



JEE : (Eclipse) Jakarta EE

(anciennement *Java 2 Enterprise Edition*, ou J2EE, puis *Java Enterprise Edition* ou Java EE),

FSTG Marrakech
BENHADDI Meriem



Exigence d'un projet informatique

- Exigences fonctionnelles

- ☐ Une application est créée d'abord pour répondre aux besoins fonctionnels des entreprises/clients

- Exigences techniques

- ☐ Les performances : temps de réponse, haute disponibilité et tolérance aux pannes, montée en charge,
- ☐ L'évolutivité : une application doit être fermée à la modification mais ouverte à l'extension
- ☐ Sécurité, portabilité, environnement distribués, services multiples (JMS, JTS, persistance, etc)

- Exigences financières de coût



Les besoins

■ **Besoins de normalisation :**

- Etablir un ensemble de règles ayant pour objet de simplifier et de rationaliser la production
- pour que les applications soient:
 - intégrables
 - communicantes / distribuées
 - adaptables
 - maintenables
 - portables

Les besoins

■ Besoins d'abstraction :

- Opération de désolidariser un objet de son contexte
- pour que les applications soient :
 - portables
 - maintenables
 - extensibles
 - intégrables / distribuées
 - adaptables



Les besoins

- Besoins de communication pour que les applications soient :
 - ☐ intégrables
 - ☐ sécurisée
 - ☐ distribuées



Les besoins

- Besoins de composants pour que les applications soient :
 - ☐ maintenables
 - ☐ sûres
 - ☐ extensibles
 - ☐ adaptables
 - ☐ portables
 - ☐ disponibles / distribuées

Et surtout :

- Comment réduire les temps et les coûts de développement et d'évolutions d'une application ?
- **Principe d'ouverture/fermeture:**
 - Les composantes d'une application doivent être ouvertes à extension mais fermées à modification !
 - Comment respecter ce principe ?
 - Des paradigmes de programmation
 - Des patrons de conception
 - Des frameworks
 - Des composants



Mise en œuvre de :

- **Inversion de contrôle/Injection de dépendances**

- ☐ Une manière automatique et directe de fournir une dépendance externe dans un composant logiciel

- **Programmation par aspects**

- ☐ Augmenter la modularité en améliorant la séparation des préoccupations : séparer l'aspect technique et l'aspect métier

- **Design pattern**

- ☐ Une solution générale et réutilisable d'un problème courant (nous utilisons les design patterns sans forcément le savoir).
Avantages : Abstraction et capitalisation de la connaissance.



Objectif

- Avoir une « plateforme » pour développer des applications d'entreprise rapidement, de qualités, sûres, sécurisées, portables, performantes, disponibles, maintenables, extensibles etc... et à moindre coûts !



Java EE

- Java Enterprise Edition est une spécification pour la plate-forme Java d'Oracle, destinée aux applications d'entreprise.
- La plate-forme étend *Java Platform, Standard Edition* (Java SE) en fournissant une API de mapping objet-relationnel, des architectures distribuées et multitiers, des modèles de composants et des services web
- Spring est apparu en 2003 et a commencé à concurrencer Java EE, poussant cette dernière à revoir ses principes (EJB3, conteneur léger, etc).
- En 2017, Java EE est renommée Jakarta EE suite au transfert du projet sur la communauté Eclipse.
- Aujourd'hui, bien que Jakarta EE et Spring sont concurrents, on peut les utiliser conjointement pour tirer partie des meilleurs aspects des 2 technologies.



Java EE est :

1. Une technologie

outils liés au langage Java + des
spécifications

ET

2. Un modèle de développement

applications découpées en tiers

Qu'est-ce que Java EE ?

- 1- Une technologie:
 - Le langage Java
 - La machine virtuelle (JVM)
 - Des APIs (le JDK + APIs applicatives)
 - Des serveurs respectant le standard Java EE (JSR)

C'est la plate-forme Java EE

Qu'est-ce que Java EE ?

- 2- Un modèle de développement:
 - Développement en tiers (multitiers) : applications découpées logiquement (correspondance avec le déploiement : clients, serveurs, SGBDs,...)
 - Ce modèle partitionne le travail en 2 parties :
 - Les aspects métiers/présentation, à la charge du développeur
 - Les services standards fournies par la plate-forme Java EE

API Java EE

Les API de Java EE peuvent être regroupées en trois grandes catégories :

- Technologies composants
 - Contient la partie la plus importante de l'application : la logique métier
 - 3 types de composants : les JSP, les Servlets et les EJB
- Technologies de services
 - fournissent aux composants de l'application des services connexes leur permettant de s'interfacer et communiquer avec d'autres systèmes: JDBC, JTA/JTS, JNDI, JCA (intégration avec des systèmes autres que J2EE: C++, COBOL), JAAS (sécurité: authentification)
- Technologies de communication
 - prévoient les mécanismes de communication entre les différentes parties de l'application, qu'elles soient locales ou distantes: RMI-IIOP, JMS, Java Mail

Principes architecturaux de Java EE

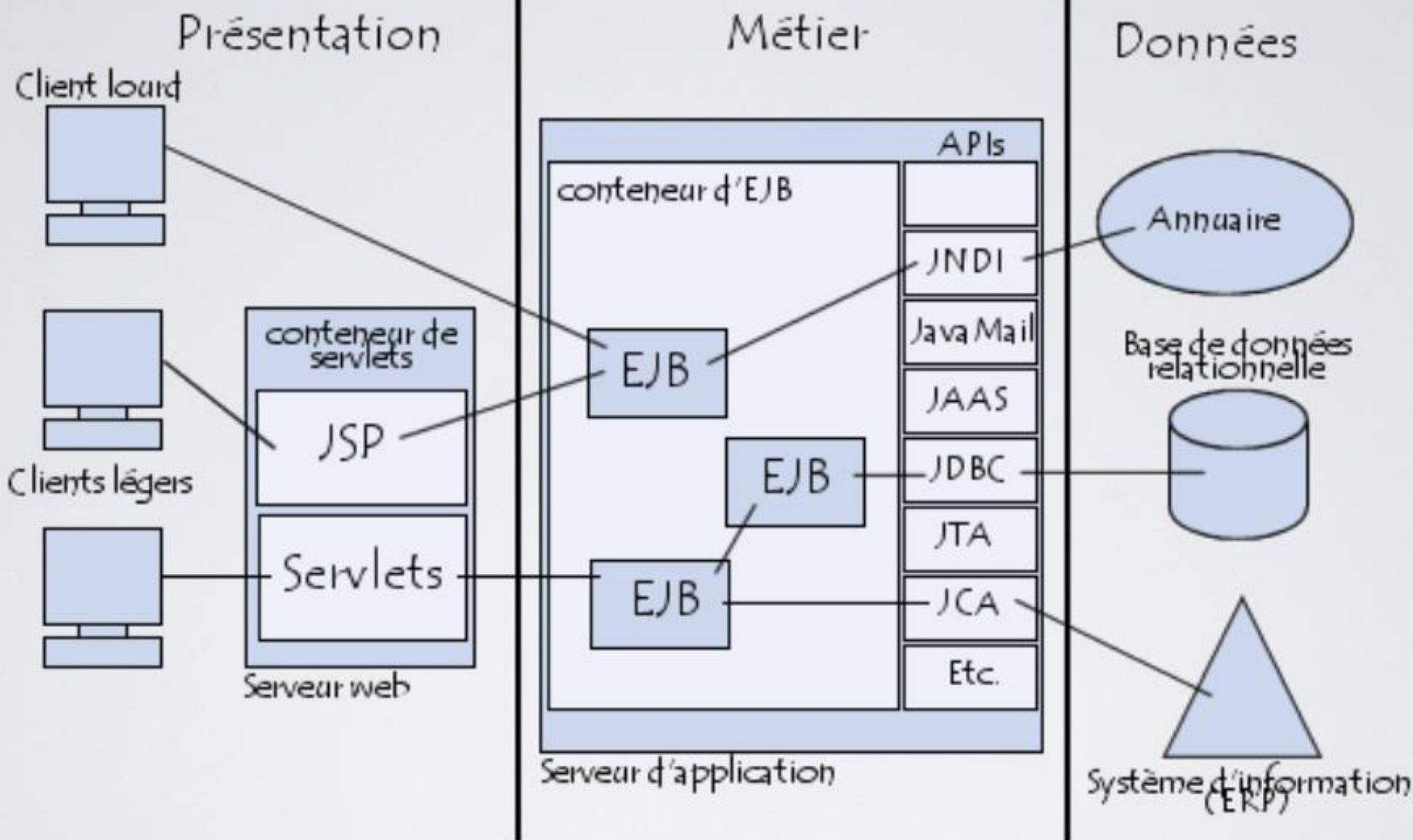
1- Architecture multi-niveaux

- Augmentation de la cohésion du code
- Découplage fort entre les couches
- Code plus facilement réutilisable

■ JEE est une architecture 3-tiers :

- Tiers présentation : affichage des données. **Client-tier**
- Tiers métier : gestion du métier de l'application. **Web-tier et Business-tier**
- Tiers donnée : persistance des données

Permet une séparation claire entre l'interface homme-machine, les traitements métiers et les données. **EIS-tier** (Enterprise Information System Tier: ERP, système transactionnel, base de données, etc)





Principes architecturaux de Java EE

2- Basée sur des composants qui sont :

- ☐ Distincts
- ☐ Interchangeables
- ☐ distribués

3- De nouveaux patrons de conception:

- ☐ Data Access Object
- ☐ Data Transfer Object
- ☐ Session Facade
- ☐ Front controller
- ☐ Modèle Vue Contrôleur



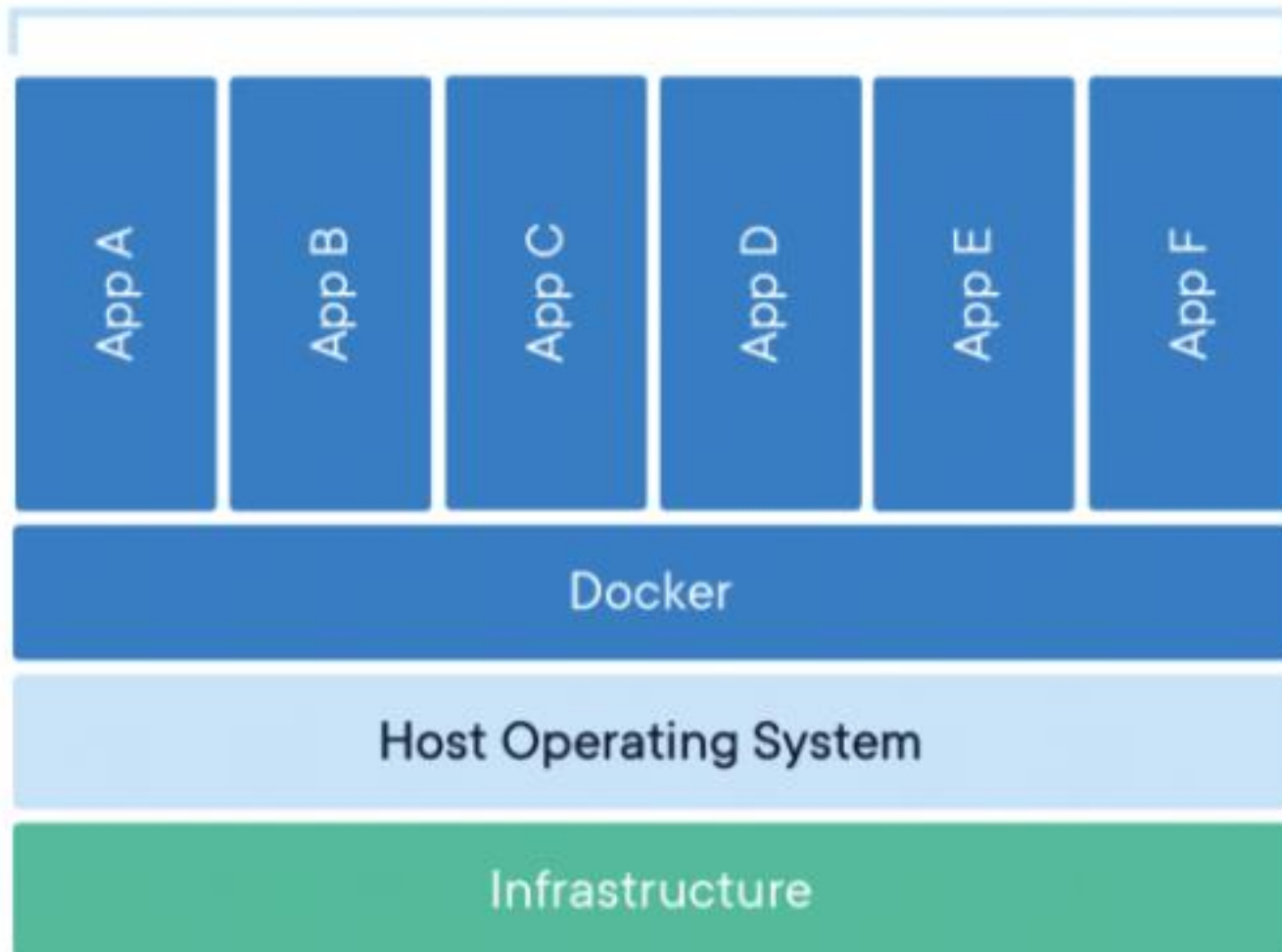
Notion de serveur d'application

- Un serveur d'application Java est une implémentation de la spécification Java EE Application Server qui est entièrement conforme et prend en charge toutes les fonctionnalités Java EE.
- Ces fonctionnalités étendent la plate-forme Java standard pour une utilisation dans les transactions commerciales basées sur Internet et incluent des éléments tels que JTA, EJB, JMS, JSF et API Java EE pour la persistance de session, l'injection de dépendances, etc.
- Les serveurs Java EE déploient des applications au format EAR (Enterprise Archive), qui contiennent un ou plusieurs fichiers WAR, des EJBs associés, des pages JSP, des adaptateurs de ressources. Ce paquet est déployé dans un conteneur unique, de sorte que ses classes et ressources peuvent être chargées et gérées séparément.

Notion de conteneur

- Les conteneurs sont une solution au problème de l'exécution fiable des logiciels lorsqu'ils sont déplacés d'un environnement informatique à un autre.
- Un conteneur est une unité logicielle standard qui regroupe le code et toutes ses dépendances afin que l'application s'exécute rapidement et de manière fiable d'un environnement informatique à un autre.
- Une image de conteneur, Docker par exemple, est un package logiciel léger, autonome et exécutable qui comprend tout ce dont vous avez besoin pour exécuter une application : code, environnement d'exécution, outils système, bibliothèques système et paramètres.

Containerized Applications



Notion de conteneur JEE

- Pour exécuter les composants de natures différentes, Java EE définit des conteneurs pour chacun de ces composants. Il définit pour chaque composant des interfaces qui leur permettront de dialoguer avec les autres composants lors de leur exécution. Les conteneurs permettent aux applications d'accéder aux ressources et aux services en utilisant les API.
- Un container est fourni par un serveur. Il permet l'exécution des composants qu'ils détient et fournit :
 - La gestion des composants (cycle de vie, injection, ...). Cette gestion implique la création de nouvelles instances de composants applicatifs ainsi que le pooling (rassemblement) et la destruction de ces composants lorsque les instances ne sont plus nécessaires
 - Les services de « bas niveaux » : Répartition de charge ou "Load Balancing »
 - Sécurité, transaction, JNDI, injection, communication inter container, RMI, ...



Notion de conteneur JEE

- Le découpage en composants est le premier point fort de la technologie Java EE.
 - Le deuxième est la gestion par conteneur des composants : pour chaque type de composants, le serveur Java EE définit un conteneur qui fournit les services associés.
- À chaque component son container
- Le déploiement consiste à placer les composants dans les conteneurs adaptés.
 - Exemple : conteneur web (JSF et servlet/JSP), conteneur EJB, conteneur Application Client



Notion de conteneur JEE

- Un conteneur (container) est l'interface entre le composant et les services de bas niveaux nécessaires
- Pour pouvoir être exécuté, un composant / application web doit être :
 - (1) assemblé dans un module Java EE
 - (2) déployé dans son conteneur.

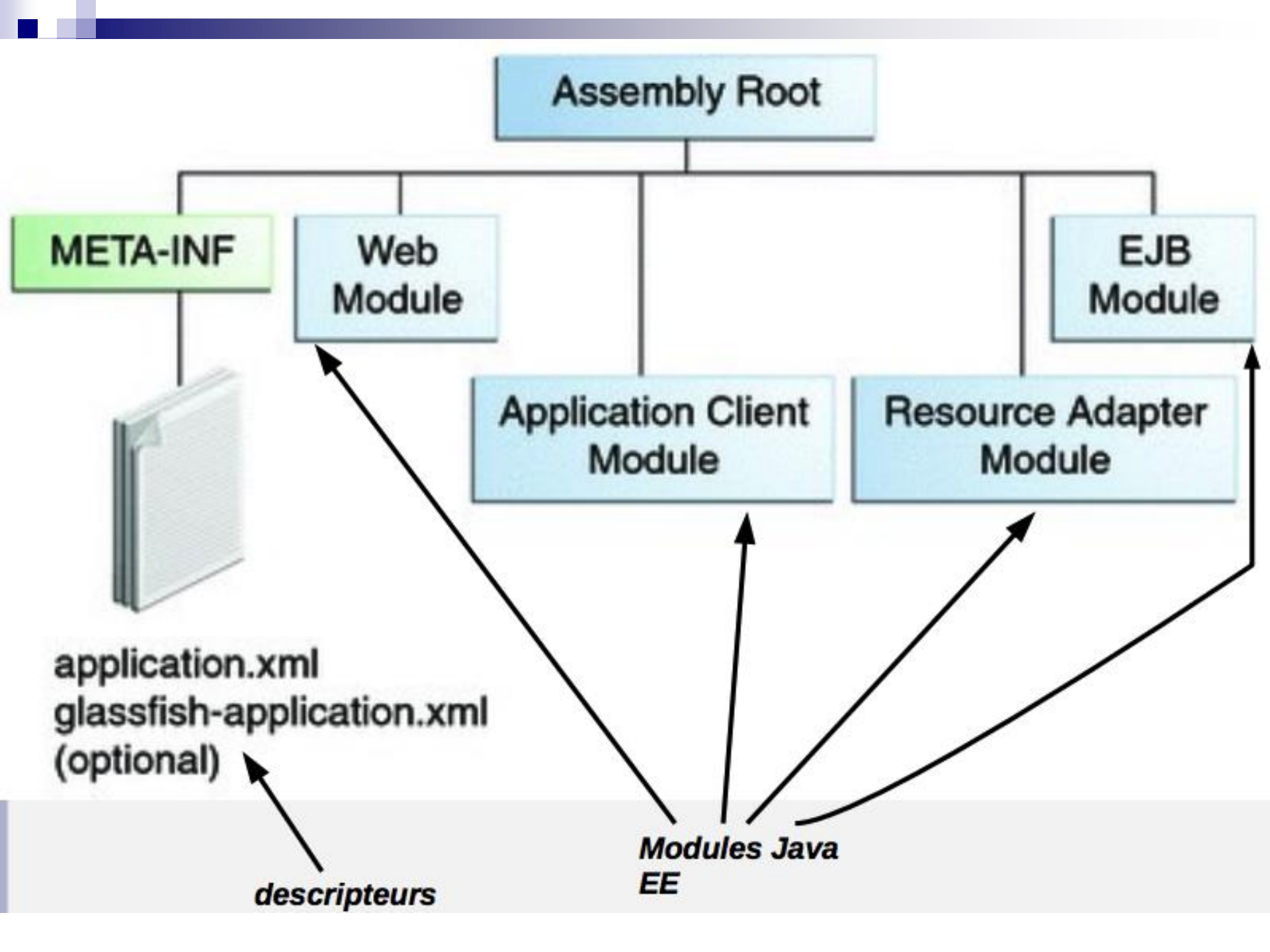


Déploiement d'une application Java EE

- Une application Java EE est donc composée d'un ensemble d'unités de programmation qui pourront ensuite être déployées sur n'importe quelle plateforme compatible avec les spécifications Java EE.
- Chaque unité contient :
 - Un ou plusieurs composants fonctionnels (ejb, pages JSP, servlet, etc.)
 - Des descripteurs de déploiement (optionnels) qui spécifient le contenu du composant.
- On parle ainsi du packaging d'une application

Les fichiers EAR

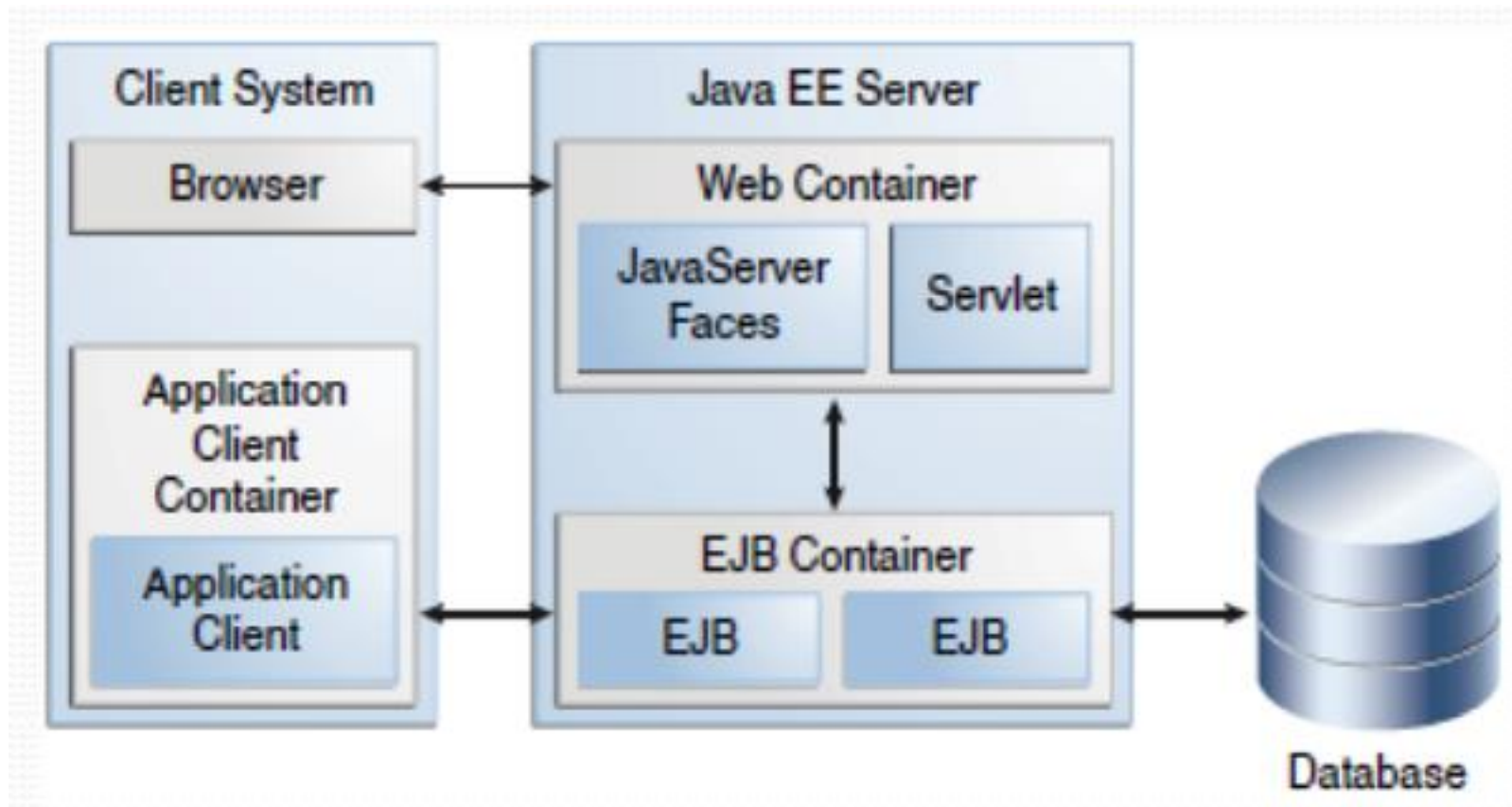
- Une application Java EE est distribuée sous forme d'un fichier Enterprise Archive (EAR)
- C'est en fait un simple fichier jar avec l'extension .ear
- Ce fichier contient :
 - des modules Java EE (.jar, .war, .rar)
 - Les descripteurs de déploiement (des fichiers xml). Ils peuvent donc être modifiés sans toucher le code de l'application.
- À l'exécution, le serveur lit les descripteurs pour utiliser les composants de manière adéquate.



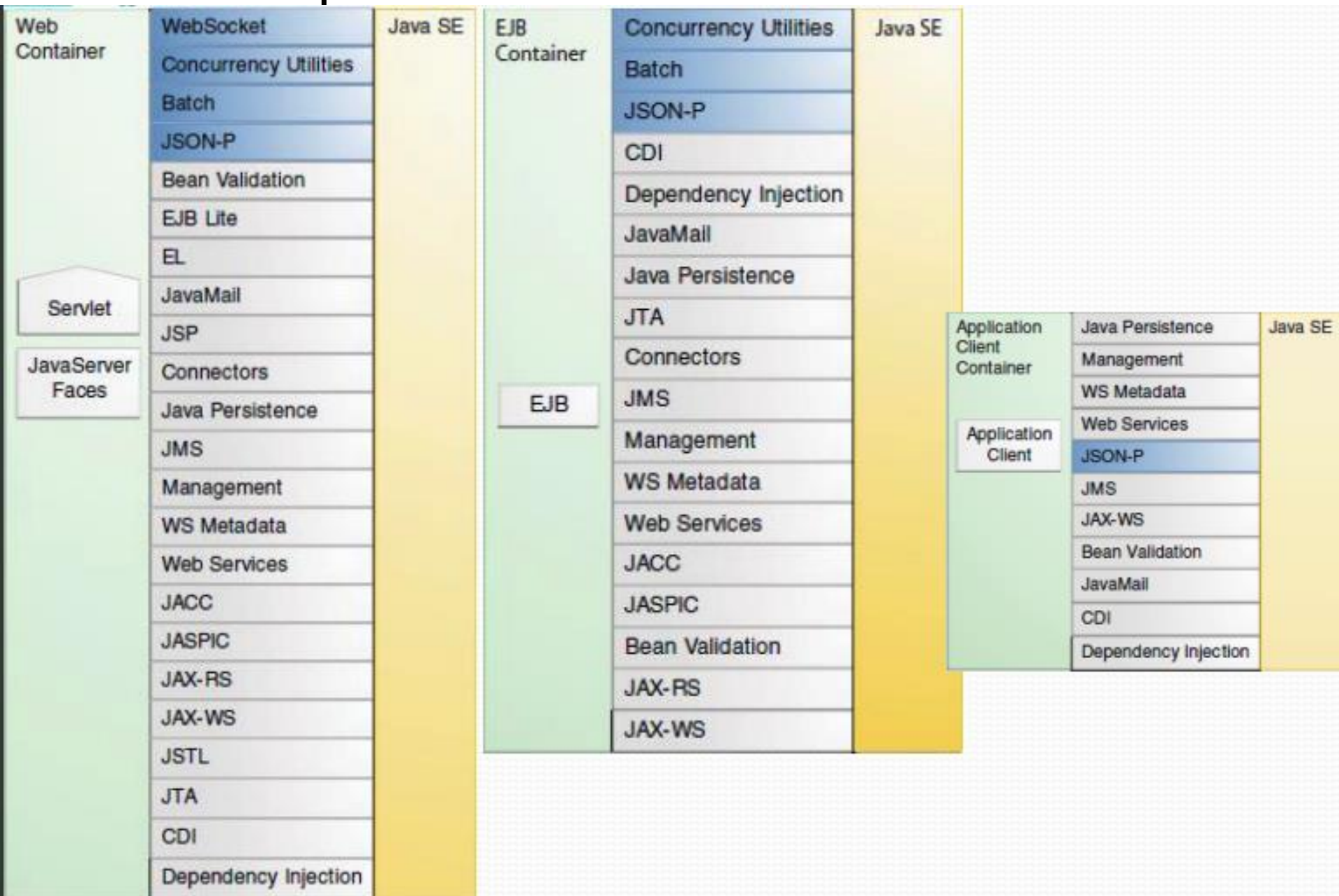
Les descripteurs

- Il existe 2 types de descripteurs :
 - Les Java EE deployment descriptor : peuvent être utilisés pour configurer les paramètres de déploiement du module
 - Les runtime deployment descriptor : utilisé pour paramétrer l'environnement hôte, c'est-à-dire n'importe quelle plate-forme compatible avec la norme Java EE.
- Par exemple, pour la plate-forme Java System Application Server Platform Edition 9, le runtime descriptor définit des informations comme la racine du répertoire web.
- Les noms de ces fichiers sont standards : `java-moduleType.xml` (dans le répertoire META-INF)

Communications possibles entre les conteneurs



Les APIs disponibles dans les différents conteneurs



Révolution dans le développement d'application d'entreprise

Conteneur léger/Agilité


1- Conteneur léger :

- Définition de conteneur : Le mot **conteneur** désigne un objet pouvant contenir d'autres objets.
- En Java un **conteneur** désigne une sous-librairie fournissant un ensemble de services spécialisés, à l'attention des développeurs. On distingue :
 - les conteneurs dits « **lourds** » qui, intégrés nativement à un serveur d'applications tel qu'Apache Tomcat ou Jboss imposent de fortes contraintes de codage, comme l'implémentation d'interface de programmation spécifique ;
 - les conteneurs dits « **légers** », moins intrusifs (par exemple Spring), qui reposent davantage sur une utilisation directe de la bibliothèque logicielle du conteneur que sur une implémentation rigoureuse de ses patterns. Il suffit de configurer le contexte d'application et créer vos Beans.

Exemple: Apache TomEE



- Apache TomEE (prononcé "Tommy") est l'édition Java Enterprise d'Apache Tomcat (Tomcat + Jakarta EE = TomEE) qui combine plusieurs projets d'entreprise Java, notamment Apache Tomcat, Apache OpenEJB, Apache OpenWebBeans, Apache OpenJPA, Apache MyFaces et autres.
- **TomEE, c'est Tomcat avec plus de fonctionnalités.**
- Apache TomEE est un excellent **serveur d'applications Java EE, léger et puissant.**
- Apache TomEE est **certifié Jakarta EE Web Profile.** (*La certification Jakarta EE Web Profile est un ensemble de critères qui définissent les exigences nécessaires de la plateforme Jakarta EE pour les applications web*)
- Si vous utilisez déjà Tomcat et que vous avez besoin pour votre projet de fonctionnalités EE, envisagez TomEE. Il **s'intègre facilement à Tomcat** et permet de **réduire le temps et le travail supplémentaires.**

- 
- Depuis Java 5, l'utilisation des *annotations* dans la norme EJB3 a permis d'alléger considérablement le développement (anciennement très lourd) d'EJBs.
 - L'utilisation d'un framework d'injection des dépendances a été rajoutée à Java EE : CDI (Context and Dependency Injection)



Java EE et le Passage à l'échelle

- On augmente le nombre de serveur (machine)
- On augmente le nombre de containers
- On répartit les composants sur les différents serveurs
- JavaEE facilite le passage à l'échelle :
 - Les composants (ie classes) ne changent pas qu'ils soient locaux ou répartis
 - La communication inter-containers est géré de façon transparente (RMI)
 - L'accès aux ressources (DB ...) reste identique



Java EE et la Sécurité

- Les contraintes de sécurités peuvent être définies lors du déploiement
- Une application JavaEE est portable sur un grand nombre d'implémentation de la sécurité
 - Tout en protégeant le développeur des problèmes d'implémentation de sécurité
- JavaEE fournit un mécanisme standard de déclaration des contrôles d'accès
- JavaEE fournit des mécanismes standard de login



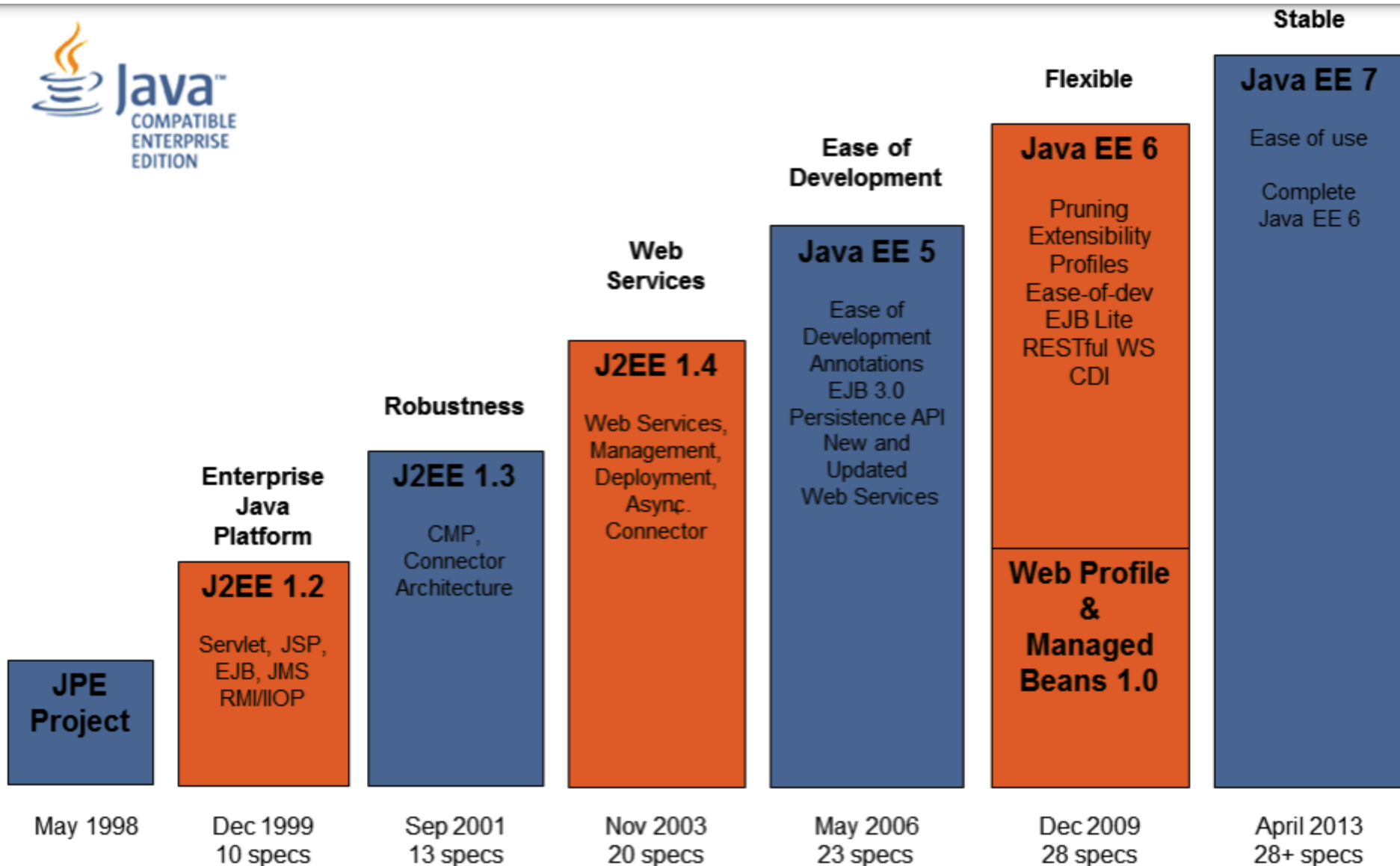
Java EE et la Persistance

- Utilisation du standard JPA
 - JPA : Java Persistence API
 - Indépendance vis-à-vis du fournisseur de BD
- Le container fournit l'accès à la BD
 - Déclaration d'une ressource BD dans le container (nom logique, fournisseur, connexion ...)
- L'application ne connaît pas :
 - Le fournisseur de la BD
 - La connexion à la BD
 - Elle connaît uniquement 'un nom de ressource' (ex: maRessource)
→ on peut changer de BD facilement.

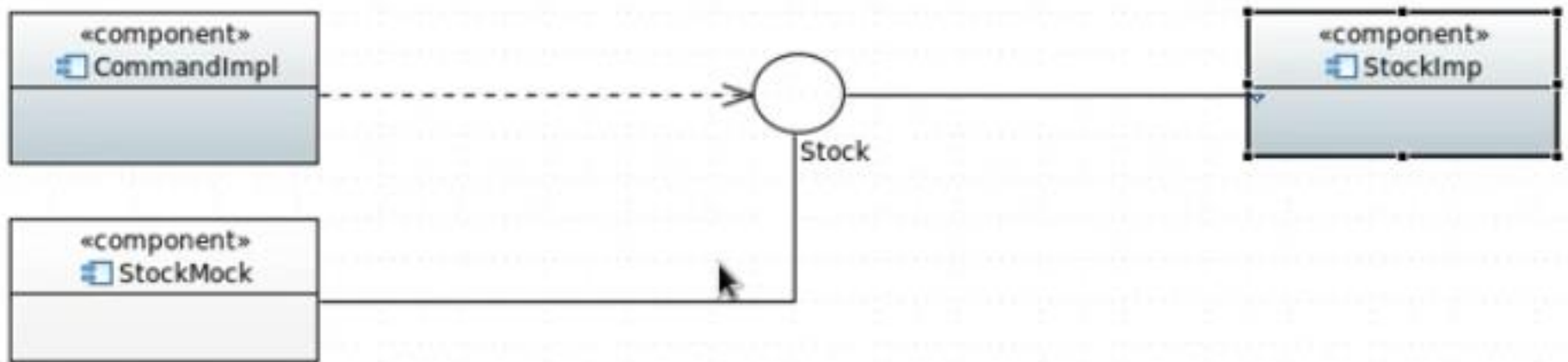


Java EE et la gestion des transactions

- Chaque container fournit un service de transaction
- Utilisation du standard JTA (Java Transaction API)
- L'application est indépendante du fournisseur de transaction



Inversion du contrôle et Injection des dépendances



- Si on avait une dépendance directe entre les composant « CommandImpl » et « StockImp », il faut attendre que le 2^{ème} composant soit écrit pour tester le premier composant.
- Le composant « CommandImpl » ne dépend pas directement du composant « StockImp », ils sont couplés par interface: l'interface « Stock » permet au premier composant d'être testé grâce à une simulation du 2^{ème} composant à travers le composant « StockMock »: on simule ce qu'on n'a pas encore → faciliter le travail entre équipes.

```
public interface Stock {  
    public boolean isArticleAvailable (int idArticle, int quantity) ;  
    public void addArticleInstances (int idArticle, int quantity) ;  
    public void removeArticleInstances (int idArticle, int quantity) throws  
StockException;  
}
```

[illegible]

Interface Command:

```
public interface Command{  
    public Stock getStock() ;  
    public void setStock(Stock stock) ;  
    public void faitqqchoz();  
}
```


Composant CommandImpl :

```
public class CommandImpl implements Command {  
    private Stock stock;  
    public CommandImpl() {... .}  
  
    public Stock getStock() {... .}  
    public void setStock(Stock stock) {... .}  
    public void faitqqchoz() {... .}  
}
```


Composant StockMock :

```
public class StockMock implements Stock {  
    public StockMock() {... }  
  
    public boolean isArticleAvailable (int idArticle, int quantity) {  
        System.out.println( "in StockMock.isArticleAvailable" ); }  
    public void addArticleInstances (int idArticle, int quantity) {  
        System.out.println( "in StockMock.addArticleInstances" ); }  
    public void removeArticleInstances (int idArticle, int quantity) throws  
        StockException {  
        System.out.println( "in StockMock.removeArticleInstances" ); }  
}
```

- StockMock est appelé bouchon: c'est un composant ultra-basique.
- Il peut être généré automatiquement par des outils tels que jMock qui est une librairie qui supporte l'approche test-driven development (TTD).


- 
- L'objectif du composant « StockMock » est de pouvoir dérouler le test et de vérifier que le composant « CommandImpl » fonctionne bien.
 - Pas la peine d'attendre que l'équipe en charge de développer le composant « StockImpl » termine sa tâche pour tester le composant « CommandImpl ».

```
public class App {  
    public static void main(String [argv]){  
        Command commandComponent = new CommandImpl();  
        Stock stockComponent = new StockImpl(); //ou bien utiliser :  
                                                // newStockMock()  
        commandComponent.setStock( stockComponent );  
        commandComponent.faitqqchoz();  
    }  
}
```

← Injection de dépendance manuelle !

Problème: L'assemblage des composants est dans le code java. Cela nécessite de l'équipe d'intégration de devoir faire des modifications dans le code java et de régénérer le jar...

→ Il faut enlever les dépendances du code java et les transférer ailleurs.

- 
- L'externalisation des injections des dépendances est réalisée automatiquement avec :
 - Le framework CDI (Context Dependencies Injection) de JEE, utilisé par JSF
 - Le framework Spring
 - Le terme « Inversion de contrôle » réfère à la même chose.
 - Ce mécanisme utilise
 - les fichiers de configuration pour charger le mécanisme d'injection des dépendances dans notre environnement
 - et les annotations: on ajoute des chaines de caractères spécifiques pour instancier les objets et les lier.

Les Annotations

- une **annotation Java** est une façon d'ajouter des méta-données à un code source Java. Elles peuvent être ajoutées aux classes, méthodes, attributs, paramètres, variables locales et paquets.
- Les annotations Java ont été introduites au JDK version 1.5.
- Elles ont été introduites en tant qu'alternative aux fichiers de configuration XML. Ceci évite d'écrire ces fichiers à la main car c'est un procédé ennuyeux et sujet à erreur.
- **Syntaxe:** Les annotations prennent la forme d'une déclaration d'interface précédée du caractère @, et facultativement suivies de méta-annotations, comme montré ci-dessous :

`@override`

`@ Named ("loginBean")`



JavaServer™ Faces

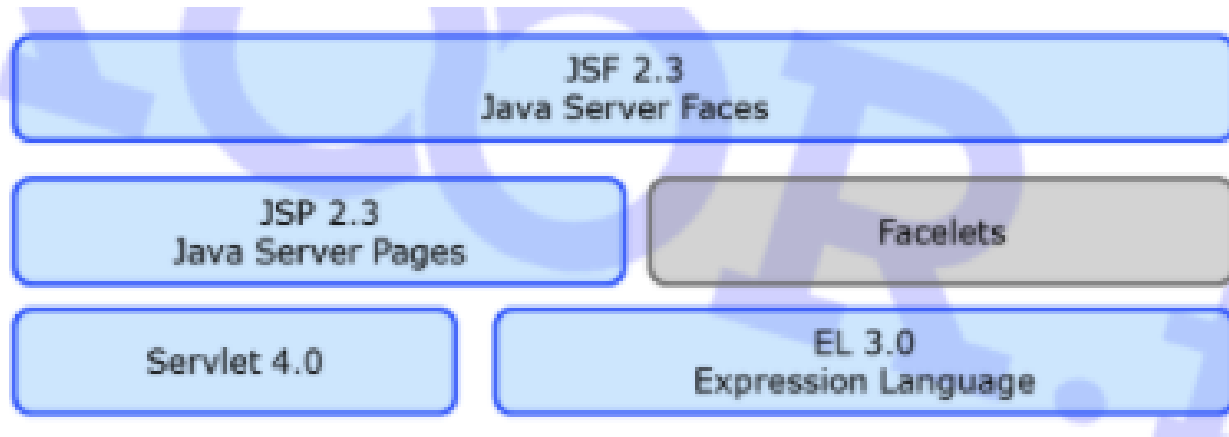
JSF



Java Server Faces

- JSF est un framework (API + cadre/méthode de travail MVC2) pour la mise en œuvre de vues
- JSF fait partie des spécifications JEE
- JSF se base et utilise l'API des servlets et des JSP
- Concurrents de JSF : Struts, Spring MVC, Wicket, etc : équivalents à JSF mais ne font pas partie des spécifications JEE (complémentaires à JEE).

Architecture JSF




- Une **Facelet** est en quelque sorte une JSP, basée sur une définition XML.
- L'API des Facelets ne fait pas partie des spécifications JEE: c'est une librairie tierce.
- **Expression Language (EL)** est un langage de script permettant l'accès à des composants Java (les JavaBeans) à travers des JSP ou JSF.

Principes JSF

- Pattern MVC
- Internationalisation des applications web
- Gestion des événements déclenchés **coté serveur** pour réagir aux interactions utilisateur (Gestion des événement coté client avec du javascript ou l'Angular)

Historique des versions

- La branche 1.x de l'API JSF était fortement orientée XML pour tout ce qui avait attrait à la configuration de l'application.
- Bien que supportant l'approche XML, la branche 2.x a proposé des mécanismes de configuration de l'application orientés **annotations : moins de code**.
- Annotations CDI (Contexts and Dependency Injection) est un framework d'injection de dépendance qui a été intégrée au Java EE à partir de sa version 6.0.

- 
- MVC2 est dérivé de l'MVC
 - MVC2 met en œuvre un contrôleur unique

GlassFish

- Glassfish est un serveur d'applications compatible Java EE, certifié JEE : il supporte donc l'ensemble des API proposées par la plate-forme Java EE: EJB3, JSF2, CDI, JPA, JAX-WS 2.x + ...
- GlassFish est le RI (Reference Implementation) développé initialement par Oracle et géré aujourd'hui par Eclipse
- Tomcat est un serveur web qui ne tient pas compte de toutes les spécifications JEE, mais qu'on peut enrichir avec les jars de la spécification JSF :
<https://github.com/javaxserverfaces> ou
<https://myfaces.apache.org/>



Autres serveurs compatibles avec JEE

- Websphere : IBM
 - Wildfly(JBOSS) : Redhat
 - WebLogic : Oracle
 - Geronimo: Apache
-
- Attention aux versions des serveurs : chaque version fais référence à des versions différentes de spécifications JEE.



Télécharger GlassFish

- Aller sur eclipse.org, cliquer sur « Projects » et taper « GlassFish »
- Version actuelle: 6.2.1. compatible avec JEE 9



Eclipse GlassFish

OSGI

Runtime

Eclipse GlassFish is a complete application **server** that implements the Jakarta EE specification.

[Read more...](#)

Latest release: 6.2.1

[Download](#)



Eclipse GlassFish Tools

Tools

Eclipse GlassFish Tools enables **publishing of Eclipse IDE projects to GlassFish server**, as well as controlling GlassFish server from within Eclipse IDE. It extends Eclipse web Tools Platform.

[Read more...](#)

Latest release: 1.0.1

[Download](#)

Plugin Eclipse

Eclipse GlassFish Tools

[Overview](#)[Downloads](#)[Who's Involved](#)[Developer Resources](#)[Governance](#)[Contact Us](#)

Prerequisite: *

- Install Eclipse IDE for Enterprise Java Developers
- Install Eclipse
Sapphire: <https://download.eclipse.org/sapphire/9.1.1/repository/>


[Download Eclipse GlassFish Tools](#)

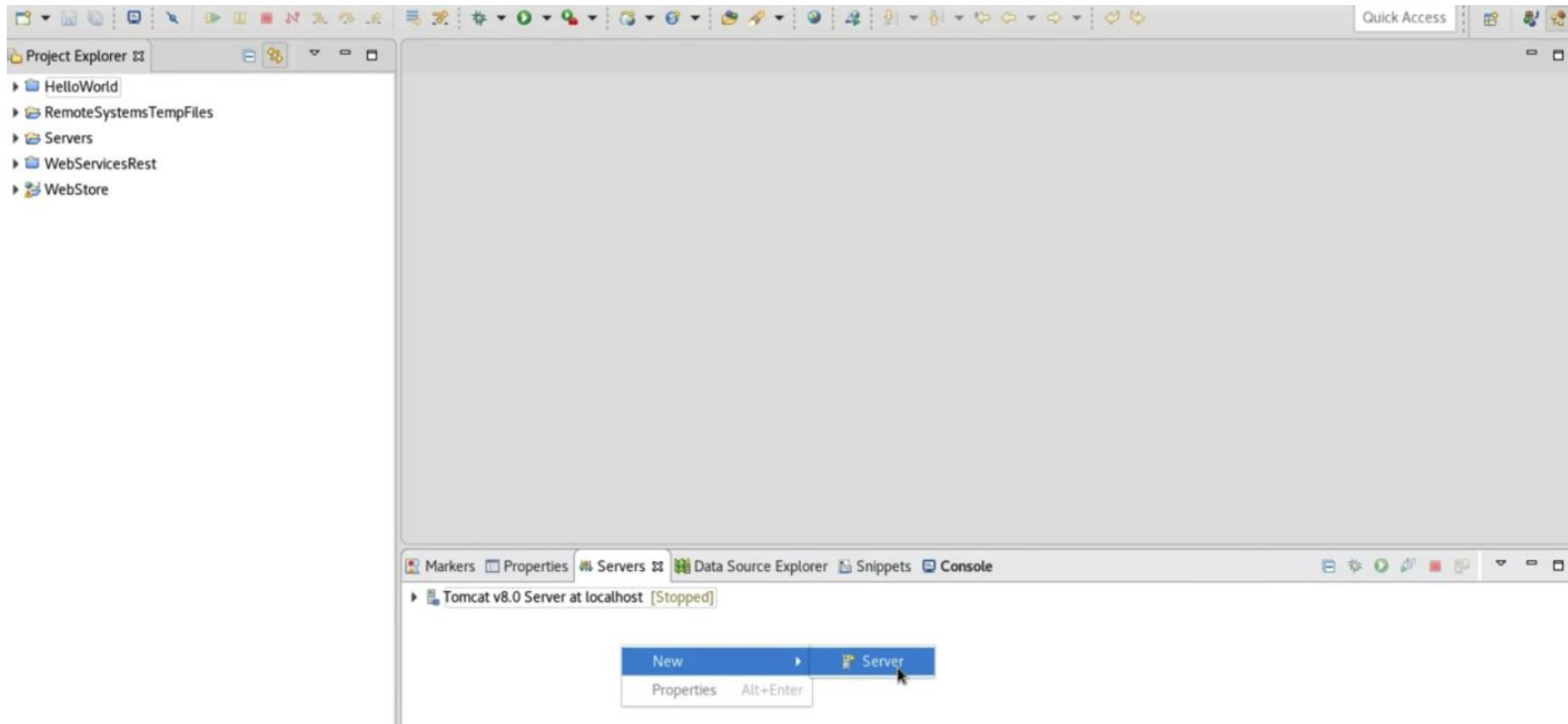
Update Sites:

Glassfish-Tools 1.0.1 Repository <https://download.eclipse.org/glassfish-tools/1.0.1/repository/>

Downloads:

1.0.1 Offline Repository ZIP file

- 
- Aller sur votre Eclipse déjà installé, cliquer sur Help->Install new software->Add et vous collez l'adresse déjà copiée
 - Redémarrer Eclipse
 - Aller sur Window->Preferences->Server->Runtime Environment->Add->GlassFish



File Edit Navigate Search Project Run Window Help




Quick Access

Project Explorer

- ▶ HelloWorld
- ▶ RemoteSystemsTempFiles
- ▶ Servers
- ▶ WebServicesRest
- ▶ WebStore

Markers Properties Servers Data Source Explorer Snippets Console

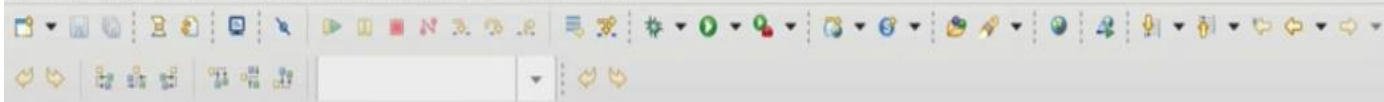
- ▶ GlassFish 5 [domaine1] [Stopped]
- ▶ Tomcat v8.0 Server at localhost [Stopped]

- 
- Démarrer et tester le serveur sur :
<https://localhost:8080>
 - Déployer des applications sur GlassFish :
 - click droit sur l'icône du serveur->Add and Remove
 - Ou click droit sur le nom de l'application->Run on server

Première page JSF

- Sur Eclipse: Project->New->Dynamic Web Project->Java EE
- Dans la sous-partie « Configuration », cocher « Java Server Faces »

File Edit View Source Navigate Search Project Run Window Help



Quick Access

Project Explorer

- WebStoreJSF
 - Deployment Descriptor: WebStoreJSF
 - JAX-WS Web Services
 - Java Resources
 - JavaScript Resources
 - build
- WebContent
 - META-INF
 - WEB-INF
 - lib
 - faces-config.xml
 - glassfish-web.xml
 - web.xml

faces-config.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <faces-config
3     xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
6     version="2.2">
7
8 </faces-config>
9
```

Introduction Overview Navigation Rule ManagedBean Component Others Source

Markers Properties Servers Data Source Explorer Snippets Problems Console Progress

- GlassFish 5 [domaine1] [Started, Synchronized]
- Tomcat v8.0 Server at localhost [Stopped]

Project Explorer

- WebStoreJSF
 - Deployment Descriptor: WebStoreJSF
 - JAX-WS Web Services
 - Java Resources
 - src
 - Libraries
 - GlassFish System Libraries
 - bean-validator.jar - /home/KooR.fr/Downloads
 - cdi-api.jar - /home/KooR.fr/Downloads/glass
 - grizzly-npn-bootstrap.jar - /home/KooR.fr/D
 - javax.annotation-api.jar - /home/KooR.fr/Dov
 - jaxb-api.jar - /home/KooR.fr/Downloads/glas
 - webservices-api-osgi.jar - /home/KooR.fr/Di
 - glassfish-api.jar - /home/KooR.fr/Downloads
 - ha-api.jar - /home/KooR.fr/Downloads/glass
 - javax.batch-api.jar - /home/KooR.fr/Downloa
 - javax.ejb-api.jar - /home/KooR.fr/Downloads
 - javax.el.jar - /home/KooR.fr/Downloads/glas
 - javax.enterprise.concurrent-api.jar - /home/f

faces-config.xml

web.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp
3 <display-name>WebStoreJSF</display-name>
4 <servlet>
5   <servlet-name>Faces Servlet</servlet-name>
6   <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
7   <load-on-startup>1</load-on-startup>
8 </servlet>
9 <servlet-mapping>
10  <servlet-name>Faces Servlet</servlet-name>
11  <url-pattern>/faces/*</url-pattern>
12 </servlet-mapping>
13 </web-app>
```

Contrôleur principal MVC2

Format de l'url associé aux pages jsf. faces est un dossier virtuel

<http://localhost:8080/WebStoreJSF/faces/.....>

Descripteur de déploiement

Design Source

Markers Properties Servers Data Source Explorer Snippets Problems Console Progress

```
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
  <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

faces-config.xml

glassfish-web.xml

web.xml

```
<context-param>
  <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
  <param-value>.xhtml</param-value>
</context-param>
```

- le tag `<context-param>` permet de définir un paramètre, nommé `javax.faces.DEFAULT_SUFFIX`, de configuration de l'application (en Java EE, context est un synonyme d'application).
- Ce paramètre est utilisé par l'API JSF pour connaître l'extension (le suffixe) des vues utilisées par le pattern MVC de JSF. Dans notre cas, nous avons retenu `.xhtml` : c'est la valeur par défaut. Mais vous pouvez changer cette valeur si vous le souhaitez : dans ce cas, il faudra correctement nommer vos vues.

Fichier web.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5 http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" version="4.0">
6
7 <display-name>WebStoreJSF</display-name>
8
9 <context-param>
10     <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
11     <param-value>.xhtml</param-value>
12 </context-param>
13
14 <servlet>
15     <servlet-name>Faces Servlet</servlet-name>
16     <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
17 <load-on-startup>1</load-on-startup>
18 </servlet>
19
20 <servlet-mapping>
21     <servlet-name>Faces Servlet</servlet-name>
22     <url-pattern>/faces/*</url-pattern>
23 </servlet-mapping>
24
25 </web-app>
```



```
<welcome-file-list>
```

```
    <welcome-file>/faces/login.xhtml</welcome-file>
```

```
</welcome-file-list>
```

-Taper : <http://localhost:8080/WebStoreJSF>

pour aller directement sur login.xhtml

Configuration de faces-config.xml

- On y touche pas pour le moment


```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <faces-config version="2.3" »
3     xmlns=http://xmlns.jcp.org/xml/ns/javaee
4     xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
5     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
6                           http://xmlns.jcp.org/xml/ns/javaee/web-
7 facesconfig 2 3.xsd">
8 </faces-config>
```

Configuration de CDI (Contexts and Dependency Injection)

- CDI est le framework d'injection de dépendances proposé par la plate-forme Java EE depuis sa version 6.0
- Le couplage JSF/CDI doit être réalisé manuellement.
- **Etape 1:** Ajouter la classe suivante: ApplicationConfiguration.java

```
1  Package fst.irisi.webstore;  
2  
3  import javax.enterprise.context.ApplicationScoped;  
4  import javax.faces.annotation.FacesConfig;  
5  
6  @ApplicationScoped  
7  @FacesConfig( version = FacesConfig.Version.JSF_2_3 )    // Activation de CDI  
8  public class ApplicationConfiguration {  
9  
10 }
```

- Au démarrage de JSF, il va chercher ces annotations et activer le lien avec CDI.




Cette classe peut être placée dans n'importe quel package de votre projet et elle peut avoir le nom de votre choix. C'est la présence des annotations qui permet le lancement de CDI. Si vous oubliez de fournir cette classe, vous pourrez avoir un message d'erreur proche de celui-ci. Il faudra juste comprendre qu'il n'aura pas réalisé certaines injections de dépendances (ligne en gras).

```
2021-06-20T15:37:13.430+0200|Avertissement: StandardWrapperValve[Faces Servlet]:  
Servlet.service() for servlet Faces Servlet threw exception  
javax.el.PropertyNotFoundException: Target Unreachable,  
identifiant 'loginBean' resolved to null  
    at com.sun.el.parser.AstValue.getTarget(AstValue.java:173) at  
com.sun.el.parser.AstValue.getType(AstValue.java:85) ...
```

- Etape 2: Il faut aussi fournir un fichier de configuration de CDI qui doit être placé dans le dossier WEB-INF de votre application Web. Ce fichier doit se nommer **WEB-INF/beans.xml**.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5 http://xmlns.jcp.org/xml/ns/javaee/beans_2_0.xsd"
6       bean-discovery-mode="annotated" version="2.0">
7
8 </beans>
```

- l'attribut `bean-discovery-mode="annotated"` est important, c'est lui qui indique que CDI doit rechercher des annotations pour trouver les injections de dépendances à réaliser.
- Il va chercher des annotations particulières proposées par le CDI.



Mise en oeuvre d'une page Web basée sur le pattern MVC2

- 1- La vue : Login.xhtml dans le dossier WebContent
- Il s'agit d'un fichier XML/une facelet
- Cette vue sera exécutée côté serveur et sera transformée en page HTML classique exécutée par le navigateur client

```
1 <!DOCTYPE html>
2 <html xmlns:f="http://xmlns.jcp.org/jsf/core"
3     xmlns:h="http://xmlns.jcp.org/jsf/html">
4     <f:view>
5         <head>
6             <title>Login screen</title>
7             <link rel="stylesheet" type="text/css" href="styles.css" />
8         </head>
9         <body>
10             <h1>Login screen</h1>
11             <h:form>
12                 Login:
13                 <h:inputText id="login" value="#{loginBean.login}" />
14                 <br/>
15                 Password:
16                 <h:inputText id="password" value="#{loginBean.password}"
17 />
18                 <br/>
19                 <h:commandButton action="#{loginBean.returnAction}"
20 value="Connect" />
21             </h:form>
22         </body>
23     </f:view>
24 </html>
```

Taglib JSF

Obligatoire! Sinon, erreur

Formulaire avec méthode POST

Liaison avec les données du Bean loginBean

- 
- 2- Le modèle : La classe LoginBean.java dans le dossier src

```
1 package fr.koor.webstore.ihm;  
2 import java.io.Serializable;  
3 import javax.enterprise.context.SessionScoped;  
4 import javax.inject.Named;  
5
```

**Annotation : objet sera injecté par le CDI.
L'instance créée sera nommée
loginBean. C'est le nom utilisé dans
Login.xhtml**

```
6 @Named /* ("loginBean") */
```

```
7 @SessionScoped
```

Une instance sera créée par session

```
8 public class LoginBean implements Serializable {  
9
```

```
10     private static final long serialVersionUID = -5433850275008415405L;  
11     private String login = "James";  
12     private String password = "007";  
13
```

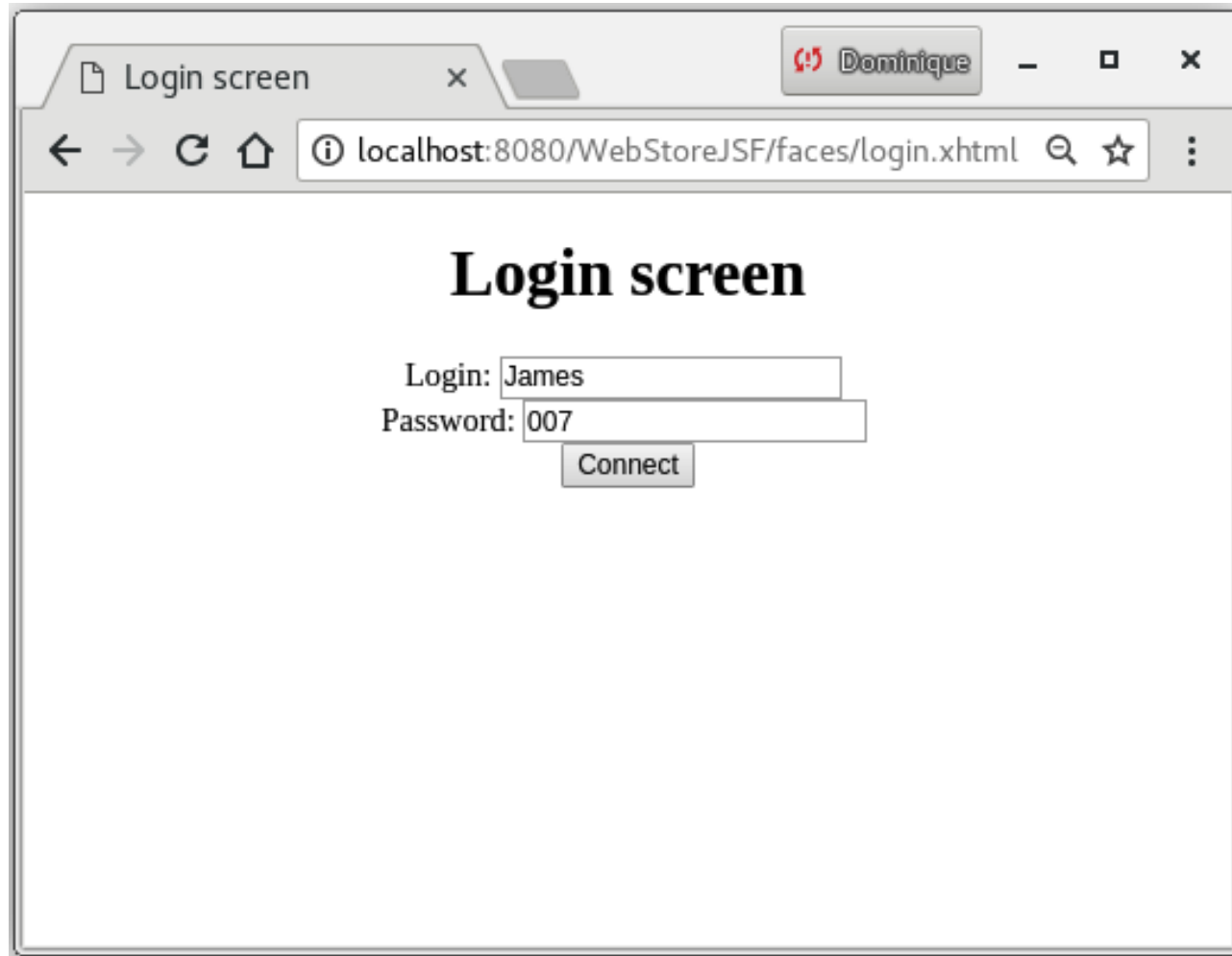
```
14     public String getLogin() {  
15         System.out.println( "in getLogin" ); return login;  
16     }  
17     public void setLogin(String login) {  
18         System.out.println( "in setLogin with " + login );  
19         this.login = login;  
20     }  
21     public String getPassword() {  
22         System.out.println( "in getPassword" );  
23         return password;  
24     }  
25     public void setPassword(String password) {  
26         System.out.println( "in setPassword with " + password );  
27         this.password = password;  
28     }  
29     public String returnAction() {  
30         System.out.println( "in returnAction" );  
31         return password.equals( "007" ) ? "success" : "failure";  
32     }  
33 }
```

**Les getters permettront d'accéder aux
valeurs des propriétés dans le
Login.xhtml à travers:**

`"#{loginBean.login}"` et
`"#{loginBean.password}"`

**La méthode d'action. Signature
obligatoire**

Déploiement dans le serveur GlassFish et exécution



Code source de la page

```
view-source:localhost:8080/WebStoreJSF/faces/login.xhtml

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Login screen</title>
5     <link rel="stylesheet" type="text/css" href="styles.css" />
6   </head>
7   <body>
8     <h1 align="center">Login screen</h1>
9     <form id="j_idt3" name="j_idt3" method="post" action="/WebStoreJSF/faces/login.xhtml;jsessionid=210e0f210fbc1f4cb7b039a59d70" enctype="application/x-www-form-urlencoded">
10    <input type="hidden" name="j_idt3" value="j_idt3" />
11
12
13    Login:
14    <input id="j_idt3:login" type="text" name="j_idt3:login" value="James" />
15    <br />
16
17    Password:
18    <input id="j_idt3:password" type="text" name="j_idt3:password" value="007" />
19    <br /><input type="submit" name="j_idt3:j_idt7" value="Connect" /><input type="hidden" name="javax.faces.ViewState" id="j_idl:javax.faces.ViewState:0"
20    value="8391137355632960865:3838074942344348638" autocomplete="off" />
21  </form>
22  </body>
23 </html>
```

- Aucune trace des tags JSF. Tout a été transformé en tag HTML et renvoyé au navigateur.

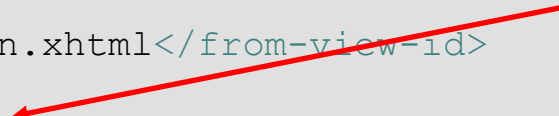
Enchainement de pages JSF

■ 1-Création de la page ViewArticle.xhtml


```
1 <!DOCTYPE html>
2 <html xmlns:f="http://xmlns.jcp.org/jsf/core"
3     xmlns:h="http://xmlns.jcp.org/jsf/html">
4     <f:view>
5         <head>
6             <title>View Article</title>
7             <link rel="stylesheet" type="text/css"
8 href="styles.css" />
9         </head>
10        <body>
11            <h1>View Article</h1> <!-- TODO: ? finir -->
12        </body>
13    </f:view>
14 </html>
```

■ 2-Modification de faces-config.xml pour définir des règles de navigation

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <faces-config version="2.3" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5 http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_3.xsd">
6
7     <navigation-rule>
8         <from-view-id>/login.xhtml</from-view-id>
9         <navigation-case>
10             <from-outcome>success</from-outcome>
11             <to-view-id>/viewArticle.xhtml</to-view-id>
12         </navigation-case>
13     </navigation-rule>
14
15 </faces-config>
```



**Voir
LoginBean.returnAction()**

- 
- On peut ne pas définir des règles de navigation explicites, mais définir des pages : `success.xhtml` et `failure.xhtml`. La redirection sera implicite et automatique.

Validation du formulaire JSF

C'est une validation côté serveur

■ 1- Par tags:

```
1 <h:form>
2     Login:
3     <h:inputText id="login" value="#{loginBean.login}"
4     required="true" requiredMessage="La saisie du login est obligatoire
5     !" />
6     <h:message for="login" styleClass="errorBlock" /> <br/>
7     Password:
8     <h:inputSecret id="password" value="#{loginBean.password}"
9     required="true" requiredMessage="La saisie du mot de passe est
10    obligatoire !" />
11    <h:message for="password" styleClass="errorBlock" />
12    <br/><br/>
13
14    <h:commandButton action="#{loginBean.returnAction}"
15    value="Connect" />
16 </h:form>
```



```
1 Password:
2 <h:inputSecret id="password" value="#{loginBean.password}" required="true"
3     requiredMessage="La saisie du mot de passe est obligatoire !"
4     validatorMessage="Votre mot de passe doit contenir entre 3 à 12
5 caractères">
6 <f:validateLength minimum="3" maximum="12" />
7 </h:inputSecret>
8 <h:message for="password" styleClass="errorBlock" />
```

```
1 Login:
2 <h:inputText id="login" value="#{loginBean.login}"
3     required="true" requiredMessage="La saisie du login est
4 obligatoire !"
5     validatorMessage="Le login doit correspondre à un
6 email"> <f:validateRegex pattern="^[\\w.+\\-]+@[\\w.+\\-]+\\. [a-z]{2,}$" />
7 </h:inputText>
8 <h:message for="login" styleClass="errorBlock" />
9 <br/>
```

2- Mise en oeuvre de validateurs personnalisés

```
1 public void validateAField(FacesContext context, UIComponent component, Object value)
2 throws ValidatorException {
3     String inputValue = (String) value; if ( ! inputValue.contains( "#" ) ) {
4         FacesMessage msg = new FacesMessage( "Mauvais format : doit contenir un
5 #" );
6         msg.setSeverity( FacesMessage.SEVERITY_ERROR );
7         throw new ValidatorException(msg);
8     }
9 }
```

```
1 <h:inputText id="aField" requiredMessage="La saisie du champ est
2 obligatoire !"
3         validator="#{backingBean.validateAField}" />
4 <h:message for="aField" styleClass="errorBlock" />
5 <br/>
```



■ 3- Annotations:

- JEE offre un framework de validation utilisé par les Bean
- On utilise des annotations de validation placées sur les attributs des beans
- Le package principal associé à l'API de validation des données est `javax.validation`.
- Notez aussi que ces annotations de validations peuvent être utilisées avec d'autres frameworks que JSF.


Définition d'une annotation de validation

■ Etape 1: Définition de l'annotation

```
1 import java.lang.annotation.Documented;
2 import java.lang.annotation.ElementType;
3 import java.lang.annotation.Retention;
4 import java.lang.annotation.RetentionPolicy;
5 import java.lang.annotation.Target;
6 import javax.validation.Constraint;
7 import javax.validation.Payload;
8
9 @Constraint(validatedBy = EmailValidator.class)
10 @Documented
11 @Retention(RetentionPolicy.RUNTIME)
12 @Target({ElementType.FIELD, ElementType.METHOD, ElementType.ANNOTATION_TYPE})
13
14 public @interface Email {
15     String message() default "Is not a valid email";
16     Class<?>[] groups() default {};
17     Class<? extends Payload>[] payload() default {};
18 }
```

Le lien avec la classe de validation est réalisé avec l'annotation `@Constraint` : la classe de validation se nomme `EmailValidator`.

Une annotation se définit via le mot clé `@interface`

- 
- **Etape 2: Définition d'un validateur associé à l'annotation de validation**
 - Celle-ci doit implémenter l'interface `ConstraintValidator`. Cette interface définit deux méthodes : `initialize` (c'est une méthode par défaut depuis Java EE, il n'est donc plus obligatoire de la redéfinir) et `isValid`. C'est cette dernière méthode qui devra effectuer la validation.

```
1 import java.util.regex.Pattern;
2 import javax.validation.ConstraintValidator;
3 import javax.validation.ConstraintValidatorContext;
4
5 public class EmailValidator implements ConstraintValidator<Email, String> {
6     private static Pattern emailPattern = Pattern.compile( "[\\w.-]+@[\\w.-"
7 ]+\\. [a-z]{2,}$" );
8
9     @Override
10    public void initialize( Email constraintAnnotation ) {
11        // Rien à faire pour cet exemple.
12    }
13
14    @Override
15    public boolean isValid( String email, ConstraintValidatorCont
16        return emailPattern.matcher( email ).find();
17    }
18 }
```

Cette classe de validateur utilise une expression régulière pour valider le format de l'email passé en paramètre.

@Override s'utilise pour annoter une méthode qui est une réécriture d'une méthode héritée. Le compilateur lève une erreur si aucune méthode héritée ne correspond. C'est un marqueur utilisé par le compilateur pour vérifier la réécriture de méthodes héritées. Son utilisation n'est pas obligatoire mais recommandée.

- 
- **Utilisation d'annotations de validation:**
 - **Login.xhtml : on retire les tags de validation précédemment utilisés**

```

<!DOCTYPE html>
<html xmlns:f="http://xmlns.jcp.org/jsf/core"
xmlns:h="http://xmlns.jcp.org/jsf/html">
    <f:view>
        <head>
            <title>Login screen</title>
            <link rel="stylesheet" type="text/css" href="../styles.css" />
        </head>
        <body>
            <h1>Login screen</h1>
            <h:form id="form">
                Login:
                <h:inputText id="login" value="#{loginBean.login}" required="true"
requiredMessage="La saisie du login est obligatoire !" /> &#160;
                <h:message for="login" styleClass="errorBlock" />
                <br/>
                Password:
                <h:inputSecret id="password" value="#{loginBean.password}"
required="true" requiredMessage="La saisie du mot de passe est obligatoire !"
validatorMessage="Votre mot de passe doit contenir entre 3 à 12 caractères" />
&#160;
                <h:message for="password" styleClass="errorBlock" />
                <br/><br/>
                <h:commandButton action="#{loginBean.returnAction}" value="Connect" />
            </h:form>
        </body>
    </f:view>
</html>

```

On pourrait aussi retirer les attributs required car les annotations que nous allons utiliser imposent de tailles pour les deux champs à valider.



Définir les règles de validation des propriétés de
notre bean :

```
1 import java.io.Serializable;
2 import javax.enterprise.context.SessionScoped;
3 import javax.inject.Named;
4 import javax.validation.constraints.Size;
5 import webstore.validator.Email;
6
7 @Named("loginBean")
8 @SessionScoped
9 public class LoginBean implements Serializable {
10     private static final long serialVersionUID = -5433850275008415405L;
11     @Email @Size(min=1)
12     private String login = "james@mi6.uk";
13     @Size(min=3, max=8)
14     private String password = "007";
15
16     public String getLogin() {
17         return login;
18     }
19     public void setLogin(String login) {
20         this.login = login;
21     }
22     public String getPassword() {
23         return password;
24     }
25     public void setPassword(String password) {
26         this.password = password;
27     }
28     public String returnAction() {
29         return password.equals( "007" ) ? "success" : "failure";
30     }
31 }
```


Annotation size pour
contrôler la taille du
champs

L'annotation @email
demande à vérifier si une
chaine de caractères
correspond bien à un email

Autres annotations du package `javax.validation.constraints` :

- **@AssertFalse** : s'applique à un attribut de type booléen et vérifie que sa valeur soit false.
- **@AssertTrue** : s'applique à un attribut de type booléen et vérifie que sa valeur soit true.
- **@DecimalMax("10.5")** : permet de vérifier qu'une valeur flottante est bien inférieure ou égale à la valeur spécifiée dans l'annotation de validation. Attention : la valeur spécifiée doit l'être sous forme d'une chaîne de caractères.
- **@DecimalMin("10.5")** : permet de vérifier qu'une valeur flottante est bien supérieure ou égale à la valeur spécifiée dans l'annotation de validation. Attention : la valeur spécifiée doit l'être sous forme d'une chaîne de caractères.
- **@Digits(integer=3, fraction=2)** : permet de vérifier le nombre de chiffres utilisé par une valeur flottante. Vous devez spécifier deux paramètres entiers dans l'annotation : integer qui indique le nombre de chiffres utilisé pour la partie entière et fraction qui indique le nombre de chiffres utilisé par la partie décimale.

- **@Future** : permet de vérifier qu'une date est postérieure à la date actuelle.
- **@FutureOrPresent** : permet de vérifier qu'une date est postérieure ou égale à la date actuelle.
- **@Max(100)** : permet de vérifier qu'un entier est inférieur ou égal à la valeur spécifiée dans l'annotation.
- **@Min(1)** : permet de vérifier qu'un entier est supérieur ou égal à la valeur spécifiée dans l'annotation.
- **@Negative** : permet de vérifier qu'une valeur numérique (entière ou flottante) soit strictement inférieure à zéro.
- **@NegativeOrZero** : permet de vérifier qu'une valeur numérique (entière ou flottante) soit inférieure ou égale à zéro.
- **@NotBlank** : permet de vérifier qu'une chaîne de caractères ne puisse pas être nulle et contienne au moins un caractère qui ne soit pas un séparateur (blanc, tabulation, retour à la ligne, ...).
- **@NotEmpty** : permet de vérifier qu'une chaîne de caractères ne puisse pas être nulle ou vide.
- **@NotNull** : permet de vérifier qu'une valeur ne puisse pas être nulle.
- **@Null** : permet de vérifier qu'une valeur soit nulle.
- **@Past** : permet de vérifier qu'une date est antérieure à la date actuelle.

- 
- **@PastOrPresent** : permet de vérifier qu'une date est antérieure ou égale à la date actuelle.
 - **@Pattern(regExp)** : permet de vérifier qu'une chaîne de caractères correspond à l'expression régulière passée en paramètre de l'annotation.
 - **@Positive** : permet de vérifier qu'une valeur numérique (entière ou flottante) soit strictement supérieure à zéro.
 - **@PositiveOrZero** : permet de vérifier qu'une valeur numérique (entière ou flottante) soit supérieure ou égale à zéro.
 - **@Size(min=3, max=8)** : permet de vérifier qu'une chaîne de caractères ait bien sa taille comprise entre une valeur minimale et une valeur maximale.