



Sébastien SERAIS
sebastien.serais@esigelec.fr

Contenu du Cours

JSE :

Présentation de Java
Instructions
Exceptions
Les classes
Les interfaces
Les Collections
Le Graphisme
Les Threads
Les Bases de données : JDBC
Quelques outils

JEE :

Rappels formulaires HTML
Les Servlets
Les cookies
Les Sessions
Les Java Server Pages (JSP)
Présentation du Design Pattern
MVC2

Frameworks JAVA :

Les tests unitaires : JUnit
Maven
Hibernate
Struts 1
Log4J

Les Bases

Historique

Le langage Java a été mis au point en 1991, par la firme
Sun Microsystems
Premier nom : oak



Objectif initial :

- ☛ créer un environnement :
 - indépendant du hardware pour pouvoir équiper l'électroménager (téléviseurs, magnétoscopes, téléphones, machines à laver, ...).
 - Robuste
 - Sur
 - Supprimer les parties difficiles du C/C++
(pointeurs, gestion de la mémoire, sécurité..)

Père créateur : James Gosling



Historique

- ▶ 1994 : oak=échec
- ▶ 1995 : essor de l'Internet et des besoins nouveaux arrivent : dynamiser les pages web HTML → amélioration de oak → apparition des Applets (fonctionne sur tous les OS)
- ▶ Oak renommé Java en 1995
- ▶ 13 novembre 2006, le code source de Java - machine virtuelle et compilateur - est disponible sous licence GPL → **Java est libre !**

Historique

☕ Pourquoi Java ?

En l'honneur de la boisson préférée des programmeurs : le café, dont une partie de la production provient de l'île Java.

En Amérique du Nord, « java » est aussi un terme argotique pour désigner le café en général



Les versions de l'édition standard

| No version | année | Nom plateforme | Nom du kit de développement | Nom du Runtime Environment |
|------------|-------|-------------------|-----------------------------|----------------------------|
| 1.0 | 1995 | JDK (Oak) | JDK | JRE |
| 1.1 | 1997 | JDK | JDK | JRE |
| 1.2 | 1998 | J2SE (Playground) | J2SDK | J2RE |
| 1.3 | 2000 | J2SE (Kestrel) | J2SDK | J2RE |
| 1.4.0 | 2002 | J2SE (Merlin) | J2SDK | J2RE |
| 5.0 | 2004 | J2SE (Tiger) | J2SDK | J2RE |
| 6 | 2006 | Java SE (Mustang) | JDK | JRE |
| 7 | 2011 | Java SE (Dolphin) | JDK | JRE |
| 8 | 2014 | Java SE | JDK | JRE |
| 9 | 2017 | Java SE | JDK | JRE |

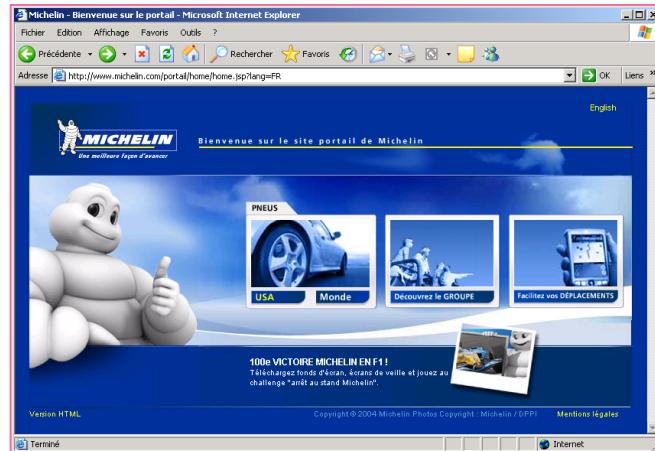
Les autres éditions

- ▶ Java EE (ancien J2EE) : Java Entreprise Edition

- Servlets/JSP
- JSF : JavaServer Faces
- EJB : Enterprise JavaBeans
- Web Services
- Java Persistence API
- JMS : Java Message Service
- ...

- ▶ JME : Java Micro Edition (ancien J2ME)
- ▶ → Applications mobiles

Aujourd’hui : Android



La plateforme JAVA

- ▶ La plateforme Java (Standard Edition) :

| | | Java Language | | | | | | | | | | |
|-----------------------|--------------------------------|-----------------------------------|---------------|---------------|-----------------------|-----------------|-----------------------|-----------|-------------------------|--|--|--|
| | | java | javac | javadoc | jar | javap | jdeps | Scripting | | | | |
| JDK | <u>Tools & Tool APIs</u> | Security | Monitoring | JConsole | VisualVM | JMC | JFR | | | | | |
| | | JPDA | JVM TI | IDL | RMI | Java DB | Deployment | | | | | |
| | | Internationalization | | Web Services | | Troubleshooting | | | | | | |
| | <u>Deployment</u> | Java Web Start | | | Applet / Java Plug-in | | | | | | | |
| JRE | <u>User Interface Toolkits</u> | JavaFX | | | | | | | | | | |
| | | Swing | | Java 2D | | AWT | Accessibility | | | | | |
| | | Drag and Drop | | Input Methods | | Image I/O | Print Service | Sound | | | | |
| | <u>Integration Libraries</u> | IDL | JDBC | JNDI | RMI | RMI-IIOP | Scripting | | | | | |
| Java SE API | | Beans | Security | | Serialization | | Extension Mechanism | | | | | |
| | <u>Other Base Libraries</u> | JMX | XML JAXP | | Networking | | Override Mechanism | | | | | |
| | | JNI | Date and Time | | Input/Output | | Internationalization | | <u>Compact Profiles</u> | | | |
| | | lang and util | | | | | | | | | | |
| Base Libraries | <u>lang and util</u> | Math | Collections | | Ref Objects | | Regular Expressions | | | | | |
| | | Logging | Management | | Instrumentation | | Concurrency Utilities | | | | | |
| | | Reflection | Versioning | | Preferences API | | JAR | Zip | | | | |
| | <u>Java Virtual Machine</u> | Java HotSpot Client and Server VM | | | | | | | | | | |

source : <http://docs.oracle.com/javase/8/docs/>

La plateforme JAVA

- ▶ **Javac** : compile le code source en bytecode
- ▶ **Java** : exécute un programme java
- ▶ **Javadoc** : permet de générer de la doc au format HTML
- ▶ **JPDA** : interface utilisée par les debugger dans les environnement de développement
- ▶ **Java Web Start** : technologie de déploiement d'applications
- ▶ **Java Plug-in** : permet l'exécution d'applets dans les browsers
- ▶ **AWT** : bibliothèque graphique
- ▶ **Swing** : bibliothèque graphique plus complète

La plateforme JAVA

- ▶ **Sound** : capture, traite, et lire des flux audio et midi
- ▶ **Input méthodes** : permet la saisie depuis d'autres périphériques
 - ▶ que le clavier (Chine,etc...)
- ▶ **Java 2D** : pour la gestion avancée de graphiques et d'images
- ▶ **Accessibility** : permet de rendre accessible les applis aux personnes possédant un handicap
- ▶ **RMI (Remote Method Invocation)**: permet la communication vers des objets distants pour des applications distribuées
- ▶ **JDBC : (Java DataBase Connectivity)** : permet les accès aux bases de données
- ▶ **JNDI (Java Naming and Directory Interface)**: permet de gérer les noms de fichiers indépendamment de l'OS

La plateforme JAVA

- ▶ **CORBA** : fournit un ORB et le support de CORBA/IOP
- ▶ **XML (Extensible Markup Language)** : permet la gestion de données XML
- ▶ **Logging** : permet la création/gestion des fichiers de log
- ▶ **Beans** : permet la création de composants réutilisables, indépendants de la plate forme
- ▶ **Locale Support** : permet l'internationalisation des applications
- ▶ **Preferences** : permet la sauvegarde et la récupération des préférences et des données de configuration
- ▶ **Collections** : permet la gestion des collections (groupes d'objets)

La plateforme JAVA

- ▶ **JNI** : interface entre java et du code natif
- ▶ **Security** : API de sécurité, de signature, de cryptographie
- ▶ **Lang/Util** : API fondamentale avec des outils
- ▶ **New I/O** : API gérant les entrées/sorties avec gestion des buffers, etc...
- ▶ **Networking** : programmation réseau (sockets, URL, authentification, etc..)
- ▶ **Java HotSpot Client/Server Compiler** : java hotspot technologie pour les performances d'exécution
- ▶ **Java HotSpot VM Runtime** : idem

Pourquoi Java ?

- ☕ Indépendance de la plateforme (Windows, Linux, UNIX, MacOS...)
- ☕ Langage simple...
- ☕ Langage Objet
- ☕ Syntaxe proche du C/C++
- ☕ Bonne gestion des erreurs
- ☕ Pas de pointeurs...(apparents), de surcharge d'opérateur,
- ☕ Organisation des fichiers/classes (packages)
- ☕ Gestion facilitée de la mémoire
- ☕ Pas d'héritage multiple (cela existe en C++)
- ☕ Gratuit et opensource

Pourquoi Java ?

De nombreuses API (Applications Programming Interface) permettant :

| | |
|-------------------------------|-----------------------------------|
| Accès contrôlés aux fichiers | Programmation en 3D |
| Accès réseaux | Envoi/Réception de messages (JMS) |
| Accès bases de données (JDBC) | Vidéo |
| Graphisme | Sons |
| Événements | Cryptographie |
| Traitement images | Applications embarquées |
| Applications WEB | Mobilité (JME puis Android) |
| | ... |

Accès contrôlés aux fichiers
Accès réseaux
Accès bases de données (JDBC)
Graphisme
Événements
Traitement images
Applications WEB

Programmation en 3D
Envoi/Réception de messages (JMS)
Vidéo
Sons
Cryptographie
Applications embarquées
Mobilité (JME puis Android)
...

Inconvénients

- ▶ Gourmand en mémoire
 - ▶ Java est moins rapide qu'un langage compilé comme le C ou C++
- Il perd en rapidité ce qu'il gagne en portabilité

La Java Virtual Machine (JVM)

- ▶ Java = langage à la fois compilé et interprété

Première étape :

- ▶ code source compilé dans un langage "appelé **pseudo-code** ou **bytecode**"
- ▶ ce langage n'est pas du langage machine → incompréhensible par la machine

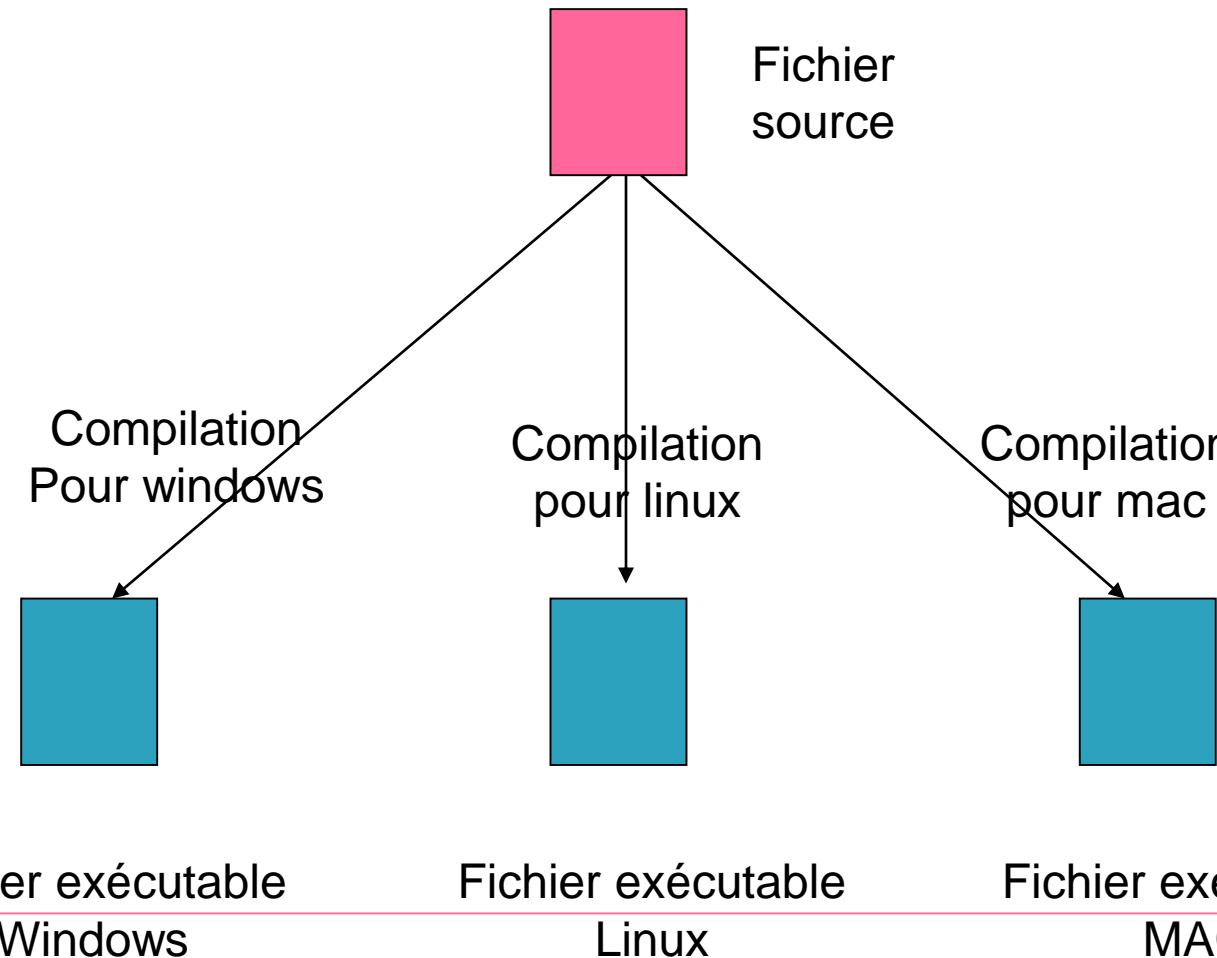
Seconde étape :

- ▶ lors de l'exécution, le bytecode est traduit en langage machine
- ▶ Pour que cela fonctionne, il faut avoir installé un programme sur la machine
- ▶ Ce programme s'appelle « Java virtual machine » Il est dépendant du matériel

Avantage :

- ▶ Le même Bytecode Java peut être exécuté sur n'importe quelle machine qui possède une machine virtuelle.

Les langages compilés

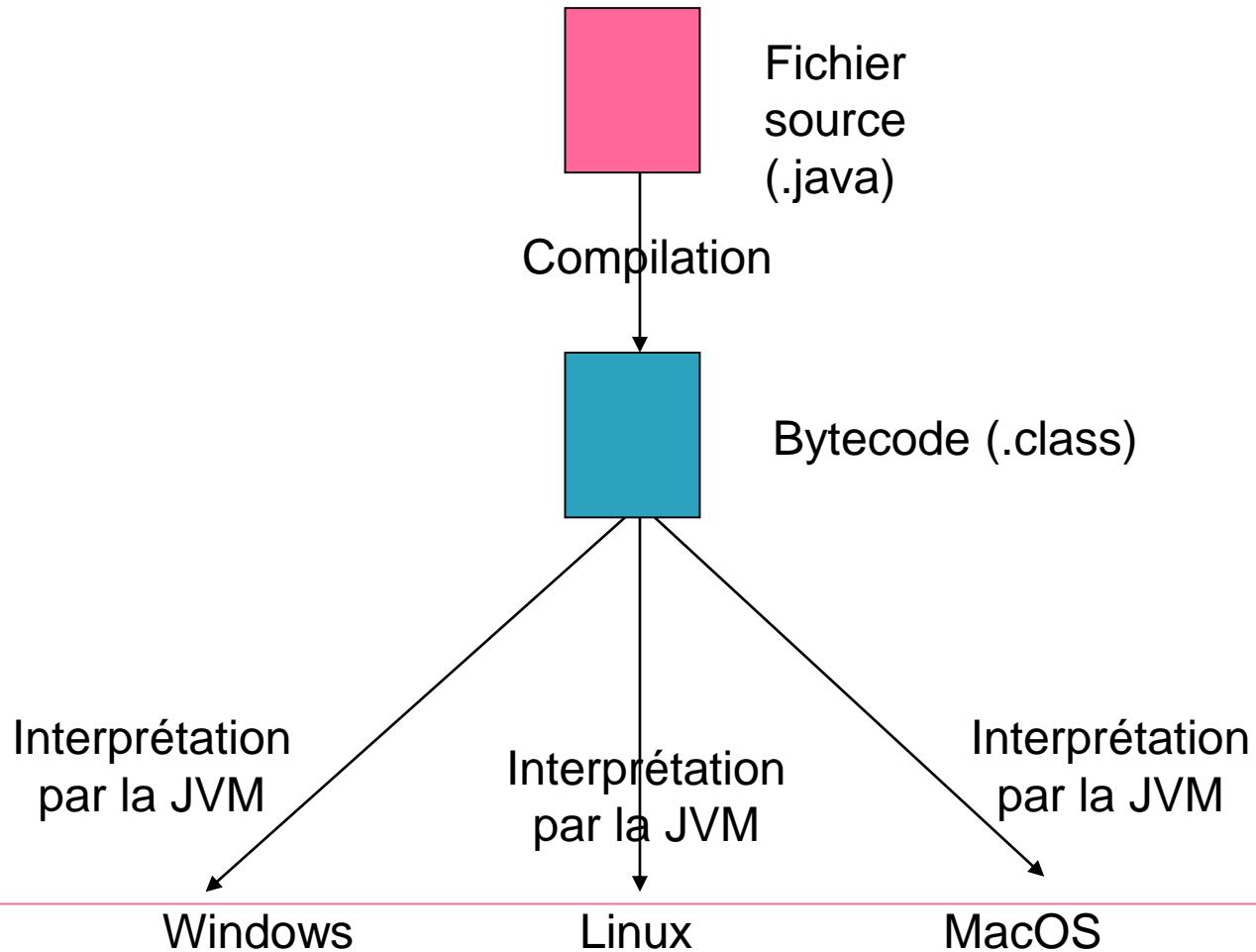


Fichier exécutable
Windows

Fichier exécutable
Linux

Fichier exécutable
MAC

Le langage Java



Premier programme java

```
// à écrire dans le fichier MonPremierProgramme.java

public class MonPremierProgramme {

    public static void main(String[] args) {

        System.out.println("Bonjour à tous");
    }

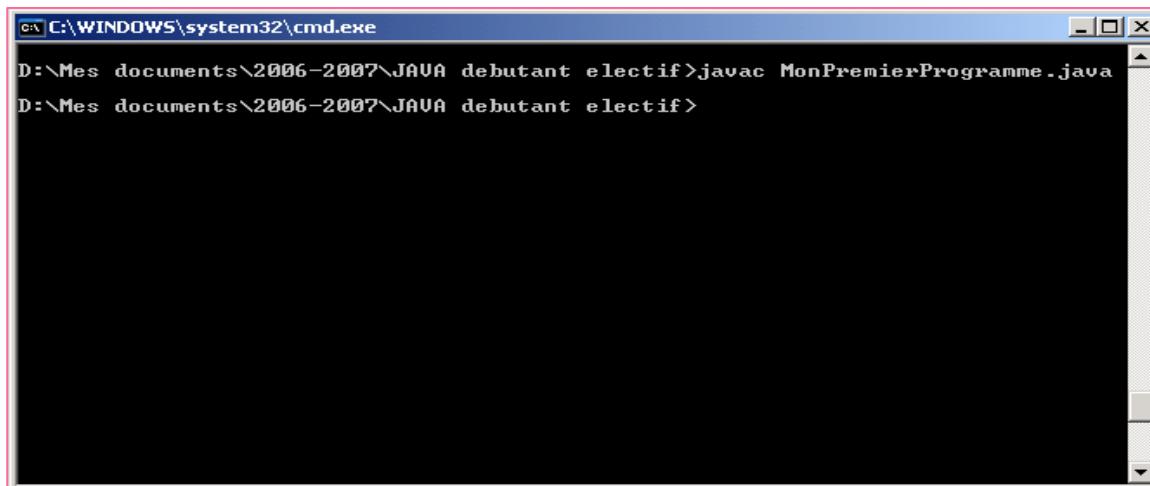
}
```

Comment compiler ?

▶ `javac MonPremierProgramme.java`

S'il n'y a pas d'erreur cela génère le fichier :

`MonPremierProgramme.class`

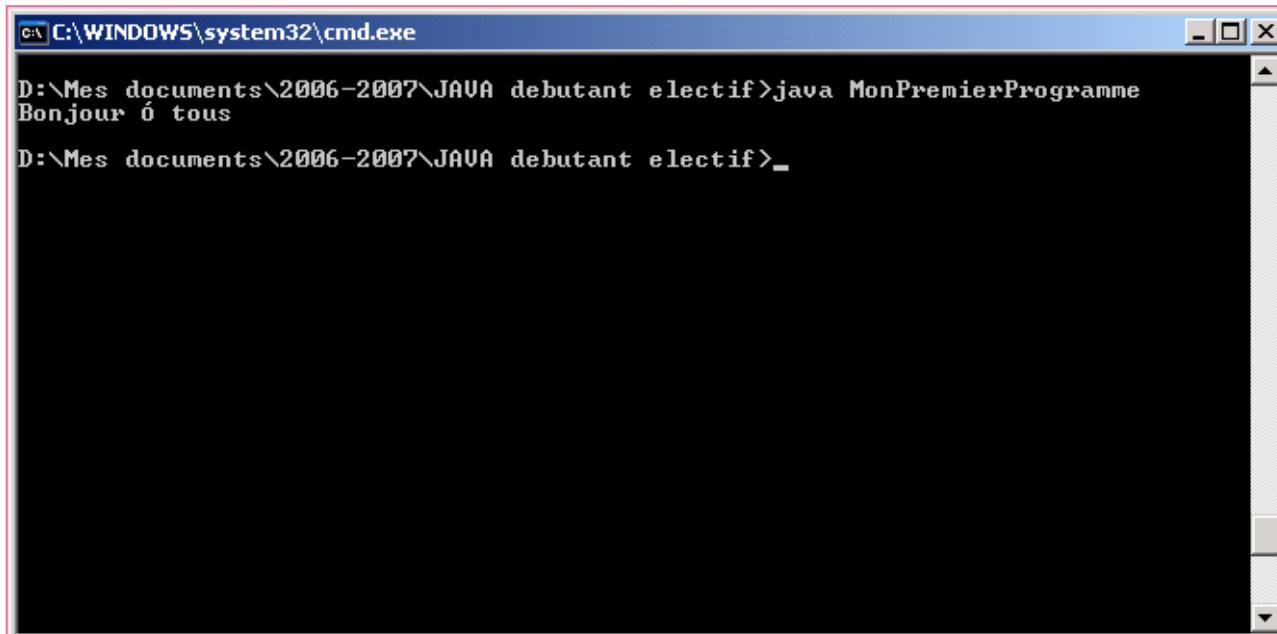


A screenshot of a Windows command prompt window titled "cmd C:\WINDOWS\system32\cmd.exe". The window shows the command "javac MonPremierProgramme.java" being run from the directory "D:\Mes documents\2006-2007\JAVA debutant electif". The output shows the command was successful, indicated by the prompt "D:\Mes documents\2006-2007\JAVA debutant electif>".

```
C:\WINDOWS\system32\cmd.exe
D:\Mes documents\2006-2007\JAVA debutant electif>javac MonPremierProgramme.java
D:\Mes documents\2006-2007\JAVA debutant electif>
```

Exécution

- ▶ `java MonPremierProgramme`
- ▶ → Affichage de "Bonjour à tous"



A screenshot of a Windows command prompt window titled "cmd C:\WINDOWS\system32\cmd.exe". The window shows the command "java MonPremierProgramme" being run from the directory "D:\Mes documents\2006-2007\JAVA debutant electif". The output of the program, "Bonjour à tous", is displayed below the command. The window has a standard Windows title bar and scroll bars.

```
C:\WINDOWS\system32\cmd.exe
D:\Mes documents\2006-2007\JAVA debutant electif>java MonPremierProgramme
Bonjour à tous
D:\Mes documents\2006-2007\JAVA debutant electif>_
```

Conventions de nommage

- ▶ Le nom des classes commencent par une **majuscule**
 - ▶ Le nom des méthodes commencent par une **minuscule**
 - ▶ Le nom des références d'objets commencent par une **minuscule**
- Et on ajoute une majuscule à chaque nouveau mot

Exemples :

Classe :
Méthode :
Référence d'objet :

MaPremiereClasse
maPremiereMethode
maPremiereReference

<http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Les packages

- ▶ Pour une meilleure organisation des fichiers de l'application, les classes java sont réparties dans des répertoires différents : les packages (ou paquetages)

exemples :

- ▶ le package `java.util` correspond au répertoire `java/util`
- ▶ le package `java.awt.event` correspond au répertoire `java/awt/event`

Les packages

- ▶ Pour utiliser des classes d'un autre package, il faut les importer au début du fichier :

Exemple :

```
import java.util.ArrayList;
import java.awt.*;

public class Test{
    ...
    ...
}
```

→ ici, on peut utiliser la classe ArrayList, ainsi que toutes les classes du package java.awt.

Les packages les plus utilisés

- ▶ **java.applet** : pour créer des applets
- ▶ **java.awt** : graphisme/dessin
- ▶ **java.awt.event**: gestion des évènements (clavier, souris,...)
- ▶ **java.io** : entrées/sorties (clavier, fichiers, flux,...)
- ▶ **java.lang** : classes de base (String, ...)
- ▶ **java.net** : réseau (Sockets,...)
- ▶ **java.sql** : connexion aux bases de données
- ▶ **java.util** : classes utilitaires (Date, Random,...)
- ▶ **javax.swing** : graphisme plus poussé
- ▶ ...

liste complète sur <http://download.oracle.com/javase/8/docs/api/>

La documentation Java de ORACLE

La documentation de java possède un format standard :

<http://download.oracle.com/javase/8/docs/api/>

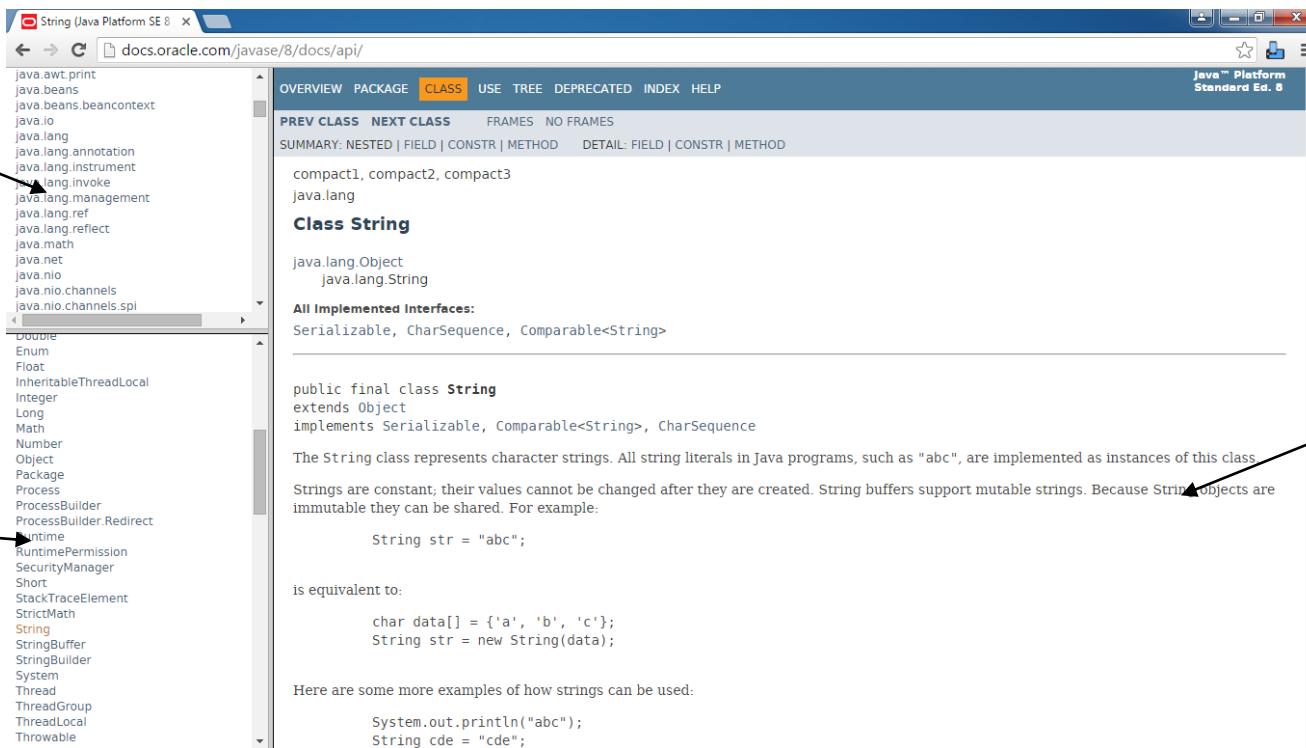
packages



classes



détail de la classe (attributs, méthodes)



OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3
java.lang

Class String

java.lang.Object
java.lang.String

All Implemented Interfaces:
Serializable, CharSequence, Comparable<String>

public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};  
String str = new String(data);
```

Here are some more examples of how strings can be used:

```
System.out.println("abc");  
String cde = "cde";
```

La documentation Java de ORACLE

- ▶ Exercice :

J'ai besoin de mettre en majuscule une chaîne de caractères.

Comment faire ?

- ▶ Remarque :

une chaîne de caractères en Java : `java.lang.String`

La documentation Java de SUN

- ▶ Réponse :

```
String maChaine="bonjour";  
  
maChaine=maChaine.toUpperCase();
```

```
System.out.println(maChaine);
```

→ BONJOUR

Les types de base

- ▶ On retrouve les même types qu'en C / C++

Les types de base

Les entiers :

| Type | Taille | Valeurs |
|--------------|----------|--------------------------|
| byte | 1 octet | -128 à 127 |
| short | 2 octets | -32768 à 32767 |
| int | 4 octets | -2147483648 à 2147483647 |
| long | 8 octets | -9 E 18 à +9 E 18 |

exemples :

```
int i=4;  
long l=12345;
```

Les types de base

Les réels :

| Type | Taille | Mantisse | Exposant |
|---------------|----------|----------|----------|
| float | 4 octets | 23 bits | 8 bits |
| double | 8 octets | 52 bits | 11 bits |

exemples :

```
double d=3.14;  
float f=2.4f;
```

Les types de base

Les caractères :

| Type | Taille |
|-------------|----------|
| char | 2 octets |

exemples :

```
char c1='A';
char c2='\n';
char c3='4';
char c4='é';
```

Les types de base

Les booléens :

boolean : vaut soit **true**, soit **false**

exemple :

```
boolean b=true;
```

Les instructions de contrôle

Les instructions de contrôle sont les mêmes qu'en C/C++

Les instructions de contrôle

SI → if

```
if(condition){  
    // instructions...  
}
```

ou

```
if(condition){  
    // instructions...  
}  
else {  
    // instructions...  
}
```

Les instructions de contrôle

Exemple :

```
if(note>=10){  
    System.out.println("Vous avez la moyenne");  
}  
else {  
    System.out.println("Vous n'avez pas la moyenne");  
}
```

Rq : Les accolades ne sont pas obligatoires car une seule instruction :

```
if(note>=10)  
    System.out.println("Vous avez la moyenne");  
else  
    System.out.println("Vous n'avez pas la moyenne");
```

Les instructions de contrôle

SELON QUE → switch

```
switch(variable){  
    case 1 :  
        // instructions...  
        break;  
    case 2 :  
        // instructions...  
        break;  
    case 3 :  
        // instructions...  
        break;  
    default :  
        // instructions...  
  
}
```

entier ou caractère seulement
(String est possible dans la
version JDK7)

Les instructions de contrôle

Exemple :

```
switch(choix){  
    case 1 :  
        System.out.println("Vous avez choisi le choix 1");  
        break;  
    case 2 :  
        System.out.println("Vous avez choisi le choix 2");  
        break;  
    default :  
        System.out.println("Mauvais choix");  
}
```

Les instructions de contrôle

Boucle POUR : for

```
for(initialisation; condition de continuité; itération){  
    //instructions  
}
```

Les instructions de contrôle

Exemple :

```
for(int i=0;i<10;i++){  
    System.out.println(i);  
}
```

ou encore :

```
for(int i=0;i<10;i++)  
    System.out.println(i);
```

Les instructions de contrôle

Boucle TANT QUE : while

```
while(condition de continuité){  
    //instructions...  
}
```

Les instructions de contrôle

Exemple :

```
while(valeur<10){  
    valeur++;  
}
```

ou encore :

```
while(valeur<10)  
    valeur++;
```

Les instructions de contrôle

Boucle FAIRE... TANT QUE : do ... while

```
do{  
    //instructions...  
}while( condition de continuité);
```

Les instructions de contrôle

Exemple :

```
do{  
    valeur++;  
}while(valeur<10);
```

ou encore :

```
do  
    valeur++;  
while(valeur<10);
```

Les instructions de contrôle

Comment choisir la bonne boucle ?

FOR, WHILE ou DO...WHILE ?

Les instructions de contrôle

SI le nombre de boucles est connu à l'avance :

FOR

SINON

SI la boucle doit être exécutée au moins une fois :

DO... WHILE

SINON

WHILE

instruction break

`break` permet de sortir d'un switch ou de quitter une boucle.

Exemple :

```
for( int i=0;i<10;i++){
    System.out.print(i+" ");
    if(i==5)
        break;
}
```

affichera : 0 1 2 3 4 5

Entrées/sorties

- ▶ L'objet associé à la sortie par défaut (écran) :
`System.out`
- ▶ L'objet associé à la sortie d'erreur (écran) : `System.err`
- ▶ L'objet associé à l'entrée par défaut (clavier) :
`System.in`

Entrées/sorties

- ▶ Pour afficher :

System.out.print(...)

System.out.println(...);

Depuis la version JDK5 :

System.out.printf(...);

- ▶ Exemples :

System.out.println("Bonjour");

System.out.printf("La valeur vaut %f",val);

Entrées/sorties

▶ Quelques caractères spéciaux :

\n : saut de ligne

\t : tabulation

\\" : backslash

\": guillemets

Entrées/sorties

- ▶ Pour lire des données au clavier :
- `java.util.Scanner`

Exemple :

```
Scanner clavier=new Scanner(System.in);
```

```
//lecture d'une chaîne  
String s=clavier.nextLine();
```

```
//lecture d'un réel  
float f=clavier.nextFloat();
```

```
//lecture d'un entier  
int i=clavier.nextInt();
```

Les Exceptions

- ▶ permettent d'intercepter les erreurs lors de l'exécution du programme et ainsi de proposer une solution adaptée

Les Exceptions

► Exemple :

```
try {  
    .....  
    .....  
}  
catch(UnknownHostException uhe){  
    //instructions exécutées si le serveur n'a pas été trouvé  
}  
catch(IOException ioe){  
    //instructions exécutées en cas d'erreur d'entrée/sortie  
}  
catch(Exception e){  
    //instructions exécutées si autre erreur  
}  
finally{  
    //instructions exécutées dans tous les cas  
}
```

Les Exceptions

Remarques :

- ▶ possibilité de mettre plusieurs `catch` à la suite
- ▶ un seul `catch` sera exécuté en cas d'erreur
- ▶ la clause `finally` sera exécutée dans tous les cas
- ▶ la clause `finally` est facultative

Les Exceptions

Exemple :

```
public void maMethode() {  
  
    int res=5/0;  
}
```

→Exception in thread "main" java.lang.ArithmetricException: / by zero
at MaClasse.main(MaClasse.java:10)

Les Exceptions

▶ Solution :

```
public static void main(String[] args) {  
    int res;  
    try{  
        res=5/0;  
    }catch(ArithmeticException ae){  
        System.out.println("Erreur de calcul");  
    }  
}
```

→ Erreur de calcul

Les Exceptions

Intérêt du finally :

```
try{
    throw new Exception(); //on génère une exception
}catch(Exception e){
    return ; //pour sortir de la méthode
}finally{
    System.out.println("finally");
}
```

→ Le programme **exécutera** le catch ET la clause finally !

Java Objet ?

- ▶ java n'est pas 100% objet à cause des types primitifs.
- ▶ Il existe cependant des classes enveloppes ([wrappers](#)) qui permettent de manipuler ces types comme des objets :
 - Boolean
 - Char
 - Byte
 - Short
 - Integer
 - Long
 - Float
 - Double
- ▶ package `java.lang`
- ▶ [exemple](#) : `Double reel=new Double(3.14);`

Classes/objets

Rappels :

- ▶ classe : moule qui permettra de fabriquer des objets
- ▶ objet : un objet est fabriqué à l'aide d'une classe.

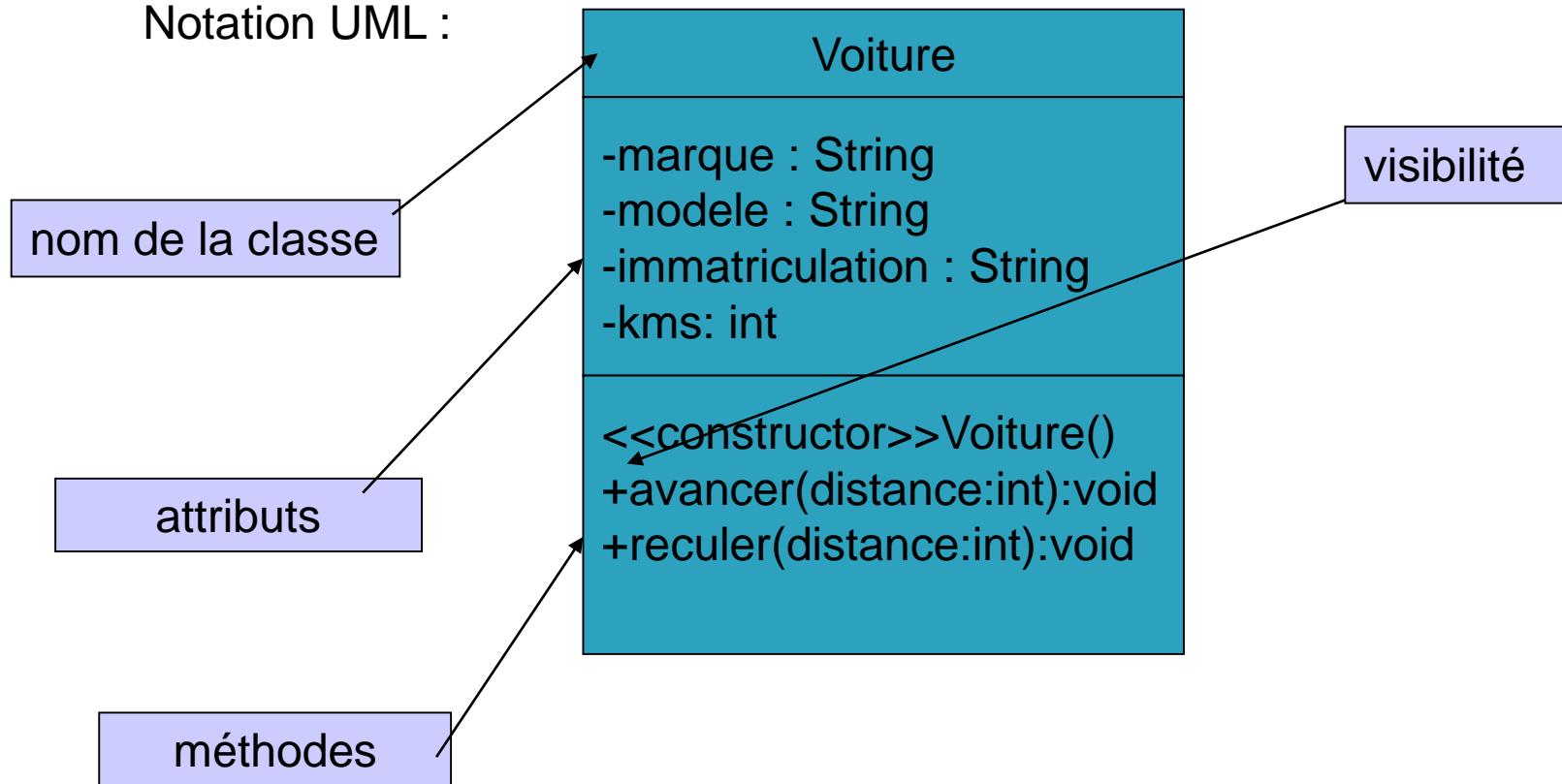
exemple :

La classe Voiture permet de fabriquer les objets suivants:

- ▶ la renault zoe d'immatriculation "AA-123-ZZ"
- ▶ la "peugeot 308" d'immatriculation "BB-444-CC"

Classes/objets

Notation UML :



Classes/objets

Traduction en Java :

| Voiture |
|--|
| -marque : String -modele : String -immatriculation : String -kms: int |
| <<constructor>> Voiture(^{marque:String,modele:S} avancer(distance:int):void reculer(distance:int):void |

```
public class Voiture{  
  
    private String marque;  
    private String modele;  
    private String immatriculation;  
    private int kms;  
  
    public Voiture(String marque,String modele,String  
    immatriculation){  
        this.marque=marque;  
        this.modele=modele;  
        this.immatriculation=immatriculation;  
        this.kms=0;  
    }  
  
    public void avancer(int distance){  
        kms=kms+distance;  
    }  
  
    public void reculer(int distance){  
        kms=kms-distance;  
    }  
}
```

visibilité

Visibilité des attributs/méthodes :

| | | Access Levels | | | | |
|-----|--------------------|---------------|-------|---------|----------|-------|
| UML | | Modifier | Class | Package | Subclass | World |
| + | public | Y | Y | Y | Y | |
| # | protected | Y | Y | Y | N | |
| - | <i>no modifier</i> | Y | Y | N | N | |
| - | private | Y | N | N | N | |

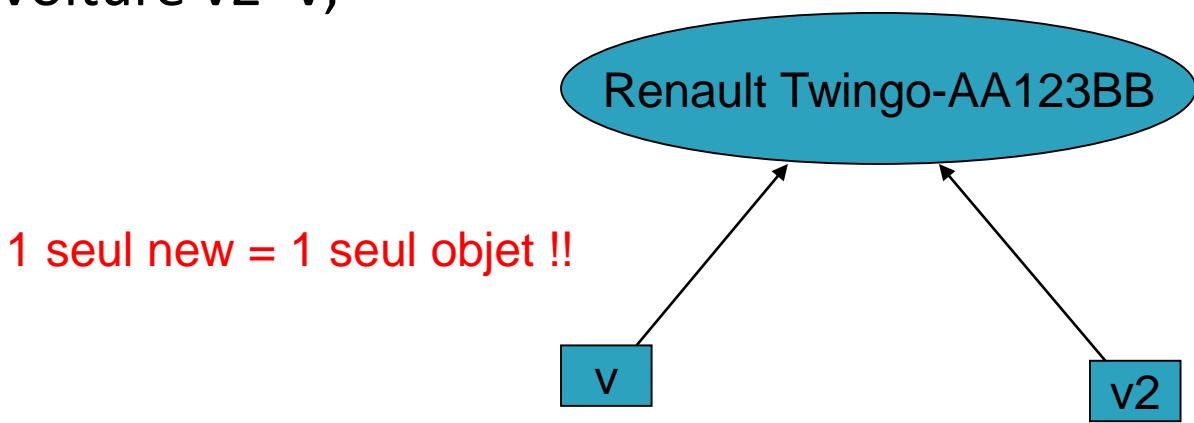
Instanciation d'une classe

Pour créer un objet : **new**

exemple :

```
Voiture v=new Voiture("Renault", "Twingo", "AA123BB");
```

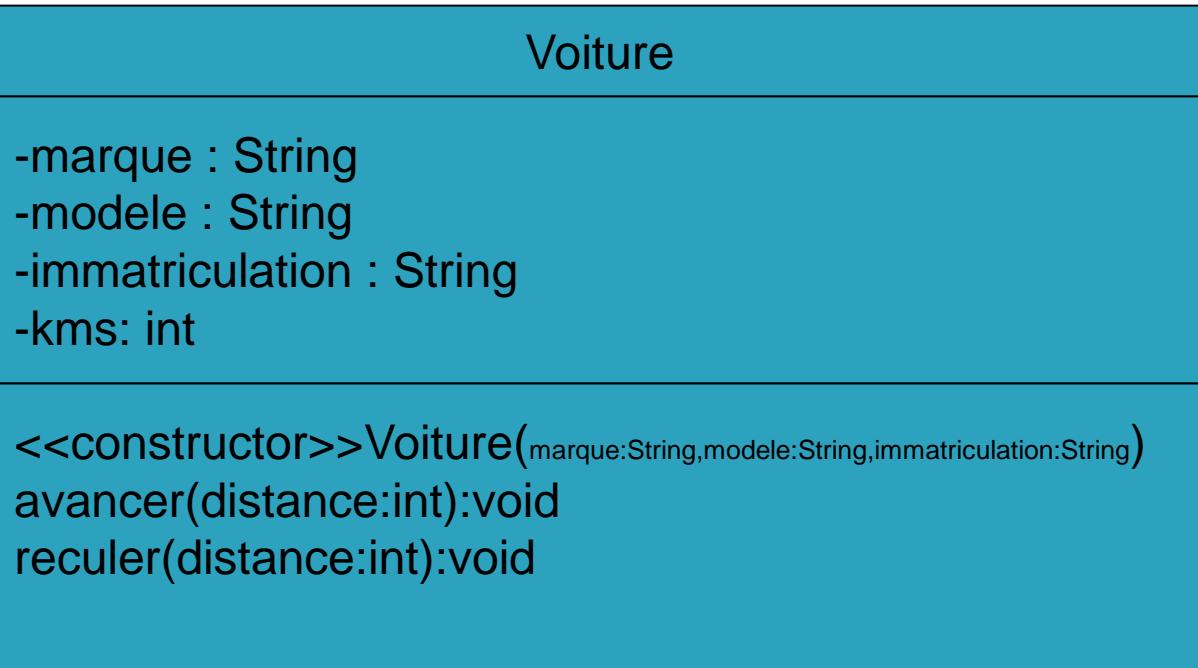
```
Voiture v2=v;
```



Appel de méthodes

```
Voiture v=new Voiture("Renault", "Twingo", "AA-123-BB");
```

```
v.avancer(100);  
v.reculer(50);
```

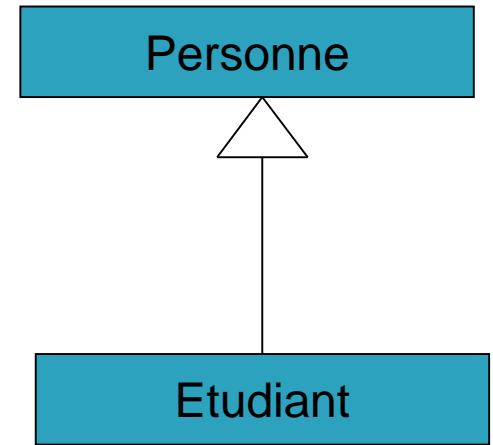


L'héritage

- ▶ pas d'héritage multiple en java
(contrairement à C++)

```
class Etudiant extends Personne {  
...  
}
```

- ▶ Toute classe dérive directement ou non de la classe `java.lang.Object`
- ▶ Possibilité de redéfinir des méthodes

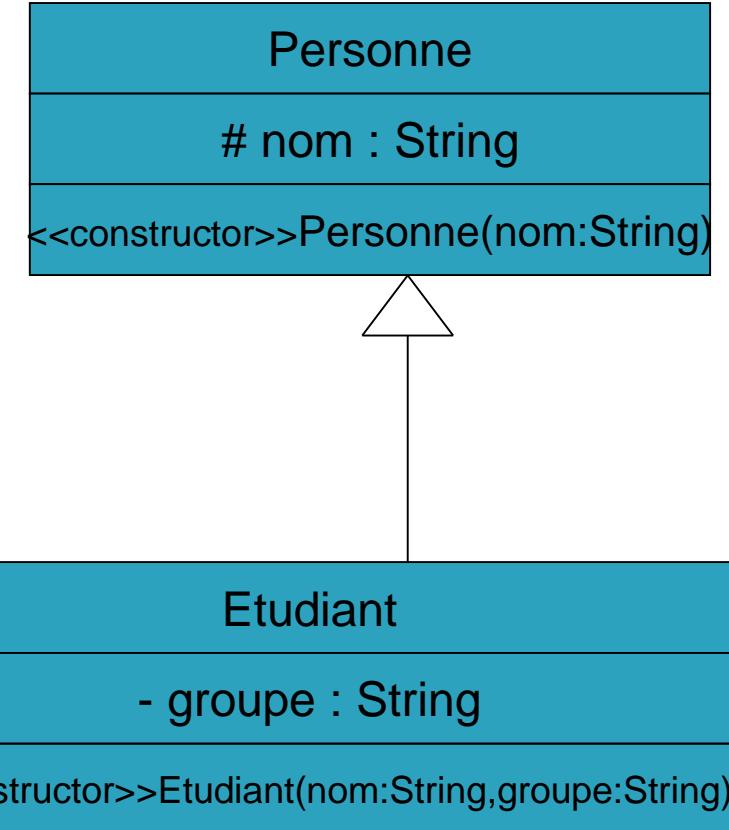


L'héritage

- ▶ `this` : représente l'objet courant
- ▶ `super(..)` permet de faire appel au constructeur de la classe mère

Exemple :

```
public class Etudiant extends Personne {  
  
    private String groupe;  
  
    public Etudiant(String nom, String groupe) {  
        super(nom);  
        this.groupe=groupe;  
    }  
}
```



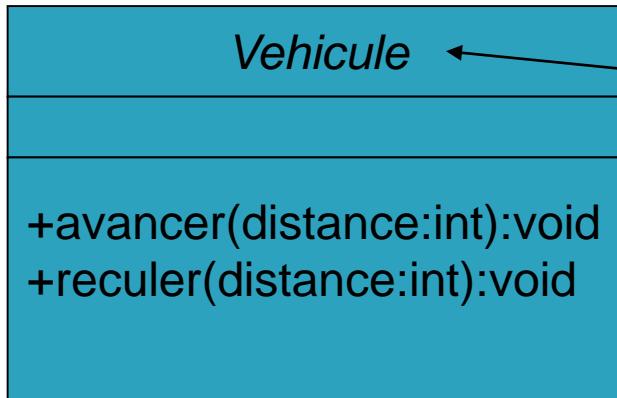
Les interfaces

- ▶ servent à spécifier le comportement d'une classe sans mettre en œuvre celui-ci
- ▶ ne contiennent que des en-têtes de méthodes
- ▶ aucune instruction
- ▶ pas d'attributs (sauf les static final)

Les interfaces



exemple :



(en italique)

Toutes les classes qui implémenteront cette interface devront posséder au minimum les 2 méthodes avancer et reculer.

Sinon le programme ne compilera pas.

On est donc certain que tous ces véhicules pourront avancer et reculer.

Les interfaces

☕ exemple :

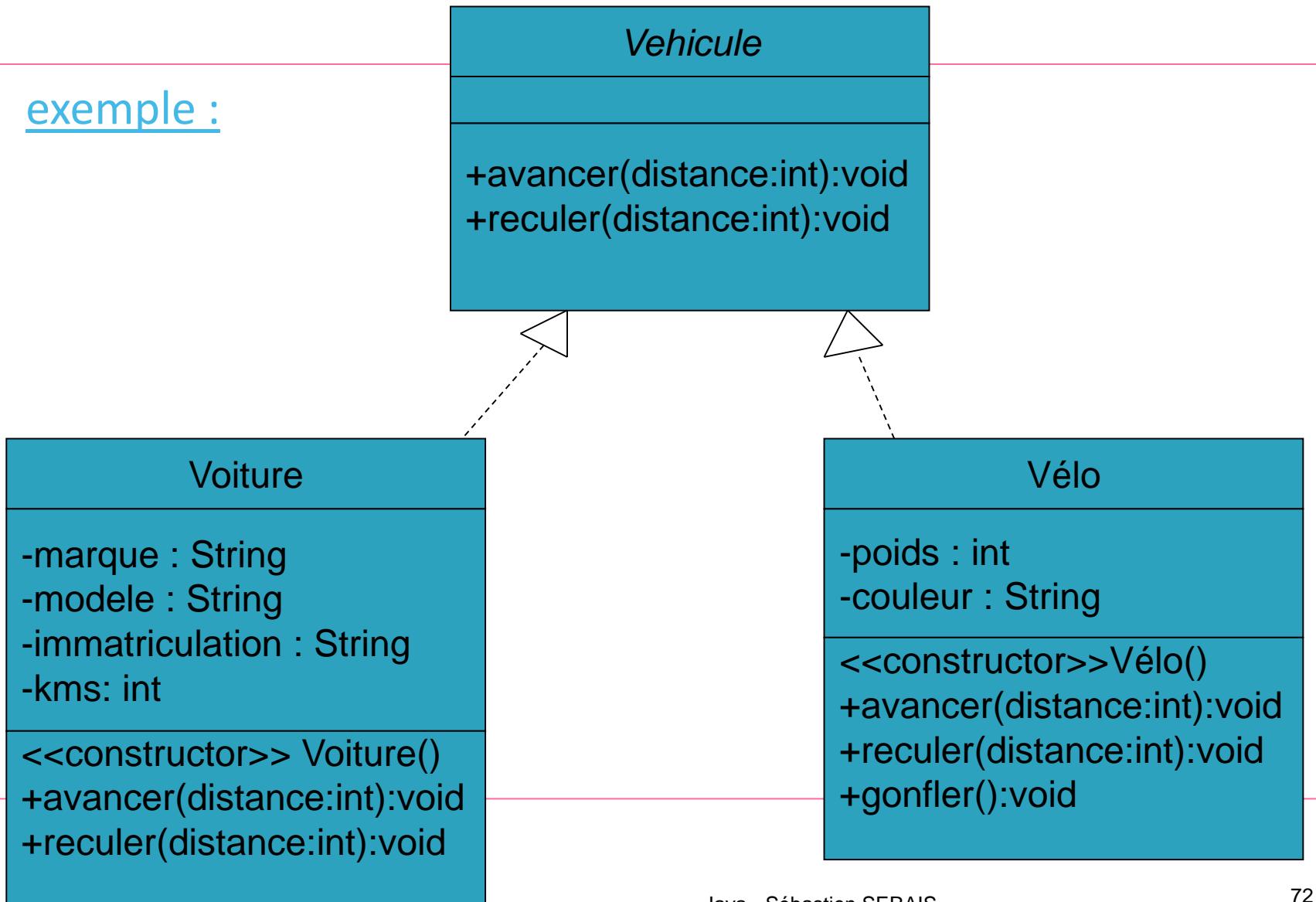
Vehicule

+avancer(distance:int):void
+reculer(distance:int):void

```
interface Vehicule {  
  
    public void avancer(int distance);  
  
    public void reculer(int distance);  
  
}
```

Les interfaces

exemple :



Les interfaces

```
public class Voiture implements Vehicule {  
  
    private String marque;  
    private String modele;  
    private String immatriculation;  
    private int kms;  
  
    public Voiture( String marque,String modele,String immatriculation){  
        this.marque=marque;  
        this.modele=modele;  
        this.immatriculation=immatriculation;  
        this.kms=0;  
    }  
  
    public void avancer(int distance){  
        kms=kms+distance;  
    }  
  
    public void reculer(int distance){  
        kms=kms-distance;  
    }  
}
```

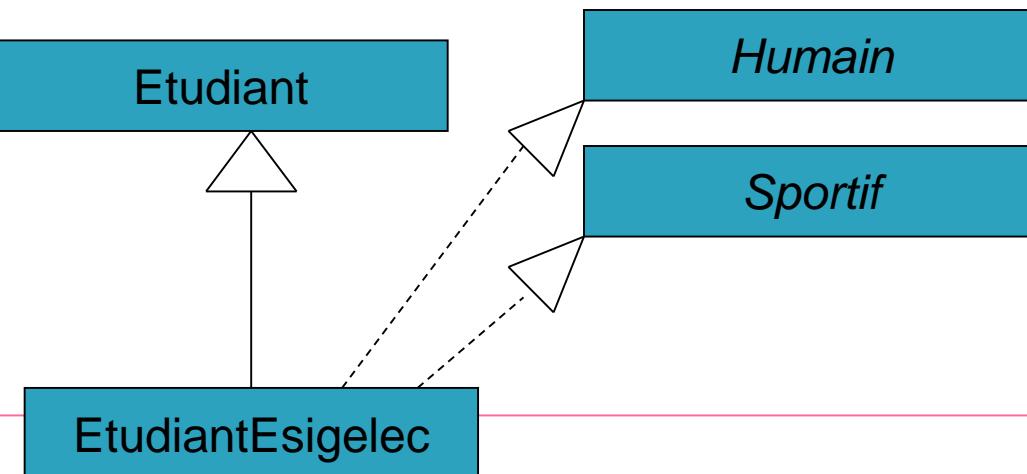
Les interfaces

Remarque :

- ☛ une classe peut dériver d'une seule classe
- ☛ une classe peut implémenter plusieurs interfaces

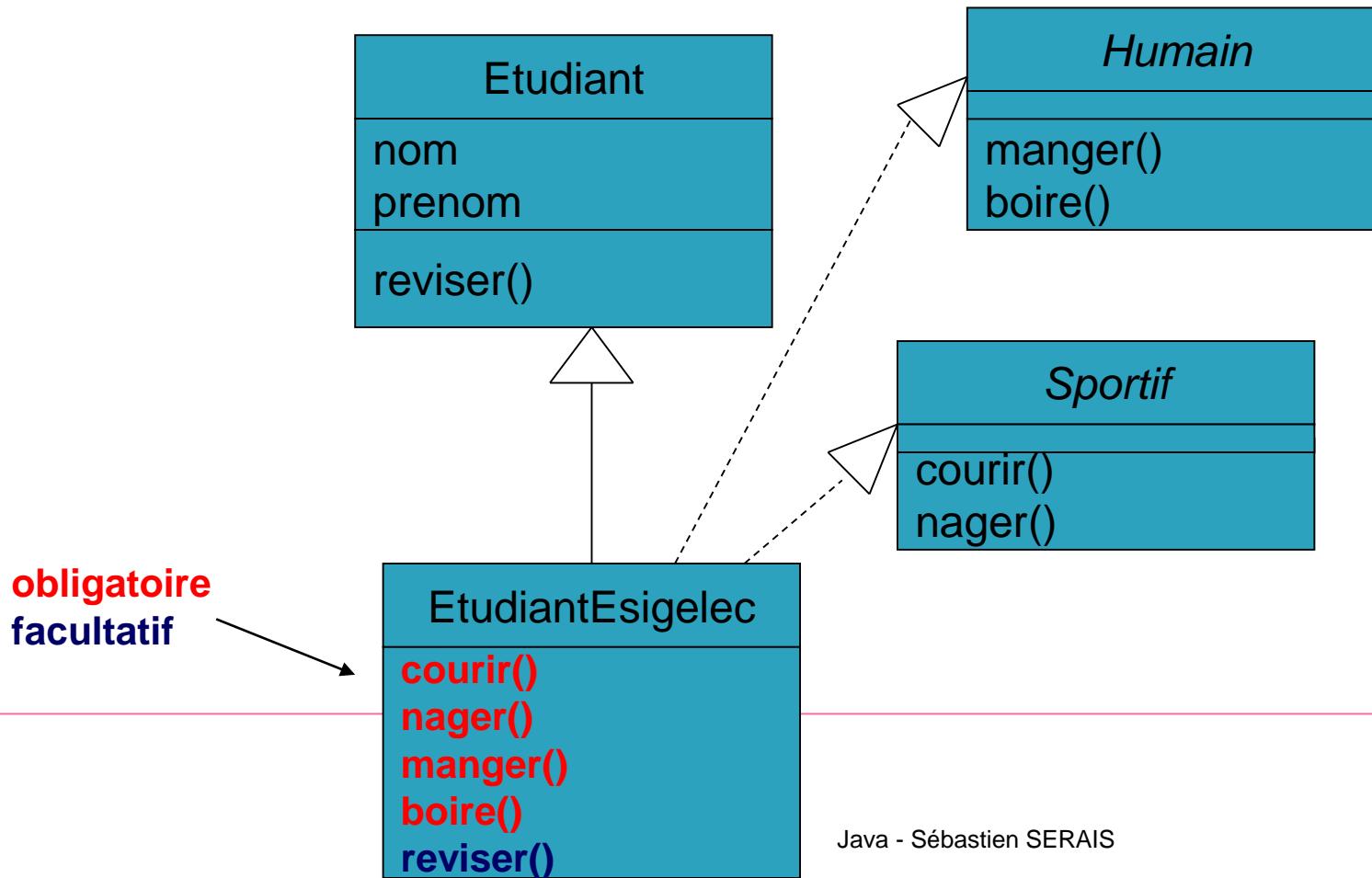
```
public class EtudiantEsigelec extends Etudiant implements Humain, Sportif{
```

```
...  
}
```



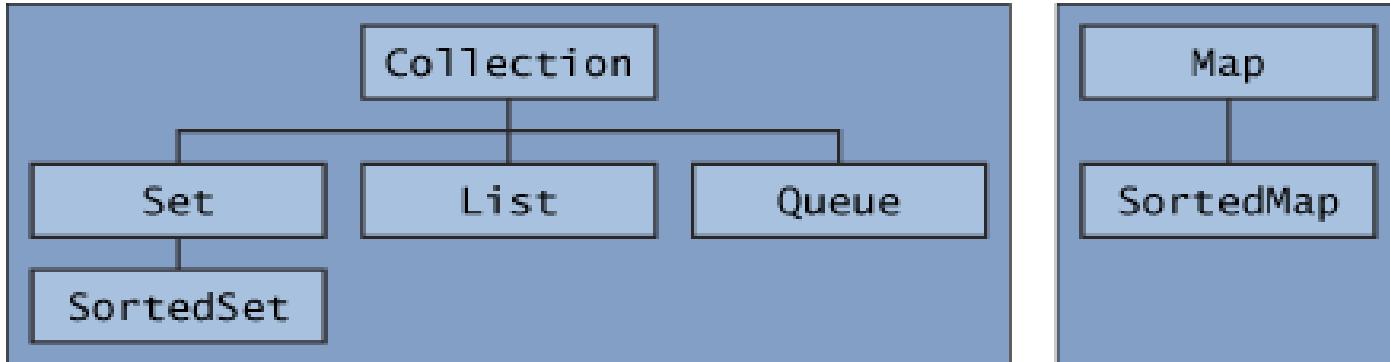
Les interfaces

Un EtudiantEsigelec **est un** Etudiant et il possède en plus le **comportement** d'un humain et d'un sportif



Les Collections/Maps

Les interfaces associées :



source image : <http://java.sun.com>

Set : ensemble d'éléments sans doublon

List : liste d'éléments ordonnée (peut contenir des doublons)

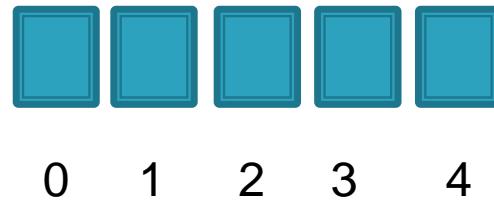
Queue : idem List mais avec FIFO

Map : ensemble basé sur le principe clé/valeur (type annuaire)

Les listes

`java.util.ArrayList (List)`

`add(Object)`
`remove (int index)`
`get(int index) : Object`
`size() : int`
...



Les listes

Exemple d'utilisation de la classe ArrayList :

```
ArrayList<String> al=new ArrayList<String>();  
al.add("Hugues");  
al.add("Germain");  
al.add("Anthony");  
System.out.println(al);
```

→ [Hugues, Germain, Anthony]

La méthode `toString()` est automatiquement appelée,
cela est équivalent à écrire `al.toString()`

Les listes

Parcours d'un ArrayList (méthode 1) :

```
ArrayList<String> al=new ArrayList<String>();  
  
al.add("Hugues");  
al.add("Germain");  
al.add("Anthony");  
  
for (int i=0;i<al.size();i++){  
    System.out.println(al.get(i));  
}
```

Les listes

Parcours d'un ArrayList (méthode 2) (depuis JDK5):

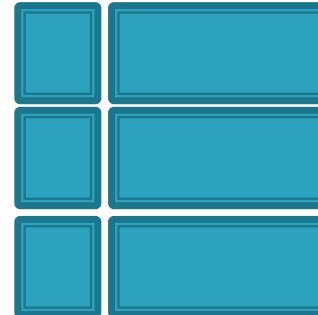
```
ArrayList<String> al=new ArrayList<String>();  
  
al.add("Hugues");  
al.add("Germain");  
al.add("Anthony");  
  
for (String s : al){  
    System.out.println(s);  
}
```

Les Maps

java.util.HashMap (Map)

get(Object key) : Object
put(K key, V value)
remove(Object key)
size() : int
...

key value



Les Maps

Exemple avec un HashMap:

```
HashMap<Integer, String> hm=new HashMap<Integer, String>();  
  
hm.put(1,"Hugues");  
hm.put(2,"Germain");  
hm.put(5,"Anthony");  
hm.put(2,"Romain");  
  
//parcours des entrées de la Map  
for(Map.Entry<Integer, String> ligne:hm.entrySet()){  
    System.out.println(ligne.getKey()+" = "+ligne.getValue());  
}
```

Affichera : 1 = Hugues
 2 = Romain
 5 = Anthony

Les Maps

Possibilité de créer une Map triée selon ses clés :

Avec la classe : `TreeMap<K,V>`

Graphisme

Le Graphisme en Java

Graphisme

AWT Abstract Window Toolkit

- Première librairie graphique proposée par SUN
- Peu de composants

SWING

- Consomme plus de mémoire
- Beaucoup de composants

AWT

Classes principales d'AWT :

- ☀ **Container** : gestionnaire de composants
- ☀ **Window** : fenêtre sans bordure
- ☀ **Frame** : fenêtre avec bordure
- ☀ **Dialog** : boîte de dialogue
- ☀ **Panel** : permet de déposer des composants
- ☀ **Canvas** : feuille de dessin

AWT

- ☀ **Button** : bouton
- ☀ **Checkbox** : case à cocher
- ☀ **Choice** : Combo box
- ☀ **Label** : Text
- ☀ **List** : Liste de sélection
- ☀ **TextField** : zone de texte
- ☀ **MenuBar** : barre de menu
- ☀ **ScrollBar** :barres de défilement
- ☀ ...

SWING

Classes principales de SWING :

- ☀ **JWindow** : fenêtre sans bordure
- ☀ **JFrame** : fenêtre avec bordure
- ☀ **JDialog** : boîte de dialogue
- ☀ **JPanel** : permet de déposer des composants

SWING

- ☀ **JButton** : bouton
- ☀ **JCheckbox** : case à cocher
- ☀ **JApplet** : applet
- ☀ **JColorChooser** : permet de sélectionner une couleur
- ☀ **JComboBox**
- ☀ **JFileChooser** : permet de sélectionner un fichier
- ☀ **JLabel** : label
- ☀ **JList** : liste de sélection
- ☀ **JMenuBar** : barre de menu

SWING

- ☀ **JMenuItem** : item de menu
- ☀ **JOptionPane** : pour des boites de dialogue
- ☀ **JProgressBar** : barre de progression
- ☀ **JRadioButton** : bouton radio
- ☀ **JScrollBar** : barre de défilement
- ☀ **JScrollPane** : composant avec scrollbar
- ☀ **JTable** : table de données
- ☀ **JTextArea** : zone de texte (plusieurs lignes)
- ☀ **JTextField** : zone de texte (1 ligne)
- ☀ **JTree** : arborescence

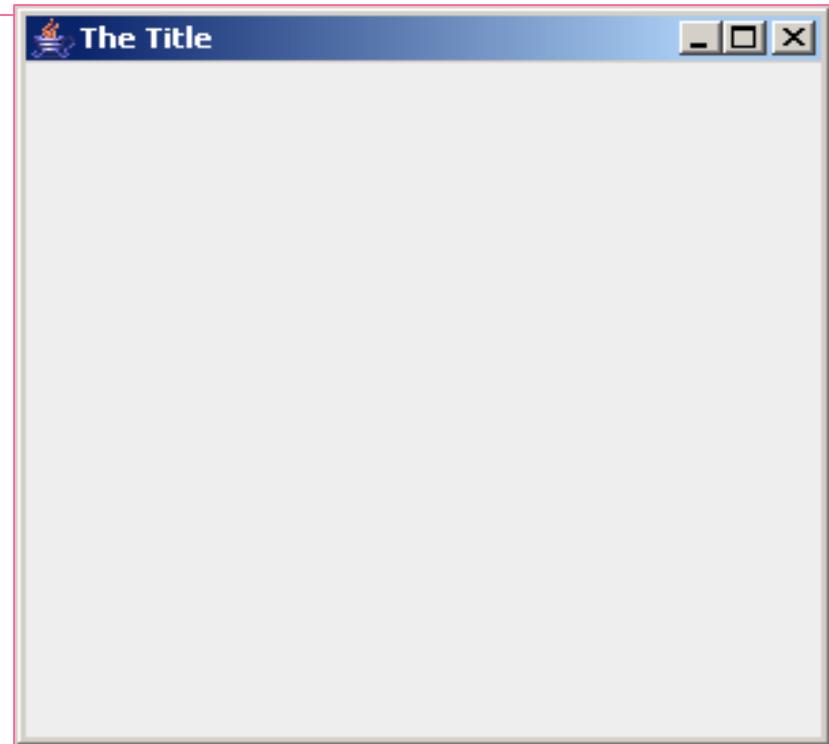
Swing

```
import javax.swing.JFrame;

public class MaFenetre extends JFrame {

    public MaFenetre(){
        super();
        setTitle("The Title");
        setSize(300,300);
        setVisible(true);
    }

    public static void main(String[] args){
        new MaFenetre();
    }
}
```



Layout Managers

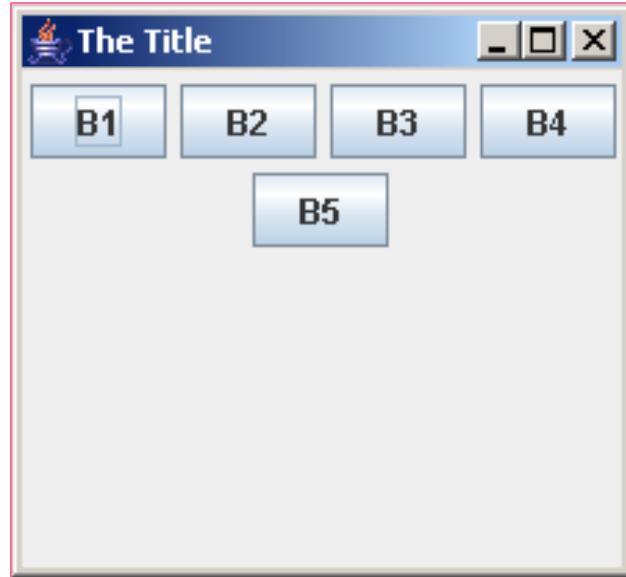
- ✳ BorderLayout
- ✳ GridLayout
- ✳ FlowLayout
- ✳ CardLayout
- ✳ BoxLayout
- ✳ GridBagLayout
- ✳ null

Swing

- ✿ **FlowLayout**

```
setLayout(new FlowLayout());
```

```
add(new JButton("B1"));  
add(new JButton("B2"));  
add(new JButton("B3"));  
add(new JButton("B4"));  
add(new JButton("B5"));
```

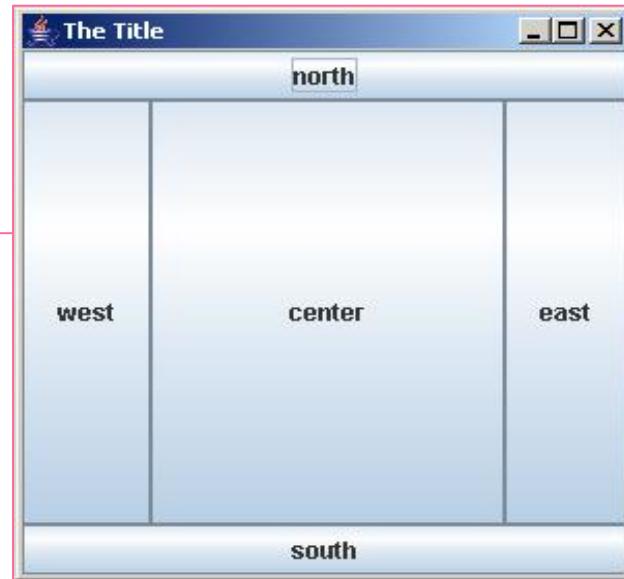


Swing

* BorderLayout

```
setLayout(new BorderLayout());
```

```
add(new JButton("north"),BorderLayout.NORTH);
add(new JButton("south"),BorderLayout.SOUTH);
add(new JButton("east"),BorderLayout.EAST);
add(new JButton("west"),BorderLayout.WEST);
add(new JButton("center"),BorderLayout.CENTER);
```

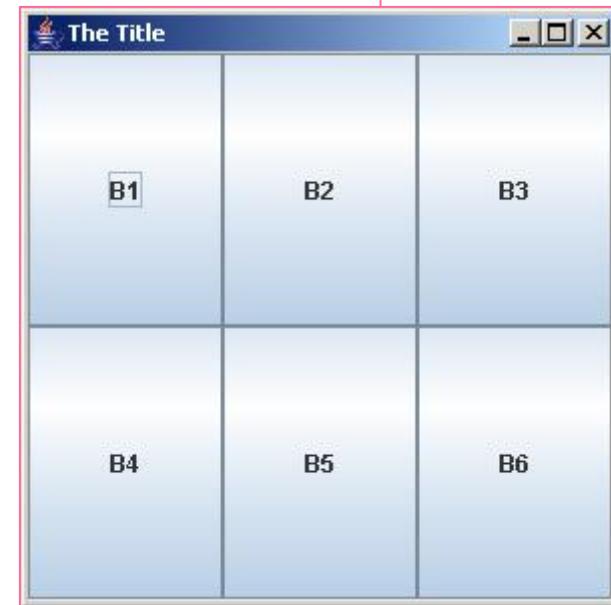


Swing

✳ GridLayout

```
setLayout(new GridLayout(2,3));
```

```
add(new JButton("B1"));
add(new JButton("B2"));
add(new JButton("B3"));
add(new JButton("B4"));
add(new JButton("B5"));
add(new JButton("B6"));
```

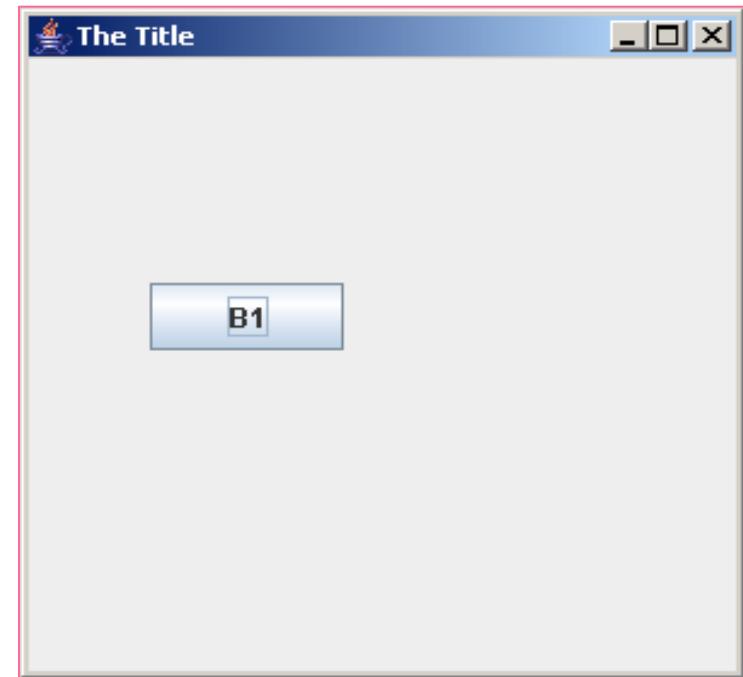


Swing

- ✿ **null**

```
setLayout(null);
```

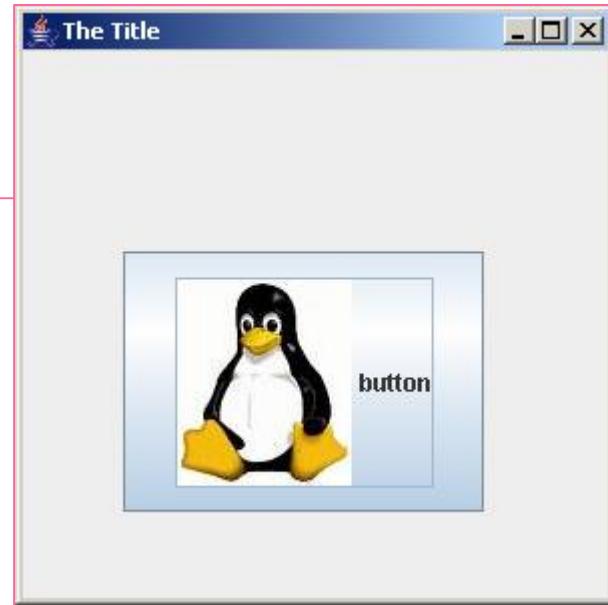
```
JButton b=new JButton("B1");  
b.setBounds(50,100,80,30);  
add(b);
```



Swing

Pour ajouter une icône sur un bouton :

```
JButton b=new JButton("button");
ImageIcon ima=new ImageIcon("c:\\tux.jpg");
b.setIcon(ima);
b.setBounds(50,100,180,130);
add(b);
```



Swing

ATTENTION : ne pas mettre de chemin absolu !!

ImageIcon ima=new

ImageIcon(Toolkit.getDefaultTollkit().getImage("./tux.png"));

--> l'image "tux.png" doit se trouver à la racine du projet

Swing

Problème :

Par défaut, lorsque l'on ferme la fenêtre, l'application continue de s'exécuter.

Pour quitter l'application lors de la fermeture de la fenêtre :

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Dessiner

★ DESSIN bas niveau

Pour dessiner sur un composant, il faut récupérer une référence sur l'objet "graphics" associé avec la méthode : Graphics
getGraphics()

[Exemple:](#)

```
Graphics g=monCanvas.getGraphics();
```

Dessiner

Principales méthodes de la classe Graphics :

- ▶ drawLine(int x1,int y1,int x2,int y2);
- ▶ drawOval(...);
- ▶ drawPolygon(...);
- ▶ drawRect(...);
- ▶ drawString(...);
- ▶ fillOval(...);
- ▶ fillPolygon(...);
- ▶ fillRect(...);
- ▶ getColor();
- ▶ setColor(java.awt.Color couleur);
- ▶ setFont(...);

Dessiner

Exemple :

Pour dessiner un disque rouge :

```
Graphics g=monCanvas.getGraphics();
g.setColor(Color.RED);
g.fillOval(20,20,30,30); //rayon=30
```

Dessiner

Remarque :

La méthode **public void paint(Graphics g)** est appelée automatiquement dès que le composant a besoin de se redessiner.

Il faut donc placer les méthodes de dessin à l'intérieur de cette méthode.

```
public void paint(Graphics g) {  
    super.paint(g);  
    g.setColor(Color.RED);  
    g.fillOval(20,20,30,30);  
}
```

Dessiner avec double Buffering

Principe : on dessine sur un objet Image en mémoire

```
//création de l'image en mémoire  
Image image = new BufferedImage(500,500, BufferedImage.TYPE_INT_ARGB);  
//recuperation du graphics de l'image mémoire  
graphics = image.getGraphics();
```

```
// la méthode paint du composant est appelée automatiquement  
// elle dessine l'image en mémoire sur le composant  
public void paint(Graphics g) {  
    g.drawImage(image, 0, 0, this);  
}
```

Événements

Pour ajouter des écouteurs d'événements, utiliser les interfaces correspondantes :

- ☀ ActionListener
- ☀ KeyListener
- ☀ FocusListener
- ☀ MouseListener
- ☀ MouseMotionListener
- ☀ WindowListener

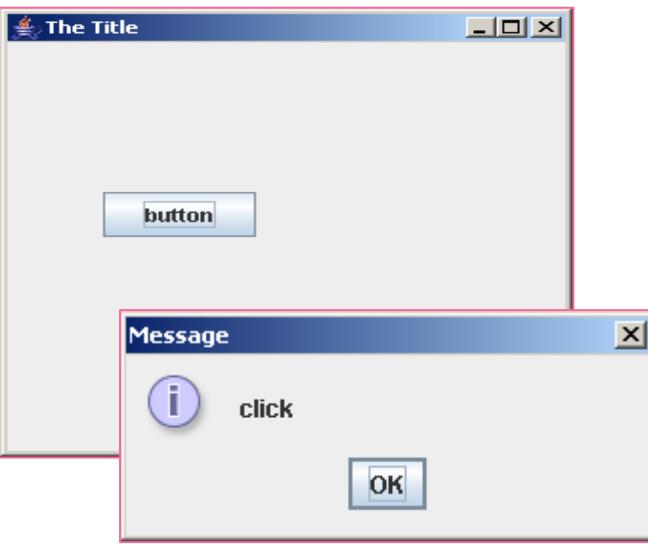
Ces interfaces se trouvent dans le package : `java.awt.event`

Exemple

Par exemple : pour ajouter un événement "click" sur un bouton :

- ★ implémenter l'interface ActionListener
- ★ ajouter un ActionListener au bouton
- ★ définir les méthodes de l'interface ActionListener

Exemple



```
import javax.swing.*;
import java.awt.event.*;

public class MaFenetre extends JFrame implements ActionListener {
    JPanel jPanel;
    JButton b;
    public MaFenetre(){
        super();
        setTitle("The Title");
        setSize(300,300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jPanel=new JPanel();
        jPanel.setLayout(null);
        b=new JButton("button");
        b.setBounds(50,100,80,30);
        b.addActionListener(this);
        jPanel.add(b);
        setContentPane(jPanel);
        setVisible(true);
    }

    public static void main(String[] args){
        new MaFenetre();
    }

    public void actionPerformed(ActionEvent ae) {
        // ouverture d'une boite de dialogue
        JOptionPane.showMessageDialog(null,"click");
    }
}
```

Exemple

Si on a plusieurs boutons,
comment savoir quel bouton a été cliqué ?

Méthode 1 :

Utiliser la méthode getSource() qui retourne l'objet qui a lancé l'événement.

Exemple :

```
public void actionPerformed(ActionEvent ae) {  
  
    if(ae.getSource()==b){  
  
        //code exécuté si clic sur bouton b1  
    }  
    else if(ae.getSource()==b2){  
  
        //code exécuté si clic sur bouton b2  
  
    }  
}
```

Exemple

Si on a plusieurs boutons,
comment savoir quel bouton a été cliqué ?

Méthode 2 :

Utiliser la méthode `getActionCommand()` pour récupérer l'action associée

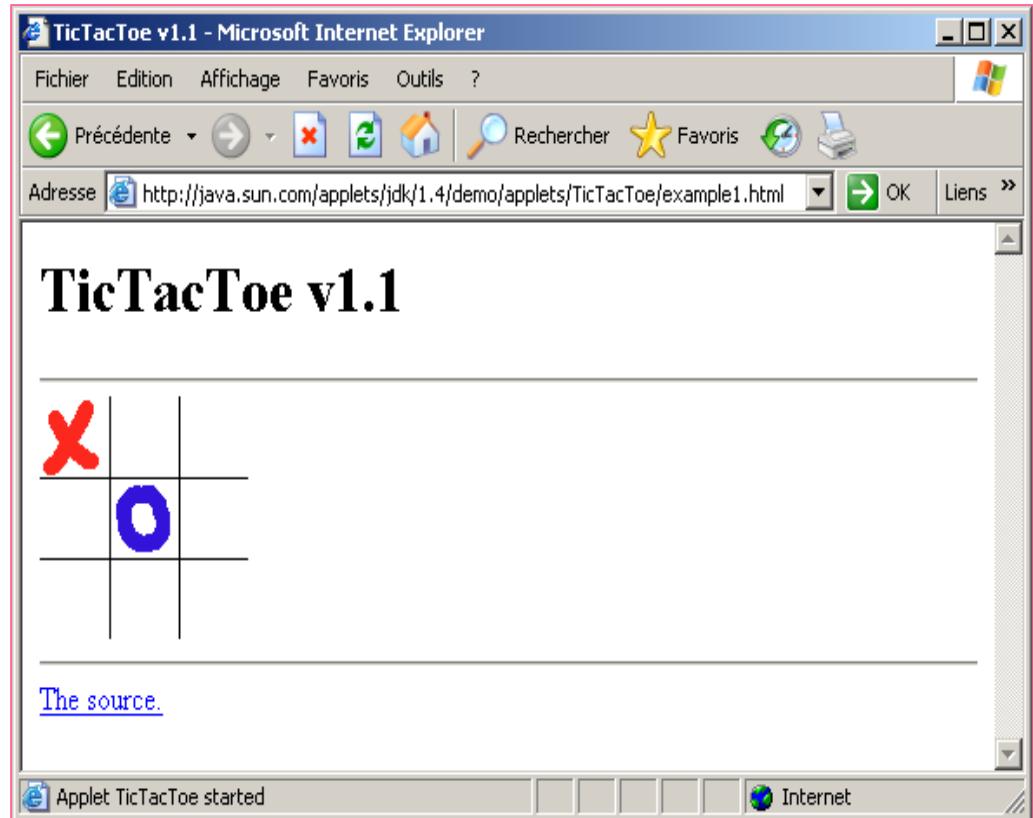
Exemple :

```
public void actionPerformed(ActionEvent ae) {  
  
    if(ae.getActionCommand().equals("afficherAide")){  
  
        //code exécuté si la commande est : afficherAide  
    }  
  
}
```

pour affecter une commande à un composant :
`setActionCommand("nomCommande");`

Les Applets

Applet = application
java intégrée à une
page web (HTML)



Les Applets

Elle s'exécute dans un navigateur web

Ou à l'aide de l'utilitaire **appletviewer**

Les Applets

Quelques exemples d'applets :

<http://java.sun.com/applets/jdk/1.4/index.html>

Les Applets

Avec AWT :

Une fenêtre est une classe qui hérite de la classe Frame

Une applet est une classe qui hérite de la classe Applet

```
import java.applet.*;
public class MonApplet extends Applet {
}
```

Les Applets

Avec SWING :

Une fenêtre est une classe qui hérite de la classe JFrame

Une applet est une classe qui hérite de la classe JApplet

```
import javax.swing.*;  
public class MonApplet extends JApplet {  
}
```

Les Applets

Méthodes particulières des applets :

`public void init()` : déclenchée lors de l'initialisation de l'applet
(appelée une seule fois)

`public void start()` : déclenchée au lancement de l'applet
`public void stop()` : déclenchée quand l'applet est arrêtée

`public void destroy()` : méthode appelée à la destruction de l'applet
(appelée une seule fois)

Les Applets

`public void paint(Graphics g)` : méthode appelée pour dessiner l'applet
(appelée automatiquement)

Pour forcer l'appel à la méthode `paint(..)`, utiliser `repaint()`

Les Applets

```
import javax.swing.*;  
public class MonApplet extends JApplet {  
  
    public void init() { ... }  
  
    public void start() { ... }  
  
    public void stop() { ... }  
  
    public void destroy() { ... }  
  
    public void paint(Graphics g) { ... }  
}
```

Les Applets

Code HTML permettant d'insérer une applet :

```
<APPLET CODE="MonApplet.class" WIDTH="600" HEIGHT="400">  
</APPLET>
```

Les Applets

Pour passer des paramètres à l'applet :

```
<APPLET CODE="MonApplet.class" WIDTH="600" HEIGHT="400">
  <PARAM NAME="nomParametre1" VALUE="toto">
  <PARAM NAME="nomParamètre2" VALUE="titi">
</APPLET>
```

Pour récupérer les valeurs des paramètres dans l'applet, utiliser la méthode `getParameter` de la classe `JApplet` :

Exemple :

```
this.getParameter("nomParametre1"); //retourne toto
```

Javadoc

Pour générer la documentation de votre projet :

utilitaire : **javadoc**

exemple :

```
javadoc *.java
```

génère tous les fichiers HTML au format de la documentation java de
ORACLE

Javadoc

The screenshot shows a Java Javadoc interface for the String class. The title bar reads "String (Java Platform SE 8)". The URL in the address bar is "docs.oracle.com/javase/8/docs/api/". The top navigation menu includes "OVERVIEW", "PACKAGE", "CLASS" (which is highlighted in orange), "USE", "TREE", "DEPRECATED", "INDEX", and "HELP". On the right, it says "Java™ Platform Standard Ed. 8". The left sidebar lists various Java classes under "java.lang": java.awt.print, java.beans, java.beans.beancontext, java.io, java.lang, java.lang.annotation, java.lang.instrument, java.lang.invoke, java.lang.management, java.lang.ref, java.lang.reflect, java.math, java.net, java.nio, java.nio.channels, and java.nio.channels.spi. Below this, a scrollable list continues with Double, Enum, Float, InheritableThreadLocal, Integer, Long, Math, Number, Object, Object, Package, Process, ProcessBuilder, ProcessBuilder.Redirect, Runtime, RuntimePermission, SecurityManager, Short, StackTraceElement, StrictMath, String, StringBuffer, StringBuilder, System, Thread, ThreadGroup, ThreadLocal, Throwable, and "...". The main content area starts with the class hierarchy: compact1, compact2, compact3, followed by java.lang. The section "Class String" is shown in bold. It lists "All Implemented Interfaces:" as Serializable, CharSequence, Comparable<String>. The class definition is shown in code:

```
public final class String  
extends Object  
implements Serializable, Comparable<String>, CharSequence
```

. A descriptive text follows: "The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class. Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:" An example code snippet is provided:

```
String str = "abc";
```

. Below it, the text "is equivalent to:" is followed by another code snippet:

```
char data[] = {'a', 'b', 'c'};  
String str = new String(data);
```

. Finally, the text "Here are some more examples of how strings can be used:" is followed by two more code snippets:

```
System.out.println("abc");  
String cde = "cde";
```

.

Javadoc

Les tags Javadoc :

Un bloc de commentaires java commençant par `/**`

Un tag Javadoc commence par un "@"
exemples :

- @author** : nom du développeur
- @deprecated** : pour préciser que la méthode est dépréciée
- @exception** : pour préciser une exception lancée par la méthode
- @param** : paramètre de la méthode
- @return** : que retourne la méthode
- @see** : documente une association à une autre méthode ou classe
- @since** : version où la méthode est apparue
- @throws** : exception lancée par la méthode
synonyme de **@exception**
- @version** : version d'une classe ou d'une méthode

Javadoc

Exemple :

```
/**  
 * Permet de faire la somme de 2 entiers  
 * @author Sébastien Serais  
 * @param entierA Premier entier à sommer  
 * @param entierB Second entier à sommer  
 * @return un entier correspondant à la somme de entierA et de entierB  
 */  
public int somme(int entierA, int entierB){  
  
    ...  
  
}
```

Toto - Microsoft Internet Explorer

Fichier Edition Affichage Favoris Outils ?

Précédente Rechercher Favoris

Adresse C:\Documents and Settings\nom\Bureau\Nouveau dossier\index.html

OK Liens

All Classes

Toto

Method Summary

int **somme**(int entierA, int entierB)
Permet de faire la somme de 2 entiers

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Toto

public **Toto**()

Method Detail

somme

public int **somme**(int entierA,
 int entierB)

Permet de faire la somme de 2 entiers

Parameters:

entierA - Premier entier à sommer
entierB - Second entier à sommer

Returns:

un entier correspondant à la somme de entierA et de entierB

Conventions de codage de SUN

Les conventions : <http://java.sun.com/docs/codeconv/>

En-tête de fichier :

```
/*
 * @(#)Blah.java 1.82 99/03/18
 *
 *Copyright (c) 1994-1999 Sun Microsystems, Inc.
 * 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information of Sun
 * Microsystems, Inc. ("Confidential Information"). You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered into
 * with Sun.
 */
```

```
package java.blah;

import java.blah.blahdy.BlahBlah;


```

```
/** ...
 * ...constructor Blah documentation comment...
 */
public Blah() {
    // ...implementation goes here...
}


```

Source : <http://java.sun.com/docs/codeconv/>

Les Threads

Gestion des processus

Les Threads

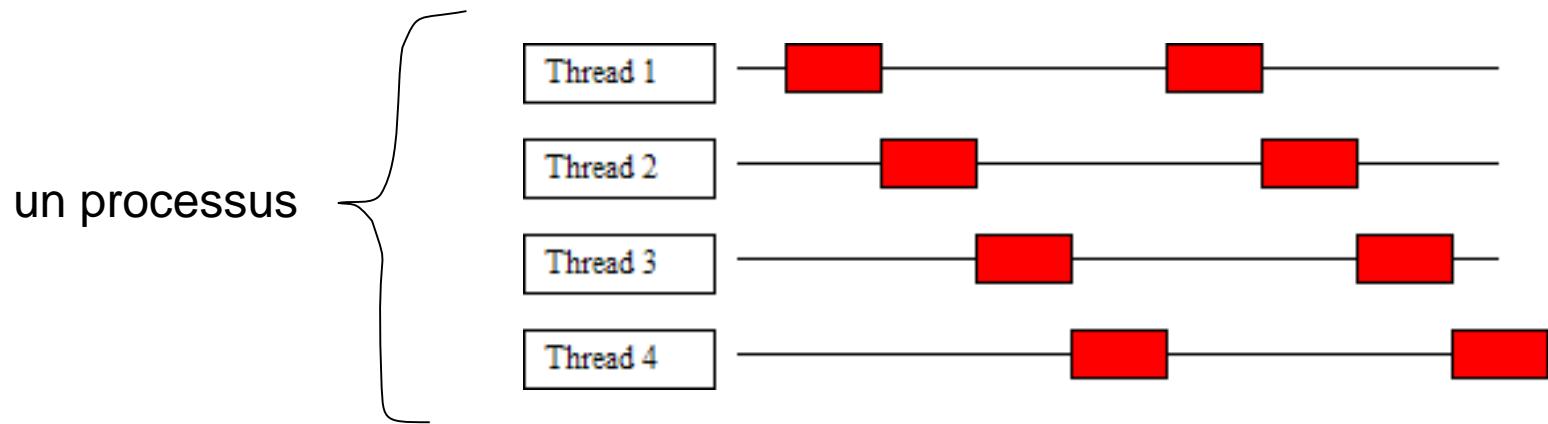
Thread n'est pas un processus (au sens lourd)

Les Threads sont des traitements qui vivent ensemble au sein d'un **même processus**

- Ils sont plus légers, plus rapides
(on parle de processus légers)

Les Threads partagent la **même mémoire** contrairement aux processus.

Les Threads



Exemple d'exécution de threads en fonction du temps
Ceci donne une impression de simultanéité

Méthode 1

Méthode 1 :

Création d'une classe qui dérive de la classe Thread :

```
public class MonThread extends Thread  
{...}
```

Méthode 1

```
public class MonThread extends Thread {  
  
    public void run() {  
  
        //...voilà le code associé au Thread...  
    }  
}
```

utilisation :

```
Thread th=new MonThread();  
th.start(); //lance la méthode run()
```

Méthode 1

Problème :

Comment passer des paramètres à la méthode run ??

Thread avec passage de paramètres

```
public class MonThread extends Thread {  
  
    private String s;  
  
    public MonThread(String s){  
        this.s = s;  
    }  
  
    public void run() {  
        // faire quelque chose avec s  
    }  
}
```

Méthode 2

Méthode 2 :

Création d'une classe qui implémente l'interface Runnable

```
public class MonThread implements Runnable  
{...}
```

Méthode 2

```
public class MonThread implements Runnable {  
  
    public MonThread(){  
        Thread th=new Thread(this);  
        th.start(); //lance la méthode run()  
    }  
  
    // méthode qui sera le cœur du processus  
    public void run() {  
        ...  
    }  
}
```

Instance de la classe qui implémente Runnable

Choix de la Méthode

- ▶ Dans quel cas utiliser la méthode 1 ou la méthode 2 ?

- ▶ Si la classe possède une classe mère, pas le choix → utiliser l'interface.

- ▶ Sinon, choisir l'une ou l'autre des méthodes, mais l'interface permet d'éviter de consommer l'héritage

Méthodes principales de la classe Thread

- ▶ `start()` → lance le Thread
(appel de la méthode `run()`)
- ▶ `sleep(500)` → met en pause le Thread (500ms ici)
- ▶ `stop()` → méthode dépréciée → ne plus utiliser !

La Méthode run()

- ▶ Elle correspond à l'exécution du thread

```
boolean arret=false; //attribut  
void run(){  
    while(arret==false){  
        ...  
    }  
}
```

Attribut de la classe qui permettra de sortir de la méthode run dès que sa valeur passera à true

Exemple

```
import java.util.Random;
public class MonThread extends Thread {
    String nom;
    public MonThread(String nom) {
        this.nom=nom;
    }

    public void run() {
        Random r=new Random();
        while(true){
            try {
                Thread.sleep(r.nextInt(3000));
            } catch (InterruptedException e) {}
            System.out.println("run "+nom);
        }
    }

    public static void main(String[] args){
        new MonThread("premier").start();
        new MonThread("deuxieme").start();
    }
}
```

Exemple

- ▶ exemple d'exécution :

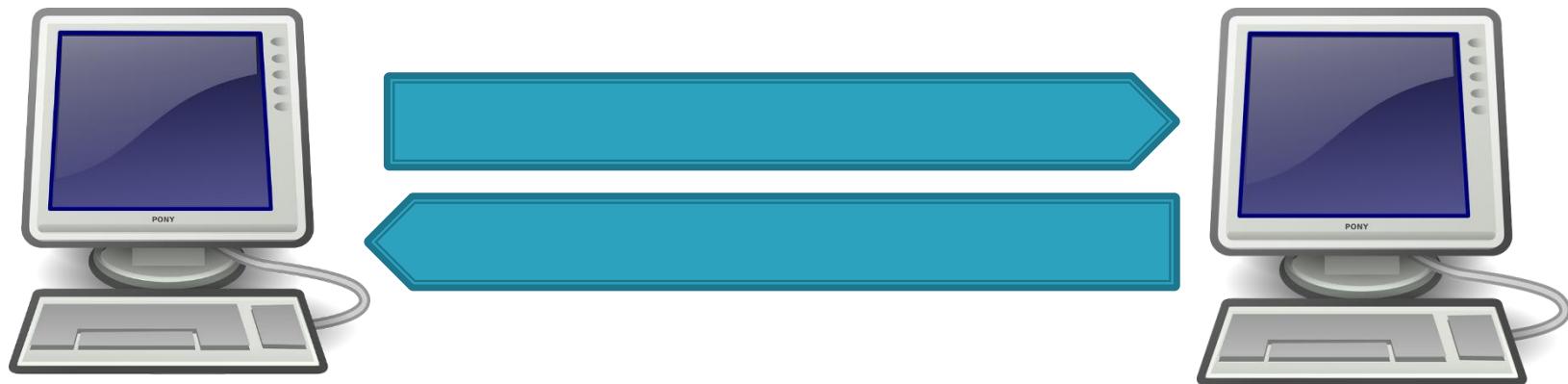
- ▶ run premier
- ▶ run deuxième
- ▶ run premier
- ▶ run deuxième
- ▶ run deuxième
- ▶ run premier
- ▶ run deuxième
- ▶ run premier
- ▶ run deuxième
- ▶ run deuxième
- ▶ run deuxième
- ▶ run premier
- ▶ ...

Le Réseau

Le réseau en Java

Les Flux

Un flux peut être représenté par un tuyau dans lequel vont circuler des informations :



Les Flux

Package `java.io`

nom des classes = préfixe + suffixe

4 suffixes possibles en fonction du type de flux et du sens du flux

| | Flux d'octets | Flux de caractères |
|-----------------------|---------------------------|---------------------------|
| Flux d'entrée | <code>InputStream</code> | <code>Reader</code> |
| Flux de sortie | <code>OutputStream</code> | <code>Writer</code> |

Les Flux

Famille des **Reader** : flux de lecture de caractères

Famille des **Writer** : flux d'écriture de caractères

Famille des **InputStream** : flux de lecture d'octets

Famille des **OutputStream** : flux d'écriture d'octets

Les Flux

Exemples de préfixes possibles :

File : pour les flux associés à des fichiers

Buffered : pour des flux avec tampon (buffer)
sert par exemple à lire une ligne entière avec
`String BufferedReader.readLine()`

Data : pour les flux de données
(permet de lire/ecrire les types de base : int, float, double,
char , byte,...)

Les Flux

Exemple : copie d'un fichier



Les Flux

```
import java.io.*;
public class CopieFichier {

    public static void main(String args[]) {
        try {

            FileInputStream fis = new FileInputStream("fichier1.mp3");
            FileOutputStream fos = new FileOutputStream("fichier2.mp3");
            while(fis.available() > 0)
                fos.write(fis.read()); // copie d'un octet
            fis.close();
            fos.close();

        } catch (Exception e) { .... }
    }
}
```

Les interactions avec le réseau

Le package java.net

Java prend en charge deux protocoles : TCP et UDP.

Pour UDP :

- classes **DatagramSocket** et **DatagramServer** permettent d'établir des connexions de type UDP

Pour TCP :

- serveur : classe **ServerSocket** : permet d'ouvrir un port
- client : classe **Socket** : permet de se connecter sur un port ouvert

Les interactions avec le réseau

URL : représente une URL Internet

URLEncoder: pour encoder les caractères spéciaux dans l'url (avec des %.. , +, ..)

URLDecoder : pour décoder une url

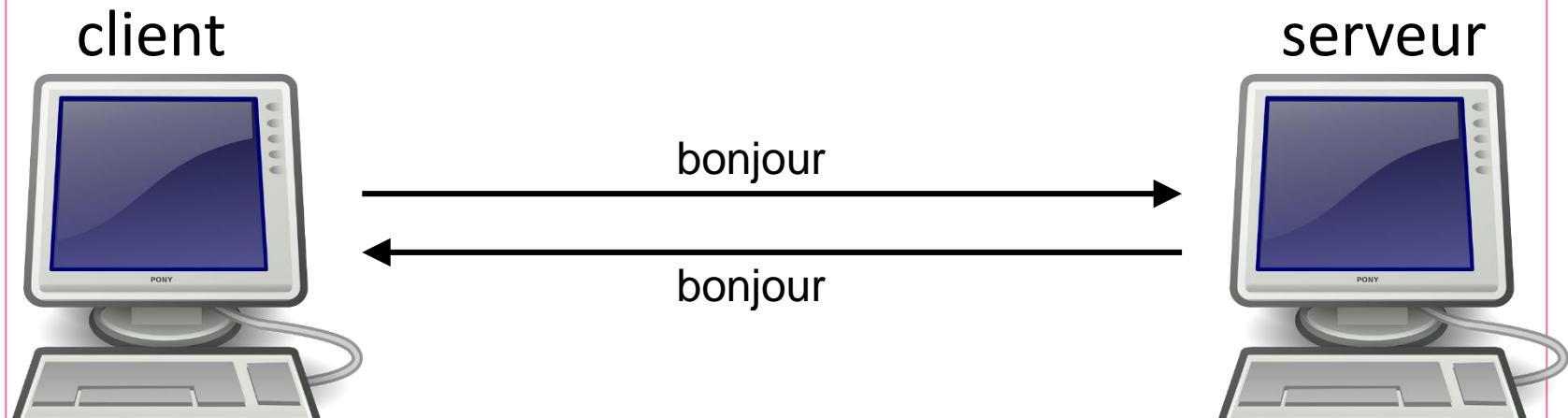
InetAdress : représente une adresse IP

→ **Inet4Adress** : représente une adresse IPv4

→ **Inet6Adress** : représente une adresse IPv6

Les interactions avec le réseau

Exemple TCP (faire un echo):



Les interactions avec le réseau

Création du serveur :

Construction d'un objet de type ServerSocket en indiquant un numéro de port supérieur à 1024 (les autres numéros sont réservés)

```
ServerSocket serveur=new ServerSocket(9999);
```

→ Le serveur écoute maintenant sur le port 9999

Les interactions avec le réseau

Attente de connexion d'un client : méthode accept()

accept() permet d' **attendre** les connexions des clients
→ méthode est **bloquante**

Lorsqu'un client se connecte :

accept() renvoie un objet de type **Socket** qui permettra de récupérer les flux d'entrée et de sortie

Les interactions avec le réseau

```
import java.io.*; import java.net.*;
public class MonServeur {
    public static void main(String[] args) {
        try {
            String message;
            ServerSocket socketServeur = new ServerSocket(9999);
            System.out.println("Le serveur écoute sur le port 9999");
            while (true) {
                Socket socketClient = socketServeur.accept();
                InputStream is = socketClient.getInputStream();
                OutputStream os = socketClient.getOutputStream();

                //création de flux plus adaptés
                BufferedReader in = new BufferedReader(new InputStreamReader(is));
                PrintStream out = new PrintStream(os);
                message = in.readLine();
                out.println(message);
                socketClient.close();
            }
        } catch (Exception e) { e.printStackTrace(); } } }
```

Les interactions avec le réseau

```
import java.net.*;
import java.io.*;
public class MonClient {
public static void main(String[] args) {
try {
    Socket socket = new Socket("nomMachineDistante",9999);

    //récupération des flux d'entree/sortie de caractères
    BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    PrintStream out = new PrintStream(socket.getOutputStream());

    out.println("bonjour");
    System.out.println(in.readLine());
    socket.close(); // ferme la socket et les 2 flux
} catch (Exception e) { e.printStackTrace(); }
}}
```

Les interactions avec le réseau

Autre exemple :

Récupération du contenu HTML d'une page WEB
(récupération du corps de la réponse HTTP)

Les interactions avec le réseau

```
import java.io.*;
import java.net.*;
public class ClientHTTP {

    public static void main(String args[])
    {
        try {
            URL url = new URL("http://www.sun.com");

            //recuperation du flux permettant la lecture des données
            URLConnection http=url.openConnection();
            InputStream is=http.getInputStream();
            int lu;
            while((lu=is.read())!=-1)
                System.out.print((char)lu);
            is.close();
        } catch (Exception e) { System.out.println("erreur"); }
    }
}
```

Les interactions avec le réseau

Si utilisation d'un proxy HTTP, ajouter :

```
Proxy prox=new Proxy(Proxy.Type.HTTP,new
```

```
    InetSocketAddress("proxy.intranet.int", 3128) );
```

et modifier :

```
URLConnection http=url.openConnection(prox);
```

La sérialisation

La sérialisation

La sérialisation

- ▶ Depuis JDK version 1.1
- ▶ Permet de rendre notamment un objet persistant
- ▶ Cet objet est mis sous une forme sous laquelle il pourra être reconstitué à l'identique.
- ▶ Pourra être stocké sur un disque dur ou transmis au travers d'un réseau pour le créer dans une autre JVM.
- ▶ C'est le procédé qui est utilisé par RMI.
- ▶ Aussi utilisé par les beans pour sauvegarder leurs états.

La sérialisation

- ▶ Mécanisme facile, transparent et standard
- ▶ Inutile de créer un format particulier pour sauvegarder et relire un objet.
- ▶ Format indépendant du système d'exploitation
- ▶ Ainsi, un objet sérialisé sur un système peut être réutilisé par un autre système pour récréer l'objet.
- ▶ Rq1 : La désérialisation de l'objet doit se faire avec la classe qui à été utilisée pour la sérialisation.
- ▶ Rq2 : La sérialisation peut s'appliquer facilement à tous les objets

La sérialisation : exemple

```
//une classe sérialisable
public class Personne implements java.io.Serializable
{
    private String nom = "";
    private String prenom = "";
    private int taille = 0;

    public Personne(String nom, String prenom, int taille)
    {
        this.nom = nom; this.taille = taille; this.prenom = prenom;
    }

    public String getNom() { return nom; }
    public void setNom(String nom) { this.nom = nom; }
    public int getTaille() { return taille; }
    public void setTaille(int taille) { this.taille = taille; }
    public String getPrenom() { return prenom; }
    public void setPrenom(String prenom) { this.prenom = prenom; }
}
```

La sérialisation : exemple

La classe **ObjectOuputStream**

Cette classe permet de **sérialiser** un objet.

Exemple : sérialisation d'un objet et enregistrement sur le disque dur

```
import java.io.*;
public class SerialiserPersonne {
    public static void main(String argv[])
    {
        Personne personne = new Personne("Dupond","Jean",175);
        try {
            FileOutputStream fichier = new FileOutputStream("personne.ser");
            ObjectOutputStream oos = new ObjectOutputStream(fichier);
            oos.writeObject(personne);
            oos.flush();
            oos.close();
        }
        catch (java.io.IOException e) { e.printStackTrace(); }
    }
}
```

La sérialisation : exemple

La classe **ObjectInputStream**

Cette classe permet de **désérialiser** un objet.

```
import java.io.*;
public class DeSerialiserPersonne {
    public static void main(String argv[]) {
        try {
            FileInputStream fichier = new FileInputStream("personne.ser");

            ObjectInputStream ois = new ObjectInputStream(fichier);
            Personne personne = (Personne) ois.readObject();

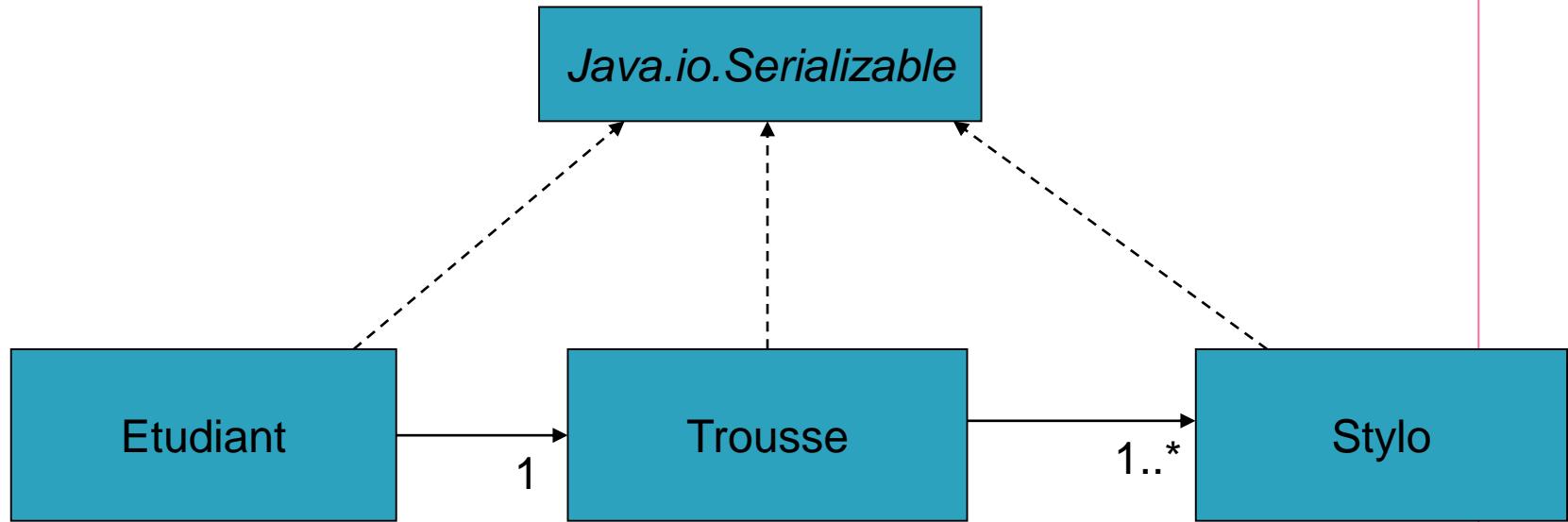
            System.out.println("Personne : ");
            System.out.println("nom : "+personne.getNom());
            System.out.println("prenom : "+personne.getPrenom());
            System.out.println("taille : "+personne.getTaille());
        } catch (java.io.IOException e) { e.printStackTrace()
        }catch (ClassNotFoundException e) { e.printStackTrace();
    }
}
```

Sérialisation et associations

La sérialisation fonctionne même si l'objet possède des attributs non primitifs (cas des associations).

Exemple :

on peut sérialiser/désérialiser un Etudiant sans aucun problème :



Exemple d'objet sérialisé

```
-í?sr?  
animaux.Ferme????????F?budgetI?compteurVisitesZ?distributeurNourritureI?effaceZ?e  
nVeilleS?etatBoulangerieZ?grillageI?  
nbPoulaillersI?niveauS?vacheL?dateCreationt?Lanimaux/DatePerso;L?datesDernieresVisi  
test?Ljava/util/Vector;L?  
debutVeilleq?~?L?mdpt?Ljava/lang/String;L?messagesq?~?L?  
theBasseCourst?Lanimaux/BasseCours;L?  
    theGranget?Lanimaux/Grange;L?theObjetAVendreq?~?xp????????????????????  
????????sr?animaux.DatePerso????????I?dateI?heureI?minutel?  
secondexp??íü????????sr?java.util.VectorÙ—  
}{€;`?I?capacityIncrementI?elementCount[?  
elementDatat?[Ljava/lang/Object;xp??????ur?[Ljava.lang.Object;ÍKÝs)I??xp???  
sq?~? ??íü?????????  
sq?~? ??íü????????sq?~? ??íü????????pppppppxpt?q?~?    ??????uq?~?  
.....
```

JDBC

Java Database Connectivity

Connexions aux BDR

- ▶ JDBC = ensemble de classes et d'interfaces permettant de se connecter aux bases de données
- ▶ package : `java.sql`

JDBC

Permet de se connecter aux bases de données suivantes :

- ▶ Oracle
- ▶ Mysql
- ▶ PostgreSQL
- ▶ Interbase (Borland)
- ▶ DB2 (IBM)
- ▶ Access, SQLServer (Microsoft)
- ▶ ...

comparatif des SGBD : <http://fadace.developpez.com/sgbdcmp/>

JDBC

Les étapes pour utiliser une base de données avec JDBC :

- 1.Charger les pilotes
- 2.Créer la connexion
- 3.Créer un Statement
- 4.Exécuter le requête SQL
- 5.Récupérer la réponse (si SELECT)
- 6.Fermer la connexion

JDBC

1 - Charger les pilotes

Exemple avec mysql:

```
Class.forName("com.mysql.jdbc.Driver"); //chargement du Driver
```

Cette instruction charge en mémoire la classe "com.mysql.jdbc.Driver".
Cette classe est contenue dans la librairie : mysql-connector-java-XXXX-bin.jar

C'est pourquoi cette librairie (.jar) doit être importée dans le projet :
projet->properties->java build path->librairies->
- add external jars si le jar n'est pas dans le projet
- add jars si je jar est copié dans le projet

JDBC

2 - Crée une connexion

```
String url="jdbc:mysql://localhost/maBase";  
String log="root";  
String mdp="";
```

```
Connection conn;  
conn=DriverManager.getConnection(url,log,mdp);
```

JDBC

3 - Crédit du Statement

```
Statement stmt=conn.createStatement();
```

Cet objet va permettre d'exécuter les requêtes SQL

JDBC

4 - Exécution de la requête SQL :

méthodes de la classe Statement :

- ✓ `executeUpdate` : permet de modifier la base (INSERT, UPDATE, DELETE,..)
- ✓ `executeQuery` : accès en lecture uniquement (SELECT)

Exemple :

```
stmt.executeUpdate("DELETE FROM personne WHERE nom LIKE 'toto' ");
```

JDBC

5 - Récupérer la réponse de la requête

JDBC renvoie les résultats dans un objet **ResultSet**

Exemple :

```
ResultSet rs = stmt.executeQuery("SELECT nom, age FROM personne");
```

JDBC

Parcours des enregistrements résultats

La méthode `next()` déplace le curseur à la ligne suivante

ATTENTION : Initialement, le curseur est positionné avant la première ligne du ResultSet

Le premier appel à `next()` déplace le curseur sur la première ligne

```
while(rs.next()) {  
    ...  
}
```

JDBC

Récupération des champs

utiliser la méthode getXXX du type approprié

Exemple :

```
String requete = "SELECT nom, age FROM personne";
```

```
ResultSet rs = stmt.executeQuery(requete);
while(rs.next()){
    String n = rs.getString("nom");
    int a = rs.getInt("age");
    System.out.println(n + " " + a);
}
```

Le programme affichera :
Anthony 25
Franck 30
Romain 24

JDBC

Remarque :

Possibilité de récupérer les champs à partir du numéro de colonne :

Exemple :

```
String n = rs.getString(1);
int a = rs.getInt(2);
```

Rq: les numéros commencent à 1

JDBC

6 - Fermeture de la connexion :

```
Connection connection = null;  
Statement statement = null;  
ResultSet resultat = null;  
try{  
    //traitements sur la base de données  
    //.....  
} catch(Exception e){  
    //gestion des erreurs  
} finally{  
    try{ if(resultat!=null) { resultat.close(); } } catch(Exception e){}  
    try{ if(statement!=null) { statement.close(); } } catch(Exception e){}  
    try{ if(connection!=null) { connection.close(); } } catch(Exception e){}  
}
```

JDBC

Utilisation des Prepared Statements

PreparedStatement (classe dérivée de Statement)

Avantage : Plus rapide et plus sécurisé

Un objet **PreparedStatement**, contrairement à l'objet Statement, fournit au SGBD une instruction SQL dès sa création

L'objet **PreparedStatement** ne contient plus seulement une instruction SQL, mais une instruction SQL précompilée

Quand le PreparedStatement est exécuté, le SGBD à juste à lancer l'instruction SQL du PreparedStatement sans avoir à le compiler

JDBC

Exemple de création d'un objet PreparedStatement

```
PreparedStatement ps = conn.prepareStatement("UPDATE personne SET age = ?  
WHERE nom = ? ");
```

La requête suivante :

"UPDATE personne SET age = ? WHERE nom LIKE ?"

est alors **précompilée**

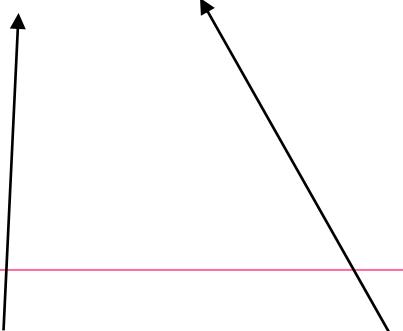
JDBC

Pour donner des valeurs aux ? :

utiliser les méthodes **setXXX** définies dans la classe PreparedStatement

exemple :

```
ps.setInt(1,25);  
ps.setString(2, "Romain");
```



no du ?

valeur

JDBC

Exemples avec et sans PreparedStatement

SANS :

```
stmt=conn.createStatement();
String requete = "UPDATE personnes SET age = 25 WHERE nom = 'Romain'";
stmt.executeUpdate(requete);
```

AVEC :

```
PreparedStatement updatePersonnes = conn.prepareStatement(
    "UPDATE personnes SET age = ? WHERE nom LIKE ?");
updatePersonnes.setInt(1,25);
updatePersonnes.setString(2, "Romain");
updatePersonnes.executeUpdate();
```

JDBC

intérêt du PreparedStatement :

si exécution de **plusieurs instructions SQL consécutives** (boucles par ex)

Exemple :

```
PreparedStatement updatePersonnes = conn.prepareStatement(  
    "UPDATE personnes SET age = ? WHERE nom LIKE ?");
```

//Change l'age de Romain

```
updatePersonnes.setInt(1,26);  
updatePersonnes.setString(2, "Romain");  
updatePersonnes.executeUpdate();
```

//Change l'age d'Anthony

```
updatePersonnes.setInt(1,26);  
updatePersonnes.setString(2, "Anthony");  
updatePersonnes.executeUpdate();
```

JDBC

Avantages du preparedStatement

Sécurité :

Gestion facilitée des **caractères spéciaux** dans les requêtes SQL.

Pas besoin d'échapper les caractères spéciaux, cela ne fera **automatiquement !**

Exemple complet d'accès à une BDD

- ▶ Voici un exemple complet d'une classe qui manipule une table d'une BDD mysql.
- ▶ La base de données s'appellera : maBDD
- ▶ Considérons la table suivante :

| message | | |
|-----------|-------------------|---|
| id | expéditeur | message |
| 1 | Sébastien | Bonjour à tous |
| 2 | Romain | C'est super le Java ! |
| 3 | Sébastien | oui, super langage ! |
| 4 | Franck | On manipule même des bases de données avec java ! |

Exemple complet d'accès à une BDD

- ▶ Pour info, voici le code SQL de création de la table :

```
CREATE TABLE `message` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `expediteur` varchar(45) NOT NULL,
  `message` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)
);
```

Ajout du pilote !!

- ▶ Pour pouvoir dialoguer avec une BDD, il faut ajouter le **pilote** de la base de données dans le projet (Driver).
- ▶ Ce pilote est téléchargeable sur le site du constructeur de la BDD. Par exemple, pour mysql, le pilote est le fichier :
 - mysql-connector-java-XXXX-bin.jar
- ▶ Pour ajouter le pilote dans le projet :
 - Projet-> properties -> java build path -> librairies -> add external Jars -> selectionner le fichier mysql...jar dans votre pc

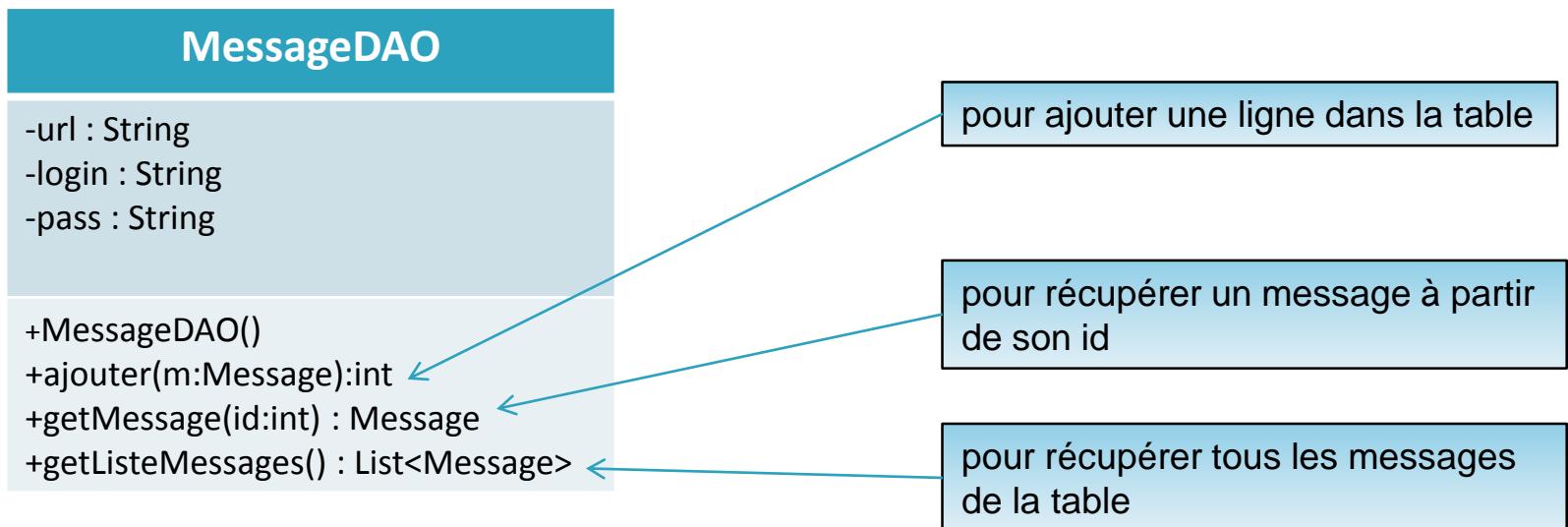
Exemple complet d'accès à une BDD

- ▶ Il faut ensuite créer une classe Java par table. Ces classes seront placées dans le package dto (Data Transfer Objects)
- ▶ Nous créons donc la classe Message :

| Message | |
|--|--|
| -id : int | |
| -expéditeur : String | |
| -message : String | |
| +Message() | |
| +getId():int | |
| +setId(int) | |
| +getExpéditeur():String | |
| +setExpéditeur(expéditeur:String):void | |
| +getMessage():String | |
| +setMessage(message:String):void | |

JDBC - Bonnes pratiques

- ▶ Il faut ensuite créer une classe Java par table qui sera chargée de communiquer avec cette table (cette classe contiendra le code SQL).
- ▶ Ces classes seront placées dans le package dao (Data Access Objects)
- ▶ Nous créons donc la classe MessageDAO :



Classe MessageDAO

```
import java.sql.*;  
public class MessageDAO {  
  
    final static String url = "jdbc:mysql://localhost/maBDD";  
    //ATTENTION sur MAC OS : "jdbc:mysql://localhost:8889/maBDD"  
    final static String login="root";  
    final static String pass="";  
  
    //constructeur  
    public MessageDAO() {  
        // chargement du pilote Mysql  
        try {  
            Class.forName("com.mysql.jdbc.Driver");  
        } catch (ClassNotFoundException e2) {  
            System.out.println("Impossible de charger le pilote de BDD, ne pas oublier ");  
            System.out.println("d'importer le fichier mysql-connector-java-XXXX.jar dans le projet");  
        }  
    }  
} //fin constructeur
```

```

/**
 * Permet d'ajouter un message dans la table message
 * @param m le message à ajouter
 * @return le nombre de ligne ajoutées dans la table
 */
public int ajouter(Message m) {
    Connection con = null;
    PreparedStatement ps = null;
    int retour=0;

    //connexion a la base de données
    try {

        con = DriverManager.getConnection(url, login, pass);
        ps = con.prepareStatement("INSERT INTO message (expediteur,message) VALUES (?,?)");
        ps.setString(1,m.getExpediteur());
        ps.setString(2,m.getMessage());

        //on execute la requete
        retour=ps.executeUpdate();
    } catch (Exception ee) {
        ee.printStackTrace();
    } finally {
        //fermeture du preparedStatement et de la connexion
        try {if (ps != null)ps.close();} catch (Exception t) {}
        try {if (con != null)con.close();} catch (Exception t) {}
    }
    return retour;
}

```

```

/**
 * Permet de récupérer un message à partir de son id
 * @param id l'id du message à récupérer
 * @return le message
 * @return null si aucun message ne correspond à cet id
 */
public Message getMessage(int id) {
    Connection con = null;
    PreparedStatement ps = null;
    ResultSet rs=null;
    Message retour=null;
    //connexion a la base de données
    try      {

        con = DriverManager.getConnection(url, login, pass);
        ps = con.prepareStatement("SELECT * FROM message WHERE id=?");
        ps.setInt(1,id);

        //on execute la requete
        rs=ps.executeQuery();
        if(rs.next())
            retour=new Message(rs.getInt("id"),rs.getString("expediteur"),rs.getString("message"));

    } catch (Exception ee) {
        ee.printStackTrace();
    } finally {
        //fermeture du rs,preparedStatement et de la connexion
        try {if (rs != null)rs.close();} catch (Exception t) {}
        try {if (ps != null)ps.close();} catch (Exception t) {}
        try {if (con != null)con.close();} catch (Exception t) {}
    }
    return retour;
}

```

```

/**
 * Permet de récupérer tous les messages de la table
 * @return la liste des messages
 */
public List<Message> getListeMessages() {
    Connection con = null;
    PreparedStatement ps = null;
    ResultSet rs=null;
    List<Message> retour=new ArrayList<Message>();

    //connexion a la base de données
    try      {
        con = DriverManager.getConnection(url, login, pass);
        ps = con.prepareStatement("SELECT * FROM message");

        //on execute la requete
        rs=ps.executeQuery();
        //on parcourt les lignes du resultat
        while(rs.next())
            retour.add(new Message(rs.getInt("id"),rs.getString("expediteur"),rs.getString("message")));

    } catch (Exception ee) {
        ee.printStackTrace();
    } finally {
        //fermeture du rs,preparedStatement et de la connexion
        try {if (rs != null)rs.close();} catch (Exception t){}
        try {if (ps != null)ps.close();} catch (Exception t){}
        try {if (con != null)con.close();} catch (Exception t){}
    }
    return retour;
}

```

```
//EXEMPLE de main permettant de tester notre classe
public static void main(String[] args) {
    MessageDAO messageDAO=new MessageDAO();
    //test de la méthode ajouter
    Message m=new Message(0,"Sébastien","Bonjour à tous");
    int retour=messageDAO.ajouter(m);
    System.out.println(retour+ " lignes ajoutées");

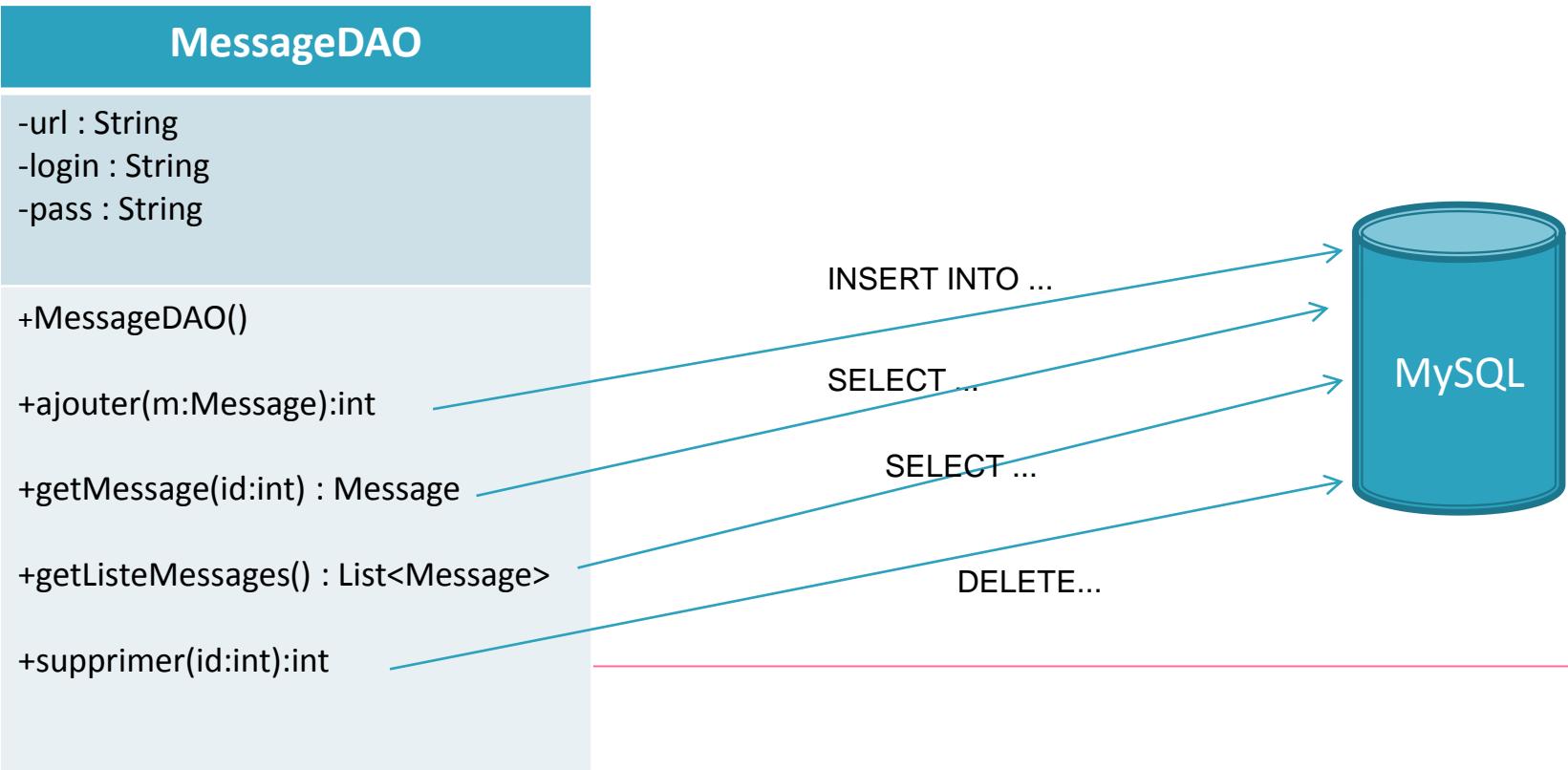
    //test de la méthode getMessage
    Message m2=messageDAO.getMessage(1);
    System.out.println(m2);

    //test de la méthode getListeMessages
    List<Message> liste=messageDAO.getListeMessages();
    System.out.println(liste);
}

} //fin de la classe
```

Résumons :

MessageDAO contient toutes les méthodes d'accès à la table message, on peut y ajouter d'autres méthodes :



Fin de l'exemple

Quelques remarques :

- ▶ Cette classe MessageDAO peut ensuite être utilisée depuis n'importe quelle autre classe (classe normale, Servlet, JSP,...)
- ▶ En production, les données de connexion à la BDD (login + mot de passe) ne doivent pas être écrites dans le code. Elles doivent idéalement être placées dans des fichiers à part (fichier .properties).

JDBC - Les transactions

Transaction = un jeu de une ou plusieurs instructions exécutées ensemble de façon unitaire

Toutes les instructions sont exécutées, ou aucune.

intérêt : banque (crédit/débit)

Pour activer les transactions : désactiver le mode autoCommit

`conn.setAutoCommit(false);`

JDBC

Pour valider définitivement la transaction : commit

Exemple :

```
conn.setAutoCommit(false);

Statement stmt=conn.createStatement();

stmt.executeUpdate("UPDATE compte1 SET solde=solde+1000 WHERE nom='Romain'");

stmt.executeUpdate("UPDATE compte2 SET solde=solde-1000 WHERE nom='Romain'");

conn.commit();
conn.setAutoCommit(true);
```

JDBC

Pour annuler une transaction :

`rollBack()`

A placer dans une `SQLException` par exemple

JDBC

ATTENTION !!

Sous Mysql : le type de tables MyISAM ne prend pas en charge les transactions !!

il faut donc changer le type des tables en INNODB par exemple....

JDBC : Accès aux méta-données

Possibilité d'accéder aux méta données en utilisant l'interface :

java.sql.ResultSetMetaData

Méthodes de l'interface ResultSetMetaData :

getColumnName()

getColumnLabel(int column)

getColumnType(int column)

...

JDBC : Accès aux métadonnées

► Exemple :

Pour afficher les noms de colonnes de la table :

```
ResultSetMetaData rsmd=rs.getMetaData();
```

```
int nbColonnes=rsmd.getColumnCount();
for(int i=1;i<=nbColonnes;i++)
    System.out.println(rsmd.getColumnLabel(i));
```

JDBC → Hibernate

- ▶ **Hibernate** est un **framework open source** très utilisé qui gère la persistance des objets dans des bases de données relationnelle.
- ▶ Le principe est le suivant :
- ▶ Le développeur écrit des fichiers XML (**mapping**) qui permettent de faire le lien entre :
 - une classe et une table
 - un attribut et une colonne de la table
- ▶ Une fois que ceci est fait, il suffit de dire que l'on souhaite enregistrer un objet et Hibernate se charge de générer les requêtes SQL adaptées (**INSERT INTO**)

Introspection

Introspection

- ▶ Existe depuis la version 1.1
- ▶ Possibilité de créer et de gérer dynamiquement des objets
- ▶ Permet de récupérer des informations sur les classes :
attributs, méthodes, ...
- ▶ package `java.lang.reflect`

Introspection

Exemple :

Récupération d'un objet Class à partir de son nom

```
Class classe = Class.forName("java.lang.String");
```

Introspection

Connaître la classe d'un objet

La méthode **getClass()** définie renvoie une instance de la classe **Class**

Exemple :

```
String chaine = "esigelec";
Class classe = chaine.getClass();
System.out.println("classe de l'objet chaine = "+classe.getName());
```

Résultat :

classe de l'objet chaine = java.lang.String

Introspection

Autres méthodes de la classe Class

- ▶ **Class[] getClasses()**
 - Renvoie les classes et interfaces publiques qui sont membres de la classe
- ▶ **Constructor[] getConstructors()**
 - Renvoie les constructeurs publics de la classe
- ▶ **Class[] getDeclaredClasses()**
 - Renvoie un tableau des classes définies comme membre dans la classe
- ▶ **Constructor[] getDeclaredConstructors()**
 - Renvoie tous les constructeurs de la classe
- ▶ **Field[] getDeclaredFields()**
 - Renvoie un tableau de tous les attributs définis dans la classe

Introspection

- ▶ **Method getDeclaredMethods()**
 - Renvoie un tableau de toutes les méthodes
- ▶ **Field getFields()**
 - Renvoie un tableau des attributs publics
- ▶ **Class[] getInterfaces()**
 - Renvoie un tableau des interfaces implémentées par la classe
- ▶ **Method getMethod()**
 - Renvoie un tableau des méthodes publiques de la classe incluant celles héritées

Introspection

- ▶ **Package getPackage()**
 - Renvoie le package de la classe
- ▶ **Classe getSuperClass()**
 - Renvoie la classe mère de la classe
- ▶ **boolean IsInterface()**
 - Indique si la classe est une interface
- ▶ **Object newInstance()**
 - Permet de créer une nouvelle instance de la classe
 - ...

Introspection

Remarque :

L'introspection est très utilisée dans les IDE pour afficher la liste des méthodes associées à un objet

Les RAD/IDE

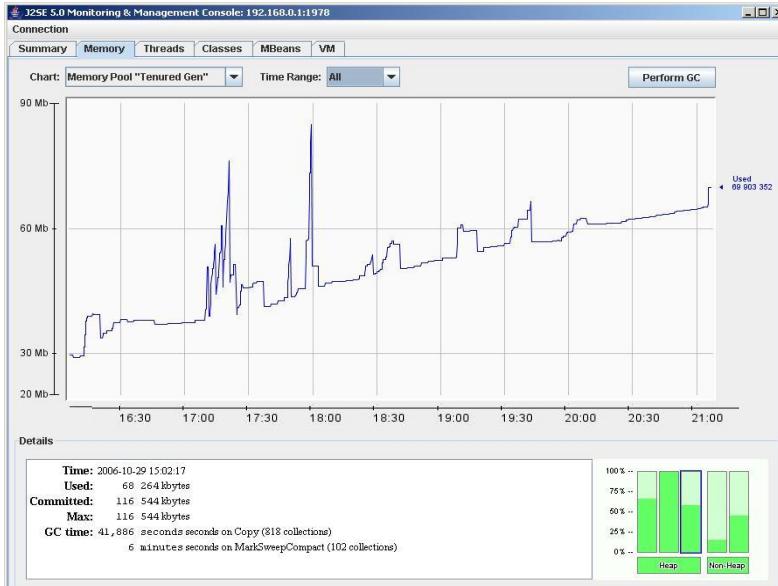
Rapid Application Development // Integrated Development Environment

sondage 2009 www.developpez.com

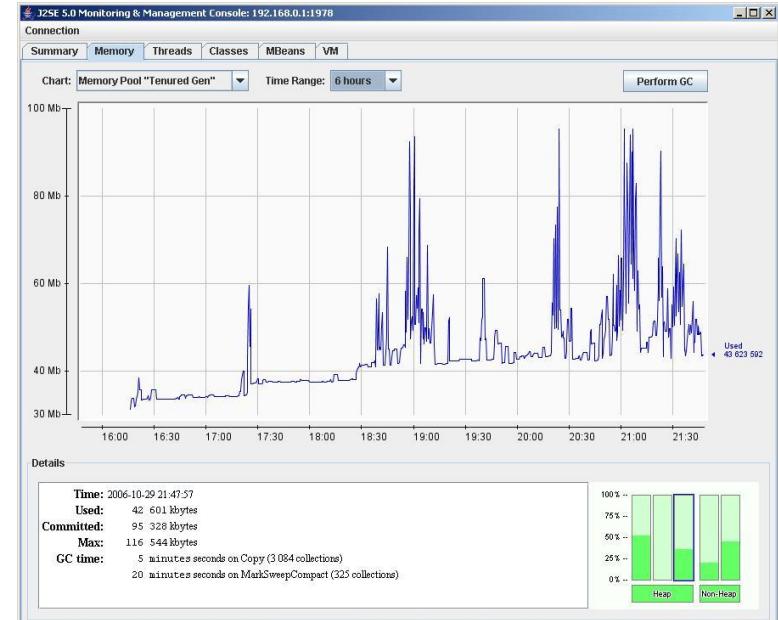
| Affichage des résultats du sondage: Quel EDI Java utilisez vous en 2009 ? | | | | |
|---|---|---------------------|--------|--|
| Eclipse |  | 188 | 49,74% | |
| NetBeans |  | 159 | 42,06% | |
| IntelliJ |  | 9 | 2,38% | |
| RAD / WSAD |  | 1 | 0,26% | |
| JDeveloper |  | 2 | 0,53% | |
| JCreator |  | 3 | 0,79% | |
| MyEclipse |  | 4 | 1,06% | |
| JBuilder (>= 2007) |  | 2 | 0,53% | |
| JBuilder (<= 2006) |  | 1 | 0,26% | |
| BEA Workshop Studio |  | 1 | 0,26% | |
| Editeurs de texte avancés (Emacs, VI, JEdit, UltraEdit, ...) |  | 6 | 1,59% | |
| Autre (précisez) |  | 2 | 0,53% | |

Monitoring mémoire

JConsole



mauvaise gestion de la mémoire
→ présence de fuites mémoire

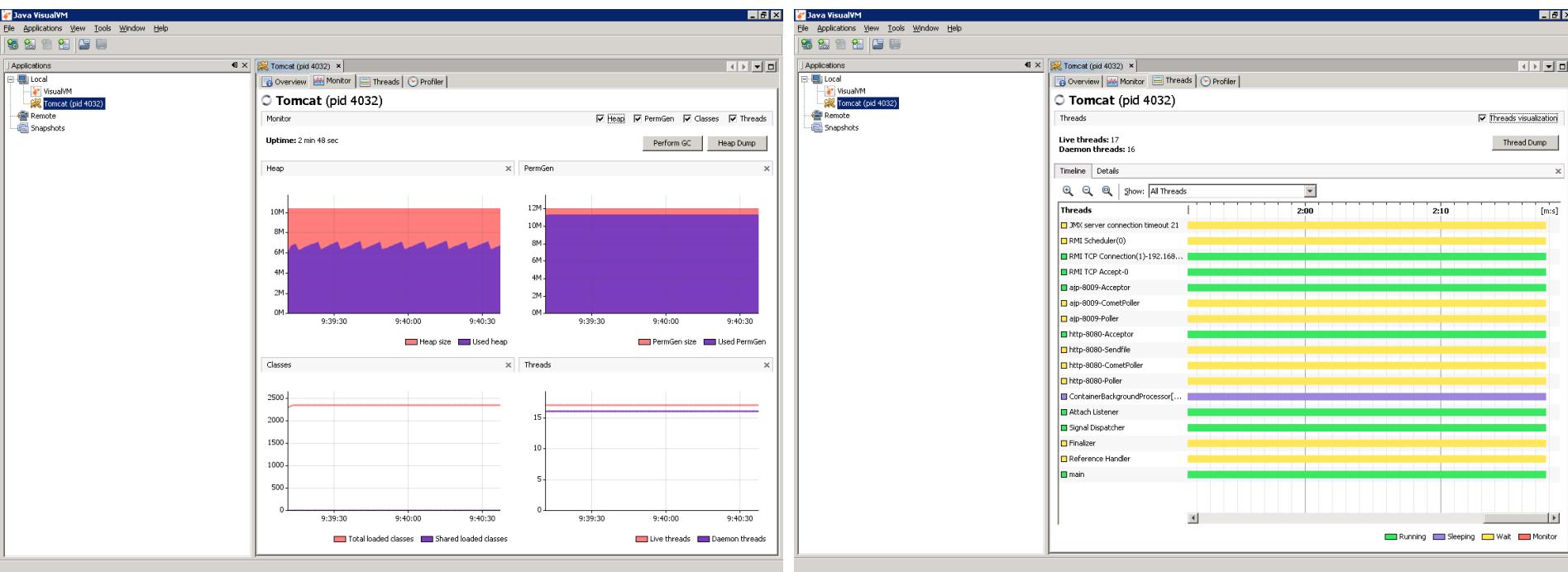


meilleure gestion de la mémoire
→ stabilisation de la consommation

Monitoring mémoire

JVisualVM

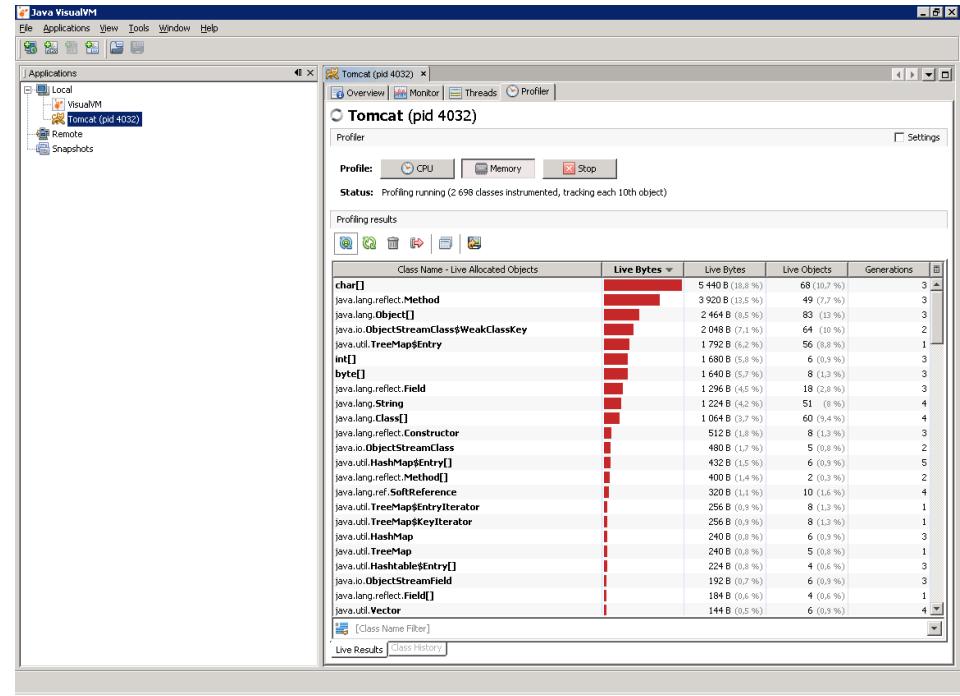
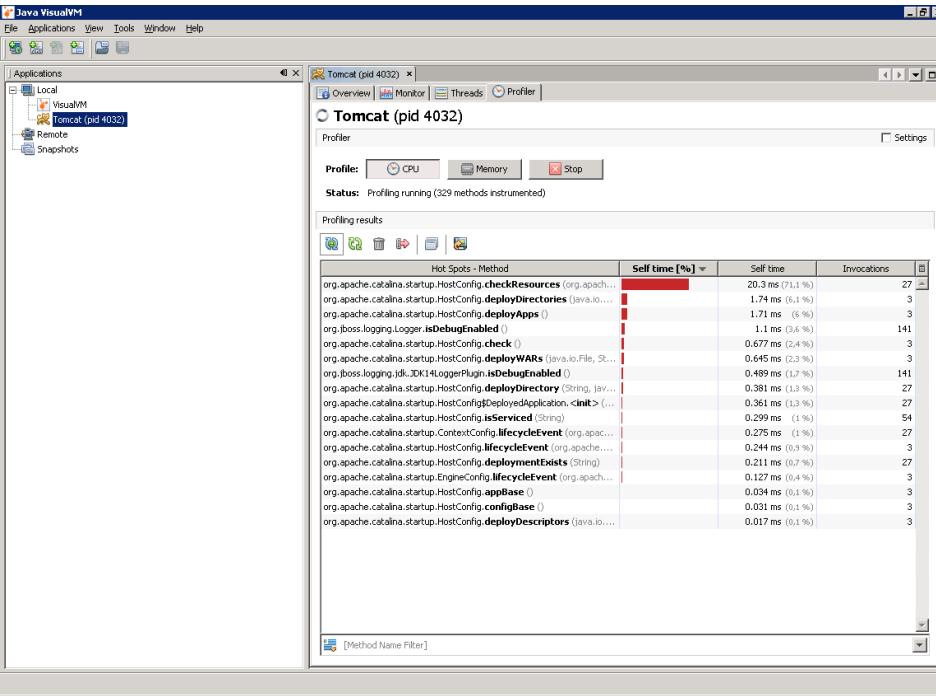
Outil de monitoring arrivé avec la version JDK6 Update 7 (2008)



Monitoring mémoire

JVisualVM

Outil de monitoring arrivé avec la version JDK6 Update 7 (2008)



Tests unitaires

JUnit :

- ▶ Outil permettant de réaliser des tests unitaires
- ▶ Eclipse intègre JUnit
- ▶ Permet d'automatiser les tests

Webographie

La référence JAVA :

- ▶ <http://java.sun.com>

Cours-tutoriaux :

- ▶ <http://java.developpez.com>
- ▶ <http://www.jmdoudoux.fr>

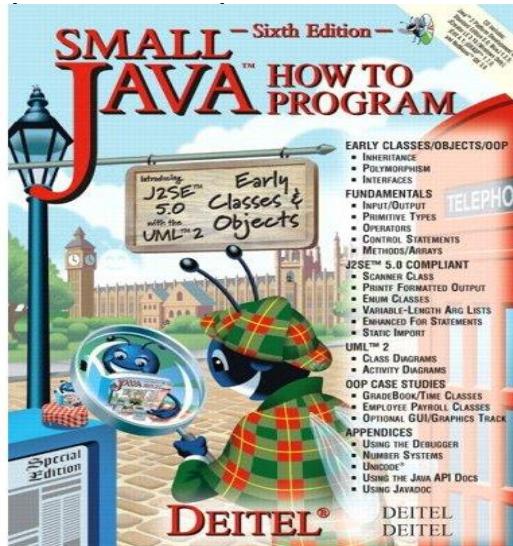
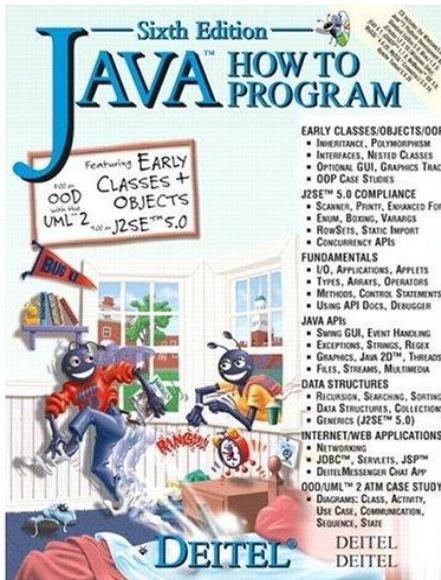
Outils de développement :

- ▶ **Eclipse** : <http://www.eclipse.org>
- ▶ **Netbeans** : <http://www.netbeans.org>

Bibliographie

▶ Livres :

- **Java How to Program** (6th Edition) (Deitel)
by Harvey M. Deitel, Paul J. Deitel (100 euros)
- **Small Java: How to Program** (Broché)
de Harvey M. Deitel, Paul J. Deitel
ISBN: 0131541579 (60 euros)
- **Java Comment Programmer** (Deitel)



JAVA côté Serveur

Objectifs

- ▶ Etre capable de concevoir des applications WEB dynamiques en Java
- ▶ Technologie JEE (ancien J2EE)

Service Internet : www

Le World Wide Web repose sur 3 standards :

- ▶ URL
- ▶ HTTP
- ▶ (x)HTML

Fonctionnement du WEB

- ▶ 1 - Connexion d 'un client sur le serveur
- ▶ 2 - Envoi d 'une **requête** par le client
- ▶ 3 - Envoi de la **réponse** du serveur
- ▶ 4 - Fermeture de la connexion

Le serveur WEB

- ▶ Programme qui réside sur un serveur connecté à un réseau
- ▶ Répond aux requêtes des clients exprimées sous forme d 'URL

exemple : <http://machine.serveur.com:8080/maPage.jsp>

Le Client WEB

- ▶ Souvent un navigateur
- ▶ Permet d'accéder aux services offerts par un serveur Web
- ▶ Comprend :
 - (x)HTML
 - feuilles de styles CSS
 - images (.gif, .jpg, .png,...)
 - javascript
 - ...

Le Client WEB

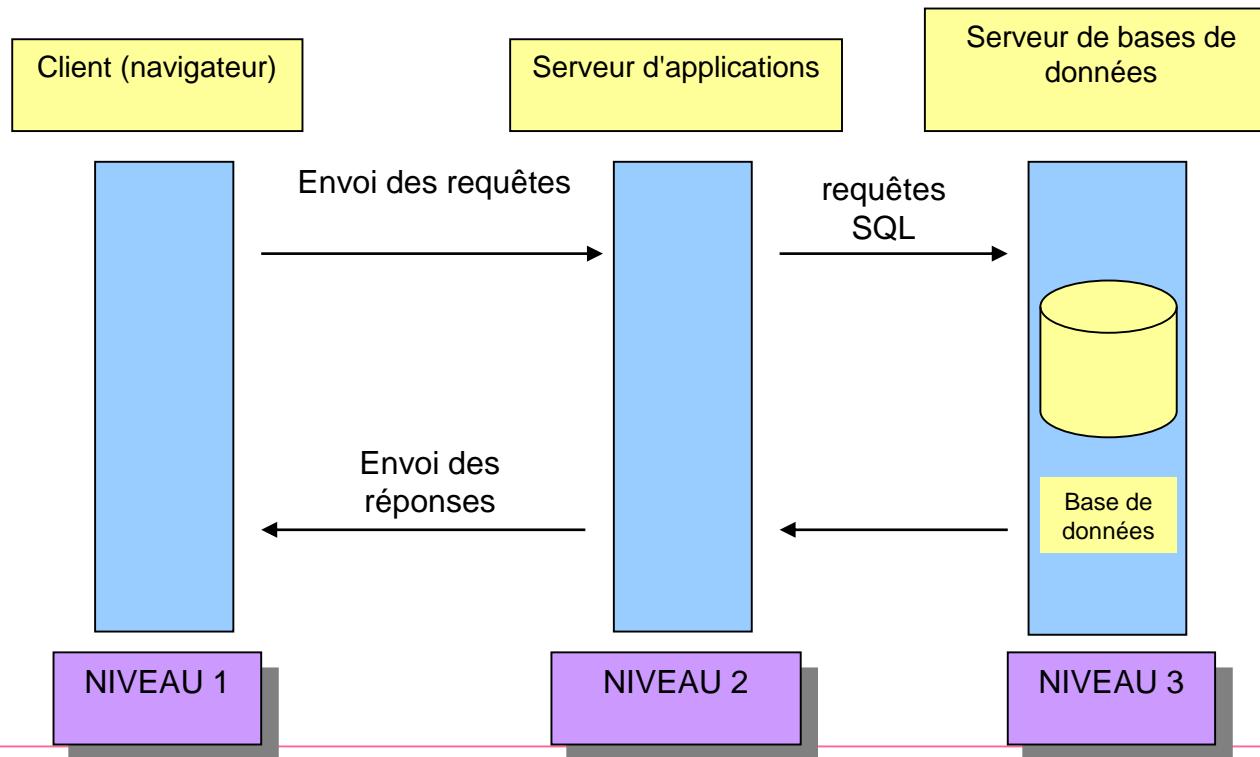
▶ Les navigateurs les plus connus sont :

- Internet Explorer,
- Mozilla Firefox,
- Google Chrome,
- Opéra,
- Safari (Apple),
- Konqueror (Projet KDE),
- ...



Architecture 3 niveaux (3-tiers)

- ▶ **niveau 1:** présentation : le navigateur du client
- ▶ **niveau 2:** applicatif (middleware) : un script ou un programme
- ▶ **niveau 3:** données: données nécessaires au niveau 2



Collecte des données utilisateur

Plusieurs solutions pour le client :

- ▶ La plus utilisée : **formulaires HTML**

Saisie de champs puis bouton de type « submit »
(Un contrôle de saisie peut être effectué en Javascript)

- ▶ D'autres méthodes possibles :
Flash, Applets Java, Applications mobiles (Android, Iphone, Blackberry), ...

Les formulaires

- ▶ Les formulaires servent à envoyer des informations au serveur web

```
<form action="page.jsp" method="post">  
...  
</form>
```

- ▶ action = page qui va recevoir les données envoyées
- ▶ method= méthode d'envoi des données (GET, POST,...)

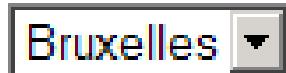
Les formulaires

- ▶ 1 : zone de texte :

```
<input type="text" name="montant" value="62€" />
```

Les formulaires

- ▶ 2 : champs de sélection :

A screenshot of a dropdown menu. The visible part of the menu shows the text "Bruxelles" in blue and red, indicating it is the selected item. To the right of the text is a small gray arrow pointing downwards, which is a standard icon for a dropdown menu.

```
<select name="ville" size="1">
  <option value="1">Paris</option>
  <option value="2" selected="selected">Bruxelles</option>
</select>
```

- ▶ Remarque : si **size="2"**

A screenshot of a dropdown menu. It displays two items: "Paris" in blue and "Bruxelles" in blue, each on a new line. The background of the menu is dark blue, and the text is white or light blue, making it stand out.

Les formulaires

► 3 : boutons radio :

Masculin
 Féminin

```
<input type="radio" value="M" name="genre" />Masculin
<br/>
<input type="radio" value="F" name="genre" checked="checked" />Féminin
```

Les formulaires

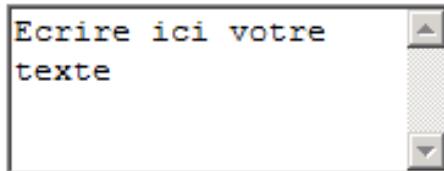
► 4 : cases à cocher :

- Windows
- Linux

```
<input type="checkbox" name="windows" />Windows  
<br />  
<input type="checkbox" name="linux" />Linux
```

Les formulaires

- ▶ 5 : zone de texte multilignes :



```
<textarea cols="20" rows="4" name="montexte">  
Ecrire ici votre texte  
</textarea>
```

Les formulaires

▶ 6 : les boutons :



```
<input type="submit" value="Valider" />  
<input type="reset" value="Annuler" />
```

Requête HTTP vers le serveur Web

- ▶ La requête HTTP contient :
 - l'URL de la ressource à accéder (page, programme,...)
 - les données de formatage (paramètres)
 - des infos d'en-tête complémentaires

Requête HTTP vers le serveur Web

Les deux requêtes les plus utilisées sont :

- ▶ la requête GET :

Les données envoyées sont ajoutées dans l'URL :

ex : `http://www.site.fr/page.jsp?nom=toto&prenom=titi&age=25`

- ▶ la requête POST :

Les données envoyées sont ajoutées dans le corps de la requête HTTP :

ex : `http://www.site.fr/page.jsp`

Retour des résultats au navigateur

- Le serveur renvoie le type de contenu renvoyé au client : le type **MIME** (ex : text/html)

par exemple : page HTML, image JPG, image GIF, document Word, Video .avi, etc...

- Ajoute ensuite le contenu de la réponse dans le flux de sortie..

La Requête HTTP en détail

Une requête HTTP comprend:

- ▶ une ligne de requête :
 - La méthode
 - L'URL
 - La version du protocole utilisé par le client (généralement HTTP/1.0 ou HTTP/1.1)
- ▶ Les champs d'en-tête de la requête
- ▶ Le corps de la requête (optionnel)

La Requête HTTP en détail

METHODE URL VERSION

EN-TETE : Valeur

...

EN-TETE : Valeur

Ligne vide

CORPS DE LA REQUETE

Exemple de requête HTTP:

GET http://www.esigelec.fr HTTP/1.0

Accept : text/html

If-Modified-Since : Thursday, 01-January-2011
15:02:55 GMT

User-Agent : Mozilla/4.0 (compatible; MSIE 7.0;
Windows NT 6.0)

Liste des commandes HTTP

GET

Requête de la ressource située à l'URL spécifiée

HEAD

Requête de la ressource située à l'URL spécifiée

Permet de récupérer uniquement l'en-tête de la réponse

POST

Envoi de données au programme située à l'URL spécifiée

PUT

Envoi de données à l'URL spécifiée

DELETE

Suppression de la ressource située à l'URL spécifiée

Liste des en-têtes HTTP

- ▶ **Accept**
Type de contenu accepté par le browser (par exemple *text/html*).
- ▶ **Accept-Charset**
Jeu de caractères attendu par le browser
- ▶ **Accept-Encoding**
Codage de données accepté par le browser
- ▶ **Accept-Language**
Langage attendu par le browser (anglais par défaut)
- ▶ **Authorization**
Identification du browser auprès du serveur
- ▶ **Content-Encoding**
Type de codage du corps de la requête
- ▶ **Content-Language**
Type de langage du corps de la requête
- ▶ **Content-Length**
Longueur du corps de la requête
- ▶ **Content-Type**
Type de contenu du corps de la requête (par exemple *text/html*).

Liste des en-têtes HTTP

- ▶ **Date**
Date de début de transfert des données
- ▶ **Forwarded**
Utilisé par les machines intermédiaires entre le browser et le serveur
- ▶ **From**
Permet de spécifier l'adresse e-mail du client

- ▶ **If-Modified-Since**
Permet de spécifier que le document doit être envoyé si il a été modifié depuis une certaine date
- ▶ **Link**
relation entre deux URL
- ▶ **Orig-URL**
URL d'origine de la requête
- ▶ **Referer**
URL du lien à partir duquel la requête a été effectuée
- ▶ **User-Agent**
Chaîne donnant des informations sur le client, comme le nom et la version du navigateur, du système d'exploitation

Réponse HTTP

- ▶ Une réponse HTTP comprend :
- ▶ **une ligne de statut :**
 - La version du protocole utilisé
 - Le code de statut
 - La signification du code
- ▶ **Les champs d'en-tête de la réponse**
- ▶ **Le corps de la réponse** : document demandé

Réponse HTTP

VERSION-HTTP CODE EXPLICATION

EN-TETE : Valeur

...

EN-TETE : Valeur

Ligne vide

CORPS DE LA REPONSE

Réponse HTTP

Exemple de réponse HTTP:

HTTP/1.0 200 OK

Date : Sat, 15 Jan 2011 14:37:12 GMT

Server : Microsoft-IIS/2.0

Content-Type : text/HTML

Content-Length : 2784

Last-Modified : Fri, 14 Jan 2011 08:25:13 GMT

En-têtes de réponse HTTP

- ▶ **Content-Encoding**
Type de codage du corps de la réponse
- ▶ **Content-Language**
Type de langage du corps de la réponse
- ▶ **Content-Length**
Longueur du corps de la réponse
- ▶ **Content-Type**
Type de contenu du corps de la réponse (par exemple *text/html*).
- ▶ **Date**
Date de début de transfert des données
- ▶ **Expires**
Date limite de consommation des données
- ▶ **Forwarded**
Utilisé par les machines intermédiaires entre le browser et le serveur
- ▶ **Location**
Redirection vers une nouvelle URL associée au document
- ▶ **Server**
Caractéristiques du serveur ayant envoyé la réponse

Techniques côté serveur

Voici les principales solutions utilisées côté serveur :

- ▶ CGI (Common Gateway Interface) (ancien)
- ▶ .NET (Microsoft) (ASP .NET, VB .NET, C#, ...)
- ▶ PHP
- ▶ J2EE ou JEE : Servlets Java / JSP (Oracle)

Architecture CGI

- ▶ Création d'un processus système pour traiter chaque requête

Avantages

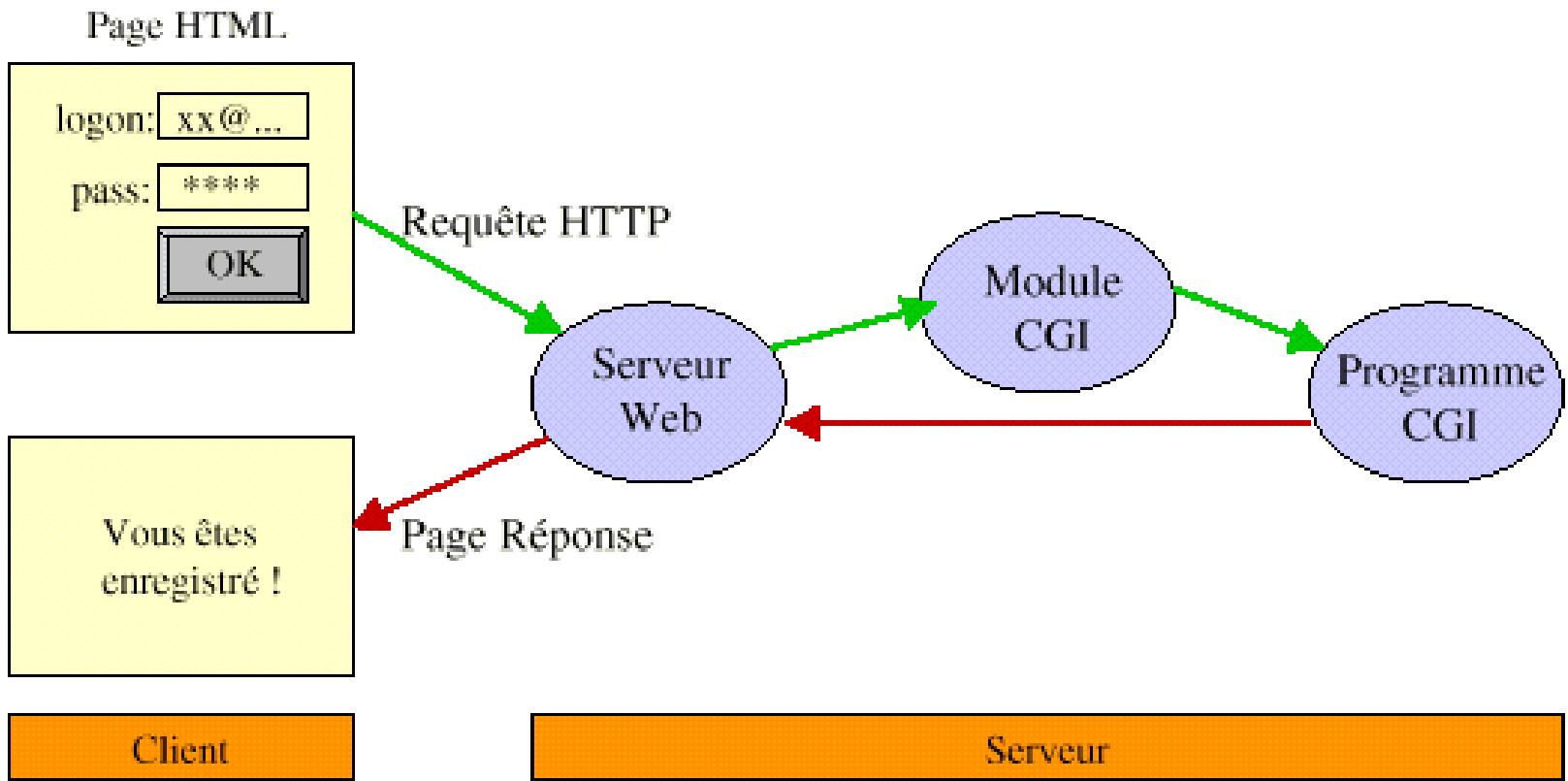
- ▶ gratuit, pris en charge par tous les serveurs Web
- ▶ peut être écrit dans n'importe quel langage (surtout Perl et C)

Architecture CGI

Inconvénients

- ▶ Manque d'évolutivité (plusieurs processus créés)
- ▶ Serveur très sollicité si plusieurs requêtes au même moment
- ▶ Assez lent
- ▶ Parfois difficile à développer

Architecture CGI

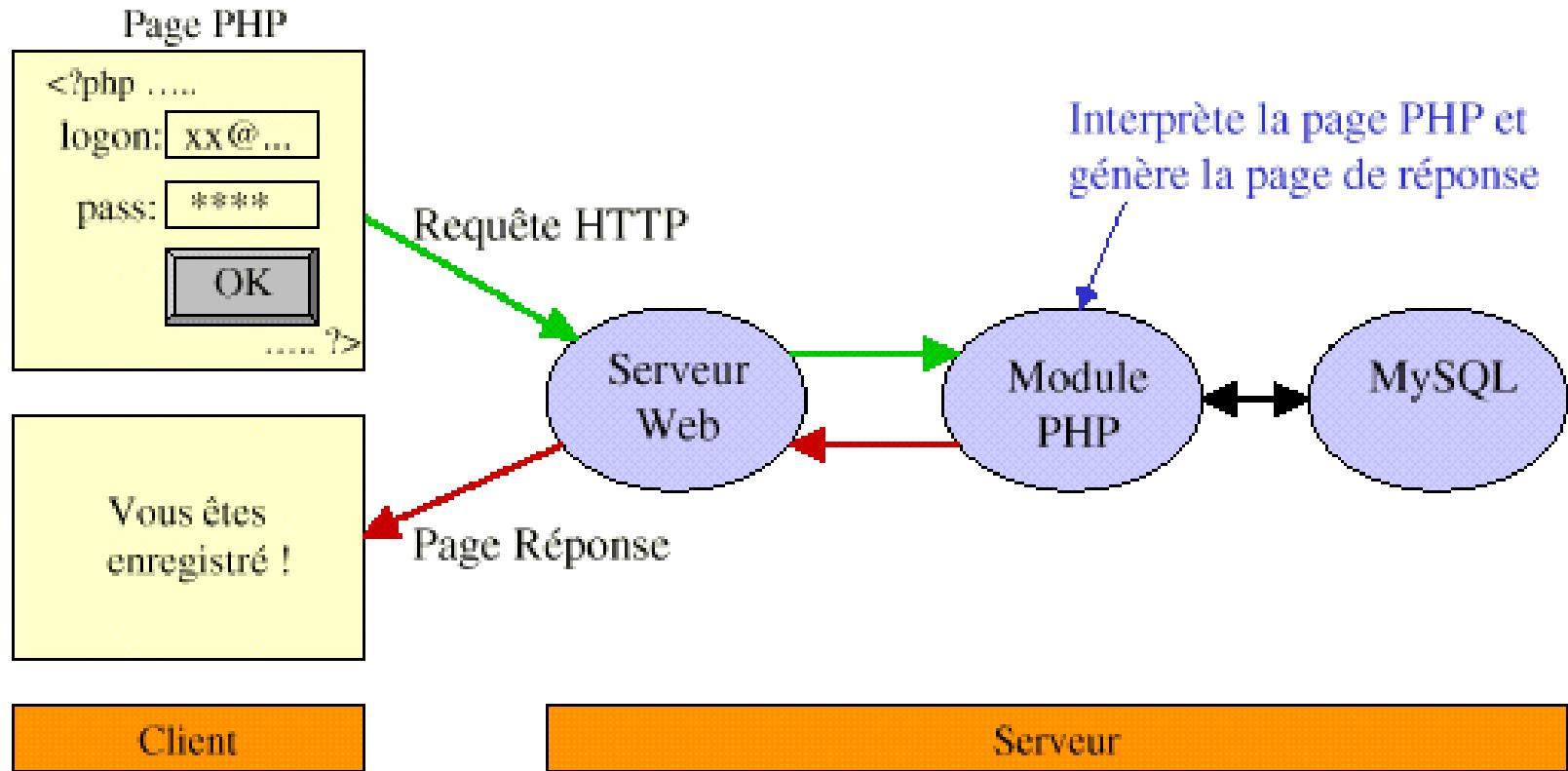


Architecture PHP

Pour chaque requête :

- ▶ Il y a une **compilation** de la page par le module PHP
- ▶ La page de réponse est alors générée et retournée au client
- ▶ possibilité d'utiliser un cache : pour éviter de recompiler à chaque fois.

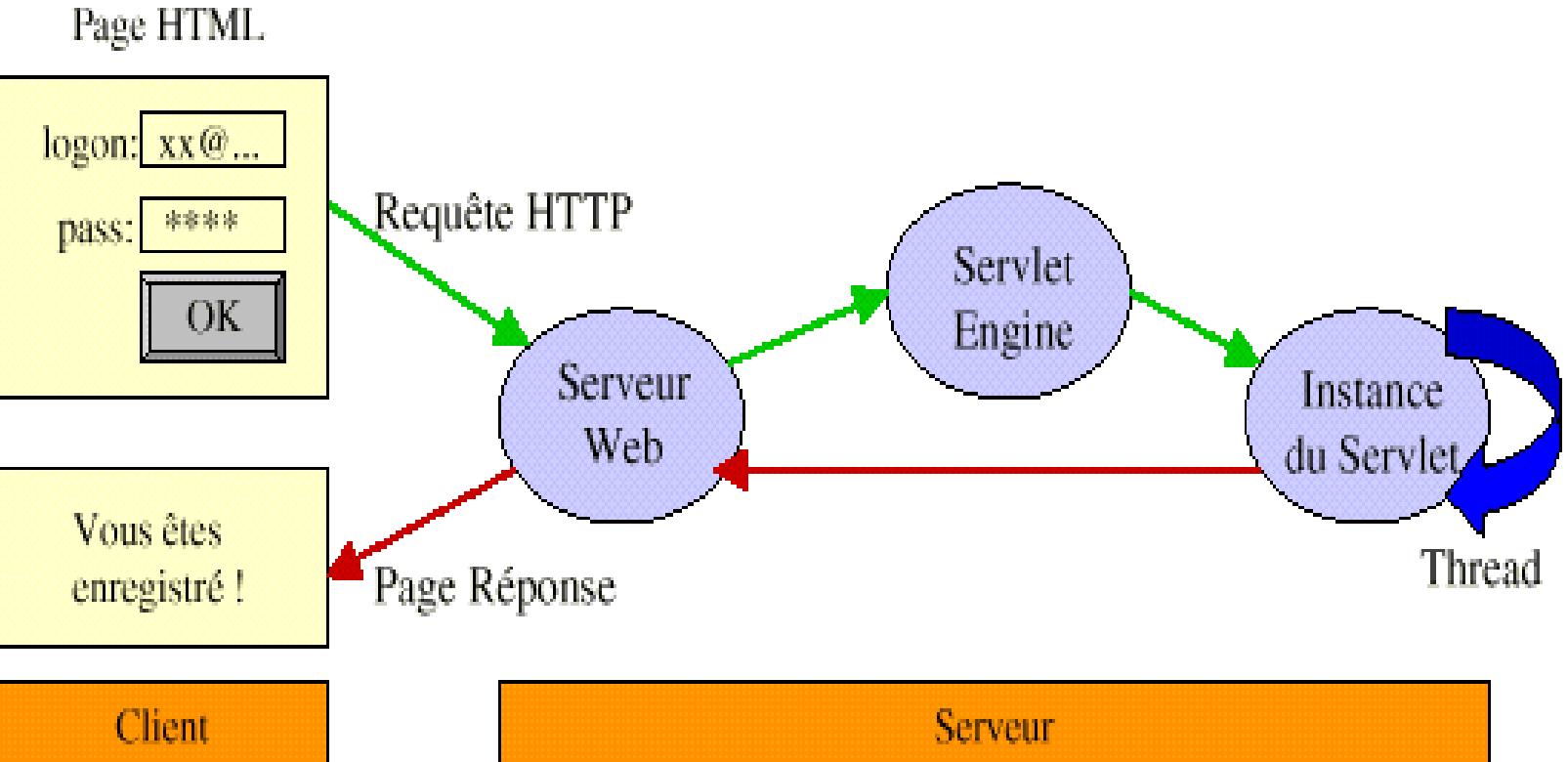
Architecture PHP



Architecture Servlet

- ▶ Servlet = classe écrite en java
- ▶ Une seule instance de la servlet est créée
- ▶ A chaque requête : lancement d'un nouveau thread sur la méthode doGet ou doPost de l'instance (unique) de la servlet

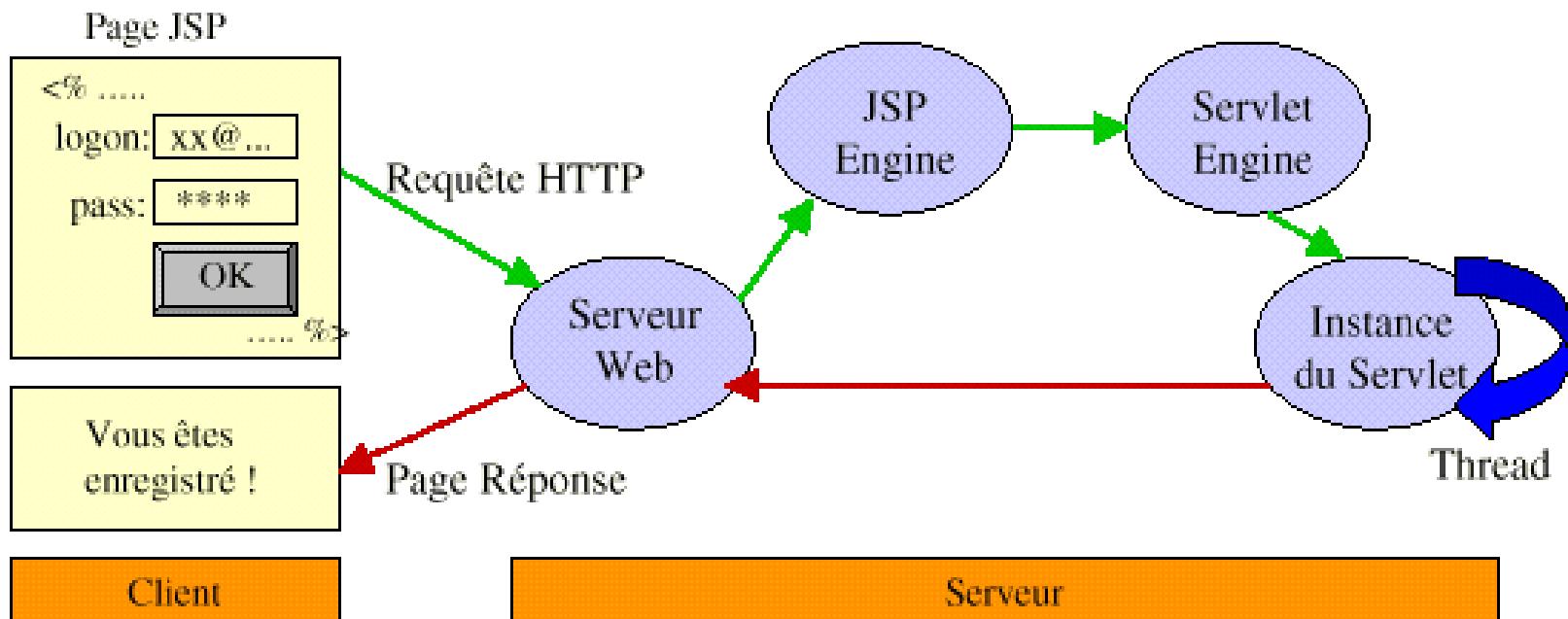
Architecture Servlet



Architecture JSP

- ▶ Si premier appel de la page JSP
 - → génération de la servlet correspondante
- ▶ La requête est ensuite envoyée à la servlet

Architecture JSP



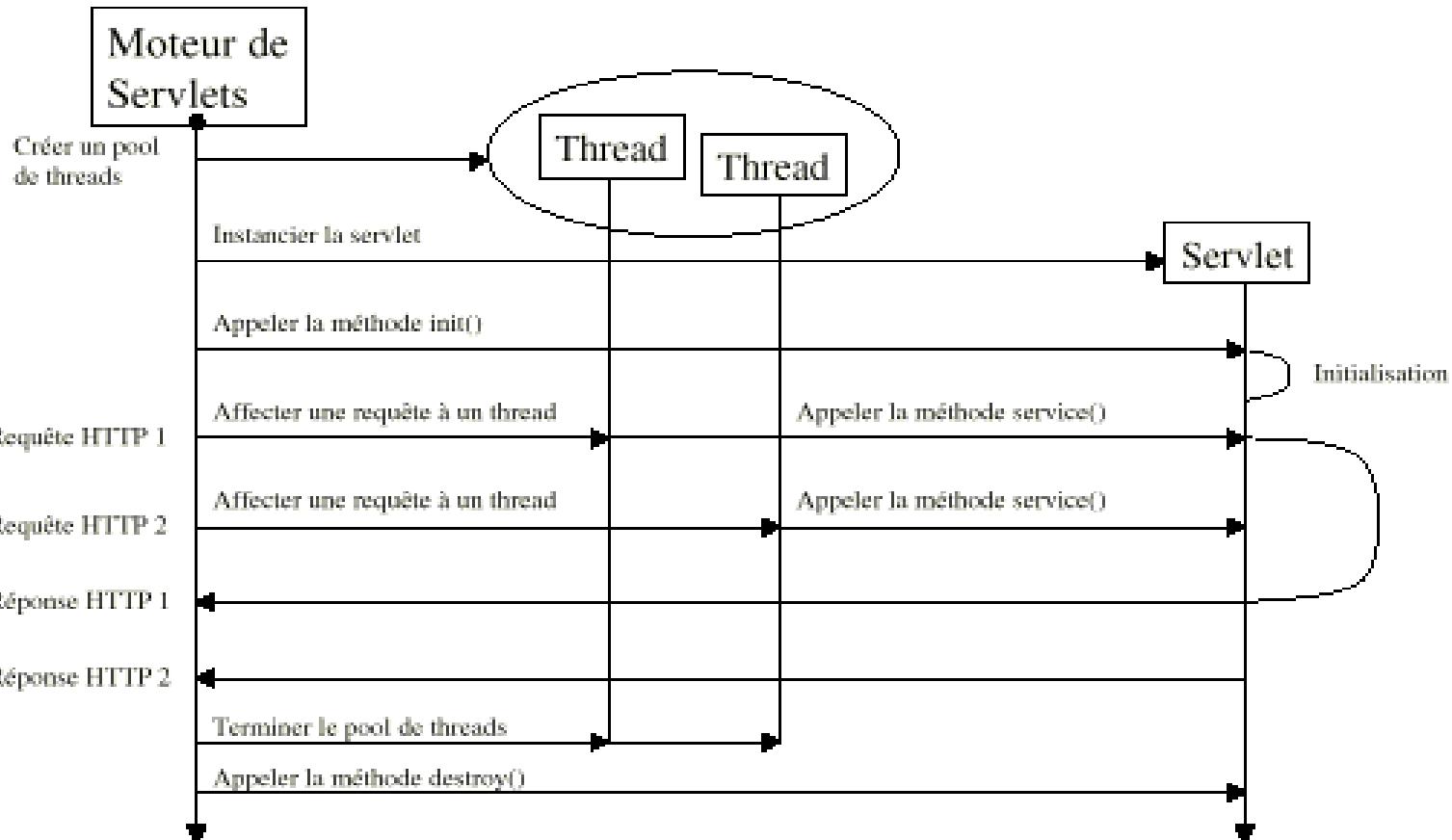
Avantages des servlets

- ▶ Efficaces
- ▶ Pratiques
- ▶ Puissantes
- ▶ Portables
- ▶ Gratuites

Servlets plus efficaces

- ▶ Résidentes en mémoire
- ▶ Chaque requête est traitée par un processus léger : le **thread** (et non un processus lourd (fork))
- ▶ Plusieurs optimisations possibles :
 - pool connexions,
 - cache, ...
- ▶ Code de la servlet chargé une seule fois en mémoire

Servlets plus efficaces



Servlets plus pratiques

- ▶ Tous les avantages liés à **Java** (bibliothèques java)
- ▶ La requête et la réponse sont représentées par des objets java :
 - javax.servlet.http.HttpServletRequest
 - javax.servlet.http.HttpServletResponse

Servlets plus puissantes

- ▶ Possibilité de **partager** des **informations** entre les différentes servlets
- ▶ Chaîner des servlets
- ▶ Pool de connexions BD

Servlets portables

- ▶ C'est du **Java** !
- ▶ Supportées par tous les serveurs WWW (Apache, Microsoft IIS, WebStar, ...) directement ou via des plugins/patches
- ▶ Principaux containers web : **Tomcat, JRun, ...**

Servlets gratuites

- ▶ Développement 100% gratuit :
 - JDK (Java SE suffit) (<http://oracle.com>)
 - Eclipse IDE for Java EE Developers (<http://www.eclipse.org>)
 - Apache/Tomcat (<http://jakarta.apache.org>)
- ▶ Tomcat est intégré dans des serveurs d'application comme JONAS, JBOSS, JBOSSWEB,...

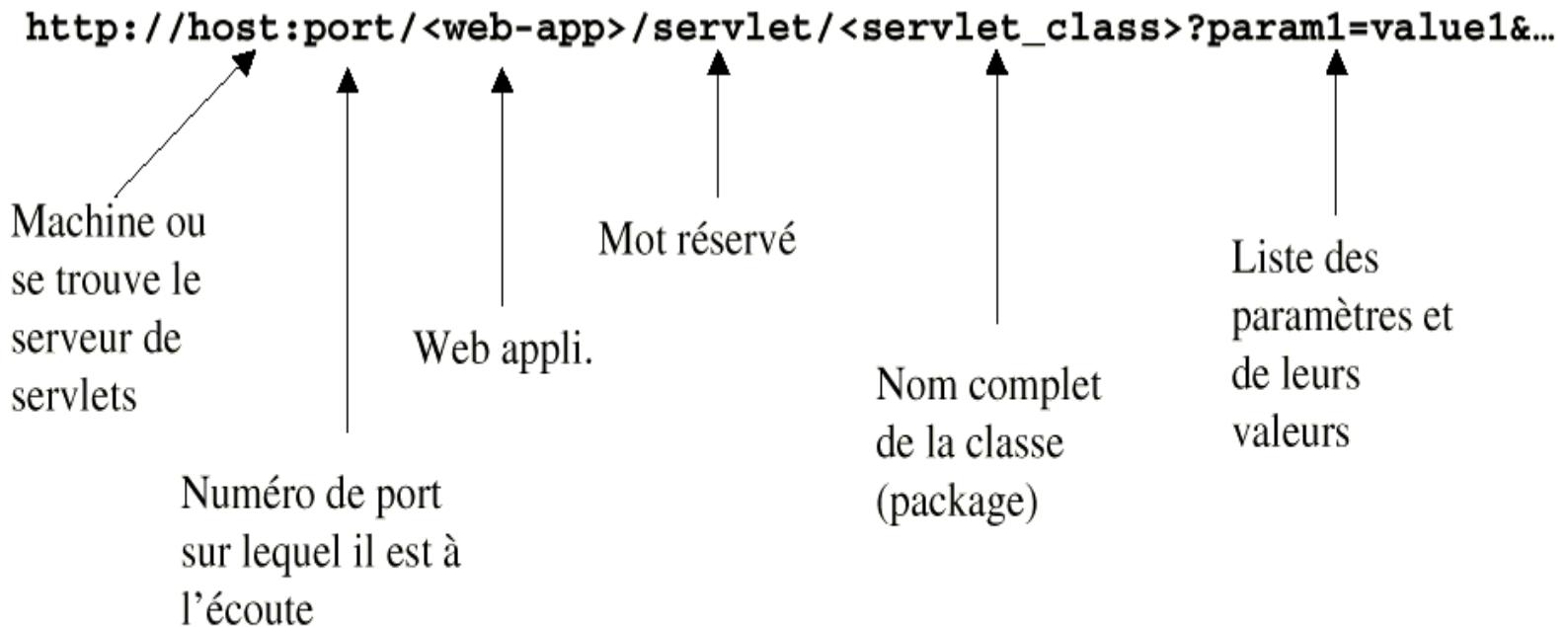
Servlet – Principe

Quand la servlet reçoit une requête HTTP :

- ▶ s'il n'existe pas encore d'instance de la Servlet, une instance est créée
- ▶ création d'un **thread** qui va traiter la requête

Charger et invoquer une servlet

L'URL permettant d'invoquer une servlet est du type :



Modèle de programmation

Une servlet est une classe java qui implémente l'interface
javax.servlet.Servlet

- soit directement
- soit en dérivant d'une classe implementant déjà cette interface comme (GenericServlet ou HttpServlet).

Le plus souvent, la classe dérivera de la classe
HttpServlet

Modèle de programmation

- ▶ L'interface javax.servlet.Servlet possède les méthodes :
 - **init()** : appelée à l'initialisation de la servlet
 - **service(..)** : appelée lors de l'arrivée d'une requête
 - **destroy()** : appelée avant la destruction de l'objet

Une servlet HTTP : HttpServlet

2 méthodes remplacent service() de la classe mère :

- ▶ doGet() : pour les requêtes Http de type GET
- ▶ doPost() : pour les requêtes Http de type POST

La servlet doit obligatoirement contenir l'une ou l'autre de ces 2 méthodes

Squelette d'une servlet HTTP

```
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleServlet extends HttpServlet {

    public void init(HttpServletRequestConfig c) throws ServletException {...}

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    {...}

    public void doPost(HttpServletRequest request, HttpServletResponse response)
    {...}

    public void destroy() {...}
}
```

Exemple

Cette servlet affiche "HELLO WWW".

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWWW extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {

        response.setContentType("text/html"); // Définit le type MIME de la réponse
        PrintWriter out = response.getWriter(); // Récupération du flux de sortie
        // on écrit dans le flux de sortie la page HTML résultat
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello WWW</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<H1>Hello WWW</H1>");
        out.println("</BODY></HTML>");
    }
}
```

Exemple (suite)

```
public void doPost(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {  
  
    doGet(request, response);  
}  
}
```

Le Cycle de Vie d'une Servlet

Etape 1 : Initialisation

- ▶ `init(HttpServletRequestConfig c)` est appelée par le serveur **une seule fois** lors du chargement en mémoire de la servlet
- ▶ Elle **ne fait rien** mais elle peut être surchargée (par ex. connexion à une BD)

Le Cycle de Vie d'une Servlet

Etape 2 : Traitement des requêtes HTTP

- ▶ Quand une requête client arrive, la méthode service() définie au niveau de la classe HttpServlet est exécutée.
- ▶ Cette méthode appelle `doGet()` ou `doPost()` suivant le type de requête HTTP reçue.
- ▶ Remarque : Les méthodes doGet et doPost définies au niveau de la classe HttpServlet ne font rien.

Le Cycle de Vie d'une Servlet

Etape 3 : Destruction

- ▶ Méthode `destroy()` est appelée par le serveur **une seule fois**
- ▶ Elle **ne fait rien mais elle peut être surchargée** (par ex. déconnexion d'une BD)

L'objet requête

Principales méthodes de **HttpServletRequest** :

- ▶ `getParameter(String parameterName)`: retourne la valeur (String) d'un paramètre mono-valué.
- ▶ `getSession()`: retourne la session sous la forme d'un objet HttpSession
- ▶ `getCookies()` : retourne un tableau d'objets Cookie.
- ▶ `getHeader(String headerName)`: retourne la valeur (String) d'un en-tête HTTP.
- ▶ `getParameterValues(String parameterName)`: retourne la valeur (tableau de String) d'un paramètre multi-valué.
- ▶ `getParameterNames()`: retourne une énumération (instance de la classe Enumeration) des noms de paramètres.
- ▶ etc.

Autres méthodes utiles de l'API: requête / réponse

Méthodes de HttpServletRequest :

- ▶ `getProtocol()` : HTTP/1.0
- ▶ `getServerName()` : company.com
- ▶ `getMethod()` : GET
- ▶ `getPathInfo()` : exemple.ShowParameters
- ▶ `getQueryString()` : les paramètres (avec leur valeurs) de la requête
- ▶ `getServletPath()` : /servlet/exemple/exemple.ShowParameters
- ▶

Méthodes de HttpServletResponse :

- ▶ `addCookie(Cookie cookie)` : ajoute un cookie
- ▶ `sendRedirect(String location)` : redirige la requête vers location

Exemple : page HTML

```
<html>
<form method="get"
      action="/monAppli/servlet/MaServlet">
<input name="login" type="text">
<input name="pass" type="password">
<input type="submit" value="OK">
</form>
</html>
```

Exemple : la servlet

```
import javax.servlet.http.*;
import java.io.*;

public class MaServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // (1) Utiliser l'objet "request" pour lire les paramètres
        String login = request.getParameter( "login");
        String pass = request.getParameter( "pass");
        // (2) Récupérer le PrintWriter pour écrire la réponse
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // (3) Vérifier que l'utilisateur est enregistré
        if("toto".equals(login) && "esigelec".equals(pass))
            out.println("<html>Vous êtes Enregistré !</html>");
        else
            out.println("<html>Erreur: Vous n 'êtes pas Enregistré !</html>");
    }
}
```

Descripteur de déploiement

Le descripteur de déploiement : **web.xml**

- ▶ Contient tous les paramètres d'une application Web
- ▶ Application web = collection de **servlets, pages html, CSS , images, classes Java, librairies .jar, ...**
- ▶ Une application Web est définie par un nom de contexte :
ex: <http://www.monsite.com/nomDuContexte>

Structure d'un projet JEE

nomDuContexte

.jsp

.htm

.css

.jpg, .gif, ...

WEB-INF

 classes

 .class

 lib

 .jar

 .zip

 web.xml

Les cookies

Qu'est-ce qu'un cookie ?

- ▶ **Morceaux d'information** (collection de couples clé/valeur) envoyés par le serveur et renvoyés par le client quand il revient visiter une page sur ce même serveur
 - ▶ **Durée de vie est réglable**
 - ▶ Permettent la **persistance** d'une information
 - ▶ Souvent utilisés pour :
 - Identifier des utilisateurs (e-commerce)
 - Eviter la saisie d'informations à répétition (login, password, adresse, téléphone...)
 - Gérer des « préférences utilisateur » (sites portails...)
 - Les suivis de session
 - ...
 - ▶ Très efficaces et faciles d'utilisation.
- MAIS** possibilité de désactiver l'acceptation des cookies...

Cookies et sécurité

- ▶ Les cookies ne sont jamais interprétés ni exécutés. Donc ils ne peuvent **pas contenir de virus** !
- ▶ taille d'un cookie **limitée à 4096 octets**
- ▶ les navigateurs se limitent à 300 cookies (20 par site)
- ▶ Pratiques pour conserver des données non sensibles (nom, l'adresse, ...) → **Pas de no de cartes bleues** !
- ▶ Mais ne pas utiliser cookie pour mettre en place de la sécurité !

Création d'un cookie

→ classe : javax.servlet.http.Cookie.

```
Cookie unCookie = new Cookie(name, value);
```

Les 2 arguments sont de type java.lang.String.

Ajout d'attributs à un cookie

- ▶ **setComment** ajoute un commentaire au cookie
- ▶ **setDomain** spécifie le nom du domaine d'application du cookie (pour élargir le cookie à d'autres machines du domaine)
- ▶ **setMaxAge** spécifie la durée de vie en secondes du cookie. Par défaut = durée de la session

Ajout d'attributs à un cookie

- ▶ **setPath** spécifie un chemin (sur le serveur) à partir duquel le cookie peut être lu
- ▶ **setSecure** spécifie si le cookie doit être envoyé via SSL (connexion sécurisée) ou non
- ▶ **setName** et **setValue** vont permettre de spécifier nom et valeur du cookie

Envoi d'un cookie

- ▶ Pour envoyer le cookie, on doit l'ajouter à l'en-tête (header) de la réponse HTTP produite par la servlet.

```
response.addCookie(monCookie);
```

Récupération des cookies

- ▶ **Cookie[] HttpServletRequest.getCookies() :**
 - Permet de récupérer les Cookies envoyés
- ▶ **exemple :**
 - `Cookie[] tab=request.getCookies();`

Exemple de récupération du cookie de nom "utilisateur"

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.*;

public class Exemple extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        Cookie[] cookies = request.getCookies();
        for(int i=0; i<cookies.length; i++) {
            Cookie cookie = cookies[i];
            String nomCookie = cookie.getName();
            if (nomCookie.equals( "utilisateur"))
                String nomUtilisateur = cookie.getValue();
        }

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println( "<BODY><P>Bonjour Mr" + nomUtilisateur + "!"</P></BODY></HTML>");
    }
}
```

Les sessions

Plusieurs possibilités :

- ▶ utilisation des cookies (déjà vu)
- ▶ réécriture d'URL : passage de paramètres
- ▶ utilisation des champs de formulaire cachés ("hidden")
- ▶ utilisation de l'objet **HttpSession**

La réécriture d'URL

Principe

- ▶ Ajout dans la chaîne de requête de la servlet des informations supplémentaires identifiant la session :

```
<a href="http://host/servlet/foo?uid=toto">Acheter</a>
```

- ▶ l'ID utilisateur est transmis en même temps que la requête, il est accédé par chaque servlet mentionnée qui récupère grâce à celui-ci les informations persistantes (BD, fichiers).

La réécriture d'URL

Avantage

- ▶ Fonctionne sur tous les serveurs

Inconvénients

- ▶ Pas d'envoi de données volumineuses
- ▶ Les données sont visibles et modifiables par l'utilisateur -> gros pb sécurité !

La réécriture d'URL: exemple

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ShoppingCartViewerRewrite extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        ...
        out.println("<HEAD><TITLE>Current Shopping Cart Items</TITLE></HEAD>");
        out.println("<BODY>");
        // Get the current session ID, or generate one if necessary
        String sessionid = req.getPathInfo();
        if (sessionid == null)
        {
            sessionid = generateSessionId();
        }
        ...
        // Include the session ID in the action URL.
        out.println("<FORM ACTION=/servlet/ShoppingCart/" +sessionid+ "\" METHOD=POST>");
        ...
        out.println("</FORM>");
        out.println("</BODY></HTML>");
    }

    private static String generateSessionId() {
        ...
    }
}
```

Champs de formulaires cachés ("Hidden")

Principe

Des champs cachés sont ajoutés à un formulaire HTML :

```
<INPUT TYPE="HIDDEN" NAME="uid" VALUE=toto>
```

▶ Avantages

Fonctionne sur tous les serveurs.

Données non visibles directement par l'utilisateur.

▶ Inconvénients

Pas d'envois de données volumineuses.

Nécessité de générer des formulaires **dynamiquement**.

Problème de sécurité car modifiable !

Champs de formulaires cachés ("Hidden") :exemple

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ShoppingCartViewerHidden extends HttpServlet {
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HEAD><TITLE>Current Shopping Cart Items</TITLE></HEAD>");
        out.println("<BODY>");
        ...
        // Ask if the user wants to add more items or check out.
        // Include the current items as hidden fields so they'll be passed on.
        out.println("<FORM ACTION="/servlet/ShoppingCart" METHOD=POST>");
        if (items != null) {
            for (int i = 0; i < items.length; i++) {
                out.println("<INPUT TYPE=HIDDEN NAME=item VALUE=\"" +items[i] + "\">");
            }
        }
        out.println("Would you like to<BR>");
        out.println("<INPUT TYPE=SUBMIT VALUE=\" Add More Items \">");
        out.println("<INPUT TYPE=SUBMIT VALUE=\" Check Out \">");
        out.println("</FORM>");
        out.println("</BODY></HTML>");
    }
}
```

Sessions avec Servlets

Objet `javax.servlet.http.HttpSession`

- ▶ Durée de vie limitée et réglable
- ▶ L'objet HttpSession est accessible via la méthode HttpServletRequest.getSession() :

```
HttpSession session = request.getSession();
```

- ▶ Une session est automatiquement créé pour chaque utilisateur qui se connecte au site : le programmeur n'a rien à faire.
 - Implémenté à l'aide de `cookies` temporaires (jsessionid)
 - ou par un `identifiant` passé en paramètre de toutes les requêtes HTTP.
- ▶ Une session se termine à l'initiative du programmeur, ou au bout d'un certain temps d'inactivité.

Accès aux données de la session

Stockage

- ▶ Le stockage d'une donnée dans la session :

setAttribute(String cle, Object valeur)

- ▶ La clé est de type String et la valeur de type Object.

- ▶ Exemple :

```
HttpSession session = request.getSession(true);  
session.setAttribute("visiteur", v);
```

Accès aux données de la session

Récupération

- ▶ Récupération d'une référence d'un objet en session :

`Object getAttribute(String clé)`

exemple :

`Visiteur v= (Visiteur)session.getAttribute("visiteur");`

- ▶ Remarque : `getAttributeNames()` : retourne la liste de toutes les clés de la session

Accès aux données de la session

Suppression

- ▶ La suppression d'un objet de la session :

`HttpSession.removeAttribute(String clé)`

exemple :

`session.removeAttribute("visiteur");`

Terminer une session

▶ 2 possibilités :

- soit la laisser s'expirer seule
- soit mettre fin à sa durée de vie en utilisant invalidate()

```
//récupération de la session  
HttpSession session = request.getSession();  
//fermeture de la session  
session.invalidate();
```

Méthodes supplémentaires de HttpSession

- ▶ **long getCreationTime()**
Retourne l'heure de création de la session mesurée en millisecondes depuis minuit le 01/01/1970.
- ▶ **java.lang.String getId()**
Retourne un String contenant l'identifiant unique associé à la session.
- ▶ **long getLastAccessedTime()**
Retourne la dernière heure où le client a envoyé une requête mesurée en millisecondes depuis minuit le 01/01/1970.
- ▶ **int getMaxInactiveInterval()**
Retourne le temps maximal (secondes) pendant lequel la session reste ouverte en l'absence de requêtes client.
- ▶ **void setMaxInactiveInterval(int interval)**
Fixe le temps maximal (secondes) pendant lequel la session reste ouverte en l'absence de requêtes client..

Connexions aux BDR

- ▶ Java dispose d'un package dédié à cette connexion : **java.sql ou JDBC pour Java DataBase Connectivity.**
- ▶ JDBC = ensemble de classes et d'interfaces permettant de réaliser des opérations sur des bases de données

un programme utilisant JDBC fonctionne de la façon suivante :

- 1 - Connexion avec la base
- 2 - Emission de requêtes
- 3 - Récupération des données
- 4 - Fermeture de la connexion.

Exemple complet d'une servlet accédant à une BD (driver de type 1)

```
import java.sql.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
  
public class ExempleBD extends HttpServlet{  
  
    public void doGet(HttpServletRequest request,HttpServletResponse  
                      response) throws ServletException,IOException  
    {  
        Connection con=null;  
        Statement stmt=null;  
        ResultSet rs=null;  
  
        response.setContentType("text/html");  
        PrintWriter out=response.getWriter();
```

Exemple complet d'une servlet accédant à une BD (driver de type 1)

```
try {  
    // enregistrement du driver  
    Class.forName(" com.mysql.jdbc.Driver ");  
    // on récupère la connexion  
    con=DriverManager.getConnection("jdbc:mysql://localhost/maBase","login","motdepasse");  
    //creation du statement  
    stmt=con.createStatement();  
    // execution d'une requête SQL, et obtention d'un resultSet  
    rs=stmt.executeQuery("SELECT nom,prenom FROM personne");  
    //affichage du resultat de la requete  
    out.println("<HTML>");  
    out.println("<BODY>");  
    out.println("<h1>Liste des Personnes</h1>");  
    while(rs.next())  
    {  
        out.println(rs.getString("nom")+" "+rs.getString("prenom"));  
    }  
    out.println("</BODY></HTML>");  
}
```

Exemple complet d'une servlet accédant à une BD (driver de type 1)

```
catch(ClassNotFoundException e)
{
    out.println("impossible de charger le driver de la BD :" +e.getMessage());
}
catch(SQLException e)
{
    out.println("SQLException générée :" +e.getMessage());
}
finally
{
    // TOUJOURS fermer la connexion à la BD
    try{ if(rs!=null) { rs.close(); } } catch(Exception e){}
    try{ if(stmt!=null) { stmt.close(); } } catch(Exception e){}
    try{ if(con!=null) { con.close(); } } catch(Exception e){}
}
} // fin methode
} // fin classe
```

Sécurité

- ▶ Attention aux failles XSS :

```
out.println(rs.getString("Nom")+" "+rs.getString("Prenom")); // NON !
```

Solution : utiliser la méthode statique proposée par apache :

```
String StringEscapeUtils.escapeHtml(java.lang.String str)  
--> échappera les caractères HTML en utilisant les HTML entities
```

(disponible dans le package **org.apache.commons.lang**)

Sécurité

▶ Comment faire ?

```
import org.apache.commons.lang.StringEscapeUtils;  
...  
out.println(  
    StringEscapeUtils.escapeHtml(rs.getString("Nom"))+ " "+  
    StringEscapeUtils.escapeHtml(rs.getString("Prenom"))  
);
```

Les Java Server Pages

Programmation JSP

Principe

Permet de combiner :

- contenu statique
- contenu dynamique

Pages (.jsp) comprenant à la fois du code HTML et des scripts "côté-serveur" écrits en Java

Les fichiers JSP sont transformés en Servlets

Exemples simples de pages JSP

Exemple 1

```
<html>
<head><title>exemple1</title></head>
<body>

<% String hello="Hello World !"; %>
<%= hello %>

</body>
</html>
```

Exemples simples de pages JSP

Exemple 2

```
<%-- définit les informations globales à la page --%>
<%@ page language="java" %>

<html>
<head><title> exemple 2 page JSP</title></head>
<body>
<%--Déclaration de la variable c --%>
<%! char c = 0; %>
<%-- Scriptlet (code java) %>
    <%
        for (int i = 0; i < 100; i++){ %}
            <%= i %>
            <br/>
        <% } %>
</body></html>
```

Les Java Beans

Un bean Java est une classe qui possède les caractéristiques de base suivantes :

- classe est **publique**
- un **constructeur public sans paramètre**
- des **getters / setters**

Syntaxe des balises JSP

Les balises JSP sont classées en 5 groupes :

1. Les directives
2. Les déclarations
3. Les scriptlets
4. Les expressions
5. Les actions :
 - `jsp:include`
 - `jsp:useBean`
 - `jsp:setProperty`
 - `jsp:getProperty`
 - `jsp:forward`

1-Les directives

Pour les informations **relatives à la page**

Syntaxe:

```
<%@ directive attribut1="valeur" attribut2="valeur"... %>
```

Les + courantes :

- page
- include
- taglib

directive page

Pour spécifier le langage des scriptlets (java, javascript) :

```
<%@ page language="java" %>
```

Pour importer classes :

```
<%@ page import="java.util.* ,java.io.*" %>
```

Pour spécifier le MIME-Type (text/html par défaut) :

```
<%@ page contentType="text/plain" %>
```

Pour utiliser ou non les sessions (true par défaut) :

```
<%@ page session="false" %>
```

directive page

Pour spécifier une page d'erreur :

```
<%@ page errorPage="url" %>
```

Pour dire si la page est une page d'erreur

```
<%@ page isErrorPage="true" %>
```

autres attributs possibles :

implements

extends

...

directive include

Syntaxe:

```
<%@ include file="relative url" %>
```

Exemple :

```
<html>
<body>
<!-- inclut une barre de navigation -->
<%@ include file="/barre.html" %>

</body>
</html>
```

L'insertion se fait **au moment de la traduction de la page JSP en servlet** (on peut donc mettre du Java)

2-Les déclarations

Cette balise permet de définir des méthodes ou des attributs au niveau de la servlet

Syntaxe:

<%! Code Java %>

Exemple:

<!-- déclaration d'un attribut -->

<%! private int compteur = 0; %>

<!-- déclaration de méthode -->

<%! public String hello() { return "hello"; } %>

Nombre d'accès à cette page :

<%= ++compteur %>

3-Les scriptlets

Les scriptlets permettent d'insérer n'importe quel code Java

Syntaxe :

<% code Java %>

3-Les scriptlets

Exemple :

```
<% if (Math.random() < 0.5) { %>
    Vous avez gagné!
<% } else { %>
    Vous avez perdu!
<% } %>
```

La servlet obtenue après compilation sera alors :

```
if (Math.random() < 0.5) {
    out.println("gagné !");
} else {
    out.println(" perdu !");
}
```

4-Les expressions

Les expressions servent à écrire dans le flot de sortie
(page réponse)

Syntaxe :

<%= expression %>

Exemple:

<% int a=5; %>

La valeur de a est : <%= a %>

Rq : <%=a%> sera traduit par out.println(a);

C'est pour cette raison qu'il n'y a pas de ; à la fin

5-Les actions

jsp:include - Inclut un fichier lors de l'exécution de la requête

jsp:useBean - Retrouve ou instancie un Bean

jsp:setProperty – appel d'un setteur d'un Bean

jsp:getProperty – appel d'un getter d'un Bean

jsp:forward - Redirige la requête vers une autre page

Action jsp:include

Syntaxe:

```
<jsp:include page="relative URL" flush="true" />
```

Cette action insère le fichier lors de l'exécution de la requête

Exemple :

```
<jsp:include page="page1.html" flush="true"/>
<jsp:include page="page2.html" flush="true"/>
```

Action jsp:forward

Syntaxe:

```
<jsp:forward page="relative URL" />
```

Permet de faire une redirection

Exemple:

```
<% String pageSuivante = "/catalogue/accueil.jsp"; %>
```

```
<jsp:forward page="<%= pageSuivante %>" />
```

jsp:useBean

Syntaxe:

```
<jsp:useBean id="visiteur" class="magasin.Visiteur"  
scope="session" />
```

Si une instance associée à la référence id n'est pas trouvée dans la portée, une nouvelle instance est créée.

La portée du bean peut avoir quatre valeurs possibles :

application: bean visible de toutes les pages, pour toutes les sessions

session: bean visible de toutes les pages pour une session considérée

request: bean visible de toutes les pages participant à une requête

page: bean visible dans la page

jsp:setProperty

Cette action permet de fixer la valeur des attributs

Syntaxe :

```
<jsp:setProperty name="visiteur" property="nom" param="Jean" />
```

Fixe l'attribut de visiteur à la valeur "Jean"

jsp:getProperty

Cette action permet de récupérer la valeur d'un attribut du bean

Synthaxe :

```
<jsp:getProperty name="visiteur" property="prenom" />
```

Les objets implicites en JSP

request : correspond à l'objet HttpServletRequest

response : correspond à l'objet HttpServletResponse

out : utilisé pour écrire dans la page réponse, c'est le flot de sortie (PrintWriter)

session : c'est l'objet session (s'il existe)

Exemple 1 d'utilisation de Beans

```
//Le code source Java du bean :  
public class Visiteur {  
  
    private String nom;  
  
    public Visiteur(){  
    }  
  
    public String getNom() {  
        return nom;  
    }  
  
    public void setNom(String nom) {  
        this.nom = nom;  
    }  
}
```

Exemple d'utilisation de Beans

La page JSP

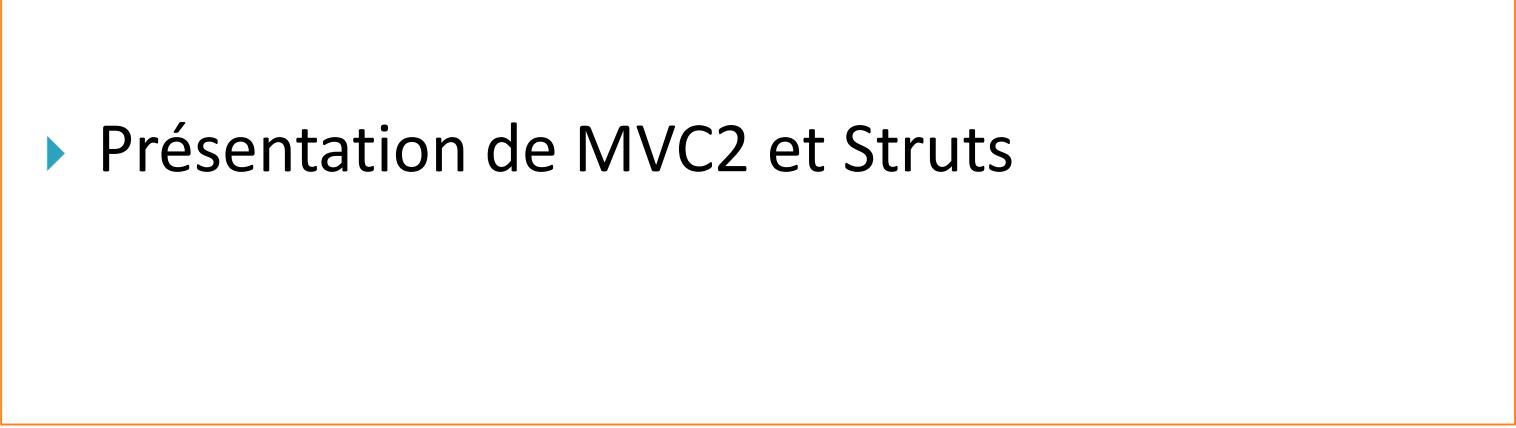
```
<html>
<jsp:useBean id="test" class="Visiteur">

<jsp:setProperty name="test" property="nom" value="toto" />
```

Le nom du visiteur est :

```
<jsp:getProperty name="test" property="nom" />

</html>
```



▶ Présentation de MVC2 et Struts

C'est quoi ?

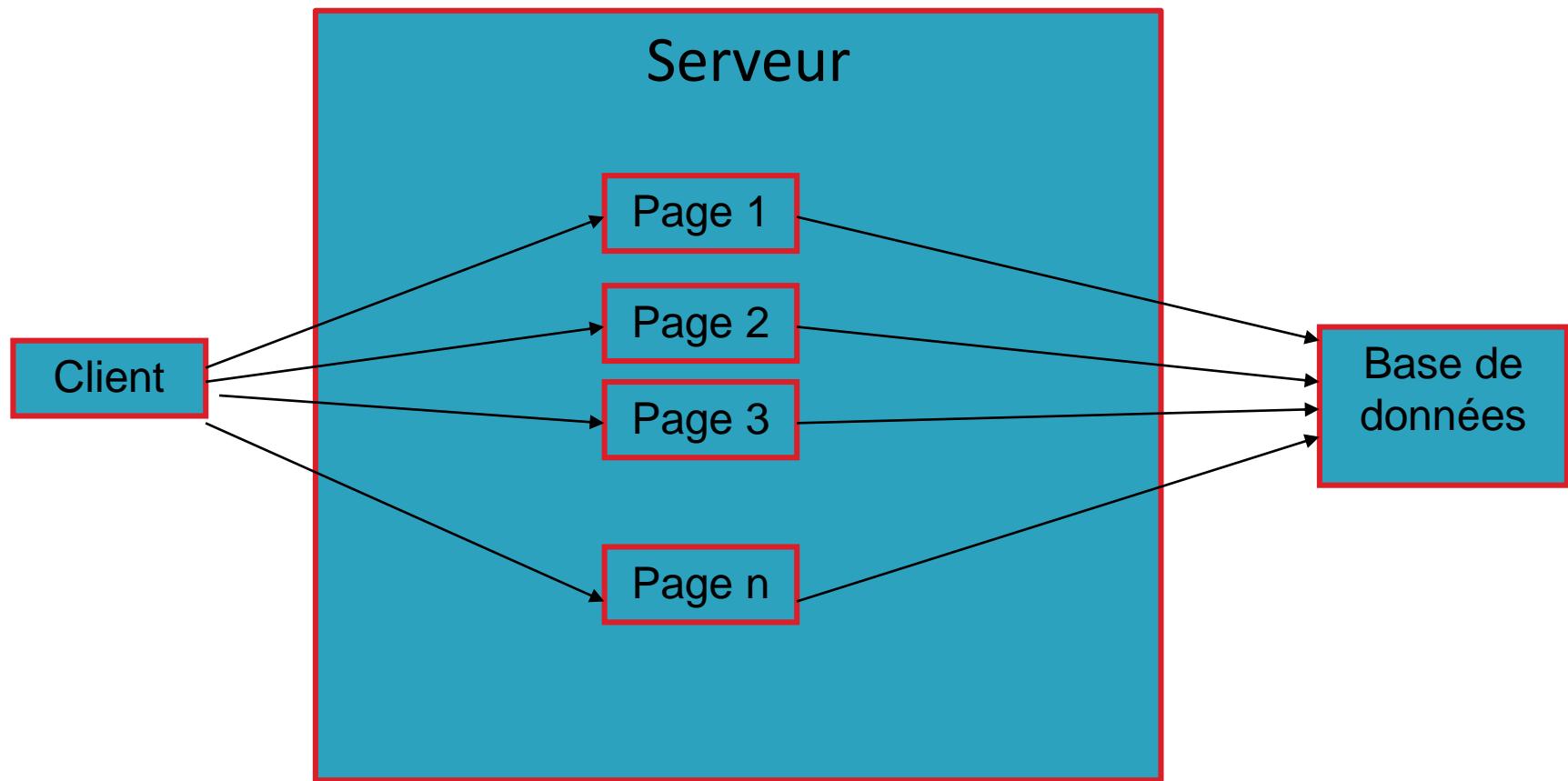
- ▶ Apache Struts est un framework permettant la création d'applications WEB en JAVA
- ▶ Gratuit
- ▶ Open source
- ▶ S'appuie sur le modèle MVC2

Pourquoi ?

- ▶ Souvent les pages WEB (jsp, php, aspx, ...) contiennent du code permettant :
 - la connexion à une base de données
 - l'envoi de requêtes SQL vers une base
 - de faire des redirections de pages, des contrôles de flux, du code métier, ...
 - ...

Cela rend les grosses applications très difficiles à maintenir !

Exemple d'application mal conçue



Avantages/Inconvénients

▶ +

Rapide à réaliser si peu de pages

▶ - - -

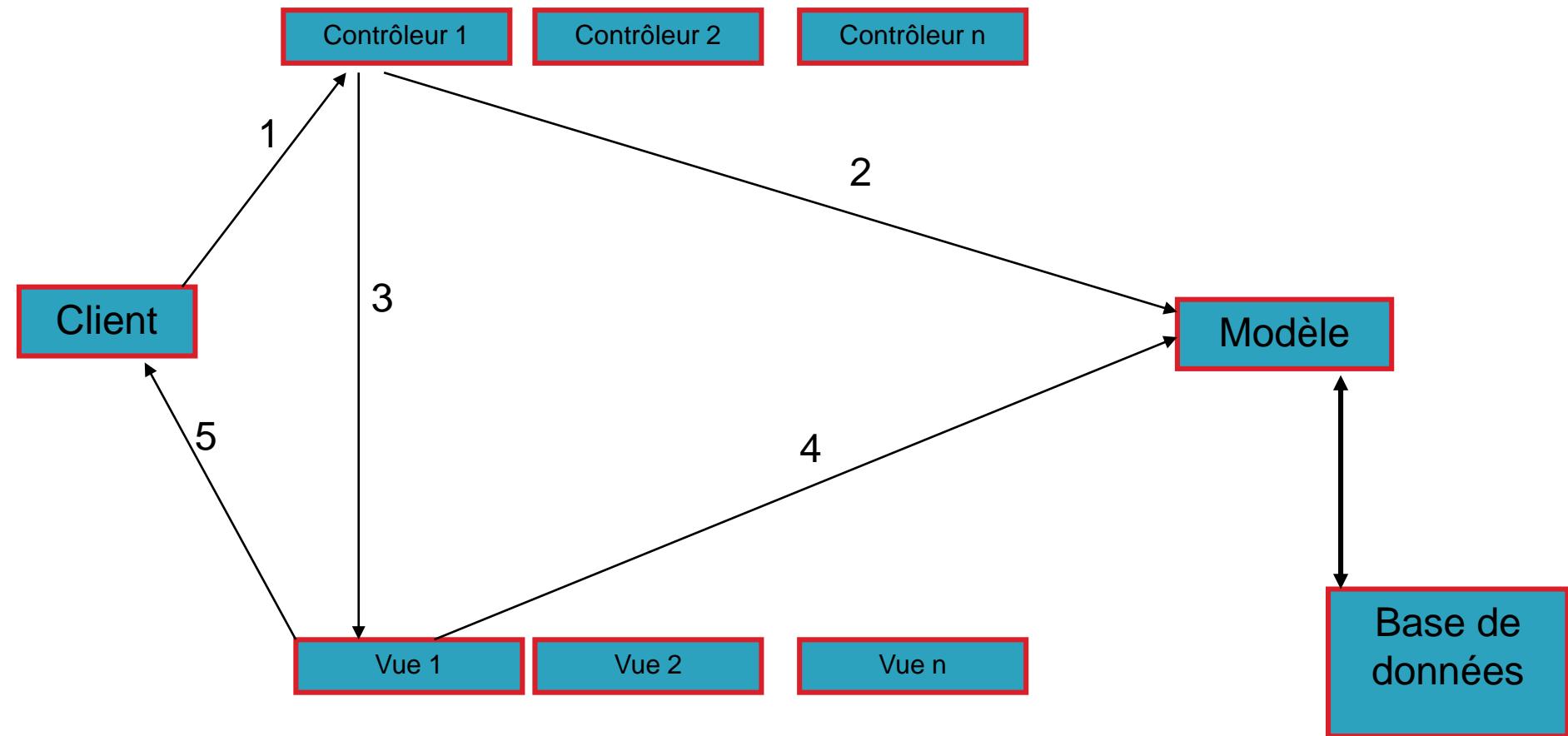
non maintenable dans la durée si grosse application et/ou plusieurs développeurs

→ Architecture à oublier !!!

Solution ?

- ▶ Il faut séparer tous ces concepts
- ▶ Une solution : Architecture Model View Controller (MVC ou MVC2)
 - Model = métier + base de données
 - View = pages web
 - Controller = navigation + contrôle

Architecture MVC (1)



Avantages/Inconvénients

▶ +++

Séparation claire des parties Modèle, Vue et Contrôleur

Possibilité de remplacer une partie sans impact sur les autres

Maintenance aisée

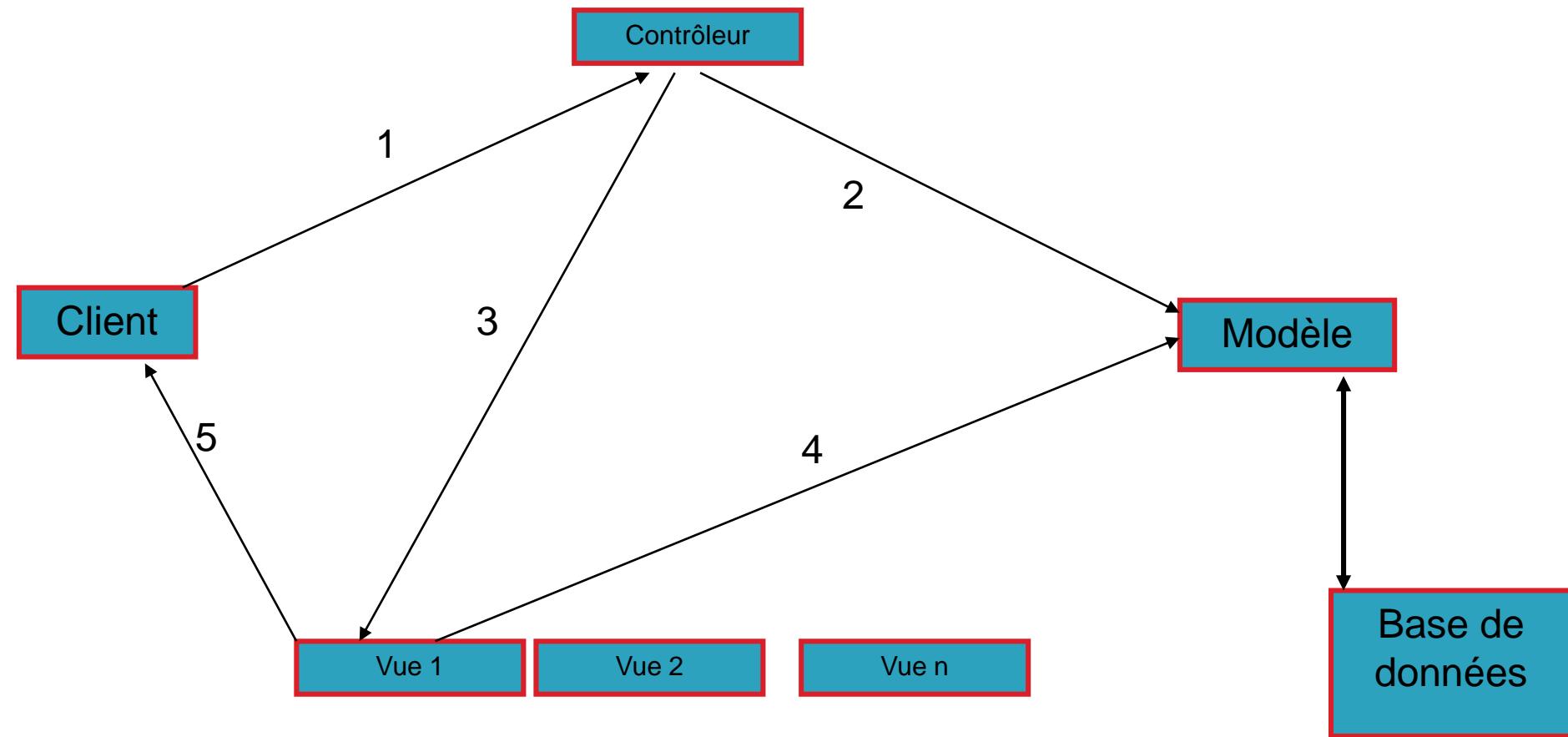
▶ -

Mise en place du projet plus difficile

Il est nécessaire de coder plusieurs Contrôleurs...

(1 par vue)

Architecture MVC 2



Avantages/Inconvénients

▶ +++

Séparation claire des parties Modèle, Vue et Contrôleur

Possibilité de remplacer une partie sans impact sur les autres

Maintenance aisée

▶ -

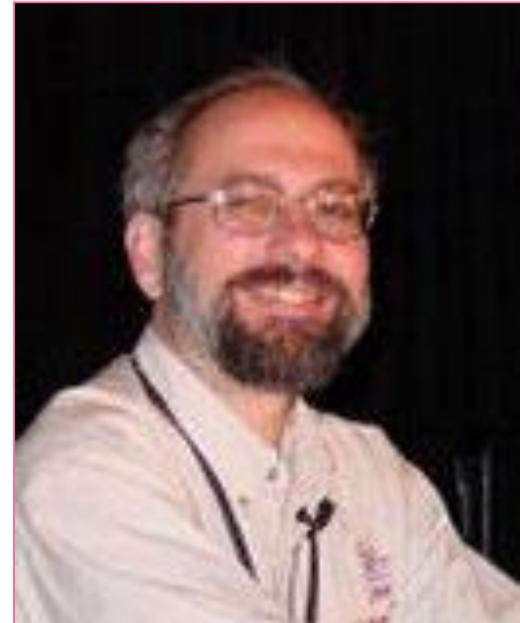
Mise en place du projet difficile

Et Struts ?

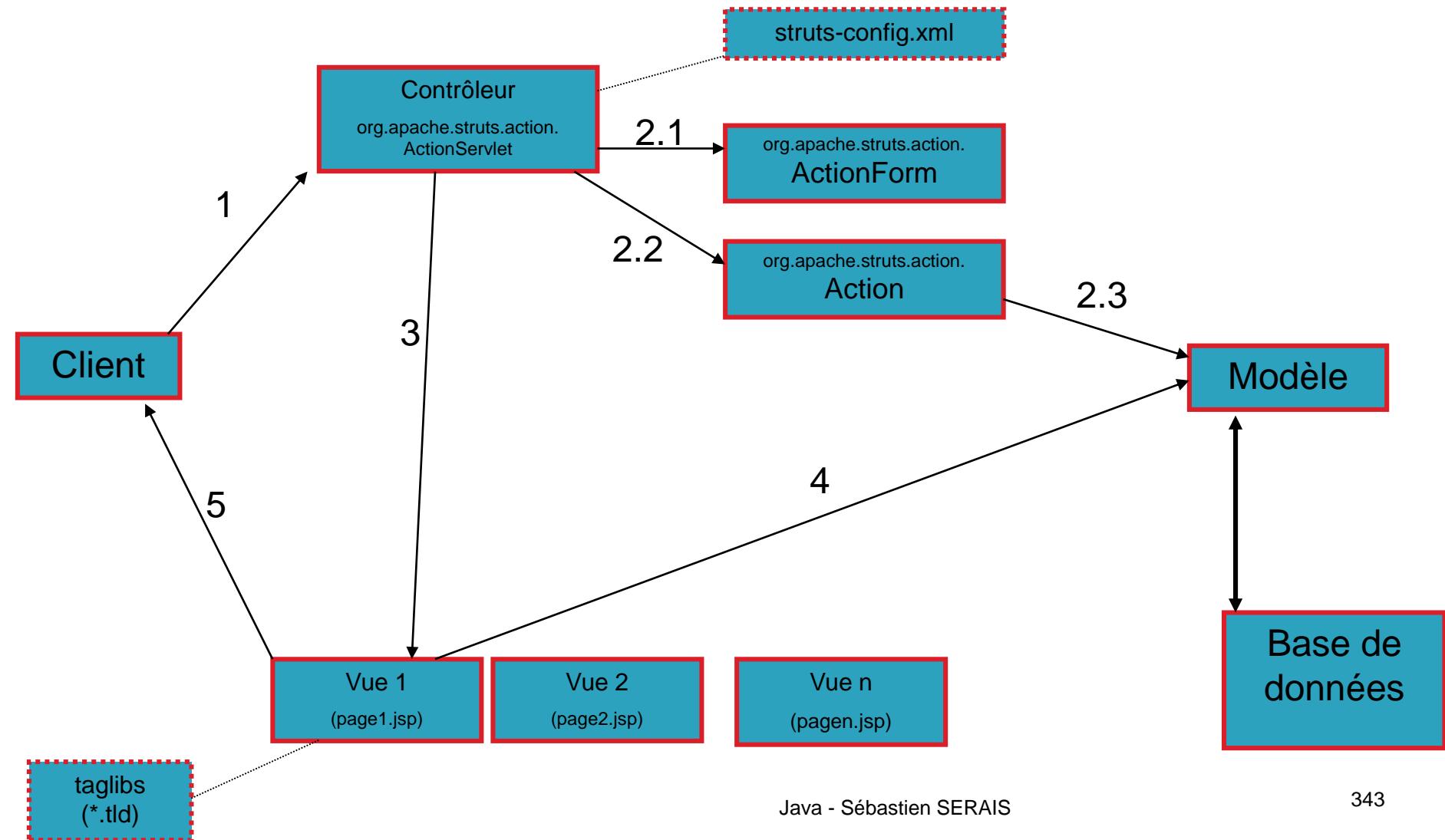
- ▶ Struts s'appuie sur l'architecture MVC2

Origine de Struts

- ▶ Développé à l'origine par Craig McClanahan puis offert à Apache en mai 2000



Architecture de Struts (1) (MVC2)



2 versions



- ▶ Struts 1 est la version la plus connue et la plus utilisée (2000)
 - meilleur choix pour les problèmes les plus courants
- ▶ Struts 2 (2007) (ancien WebWork 2)
 - solution pour problèmes plus complexes

Pour aller plus loin

- ▶ Struts : Framework couche présentation
- ▶ Maven : gestion de dépendances
- ▶ JUnit : pour les tests unitaires
- ▶ Spring : Inversion of Control
- ▶ Hibernate : Mapping O/R
- ▶ ...

Les Tests Unitaires en JAVA

Le Framework JUnit 4

Problématique :

▶ Les logiciels sont souvent générateurs
d'erreurs, de bugs dus à des spécifications
insuffisantes et/ou des applications peu/mal testées....

Voici quelques rappels...

Premier BUG (09/09/1945)



- ▶ 9 septembre 1945

premier bug de l'histoire informatique

- ▶ CAUSE :

- une mite bloquée dans le calculateur Mark II de l'Université de Harvard aux USA

--> C'est l'origine du mot "bug" (insecte en anglais)

Le **Harvard Mark II** était le calculateur le plus puissant de l'époque :

- une multiplication ou une addition prenait environ 1 seconde
 - une racine carrée prenait 5 secondes

Bourse de Tokyo (01/11/05)



- ▶ Bug qui a empêché l'ouverture à 9h00 GMT de la bourse en **bloquant toutes les cotations**. (...)
- ▶ Il s'agirait du plus important chaos boursier qu'aït connu la bourse depuis août 1997.
(...) Le marché financier de Tokyo représente un volume quotidien moyen de transactions estimé à 13,2 milliards de dollars.
- ▶ **CAUSE : Trop gros volume de transactions à traiter**

source <http://www.generation-nt.com>

Terminaux des gares (04/12/04)



- ▶ Panne informatique qui a paralysé vendredi, à la veille du week-end, 800 terminaux de vente aux guichets des gares, qui ne pouvaient émettre de billets.
- ▶ CAUSE : Algorithme défectueux qui a progressivement contaminé les terminaux de vente en gare

source : http://www.mines.inpl-nancy.fr/~tisseran/cours/qualite-logiciel/qualite_logiciel.html

Mission Vénus



- ▶ Passage à 5 000 000 de Km de la planète, au lieu de 5 000 Km prévus

CAUSE : Une virgule a été remplacée par un point (au format US des nombres)

Le bogue de l'an 2000



- ▶ La lutte contre le bogue de l'an 2000 a coûté à la France 500 milliards de francs.

CAUSE : L'année était codée sur deux caractères, pour gagner de la place

Ariane V (04/06/96)



- ▶ Premier lancement de Ariane V, explosion en vol
- ▶ Coût du programme d'étude d'Ariane V : 38 milliards de Francs.
CAUSE : Logiciel de plate forme inertielle repris tel quel d'Ariane IV sans nouvelle validation.

Ariane V ayant des moteurs plus puissants s'incline plus rapidement que Ariane IV, pour récupérer l'accélération due à la rotation de la Terre.

Les capteurs ont bien détecté cette inclinaison d'Ariane V, mais le logiciel l'a jugée non conforme au plan de tir (d'Ariane IV), et a provoqué l'ordre d'auto destruction.

En fait tout se passait bien !

- ▶ (source : http://www.mines.inpl-nancy.fr/~tisseran/cours/qualite-logiciel/qualite_logiciel.html)

Sécurité Sociale (2001)



- ▶ Un bug informatique aurait coûté 10 millions d'euros à la Sécurité Sociale, selon le Parisien. De nombreuses cliniques ont été remboursées deux fois.

CAUSE : bug dans le nouveau logiciel de télétransmission des demandes de remboursement.

- ▶ «la plupart des cliniques ont maintenant accepté de rembourser». Seule une dizaine d'entre elles contestent le remboursement et ont porté l'affaire devant les tribunaux des affaires sociales.

▶ (source : <http://www.liberation.fr>)

FDJeux (27/02/07)



FRANÇAISE DES JEUX

- ▶ Un bug informatique joue un vilain tour aux joueurs de La Française des Jeux

Environ 25 000 parieurs qui ont joué en ligne au Joker Plus vont être remboursés. Le numéro 9 ne sortait jamais dans les combinaisons.

Le chiffre 9 n'apparaissait jamais parmi les 7 numéros attribués automatiquement par un système Flash aux joueurs en ligne.

Les 25 000 joueurs lésés séduits par le Joker+ sur Internet seront dédommagés.

- ▶ CAUSE : oubli du chiffre 9 dans la matrice de tirage aléatoire du logiciel

(source : <http://www.01net.com>)

BNP Paribas (26 février 2009)



- ▶ Des milliers de clients, particuliers et entreprises, ont été débités plusieurs fois d'un même chèque, virement ou prélèvement.
- ▶ Ce bug concerne près de 586.000 opérations.
- ▶ CAUSE : BUG informatique

(source : <http://www.01net.com>)

Caisse Nationale d'Assurance Vieillesse (CNAV) (juin 2009)

- ▶ Erreur de programmation remontant à 1984 s'est traduite par un calcul erroné des retraites pour plusieurs millions de salariés
- ▶ Mauvaise comptabilisation des trimestres dans le calcul des pensions de retraite
- ▶ Coût du bug estimé à 1,8 milliard d'euros



BlackBerry

▶ BlackBerry, le bug qui fait tâche

Lundi 10 octobre 2011, le groupe Canadien RIM connaît la plus grosse panne de son histoire. La faute à un problème d'engorgement qui entraîne ni plus ni moins qu'une paralysie du réseau mondial de son téléphone BlackBerry. Messagerie texte inactive, messagerie instantanée bloquée, navigateur internet en rade, environ 70 millions d'utilisateurs sont touchés à travers le monde. Le tout en pleine campagne de lancement de son concurrent, l'I-Phone 4S...

IPhone / Ipad (11 février 2016)

- ▶ si un utilisateur fixe la date, sur un iPhone ou un iPad, au 1er janvier 1970, le téléphone... meurt (Processeurs A7 et supérieurs 64 bits)
- ▶ <https://www.youtube.com/watch?v=fY-ahR1R6IE>
- ▶ Cause : Division par zéro ?

bug de l'An 2038 (19/01/2038)

- ▶ Sur la majorité des systèmes d'exploitation actuels :
le temps = nb de secondes depuis le 01/01/1970 et codé
sur **32 bits** (maxi = 2 147 483 647 secondes)
- ▶ la limite sera atteinte le **19/01/2038 !!**

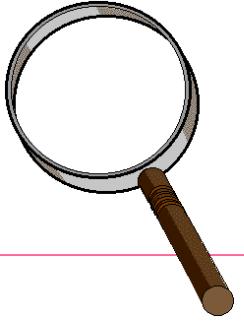


Qualité



- ▶ Nécessité d'améliorer la qualité logicielle
- ▶ Mais Comment ?

Qualité



- ▶ Définition :
- ▶ Degré auquel un système, un composant ou un processus satisfait les **besoins spécifiés**
- ▶ Degré auquel un système, un composant ou un processus satisfait les attentes du **client** ou les besoins des **utilisateurs**

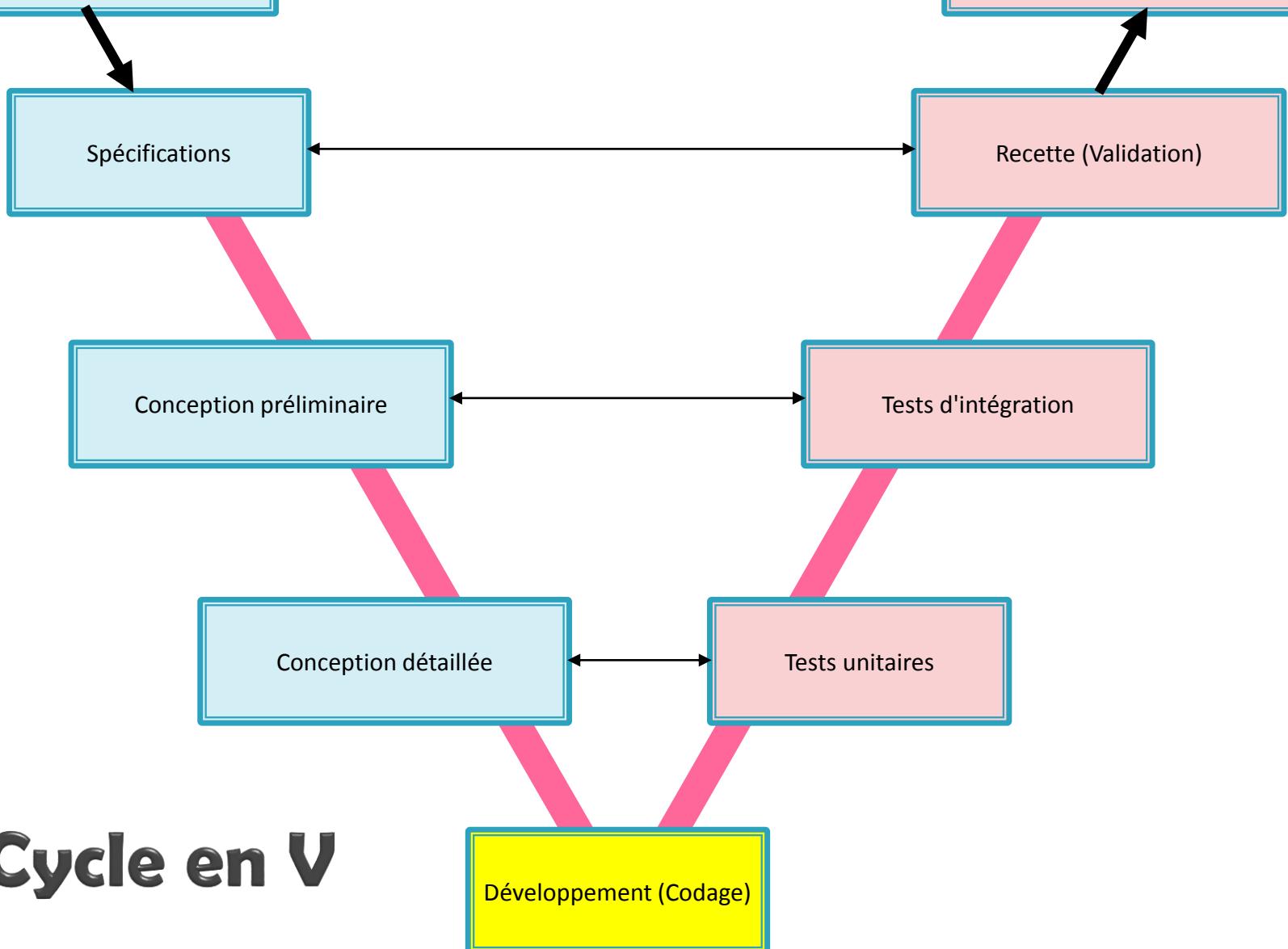
Qualité



- ▶ Afin d'améliorer au maximum la qualité logicielle, il est nécessaire d'insister sur les phases de **spécification/conception** et de **tests**
- ▶ Assurance qualité et cycle de vie sont fortement liés
- ▶ Le **génie logiciel** désigne l'ensemble des méthodes, des techniques et outils concourant à la production d'un logiciel

**Expression des besoins
du client**

**Mise en production
+
maintenance**



Le Cycle en V

Les types de tests

- ▶ Tests unitaires
- ▶ Tests d'intégration
- ▶ Tests vibratoires
- ▶ Tests CEM
- ▶ Tests fonctionnels
- ▶ Tests IHM
- ▶ Tests de montée en charge (réseau, CPU, mémoire,...)
- ▶ Test de performance (temps de réponse,...)
- ▶ ...

Les Tests Unitaires en JAVA

Le Framework JUnit 4

Pourquoi les tests unitaires ?

- ▶ De nombreuses applications mal testées → nombreux bugs
- ▶ Nécessité de mieux tester les applications
- ▶ Insister sur les tests et notamment les tests unitaires du cycle en V

Inconvénients (*a priori...*)

- ▶ Coûteux en temps
- ▶ Peut paraître inutile (au premier abord)
- ▶ Cela ne donne pas l'impression d'avancer sur le projet

Avantages

- ▶ Garantissent une **stabilité** et **fiabilité** du logiciel → augmentent la qualité logicielle
- ▶ Forcent à penser aux **cas particuliers** dès le départ et évitent ainsi d'avoir à ajouter des mises à jour après coup (ce qui est plus coûteux !)
- ▶ Le temps perdu est largement récupéré à la fin (lors du débugage ou évolutions du logiciel)
- ▶ Permettent de s'assurer de la **non régression** du logiciel (tests de non régression)

JUnit

- ▶ JUnit = framework permettant de :
 - Rédiger des tests unitaires
 - Exécuter des tests unitaires
- ▶ Développé en Java par Kent Beck et Erich Gamma

Rq:

Kent Beck est aussi l'inventeur du concept d'eXtreme Programming (XP)

Erich Gamma: initiateur du projet Eclipse



JUnit

- ▶ Pas besoin d'interprétations humaines des résultats
- ▶ 3 Résultats possibles :
 - Succès du test
 - Échec (anticipé)
 - Erreur (imprévue)
- ▶ Possibilité de lancer simultanément une grande quantité de tests

Principe

- ▶ Créer pour chaque classe une classe de test :
 - MaClasse --> MaClasse**Test**
- ▶ Créer pour chaque méthode au moins une méthode de test :
 - maMethode(...) --> **testMaMethode(...)**

Exemple

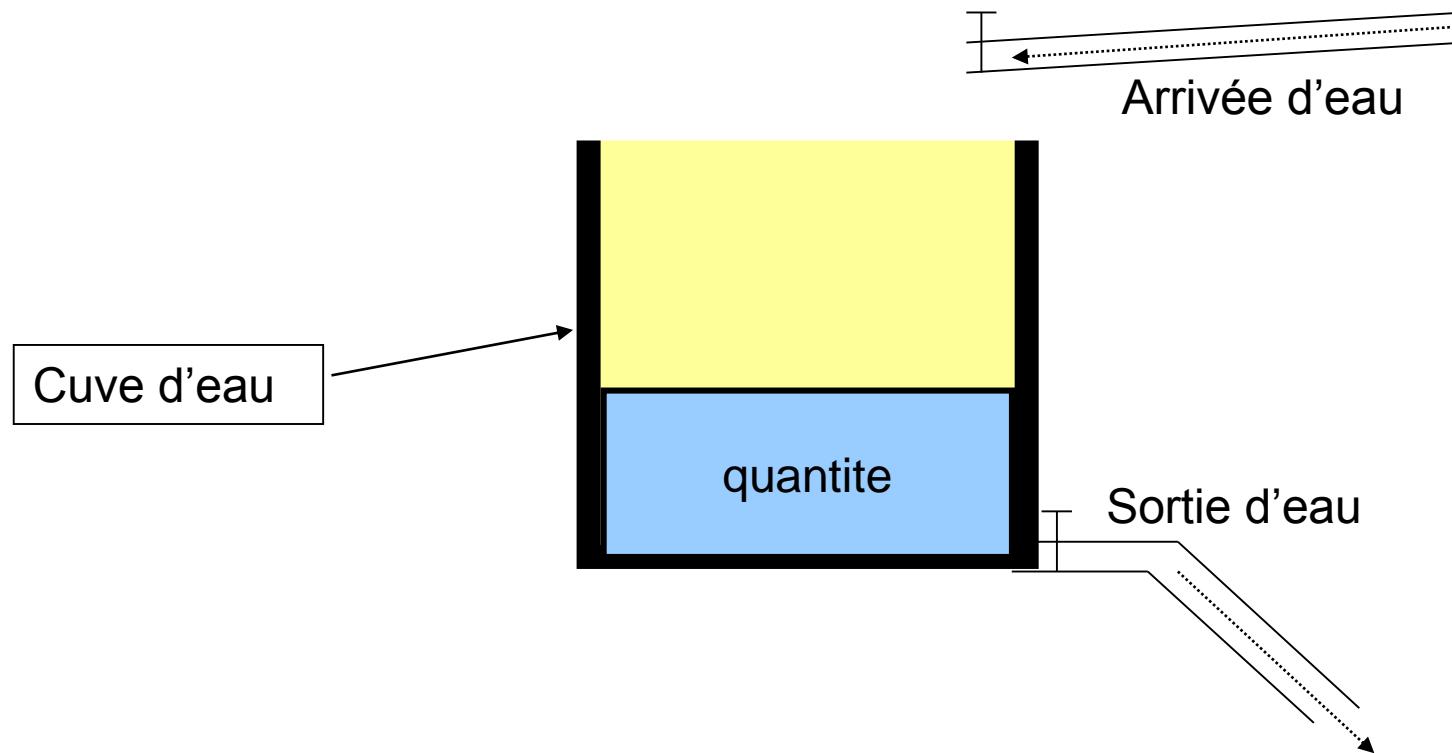
- ▶ Créer un projet Java
- ▶ Supprimer le répertoire source src
- ▶ Créer les 2 répertoires sources :
 - src/main/java (pour les fichiers sources)
 - src/test/java (pour les tests unitaires)

Ces 2 répertoires ne sont pas obligatoires, mais cela permet de respecter l'architecture par défaut de Maven 2

Exemple

- ▶ Créer le package tutoJUnit dans les 2 répertoires sources
- ▶ Puis écrire la classe à tester : Cuve

Modélisation d'une Cuve à eau



Cuve.java

```
package tutoJUnit;

public class Cuve {

    double quantite;
    double quantiteMax;

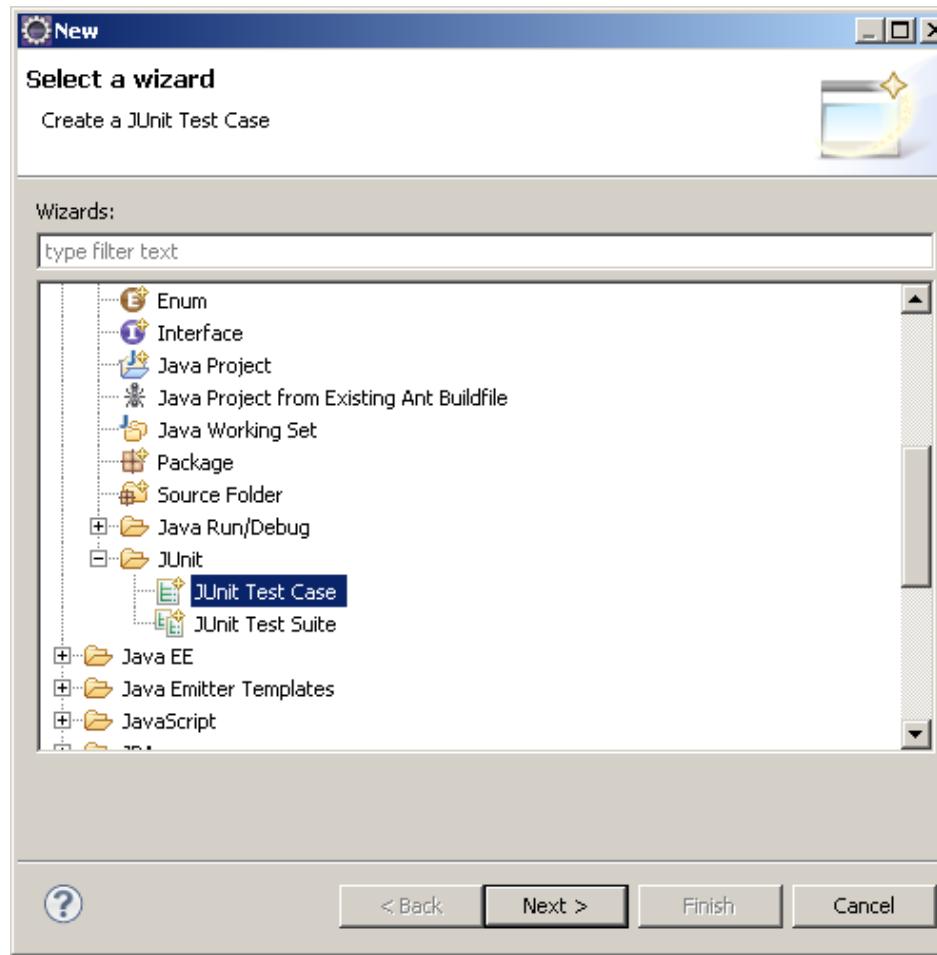
    public Cuve(double quantite, double quantiteMax) {
        this.quantite = quantite;
        this.quantiteMax = quantiteMax;
    }

    public void remplir(double quantite){
        this.quantite+=quantite;
    }

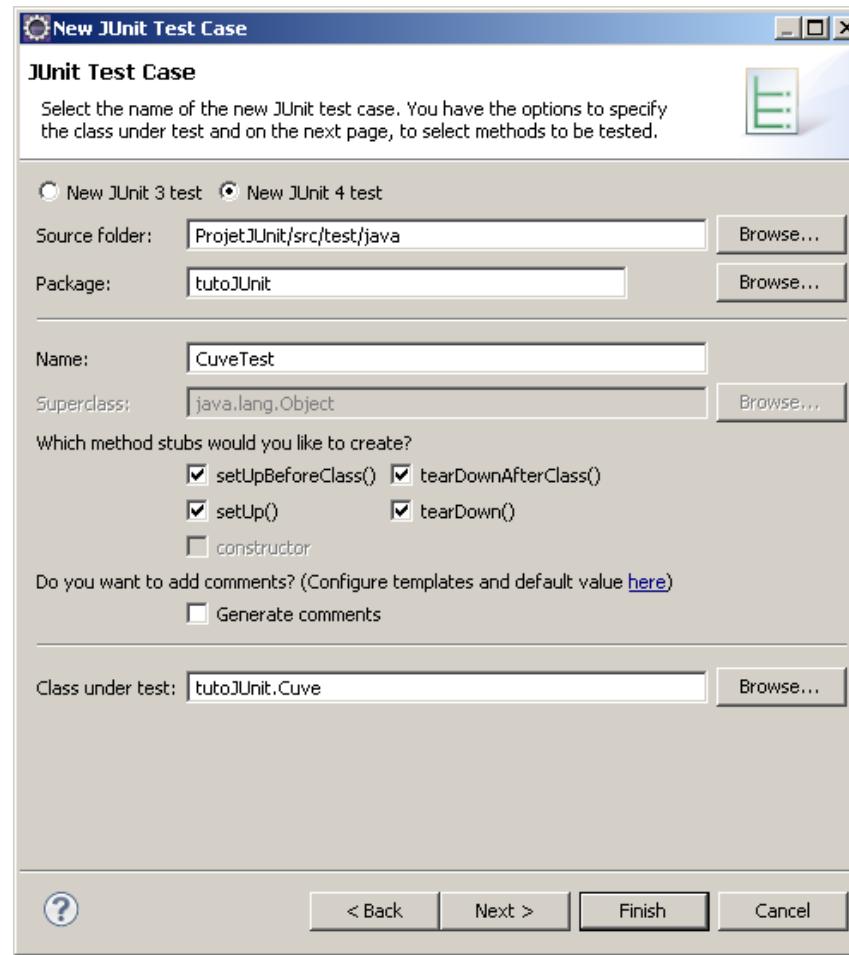
    public void vider(double quantite){
        this.quantite-=quantite;
    }

    public String toString(){
        return quantite+" "+quantiteMax;
    }
}
```

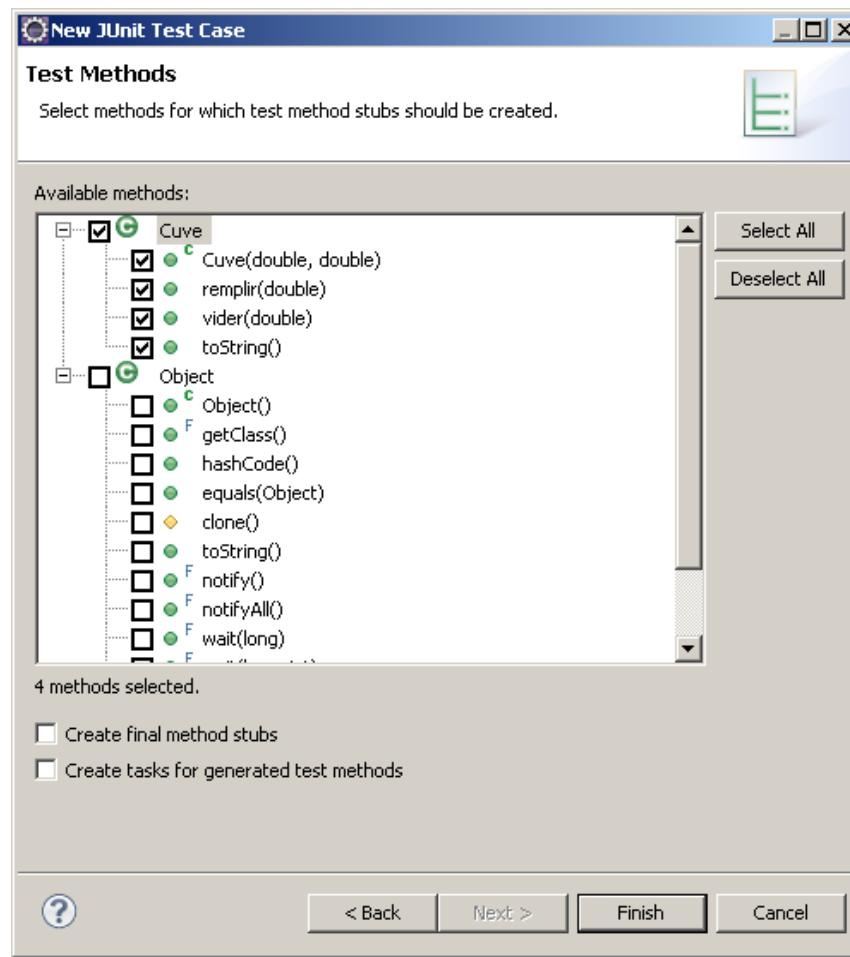
Création de la classe de Test



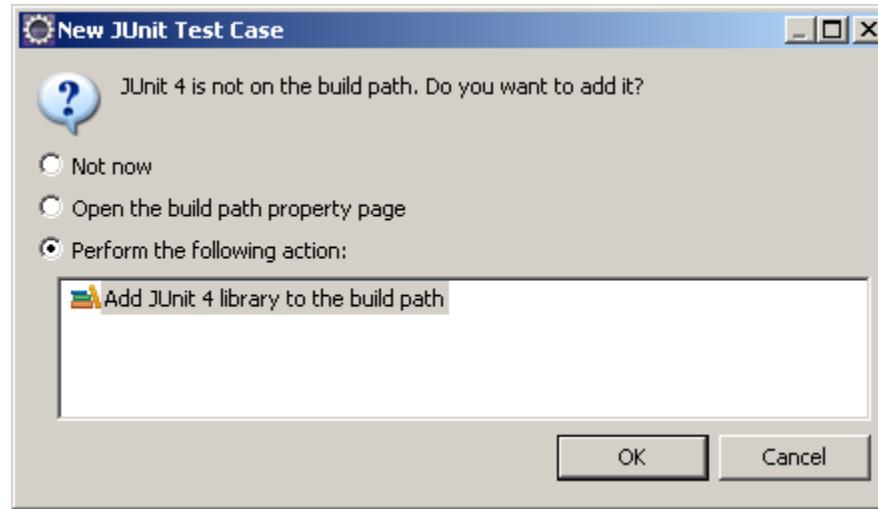
Création de la classe de Test



Sélection des méthodes à tester



Ajout de JUnit 4 au projet



Classe de Test générée : CuveTest

ajouter des affichages dans toutes les méthodes

```
package tutoJUnit;
import static org.junit.Assert.*;
import org.junit.*;
public class CuveTest {
    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
        System.out.println("setUpBeforeClass");
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
        System.out.println("tearDownAfterClass");
    }
    @Before
    public void setUp() throws Exception {
        System.out.println("setUp");
    }
    @After
    public void tearDown() throws Exception {
        System.out.println("tearDown");
    }
    @Test
    public void testCuve() {
        System.out.println("testCuve");
        fail("Not yet implemented");
    }
    @Test
    public void testRemplir() {
        System.out.println("testRemplir");
        fail("Not yet implemented");
    }
    @Test
    public void testVider() {
        System.out.println("testVider");
        fail("Not yet implemented");
    }
    @Test
    public void testToString() {
        System.out.println("testToString");
        fail("Not yet implemented");
    }
}
```

Exécution du Test

- ▶ Run As... JUnit Test (en utilisant le Eclipse Junit Launcher)
- ▶ Résultat :

setUpBeforeClass

setUp

testCuve

tearDown

setUp

testRemplir

tearDown

setUp

testVider

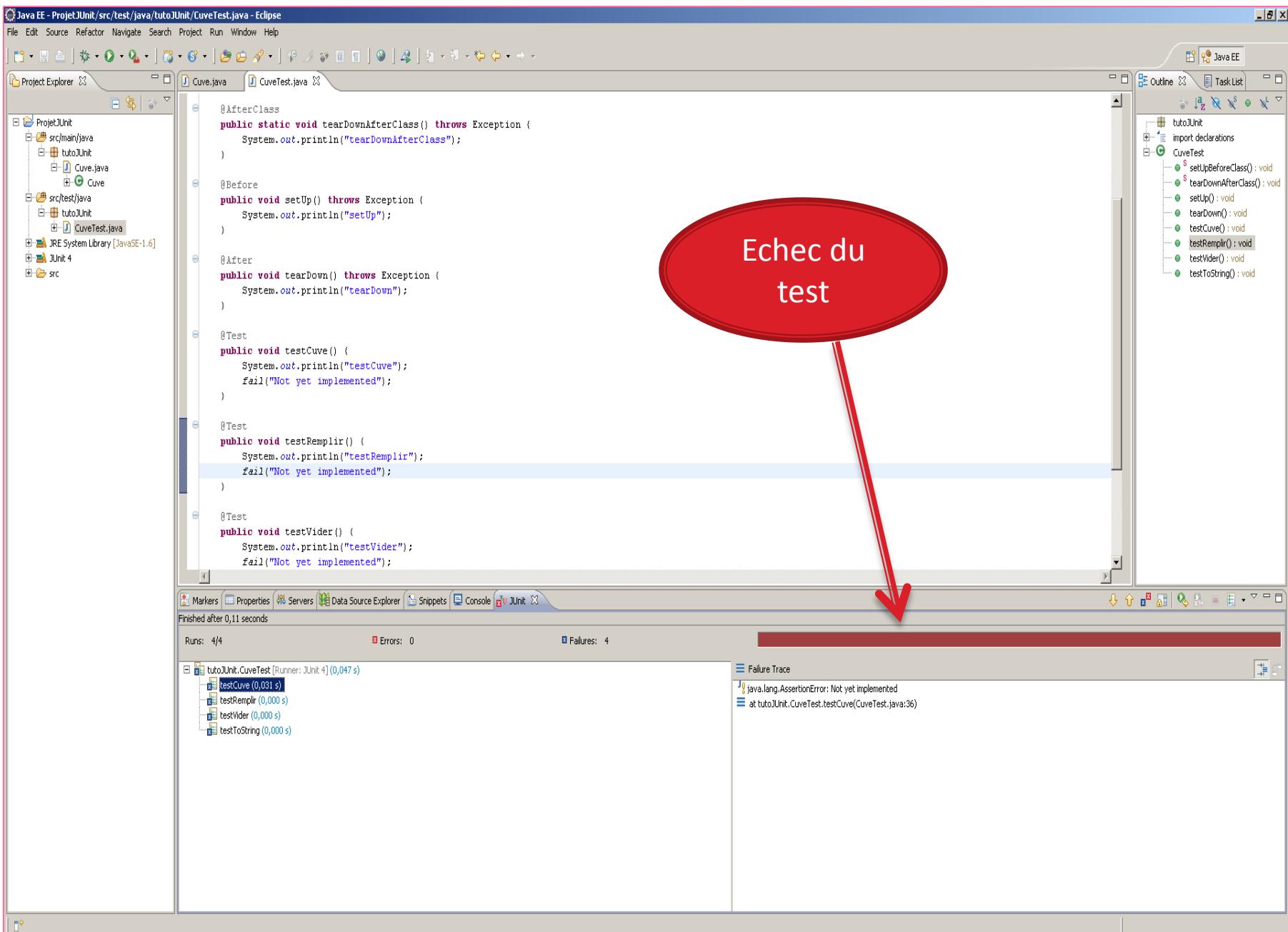
tearDown

setUp

testToString

tearDown

tearDownAfterClass



Ecriture de la classe de Test

- ▶ En utilisant les méthodes suivantes de la classe Assert :
 - `assertEquals` : Vérifie que deux objets sont égaux
 - `assertFalse` : Vérifie que l'expression est fausse
 - `assertNotNull` : Vérifie que l'objet n'est pas nul
 - `assertNotSame` : Vérifie que deux références ne sont pas les mêmes
 - `assertNull` : Vérifie qu'un objet est nul
 - `assertSame` : Vérifie que deux références sont les mêmes
 - `assertTrue` : Vérifie que l'expression est vraie
 - `fail` : Provoque l'échec du test
- ▶ Ecrire les méthodes de la classe de Test

Méthodes de test

```
@Test  
public void testCuve() {  
    Cuve c=new Cuve(100.0,200.0);  
    Assert.assertEquals(100.0,c.quantite,0.001);  
    Assert.assertEquals(200.0,c.quantiteMax, 0.001);  
}  
  
@Test  
public void testRemplir() {  
    Cuve c=new Cuve(100.0,200.0);  
    c.remplir(50.0);  
    Assert.assertEquals(150.0,c.quantite, 0.001);  
}
```

Java EE - ProjetJUnit/src/test/java/tutoJUnit/CuveTest.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer Cuve.java CuveTest.java

```
ProjetJUnit
  src/main/java
    tutoJUnit
      Cuve.java
      Cuve
  src/test/java
    tutoJUnit
      CuveTest.java
  JRE System Library [JavaSE-1.6]
  JUnit 4
  src
```

Cuve.java

```
  @AfterClass
  public static void tearDownAfterClass() throws Exception {
  }

  @Before
  public void setUp() throws Exception {
  }

  @After
  public void tearDown() throws Exception {
  }

  @Test
  public void testCuve() {
    Cuve c=new Cuve(100.0,200.0);
    Assert.assertEquals(c.quantite,100.0);
    Assert.assertEquals(c.quantiteMax,200.0);
  }

  @Test
  public void testRemplir() {
    Cuve c=new Cuve(100.0,200.0);
    c.remplir(50.0);
    Assert.assertEquals(c.quantite,150.0);
  }

  @Test
  public void testVider() {
```

Outline Task List

```
tutoJUnit
  import declarations
  CuveTest
    setUpBeforeClass()
    tearDownAfterClass()
    setUp()
    tearDown()
    testCuve()
    testRemplir()
    testVider()
    testToString()
```

Markers Properties Servers Data Source Explorer Snippets Console JUnit

Finished after 0,094 seconds

Runs: 4/4 Errors: 0 Failures: 2

tutoJUnit.CuveTest [Runner: JUnit 4] (0,047 s)

- testCuve (0,016 s)
- testRemplir (0,000 s)
- testVider (0,015 s)
- testToString (0,000 s)

Failure Trace

```
java.lang.AssertionError: Not yet implemented
  at tutoJUnit.CuveTest.testVider(CuveTest.java:52)
```

Writable Smart Insert 47 : 6

Test plus complet !

```
@Test  
public void testRemplir() {  
    Cuve c=new Cuve(100.0,200.0);  
    c.remplir(50.0);  
    Assert.assertEquals(150.0,c.quantite, 0.001);  
    c.remplir(300.0);  
    Assert.assertEquals(200.0,c.quantite, 0.001);  
}
```

Java EE - ProjetJUnit/src/test/java/tutoJUnit/CuveTest.java - Eclipse

File Edit Source Refactor Navigate Project Run Window Help

Project Explorer Cuve.java CuveTest.java Outline Task List

```
Project JUnit
  src/main/java
    tutoJUnit
      Cuve.java
      CuveTest.java
  src/test/java
    tutoJUnit
      CuveTest.java
  JRE System Library [JavaSE-1.6]
  JUnit 4
  src
```

```
  @AfterClass
  public static void tearDownAfterClass() throws Exception {
  }

  @Before
  public void setUp() throws Exception {
  }

  @After
  public void tearDown() throws Exception {
  }

  @Test
  public void testCuve() {
    Cuve c=new Cuve(100.0,200.0);
    Assert.assertEquals(c.quantite,100.0);
    Assert.assertEquals(c.quantiteMax,200.0);
  }

  @Test
  public void testRemplir() {
    Cuve c=new Cuve(100.0,200.0);
    c.remplir(50.0);
    Assert.assertEquals(c.quantite,150.0);
    c.remplir(300.0);
    Assert.assertEquals(c.quantite,200.0);
  }

  @Test
  public void testToString() {
  }
```

Markers Properties Servers Data Source Explorer Snippets Console JUnit

Runs: 4/4 Errors: 0 Failures: 3

Finished after 0,11 seconds

tutoJUnit.CuveTest [Runner: JUnit 4] [0,047 s]

- testCuve (0,016 s)
- testRemplir (0,015 s)
- testVider (0,005 s)
- testToString (0,000 s)

Failure Trace

```
junit.framework.AssertionFailedError: expected:<450.0> but was:<200.0>
  at tutoJUnit.CuveTest.testRemplir(CuveTest.java:47)
```

Writable Smart Insert 48 : 6

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with files like Cuve.java, CuveTest.java, and various JUnit-related files.
- CuveTest.java Content:** Contains JUnit test cases for Cuve class methods like testCuve, testRemplir, testVider, and testToString.
- Execution Results:** Shows 4 tests run, 0 errors, and 3 failures. The failure for testRemplir is detailed in the Failure Trace.
- Failure Trace:** Displays the stack trace for the junit.framework.AssertionFailedError.

Correction du problème

```
public void remplir(double quantite) {  
    this.quantite += quantite;  
    if(this.quantite>quantiteMax)  
        this.quantite=quantiteMax;  
}
```

Java EE - Projet JUnit/src/main/java/tutoJUnit/Cuve.java - Eclipse

File Edit Source Refactor Navigate Project Run Window Help

Project Explorer Cuve.java CuveTest.java

```
package tutoJUnit;

public class Cuve {

    double quantite;
    double quantiteMax;

    public Cuve(double quantite, double quantiteMax) {
        this.quantite = quantite;
        this.quantiteMax = quantiteMax;
    }

    public void remplir(double quantite) {
        this.quantite += quantite;
        if(this.quantite>quantiteMax)
            this.quantite=quantiteMax;
    }

    public void vider(double quantite) {
        this.quantite -= quantite;
    }

    public String toString() {
        return quantite + " " + quantiteMax;
    }
}
```

Outline Java EE

tutoJUnit Cuve

- quantite : double
- quantiteMax : double
- Cuve(double, double)
- remplir(double) : void
- vider(double) : void
- toString() : String

Markers Properties Servers Data Source Explorer Snippets Console JUnit

Finished after 0,094 seconds

Runs: 4/4 Errors: 0 Failures: 2

tutoJUnit.CuveTest [Runner: JUnit 4] (0,047 s)

- testCuve (0,016 s)
- testRemplir (0,000 s)
- testVider (0,015 s)
- testToString (0,000 s)

Failure Trace

- java.lang.AssertionError: Not yet implemented
- at tutoJUnit.CuveTest.testVider(CuveTest.java:53)

The screenshot shows the Eclipse IDE interface with the following components visible:

- Top Bar:** Java EE - Projet JUnit/src/main/java/tutoJUnit/Cuve.java - Eclipse, File, Edit, Source, Refactor, Navigate, Project, Run, Window, Help.
- Project Explorer:** Shows the project structure with src/main/java/tutoJUnit containing Cuve.java and CuveTest.java, and src/test/java/tutoJUnit containing CuveTest.java.
- Editor:** Displays the Java code for Cuve.java, which includes fields quantite and quantiteMax, a constructor, methods remplir and vider, and a toString method.
- Outline View:** Shows the outline of the Cuve class with its methods and fields.
- Bottom Bar:** Markers, Properties, Servers, Data Source Explorer, Snippets, Console, JUnit.
- Console:** Shows the message "Finished after 0,094 seconds".
- JUnit View:** Shows the test results: Runs: 4/4, Errors: 0, Failures: 2. It lists four test methods: testCuve, testRemplir, testVider, and testToString. The testVider method is marked as failed.
- Failure Trace:** Displays the error message "java.lang.AssertionError: Not yet implemented" and the stack trace pointing to line 53 of CuveTest.java.

Test encore meilleur !

```
@Test  
public void testRemplir() {  
    Cuve c=new Cuve(100.0,200.0);  
    c.remplir(50.0);  
    Assert.assertEquals(150.0,c.quantite, 0.001);  
    c.remplir(300.0);  
    Assert.assertEquals(200.0,c.quantite, 0.001);  
    c.remplir(-10.0);  
    Assert.assertEquals(200.0,c.quantite, 0.001);  
}
```

→ Echec du test

Correction 2 de la méthode

```
public void remplir(double quantite) {  
    if(quantite>=0){  
        this.quantite += quantite;  
        if(this.quantite>quantiteMax)  
            this.quantite=quantiteMax;  
    }  
}
```

- ▶ Modifier de la même manière la méthode vider et son test associé
- ▶ Modifier de la même manière la méthode toString et son test associé

Cuve.java corrigée

```
package tutoJUnit;
public class Cuve {
    double quantite;
    double quantiteMax;

    public Cuve(double quantite, double quantiteMax) {
        this.quantite = quantite;
        this.quantiteMax = quantiteMax;
    }

    public void remplir(double quantite) {
        if(quantite>=0){
            this.quantite += quantite;
            if(this.quantite>quantiteMax)
                this.quantite=quantiteMax;
        }
    }

    public void vider(double quantite) {
        if(quantite>=0){
            this.quantite -= quantite;
            if(this.quantite<0)
                this.quantite=0;
        }
    }

    public String toString() {
        return quantite + " " + quantiteMax;
    }
}
```

CuveTest

```
@Test
public void testRemplir() {
    Cuve c=new Cuve(100.0,200.0);
    c.remplir(50.0);
    Assert.assertEquals(150.0,c.quantite,0.001)
    c.remplir(300.0);
    Assert.assertEquals(200.0,c.quantite,0.001);
    c.remplir(-10.0);
    Assert.assertEquals(200.0,c.quantite,0.001);
}

@Test
public void testVider() {
    Cuve c=new Cuve(100.0,200.0);
    c.vider(50.0);
    Assert.assertEquals(50.0,c.quantite,0.001);
    c.vider(300.0);
    Assert.assertEquals(0.0,c.quantite,0.001);
    c.vider(-10.0);
    Assert.assertEquals(0.0,c.quantite,0.001);
}

@Test
public void testToString() {
    Cuve c=new Cuve(100.0,200.0);
    Assert.assertEquals("100.0 200.0",c.toString());
}
```

Java EE - ProjetJUnit/src/test/java/tutoJUnit/CuveTest.java - Eclipse

File Edit Source Refactor Navigate Project Run Window Help

Project Explorer Cuve.java CuveTest.java Outline Task List

```
ProjectJUnit
  src/main/java
    tutoJUnit
      Cuve.java
  src/test/java
    tutoJUnit
      CuveTest.java
  JRE System Library [JavaSE-1.6]
  JUnit 4
  src
```

```
@Test
public void testRemplir() {
    Cuve c=new Cuve(100.0,200.0);
    c.remplir(50.0);
    Assert.assertEquals(c.quantite,150.0);
    c.remplir(300.0);
    Assert.assertEquals(c.quantite,200.0);
    c.remplir(-10.0);
    Assert.assertEquals(c.quantite,200.0);
}

@Test
public void testVider() {

    Cuve c=new Cuve(100.0,200.0);
    c.vider(50.0);
    Assert.assertEquals(c.quantite,50.0);
    c.vider(300.0);
    Assert.assertEquals(c.quantite,0.0);
    c.remplir(-10.0);
    Assert.assertEquals(c.quantite,0.0);
}

@Test
public void testToString() {
    Cuve c=new Cuve(100.0,200.0);
    Assert.assertEquals(c.toString(),"100.0 200.0");
}
```

Markers Properties Servers Data Source Explorer Snippets Console JUnit

Finished after 0,063 seconds

Runs: 4/4 Errors: 0 Failures: 0

tutoJUnit.CuveTest [Runner: JUnit 4] (0,016 s)

- testCube (0,016 s)
- testRemplir (0,000 s)
- testVider (0,000 s)
- testToString (0,000 s)

Failure Trace

Import statique

```
Assert.assertEquals(150.0,c.quantite ,0.001);
```

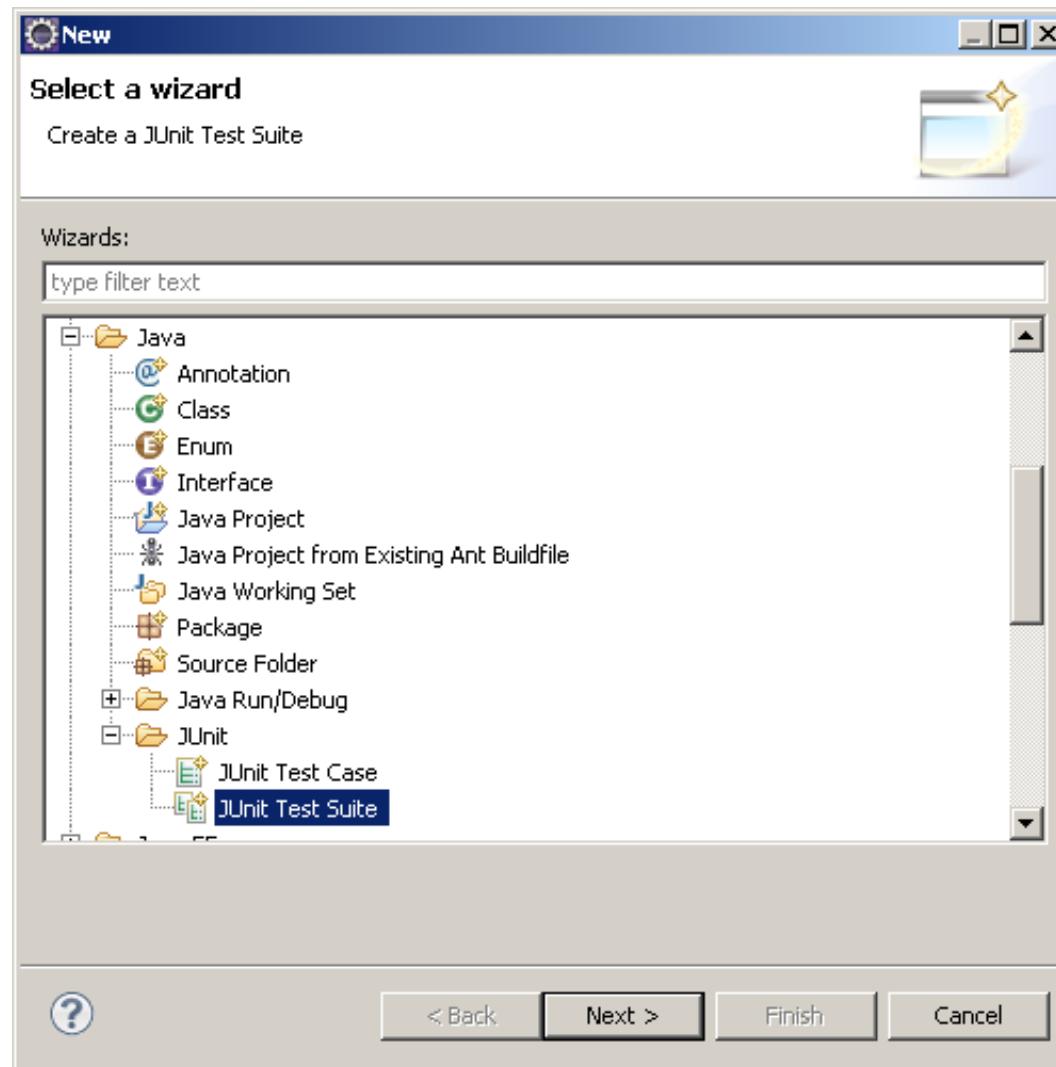
Peut être remplacé par :

```
assertEquals(150.0,c.quantite,0.001);
```

Grâce à l'import statique :

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

JUnit Test Suite



JUnit Test Suite (pour JUnit 4)

```
package tutoJUnit;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.SuiteClasses;
@RunWith(Suite.class)
@SuiteClasses({
    CuveTest.class,
    LiquideTest.class
})
public class AllTests {
```

Liste des classes de Test de la suite

Les annotations de JUnit

- ▶ Les annotations sont arrivées depuis le JDK5 (tiger)

| Annotation | Description |
|--|--|
| @Test public void methode() | @Test permet de dire que c'est une méthode de test |
| @Before public void methode() | La méthode sera exécutée avant chaque test (par ex pour mettre en place l'environnement) |
| @After public void method() | La méthode sera exécutée après chaque test |
| @BeforeClass public void methode() | La méthode sera exécutée 1 fois avant que les tests commencent |
| @AfterClass public void methode() | La méthode sera exécutée 1 fois après l'exécution de tous les tests |
| @Ignore | Permet d'ignorer temporairement ce test |
| @Test(expected=IllegalArgumentException.class) | Permet de vérifier que la méthode génère une exception particulière |
| @Test(timeout=300) | La méthode échouera si elle met plus de 300ms à s'exécuter |

Test-Driven Développement ou TDD

- ▶ développement piloté par les tests
- ▶ **Principe : avant même d'écrire le code d'une fonctionnalité, écrire le test pour ce futur code.** Une fois le test écrit, le code de la fonctionnalité devra toujours passer correctement le test avant de pouvoir être validé.

***xUnit* : pour les autres langages ?**

- ▶ PHP : PHPUnit
- ▶ .NET : nUnit
- ▶ C++ : cppUnit
- ▶ ActionScript 2 : AS2Unit
- ▶ Smalltalk : Sunit
- ▶ Python : PyUnit
- ▶ ...

Maven 2

Le Framework Maven 2

- ▶ cf : support de cours Maven

Hibernate

Le Framework Hibernate

- ▶ cf : support de cours hibernate

Struts 1

Le Framework Struts

Plan du cours Struts

- ▶ Partie 1 : Rappels sur les JSP
- ▶ Partie 2 : Présentation de Struts
- ▶ Partie 3 : Configuration de Struts
- ▶ Partie 4 : Les Taglibs Struts

Partie 1

Rappels sur les Java Server Pages (JSP)

Rappels Programmation JSP

Principe

- ▶ Permet de combiner :
 - contenu statique
 - contenu dynamique
- ▶ Une page .jsp comprend à la fois du code HTML et des scripts "côté-serveur" écrits en Java
- ▶ Les fichiers JSP sont **transformés en Servlets** (lors du premier appel de la page)

Les objets implicites en JSP

- ▶ **request** : correspond à l'objet HttpServletRequest
- ▶ **response** : correspond à l'objet HttpServletResponse
- ▶ **out** : utilisé pour écrire dans la page réponse, c'est le flot de sortie (PrintWriter)
- ▶ **session** : c'est l'objet session

Exemples simples de pages JSP

Exemple 1

```
<%@ page language="java" %>

<html>
<head><title>exemple1 page JSP</title></head>

<body>
<% String hello="Hello World !"; %>
<%= hello %>
</body>
</html>
```

Exemples simples de pages JSP

Exemple 2

```
<%-- informations globales à la page --%>
<%@ page language="java" %>

<html><head><title> exemple 2 page JSP</title></head>
<body>
<%-- Déclaration de la variable c (futur attribut de la servlet) --%>
<%! char c = 0; %>
<%-- Scriptlet (code java qui sera dans le doGet/doPost) --%>
<% for (int i = 0; i < 100; i++){ %>
    <%= i %>
    <br/>
<% } %>

</body></html>
```

Les Java Beans

Un bean Java =

- ▶ Classe publique
- ▶ Un constructeur public sans argument
- ▶ Tous les `setXXX()` et `getXXX()`

Syntaxe des balises JSP

Les balises JSP sont classées en 5 groupes :

1. Les directives
2. Les déclarations
3. Les scriptlets
4. Les expressions
5. Les actions

1-Les directives

Ce sont les informations **relatives à la page**

Syntaxe:

```
<%@ directive attribut1="valeur" attribut2="valeur"... %>
```

Les directives courantes sont les suivantes :

- **page**
- **include**
- **taglib**

Attributs de la directive page

Importer des classes :

```
<%@ page import="java.util.* ,java.io.*" %>
```

Spécifier le MIME-Type et l'encodage :

```
<%@ page contentType="text/html; charset=ISO-8859-15" %>
```

Pour utiliser ou non les sessions (true par défaut) :

```
<%@ page session="true|false" %>
```

Attributs de la directive page

Spécifier une page d'erreur :

```
<%@ page errorPage="url" %>
```

Préciser que la page est une page d'erreur

```
<%@ page isErrorPage="true" %>
```

autres attributs possibles moins utilisés:

implements

extends

...

Attributs de la directive include

Syntaxe:

```
<%@ include file="relative url" %>
```

Exemple :

```
<html>
<body>

<%@ include file="/menu.jsp" %>

</body>
</html>
```

Le fichier '/menu.jsp' est inclus avant la compilation

2-Les déclarations

Permettent de définir des méthodes ou des attributs (au niveau de la servlet)

Syntaxe:

<%! Code Java %>

Exemples:

- ▶ déclaration d'un attribut :

<%! private int compteur = 0; %>

Nombre d'accès à cette page :

<%= ++compteur %>

- ▶ déclaration d'une méthode :

```
<%!
public String hello() {
    return "hello";
} %>
```

3-Les scriptlets

Permettent d'insérer n'importe quel code Java

- ▶ **Syntaxe :**

<%

code Java

%>

3-Les scriptlets

- ▶ Exemple :

```
<% if (Math.random() < 0.5) { %>
    Vous avez gagné!
<% } else { %>
    Vous avez perdu!
<% } %>
```

- ▶ La servlet obtenue après compilation sera alors :

```
if (Math.random() < 0.5) {
    out.println("Vous avez gagné!");
} else {
    out.println(" Vous avez perdu!");
}
```

4-Les expressions

- ▶ Les expressions servent à écrire dans le flot de sortie
(page réponse)
- ▶ Syntaxe :
`<%= expression %>`
- ▶ Exemple:
`<% int a=5; %>`
La valeur de a est : `<%= a %>`
- ▶ `<%=a%>` sera traduit par `out.println(a);` dans la servlet

5-Les actions

- ▶ **jsp:include**

Inclut un fichier lors de l'exécution de la requête (et non avant compilation)

- ▶ **jsp:useBean**

Retrouve ou instancie un Bean

- ▶ **jsp:setProperty**

appel d'un setteur d'un Bean

- ▶ **jsp:getProperty**

appel d'un getter d'un Bean

- ▶ **jsp:forward**

Redirige la requête vers une autre page

Action jsp:include

Syntaxe:

```
<jsp:include page="relative URL" flush="true" />
```

Cette action insère le fichier lors de l'exécution de la requête

- ▶ Exemple :

```
<jsp:include page="page1.html" flush="true"/>  
<jsp:include page="page2.html" flush="true"/>
```

Action jsp:forward

Syntaxe:

```
<jsp:forward page="relative URL" />
```

Permet de faire une redirection

► Exemple:

```
<% String pageSuivante = "/catalogue/accueil.jsp"; %>
```

```
<jsp:forward page="<%= pageSuivante %>" />
```

jsp:useBean

Syntaxe:

```
<jsp:useBean id="visiteur" class="magasin.Visiteur" scope="session" />
```

Si une instance associée à la référence id n'est pas trouvée dans la portée, une nouvelle instance est créée

- ▶ 4 portées possibles :

application: bean visible dans toutes les pages, pour toutes les sessions

session: bean visible dans toutes les pages d'une même session

request: bean visible dans toutes les pages participant à une requête

page: bean visible dans la page

jsp:setProperty

Permet de définir les valeurs des attributs d'un bean

Syntaxe :

```
<jsp:setProperty name="visiteur" property="nom" param="Jean" />
```

- ▶ Fixe l'attribut de visiteur à la valeur "Jean"
- ▶ Cela reviendrait à écrire :

```
visiteur.setNom("Jean");
```

jsp:getProperty

Cette action permet de récupérer la valeur d'un attribut du bean

Synthaxe :

```
<jsp:getProperty name="visiteur" property="prenom" />
```

- ▶ revient à écrire :

```
out.println(visiteur.getProperty("prenom"));
```

Exemple 1 d'utilisation de Beans

Le code source Java du bean :

```
public class Visiteur {  
  
    private String nom;  
  
    public Visiteur(){}  
  
    public String getNom() {  
        return nom;  
    }  
  
    public void setNom(String nom) {  
        this.nom = nom;  
    }  
}
```

Exemple d'utilisation de Beans

- ▶ La page JSP :

```
<html> ...
<jsp:useBean id="test" class="Visiteur" />

<%-- fixe le nom du visiteur à toto --%>
<jsp:setProperty name="test" property="nom" value="toto" />
```

Le nom du visiteur est :

```
<%-- récupère et affiche le nom du visiteur --%>
<jsp:getProperty name="test" property="nom" />

</html>
```

Partie 2

Présentation de Struts

C'est quoi ?

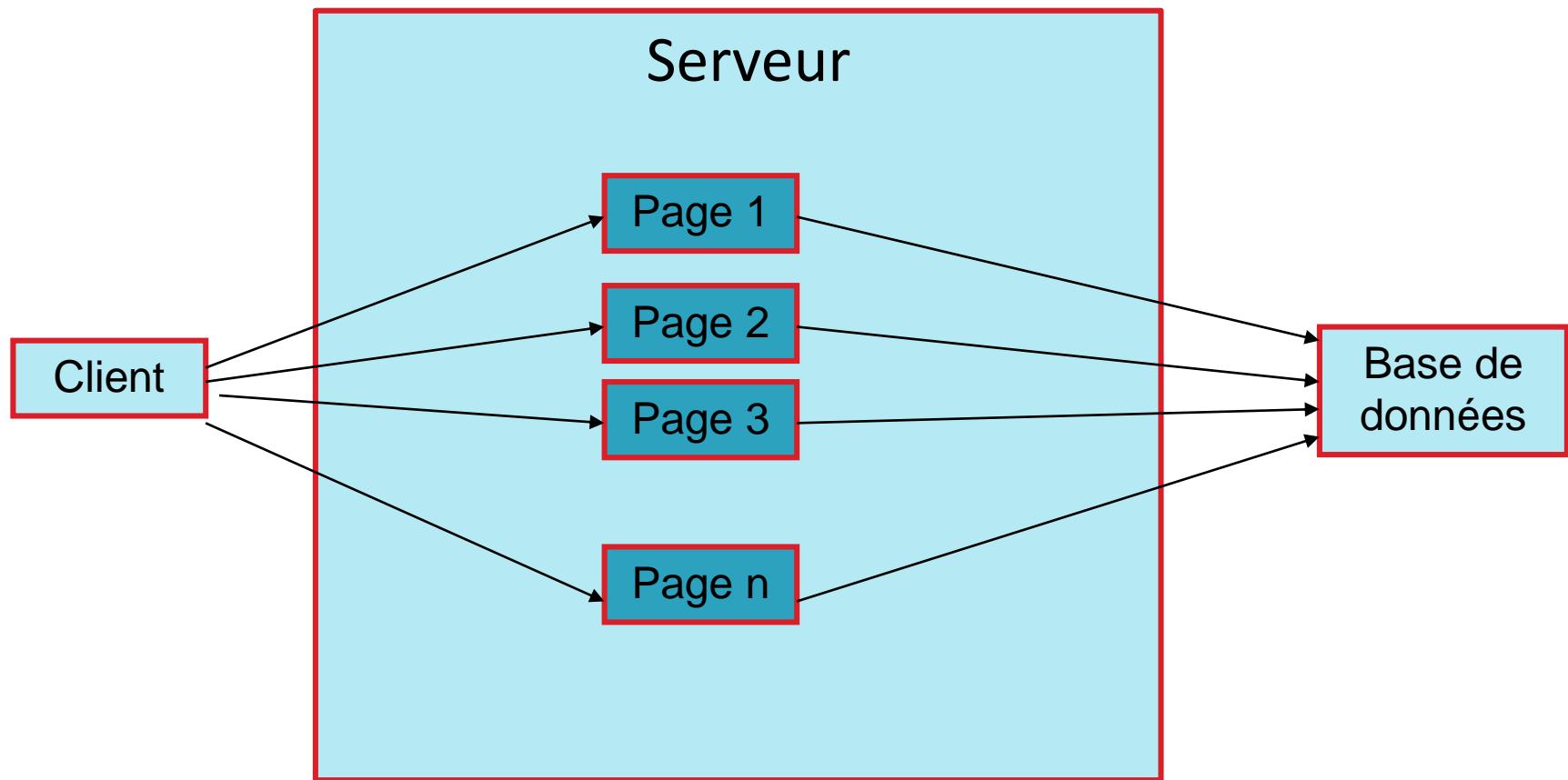
- ▶ Apache Struts est un framework permettant la création d'applications WEB en JAVA
- ▶ Gratuit
- ▶ Open source
- ▶ S'appuie sur le modèle MVC2

Pourquoi ?

- ▶ Souvent les pages WEB (jsp, php, aspx, ...) contiennent du code permettant :
 - la connexion à une base de données
 - l'envoi de requêtes SQL vers une base
 - de faire des redirections de pages, des contrôles de flux, du code métier, ...
 - ...

Cela rend les grosses applications très difficiles à maintenir !

Exemple d'application mal conçue



Avantages/Inconvénients

▶ +

Rapide à réaliser si peu de pages

▶ - - -

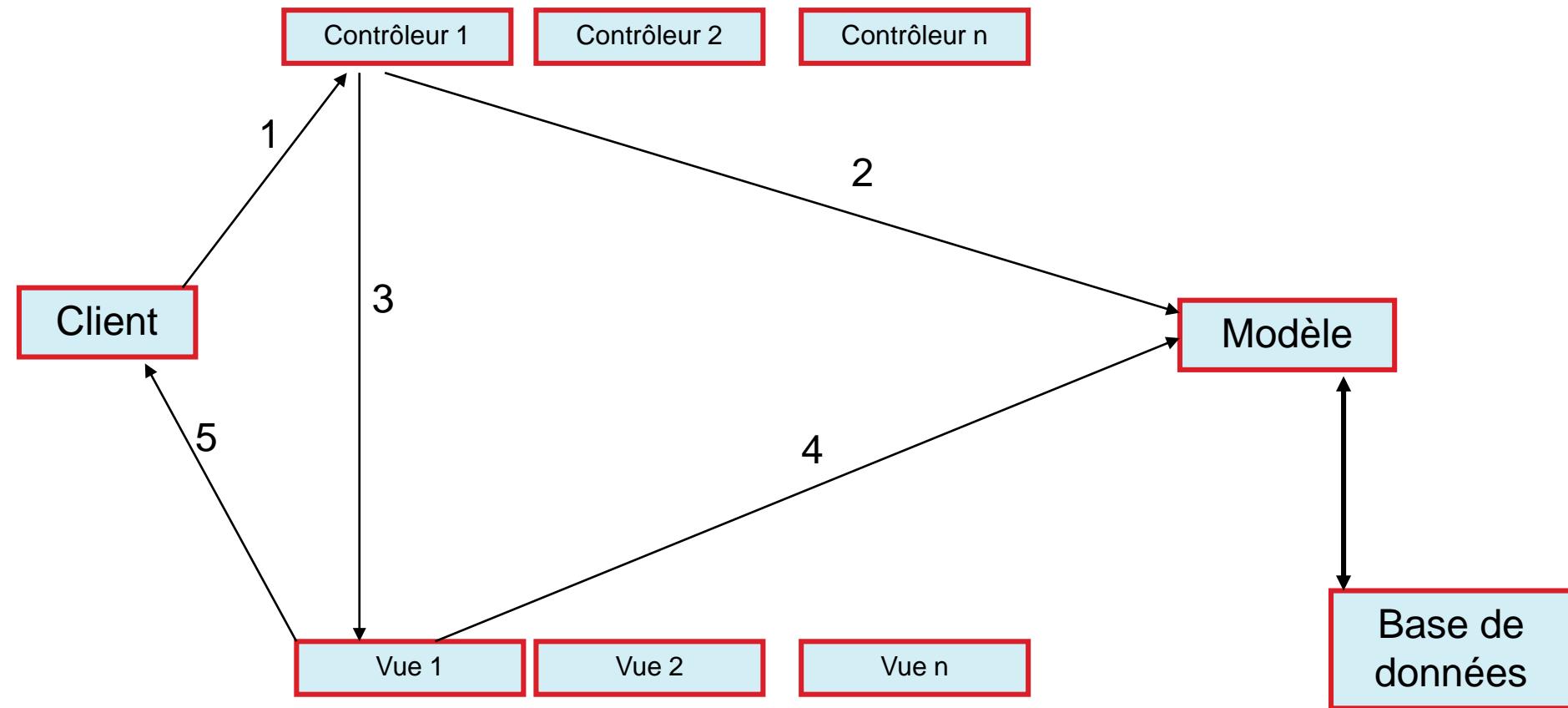
non maintenable dans la durée si grosse application et/ou plusieurs développeurs

→ Architecture à oublier !!!

Solution ?

- ▶ Il faut séparer tous ces concepts
- ▶ Une solution : Architecture Model View Controller (MVC ou MVC2)
 - Model = métier + base de données
 - View = pages web
 - Controller = navigation + contrôle

Architecture MVC (1)



Avantages/Inconvénients

▶ +++

Séparation claire des parties Modèle, Vue et Contrôleur

Possibilité de remplacer une partie sans impact sur les autres

Maintenance aisée

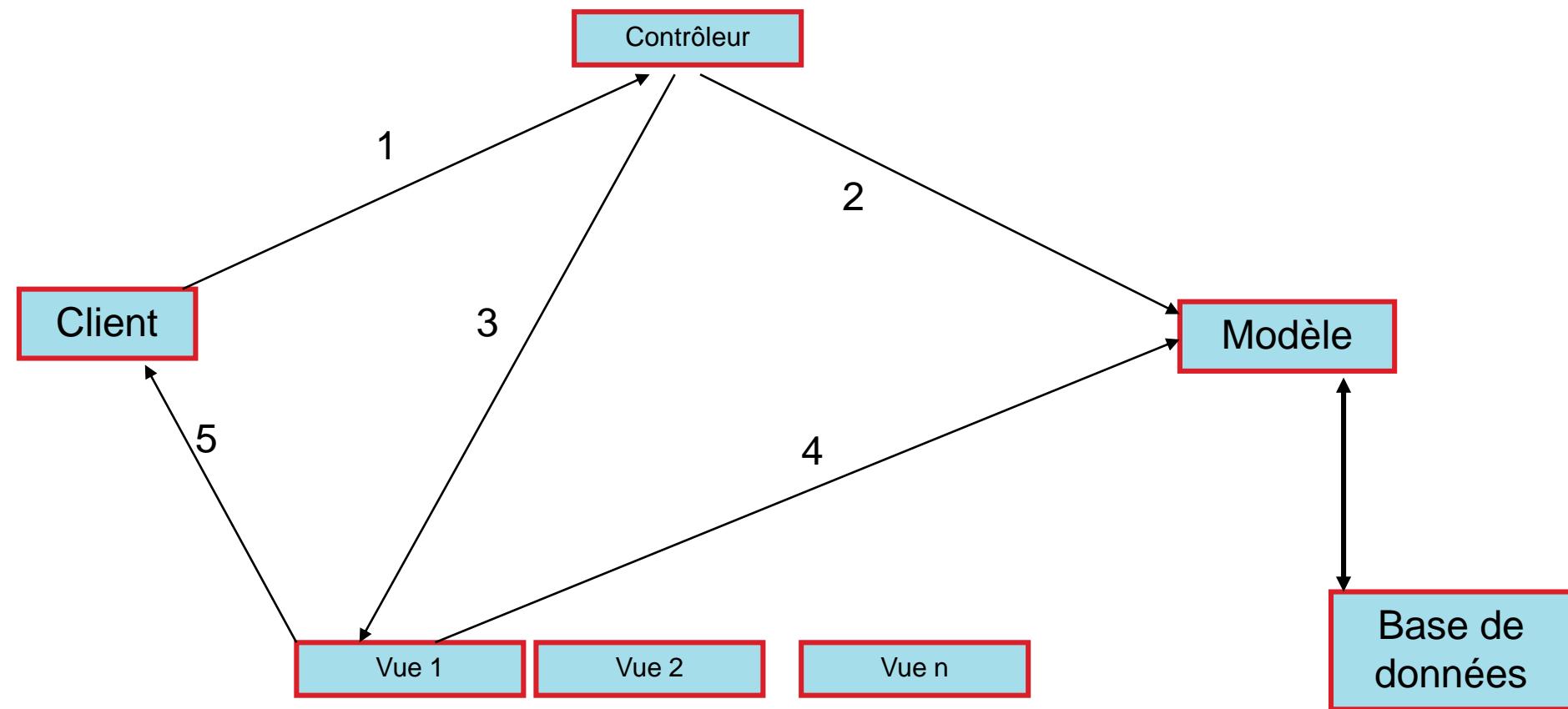
▶ -

Mise en place du projet plus difficile

Il est nécessaire de coder plusieurs Contrôleurs...

(1 par action)

Architecture MVC 2



Avantages/Inconvénients

▶ +++

Séparation claire des parties Modèle, Vue et Contrôleur

Possibilité de remplacer une partie sans impact sur les autres

Maintenance aisée

▶ -

Mise en place du projet difficile

Et Struts ?

- ▶ Struts s'appuie sur l'architecture MVC2

Origine de Struts

- ▶ Développé à l'origine par Craig McClanahan puis offert à Apache en mai 2000

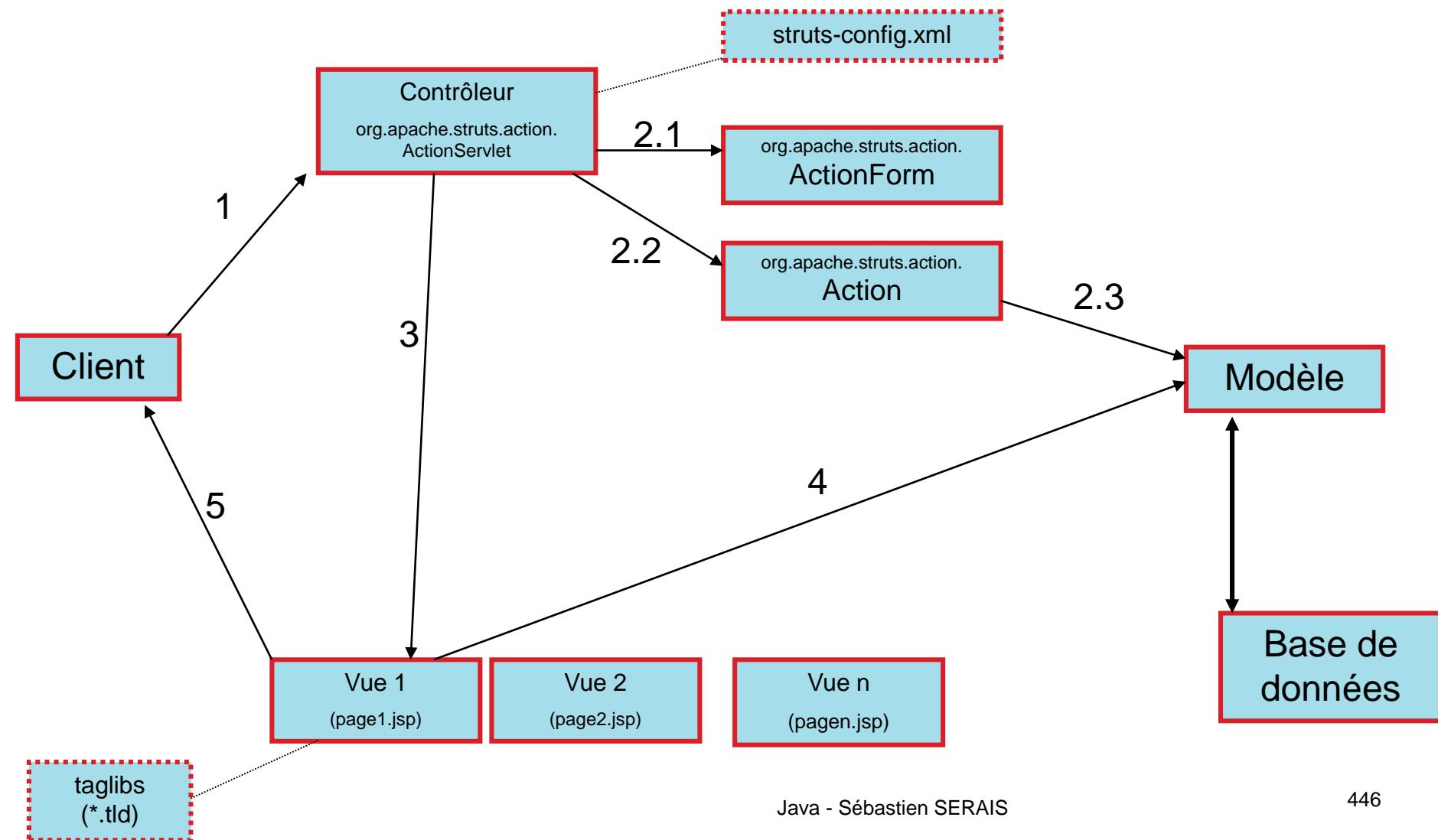


2 versions



- ▶ Struts 1 est la version la plus connue et la plus utilisée (2000)
 - bon choix pour les problèmes les plus courants
- ▶ Struts 2 (2007) (ancien WebWork 2)
 - se développe sur les nouveaux projets

Architecture de Struts (1) (MVC2)



Partie 3

Configuration de Struts

Mise en place du projet

- ▶ Créer un projet Java
- ▶ Ajouter les dépendances Struts dans Maven :
 - struts-core (org.apache.struts)
 - struts-taglib (org.apache.struts)
 - struts-extras (org.apache.struts)

Configuration du web.xml

- ▶ Déclaration de la servlet de struts (contrôleur) :

```
<web-app>
<servlet>
<servlet-name>action</servlet-name>
<servlet-class>org.apache.struts.action.ActionServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
...
</web-app>
```

Configuration du web.xml

- ▶ load-on-startup permet de dire au container de charger la servlet au démarrage
- ▶ L'entier correspond à la priorité
(1=priorité la plus élevée)

Configuration du web.xml

- ▶ Les url sont de type :

`http://host:port/contexte/NomAction.do`

- ▶ Configuration du mapping :

```
<web-app>
```

```
...
```

```
<servlet-mapping>
```

```
<servlet-name>action</servlet-name>
```

```
<url-pattern>*.do</url-pattern>
```

```
</servlet-mapping>
```

```
</web-app>
```

Configuration : struts-config.xml

- ▶ par défaut dans WEB-INF
- ▶ exemple :

```
<?xml version="1.0" encoding="ISO-8859-15" ?>
<!DOCTYPE struts-config PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>
    <form-beans> ←
        <form-bean name="personneForm" type="forms.PersonneForm" />
    </form-beans> ←
    <action-mappings> ←
        <action scope="request" path="/AjouterPersonne" name="personneForm" type="controleur.AjouterPersonneAction"
            input="/">
            <forward name="succes" path="/WEB-INF/jsp/listePersonnes.jsp"></forward>
            <forward name="erreur" path="/WEB-INF/jsp/erreurAjout.jsp"></forward>
        </action>
    </action-mappings> ←
    <message-resources parameter="mesMessages" /> ←
</struts-config>
```

declaration des *ActionForm*

configuration des *Action*

declaration des fichiers de ressources

struts-config.xml

▶ autres éléments possibles :

- global-exceptions : déclarations d'exceptions globales
- global-forwards : déclarations de redirections globales
- plug-in : permet de découvrir dynamiquement des ressources au démarrage

Les ActionForm

- ▶ Ils permettent le stockage des données des formulaires envoyés
- ▶ Ils doivent être déclarés dans struts-config.xml
- ▶ La portée d'un ActionForm peut être request ou session

Cycle de vie d'un ActionForm

1. Le client valide un formulaire
2. Le contrôleur reçoit la requête
3. Contrôleur crée objet ActionForm correspondant
4. La méthode reset() est appelée
5. L'objet ActionForm est enregistré dans la portée choisie (request ou session)
6. L'objet ActionForm est rempli avec les données du formulaire
7. La méthode ActionForm.validate() est appelée (si demandé) pour valider les données du formulaire
 1. Si invalide : forward vers la page définie par attribut "input"
 2. Si valide : appel de la méthode execute() de l'action avec en entrée cet ActionForm

Exemple de configuration

```
<form-beans>
    <form-bean name="personneForm" type="forms.PersonneForm"/>
</form-beans>

<action-mappings>
    <!--
        input : page où il revient si erreur de validation des données du formulaire
        name : nom du form-bean
        type : nom de la classe Action
        scope : portée du form-bean (request ou session)
        path : chemin d'accès à la requête soumise
        validate : permet de préciser si la méthode validate doit être exécutée (true par défaut) -->
    <!-- scope=session permettra de retrouver le formulaire rempli si réaffichage de celui ci -->
    <action scope="request" path="/AjouterPersonne" name="personneForm"
        type="controleur.AjouterPersonneAction" input="/">

        <forward name="succes" path="/WEB-INF/jsp/listePersonnes.jsp"></forward>
        <forward name="erreur" path="/WEB-INF/jsp/erreurAjout.jsp"></forward>

    </action>
</action-mappings>
```

pour aller plus loin...

- ▶ La classe DynaActionForm permet d'éviter d'avoir à écrire des ActionForm
- ▶ Tout se passe alors dans le fichier struts-config.xml
- ▶ Pratique en cas de modification des formulaires, cela évitera d'avoir à recompiler

L' Action

Définir une classe qui hérite de Action

La classe doit redéfinir la méthode :

```
public ActionForward execute(ActionMapping mapping, ActionForm form,  
    HttpServletRequest request, HttpServletResponse response){...}
```

- ▶ Cette méthode correspond au code de l'action demandée
- ▶ Elle retourne le Forward approprié (erreur, succès, ...)

Exemple de classe Action

```
public class AjouterPersonneAction extends Action {  
  
    public ActionForward execute(ActionMapping mapping, ActionForm form,  
        HttpServletRequest request, HttpServletResponse response)  
    throws Exception {  
  
        //on instancie une Personne à partir des données du formulaire  
        PersonneForm f=(PersonneForm)form;  
        Personne p=new Personne(f.getNom(),f.getPrenom());  
  
        //récupération de l'instance de la classe DAO correspondante  
        IPersonneDAO personneDAO=PersonneDAOFactory.getPersonneDAO ();  
        //appel de la méthode permettant d'ajouter une personne dans la liste  
        personneDAO.ajouter(p);  
  
        //on ajoute dans la requête la liste des personnes qui sera affichée sur la vue (page .jsp)  
        request.setAttribute("liste",personneDAO.getListe());  
  
        //on indique que la page suivante est celle qui correspond au mot clé "succes"  
        //on aurait pu ajouter en cas d'erreur un : return mapping.findForward("erreur");  
        return mapping.findForward("succes");  
    }  
}
```

Partie 4

- ▶ Les Taglibs Struts

Les Taglibs

- ▶ Possibilité de créer ses propres balises → taglib (tag library)
- ▶ Une taglib est déclarée dans un fichier .tld (taglib description)
- ▶ exemple : <c:out />

espace de nom
défini dans fichier
.tld

correspond à une
classe

Les Taglibs

- ▶ Struts propose d'utiliser ses taglibs

- ▶ exemple :

```
<%@taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
```

permet d'écrire dans la page jsp:

```
<html:text property="nom" >
```

cela générera le code HTML suivant :

```
<input type="text" name="nom" value="">
```

Les Taglibs Struts

- ▶ Les taglibs de struts permettent d'aider le développement de la couche présentation
- ▶ Composé de 4 librairies principales :
 - Bean
 - HTML
 - Logic
 - Nested

Les taglibs Struts

- ▶ struts-taglib-X.X.X.jar
 - struts-bean.tld
 - struts-html.tld
 - struts-logic.tld
 - struts-nested.tld
- ▶ <http://struts.apache.org/1.x/struts-taglib/index.html>

Les taglibs Struts

Pour utiliser les taglibs, il faut le préciser dans la page JSP :

```
<%@taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@taglib uri="http://struts.apache.org/tags-nested prefix="nested" %>
```

Les taglibs Struts

▶ Les Tags : Bean

- donnent accès aux JavaBean et à leurs propriétés

exemple :

```
<bean:write name="pers" property="texte"/>
```

- permettent de définir de nouveaux beans

Les taglibs Struts

▶ Les tags : HTML

- permettent la création de formulaires HTML ainsi que d'autres composants HTML

exemples :

champ texte : <html:text property="auteur" />

lien : <html:link action="retourIndex">Retour Index</html:link>

bouton submit : <html:submit value="Ajouter l'histoire" />

Les taglibs Struts

▶ Les tags : Logic

- permettent de définir des conditions
- permettent de définir des boucles sur des collections

exemple :

```
<logic:iterate id="e" name="monHastable">
    <bean:write name="e" property="value"/>
</logic:iterate>
```

- permettent de faire des redirections ou des réexpéditions (forward ou redirect)

Les taglibs Struts

- ▶ Les tags : Nested
 - permettent de créer des balises emboîtées

Documentation des tags de struts

Pour aller plus loin :

<http://struts.apache.org/1.x/struts-taglib/index.html>



Apache : Log4j

Framework de Logging pour Java

Les Logs : pourquoi ?

- ▶ permettent parfois de déboguer quand pas de débogueur
- ▶ permettent de conserver des traces de l'application :
 - quelles actions ont été réalisées ?
 - par qui ?
 - à quel moment ?
 - quels ont été les éventuelles erreurs ?
 - ...

Les Logs : comment ?

System.out.println() ??

▶ **inconvénients :**

- surcharge le programme
- ralentissent l'exécution du programme
- difficile de désactiver les logs simplement (à moins de tout mettre en commentaire...)
- tout est écrit dans le même fichier

Une solution possible : log4j

► Log4j :

Version 1.0 : 1996

Version 1.2 : 2004

version 1.3 : abandonnée

version 2.0 : juillet 2014...

Créateur initial : Ceki Gulku

Licence : Apache License 2.0

Log4J est l'un des frameworks de log en java parmi (Java Logging API, Apache Commons Logging, SLF4J, tinylog, ...)

Une solution possible : log4j

avantages :

- ▶ possibilité d'activer/désactiver les logs en modifiant simplement un fichier de configuration (.properties ou .xml)
- ▶ possibilité de classer les logs par catégorie
- ▶ possibilité d'envoyer les logs vers des destinations différentes (console, fichiers, réseaux, Bdd, ...)

Une solution possible : log4j

avantages (suite) :

- ▶ permet de définir pour chaque message un certain niveau de criticité
 - TRACE
 - DEBUG,
 - INFO,
 - WARNING,
 - ERROR,
 - CRITICAL
- ▶ possibilité de connaître pour un log la méthode/classe qui l'a généré
- ▶ possibilité de connaître le n° de ligne qui a généré le log

Installation

- ▶ Log4j :

log4j-1.2.17.jar



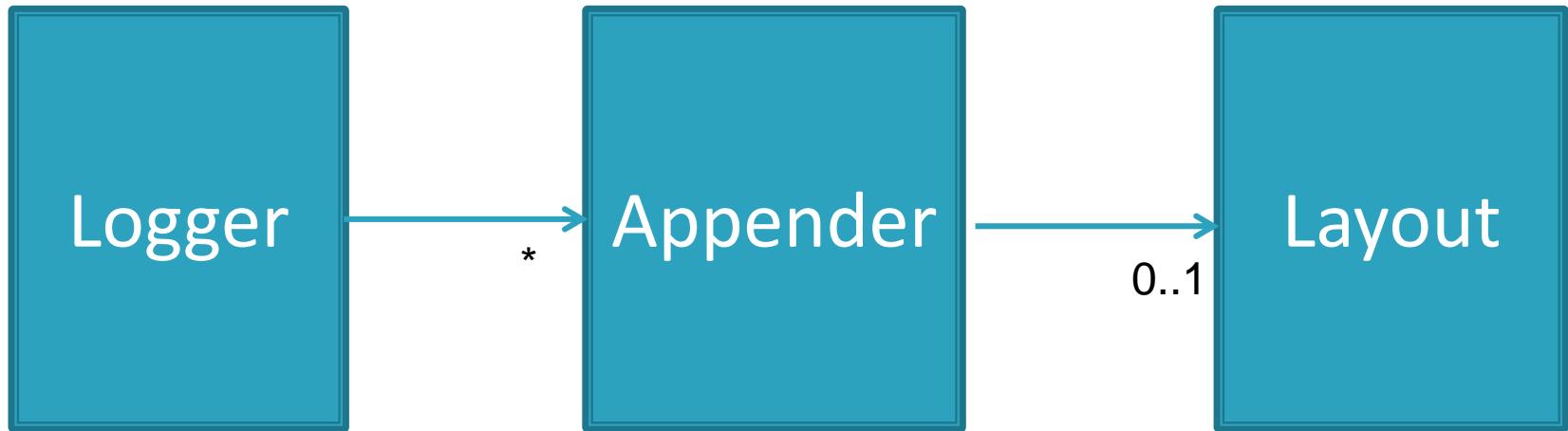
- ▶ à importer à la main
ou
- ▶ ajouter dans le pom.xml de Maven

Log4j : 3 composantes

- ▶ **QUOI ?** : Les **Loggers** : pour écrire les messages
- ▶ **Où ?** : Les **Appenders** : pour choisir la destination des messages (fichiers, BDD,...) et permet la rotation éventuelle des noms de fichiers
- ▶ **COMMENT ?** : Les **Layouts** : pour la mise en forme des messages

Log4j : 3 composantes

- ▶ Vue UML globale :



Logger

- ▶ Le Logger est la classe qui permet de générer les logs
- ▶ Comment récupérer un Logger :

```
package fr.esigelec;  
import org.apache.log4j.Logger;  
  
public class MaClasse {  
  
    private static final Logger LOGGER1 =Logger.getLogger(MaClasse.class);  
    private static final Logger LOGGER2 =Logger.getLogger("fr.esigelec.MaClasse");  
  
    ...  
}
```

--> les 2 loggers sont les mêmes

nom du logger (String),
ou classe elle-même (Class)
-> équivalent

Le rootLogger

- ▶ Tous les loggers héritent du logger racine : **rootLogger**
- ▶ Le rootLogger :
 - existe toujours
 - ne possède pas de nom
- ▶ Tous les loggers vont donc hériter de sa configuration

Niveau de journalisation

- ▶ La classe **Level** permet de définir le niveau de journalisation des messages :
 - **FATAL** : erreur très grave qui peut mettre fin à l'exécution
 - **ERROR** : erreur (qui n'arrête pas l'application)
 - **WARN** : warning (avertissement)
 - **INFO** : information
 - **DEBUG** : messages utiles pour débogger
 - **TRACE** : correspond à des messages de traces d'exécution (depuis la version 1.2.12)

Niveau de journalisation

- ▶ Deux autres niveaux sont définis et utilisables dans la configuration :
 - **OFF** : aucun niveau de gravité n'est pris en compte
 - **ALL** : tous les niveaux de gravité sont pris en compte

Niveau de journalisation

- ▶ Le message sera journalisé seulement si sa priorité est supérieure ou égale à la priorité de son Logger
- ▶ Pour modifier la priorité d'un logger : **setLevel(..)**
- ▶ Exemple :
 - `logger.setLevel(Level.INFO);`
 - Dans cet exemple, tous les logs de niveaux inférieurs à INFO seront ignorés (DEBUG, TRACE)

Générer des logs

- ▶ logger.**log**(Priority priority, Object message)

OU

- ▶ logger.**log**(Priority priority, Throwable exception)

Générer des logs

Exemples :

- ▶ logger.log(Level.FATAL,"message");
- ▶ logger.log(Level.ERROR,"message");
- ▶ logger.log(Level.WARN, "message");
- ▶ logger.log(Level.INFO,"message");
- ▶ logger.log(Level.DEBUG,"message");
- ▶ logger.log(Level TRACE,"message");



Générer des logs

méthodes équivalentes :

- ▶ `logger.log(Level.FATAL,"message");` ou `logger.fatal("message");`
- ▶ `logger.log(Level.ERROR,"message");` ou `logger.error("message");`
- ▶ `logger.log(Level.WARN, "message");` ou `logger.warn("message");`
- ▶ `logger.log(Level.INFO,"message");` ou `logger.info("message");`
- ▶ `logger.log(Level.DEBUG,"message");` ou `logger.debug("message");`
- ▶ `logger.log(Level TRACE,"message");` ou `logger.trace("message");`

Questions ? 1/2

Que vont afficher les 2 codes suivants :

1:

```
Logger logger =Logger.getLogger("fr.esigelec.MonExemple");
logger.setLevel(Level.INFO);
logger.error("mon message d'erreur");
```

2:

```
Logger logger =Logger.getLogger("fr.esigelec.MonExemple");
logger.setLevel(Level.INFO);
logger.debug("mon message de debug");
```

Questions ? 2/2

Que vont afficher les 2 codes suivants :

3:

```
Logger logger =Logger.getLogger("fr.esigelec.MonExemple");
logger.setLevel(Level.OFF);
logger.error("mon message d'erreur");
```

4:

```
Logger logger =Logger.getLogger("fr.esigelec.MonExemple");
logger.setLevel(Level.ALL);
logger.warn("mon message de warning");
```

Les Appenders

- ▶ org.apache.log4j.Appender
- ▶ Cette interface permet de définir la stratégie d'écriture des logs -> le lieu dans lequel seront stockés les logs
- ▶ Plusieurs implémentations sont fournies avec Log4J :

Les Appenders (les plus courants)

- ▶ **ConsoleAppender** : envoie les logs dans la console
- ▶ **FileAppender**: envoie les logs dans un fichier
- ▶ **DailyRollingFileAppender** : envoie les logs vers des fichiers avec noms qui tournent :
 - fichier.log
 - puis : fichier.log + fichier.log.1
 - puis : fichier.log + fichier.log.1 + fichier.log.2
 - ...

Rq : le dernier fichier est toujours fichier.log

Rq2 : fichier.log.n sera effacé si n > valeur de config (MaxIndexBackup)

Les Appenders (les plus courants)

- ▶ **JDBCAppender** : envoie les logs dans une base de données
- ▶ **NullAppender** : envoie les logs vers null (rien)
- ▶ **SMTPAppender** : envoie les logs par mail
- ▶ **LF5Appender** : envoie les logs vers une application graphique Swing (Log Force 5)

exemple utilisation LF5Appender

The screenshot shows the LogFactor5 application window. The menu bar includes File, Edit, Log Level, View, Configure, and Help. A toolbar below the menu has a font selector (Font: Dialog, Size: 12), a clear log table button, and a status message "Displaying: 2 records out of a total of: 2 records.". On the left, a tree view under the 'Categories' node shows a single expanded node named 'exception'. The main area is a table with the following data:

| | Date | Thread | Messa... | Level | NDC | Category | Message | Location | Thrown |
|--------------------------|---------------|--------|----------|-------|-----|-----------|----------------------------|------------------|---------------------|
| <input type="checkbox"/> | Mon Oct 29... | main | 1 | FATAL | | exception | Une exception est survenue | EssaiLog.main... | java.lang.Arithm... |
| <input type="checkbox"/> | Mon Oct 29... | main | 2 | INFO | | exception | Voici une information | EssaiLog.main... | java.lang.Arithm... |

Les Appenders (les autres)

- ▶ **NTEventLogAppender** : envoie les logs vers le journal d'événements de Windows
- ▶ **RewriteAppender** : envoie le message vers un autre appender
- ▶ **RollingFileAppender** : envoie dans un fichier, et recrée un nouveau fichier si celui ci devient trop long
- ▶ **SocketAppender** : envoie les logs vers une socket
- ▶ **SocketHubAppender** : envoie les logs vers plusieurs sockets

Les Appenders (les autres)

- ▶ **SyslogAppender** : envoie les logs vers un deamon syslog unix
- ▶ **JMSAppender** utilise JMS pour envoyer les logs
- ▶ **TelnetAppender** : envoie vers une socket consultable avec telnet
- ▶ **AsyncAppender** : envoie vers différents appenders de manière asynchrone
- ▶ **ExternallyRolledFileAppender** : écoute sur une socket un message et envoie ensuite un accusé de réception

Les Layouts

Permettent de définir le format des logs :

- ▶ **SimpleLayout** :
 - Niveau - Message[Retour à la ligne]
 - exemple :
 - WARN - Voici le message de warning
- ▶ **PatternLayout** : un pattern permet de fixer précisément le contenu du log (date, heure, message, niveau,...)

Patterns possibles

- ▶ **%c** : nom du logger
- ▶ **%C** : nom de la classe qui génère le log
- ▶ **%d** : date et heure du message
- ▶ **%m** : message
- ▶ **%n** : saut de ligne
- ▶ **%p** : niveau de gravité du message
- ▶ **%t** : nom du thread
- ▶ **%L** : n° de la ligne qui a généré le log
- ▶ **%F** : nom du fichier qui a généré le log
- ▶ **%M** : nom de la méthode qui a généré le log

Les Layouts (suite)

- ▶ **XMLLayout** : logs au format XML (à utiliser avec un Appender de la famille des FileAppenders)
- ▶ **HTMLLayout** : logs au format HTML (à utiliser avec un Appender de la famille des FileAppenders)

Méthodes principales

Méthodes utiles :

```
//ajout d'un appender à un logger  
Logger.addAppender(Appender)
```

```
//fixe le niveau de journalisation  
Logger.setLevel(Level)
```

```
//modifie le nom de l'appender  
Appender.setName(String);
```

```
//fixe le layout d'un appender  
Appender.setLayout(Layout);
```

```
//active les modifications apportées à un appender  
Appender.activateOptions();
```

log4j.properties

- ▶ **log4j.xml** ou **log4j.properties** permettent de configurer de façon externe log4j
 - à mettre dans le classpath
- ▶ exemple de contenu de log4j.properties:

log4j.rootLogger=INFO, **monAppender**

log4j.appender.**monAppender**=org.apache.log4j.ConsoleAppender

log4j.appender.**monAppender**.layout=org.apache.log4j.PatternLayout

log4j.appender.**monAppender**.layout.ConversionPattern=%d %m%n

log4j.properties

- ▶ Exemple 2 :
- ▶ Configuration d'un logger autre que le rootLogger :

log4j.logger.**fr.esigelec.MaClasse**=INFO, monAppender

...

...

Nom de la classe ou nom du logger

Héritage de Logger

- ▶ Tous les Logger héritent du rootLogger, puis la hiérarchie est basée sur les noms :

- Le Logger : **fr.esigelec.GSI**
- hérite du logger **fr.esigelec**,
- qui hérite du logger **fr**
- qui hérite de **rootLogger**

rootLogger

fr

fr.esigelec

fr.esigelec.
GSI

- ▶ Pour stopper un héritage de loggers :
 - utiliser la méthode `Logger.setAdditivity(false)`
 - Le logger (et ses enfants) ne récupèrera pas les caractéristiques de ses parents

pour aller plus loin...

- ▶ <http://beuss.developpez.com/tutoriels/java/jakarta/logging/>
- ▶ <http://www.jmdoudoux.fr/java/dej/chap-logging.htm>
- ▶ <http://www.tutorialspoint.com/log4j/index.htm>

Pratique

- ▶ cf. sujet de TP



Bon travail
avec Log4J !