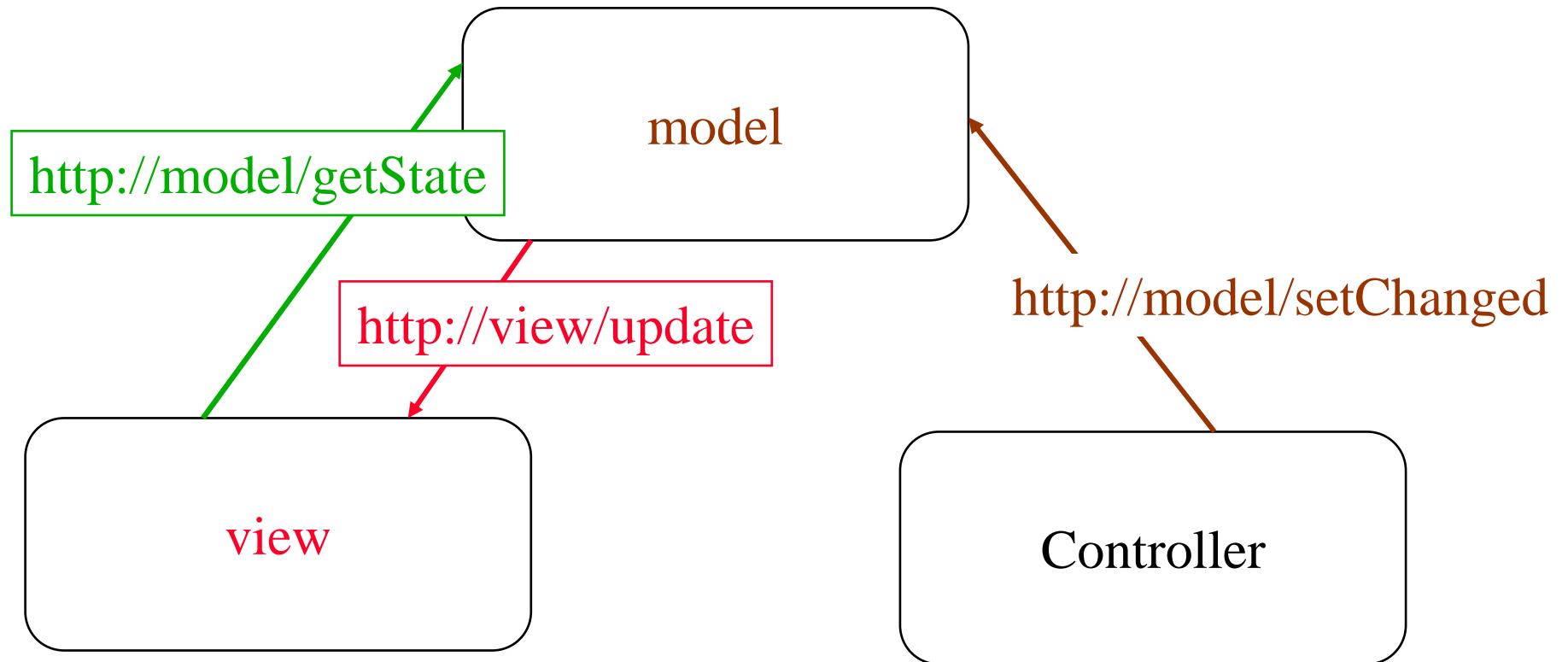
            String inputLine = in.readLine();
            while(inputLine != null){
                result = result + inputLine;
                inputLine = in.readLine();
            }
            in.close();
        }catch(Exception e){
            e.printStackTrace();
        }
        return result;
    }
}
```

# MVC rappel



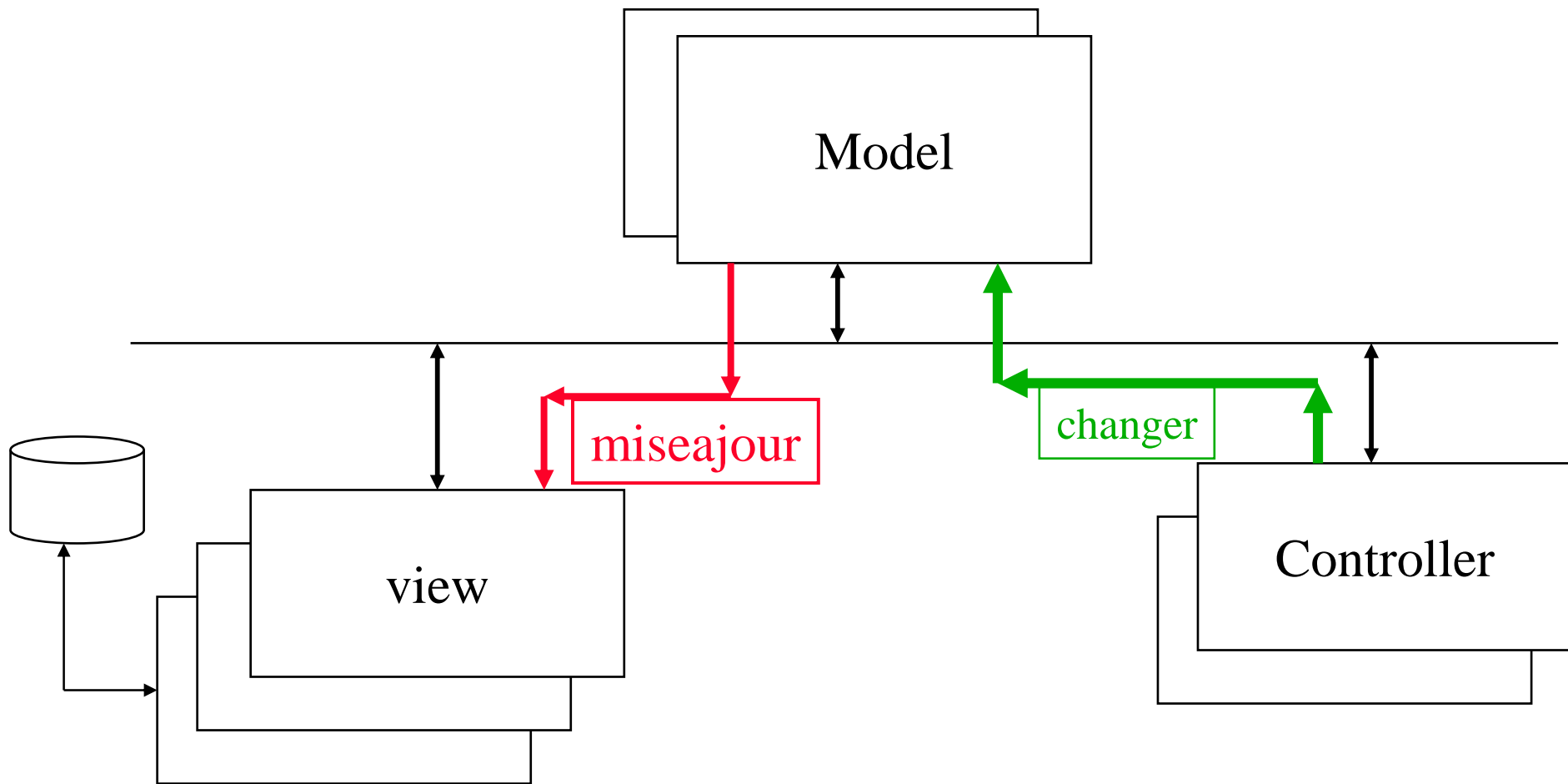
- <http://java.sun.com/blueprints/patterns/MVC-detailed.html>

# MVC distribué



- **Adéquation appels de méthodes / requêtes HTTP**
  - Application existante + Procurations

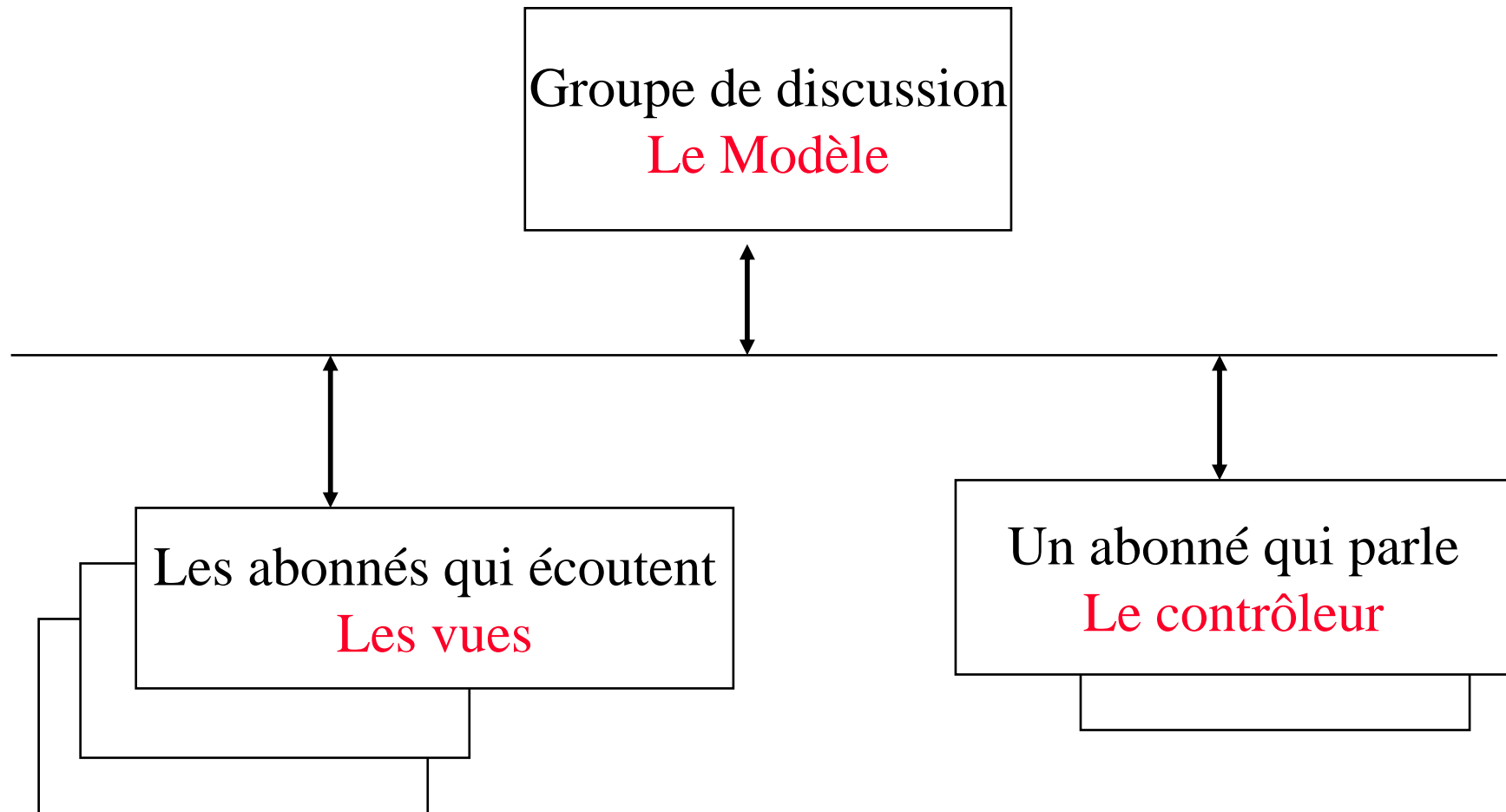
## 3 Vues, « 2 modèles », 2 contrôleurs



- Un deuxième Modèle éventuel pour la redondance
- Une Vue peut assurer la persistance

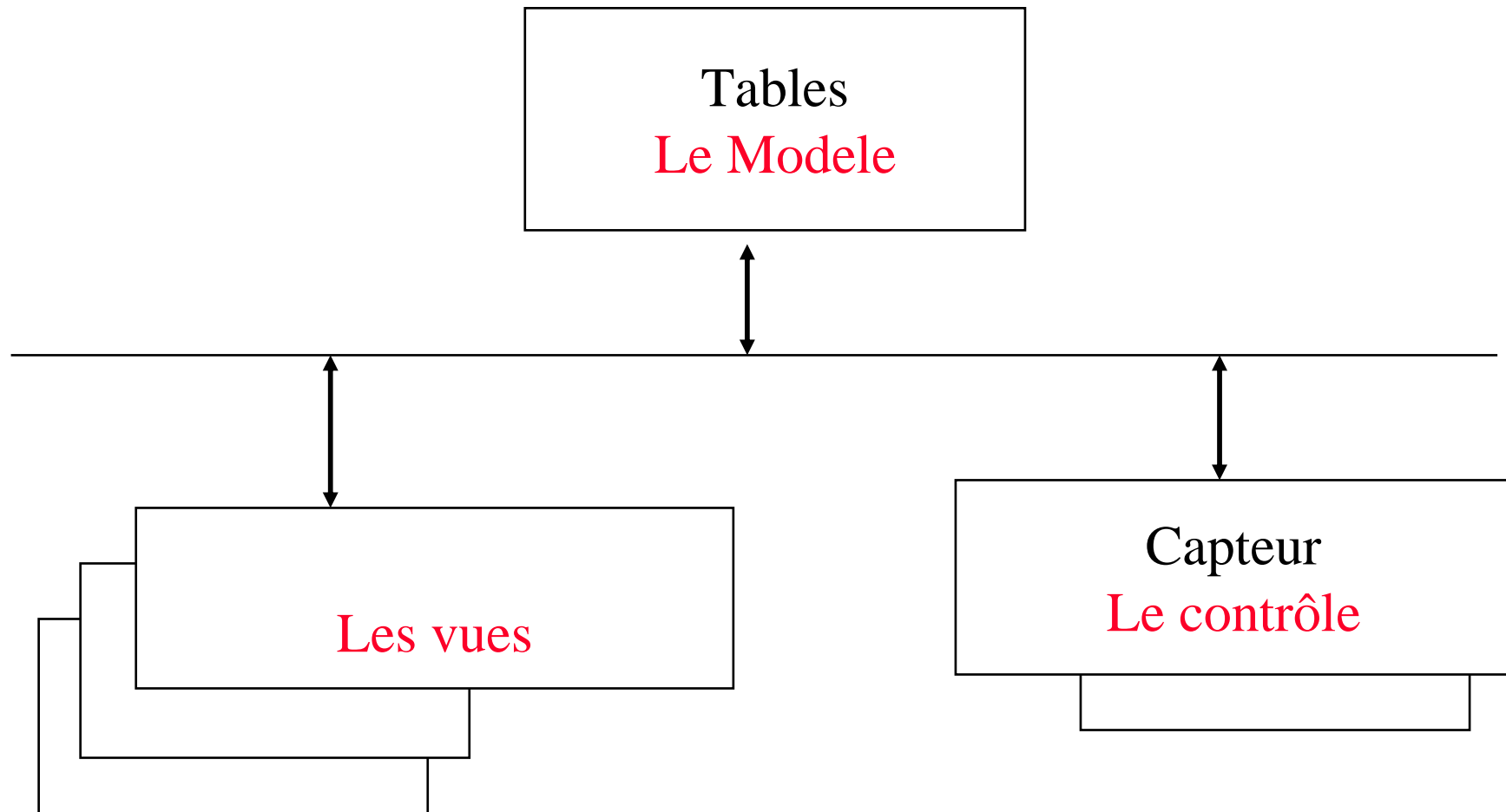


# Un exemple : un chat/ publish-subscribe



- Une instance de MVC

## Autre Application possible



- **Même architecture**

# Une démonstration en ligne au Cnam

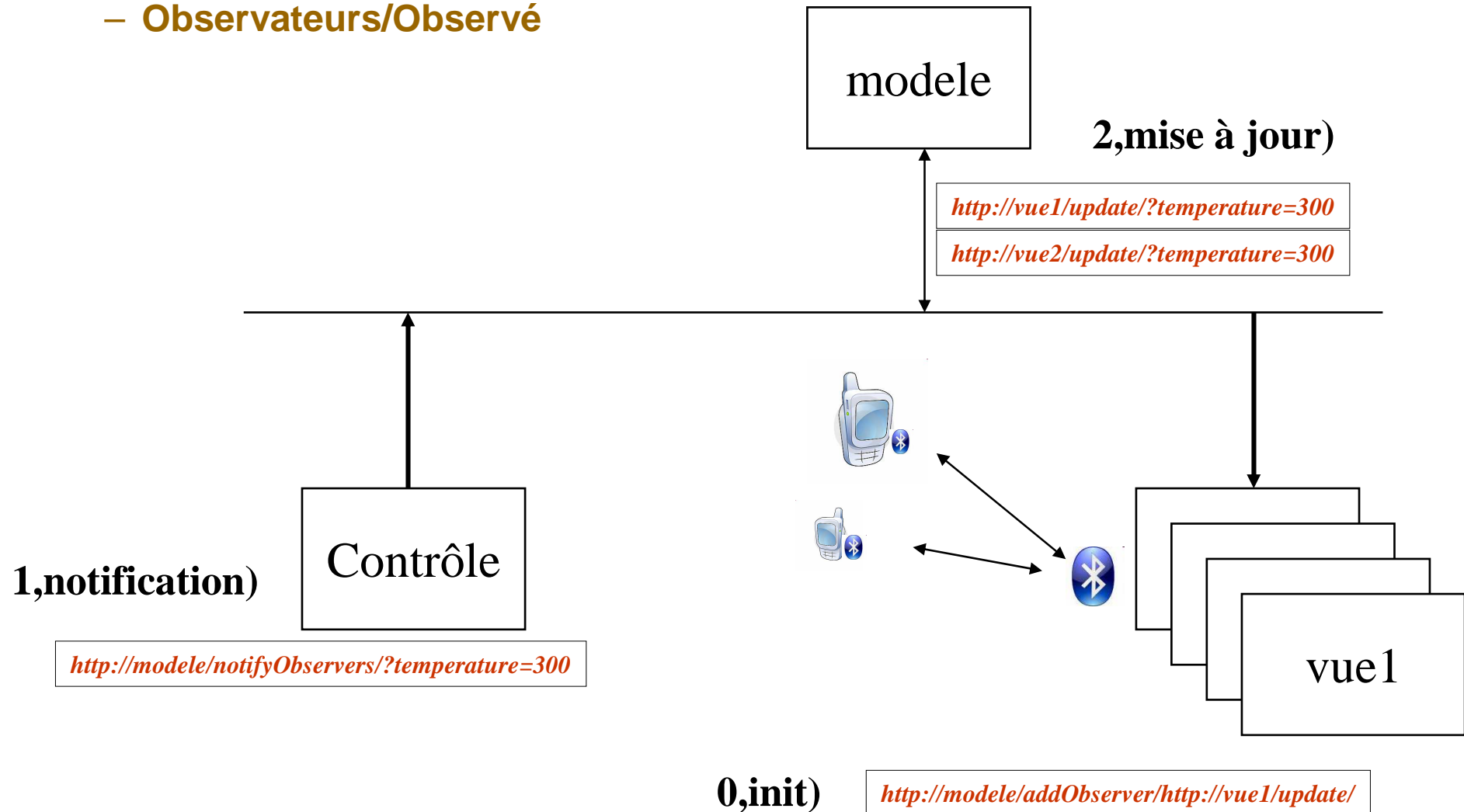
---

- **Un logiciel de causerie**
  - En intranet seulement
    - Contraintes des différents pare-feux
- **Un gestionnaire du groupe de discussion est en**
  - <http://vivaldi.cnam.fr:8200/chat/>
  - Les sources ici [http://jfod.cnam.fr/NSY102/http\\_causerie.jar](http://jfod.cnam.fr/NSY102/http_causerie.jar)
- **Ajouter un client**
  - Depuis votre navigateur
    - <http://vivaldi.cnam.fr:8200/chat/?cmd=applette&nom=pierre&port=9100>
  - Ou bien depuis une console avec appletviewer
    - appletviewer <http://vivaldi.cnam.fr:8200/chat/?cmd=applette&nom=paul&port=9900>

# Autre exemple ?

- HTTP

- Observateurs/Observé



# Bluetooth

---

- **Protocole de communication sans fil**

peu chère, peu consommatrice d'énergie ... (enfin presque / wifi),  
adapté aux mobiles ...

- **Spécifications ici <http://www.bluetooth.com/bluetooth/>**

– *Bluetooth is not a one-on-one data transmission technology so it can communicate with up to eight devices within its transmission radius at one time*

- **Vocable « habituel » :**

– *Découverte multicast des services*

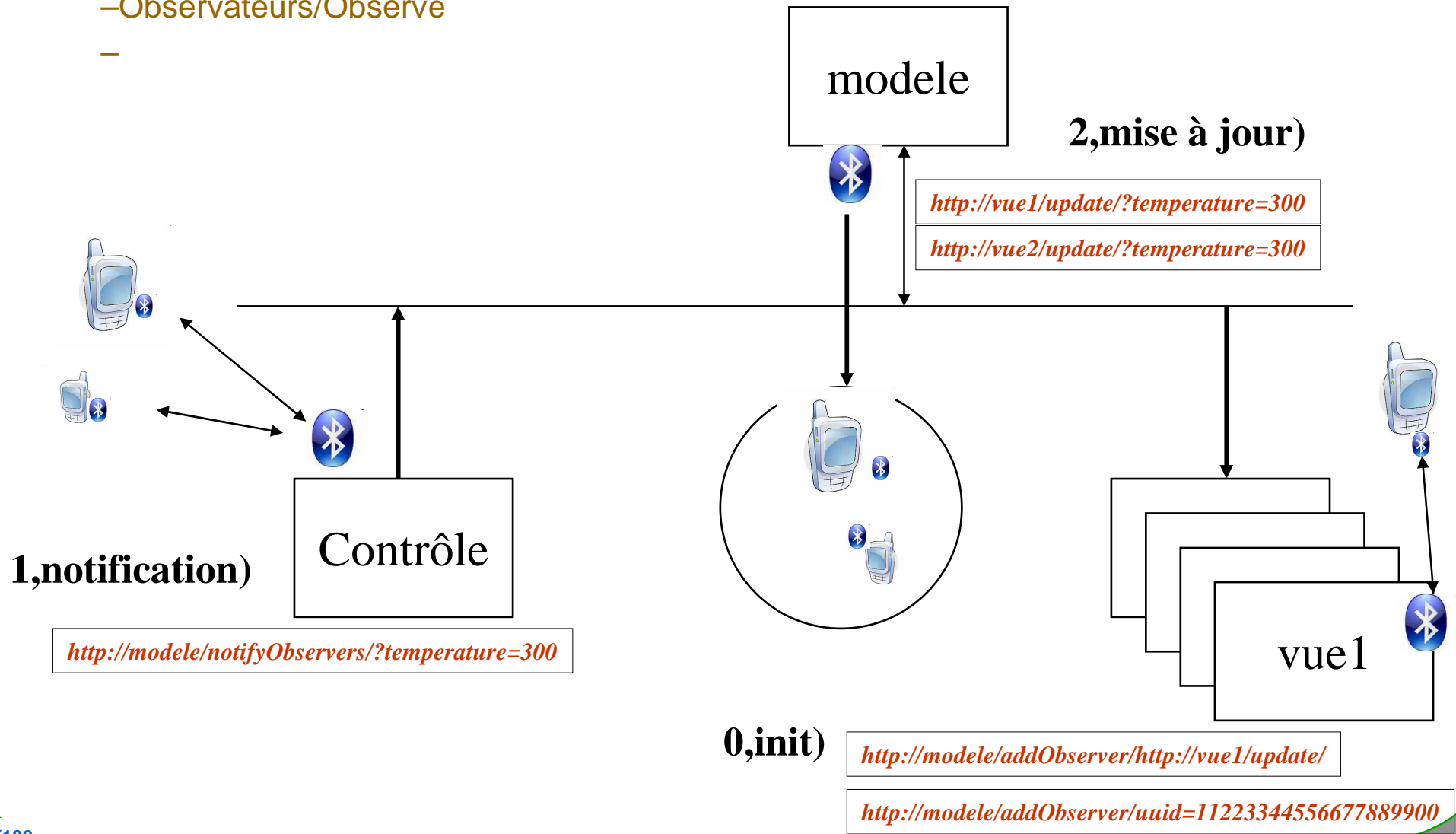
- *Client/serveur, serveur/serveur*

# Un Objectif : MVC HTTP+BT ?

- HTTP/BT

- Observateurs/Observé

- 



# JSR 82, prémisses

---

- **javax.bluetooth.\*;**

- Nommage,
- Découverte
- Recherche de services
- Communication

- `btsp://localhost:{UUID} + (flux habituels)`

- <http://java.sun.com/javame/reference/apis/jsr082/javax/bluetooth/package-summary.html>
- <http://www.jsr82.com>
- <http://www.bluecove.org/>

- Chaque périphérique BT possède une adresse (*physique*) sur 48 bits et unique
  - (MAC adresse)
  - Un nom lui est associé (en général)

# Exemple : quelle est mon adresse BT ?

---

```
import javax.bluetooth.LocalDevice;

public class QuiSuisJe{
    public static void main(String[] args)throws Exception{

        LocalDevice localDevice = LocalDevice.getLocalDevice();

        System.out.println(localDevice.getBluetoothAddress() + " : " +
            localDevice.getFriendlyName());
    }
}
```

## Depuis un PC, J2SE

```
javac -cp .;bluecove-2.1.0.jar QuiSuisJe.java
java -cp .;bluecove-2.1.0.jar QuiSuisJe
```

<http://www.bluecove.org/bluecove/apidocs/>

```
BlueCove version 2.1.0 on winsock
0019EF0117DA : LMI90
BlueCove stack shutdown completed
```



## Parmi mes voisins ...

---

- **Parmi les périphériques BT aux alentours**
  - **Quel est celui qui possède ce service ...**
    - **Apparenté à la découverte multicast...**
  
  - **Un service : un UUID**
    - **new UUID("102030405060708090A0B0C0D0E0F011" ,false)**
      - 128 bits ... pourquoi pas celui-ci ...
  
  - **Plusieurs BT peuvent répondre pour un service souhaité**
    - « dépôt d'un fichier », mp3 ...
    - redondance

# A la découverte de ... obtention d'un agent

---

```
import javax.bluetooth.LocalDevice;
import javax.bluetooth.DiscoveryAgent;

public class ALaDécouverteDe{

    public static void main(String[] args) throws Exception{
        LocalDevice local = LocalDevice.getLocalDevice();

        local.setDiscoverable( DiscoveryAgent.GIAC );
        // GIAC General inquire Access Code

        DiscoveryAgent agent = local.getDiscoveryAgent();
    }
}
```

## un agent capable de tout faire

Sélection d'un service

Effectuer une recherche exhaustive

# Adéquation UUID / URL

---

```
LocalDevice local = LocalDevice.getLocalDevice();  
local.setDiscoverable( DiscoveryAgent.GIAC ); // General Inquire Access Code  
DiscoveryAgent agent = local.getDiscoveryAgent();
```

```
UUID uuid = new UUID("102030405060708090A0B0C0D0E0F011", false);
```

```
String connString = agent.selectService(uuid,  
    ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
```

```
System.out.println("connString : " + connString);
```

Un exemple d'affichage :

connString:

```
btsp://0019EF01194C:1;authenticate=false;encrypt=false;master=false
```

0019EF01194C l'adresse physique

:1 le port de communication

# Un service comme un autre

---

- **Nous avons**

- Un service -> un UUID
- un UUID -> une URL
- Une URL -> une communication
- Une communication
  - Un flux d'octets, `btsp://`

# Exemple de service/UUID

---

C'est un serveur ... protocole btspp://

```
StreamConnectionNotifier notifier =  
    (StreamConnectionNotifier)Connector.open(  
        "btspp://localhost:102030405060708090A0B0C0D0E0F010");
```

attente d'une requête

```
StreamConnection conn = notifier.acceptAndOpen();
```

# Un serveur au complet

```
public class SimpleBTSPPServer {

    public static void main(String[] args) throws Exception{
        LocalDevice local = LocalDevice.getLocalDevice();
        local.setDiscoverable(DiscoveryAgent.GIAC);

        StreamConnectionNotifier notifier =

            (StreamConnectionNotifier)Connector.open(
                "btspp://localhost:" + "102030405060708090A0B0C0D0E0F010");

        StreamConnection conn = notifier.acceptAndOpen();

        InputStream in = conn.openInputStream();
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        int data;
        while ((data = in.read()) != -1) {
            out.write(data);
        }
        System.out.println(" message recu : " + out.toString());
        in.close();
        conn.close();
        notifier.close();
    }
}
```

UUID

note : ci-dessus vous avez un serveur d'une seule connexion... c'est peu...

# Une démonstration ...

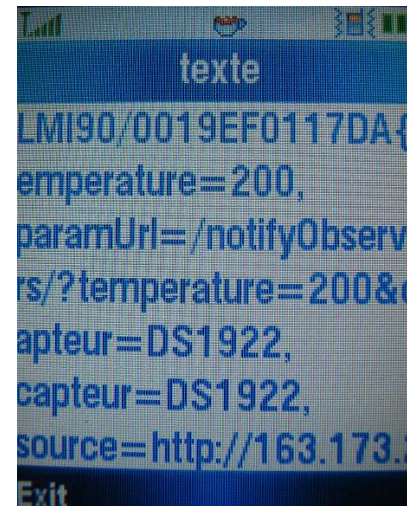
- **Démonstration en ligne :**

- <http://jfod.cnam.fr/SEJA/jnlp/TestsObservable.html>

- **Traces côté Observable**

- BlueCove version 2.1.0 on winsock
  - Request: GET /notifyObservers/?temperature=200&capteur=DS1922 HTTP/1.1
  - Request: GET /addObserver/?uuid=102030405060708090A0B0C0D0E0F088 HTTP/1.1
  - addObserver : 102030405060708090A0B0C0D0E0F088
  - Request: GET /notifyObservers/?temperature=300&capteur=DS1921 HTTP/1.1
  - Request: GET /notifyObservers/?temperature=200&capteur=DS1922 HTTP/1.1
  -

- **Côté Mobile, bluetooth**



# Conclusion intermédiaire

---

- **MVC, web**
  - Déjà vu
  - JMS en standard ...
- **Classes essentielles**
  - **ServerSocket**
    - Méthode **accept** bloquante ou avec un délai de garde
  - **Socket**
    - Un thread à chaque connexion,
    - Usage d'un d'un pool de Thread
- **Un Serveur en quelques lignes de Java**
- **Bien mais**



# Bien mais

- Dégradation des performances si le nombre de clients croît
- Couplage fort : connexion/traitement
  - Un thread est engendré à chaque requête pour le traitement
    - Séquence typique : read -> decode -> traitement -> encode -> write

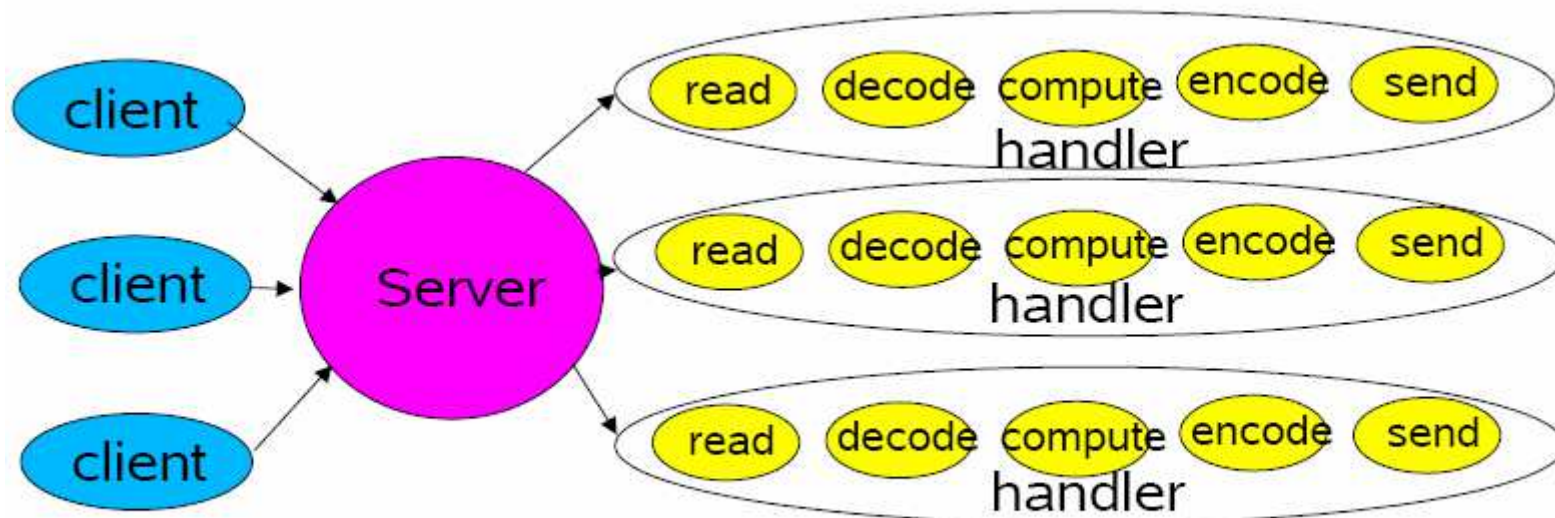
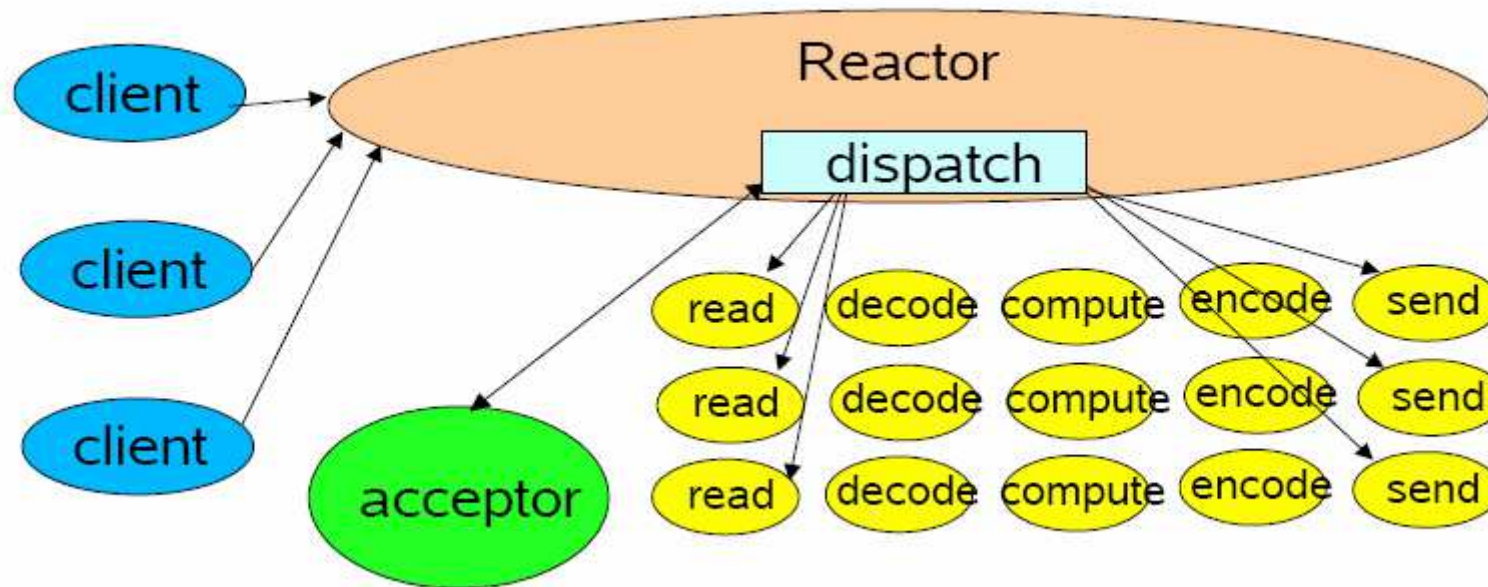


Schéma extrait de <http://gee.cs.oswego.edu/dl/cpjslides/nio.pdf>

- Une solution :
  - Découpler la connexion du service à effectuer
  - Permettre un parallélisme « plus fin »

# Une proposition : Patrons Reactor et Acceptor



- Couplage faible, connexion/traitement
  - Grain de parallélisme plus fin
  - Chaque tâche est un processus non bloquant
- extrait de <http://gee.cs.oswego.edu/dl/cpjslides/nio.pdf>

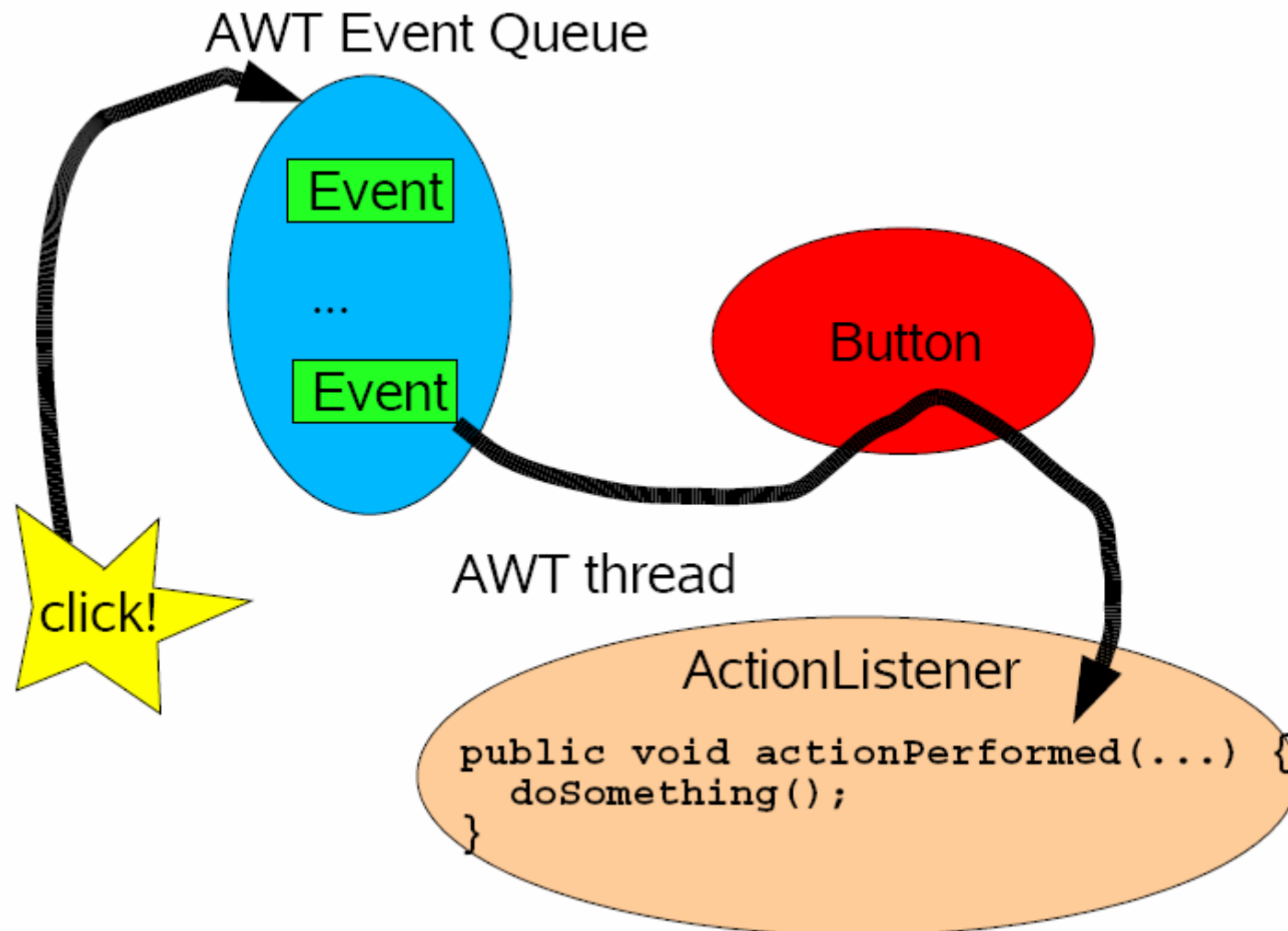
# Sommaire

---

- **Patron Reactor :**
  - déjà vu et utilisé, peut-être mais où ?
  
- **java.nio.\* depuis le jdk1.4**
  - Buffer
  - Channel
  - SocketChannel
  - ServerSocketChannel
  
- **Patrons Reactor et Acceptor en détail**
  - **Extraits de** <http://gee.cs.oswego.edu/dl/cpjslides/nio.pdf>

# Couplage faible : Connexion/traitement

- Déjà vu et utilisé : la gestion des évènements de l'AWT



# Patron Reactor & AWT event

---

- **Le patron Reactor réagit aux événements en déclenchant le gestionnaire approprié**
  - AWT Thread sur le transparent précédent
- **Les Gestionnaires s'exécutent et sont non bloquants**
  - AWT listener
  - Le source de la méthode actionPerformed
- **Ce patron associe les gestionnaires aux événements**
  - Analogue à addActionListener

# java.nio orienté réseau...

---

- **Channel**

- **Connection aux fichiers, sockets,... en lecture non-bloquante**
- **java.nio.channels.SocketChannel;**

- **Buffer**

- **Zone d'échange lue, comme écrite, par les canaux (Channel)**
- **Ex. java.nio.ByteBuffer;**

# Java.nio.Selector SelectorKey

---

- **Selector**
  - Quels sont les canaux agréés pour ce type d'évènements
  - `java.nio.channels.Selector`;
  
- **SelectionKey**
  - L'agrément lié au « selector » du canal
  - Ex. `java.nio.channels.SelectionKey`;

# Saut possible à la conclusion

---

- **Exemple à détailler à domicile ...**
  - Sources et biblio
    - [http://jod.cnam.fr/NSY102/cours\\_reactor\\_pattern.jar](http://jod.cnam.fr/NSY102/cours_reactor_pattern.jar)



## Un exemple

---

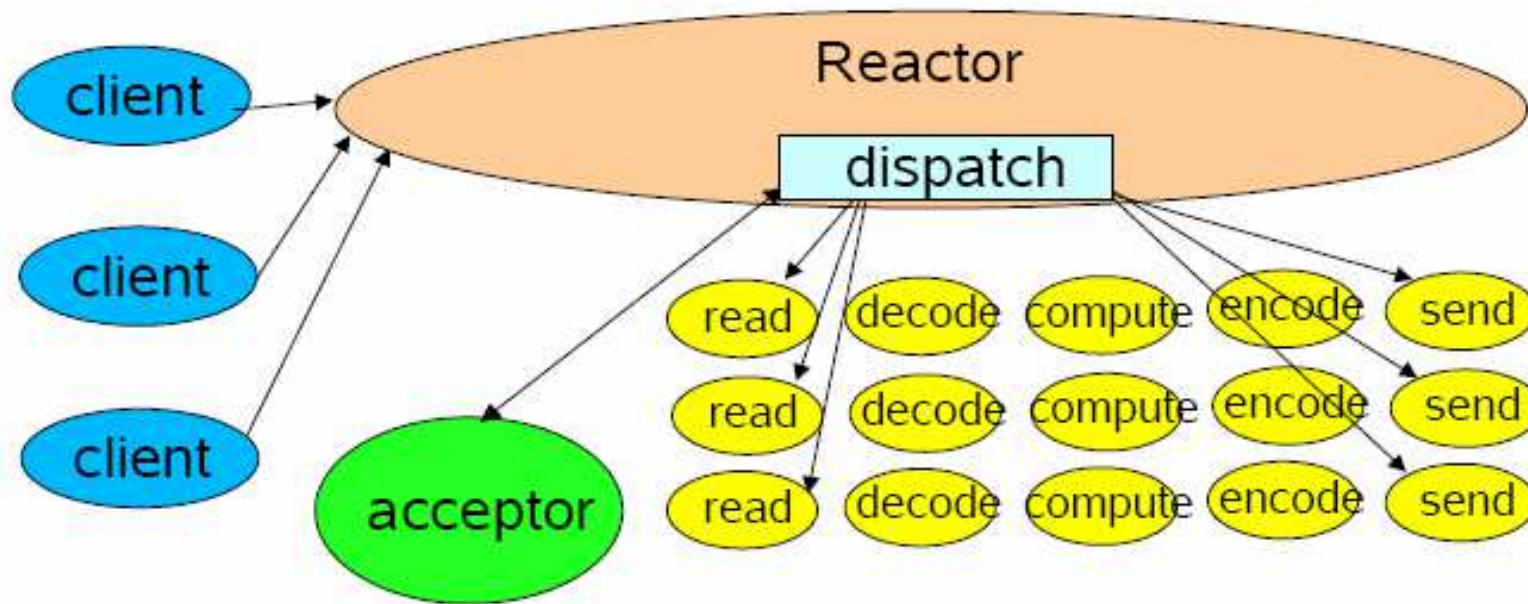
```
SocketChannel channel = serverChannel.accept();  
channel.configureBlocking(false);
```

```
SelectionKey sk;  
sk = channel.register(selector, SelectionKey.OP_READ);
```

- **À l'occurrence des évènements, dispatch ...**

```
int eventCount = selector.select();  
Iterator<SelectionKey> it = selector.selectedKeys().iterator();  
while (it.hasNext()) {  
    SelectionKey key = it.next(); it.remove();  
  
    if (key.isValid() && key.isReadable()) {  
        eventHandler.onReadableEvent(key.channel()); }  
... } ... }
```

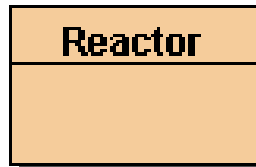
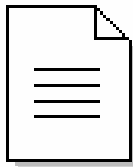
# Patron Reactor, bis



- **Les Clients sont sur le web**
- **La classe/Patron Reactor accepte des requêtes et les « dispatch » ...**
- **Acceptor à suivre...**

# Patron Reactor - et les Gestionnaires

---



- **L'exemple de doug lea**
  - Deux classes : Reactor & Handler
  - Reactor contient une classe interne : Acceptor

# Patron Reactor & java.nio.channels.ServerSocketChannel

---

```
public class Reactor implements Runnable{
    final Selector selector;
    final ServerSocketChannel serverSocket;

    public Reactor(int port) throws IOException {
        selector = Selector.open();
        serverSocket = ServerSocketChannel.open();
        serverSocket.socket().bind(new InetSocketAddress(port));
        serverSocket.configureBlocking(false);
        SelectionKey sk;
        sk = serverSocket.register(selector, SelectionKey.OP_ACCEPT);

        sk.attach(new Acceptor()); // déclenché à chaque requête

        new Thread(this).start(); // transparent suivant
    }
}
```

- SelectionKey.OP\_ACCEPT

*If the selector detects that the corresponding server-socket channel is ready to accept another connection, or has an error pending, then it will add OP\_ACCEPT to the key's ready set and add the key to its selected-key set.*

## Reactor 2, select & dispatch

---

```
public void run() {
    try {
        while (!Thread.interrupted()) {
            selector.select();
            Set<SelectionKey> selected = selector.selectedKeys();

            Iterator<SelectionKey> it = selected.iterator();
            while (it.hasNext()) // les élus
                dispatch(it.next());

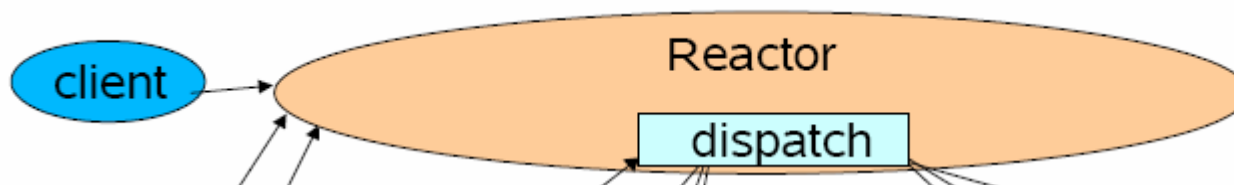
            selected.clear();
        }
    } catch (IOException ex) { /* ... */ }
}
```

## Reactor 3 : dispatch

---

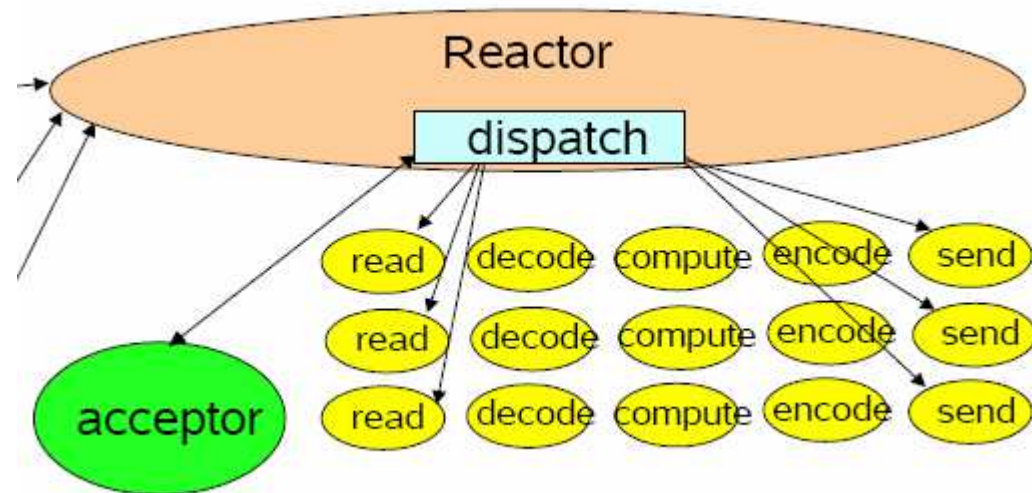
Quels sont les sélectionnés ?

```
void dispatch(SelectionKey k) {  
    Runnable r = (Runnable)(k.attachment());  
    if (r != null)  
        r.run();  
}
```



# Reactor-Acceptor, à chaque requête

```
class Acceptor implements Runnable { // classe interne, et membre
    public void run() {
        try {
            SocketChannel c = serverSocket.accept();
            if (c != null)
                new Handler(selector, c);
        } catch (IOException ex) { /* ... */ }
    }
}
```



# Gestionnaire comme Handler

---

```
final class Handler implements Runnable {
    final SocketChannel socket;
    final SelectionKey sk;
    ByteBuffer input = ByteBuffer.allocate(MAXIN);
    ByteBuffer output = ByteBuffer.allocate(MAXOUT);
    static final int READING = 0, SENDING = 1;
    int state = READING;           // état courant

    public Handler(Selector sel, SocketChannel s) throws IOException {
        s.configureBlocking(false);
        sk = s.register(sel, 0);           // initialisation
        sk.attach(this);
        sk.interestOps(SelectionKey.OP_READ);
        sel.wakeup();
    }
    boolean inputIsComplete() { /* . . . */ }
    boolean outputIsComplete() { /* . . . */ }
    void process() /* . . . */ }
```



# Handler, méthode run

---

```
// deux états possibles READING ou SENDING
```

```
public void run() {  
    try {  
        if (state == READING) read();  
        else if (state == SENDING) send();  
    } catch (IOException ex) { /* ... */ }  
}
```

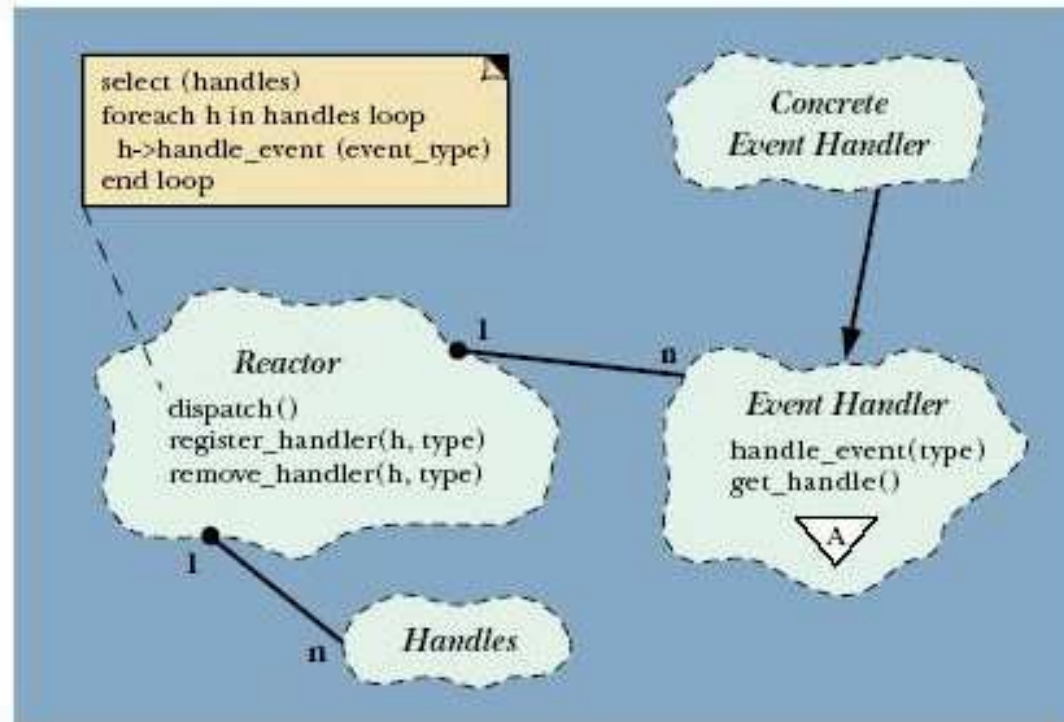
# Méthode read et send

---

```
void read() throws IOException {
    int len = socket.read(input);
    if (inputIsComplete()) {
        process();
        state = SENDING;
        sk.interestOps(SelectionKey.OP_WRITE);
    }
}
```

```
void send() throws IOException {
    socket.write(output);
    if (outputIsComplete()) sk.cancel();
    socket.close();
}
```

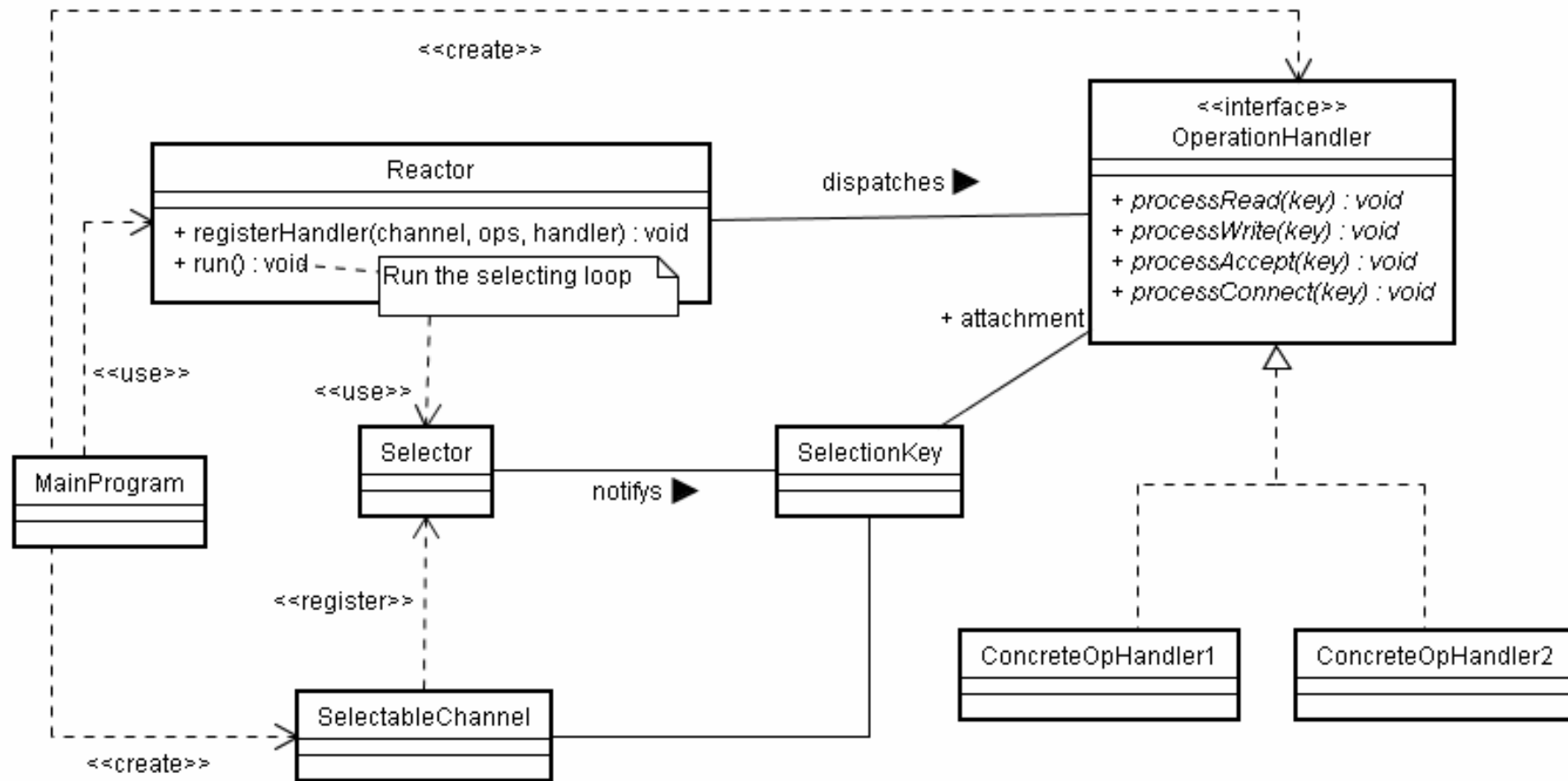
# Diagramme selon D. Schmidt



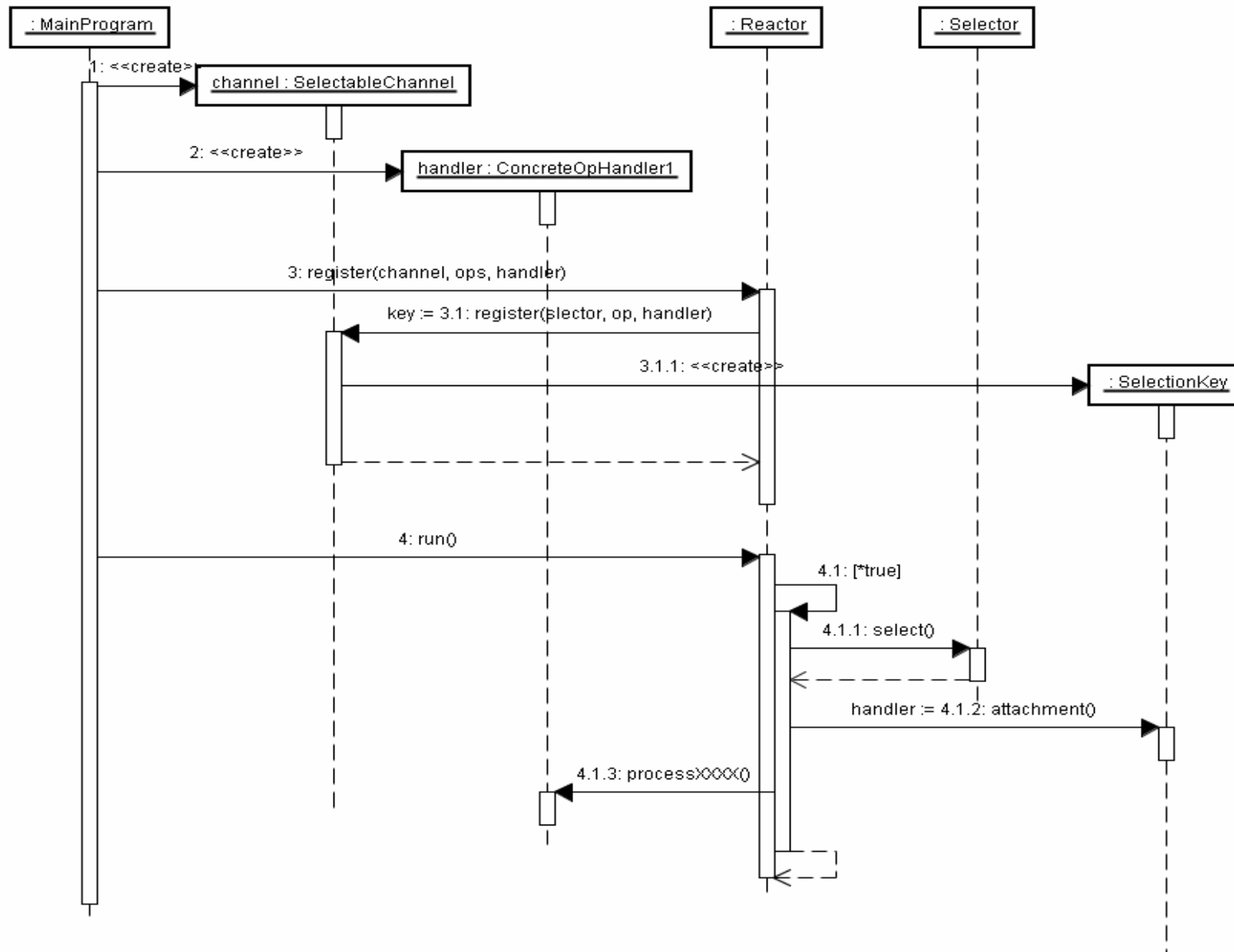
- À lire

- [http://www.cs.bgu.ac.il/~spl051/Personal\\_material/Practical\\_sessions/Ps\\_12/ps12.html](http://www.cs.bgu.ac.il/~spl051/Personal_material/Practical_sessions/Ps_12/ps12.html)

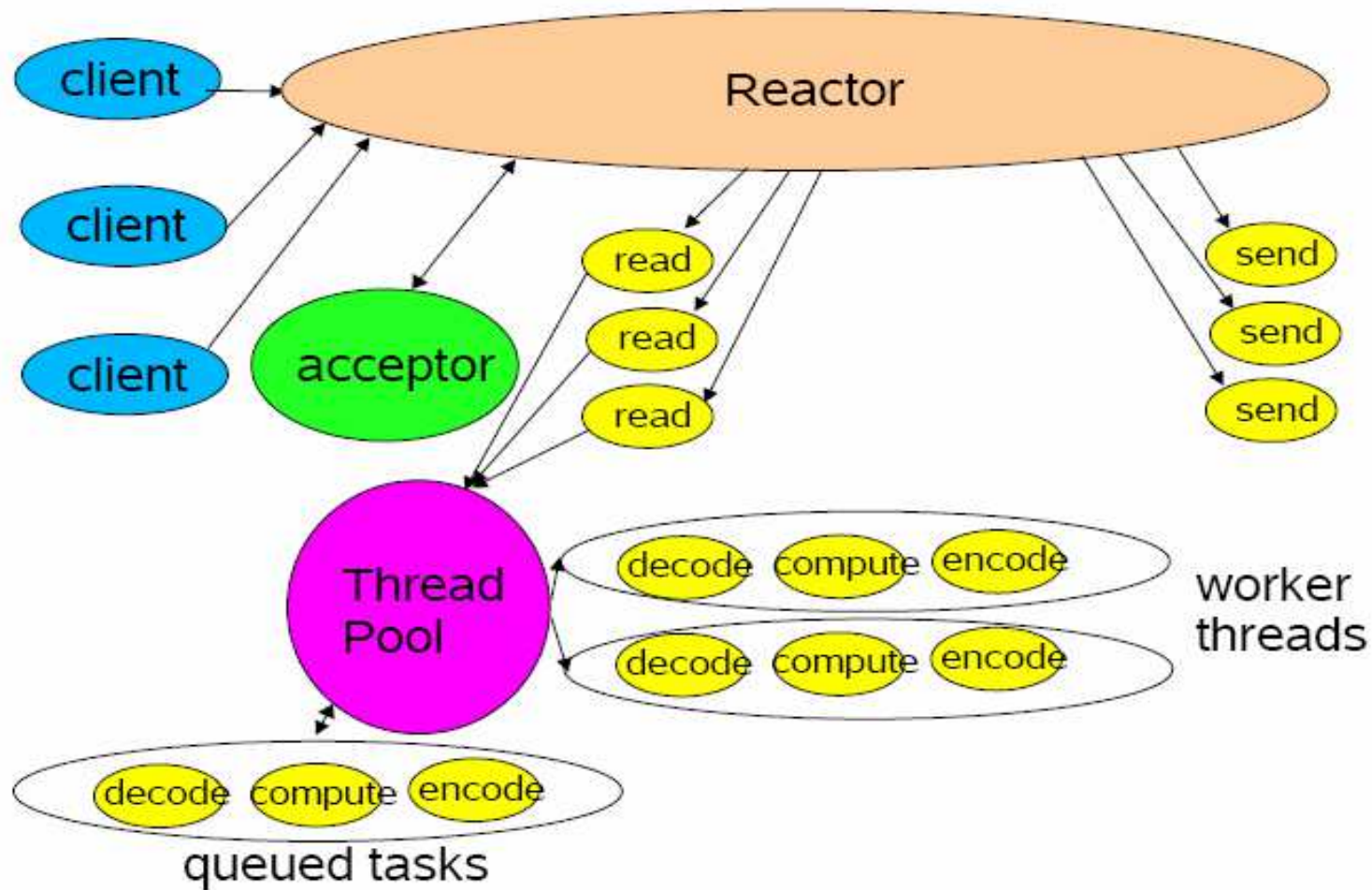
# Diagramme UML



# Diagramme de séquence



# Pool de Thread



- **Source de parallélisme**

# Conclusion Sur Reactor/Acceptor

---

- **Indispensable pour de meilleures performances**
- **Lire**
  - Douglas Schmidt
  - <http://jerry.cs.uiuc.edu/~plop/plop99/proceedings/Fernandez3/RACPattern.PDF>

# Le patron HeartBeat

---

- **Le serveur est-il en état de fonctionner ?**
- **Requête régulière et décision en conséquence**

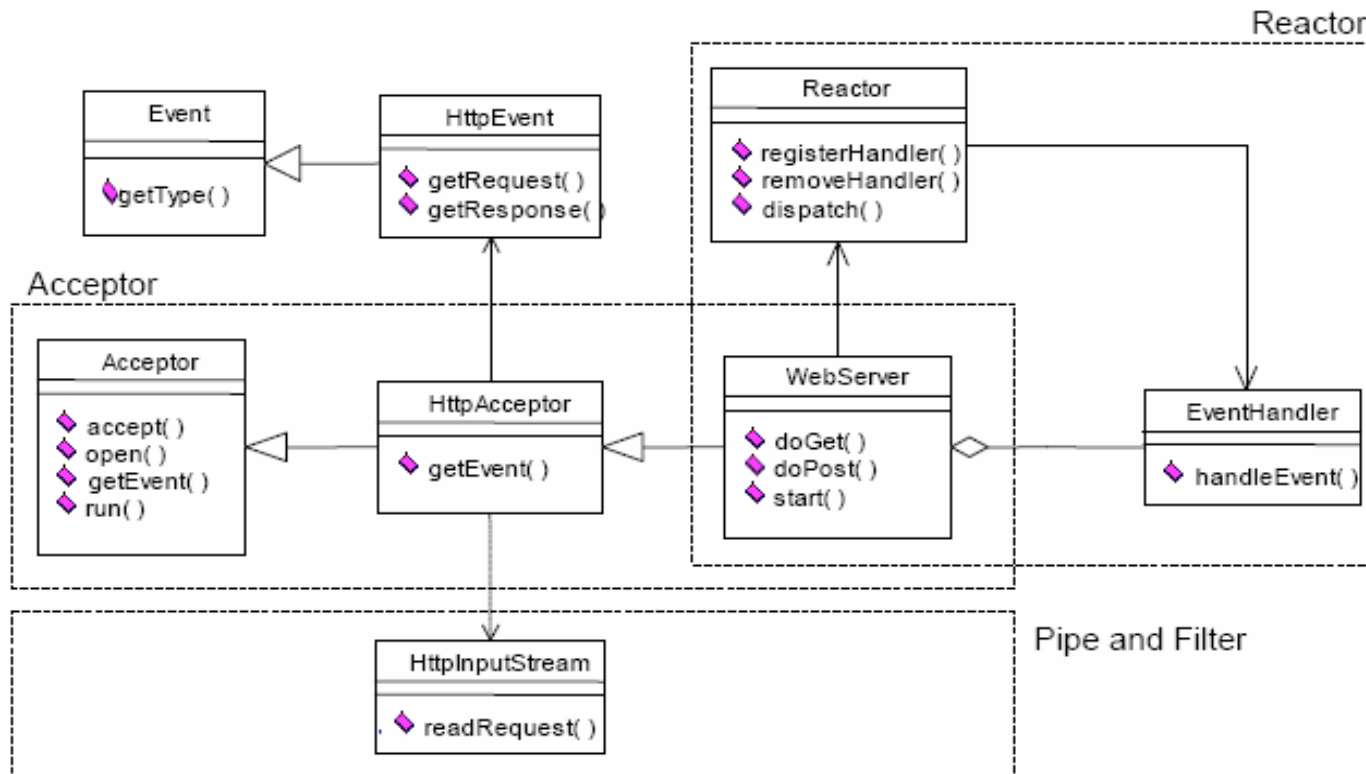
**ou**

- **Si la réponse dépasse une certaine durée, le serveur est considéré comme en panne**



# Résumé d'un Serveur Web possible

## Web Server Design



Copyright 2005, Michael Weiss • COMP 4104 • Fall 2005

- <http://www.scs.carleton.ca/~weiss/courses/4104/lectures/06-Review-1.pdf>

# Conclusion

---

- **Orienté Réseau**
  - Simple
  
- **Patrons Reactor et Acceptor**
  - Depuis toujours ?
  
  - Depuis le j2se 1.4

# Annexes

---

- **Junit & http = httpjunit**
- **Java Web Start**
- **Applette et MBean**
- **JAWS**

# URL en java : java.net.URL

---

- `URL gamelan = new URL("http://www.gamelan.com/");`
- `URL gamelan = new URL("http://www.gamelan.com/pages/");`
- `URL gamelanGames = new URL(gamelan, "Gamelan.game.html");`
- `URL gamelanNetwork = new URL(gamelan, "Gamelan.net.html");`
- `URL gamelanNetworkBottom = new URL(gamelanNetwork, "#BOTTOM");`

```
public class ParseURL { public static void main(String[] args) throws Exception {
    URL aURL = new URL("http://java.sun.com:80/docs/books/tutorial" +
                      "/index.html?name=networking#DOWNLOADING");

    System.out.println(aURL.getProtocol());      // http
    System.out.println(aURL.getAuthority());     // java.sun.com:80
    System.out.println(aURL.getHost());         // java.sun.com
    System.out.println(aURL.getPort());         // 80
    System.out.println(aURL.getPath());         // /docs/books/tutorial/index.html
    System.out.println(aURL.getQuery());        // name=networking
    System.out.println(aURL.getFile());         // /docs/books/tutorial/index.html?name=networking
    System.out.println(aURL.getRef());          // DOWNLOADING
}
```

- <http://java.sun.com/docs/books/tutorial/networking/urls/creatingUrls.html>

## En résumé

---

- **Classe d'accès aux informations**
  - indépendante du protocole choisi
  
- **Lecture écriture en 7 étapes**
  1. Après avoir créé l'URL.
  2. Obtenir l'instance URLConnection.
  3. Installer les capacités en sortie de cette instance de URLConnection.
  4. Ouvrir le flot en entrée.
  5. Obtenir le flot en sortie.
  6. Écrire sur ce flot.
  7. Fermer celui-ci.

## Divers : derrière un proxy

---

```
java -cp . -Dhttp.proxyHost=cache.cnam.fr  
        -Dhttp.proxyPort=3128 ClientHTTP
```

ou

```
public static  
    void setHttpProxy(String proxyHost,int proxyPort){  
        Properties prop = System.getProperties();  
        prop.put("proxySet","true");  
        prop.put("http.proxyHost",proxyHost);  
        prop.put("http.proxyPort",Integer.toString(proxyPort));  
    }
```

## Annexe : analyse du contenu (*simplifié*)

---

A chaque requête

```
String request = in.readLine();
StringTokenizer st = new StringTokenizer(request);
String token = st.nextToken();
if(token.equals("GET")){
    String paramUrl = st.nextToken();
    File file = new File(".") + paramUrl);
    send(out, file);
} else {
    ...}
```

## Annexe : envoi du document (*simplifié*)

---

```
private void send(DataOutputStream os, File file) throws Exception{
    try{
        BufferedInputStream in = new BufferedInputStream(
            new FileInputStream(file));
        os.write("HTTP/1.0 200 OK\r\n".getBytes());
        os.write(new String("Content-Length" + new Long(file.length()) +
            "\r\n").getBytes());

        os.write(new String("Content-Type: " + guessType(file.getPath())
            + "\r\n\r\n").getBytes());

        byte[] buf = new byte[1024];
        int len;
        while((len = in.read(buf,0,1024)) != -1){
            os.write(buf,0,len);
        }
        in.close();
    }catch(FileNotFoundException fnfe){
        ...
    }
}
```



# Annexe : http Test unitaires

- **HttpUnit** <http://httpunit.sourceforge.net/>
- **Exemple ici intégré à BlueJ\*** :

```
public void testApplette() throws Exception {  
    WebConversation conversation = new WebConversation();  
    WebRequest request = new GetMethodWebRequest(  
        "http://localhost:8100/test/?test=truc" );  
  
    WebResponse response = conversation.getResponse( request );  
    assertEquals( " réponse ???", "<b>ok</b>", response.getText() );  
}
```



- \* **httpunit.jar** et **Tidy.jar** sont dans <BlueJ\_HOME>/lib/userlib/
- **Voir**
- **Voir aussi jwebunit** <http://jwebunit.sourceforge.net>

# Annexe : Java Web Start

---

- Téléchargement depuis le Web d'applications Java certifiées
- Assurance de toujours disposer de la dernière version
- Principe de mise en œuvre
  - Côté serveur
    - Signer une archive Java (.jar)
    - Proposer le fichier JNLP (Java Network Launch Protocol)
    - Installer ce fichier et cette archive sur un serveur
  - Côté client
    - Depuis un navigateur télécharger le fichier JNLP
    - Ou depuis une console exécuter l'outil >javaws

# Exemple le serveur au protocole « maison »

```
import java.net.Socket;
import java.net.ServerSocket;
import java.io.*;
public class Serveur{
    public static void main(String[] args) throws Exception{
        ServerSocket serveur = new ServerSocket(5000);
        while(true) {
            Socket socket = serveur.accept();

            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
            String cmd = in.readLine();
            // parle !!!
            DataOutputStream out = new DataOutputStream( socket.getOutputStream());
            if(cmd.equals("parle")){
                out.write("bonjour\n".getBytes());
            }else{
                out.write("commande inconnue ?\n".getBytes());
            }
            socket.close();
        }
    }
}
```

## 1. Génération de l'archive serveur.java

- Le fichier MANIFEST.MF contient ces 3 lignes  
Manifest-Version: 1.0  
Class-Path:  
Main-Class: Serveur

# Exemple suite

---

- Créer une signature
  - `keytool -genkey -alias jmd -keypass nsy102`

- Signer cette archive
  - `jarsigner server.jar jmd`

- Proposer le fichier JNLP, [ici serveur.jnlp](#)

```
<jnlp spec="1.0+«  
  codebase="http://jfod.cnam.fr/NSY102/serveurs/"  
  href="serveur.jnlp">  
  <information>  
    <title>Serveur Maison NSY102</title>  
    <vendor>Cnam NSY102</vendor>  
    <description>Serveur maison</description>  
    <description kind="short">dis bonjour</description>  
    <offline-allowed/>  
  </information>  
  <security>  
    <all-permissions/>  
  </security>  
  <resources>  
    <j2se version="1.5+"/>  
    <jar href="serveur.jar"/>  
  </resources>  
  <application-desc main-class="Serveur"/>  
</jnlp>
```

# Exemple fin

---

Il suffit de cliquer ici

<http://jfod.cnam.fr/NSY102/serveurs/serveur.jnlp>

Ou bien depuis une console

javaws <http://jfod.cnam.fr/NSY102/serveurs/serveur.jnlp>

*Voir le mode d'emploi*

[http://jfod.cnam.fr/tp\\_cdi/jnlp/](http://jfod.cnam.fr/tp_cdi/jnlp/)

*Et aussi*

<http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/keytool.html>

<http://ragingcat.developpez.com/java/outils/keytool/ui/>

<http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/jarsigner.html>

# JConsole et Applet

---

Extrait de <http://blogs.sun.com/lmalventosa/>

## Développer

1. une interface AppletteXXXXXMBean
2. une classe AppletteXXXXX extends JApplet implements AppletteXXXXXMBean
3. Puis générer l'archive AppletteXXXXX.jar

## Générer un certificat

```
keytool -genkey -alias nsy -keypass PWD -keystore nsystore -storepass PWD
```

## Signer l'archive

```
jarsigner -keystore nsystore AppletteXXXXX.jar nsy
```

Exemple : **ici** <http://jfod.cnam.fr/NSY102/AppletteServeurWeb.html>

C'est une page html standard

```
<applet code="AppletteServeurWeb.class"  
        width=500 height=200  
        archive="AppletteServeurWeb.jar"  
>
```

# JConsole et Applet démo

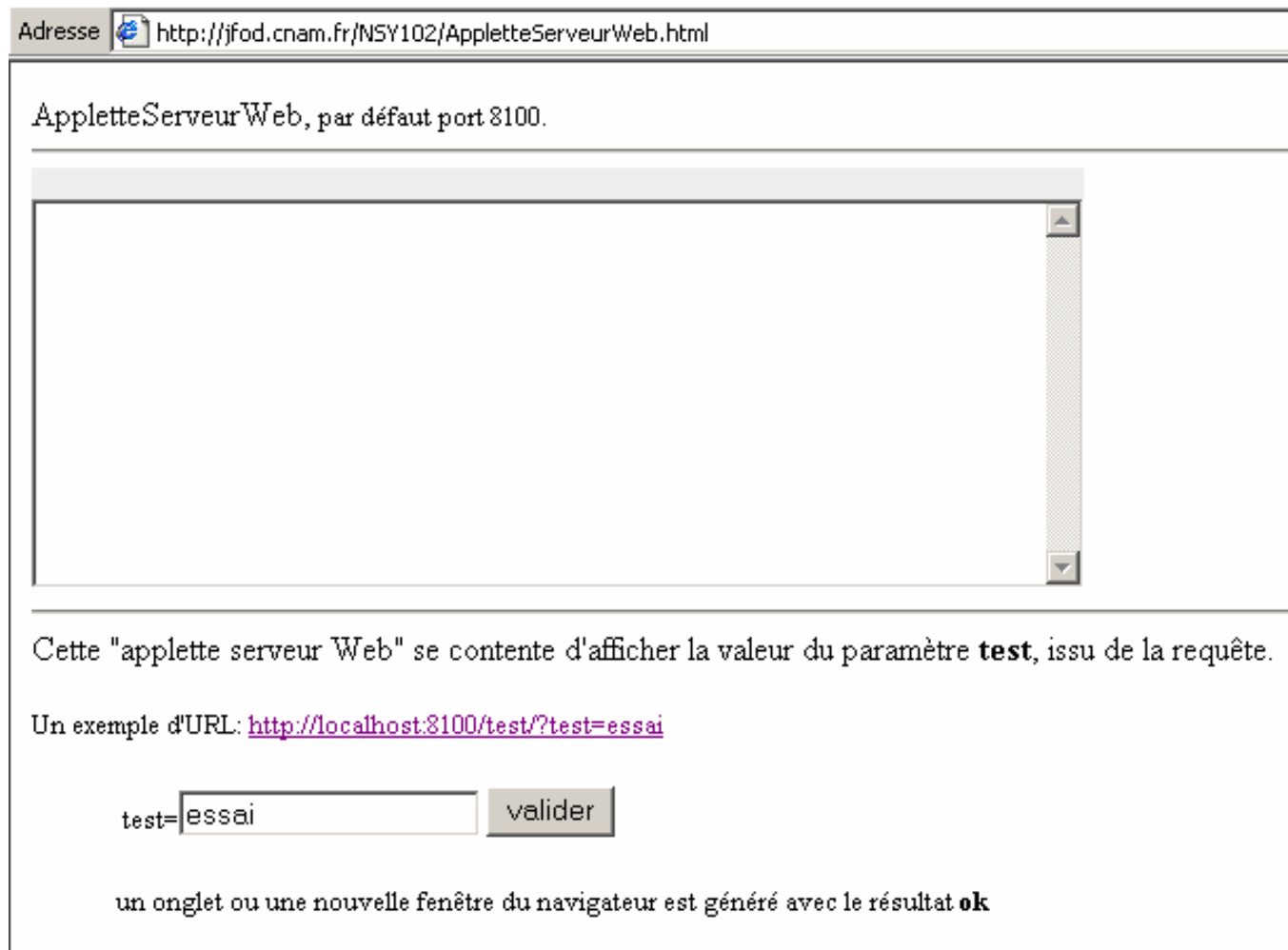
The screenshot shows a Mozilla Firefox browser window titled "AppletteServeurWeb Applet - Mozilla Firefox" with the address bar containing "http://jfod.cnam.fr/NSY102/AppletteServeurWeb.html". Overlaid on the browser is the Java Monitoring & Management Console (JConsole) window, titled "Java Monitoring & Management Console" and "pid: 2336". The JConsole window displays a tree view of MBeans under "AppletteMBean", with "AppletteServeurWeb" expanded to show "serveur\_port\_8100", which has an attribute "ServeurActif" selected. The right pane shows the "Attribute value" table with "ServeurActif" set to "true", and the "MBeanAttributeInfo" table with details for the attribute.

Name	Value
ServeurActif	true


  

Name	Value
Attribute:	
Name	ServeurActif
Description	Attribute exposed for management
Readable	true
Writable	false
Is	false
Type	boolean

# Jconsole et Applet : un exemple en images



The screenshot shows a web browser window with the address bar containing `http://fod.cnam.fr/NSY102/AppletteServeurWeb.html`. The page content includes the text "AppletteServeurWeb, par défaut port 8100." followed by a large, empty rectangular area with a vertical scrollbar on the right side. Below this area, there is a text description: "Cette 'applette serveur Web' se contente d'afficher la valeur du paramètre **test**, issu de la requête." and an example URL: "Un exemple d'URL: <http://localhost:8100/test/?test=essai>". At the bottom of the page, there is a form with a text input field containing "essai" and a "valider" button.

Adresse  `http://fod.cnam.fr/NSY102/AppletteServeurWeb.html`

AppletteServeurWeb, par défaut port 8100.

Cette "applette serveur Web" se contente d'afficher la valeur du paramètre **test**, issu de la requête.

Un exemple d'URL: <http://localhost:8100/test/?test=essai>

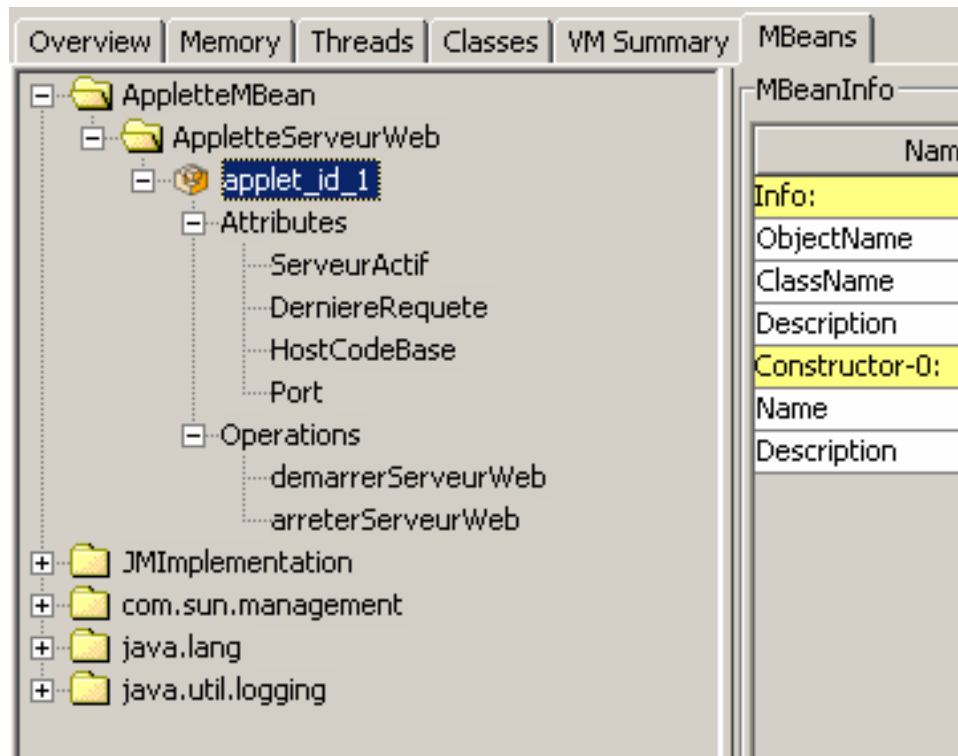
test=

un onglet ou une nouvelle fenêtre du navigateur est généré avec le résultat **ok**

- Cliquez sur valider engendre une nouvelle fenêtre avec le résultat de la requête effectuée en **`http://localhost:8100/test/?test=essai`**



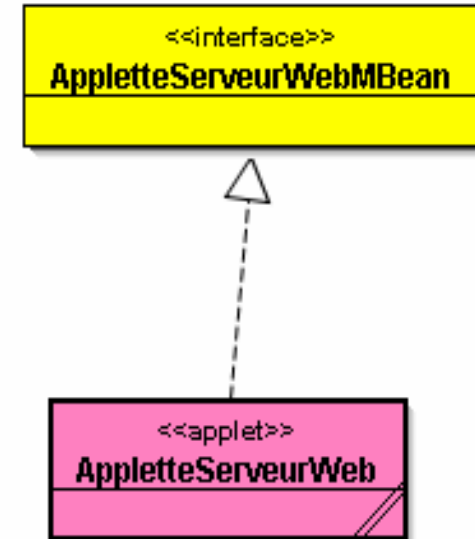
# Jconsole et applet suite



- **Un scenario :**
  - 1) **arreterServeurWeb**
  - 2) **demarrerServeurWeb port 8200**
  - 3) **ouvrir un nouvel onglet sur le navigateur puis recharger la page initiale**
    - **2 serveurs en cours d'exécution 8100 et 8200**
  - 4) **applet\_i\_2 est apparu dans l'onglet MBeans de jconsole**
  - ...

# L'interface « Mbean » de l'applette

```
public interface AppletteServeurWebMBean {  
  
    // opérations  
    public void demarrerServeurWeb(int port);  
    public void arreterServeurWeb();  
  
    // lectures, getter  
    public int getPort();  
    public boolean getServeurActif();  
    public String getHostCodeBase();  
    public String getDerniereRequete();  
}
```

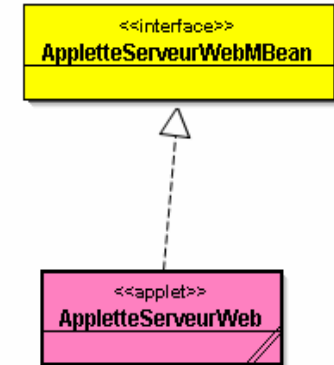


# L'applette, un extrait du code source

```
private static AtomicInteger applet_id = new AtomicInteger(1);
private MBeanServer mbs;
private ObjectName name;

public void init(){
    // initialisation de l'applette
    // et de son MBean ...
    mbs = ManagementFactory.getPlatformMBeanServer();
    name = new ObjectName("AppletteMBean: type=AppletteServeurWeb, "+
        "name=applet_id_" + applet_id.getAndIncrement())
    mbs.registerMBean(this, name);
}

public synchronized void stop(){
    // terminaison de l'applette
    // et de son MBean ...
    mbs.unregisterMBean(name);
}
```



source complet ici

<http://jfod.cnam.fr/NSY102/appletteServeurWebMBean/>