# Java : notions fondamentales pour le multimédia

# Programmer le traitement de la parole

Jean-Marc Farinone
Xavier Castellani

# Building Multimedia Programs in Java

**Jean-Marc Farinone**
**farinone@cnam.fr**

**Assistant Professor**
**Conservatoire National des Arts et Métiers**
**CNAM Paris (France)**

**Tunis AICSSA Congress 17 July 2003**

# Fundamental notions

- Definition
- 100% pure Java
- 2 design patterns : Bridge and Abstract Factory

# Multimedia = ?

- Answer of Sun Microsystems :
  - 2D | 3D | Advanced Imaging | Image I/O | JMF | Shared Data Toolkit | Sound | Speech
- source `http://java.sun.com/products/java-media/`
- Sometimes for members of Sun Developer Network (free)

# Reuse in Java

- With Java we can reuse other Java programs with JNI
- Hmm! Sure for C and C++ programs
- => Java is great for :
  - IBM, Oracle, HP, Informix, Ingres, Sybase, Apple, etc.
    (`http://servlet.java.sun.com/products/jdbc/drivers`)
  - … may be for Microsoft too ;-)

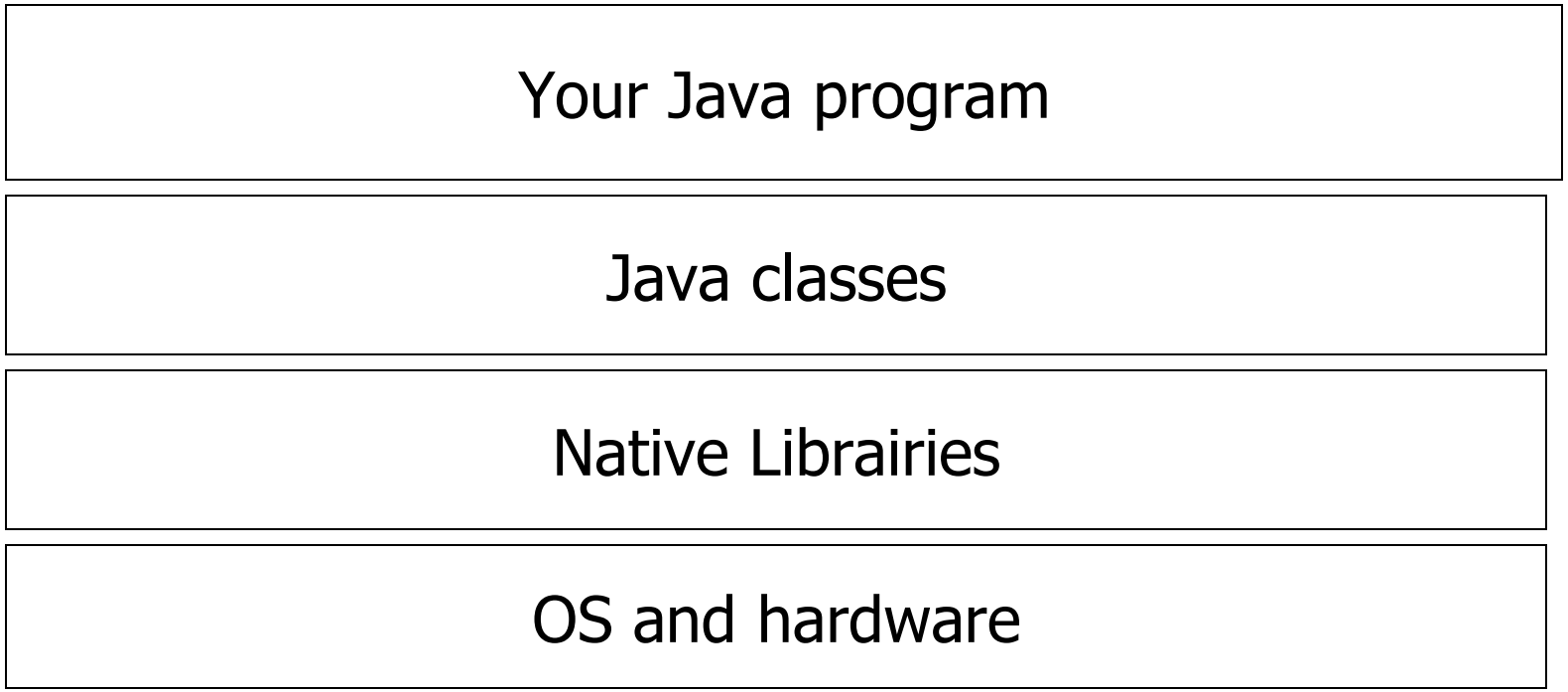
# From a Java Program to OS

Your Java program

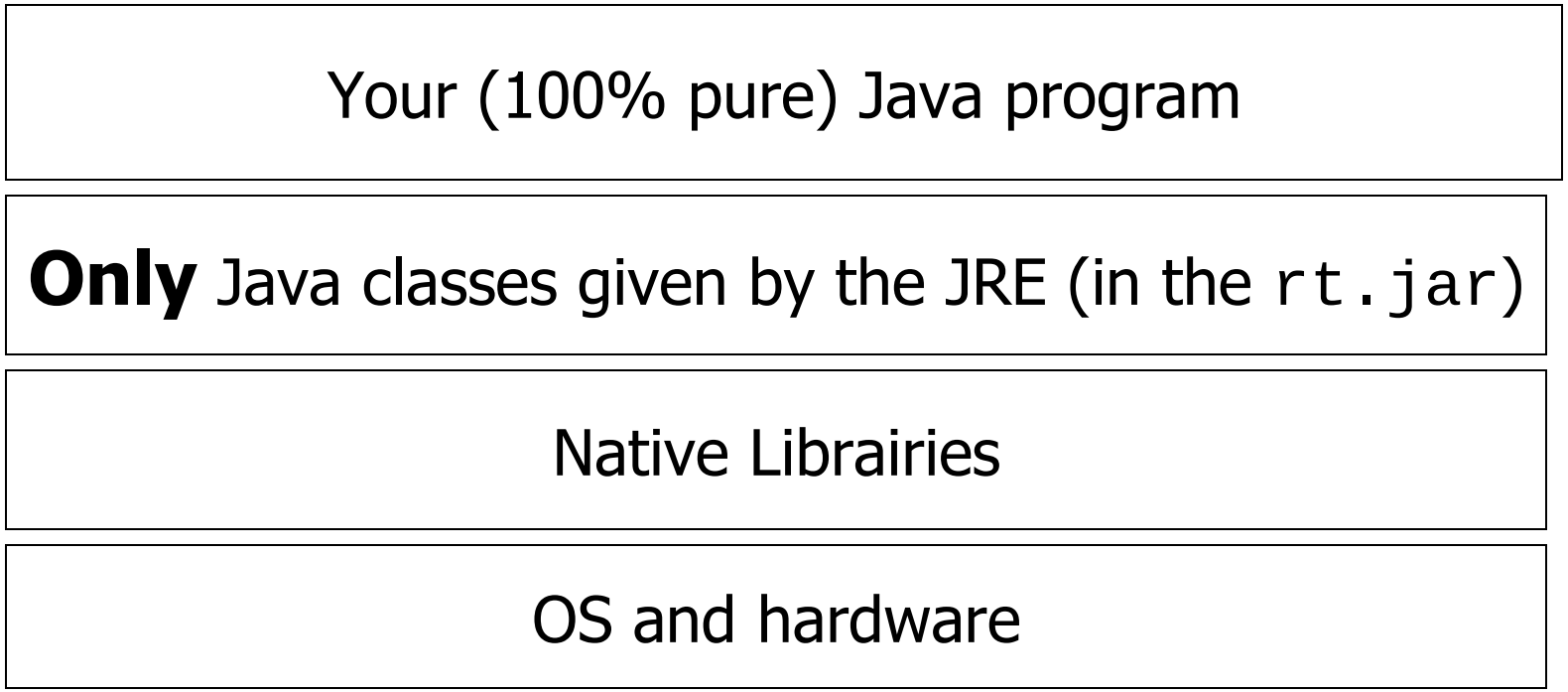
??

OS and hardware

# Every Java Program



# A 100% pure Java Program



# A non 100% pure Java program

- It's OK !!
- You reuse other codes
- Some limitations:
  - non dynamic loading of classes (which classes ?)
  - security restrictions
  - non portability

# Two design patterns

- Java was built with a lot of good ideas
- From … every domain in computer science
- From software engineering design: The Design Patterns (Gamma et al.)
  - Bridge
  - Abstract Factory

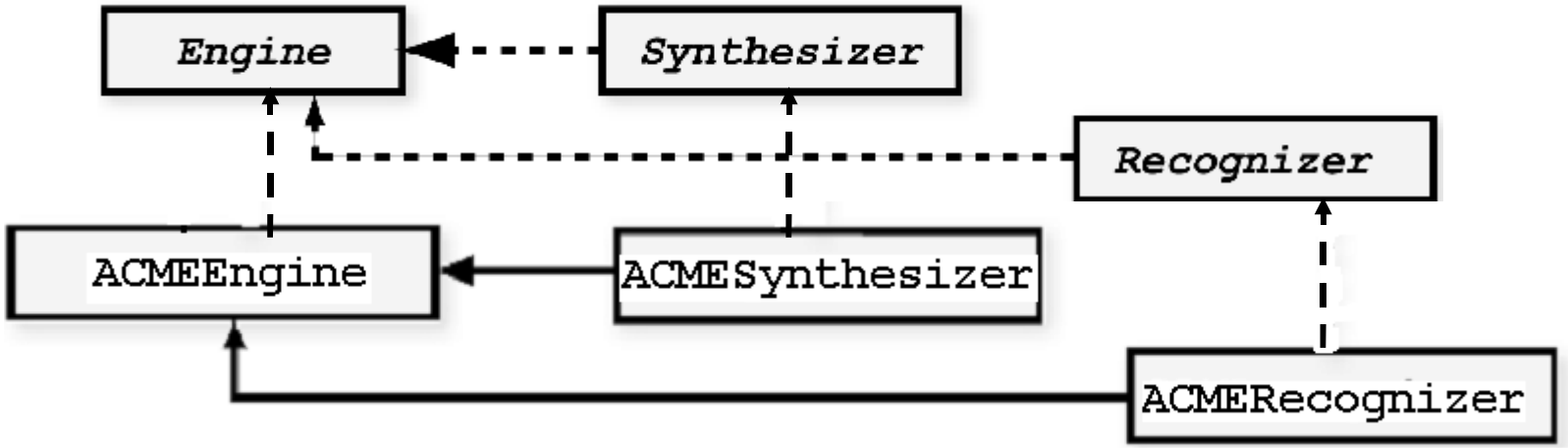
# Rappel Java

- Une référence d'interface peut repérer tout objet ...
- … non pas d'une interface (pourquoi ?)
- … d'une classe qui implémente cette interface.

```
public interface MonInterface { …}
public class MaClasse implements MonInterface { …}

…

    MonInterface ref = new MaClasse(); // ou bien un objet
                                       // récupéré
```

# The pattern Bridge

- An inheritance tree of interfaces
- Use of interface references in the program




- "Parallel" inheritance tree of implementing classes

# The pattern Abstract Factory

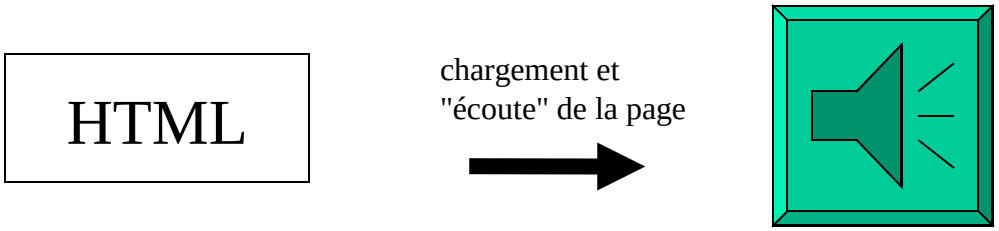
- Create the entrance to a specific implementation
- For example
  `Synthesizer s = Central.createSynthesizer(...);`
  where … is the way to find the appropriate implementation
- Bridge + Abstract Factory mostly used in Java (AWT, JDBC, JNDI, JMS, ... Multimedia)

# Plan de l'exposé

- Présentation
- Une application traitant la parole : étude UML
- La "pile" pour traiter la parole
- Synthèse de la parole : JSML
- Reconnaissance de la parole : JSGF
- Exemples d'applications
- Bibliographie

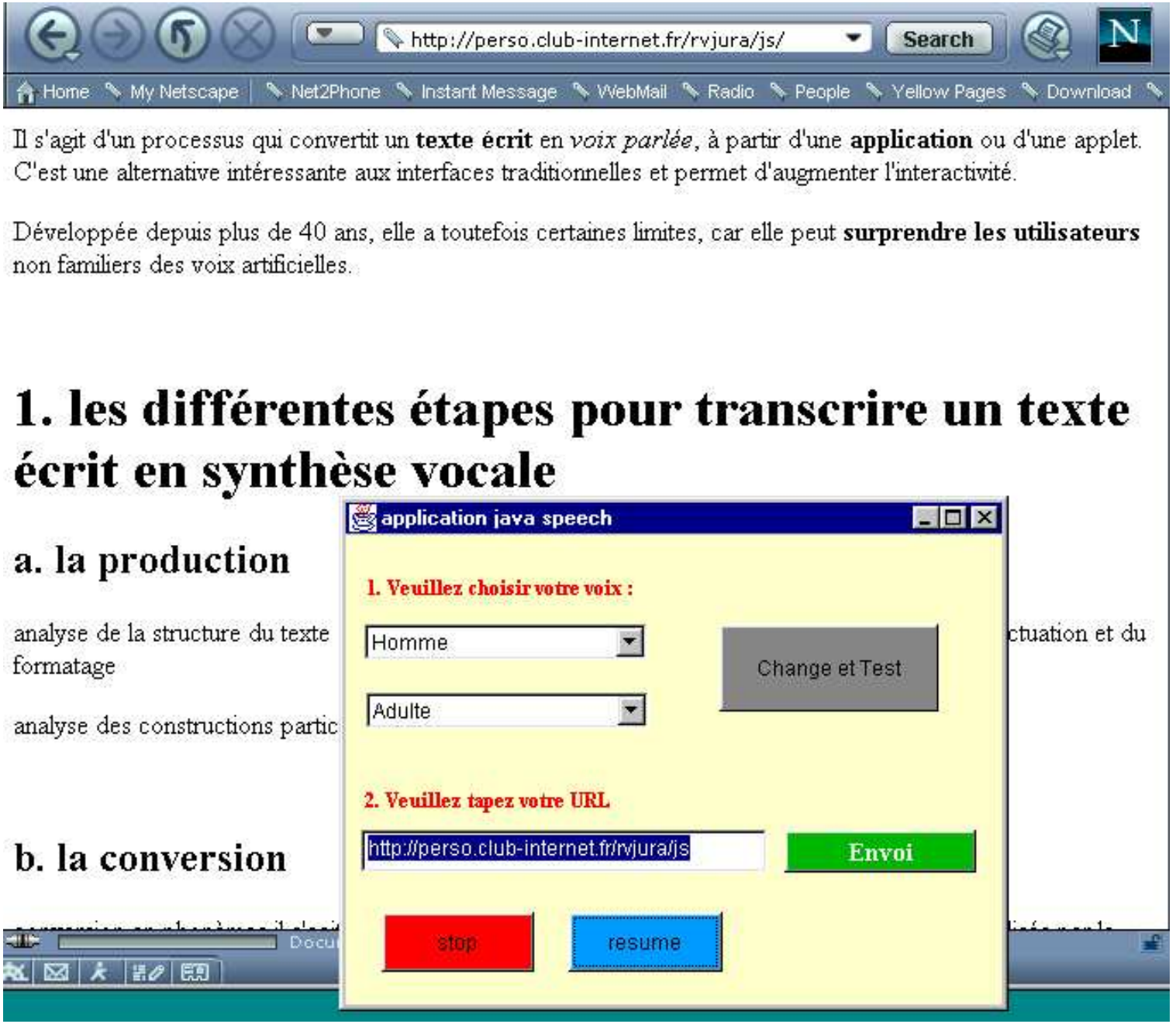
# Logiciel multimédia : un exemple

- "Ecouter les pages Web"
- Hervé Declipeur (valeur C CAM du CNAM) RVDesign, `declipeur@noos.fr`




Voir page html à lire (page suivante) puis démonstration : `1lanceDeclipeur.bat`

# Démonstration



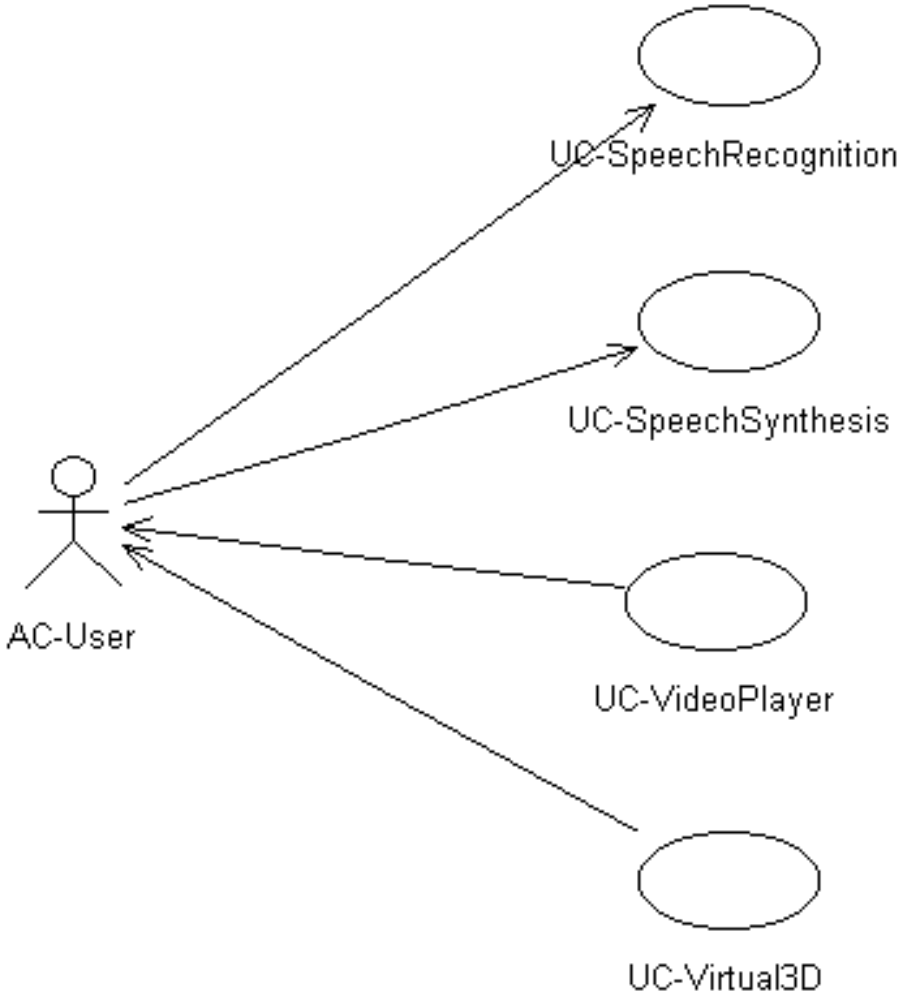
# Analyse UML : les diagrammes

- Diagramme des cas d'utilisation
- Diagramme de composants
- Diagramme de déploiement
- Diagramme de classes
- Diagramme de collaboration
- Diagramme de séquences
- Diagramme d'états

# Diagramme des cas d'utilisation

- Interaction externes au système (utilisateur(s), ...) <--> le système informatique

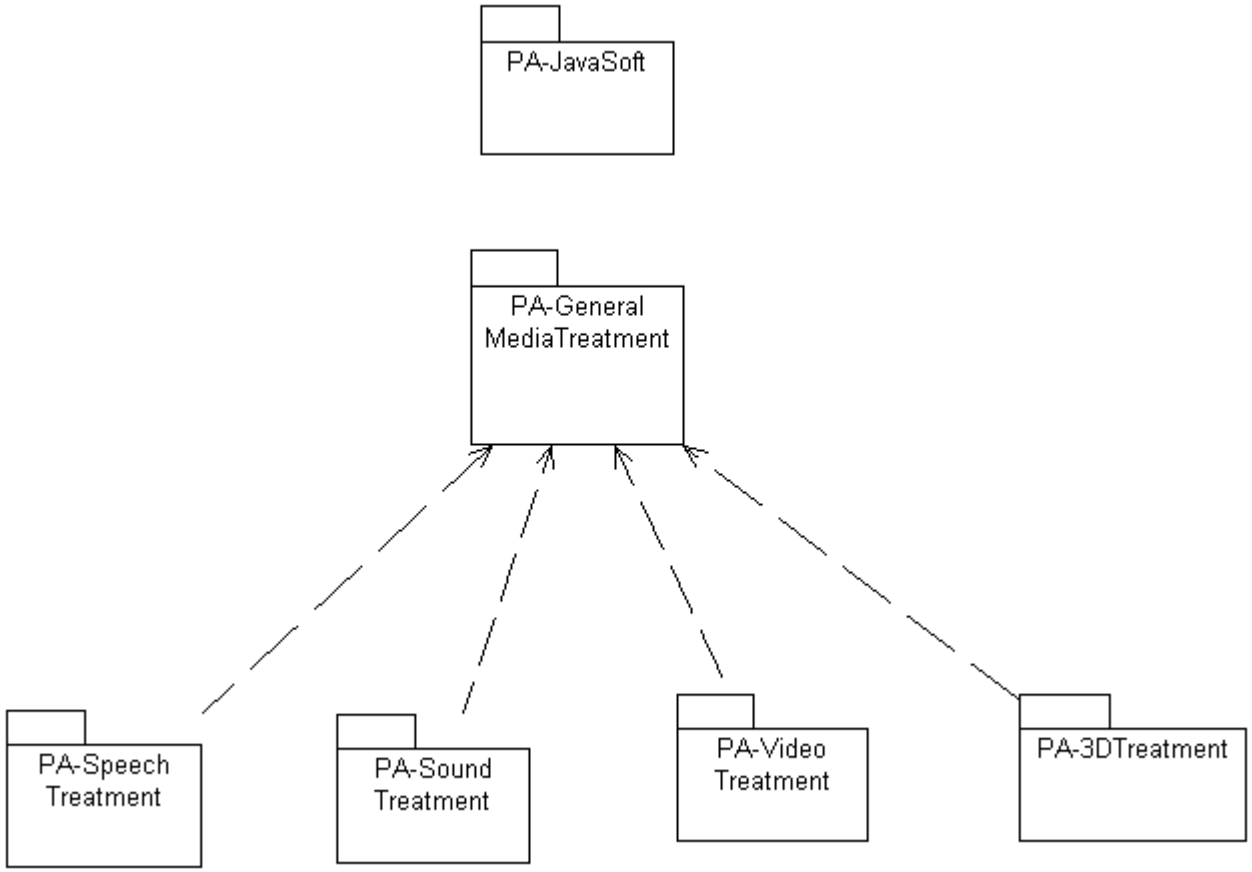
# Cas d'utilisation



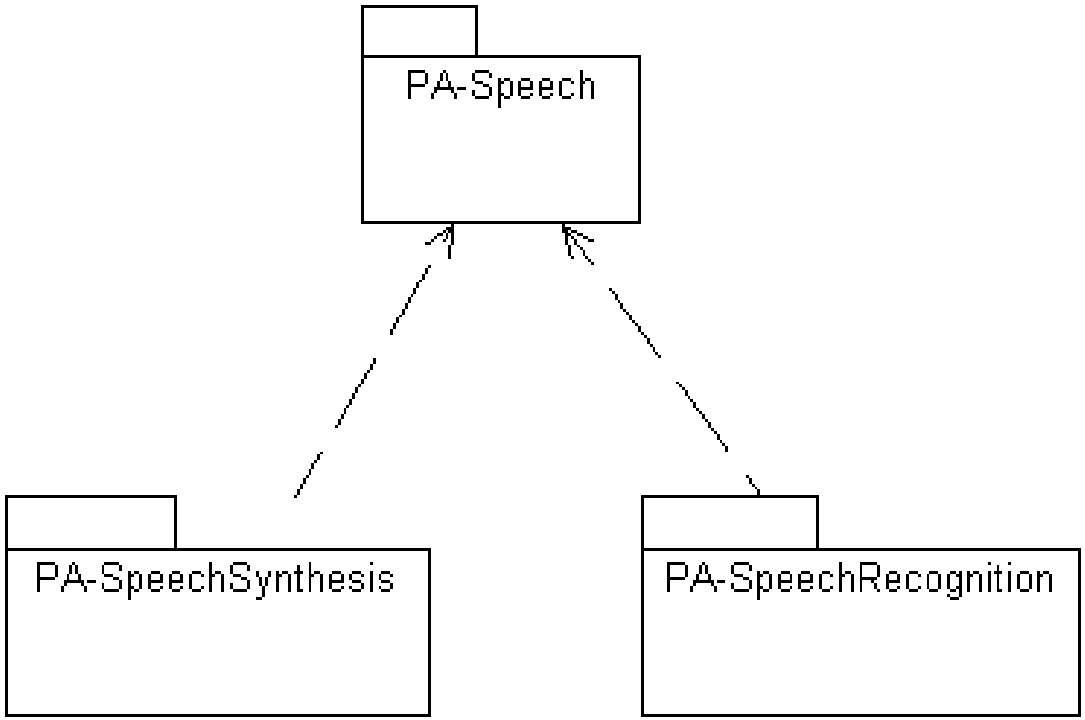
# Diagramme de composants

- composant = module logique de code
- Souvent regroupé dans un paquetage (c'est le cas ici pour ... simplifier !)

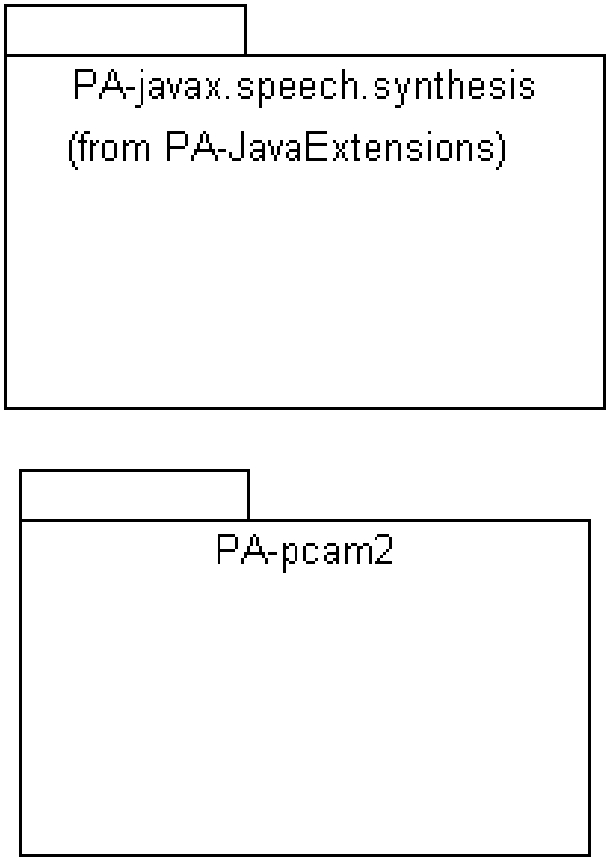
# Diagramme général des composants



# Diagramme des composants : traitement de la parole



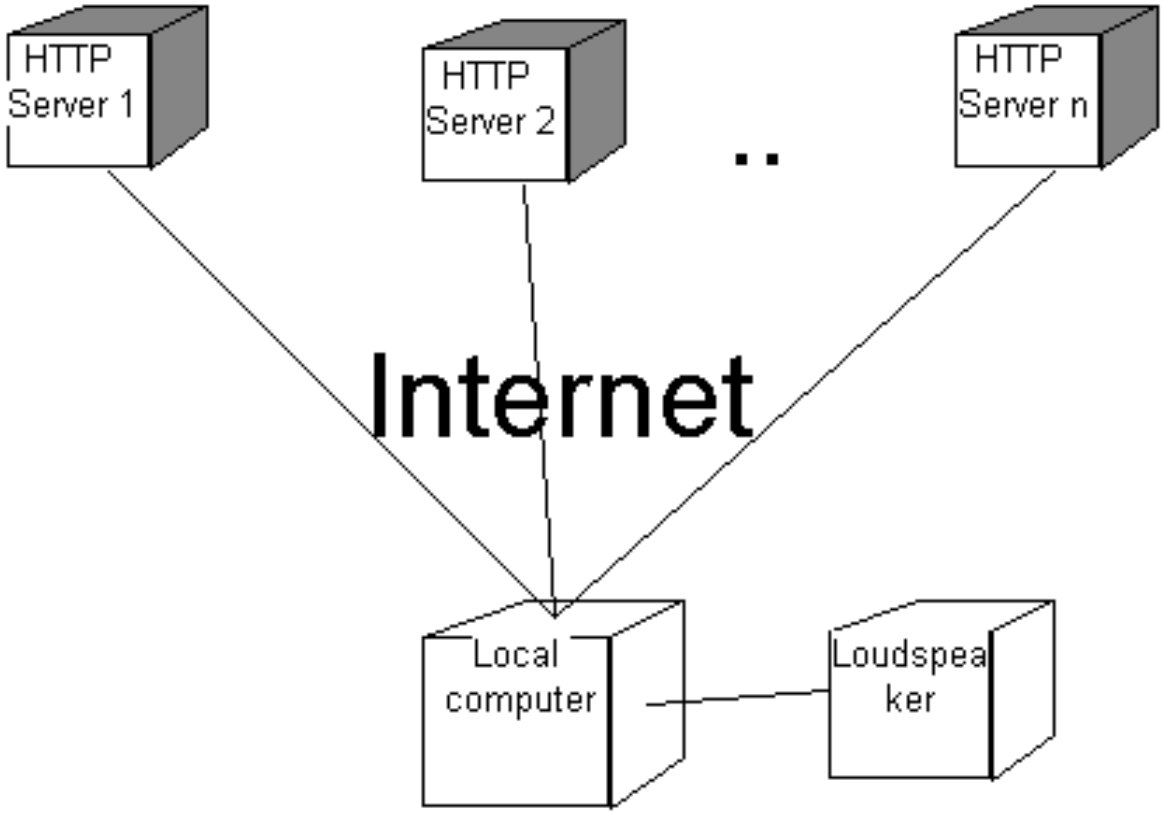
# Diagramme des composants : synthèse de la parole



# Diagramme de déploiement

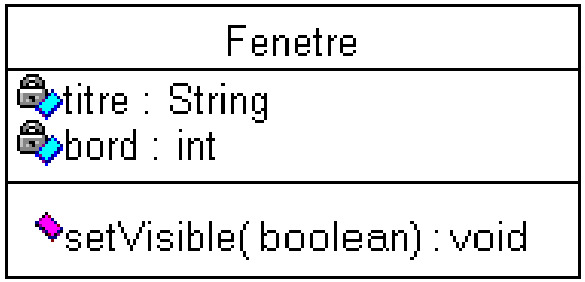
- = Diagramme de mise en service
- Indique les différents "acteurs logiciels et matériels" lors de l'utilisation du système

# Diagramme de déploiement



# Notion orientés objets

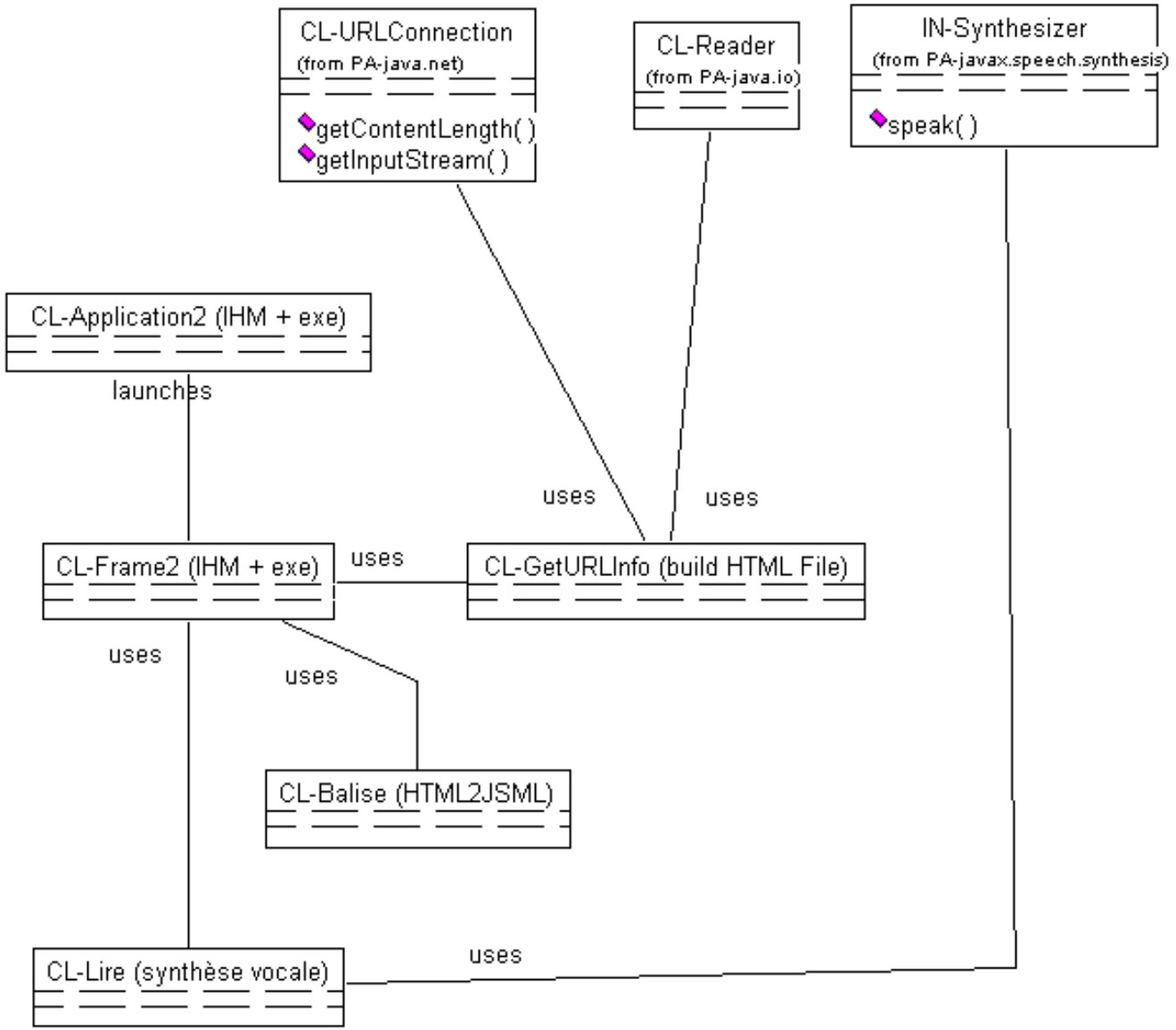
- Classe = type = caractéristiques communes à tous les objets de cette classe
- Classe = description des données et du code possédés par les objets de la classe
- Exemple : Les fenêtres ont un bord, un titre et on peut leur demander d'être affichées ou pas.



# Diagramme de classes

- Définitions des classes et relations entre ces classes

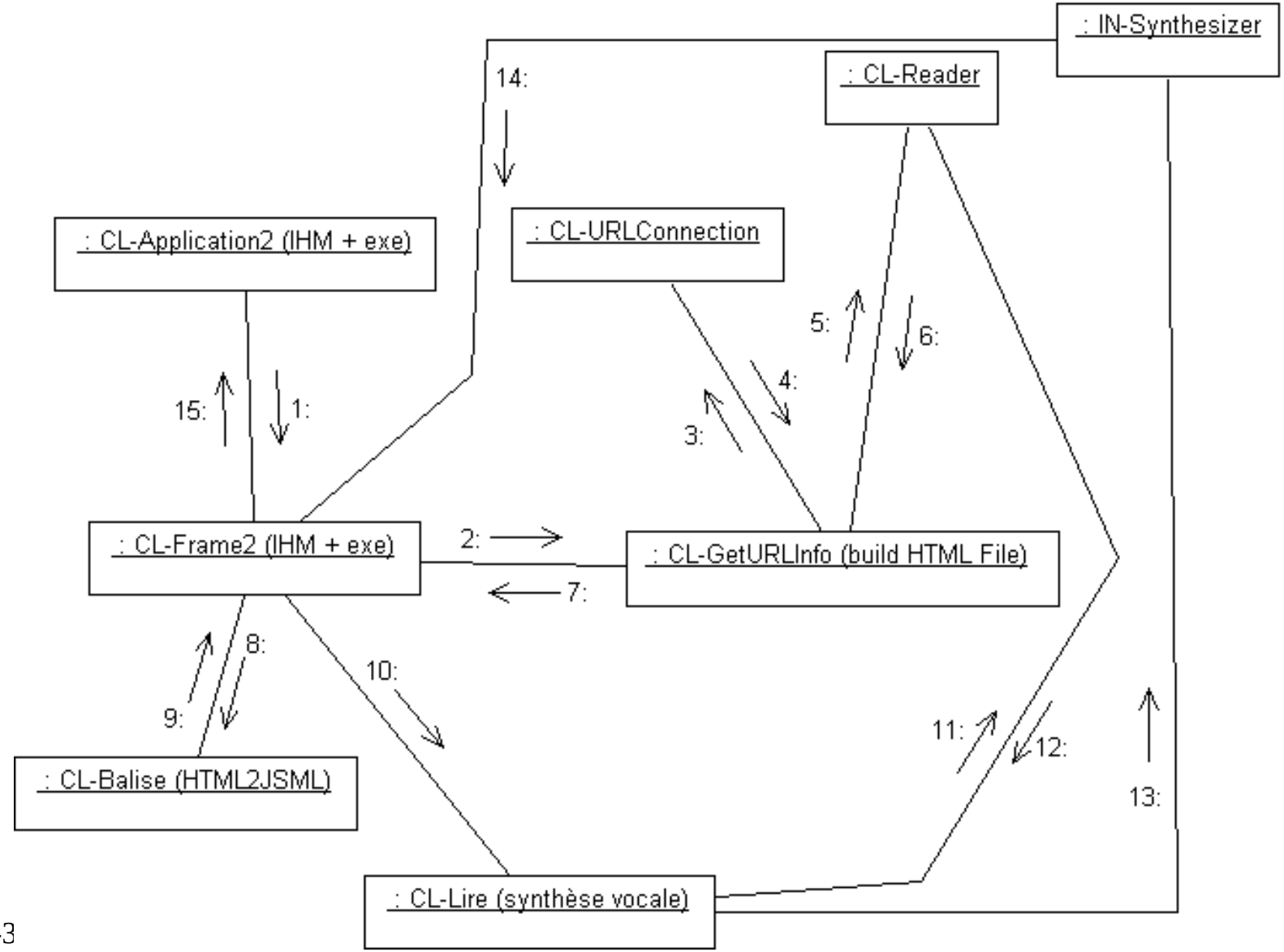
# Diagramme de classes



# Diagramme de collaboration

- Liens entre objets et les messages échangés entre eux.

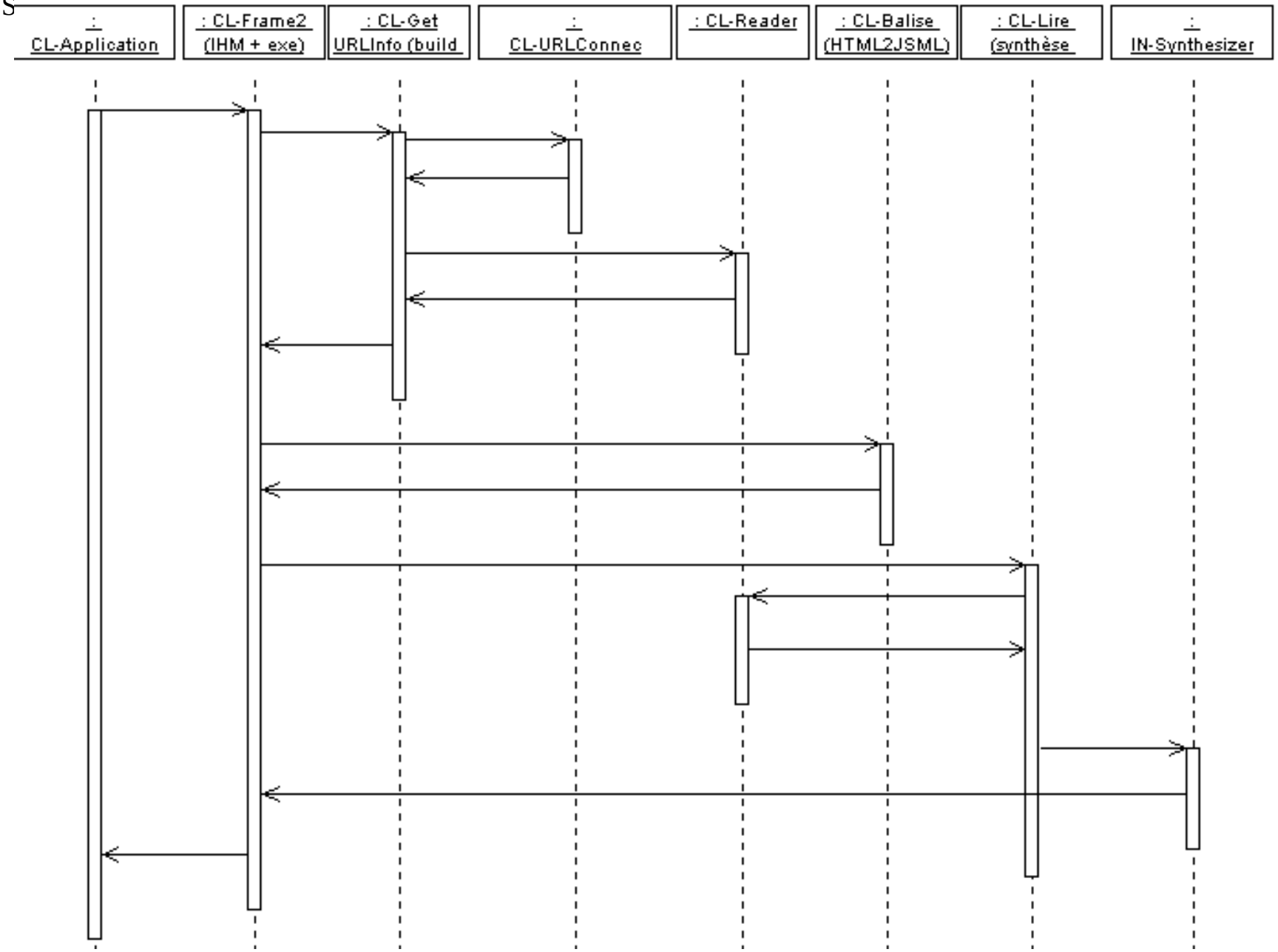
# Diagramme de collaboration



# Diagramme de séquences

- Description d'un scénario d'utilisation du logiciel

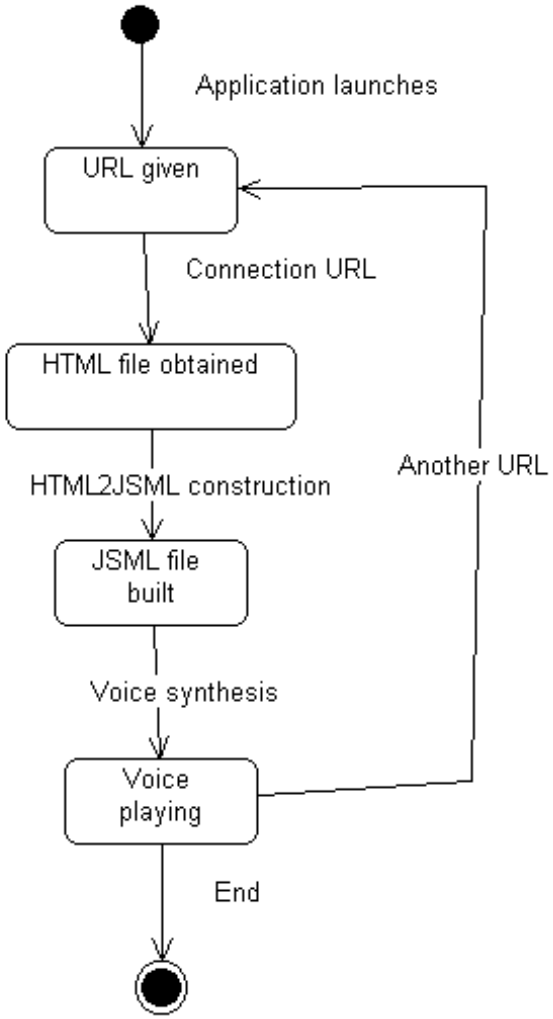
Diagramme de séquence entre des objets des classes de PA-pcam2



# Diagramme d'états

- Description du comportement d'un objet i.e. les différents états dans lequel il peut être durant l'exécution du logiciel, les causes (et les conséquences) d'un changement d'état.
- = Cycle de vie d'un objet

# Diagramme d'états de l'objet frame2



# Programmer le traitement de la parole

# Des buts et un moyen

- Intégrer la parole dans les interfaces homme-machine
- En sortie => synthèse
- En entrée => reconnaissance
- Ecrire des programmes qui "parlent correctement" (emphase, fin de phrase, ...) qui lance des actions lors de phrases reconnues.
- Un outil : Java

# Java Speech

- = Spécification des acteurs du monde du traitement de la parole dans les programmes IHM (SUN + Apple Computer, AT&T, Dragon Systems, IBM, Novell, Philips Speech Processing, and Texas Instruments Incorporated)
- Sun ne fournit pas d'implémentation complète de Java Speech
- Voir à `http://java.sun.com/products/java-media/speech`

# Java Speech (suite)

- Permet la synthèse et la reconnaissance de la parole
- Utilisent des couches basses des moteurs de synthèse et reconnaissance, couches non livrées avec Java SE.
- Traitement fonction d'une langue français, anglais, américain, espagnol, allemand, italien et d'une voix.

# Implantation de Java Speech

- IBM's "Speech for Java" (`http://www.alphaworks.ibm.com/tech/speech`)
  - avec Via Voice. Pour l'anglais US et UK, français, allemand, italien, espagnol, japonais.
  - Windows 9x, NT et Linux (RedHat Linux 6)
  - **Remarque : cette implémentation n'est (malheureusement !) plus disponible depuis Janvier 2006**

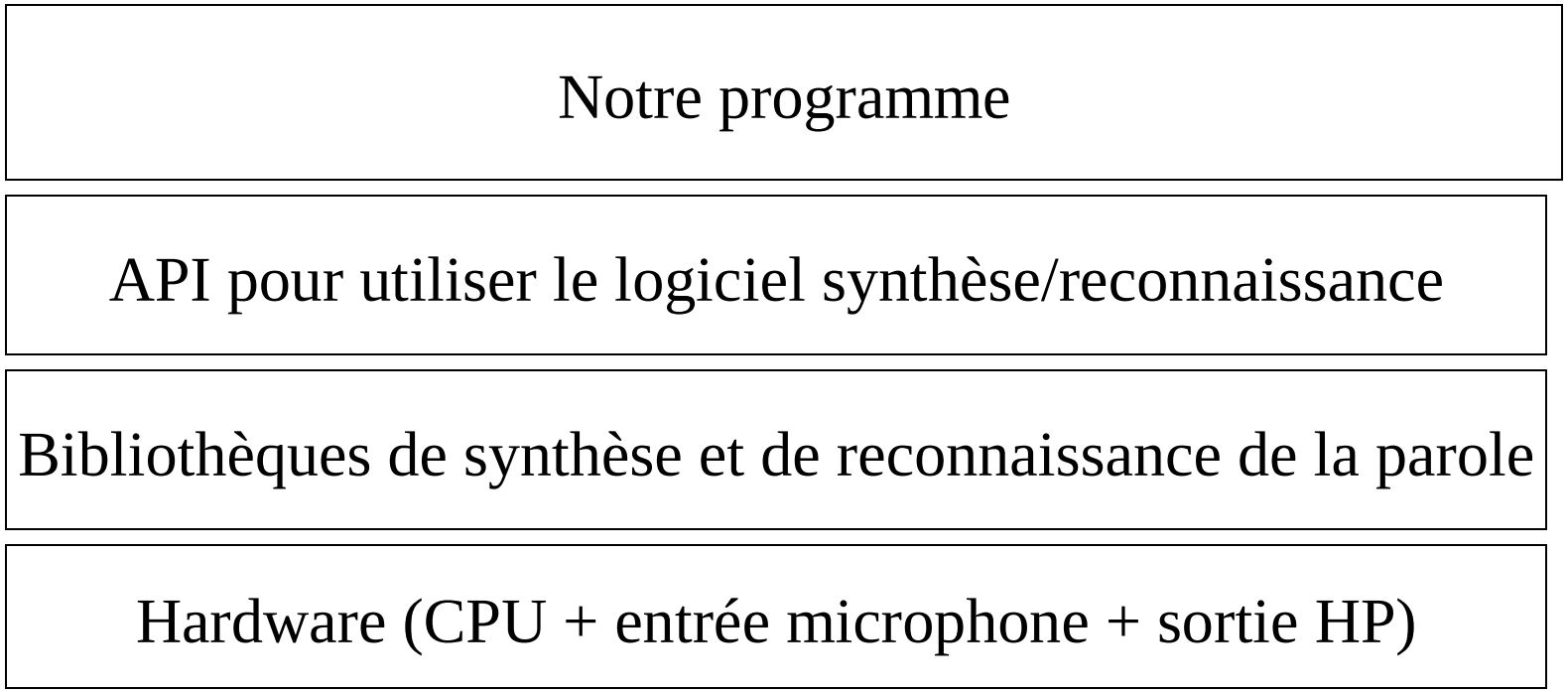
# Implantation de Java Speech (suite)

- ScanSoft (anciennement Lernout & Hauspie's) (`http://www.lhs.com/`)
  - Sun Solaris OS version 2.4 et plus.Festival (U. Edimbourg, …) sur les Unix
- Conversa Web 3.0 : browser avec de la parole pour Win32
- voir à : `http://java.sun.com/products/java-media/speech/forDevelopers/jsapifaq.html`

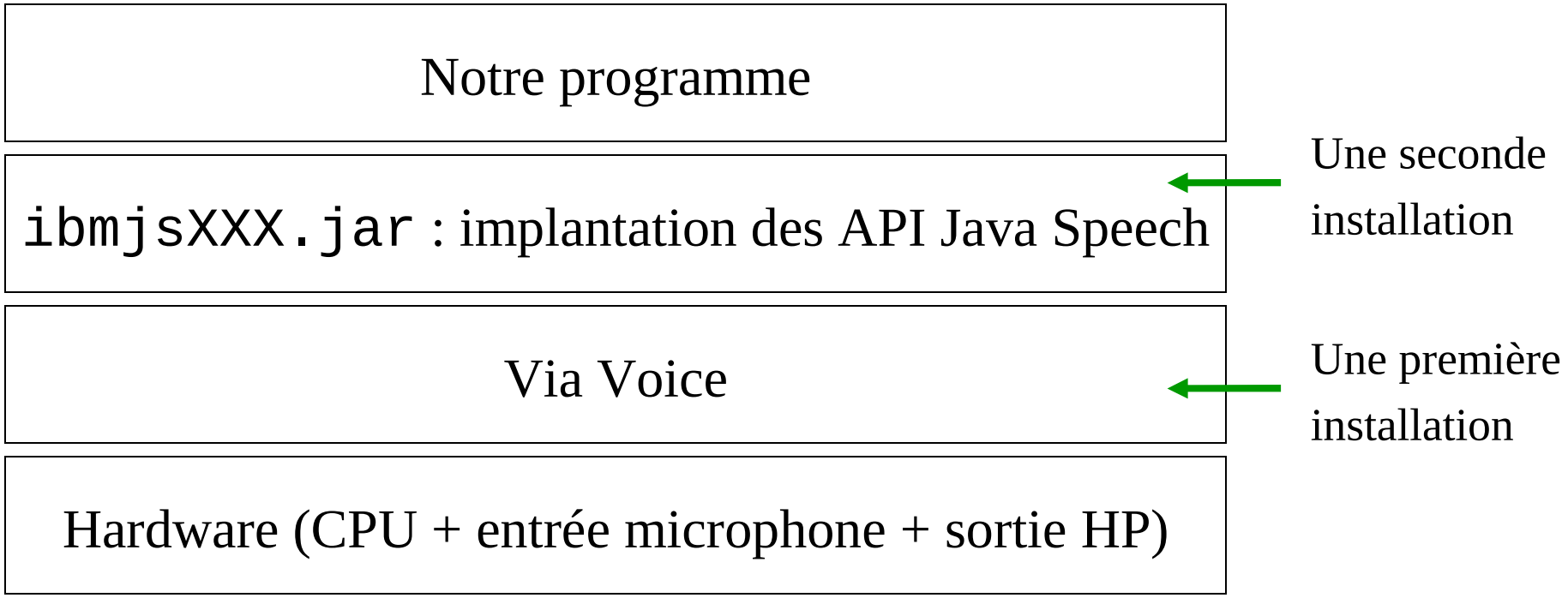
# Pour faire du traitement de la parole en Java

- Il faut deux "couches logicielles" pour utiliser la parole en Java :
  - des bibliothèques natives de traitement de la parole
  - les APIs Java Speech qui modélisent les notions du domaine et accède à ces bibliothèques natives
- Cette implantation utilise du code propre à la plate-forme (méthodes natives) : 30 ans de travail et de recherche à IBM.

# La "pile" pour traiter la parole



# La "pile" Java Speech d'IBM
## (plus disponible en janvier 2006 !)




C'est la partie `ibmjsXXX.jar` qui n'est plus disponible en janvier 2006

# Une des grandes idées de Java

- On décrit d'abord ce qu'on voudrait avoir.
- On implémentera plus tard
- cf. techniques Java pour l'entreprise : JDBC (accès aux BD), JNDI (nommage universel), …
- … et le multimédia

# Description en Java

- A l'aide d'une interface

```
package javax.speech.recognition;
/**
 * A Grammar defines a set of tokens (words) that may
 * be spoken and the patterns in which those tokens may
 * be spoken. Different grammar types (dictation vs.
 * rule) define the words and the patterns in different
 * ways.
 */
public interface Grammar {
/**
 * Request notifications of events from any Result that
 * matches this Grammar.
 */
public void addResultListener(ResultListener listener);
...
}
```

# Implantation en Java

- A l'aide d'une classe

```
package cnam.rsx206.speech;

import javax.speech.recognition.*;
/**
 * A Grammar defines a set of tokens (words) that may
 * be spoken and the patterns in which those tokens may
 * be spoken. Different grammar types (dictation vs.
 * rule) define the words and the patterns in different
 * ways.
 */
public class MaClasse implements Grammar {
/**
 * Request notifications of events from any Result that
 * matches this Grammar.
 */
public void addResultListener(ResultListener listener)
    {// le code spécifique …}
    ...
}
```

# Description/Implantation en Java

- Le compilateur vérifie que la classe donne bien un corps à la méthode spécifiée par l'interface
- => L'interface a indiqué complètement la signature de la méthode à implanter
- => tous les développeurs savent que c'est cette méthode qui est appelée
- => Le langage, les environnements aussi !!

# Package = {classes} U {interfaces} U {exceptions}

Adresse http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-doc/javax/speech/synthesis/package-summary.html

Overview | **Package** | Class | Tree | Index | Help

PREV PACKAGE | NEXT PACKAGE FRAMES | NO FRAMES

## Package javax.speech.synthesis

| Interface Summary | |
|---|---|
| ***Speakable*** | An object implementing the `Speakable` interface can be provided to the `speak` method |
| ***SpeakableListener*** | The listener interface for receiving notification of events during spoken output of a `Speak` |
| ***Synthesizer*** | The `Synthesizer` interface provides primary access to speech synthesis capabilities. |
| ***SynthesizerListener*** | An extension to the `EngineListener` interface for receiving notification of events assoc |
| ***SynthesizerProperties*** | Provides control of the run-time properties of a `Synthesizer`. |

| Class Summary | |
|---|---|
| **SpeakableAdapter** | Adapter that receives events associated with spoken output of a `Speakable` object. |
| **SpeakableEvent** | Event issued during spoken output of text. |
| **SynthesizerAdapter** | Adapter that receives events associated with a `Synthesizer`. |
| **SynthesizerEvent** | Event issued by `Synthesizer` to indicate a change in state or other activity. |

# Synthèse de la parole (TTS)

# Synthèse de la parole (TTS)

- Texte => phrases parlées : Text To Speech
- Il faut :
  - 1) Analyser le texte d'entrée en paragraphes, phrases, début et fin de phrase pour une meilleure intonation.
  - 2) Repérer les constructions idiomatiques de la langue (abréviation, lecture des dates, des sommes d'argent, ...) et savoir les différencier :
    - St. Mathews hospital is on Main St. -> "Saint Mathews hospital is on Main street"
  - 3) Convertir chaque mot en suite de phonèmes (unité sonore élémentaire)
  - 4) Traiter la prosodie i.e. le rythme, la "mélodie" de l'élocution, ...
  - 5) La production sonore

# Le synthétiseur de la parole

- Est fonction d'une langue (français, allemand, anglais).
- Les divers états d'un synthétiseur sont l'activation, la mise en pause, la reprise et l'arrêt de la lecture.

# Programme (complet) de synthèse de la parole



```
import javax.speech.*;
import javax.speech.synthesis.*;
import java.util.Locale;

public class HelloWorld2 {
    public static void main(String args[]) {
        try {
            // Récupérer le synthetiseur francais
            Synthesizer synth = Central.createSynthesizer(
                new SynthesizerModeDesc(Locale.FRENCH));

            // Prepare le synthetiseur pret a parler
            synth.allocate();
            synth.resume();

            // Prononce une phrase "C'est  ... "
            String phraseAProndncer = "C'est ";
            for (int i=0; i < args.length; i++)
                phraseAProndncer += args[i] + " ";
            synth.speakPlainText(phraseAProndncer, null);

            // Attend jusqu'a la fin de la lecture
            synth.waitEngineState(Synthesizer.QUEUE_EMPTY);

            // Désalloue le synthetiseur
            synth.deallocate();
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

# Programme (complet) de synthèse de la parole

- Une démonstration :
- `2litValCCAM.bat`

ou

`2bisLitUERSX206.bat`

# Explication du programme

- Recherche du synthétiseur français
- Positionne le synthétiseur prêt à parler.
- Utilisation de `speakPlainText()`
- MAIS, on aimerait :
  - lire des phrases avec une certaine voix, une certaine prosodie, en insistant sur certains mots, avec des volumes (du chuchotement au volume maximum) et changer cela dynamiquement.

# Le synthétiseur de la parole (suite)

- On peut, dynamiquement, changer certaines caractéristiques comme :
  - le volume sonore : du chuchotement au volume maximum.
  - la vitesse d'élocution
  - la tessiture (hauteur moyenne et intervalle autour de cette moyenne) de la voix
  - le style de voix (homme, femme, robot, agé, jeune, enrhumé, heureux, ...)

# Synthèse de la parole : JSML

- JSML = langage de balisage (DTD XML)
- Annotation du texte voir à
  `http://java.sun.com/products/java-media/speech/forDevelopers/JSML/index.html`
- Exemple de balises JSML
  - <BREAK MSECS="1000"/> arrêt de la lecture pendant 1 seconde.
  - <EMP attributs> ... </EMP> insister sur le texte compris entre les balises.
    <EMP LEVEL="strong">mesdames</EMP>
  - <SAYAS attributs> ... </SAYAS>
    <SAYAS SUB="S S deux zI">SSII</SAYAS>
  - <PROS attributs> ... </PROS> caractéristique de prosodie (tessiture de la voix, vitesse d'élocution, ...)
    <PROS RATE="150">texte dit à 150 mots par minute</PROS>
    <PROS PITCH="100">voix grave</PROS>
    <PROS PITCH="200">voix aiguë</PROS>
    <PROS VOL="0.1">chuchotement</PROS>
    <PROS VOL="0.9">je parle très fort</PROS>

# Démonstration JSML

```
"Bienvenue à ce cours. Bonjour
  <EMP LEVEL="strong">mesdames</EMP>, bonjour
 <EMP LEVEL="reduced">, messieurs</EMP>
 <BREAK MSECS="1000" />
 avec la balise SAYAS, le sigle<SAYAS SUB="S S deux zI">SSII</SAYAS>
 <BREAK MSECS="1000" />
  sans balise SSII
  <BREAK MSECS="1000" />
 <PROS PITCH="100">essai d'une voix grave</PROS>
 <PROS PITCH="200">essai d'une voix aiguë</PROS>
 <BREAK MSECS="1000" />
 <PROS VOL="0.1">en parlant très doucement</PROS>
 <BREAK MSECS="1000" />
  <PROS VOL="0.9">ou encore très fort.</PROS>
 <BREAK MSECS="1000" /> Etes vous convaincu ?
  fin du message"
```

- `3bienvenue.bat`

# Programmation JSML

```
import javax.speech.*;
import javax.speech.synthesis.*;

 public class Bienvenue {
    public static void main(String args[]) {
       try {
          Synthesizer synth = Central.createSynthesizer(
             new SynthesizerModeDesc(Locale.FRENCH));
          synth.allocate();
          MonSpeakable monSpeak = new MonSpeakable();
          synth.speak(monSpeak, null);
          synth.speak("fin du message", null);
     }
```

# Programmation JSML (suite)

- avec :

```
class MonSpeakable implements Speakable {
   public String getJSMLText() {
     StringBuffer buf = new StringBuffer();
    buf.append("Bienvenue à ce séminaire. Bonjour ");
 buf.append("<EMP LEVEL=\"strong\">" + "mesdames" + "</EMP>");
 buf.append("<EMP LEVEL=\"reduced\">" + ", messieurs" +
   "</EMP>");
        ...
        return buf.toString();
  }
}
```

# Programmation JSML (fin)

- Utilisation de la méthode `speak(Speakable, SpeakableListener)` sur l'objet synthétiseur.
- `Speakable` est une interface qui spécifie la méthode `public String getJSMLText()` qui doit renvoyer une `String` contenant un texte (éventuellement) balisé JSML.

# Reconnaissance de la parole

# Reconnaissance de la parole

- Java Speech se "restreint à" :
  - une seule langue
  - une seule entrée audio
- Les implémentations de Java Speech peuvent s'adapter à une voix quelconque
- L'ensemble des phrases à reconnaître = les grammaires peuvent être dynamiquement chargées.

# Travail d'un reconnaisseur de parole

- Traitement grammatical : créer et activer une grammaire pour connaître ce qui est significatif d'écouter
- Analyse du signal : Faire une analyse spectrale de l'entrée audio
- Reconnaissance des phonèmes : comparer les motifs spectraux aux phonèmes du langage

# Travail d'un reconnaisseur de parole (suite)

- Reconnaissance des mots : comparer les suites de phonèmes aux mots à reconnaître spécifiés par la grammaire active.
- Générer le résultat : avertir l'application des phrases de la grammaire qui ont été reconnues.

# Démonstration

- Dialogue entre l'humain et l'ordinateur

ordinateur> Bonjour humain, mon nom est ordinateur, quel est votre nom ?
humain> Je m'appelle *prénom nom*
ordinateur> Bonjour *prénom nom*
humain> Répétez après moi
ordinateur> J'écoute

humain> dictée terminée par "C'est fini"
ordinateur> répète la dictée

humain> au revoir
ordinateur> A bientôt

- `4demoJS`

# Remarques sur la démo

- reconnaissance de C'est fini, au revoir, ... ainsi que des non-terminaux *prénom*, *nom*
- reconnaissance d'une dictée quelconque
- synthèse de la parole :
  - traitement associé (bonjour *prénom nom*)
  - synthèse de la dictée
- Remarque : programme très facilement transposable en allemand, anglais, etc.

# Reconnaissance de la parole

- Demande d'indiquer la langue, l'empreinte vocale de l'utilisateur
- Demande d'indiquer une grammaire :
  - de dictée continue (reconnaissance du langage naturel)
  - ou de commandes vocales

# Les "Grammars"

- Une grammaire (`Grammar`) doit être créée et activée pour un reconnaisseur (`Recognizer`) afin qu'il sache ce qu'il doit écouter principalement.
- *DictationGrammar* : grammaire de dictée = ce qui permet de reconnaitre les mots d'une langue naturelle (pour les dictées dans les mails, les traitements de texte, ...)
- *RuleGrammar* = grammaire de règles = grammaire hors contexte (construite à l'aide de JSGF) donnée par un programmeur décrivant des phrases susceptibles d'être dites par un utilisateur.

# Reconnaissance de la parole (suite)

- Les états d'un reconnaisseur sont l'activation, la désactivation.
- Il gère les diverses grammaires qu'il utilise
- Java Speech utilise la technique des auditeurs lorsqu'une phrase a été reconnue.

# Auditeur (Listener) en Java

- Une des idées en Java : programmation par délégation
- cf. cours IHM sur les événements
- A un objet susceptible d'être utilisé à n'importe quel moment (par l'utilisateur ou autre ...) est associé des objets "auditeur" qui sont avertis lorsque l'objet source a été activé.

# Auditeur (Listener) en Java

- Ces auditeurs lancent alors une méthode appropriée spécifiée à l'avance
- Question :
  - comment Java le permet il ?
- Réponse :
  - interface, méthodes spécifiées dans cet interface, classe qui doit implanter cette interface, `Vector` de listeners dans l'objet.

# Grammaire de dictée (Dictation Grammar)

- Une telle grammaire impose peu de contraintes i.e. s'intéresse à tout ce qui peut être dit dans une langue donnée
- Dictation Grammar = "grammaire" d'un langage naturel (français, anglais, etc.)
- Nécessite plus de ressources système que les rule grammar

# Grammaire de dictée (Dictation Grammar)

- On ne créée pas, dans le programme, de dictation grammar : on récupère une telle grammaire à "l'aide des couches basses"
- Peut être optimisées pour certains domaines (médecine, commercial, etc.)

# Grammaire de règles (Rule Grammar)

- Java Speech permet :
  - de définir des grammaires (de règles) qui décrivent ce qu'un utilisateur peut dire
  - d'associer un traitement aux phrases reconnues.
- Grammaires de règles sont définies par JSGF (Java Speech Grammar Format)
- Spécifications de JSGF à : `http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/index.html`

# Programmer la reconnaissance de la parole

- Les diverses notions vues précédemment sont modélisées par des objets de classes.
- Ces objets sont :
  - construits (par `new UneClasse()`)
  - récupérés car accessibles par la bibliothèque Java Speech.

# Un objet pour la reconnaissance de la parole

- Un reconnaisseur de parole est une "machine" qui convertit de la parole en format texte.
- Les classes et interfaces utiles se trouvent dans le paquetage `javax.speech.recognition`.
- dont l'interface `Recognizer` (qui hérite de `javax.speech.Engine`).

# Reconnaissance : un premier exemple simple

- Voir à : `http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-guide/Recognition.html`

```
grammar javax.speech.demo;

public <phrase> = bonjour à tous
    | meilleurs voeux
    | merci mon cher ordinateur;
```

- `5RecSimple.bat`

# Reconnaissance : un premier exemple simple (et complet) 1/3

```
import javax.speech.*;
import javax.speech.recognition.*;
import java.io.FileReader;
import java.util.Locale;

public class SimpleReconnaissance extends ResultAdapter {
  Recognizer rec;
  public static void main(String args[]) {
    try {
        // Récupère un reconnaisseur de parole pour la langue
        // française.
       rec = Central.createRecognizer(new
EngineModeDesc(Locale.FRENCH));

       // Lance ce reconnaisseur
       rec.allocate();

       // Charge un fichier contenant une grammaire de règles,
       // construit un objet RuleGrammar à l'aide de ce fichier et
       // le rend utilisable.
       FileReader reader = new FileReader(args[0]);
       RuleGrammar gram = rec.loadJSGF(reader);
       gram.setEnabled(true);
```

# Reconnaissance : un premier exemple simple (et complet) 2/3

```
      // Associe un listener à ce reconnaisseur
      rec.addResultListener(new SimpleReconnaissance());

      // Avertit des changements d'états du reconnaisseur
      // ici sa création
      rec.commitChanges();

      // Demande le focus et lance l'écoute
      rec.requestFocus();
      rec.resume();
      System.out.println("L'ordinateur vous écoute");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
```

# Reconnaissance : un premier exemple simple (et complet) 3/3

```
// Reçoit uniquement les événements RESULT_ACCEPTED :
// Ecrit ce qui a été reconnu,
// désalloue les ressources et termine le programme

public void resultAccepted(ResultEvent e) {
    Result r = (Result)(e.getSource());
    ResultToken tokens[] = r.getBestTokens();

    for (int i = 0; i < tokens.length; i++)
       System.out.print(tokens[i].getSpokenText() + " ");
    System.out.println();

    rec.deallocate();
    System.exit(0);
}
}
```

# Explication de demoJS

- grammaire de texte à reconnaître

hello_fr.gram

```
grammar hello;
  <first> = Henri {Henri}
          | Maurice {Maurice}
          | Victor {Victor}
  ;
  <last>  = Matisse {Matisse}
          | Chevalier {Chevalier}
          | Hugo {Hugo}
 ;
  <name> = <first> <last>;
  public <nameis> = Je m'appelle {name} <name>;
  public <begin> = Répétez après moi {begin};
  public <stop> = C'est fini {stop};
  public <bye> = Au revoir {bye};
```

# Une Rule grammar

- A une telle grammaire sera associé un objet `RuleGrammar`
- Une règle `public` est une règle qui est activée lorsqu'elle a été reconnue.
- La balise (tag) {} est retournée sous forme de `String` lorsque la règle a été reconnue.
- < > = non terminaux
- terminaux écrits en texte plein

# Rule grammar

- | = ou bien
- [ ] optionnel (i.e. 0 ou 1)
- () = regroupement
- * = 0 ou plusieurs

# Une Rule grammar : exercice

- Donner des phrases construites par la grammaire :

```
grammar SimpleCommandes;
public <Commande> = [<Politesse>]
<Action> <Objet> (et <Action>
<Objet>)*;
<Action> = ouvrez | fermez;
<Objet> = la fenêtre | la porte;
<Politesse> = s'il vous plaît;
```

# Programme reconnaissance de la parole

- Chargement d'un reconnaisseur de parole
- Chargement de la grammaire
- Associer un auditeur au reconnaisseur
- L'auditeur lance sa méthode `resultAccepted()` lorsqu'une règle a été reconnue.

# Reconnaissance/synthèse de la parole

- Le fichier de configuration (de propriétés) entre autre pour les réponses de l'ordinateur.

res_fr.properties

```
# file for grammar (pour la reconnaissance)
grammar   = hello_fr.gram

# things we say (pour la synthèse)
greeting  = Bonjour Humain. Mon nom est Ordinateur. Quel
est <EMP/>votre nom?
hello     = Bonjour
listening = J'écoute.
eh        = Pardon? Eh? Quoi?
bye       = À bientôt!
```

# Reconnaissance/synthèse de la parole (suite)

- Le fichier de grammaire pour les phrases que doit reconnaître l'ordinateur.

`hello_fr.gram`

```
grammar hello;

<first> = Henri {Henri}
        | Maurice {Maurice}
        | Victor {Victor}
     ;

<last>  = Matisse {Matisse}
        | Chevalier {Chevalier}
        | Hugo {Hugo}
     ;

<name> = <first> <last>;
public <nameis> = Je m' appelle {name} <name>;
public <begin> = Répétez après moi {begin};
public <stop> = C'est fini {stop} | Fin de la dictée {stop};
public <bye> = Au revoir {bye};
```

# Rule Grammar: exemples

```
grammar hello;
  <first> = Henri {Henri}
          | Maurice {Maurice}
          | Victor {Victor}
  ;
  <last>  = Matisse {Matisse}
          | Chevalier {Chevalier}
          | Hugo {Hugo}
 ;
  <name> = <first> <last>;
  public <nameis> = Je m'appelle {name} <name>;
  public <begin> = Répétez après moi {begin};
  public <stop> = C'est fini {stop};
  public <bye> = Au revoir {bye};
```

← La rule grammar française

```
grammar hello;
<first> = Bruce {Bruce}
      | Andrew {Andrew}
      | Stuart {Stuart}
      ;
<last>  = Lucas {Lucas}
      | Hunt {Hunt}
      | Adams {Adams}
      ;
<name> = <first> <last>;
public <nameis> = My name is {name} <name>;
public <begin> = Repeat after me {begin};
public <stop> = That's all {stop};
public <bye> = Good bye {bye} | So long {bye};
```

← La rule grammar anglaise

# Multithreading Java

- Un programme Java lance plusieurs threads à l'exécution.
- Un programme Java se termine lorsque toutes les threads non démons sont terminées
- Ce qui explique que le programme suivant ne se termine pas.

# Multithreading Java (suite)

```
import java.awt.*;

public class ButtonTestApp extends Frame {
    public ButtonTestApp () {
        add("Center", new Button("Premier bouton"));
        pack();
        setVisible(true);
    }
    public static void main(String args[ ]) {
        Frame fr = new ButtonTestApp ();
    }
}
```

- Une fenêtre reste affichée. Pourquoi ?
- Car thread (non démon) d'affichage (le thread AWT)

# Classe anonyme en Java

- C'est un raccourci syntaxique permettant de créer "à la volée", un objet d'une classe sans indiquer le nom de la classe
- Syntaxe :

```
new ClasseDeBaseOuInterfaceAImplémenter() { … }
```

- Souvent mis comme argument d'une méthode ce qui donne :

```
ref.meth(new
ClasseDeBaseOuInterfaceAImplémenter() { … }
);
```

# Programme traitant la parole : syntaxe (1/7)

```
import java.io.*;
import java.util.*;
import javax.speech.*;
import javax.speech.recognition.*;
import javax.speech.synthesis.*;

public class Hello {
    static RuleGrammar ruleGrammar;
    static DictationGrammar dictationGrammar;
    static Recognizer recognizer;
    static Synthesizer synthesizer;
    static ResourceBundle resources;

    // L’auditeur pour la rule grammar. Sa méthode
    // resultAccepted() est appelée lorsqu'une
    // règle publique a été reconnue.
    // On teste alors les balises associées à cette grammaire
    // et on lance l'action correspondante à la balise
    // (filtre sur les balises).
    // Utiliser les balises plutôt que de tester
    // directement les phrases dites, permet d'être
    // indépendant de la langue. On peut alors changer la
    // langue sans changer ce programme.
    // Remarque : on utilise une classe interne


  static ResultListener ruleListener = new ResultAdapter() {
```

# Programme traitant la parole : syntaxe (2/7)

```
public void resultAccepted(ResultEvent e) {
        try {
        // On récupère les balises à l'aide de l'événement
FinalRuleResult result = (FinalRuleResult) e.getSource();
String tags[] = result.getTags();
    // L'utilisateur a dit la phrase correspondant à la
    //  règle "name" ("Je m'appelle ...", "my name is ...")
        if (tags[0].equals("name")) {
            // On construit la réponse ("bonjour …")
            // On récupère ce qui correspond à "hello" dans
            // le fichier de configuration des phrases locales
            String s = resources.getString("hello");
            for (int i=1; i<tags.length; i++)
                s += " " + tags[i];
                // on lance la synthèse vocale
                speak(s);
// L'utilisateur a dit la phrase correspondant à la
//  règle "begin"("répéter après moi" ou "repeat after me", …)
        } else if (tags[0].equals("begin")) {
        // l'ordinateur répond et se prépare pour la dictée
            speak(resources.getString("listening"));

        // la Rule grammar est désactivée (hormis sa règle
        // <stop>) au profit de la dictation grammar.
            ruleGrammar.setEnabled(false);
            ruleGrammar.setEnabled("<stop>", true);
            dictationGrammar.setEnabled(true);
            recognizer.commitChanges();
```

# Programme traitant la parole : syntaxe (3/7)

```
    // L'utilisateur a dit "c'est fini"
    } else if (tags[0].equals("stop")) {
      // la dictation grammar est désactivée
      // au profit de la rule grammar.
      dictationGrammar.setEnabled(false);
      ruleGrammar.setEnabled(true);
      recognizer.commitChanges();

    // L'utilisateur a dit "au revoir"
    } else if (tags[0].equals("bye")) {
      // L'ordinateur dit à bientot
      speak(resources.getString("bye"));
      if (synthesizer!=null)
      synthesizer.waitEngineState(Synthesizer.QUEUE_EMPTY);
      Thread.sleep(1000);
       System.exit(0);
     }
   } catch (Exception ex) { ex.printStackTrace();}
  }};

  // L'auditeur pour la dictation grammar.
  // Sa méthode resultUpdated() est appelée
  // pour chaque mot reconnu.
  // Sa méthode resultAccepted() est appelée
  // lorsque l'objet dictation grammar est désactivé i.e.
  //  quand l'utilisateur a dit "c'est fini".
   static ResultListener dictationListener = new
ResultAdapter() {
```

# Programme traitant la parole : syntaxe (4/7)

```
  int n = 0; // nombre de mots lus
  public void resultUpdated(ResultEvent e) {
        Result result = (Result) e.getSource();
        for (int i=n; i<result.numTokens(); i++)
System.out.println(result.getBestToken(i).getSpokenText());
            n = result.numTokens();
        }

    public void resultAccepted(ResultEvent e) {
        // On récupère tout ce qui a été dit
        Result result = (Result) e.getSource();
        String s = "";
        // L'ordinateur le répète
        for (int i=0; i<n; i++)
        // une seule instruction dans le for : IMPORTANT
            s += result.getBestToken(i).getSpokenText() + " ";
        speak(s);
        n = 0;
    }
}; // ce ; est juste si, si
    // L'audio listener imprime des traces du volume d'entrée
    static RecognizerAudioListener audioListener =
        new RecognizerAudioAdapter(){
            public void audioLevel(RecognizerAudioEvent e) {
                System.out.println("volume " + e.getAudioLevel());
            }
    };
```

# Programme traitant la parole : syntaxe (5/7)

```
    // La méthode qui "fait parler l'ordinateur".
   // Si le synthesizer n'est pas disponible
   // elle écrit les mots à l'écran.
    static void speak(String s) {
     if (synthesizer != null) {
       try {
         synthesizer.speak(s, null);
       } catch (Exception e) {e.printStackTrace();}
     } else {
       System.out.println(s);
     }
   }

    //
    // la méthode main()
    //
```

# Programme traitant la parole : syntaxe (6/7)

```
  public static void main(String args[]) {
    try {
      // Chargement de ressources locales
// la "culture" locale (exemple le français métropolitain fr_FR)
      System.out.println("locale is " + Locale.getDefault());

// recherche d'un fichier propre à cette culture res_fr.properties
      resources = ResourceBundle.getBundle("res");

// récupération et allocation du reconnaisseur par défaut :
// celui correspondant à la culture locale (null)
      recognizer = Central.createRecognizer(null);
      recognizer.allocate();
// ajout d'un audio listener (qui testera le volume sonore)
       recognizer.getAudioManager().addAudioListener(audioListener);

// récupération de la dictation grammar
      dictationGrammar = recognizer.getDictationGrammar(null);
// ajout d'un listener à cette dictation grammar
      dictationGrammar.addResultListener(dictationListener);

// crée une rule grammar à partir du fichier hello_fr.gram
// (lu dans le fichier des propriétés locales). Lui associe
// un listener et active cette grammaire.
     Reader reader = new FileReader(resources.getString("grammar"));
     ruleGrammar = (RuleGrammar) recognizer.loadJSGF(reader);
     ruleGrammar.addResultListener(ruleListener);
     ruleGrammar.setEnabled(true);
```

# Programme traitant la parole : syntaxe (7/7)

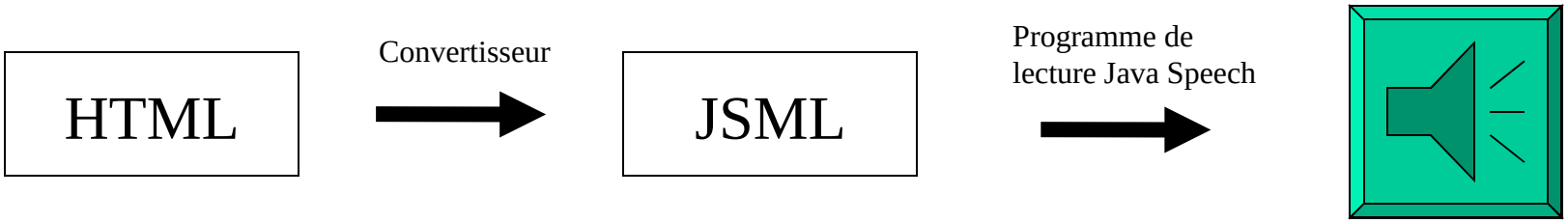
```
  // informe le reconnaisseur des états des grammaires et le lance.
    recognizer.commitChanges();
    recognizer.requestFocus();
    recognizer.resume();

  // Synthèse de la parole
  // crée un synthesizer correspondant à la culture locale (null),
  // lance un message de bienvenu (méthode statique de cette classe)
    synthesizer = Central.createSynthesizer(null);
    if (synthesizer!=null) synthesizer.allocate();
    speak(resources.getString("greeting"));
    } catch (Exception e) { e.printStackTrace();System.exit(-1);}
  }
}
```

# Exemples d'applications

- "Ecouter les pages Web" (Hervé Declipeur)
- Interroger une base de données par la parole (Patrice Gangnard projet valeur C IAGL 1999) : exercice C CAM 2005
- Méthode Assimil JavaSpeech Thomas Jgenti `http://jgenti.free.fr/cnam/cam/index.html`

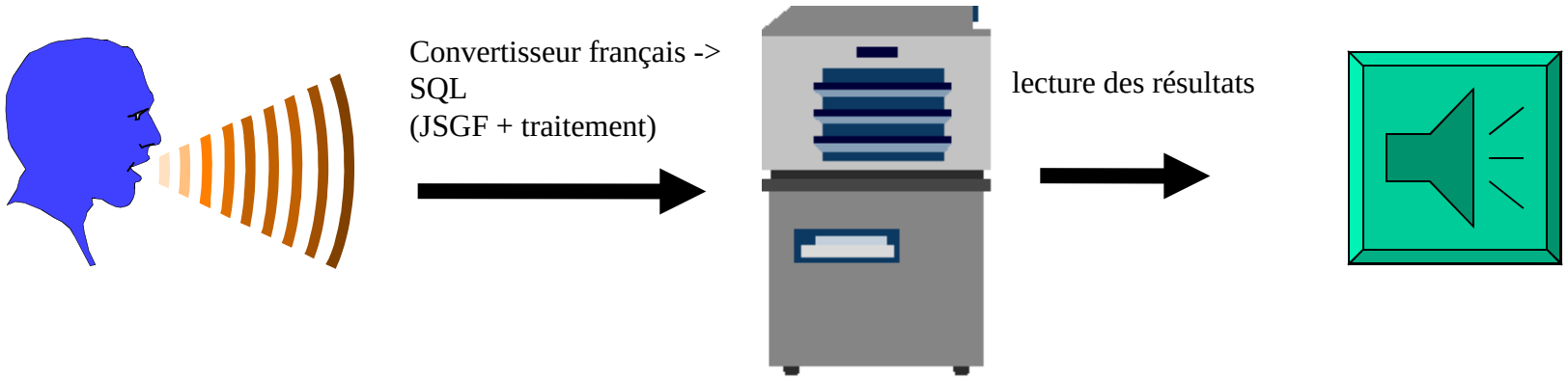
# Lecture de pages HTML

- Hervé Déclipeur (valeur C CAM du CNAM) RVDesign, `declipeur@noos.fr`



# Interroger une BD par la parole

- Patrice Gangnard projet valeur C IAGL 1999



# Bibliographie


- Cours UML Laurence Duchien (CNAM)
- The Unified Modeling Langage User Guide. G. Booch, J. Rumbaugh, I. Jacobson. Ed Addison Wesley. ISBN 0-201-57168-4
- `http://java.sun.com/products/java-media/speech` :

# Bibliographie (suite)

- `http://java.sun.com/products/java-media/speech` : page d'accueil Java Speech. Les technologies Java pour le traitement de la parole
- Récupérer les archives du groupe de discussion sur Java Speech à `http://archives.java.sun.com/javaspeech-interest.html`

# Bibliographie (suite)

- IBM's "Speech for Java" (`http://www.alphaworks.ibm.com/tech/speech`)
- Spécifications de JSGF à : `http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/index.html`

# Bibliographie (fin)

- `http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-guide/index.html` : le guide de programmation Java Speech

# Fin