

# ARCHITECTURE DES COMPOSANTS D'ENTREPRISES



Pr. Sara Belattar  
Email : Sarah.belattar12@gmail.com

## Objectif du module



Créer des **applications distribuées** et sécurisées avec un **niveau abstraction très élevé**

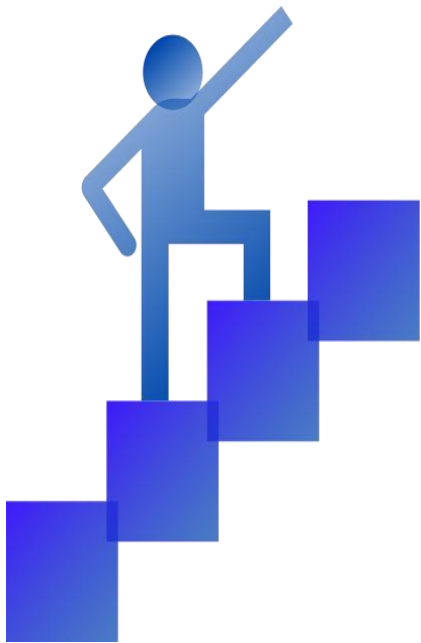


Une application distribuée est un système informatique où les composants ou services sont répartis sur plusieurs ordinateurs connectés par un réseau



Un niveau d'abstraction très élevé simplifie la complexité en se concentrant sur les aspects conceptuels et en masquant les détails techniques.

# Acquis d'apprentissages



1ER

Maîtriser IoC et  
architecture  
Microservices

2E

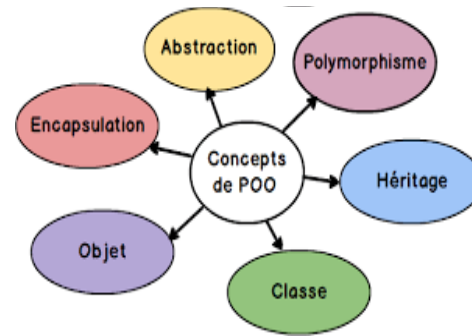
Maîtriser  
l'architecture  
distribuée

3E

Maîtriser les  
techniques  
d'authentification

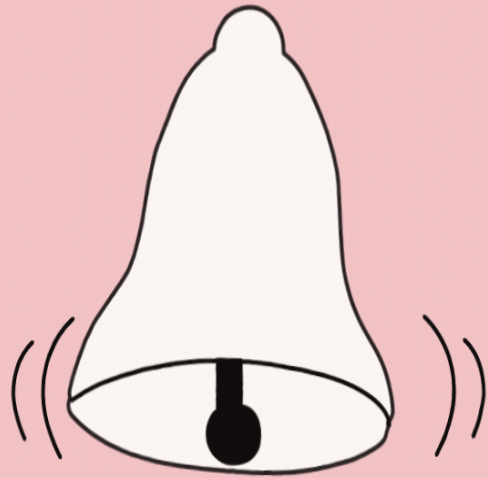
# Prérequis

JAVA, POO ET JAVA EE



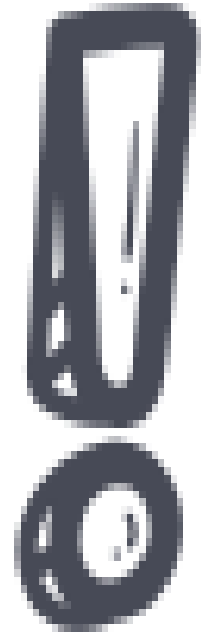
DESIGN PATTERNS (CONCEPTION)





**REMINDER**

Quelques Rappels



# **Java EE et Architecture applicative**

# Plan du cours

1. **Présentation de la plateforme J2EE**
2. **Les API de J2EE**
3. **Java EE 7.0**
4. **Les implémentations de Java EE 7.0**
5. **EJB**
6. **Les Annotations**
7. **JPA**
8. **Spring**
9. **Les Design Pattern : Singleton, Facade, IOC, AOP et le Value Objet.**

# La présentation de J2EE (1/3)

- J2EE est une plate-forme **fortement orientée serveur** pour le développement et l'exécution d'applications distribuées. Elle est composée de deux parties essentielles :
  - un ensemble de spécifications pour une infrastructure dans laquelle s'exécutent les composants écrits en Java : un tel environnement se nomme **serveur d'applications**.
  - un ensemble d'**API** qui peuvent être obtenues et utilisées séparément. Pour être utilisées, certaines nécessitent une implémentation de la part d'un fournisseur tiers.



# La présentation de J2EE (2/3)

L'utilisation de J2EE pour développer et exécuter une application offre plusieurs avantages :

- une architecture d'applications basée sur les composants qui permet un découpage de l'application et donc **une séparation des rôles** lors du développement
- la possibilité **de s'interfacer** avec le système d'information existant grâce à de nombreuses API : JDBC, JNDI, JMS, JCA ...
- la possibilité **de choisir** les outils de développement et le ou les serveurs d'applications utilisés qu'ils soient commerciaux ou libres.

# La présentation de J2EE (3/3)

J2EE permet **une grande flexibilité** dans le choix de l'architecture de l'application en combinant les différents composants. Ce choix dépend des besoins auxquels doit répondre l'application mais aussi des compétences dans les différentes API de J2EE. L'architecture d'une application se découpe idéalement en au moins trois tiers :

- **la partie cliente (couche Présentation)** : c'est la partie qui permet le dialogue avec l'utilisateur. Elle peut être composée d'une application standalone, d'une application web ou d'applets.
- **la partie métier (couche Métier ou Service)** : c'est la partie qui encapsule les traitements (dans des EJB ou des JavaBeans)
- **la partie données (couche DAO)** : c'est la partie qui stocke les données

# Les API de J2EE

| API  | version de l'API dans |     |       |     |     |     |
|--|-----------------------|-----|-------|-----|-----|-----|
|  | 1.2                   | 1.3 | 1.4   | 5   | 6   | 7   |
| Entreprise Java Bean (EJB)                   | 1.1                   | 2.0 | 2.1   | 3.0 | 3.1 | 3.2 |
| Remote Method Invocation (RMI) et RMI-IIOP   | 1.0                   |     |       |     |     |     |
| Java Naming and Directory Interface (JNDI)   | 1.2                   | 1.2 | 1.2.1 |     |     |     |
| Java Database Connectivity (JDBC)            | 2.0                   | 2.0 | 3.0   |     |     |     |
| Java Transaction API (JTA)                   | 1.0                   | 1.0 | 1.0   | 1.1 | 1.1 | 1.2 |
| Java Transaction Service (JTS)               |                       |     |       |     |     |     |
| Java Messaging service (JMS)                 | 1.0                   | 1.0 | 1.1   | 1.1 | 1.1 | 2.0 |
| Servlets                                     | 2.2                   | 2.3 | 2.4   | 2.5 | 3.0 | 3.1 |
| Java Server Pages (JSP)                      | 1.1                   | 1.2 | 2.0   | 2.1 | 2.2 | 2.2 |
| Java Server Faces (JSF)                      |                       |     |       | 1.2 | 2.0 | 2.2 |
| Expression Language (EL)                     |                       |     |       |     | 2.2 | 3.0 |
| Java Server Pages Standard Tag Libray (JSTL) |                       |     |       | 1.2 | 1.2 | 1.2 |

# Les API de J2EE

| API  | version de l'API dans |       |       |     |     |     |
|--|-----------------------|-------|-------|-----|-----|-----|
|  | 1.2                   | 1.3   | 1.4   | 5   | 6   | 7   |
| JavaMail   | 1.1                   | 1.2   | 1.3   | 1.4 | 1.4 | 1.4 |
| J2EE Connector Architecture (JCA)                                  |                       | 1.0   | 1.5   | 1.5 | 1.6 | 1.6 |
| Java API for XML Parsing (JAXP)                                    |                       | 1.1   | 1.2   |     |     |     |
| Java Authentication and Authorization Service (JAAS)               |                       | 1.0   |       |     |     |     |
| JavaBeans Activation Framework                                     |                       | 1.0.2 | 1.0.2 |     |     |     |
| Java API for XML-based RPC (JAXP-RPC)                              |                       |       | 1.1   | 1.1 | 1.1 | 1.1 |
| SOAP with Attachments API for Java (SAAJ)                          |                       |       | 1.2   | 1.3 |     |     |
| Java API for XML Registries (JAXR)                                 |                       |       | 1.0   | 1.0 | 1.0 | 1.0 |
| Java Management Extensions (JMX)                                   |                       |       | 1.2   |     |     |     |
| Java Authorization Service Provider Contract for Containers (JACC) |                       |       | 1.0   | 1.1 | 1.4 | 1.4 |
| Java API for XML-Based Web Services (JAX-WS)                       |                       |       |       | 2.0 | 2.2 | 2.2 |

# Les API de J2EE

| API   | version de l'API dans |     |     |     |     |     |
|---|-----------------------|-----|-----|-----|-----|-----|
|   | 1.2                   | 1.3 | 1.4 | 5   | 6   | 7   |
| Java Architecture for XML Binding (JAXB)    |                       |     |     | 2.0 | 2.2 |     |
| Streaming API for XML (StAX)                |                       |     |     | 1.0 |     |     |
| Java Persistence API (JPA)                  |                       |     |     | 1.0 | 2.0 | 2.1 |
| Java API for RESTful Web Services (JAX-RS)  |                       |     |     |     | 1.1 | 2.0 |
| Web Services                                |                       |     |     | 1.2 | 1.3 | 1.3 |
| Web Services Metadata for the Java Platform |                       |     |     |     | 2.1 | 2.1 |
| Java APIs for XML Messaging (JAXM)          |                       |     |     |     | 1.3 |     |
| Context and Dependency Injection (CDI)      |                       |     |     |     | 1.0 | 1.1 |
| Dependency Injection (DI)                   |                       |     |     |     | 1.0 |     |
| Bean Validation                             |                       |     |     |     | 1.0 | 1.1 |
| Managed beans                               |                       |     |     |     | 1.0 | 1.0 |

# Les API de J2EE

| API                                      | version de l'API dans |     |     |   |     |     |
|--|-----------------------|-----|-----|---|-----|-----|
|  | 1.2                   | 1.3 | 1.4 | 5 | 6   | 7   |
| Interceptors                             |                       |     |     |   | 1.1 | 1.1 |
| Common Annotations                       |                       |     |     |   | 1.1 | 1.1 |
| Java API for WebSocket                   |                       |     |     |   |     | 1.0 |
| Java API for JSON                        |                       |     |     |   |     | 1.0 |
| Concurrency Utilities for Java EE        |                       |     |     |   |     | 1.0 |
| Batch Applications for the Java Platform |                       |     |     |   |     | 1.0 |

- Ces API peuvent être regroupées en trois grandes catégories :
  - les composants : Servlet, JSP, JSF, EJB
  - les services : JDBC, JTA/JTS, JNDI, JCA, JAAS
  - la communication : RMI-IIOP, JMS, Java Mail

# L'environnement d'exécution des applications J2EE

- J2EE propose des spécifications pour une infrastructure dans laquelle s'exécutent les composants. Ces spécifications décrivent les rôles de chaque élément et précisent un ensemble d'interfaces pour permettre à chacun de ces éléments de communiquer.
- Ceci permet de séparer les applications et l'environnement dans lequel elles s'exécutent. Les spécifications précisent à l'aide des API un certain nombre de fonctionnalités que doit implémenter l'environnement d'exécution. Ces fonctionnalités permettent aux développeurs de se concentrer sur la logique métier.
- Pour exécuter ces composants de natures différentes, J2EE définit des conteneurs pour chacun d'eux. Il définit pour chaque composant des interfaces qui leur permettront de dialoguer avec les composants lors de leur exécution.

# Les conteneurs (1/2)

- Les conteneurs assurent la gestion du cycle de vie des composants qui s'exécutent en eux. Les conteneurs fournissent des services qui peuvent être utilisés par les applications lors de leur exécution.
- Il existe trois (03) conteneurs définis par J2EE:
  - **conteneur web** : pour exécuter les servlets, les JSP et les JSF.
  - **conteneur d'EJB** : pour exécuter les EJB.
  - **conteneur client** : pour exécuter des applications standalone sur les postes qui utilisent des composants J2EE.
- Pour déployer une application dans un conteneur, il faut lui fournir deux éléments :
  - l'application avec tous les composants (classes compilées, ressources ...) regroupées dans une archive ou module. Chaque conteneur possède son propre format d'archive.
  - un fichier descripteur de déploiement contenu dans le module qui précise au conteneur des options pour exécuter l'application.



## Les conteneurs (2/2)

- Il existe trois types d'archives :

| Archive / module   | Contenu  | Extension                | Descripteur de déploiement |
|--------------------|--|--------------------------|----------------------------|
| bibliothèque       | Regroupe des classes   | jar (Java ARchive)       |                            |
| application client | Regroupe les ressources nécessaires à leur exécution (classes, bibliothèques, images, ...)   | jar                      | application-client.jar     |
| web                | Regroupe les Servlets, les JSP et les JSF ainsi que les ressources nécessaires à leur exécution (classes, bibliothèques de balises, images, ...) | War (Web ARchive)        | web.xml                    |
| EJB                | Regroupe les EJB et leurs composants (classes)   | jar                      | ejb-jar.xml                |
| Application J2EE   | Regroupe un ou plusieurs modules   | ear (Entreprise ARchive) | application.xml            |

# Les services proposés par la plate-forme J2EE

- Une plate-forme d'exécution J2EE complète implémentée dans un serveur d'applications propose les services suivants :
  - service de nommage (naming service)
  - service de déploiement (deployment service)
  - service de gestion des transactions (transaction service)
  - service de sécurité (security service)
- Ces services sont utilisés directement ou indirectement par les conteneurs mais aussi par les composants qui s'exécutent dans les conteneurs grâce à leurs API respectives.

# J2EE historique

## J2EE 1.4 (Diffusée en novembre 2003)

- Le support des services web (nouveau)
- J2EE deployment API 1.1 (nouveau)
- J2EE management API 1.0 (nouveau)
- Java ACC : Java Authorization Contract for Container (nouveau)
- EJB 2.1 (update)

## Java EE 5.0 (Diffusée en 2007)

- Les annotations
- JSF
- JPA
- EJB 3.0
- Le support des dernières versions des API concernant les services web et permettant une mise en œuvre d'une architecture de type SOA.

## Java EE 6.0 (Diffusée en décembre 2009)

- Plus riche : de nouvelles spécifications sont ajoutées et les spécifications majeures sont enrichies
- Plus facile : généralisation de l'utilisation des POJO et des annotations notamment dans le tiers web
- Plus légère : création de la notion de profiles et de pruning, EJB Lite
- Toujours aussi robuste après 10 ans d'existence.
- L'implémentation de référence est le projet open source GlassFish version 3.

# La notion de Profile

- Java EE 6 définit la notion de profile qui est un sous-ensemble et/ou un sur-ensemble de Java EE 6 pour des besoins particuliers.
- Les profiles sont conçus pour proposer une solution technique particulière pour des besoins spécifiques. Cela permet à un fournisseur de n'avoir à proposer que le support des technologies incluses dans le profile plutôt que d'avoir à implémenter toutes celles de la plate-forme Java EE.
- Java EE 6 définit un premier profile : le web profile qui se concentre sur le développement d'applications de type web. Il doit pouvoir être exécuté dans un simple conteneur web car le packaging se fait dans une archive de type war.
- Le web profile est composé de plusieurs spécifications pour définir un sous-ensemble de Java EE : Servlet 3.0, JSP 2.2, EL 1.2, JSTL 1.2, JSF 2.0, EJB Lite 3.1, JTA 1.1, JPA 2.0, Bean Validation 1.0, DI 1.0, CDI 1.0 et Interceptors 1.1.

# La notion de Pruning (1/2)

- La plate-forme Java EE est devenue au fil des versions imposante en terme de nombre de spécifications, de fonctionnalités et d'API. Aucune des précédentes versions n'a supprimé de fonctionnalités même si certaines sont obsolètes car remplacées, rarement adoptées ou utilisées. Cela pose plusieurs problèmes :
  - Pour le développeur : la taille du SDK augmente avec le nombre d'API
  - Pour les fournisseurs de serveurs d'applications : l'obligation de support pour ces fonctionnalités avec un accroissement de la taille des serveurs, de leur consommation en ressources et de leur temps de démarrage.
- Certaines de ces fonctionnalités sont de plus obsolètes car remplacées par une plus récente ou ne sont que peu ou pas utilisées.
- Java EE 6 entame donc une cure d'amaigrissement de la plate-forme en définissant un ensemble d'API déclarées comme **pruned**.

## La notion de Pruning (2/2)

- Les fonctionnalités déclarées **pruned** dans Java EE 6 sont :
  - Entity CMP 2.x : cette API est avantageusement remplacée par JPA
  - JAX-RPC : cette API est avantageusement remplacée par JAX-WS
  - JAX-R : cette API est très peu utilisée car l'utilisation de UDDI n'est pas très répandue
  - JSR 88 (Java EE Application Deployment) : cette API est très peu mise en œuvre par les fournisseurs de serveurs d'applications
- Elles doivent toujours être disponibles dans une implémentation de Java EE 6 mais elles seront amenées à disparaître dans les prochaines versions de la plate-forme Java EE et leur support ne sera plus obligatoire dans les implémentations des serveurs d'applications.
- Le concept de **pruned** est plus fort que le concept de **deprecated** de la plate-forme Java SE.



# La présentation de Java EE 7.0



# Java EE 7.0 (1/5)

- Les spécifications de Java EE 7 sont définies dans la **JSR 342**. Cette spécification ne définit pas directement d'API mais elle liste celles qui sont incluses dans la plate-forme et comment celles-ci interagissent entre-elles.
- Elle définit aussi des éléments précis de la plate-forme comme les transactions, la sécurité, l'assemblage, le déploiement, ...
- Java EE 7 a plusieurs objectifs :
  - poursuivre l'amélioration de la productivité et la simplification de l'utilisation de la plate-forme.
  - le support de HTML 5 (WebSocket, Json, HTML5 forms, ...).
  - l'ajout de nouvelles fonctionnalités : Batch API (modèle de programmation pour les applications batch) et Concurrency Utilities API (exécution de traitements asynchrones sans avoir à utiliser JMS).
- Les spécifications de Java EE 7 furent officiellement diffusées **en juin 2013**.



# Java EE 7.0 (2/5)



# Java EE 7.0 (3/5)

- Java EE 7 inclut plusieurs **nouvelles API** :
  - Java API for JSON Processing 1.0 ([JSR 353](#))
  - Java API for WebSocket 1.0 ([JSR 356](#))
  - Batch Applications for the Java Platform 1.0 ([JSR 352](#))
  - Concurrency utilities for Java EE 1.0 ([JSR 236](#))
- Java EE 7 inclut des API ayant **une mise à jour majeure** :
  - Java Message Service 2.0 ([JSR 343](#))
  - Expression Language 3.0 ([JSR 341](#))
  - JAX-RS : Java API for Restful Web Services 2.0 ([JSR 339](#))
- Java EE 7 inclut des API mises à jour :
  - JPA 2.1 ([JSR 338](#))
  - EJB 3.2 ([JSR 345](#))
  - Servlet 3.1 ([JSR 340](#))
  - Java Server Faces 2.2 ([JSR 344](#))
  - Contexts and Dependency Injection for Java EE 1.1 ([JSR 346](#))
  - Bean Validation 1.1 ([JSR 349](#))
  - Interceptors 1.2 (JSR 318)
  - Java EE Connector Architecture 1.7 ([JSR 322](#))

# Java EE 7.0 (4/5)

- Les JSR mises à jour (Maintenance Release) sont :
  - Web Services for Java EE 1.4 ([JSR 109](#))
  - Java Authorization Service Provider Contract for Containers 1.5 (JACC 1.5) ([JSR 115](#))
  - Java Authentication Service Provider Interface for Containers 1.1 (JASPIC 1.1) ([JSR 196](#))
  - JavaServer Pages 2.3 ([JSR 245](#))
  - Common Annotations for the Java Platform 1.2 ([JSR 250](#))
  - Java Transaction API 1.2 ([JSR 907](#))
  - JavaMail 1.5 ([JSR 919](#))
- Initialement Java EE 7 devait intégrer la spécification Java Temporary Caching API 1.0 (JSR 107) et des fonctionnalités relatives au Cloud (par exemple State Management (JSR 350)) mais elles sont repoussées dans la prochaine version de Java EE.

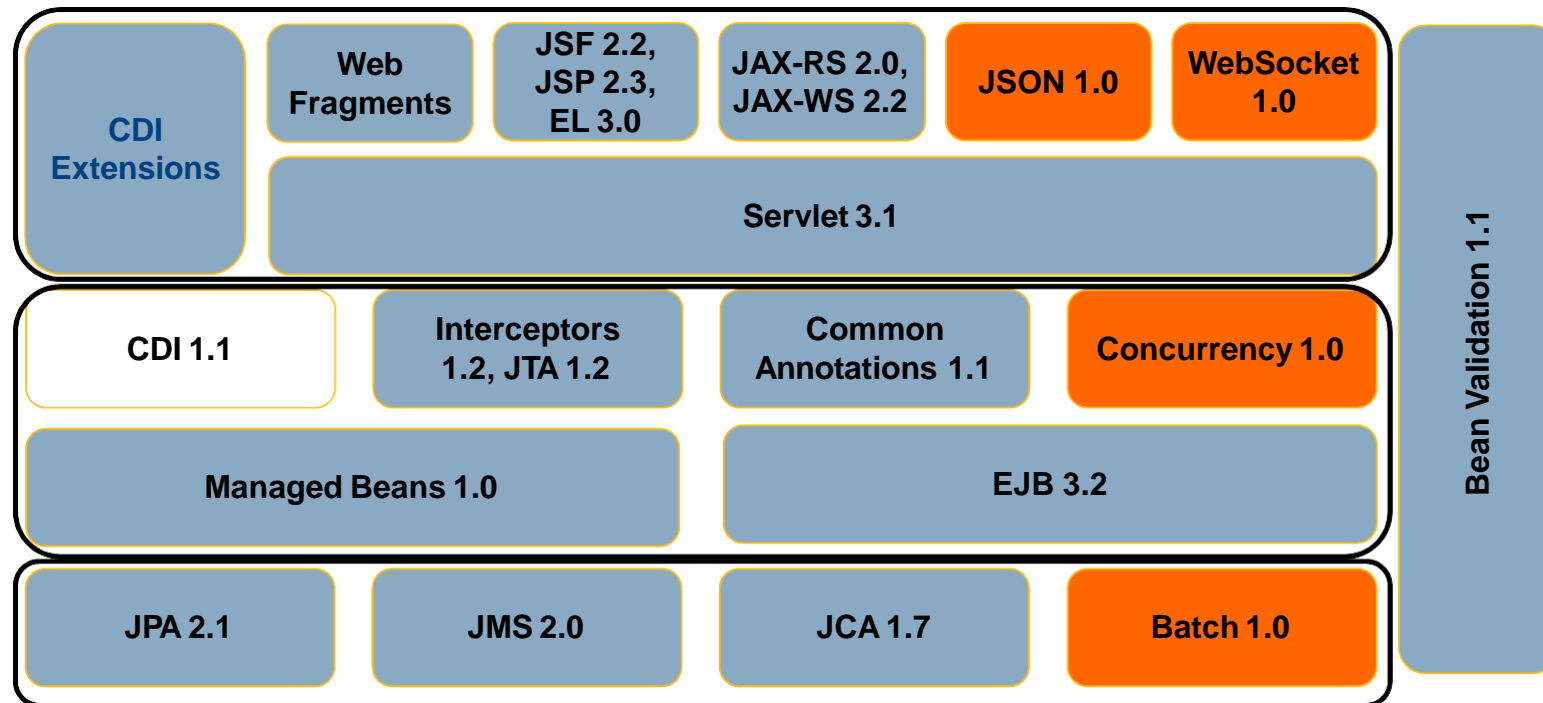
## Java EE 7.0 (5/5)

- Java EE 7 poursuit la simplification de l'utilisation de la plate-forme entamée depuis Java EE 5 notamment :
  - la nouvelle version 2.0 de l'API JMS
  - des changements dans la configuration : ressources de type fabrique de connexions par défaut pour la base de données et le broker JMS, amélioration dans la déclaration des permissions de sécurité, ...
  - les technologies déclarées pruned dans Java EE 6 sont optionnelles dans Java EE 7 : EJB Entity Beans, JAX-RPC 1.1, JAXR 1.0 et la JSR-88
- JAX-RS 2.0 est ajoutée dans le Web Profile.
- L'implémentation de référence est la **version 4.0 du serveur d'applications Glassfish**.

# Top Ten Features dans Java EE 7

1. WebSocket client/server endpoints
2. Batch Applications
3. JSON Processing
4. Concurrency Utilities
5. Simplified JMS API
6. @Transactional et @TransactionScoped
7. JAX-RS Client API
8. Default Resources
9. More annotated POJOs
10. Faces Flow

# Java EE 7 JSRs



- Les spécifications (JSR) sont disponibles moyennant le lien suivant : <https://java.net/projects/javaee-spec/pages/Home>

# Java EE 7.0



<http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html>

# Java EE 7.0 : les serveurs

## Java EE 7 Full Platform Compatible Implementations



GlassFish Server Open Source Edition 4.0

Tested Configuration



Wildfly 8.x

Tested Configuration



IBM WebSphere Application Server  
Version 8.5.5.6(Liberty Profile)

Tested Configuration

**RED HAT JBOSS**  
ENTERPRISE  
APPLICATION PLATFORM 7

Red Hat JBoss Enterprise Application  
Platform 7.0

Tested Configuration



TMAX JEUS 8

Tested Configuration



Cosminexus: Hitachi Application Server  
v10.0

Tested Configuration



**FUSION MIDDLEWARE**  
WEBLOGIC SERVER

Oracle Weblogic Server 12.2.1

Tested Configuration



IBM WebSphere Application Server  
Version 9.x

Test Configuration



# Java EE 7.0 Web Profile : les serveurs

## Java EE 7 Web Profile Compatible Implementations



GlassFish Server Open Source Edition 4.0  
Web Profile

Tested Configuration



IBM WebSphere Application Server  
Version 8.5.5.6(Liberty Profile)

Tested Configuration



IBM WebSphere Application Server  
Version 9.x

Test Configuration



Wildfly 8.x Web Profile

Tested Configuration

**RED HAT JBOSS**  
ENTERPRISE  
APPLICATION PLATFORM 7

Red Hat JBoss Enterprise Application  
Platform 7.0

Tested Configuration



# Enterprise Java Bean (EJB)



# Qu'est-ce qu'EJB ?

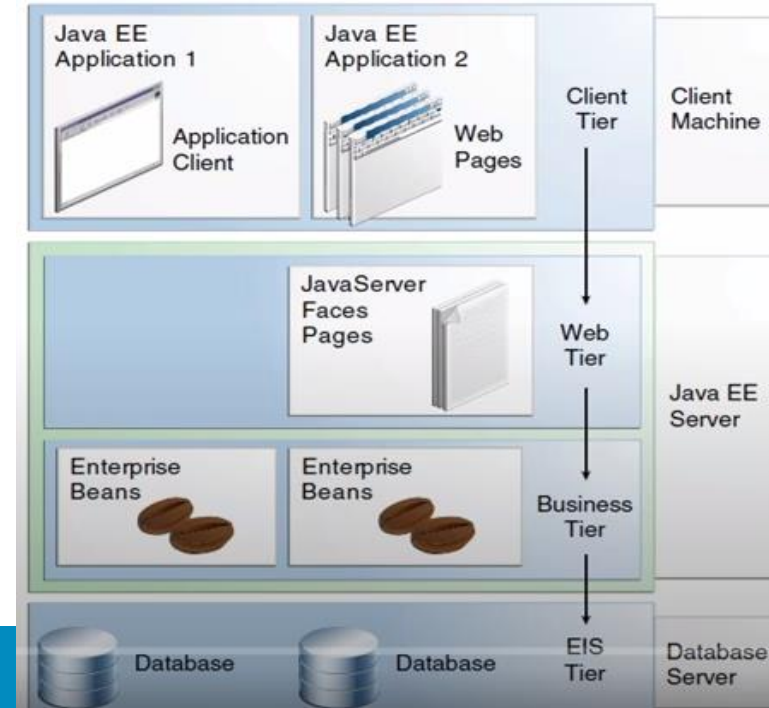
- EJB signifie Enterprise JavaBeans.
- Une spécification de composants pour le développement d'applications d'entreprise en Java.

## Rôles d'EJB

- EJB fournit une architecture pour développer des composants réutilisables et gérés par le serveur.
- Il facilite la construction d'applications d'entreprise robustes et évolutives.

## Types d'EJB

- EJB est divisé en trois types principaux :
  - ✓ EJB Session Beans
  - ✓ EJB Entity Beans (plus utilisés avec Java EE 6 et versions antérieures)
  - ✓ EJB Message-Driven Beans



## EJB Session Beans

- Utilisés pour gérer la logique métier de l'application.
- Deux sous-types : Stateless et Stateful Session Beans.
- Stateless : sans état, conçus pour des opérations sans état, idéaux pour la montée en charge.
- Stateful : conservent l'état pour une session client, utilisés pour des interactions de session.

## EJB Entity Beans (Legacy)

- Utilisés pour représenter les données dans la base de données.
- Remplacés par JPA (Java Persistence API) dans les versions Java EE ultérieures.

## EJB Message-Driven Beans

- Utilisés pour la gestion des messages asynchrones.
- Réagissent aux messages provenant de systèmes externes.

### Exemple: Un EJB sans état simple

```
import javax.ejb.Stateless;  
@Stateless  
public class CalculatorEJB{  
    public double add(double v1,double v2) {  
        return v1+v2;  
    }  
}
```

# Caractéristiques clés d'EJB

- Transaction Management : Prise en charge de la gestion de transactions distribuées.
- Sécurité : Gestion des autorisations d'accès aux méthodes EJB.
- Cycle de vie géré par le conteneur : Le serveur d'application gère la création, la destruction et la réutilisation des composants EJB.

# Annotations dans Java EE

- Les annotations sont des métadonnées intégrées au code source Java pour fournir des informations supplémentaires au compilateur ou à d'autres outils.
- Dans Java EE, les annotations sont largement utilisées pour simplifier la configuration et la gestion des composants.

## Avantages des annotations

- Réduction de la configuration XML : Les annotations permettent de spécifier des informations directement dans le code source, évitant ainsi la configuration XML volumineuse.
- Meilleure lisibilité : Les annotations sont plus lisibles et maintenables que la configuration XML.

# Annotations couramment utilisées

**@Entity** : Utilisée pour marquer une classe comme une entité persistante dans JPA (Java Persistence API).

**@Servlet** : Définit une servlet dans une application web.

**@EJB** : Injecte une référence à un composant EJB.

**@WebService** : Indique qu'une classe est un point de terminaison de service web.

**@Resource** : Injecte une ressource, telle qu'une source de données ou une file d'attente, dans une classe.

**@Path** : Spécifie le chemin URI d'une ressource RESTful.

**@TransactionAttribute** : Contrôle le comportement de transaction d'une méthode EJB.

## Personnalisation avec des métadonnées

Vous pouvez personnaliser le comportement de vos composants en utilisant des annotations avec des paramètres.

Exemple : **@Column(name = "nom\_utilisateur")** dans JPA permet de spécifier le nom de la colonne dans la base de données.



# Validation avec Bean Validation

**Java EE** inclut également Bean Validation, qui utilise des annotations pour définir des règles de validation sur les propriétés des classes.

Exemple : **@NotNull**, **@Size**, **@Email**, etc.

## Annotations personnalisées

Vous pouvez créer vos propres annotations personnalisées pour simplifier la configuration répétitive et ajouter des métadonnées spécifiques à votre application.

# Introduction à JPA dans JEE

- JPA signifie Java Persistence API.
- Une spécification Java EE qui définit une interface commune pour la gestion des données dans les applications Java EE.

## Objectif de JPA

- Simplifier l'accès aux bases de données relationnelles dans les applications Java EE.
- Permettre la persistance des objets Java dans des bases de données.

## Caractéristiques clés de JPA

- Mapping Objet-Relationnel (ORM) : Associe des classes Java à des tables de base de données.
- Gestion de la persistance : Gère automatiquement la création, la mise à jour et la suppression des enregistrements en fonction de l'état des objets.
- Langage de requête JPQL (Java Persistence Query Language) : Un langage de requête orienté objet pour interroger les données.

# Entités JPA

- Les classes annotées avec **@Entity** sont utilisées pour représenter des entités persistantes.
- Les instances de ces classes sont gérées par le contexte de persistance.

## Relations entre entités

- JPA permet de définir des relations entre les entités, telles que les associations un-à-un, un-à-plusieurs, et plusieurs-à-plusieurs.

## Avantages de JPA

- Réduction de la dépendance au fournisseur de base de données grâce à l'utilisation de l'API standard.
- Portabilité des applications entre différents systèmes de gestion de base de données.

# Spring Boot

- Si vous êtes amenés à travailler sur une application développée autour de l'architecture Microservices en Java, vous aurez tôt ou tard affaire à Spring Boot.
- Spring Boot est un framework qui facilite le développement d'applications fondées sur Spring en offrant des outils permettant d'obtenir une application packagée en jar, totalement autonome

## Ce que propose Spring Boot

- **L'auto-configuration**

Cette fonctionnalité est la plus importante de Spring Boot. Elle permet de **configurer automatiquement** votre application à partir des *jar* trouvés dans votre Classpath.

Exemple :

Prenons l'exemple d'une application web dans laquelle vous avez les dépendances : Hibernate et Spring MVC. Normalement, vous devez créer les fichiers de configuration suivants :

- appconfig-mvc.xml
- web.xml
- persitence.xml

# Ce que propose Spring Boot

- Comme vous ne connaissez pas nécessairement la syntaxe de ces fichiers par cœur, il vous faut consulter la documentation ou vous inspirer d'un ancien projet. Vous devez ensuite écrire le code Java qui permet de lier ces fichiers XML à *ApplicationContext* de Spring

Voici l'équivalent de l'ensemble de ces étapes avec Spring MVC :

```
1 @EnableAutoConfiguration
```

## Les Starters

- Les starters viennent compléter l'auto-configuration et font gagner énormément de temps, notamment lorsqu'on commence le développement d'un Microservice.

Prenons l'exemple où vous souhaitez créer un Microservice. En temps normal, vous aurez besoin des dépendances suivantes :

- Spring ;
- Spring MVC ;
- Jackson (pour json) ;
- Tomcat ;
- ...

Avec Spring Boot, vous allez tout simplement avoir une seule dépendance dans votre **pom.xml** :

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-web</artifactId>
4 </dependency>
```



Tous les starters de Spring Boot sont au format *spring-boot-starter-NOM\_DU\_STARTER*

Ce starter va charger les dépendances présentes dans le pom suivant :

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
  4.0.0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <parent>
5     <groupId>org.springframework.boot</groupId>
6     <artifactId>spring-boot-starters</artifactId>
7     <version>1.5.9.RELEASE</version>
8   </parent>
9   <artifactId>spring-boot-starter-web</artifactId>
10  <name>Spring Boot Web Starter</name>
11  <description>Starter for building web, including RESTful, applications using Spring
12    MVC. Uses Tomcat as the default embedded container</description>
13  <url>http://projects.spring.io/spring-boot/</url>
14  <organization>
15    <name>Pivotal Software, Inc.</name>
16    <url>http://www.spring.io</url>
17  </organization>
18  <properties>
19    <main.basedir>${basedir}/../..</main.basedir>
20  </properties>
21  <dependencies>
22    <dependency>
23      <groupId>org.springframework.boot</groupId>
24      <artifactId>spring-boot-starter</artifactId>
25    </dependency>
26    <dependency>
27      <groupId>org.springframework.boot</groupId>
```

```
28     <artifactId>spring-boot-starter-tomcat</artifactId>
29 </dependency>
30 <dependency>
31     <groupId>org.hibernate</groupId>
32     <artifactId>hibernate-validator</artifactId>
33 </dependency>
34 <dependency>
35     <groupId>com.fasterxml.jackson.core</groupId>
36     <artifactId>jackson-databind</artifactId>
37 </dependency>
38 <dependency>
39     <groupId>org.springframework</groupId>
40     <artifactId>spring-web</artifactId>
41 </dependency>
42 <dependency>
43     <groupId>org.springframework</groupId>
44     <artifactId>spring-webmvc</artifactId>
45 </dependency>
46 </dependencies>
47 </project>
```



| Name   | Description  |
|--|--|
| <code>spring-boot-starter</code>                         | Core starter, including auto-configuration support, logging and YAML   |
| <code>spring-boot-starter-activemq</code>                | Starter for JMS messaging using Apache ActiveMQ  |
| <code>spring-boot-starter-amqp</code>                    | Starter for using Spring AMQP and Rabbit MQ  |
| <code>spring-boot-starter-aop</code>                     | Starter for aspect-oriented programming with Spring AOP and AspectJ  |
| <code>spring-boot-starter-artemis</code>                 | Starter for JMS messaging using Apache Artemis   |
| <code>spring-boot-starter-batch</code>                   | Starter for using Spring Batch   |
| <code>spring-boot-starter-cache</code>                   | Starter for using Spring Framework's caching support   |
| <code>spring-boot-starter-cloud-connectors</code>        | Starter for using Spring Cloud Connectors which simplifies connecting to services in cloud platforms like Cloud Foundry and Heroku |
| <code>spring-boot-starter-data-cassandra</code>          | Starter for using Cassandra distributed database and Spring Data Cassandra   |
| <code>spring-boot-starter-data-cassandra-reactive</code> | Starter for using Cassandra distributed database and Spring Data Cassandra Reactive  |
| <code>spring-boot-starter-data-couchbase</code>          | Starter for using Couchbase document-oriented database and Spring Data Couchbase   |
| <code>spring-boot-starter-data-couchbase-reactive</code> | Starter for using Couchbase document-oriented database and Spring Data Couchbase Reactive  |

## Design patterns

(Quelques rappels)

### Historique



Les design patterns (**motifs de conception en français**) sont des solutions éprouvées aux problèmes de conception de logiciels qui se sont développées au fil du temps dans le domaine de la programmation informatique.

Les design patterns, ou motifs de conception, ont vu leur émergence dans le domaine du développement logiciel à partir des années 1960 et 1970 avec des langages tels que Simula et Smalltalk.

Toutefois, leur formalisation en tant que concept clair est survenue en 1994 avec la publication du livre "Design Patterns: Elements of Reusable Object-Oriented Software" par le groupe connu sous le nom de "Gang of Four" (GoF)



## Design patterns

(Quelques rappels)

### Définitions

#### Un Design pattern décrit à la fois

- Un problème qui se produit très fréquemment, dans un environnement,
- et l'architecture de la solution à ce problème de telle façon que l'on puisse utiliser cette solution des milliers de fois.
- Permet de décrire avec succès des types de solutions récurrentes à des problèmes communs dans des types de situations




## Design patterns

(Quelques rappels)

### Définitions

Les design patterns offrent :

- Une documentation d'une expérience éprouvée de conception
  - Une identification et spécification d'abstractions qui sont au dessus du niveau des simples classes et instances
  - Un vocabulaire commun et aide à la compréhension de principes de conception
  - Une Aide à la construction de logiciels complexes et hétérogènes, répondant à des propriétés précises.
- 

# Design patterns (Quelques rappels)

## Catégories de Design Patterns

### Création

- Description de la manière dont un objet ou un ensemble d'objets peuvent être créés, initialisés, et configurés
- Isolation du code relatif à la création, à l'initialisation afin de rendre l'application indépendante de ces aspects

**Exemples:** Abstract Factory, Builder, Prototype, Singleton

### Structure

- Description de la manière dont doivent être connectés des objets de l'application afin de rendre ces connections indépendantes des évolutions futures de l'application

**Exemples:** Adapter(objet), Composite, Bridge, Decorator, Facade, Proxy

### Comportement

- Description de comportements d'interaction entre objets
- Gestion des interactions dynamiques entre des classes et des objets

**Exemples:** Strategy, Observer, Iterator, Mediator, Visitor, State

# Design patterns (Quelques rappels)

## Portée des Design Patterns

### Portée de Classe

- Focalisation sur les relations entre classes et leurs sous-classes
- Réutilisation par héritage

### Portée d'Instance (Objet)

- Focalisation sur les relations entre les objets
- Réutilisation par composition

## Design Patterns du GoF (Gang of Four) (Gamma, Helm, Johnson, Vlissides)

| PURPOSE       |                   |                   |                    |
|---------------|-------------------|-------------------|--------------------|
| SCOPE         | <u>CREATIONAL</u> | <u>STRUCTURAL</u> | <u>BEHAVIOURAL</u> |
| <u>CLASS</u>  | Factory Method    | Adapter (class)   | Interpreter        |
|               |                   |                   | Template Method    |
|               |                   |                   |                    |
| <u>OBJECT</u> | Abstract Factory  | Adapter (object)  | Command.           |
|               | Builder           | Bridge            | Iterator           |
|               | Prototype         | Composite         | Mediator           |
|               | Singleton         | Decorator         | Memento            |
|               |                   | Facade            | Observer           |
|               |                   | Flyweight         | State              |
|               |                   | Proxy             | Strategy           |
|               |                   |                   | Visitor            |
|               |                   |                   | Chain Of Resp.     |

## Exemple : Création \_Singleton Pattern

L'objectif principal du **Singleton Pattern** (**motif de conception Singleton en français**) est de garantir qu'une classe n'a qu'une seule instance dans un programme et de fournir un point d'accès global à cette instance.



Création d'une seule instance



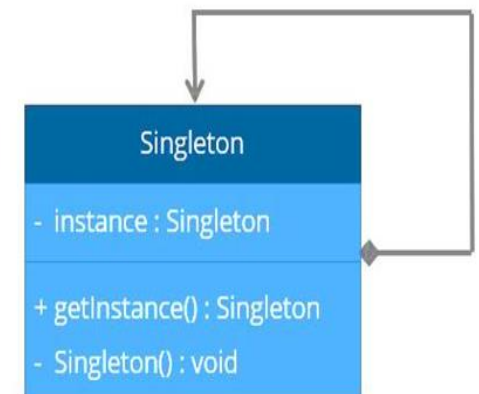
Accès global



Contrôle de l'instanciation



Éviter la consommation excessive de mémoire





## Exemple : Création \_Singleton Pattern

Voici un exemple en Java

```
:
public class Singleton
{
    private static Singleton theInstance = null;
    // Le constructeur en privé pour interdire l'instanciation de classe de
    //l'extérieur
    private Singleton() {}

    // On passera par cette méthode pour instancier la classe
    public static Singleton getInstance()
    {
        if (theInstance == null)
            theInstance = new Singleton();
        return theInstance;
    }
}
```

## Exemple : Structure Facade **Method-Pattern**

Le design pattern Facade (ou Patron de Facade) est un modèle de conception structurelle qui vise à simplifier l'interface d'un sous-système complexe en fournissant une interface unifiée et de haut niveau pour l'accès à ce sous-système.



Simplifier l'interface



Masquer la complexité

Supposons que vous ayez un système audio complexe composé de plusieurs composants, tels qu'un lecteur de CD, un amplificateur et des haut-parleurs. Plutôt que de faire interagir directement les clients avec chaque composant, vous pouvez utiliser le design pattern Facade pour simplifier l'accès à ce système audio.

Voici un exemple de code en Java pour illustrer cela :

```
// Sous-système complexe : Lecteur de CD
class CDPlayer {
    public void on() {
        System.out.println("Lecteur de CD allumé");
    }

    public void play() {
        System.out.println("Lecture du CD en cours");
    }

    public void off() {
        System.out.println("Lecteur de CD éteint");
    }
}
```

```
// Sous-système complexe : Amplificateur
class Amplifier {
    public void on() {
        System.out.println("Amplificateur allumé");
    }

    public void setVolume(int volume) {
        System.out.println("Réglage du volume à " + volume);
    }

    public void off() {
        System.out.println("Amplificateur éteint");
    }
}
```

```
// Sous-système complexe : Haut-parleurs
class Speakers {
    public void on() {
        System.out.println("Haut-parleurs allumés");
    }

    public void playSound() {
        System.out.println("Diffusion du son");
    }

    public void off() {
        System.out.println("Haut-parleurs éteints");
    }
}
```

```
class HomeTheaterFacade {  
    private CDPlayer cdPlayer;  
    private Amplifier amplifier;  
    private Speakers speakers;  
  
    public HomeTheaterFacade() {  
        this.cdPlayer = new CDPlayer();  
        this.amplifier = new Amplifier();  
        this.speakers = new Speakers();  
    }  
  
    public void watchMovie() {  
        System.out.println("Préparation pour regarder un film...");  
        cdPlayer.on();  
        cdPlayer.play();  
        amplifier.on();  
        amplifier.setVolume(5);  
        speakers.on();  
        speakers.playSound();  
    }  
  
    public void endMovie() {  
        System.out.println("Arrêt du film...");  
        cdPlayer.off();  
        amplifier.off();  
        speakers.off();  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        // Utilisation de la Facade pour simplifier l'utilisation du système  
        HomeTheaterFacade homeTheater = new HomeTheaterFacade();  
        homeTheater.watchMovie();  
        homeTheater.endMovie();  
    }  
}
```

## Constat

Il est très difficile de développer un système logiciel qui respecte ces exigences sans utiliser l'expérience des autres

- Réutiliser des modèles de conceptions (Design Patterns)
- Java, POO
- Bâtir les applications sur des architectures existantes: Architecture JEE

Ces architectures offrent :

- Des Frameworks qui permettent de satisfaire les exigences techniques:
- Framework pour l'Inversion de contrôle
- Gérer le cycle de vie des composants de l'application
- Séparer le code métier du code technique
- Framework ORM (Mapping Objet Relationnel) Framework MVC WEB