

Janvier 2024



<https://perso.isima.fr/loic>





ISIMA 

Philosophie ?

- Faire des applications [WEB] professionnelle avec JAVA
- Plus simple que Java EE / Jakarta EE
 - Partage de certaines JSR
- Convention plutôt que configuration
 - Annotation (décorations)
 - Fichiers `.properties`
 - XML ?
- Gestion du cycle de vie des objets
 - Injection de dépendances



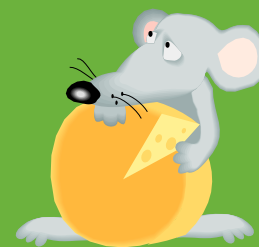
Ecosystème ? Concurrents ?

- J2E / Java EE / Jakarta EE 1999
- Spring  2003
- Spring Boot 3.2  2014

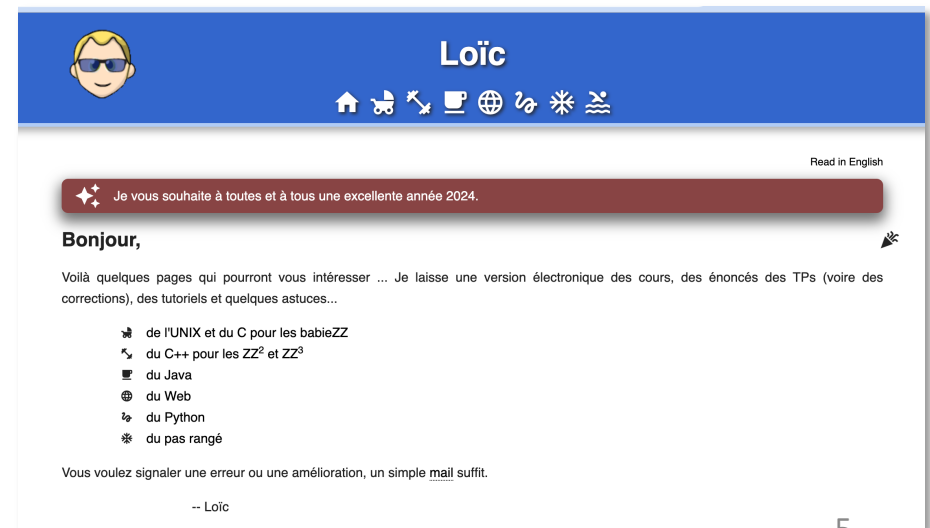
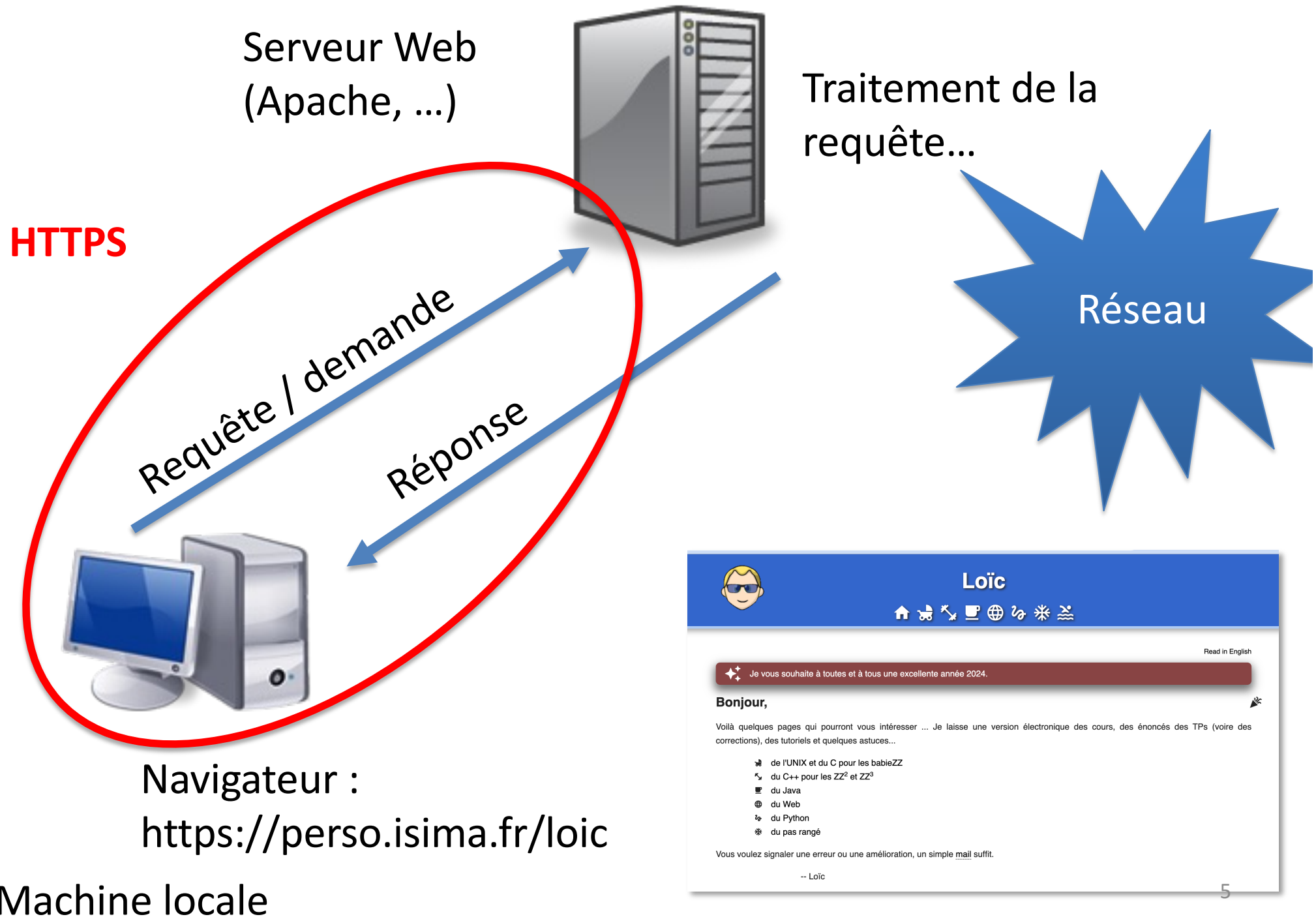
Version 2	Version 3
Java 8->19 Java EE 8	Java 17-> Jakarta EE 9

<https://spring.io>

UNE PINCEE DE WEB



ISIMA 



Que renvoie le serveur ?

Que comprend le navigateur ?

- Du texte
 - HTML
 - Style (CSS)
 - Javascript
- Mais encore
 - XML
 - JSON
- Du binaire
 - Images (format ?)
- Du binaire
 - Multimédia
 - Données ?

Exemple de contenu "statique"

```
<!doctype html>
<html>
  <head>
    <title> Page du loïc </title>
  </head>
  <body>
    <p>Bienvenue sur ma page</p>
  </body>
</html>
```

Envoyé par le serveur Web
Sans traitement

Traduit par le navigateur

Fichier texte : index.html



Interactions basiques

- Demande simple : GET
 - Obtention de ressource
 - Communication non sécurisée par passage par paramètre

`https://monsite.fr/index.html?nom=yon&prenom=loic`

- Envoi de données : POST
 - Formulaire
 - Nécessite un traitement côté serveur





Affichage d'un page statique index.html

- Création d'un premier projet
 - Spring Initializr
- Lancement du serveur
 - Erreur à l'exécution ? configuration
- Affichage de la première page web statique
 - Page de fallback



Outils ?

- Maven / Gradle
- Spring Initializr



The screenshot shows the Spring Initializr web application interface. On the left, there is a navigation menu (hamburger icon) and the Spring Initializr logo. The main content area is divided into several sections:

- Project:** Radio buttons for Gradle - Groovy, Gradle - Kotlin, and Maven.
- Language:** Radio buttons for Java, Kotlin, and Groovy.
- Spring Boot:** Radio buttons for 3.3.0 (SNAPSHOT), 3.2.2 (SNAPSHOT), 3.2.1, 3.1.8 (SNAPSHOT), and 3.1.7.
- Project Metadata:** Form fields for Group (app), Artifact (bonjour), Name (Bonjour), Description (first glimpse of spring boot), and Package name (app).
- Packaging:** Radio buttons for Jar and War.
- Java:** Radio buttons for 21 and 17.
- Dependencies:** A button labeled "ADD DEPENDENCIES... ⌘ + B". Below it, two dependency cards are visible:
 - Spring Boot DevTools** (DEVELOPER TOOLS): Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
 - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

At the bottom of the interface, there are three buttons: "GENERATE ⌘ + ↵", "EXPLORE CTRL + SPACE", and "SHARE...".

On the right side of the interface, there is a dark theme toggle button (moon icon) and a large blue link: <https://start.spring.io/>

Tâches Gradle

- gradle[w] build
- gradle[w] bootRun

```
hello -- java -Xmx64m -Xms64m -javaagent:/opt/homebrew/Cellar/gradle/8.5/libexec/lib/agents/gradle-instrumentation-agent-8.5.jar -Dorg.gradle.appname=gradle -classpath /opt/homebrew/Cellar/gradle/8.5/libexec/lib/gradle-launcher-8.5.jar org.gradle.launcher.GradleMain boot...

> Task :bootRun

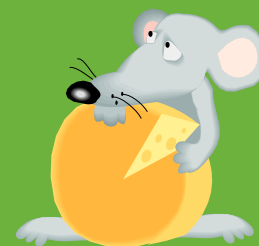
  ____  _
 / ___|| | | |
 \___ \| |_| |
  ___) | | | |
 / ___|| | | |
 \___) |_| |_|
      |_|_|_|

:: Spring Boot :: (v3.2.1)

2024-01-08T09:19:27.027+01:00 INFO 32825 --- [main] hello.HelloApplication : Starting HelloApplication using Java 21.0.1 with PID 32825 (/Users/loic/Downloads/hello/build/classes/java/main started by loic in /Users/loic/Downloads/hello)
2024-01-08T09:19:27.029+01:00 INFO 32825 --- [main] hello.HelloApplication : No active profile set, falling back to 1 default profile: "default"
2024-01-08T09:19:27.362+01:00 INFO 32825 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2024-01-08T09:19:27.369+01:00 INFO 32825 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-01-08T09:19:27.369+01:00 INFO 32825 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.17]
2024-01-08T09:19:27.390+01:00 INFO 32825 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-01-08T09:19:27.390+01:00 INFO 32825 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 344 ms
2024-01-08T09:19:27.525+01:00 INFO 32825 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-01-08T09:19:27.529+01:00 INFO 32825 --- [main] hello.HelloApplication : Started HelloApplication in 0.642 seconds (process running for 0.773)

<=====> 80% EXECUTING [24s]
> :bootRun
```

PRINCIPES DE BASE



ISIMA 

Application Web

FRONT

Requête / demande

Réponse



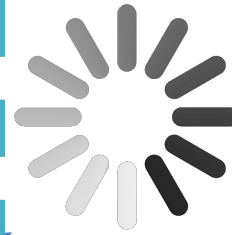
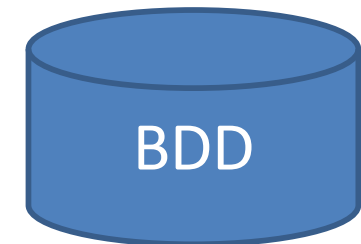
Navigateur :
`https://a/p`

Machine locale

Serveur Applicatif



BACK



Contenu "dynamique"

- Contenu généré à la volée
 - JS, JAVA, PHP , Python, ...
- Mais pas limité aux pages web
- Interaction avec d'autres éléments ?

- Demandes au serveur
- Réponses
- Servlets
- Templates

Requêtes (verbes HTTP)

- URI + Verbe => Code de retour
- Entête (head) + Corps (Body)

C	POST	<ul style="list-style-type: none">• Création de ressource• Formulaire
R	GET	<ul style="list-style-type: none">• Obtention d'une ressource• Page Web
U	PUT [PATCH]	<ul style="list-style-type: none">• Modification de ressource [partielle]
D	DELETE	<ul style="list-style-type: none">• Suppression

Quelques codes de retour

- **200 : OK**
- 201 : Created
- 204 : No content
- 400 : Bad request
- 404 : Not found
- 405 : Method not allowed
- 500 : Internal server

Servlet

- Classe java qui s'exécute grâce à un moteur de servlet / conteneur Web
- Historiquement, un des premiers types de composants JAVA
- Toujours utilisé pour la configuration
- Ou les traitements plus complexes

- Cycle de vie ?
 - init, service, destroy

jakarta.servlet.http

Class HttpServlet

java.lang.Object
 jakarta.servlet.GenericServlet
 jakarta.servlet.http.HttpServlet

All Implemented Interfaces:
Servlet, ServletConfig, Serializable

```
public abstract class HttpServlet  
extends GenericServlet
```

Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of `HttpServlet` must override at least one method, usually one of these:

- `doGet`, if the servlet supports HTTP GET requests
- `doPost`, for HTTP POST requests
- `doPut`, for HTTP PUT requests
- `doDelete`, for HTTP DELETE requests
- `init` and `destroy`, to manage resources that are held for the life of the servlet
- `getServletInfo`, which the servlet uses to provide information about itself

There's almost no reason to override the `service` method. `service` handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (the `doXXX` methods listed above).

Likewise, there's almost no reason to override the `doOptions` and `doTrace` methods.

Servlets typically run on multithreaded servers, so be aware that a servlet must handle concurrent requests and be careful to synchronize access to shared resources. Shared resources include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections. See the [Java Tutorial on Multithreaded Programming](#) for more information on handling multiple threads in a Java program.

Author:

Various

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor and Description

Requête : url / formulaire

```
public class PremsServlet extends HttpServlet
{
    public void doGet (HttpServletRequest req,
                      HttpServletResponse res)
                      throws ServletException, IOException {
        ACTIONS : GET, POST, PUT, DELETE, ...
    }
}
```

```
public void doPost (HttpServletRequest req,
                   HttpServletResponse res)
                   throws ServletException, IOException {
}
```

```
};
```

Réponse : page ouaib

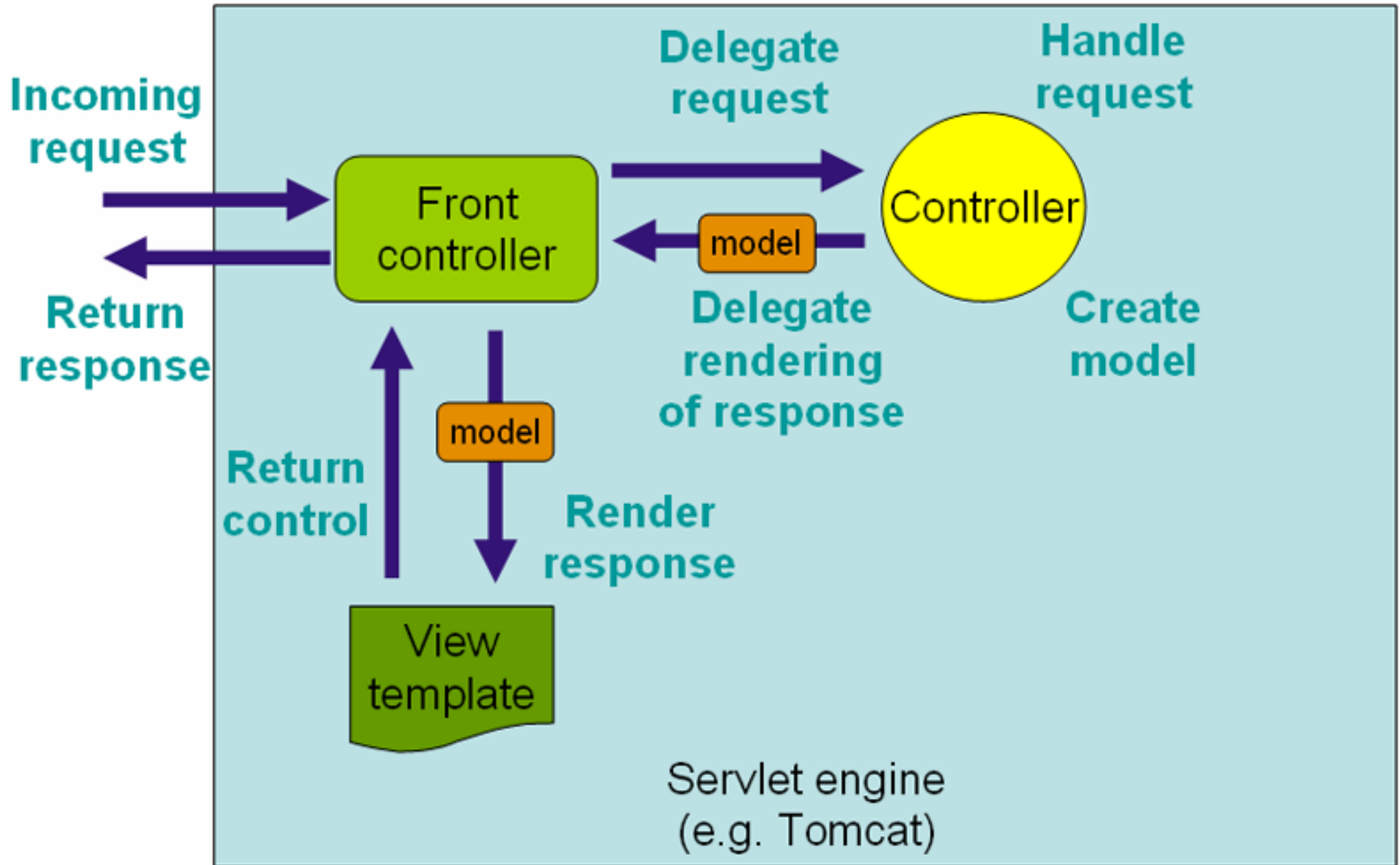


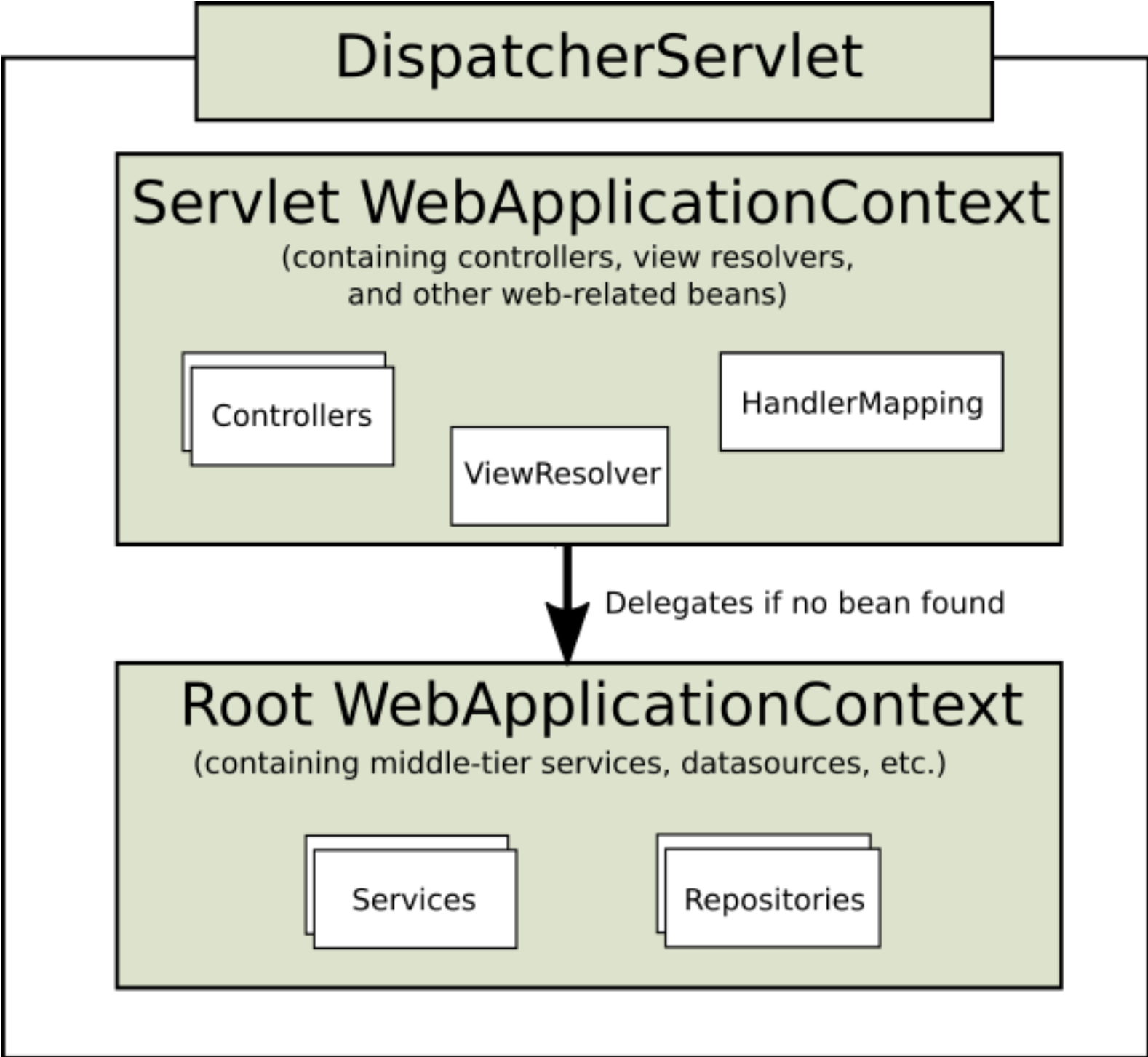
Première servlet

- Appel de la première servlet
- Manipulation d'un attribut
 - Tester avec différents navigateurs
- Affichage de paramètres en GET ou POST

Modèle MVC

- À la sauce web
- Avec un contrôleur principal (front controller pattern)
- Modèle
 - Des POJOs
 - Gestions des requêtes par annotation
- Vue
 - Des pages web le plus souvent
- Contrôleurs
 - Comment gérer les interactions utilisateurs
 - Requêtes







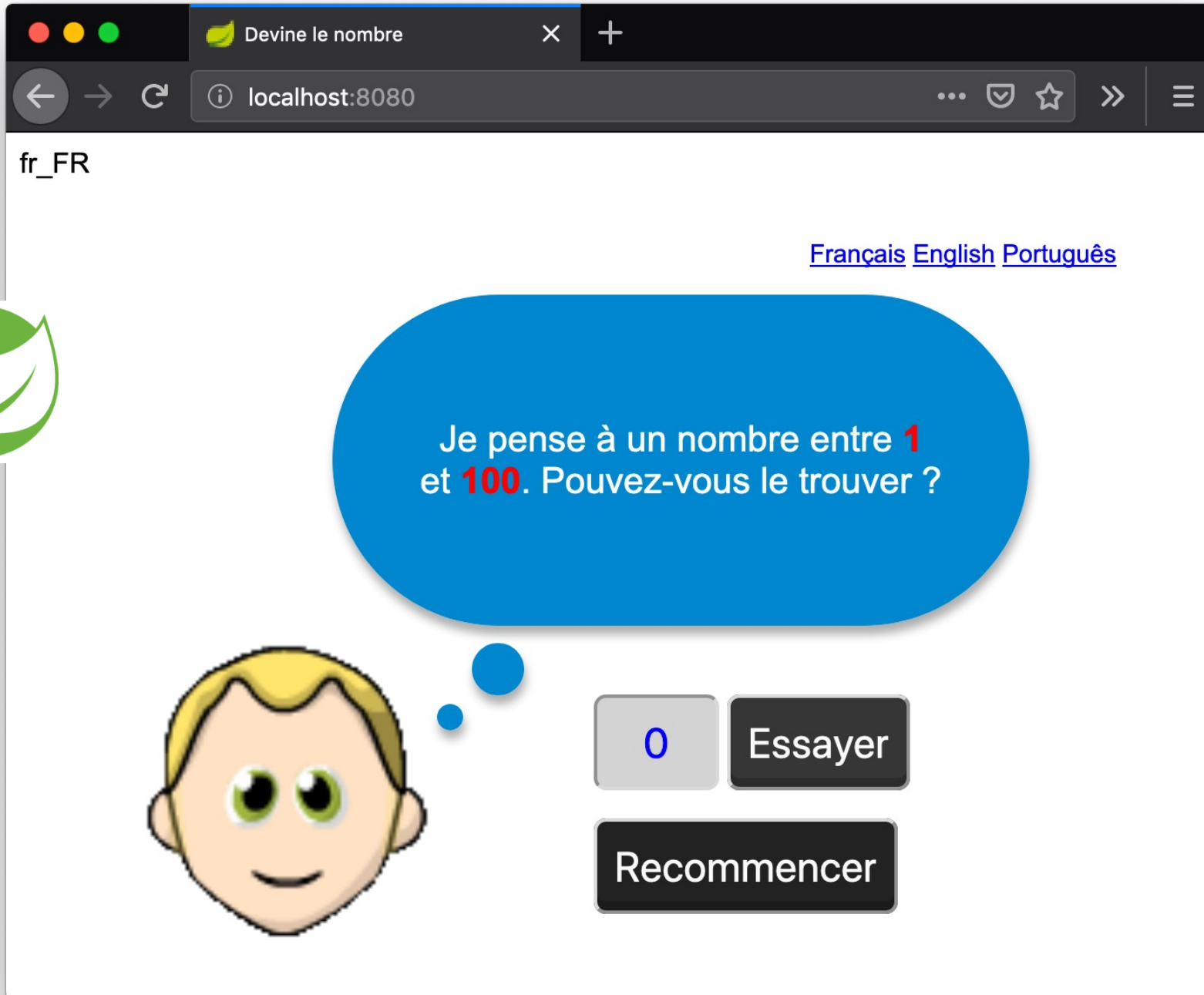
Page dynamique HTML

- Moteur de template ThymeLeaf
 - Enrichissement des balises HTML
 - Emplacement particulier
- Mise en place d'un contrôleur
 - Spring Web MVC
 - DispatcherServlet



Devine le nombre

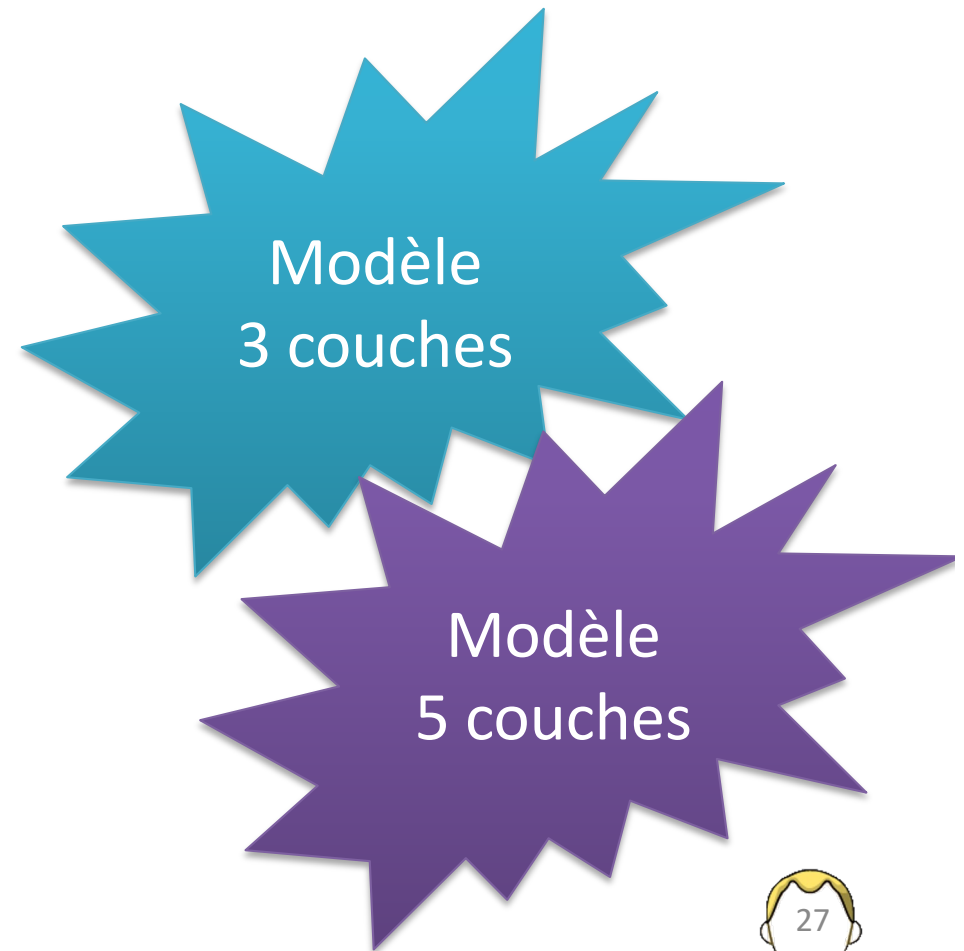
- Mise en place du formulaire et de l'interaction
 - Message « plus grand », « plus petit », « trouvé »
 - Compteur d'essais
- Gestion des erreurs sur la saisie
- Internationalisation



The screenshot shows a web browser window with a single tab titled "Devine le nombre". The address bar displays "localhost:8080". The page content includes the text "fr_FR" in the top left corner and language links "Français", "English", and "Português" in the top right. A large blue speech bubble contains the text "Je pense à un nombre entre 1 et 100. Pouvez-vous le trouver ?". Below the speech bubble is a cartoon character with blonde hair and green eyes. To the right of the character are three input elements: a text box containing the number "0", a dark button labeled "Essayer", and a larger dark button labeled "Recommencer". A green leaf icon is positioned to the left of the character's head.

Une application ?

- Couche de données (persistance)
 - Entité et bases de données, ORM
- Couche métier
 - Java : Beans
- Couche présentation
 - , ...
- Interopérable ?



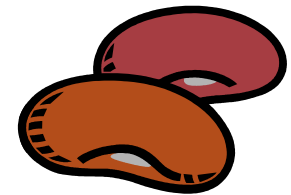
REST ?

- Interaction page web / API
- Verbes + URL

- <https://localhost/index.php?name=loic&qualite=rien>
- <https://monsite/users>
- <https://monsite/user/32>

Objets java ?

- Servlets / JSP
- Bean – *Plain Old Java Object* (POJO)
 - Constructeur par défaut
 - Propriétés avec getter / setter
 - Sérialisable
- Gestion de la vie de ces objets ?

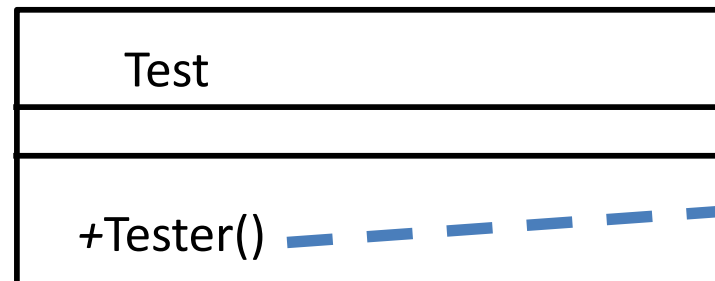
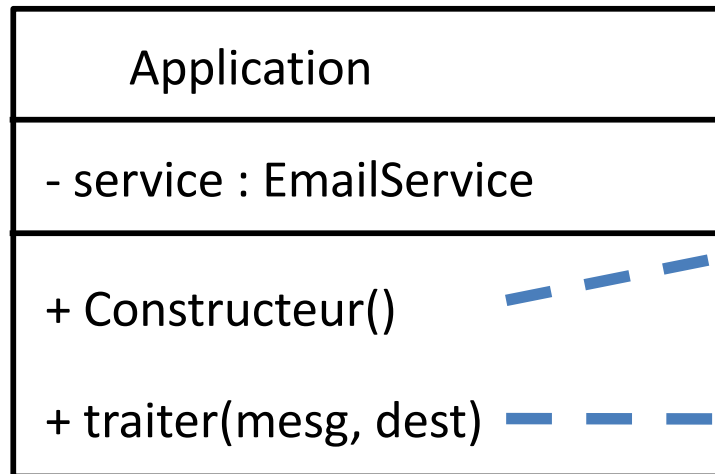
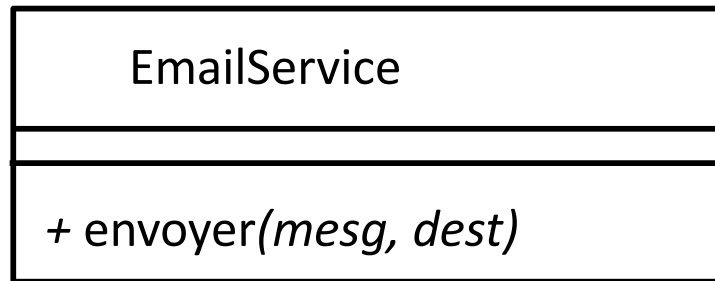


Injection de dépendances

- Inversion de contrôle
- Limiter les dépendances (composition / héritage)
- Résoudre à ~~la compilation~~ l'exécution

- Conteneur de dépendances ?
- Cycle de vie des objets

- Sympathique pour les tests
 - Mockups utilisables quand c'est compliqué



Dépendance codée en dur

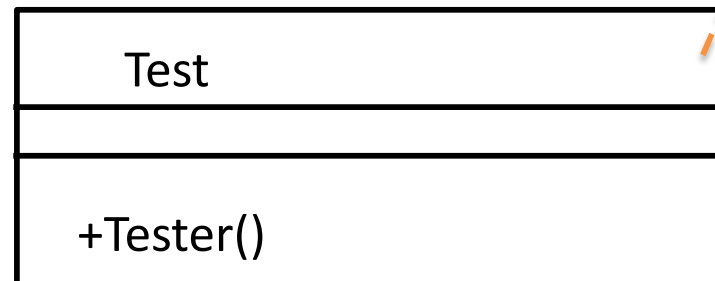
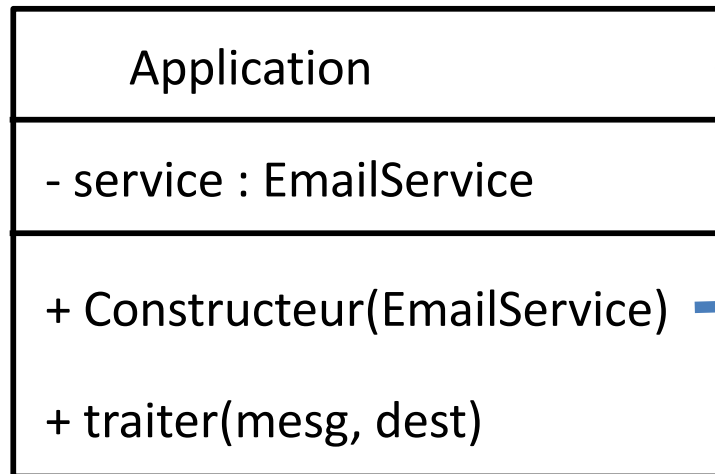
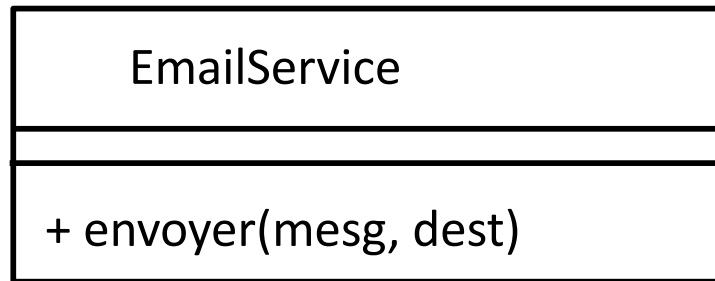
Changement de service ?

Instancier service

service.envoyer(msg, dest)

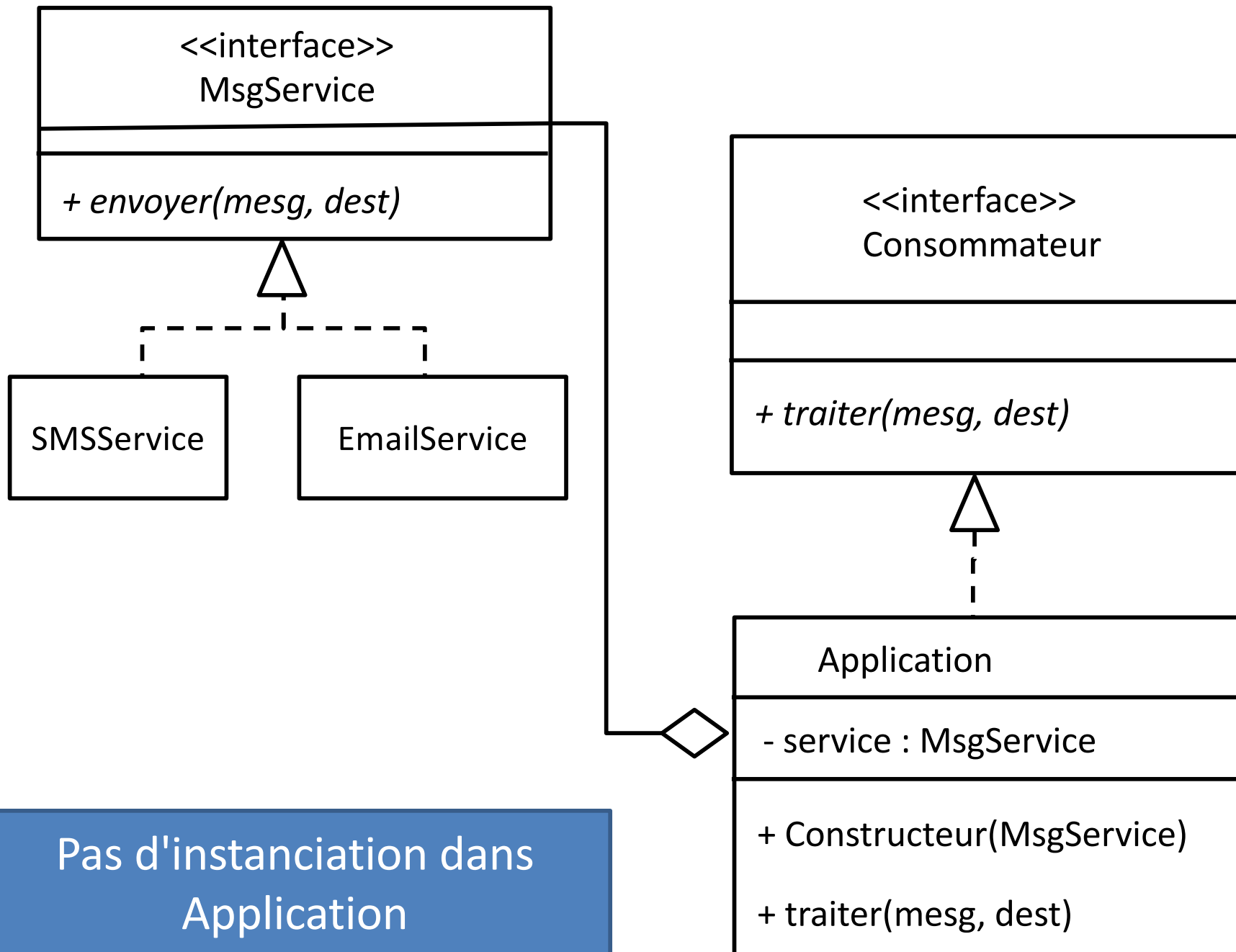
Test délicat...

Instancier Application

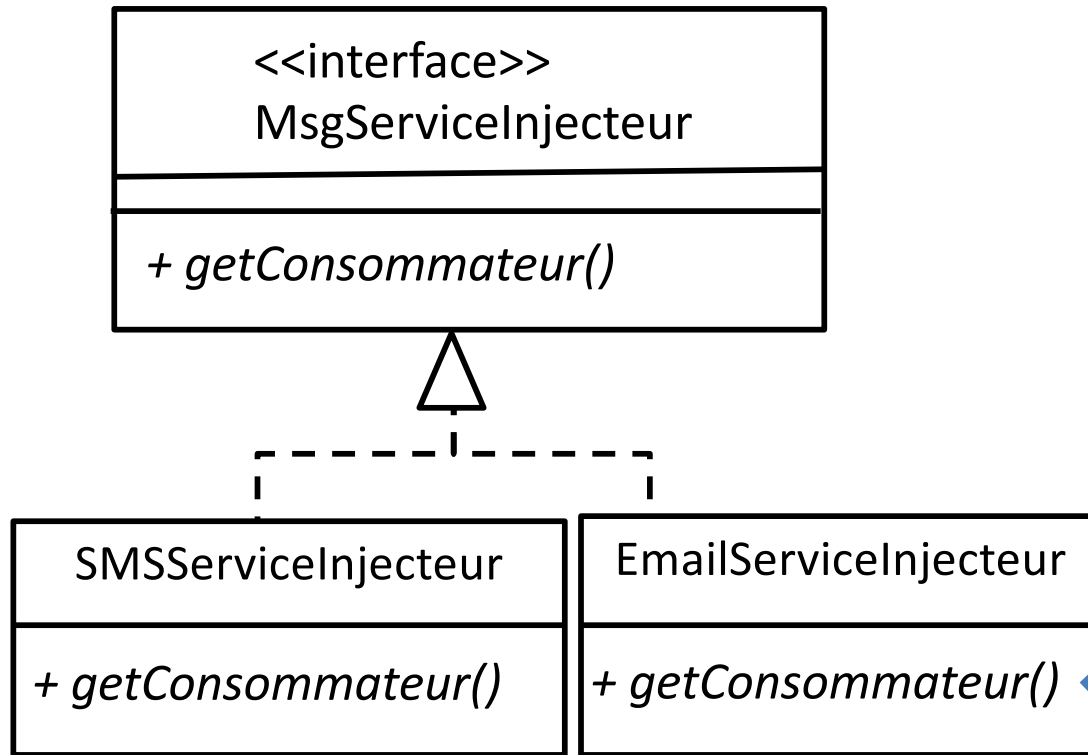


Qui instancie ?

Affectation



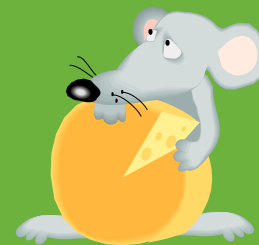
Pas d'instanciation dans Application



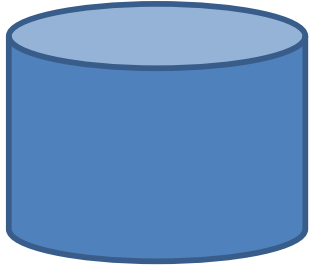
Instance l'application
Instance également le service spécial

Composant spécifique de gestion

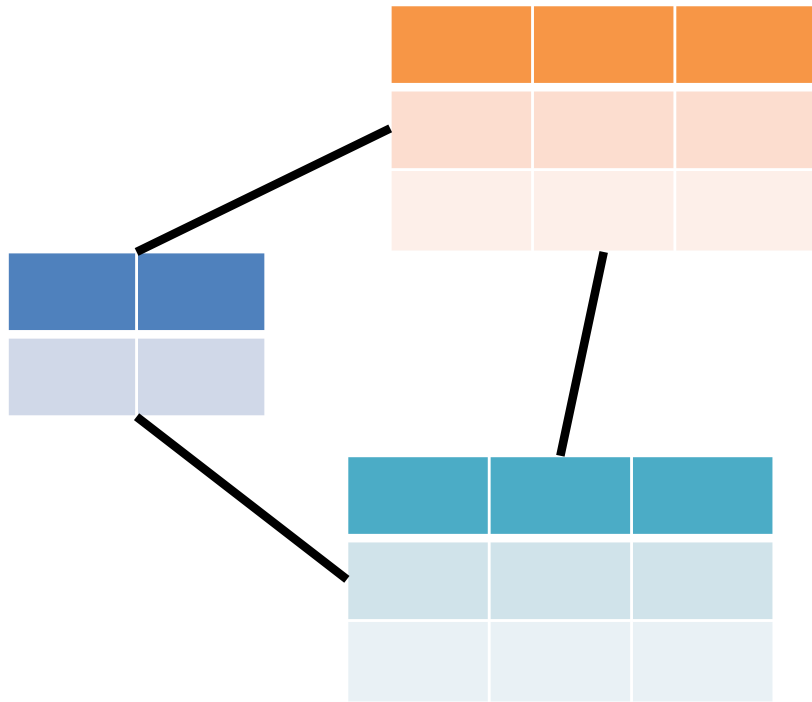
SPRING DATA



ISIMA 

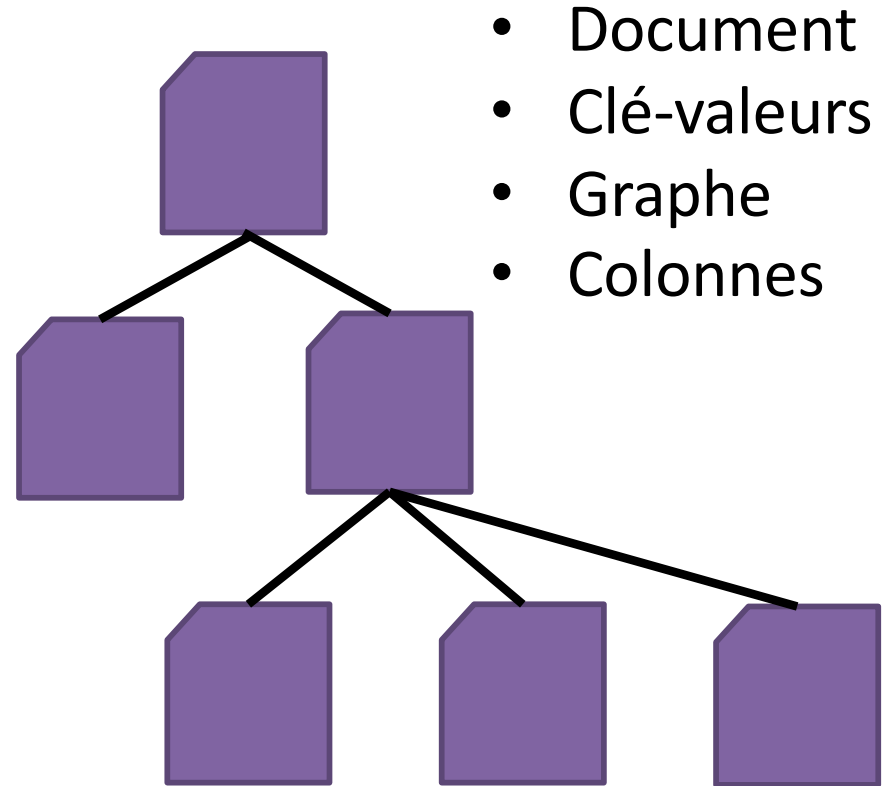


Bases de données ?

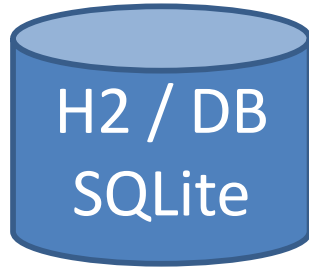
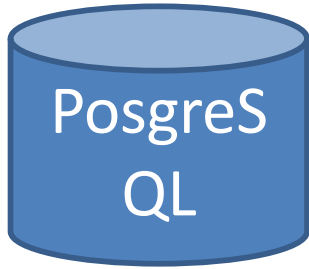
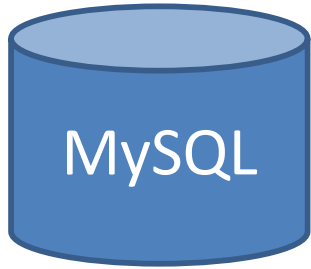
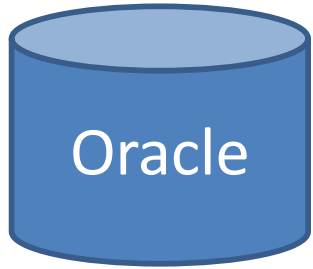


Relationnel

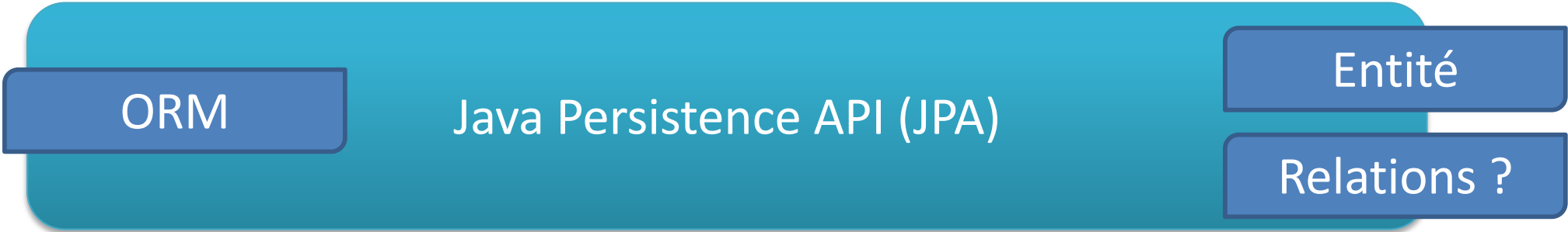
SQL

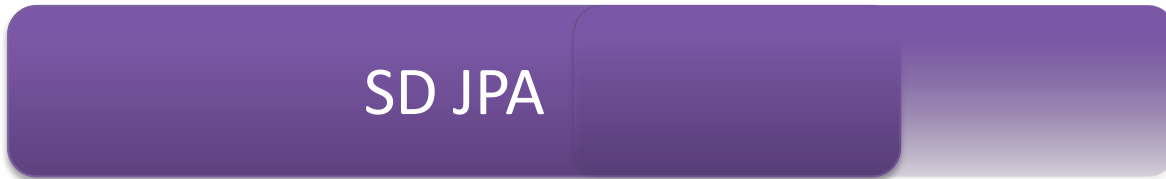
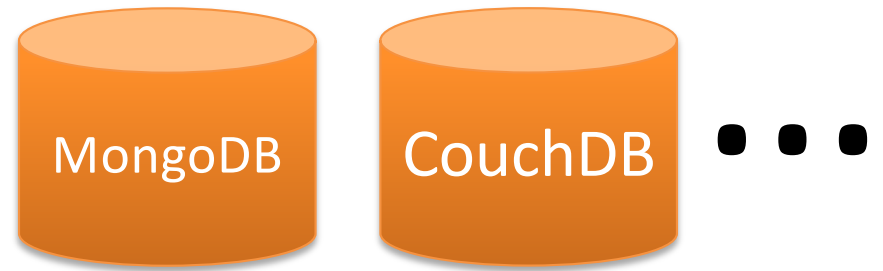
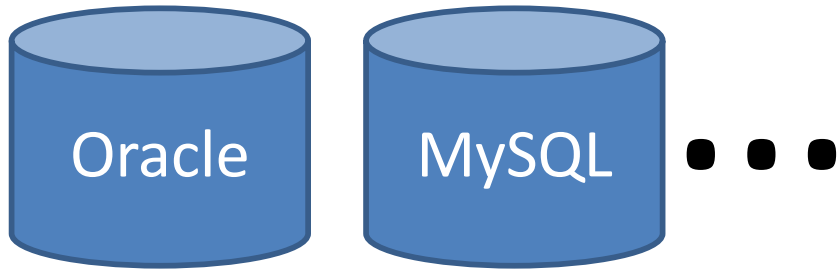


NOSQL



...





Spring Data JPA

- Entité / Classe / Table
- Attribut / Colonne
- Repository / opérations sur la base
- Méthode / requête

Données en base...

- Base relationnelles mais pas seulement
- ORM : Java Persistence API
 - Hibernate
 - EclipseLink
- Java DataBase Connectivity
 - Abstraction de la base
 - Connecteurs
 - SQLite