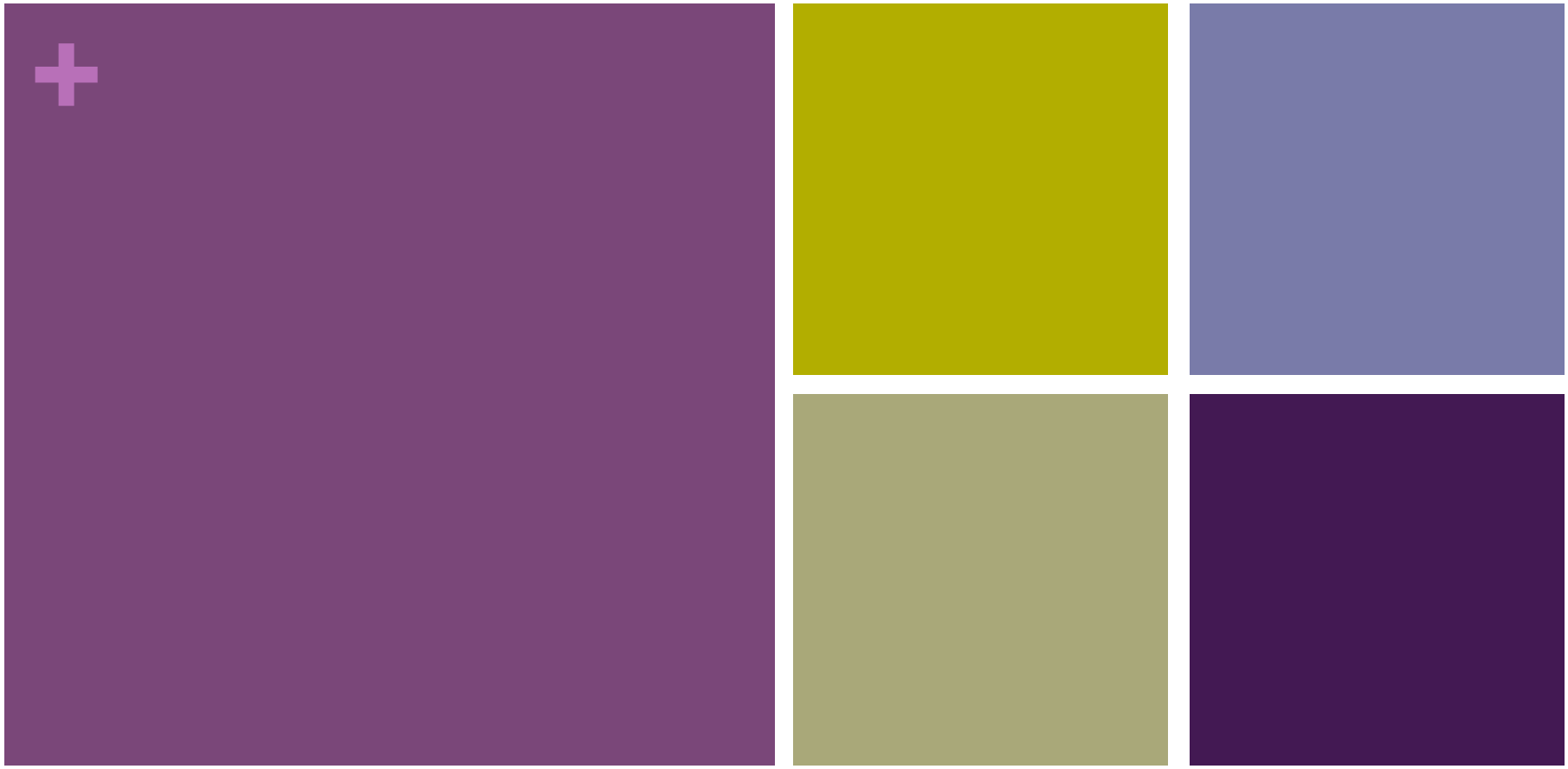


Master 2 Informatique

Java EE

Présentation

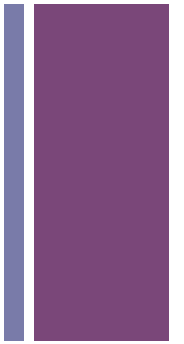
- Thierry Poutrain
- thierry.poutrain@kearis.fr
- 48h de formation (13 sessions)
- Théorie / Pratique
- Examen
- Projets en groupe

Déroulement

- Java Persistence API (ORM / Bean Validation)
- Enterprise JavaBean
- Web Service
- JSP et JSF
- Messaging
- Autour de Java EE

Le langage Java

- Orienté objet (1995)
- Sun puis Oracle
- API et bibliothèque de base
- Multi-plateforme via JVM
- Compilation
- Fortement typé
- GC
- Encapsulation / Polymorphisme



Le langage Java

Le langage Java

- Java SE 6 : 2006
- Java SE 7 : 2011
- Java SE 8 : 2014
- Java SE 9 : 2016
- Et Java EE… c'est quoi ?

Java EE

- APIs Java : Collection / Hashmap / IO / …
  - On les utilise et on invente rien
- Java EE : idem
- Avec des fonctionnalités d'entreprise :
  - persistance / sécurité / webservice …
- Spécifications dont l'implémentation permet de créer des applications d'entreprise
- Spécifications selon processus standardisé
  - établies selon le *Java Community Process*
  - regroupées au travers des *Java Specification Requests*

Java EE

- Les *Java Specification Requests* :
  - Pourquoi :
    - Système normalisé
    - Rôles bien précis
    - Optionnelle / Obligatoire
  - Sur chaque techno de Java
  - https://www.jcp.org/en/jsr/all
  - Mis en avant via le Java Community Process
    - https://www.jcp.org/en/home/index
  - Exemples :
    - NIO, *JDBC*, Java Compiler, OSGi (Java SE)
    - JMS, EJB, JPA, JSF (Java EE)
    - Java USB, Bluetooth, MMAPI (Java ME)

Java EE

- Les *Java Specification Requests* :
  - Historique :

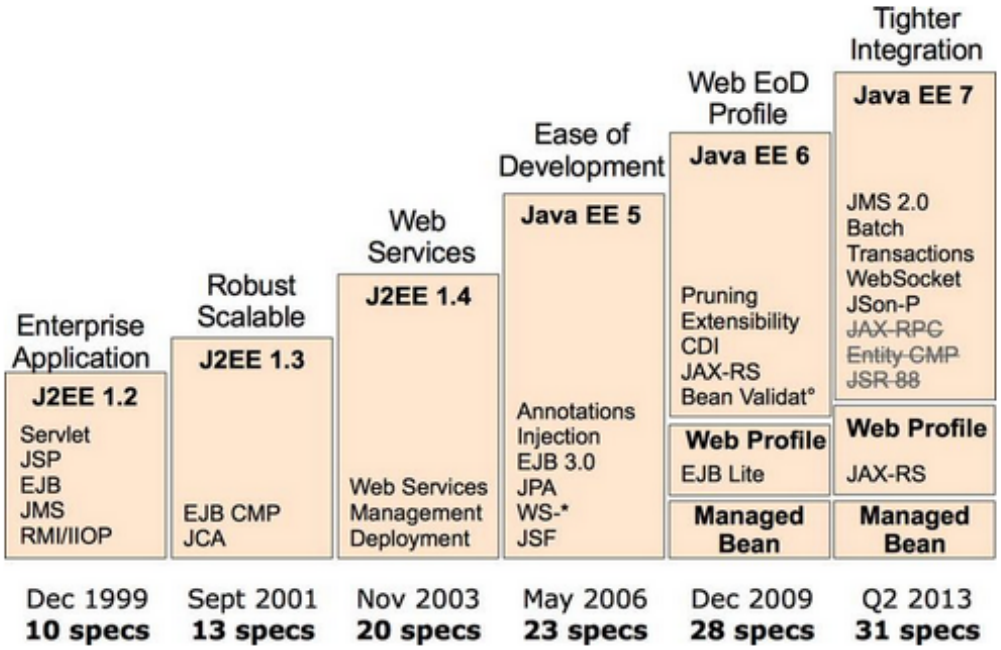


Figure 1-4. History of J2EE/Java EE

Java EE

- Chaque implémentation doit supporter :
  - Applet (swing dans navigateur)
  - Application (gui/batch)
  - Web application
  - Enterprise application

- Installation du socle de développement
  - JDK 7
  - IDE Eclipse
  - Implémentation Glassfish 4
  - Maven 3

JPA

Java Persistence API

- Pourquoi persister des données
  - Manipuler / Enregistrer / Rechercher
  - Procédure stockée / Index / Relation
- Comment l'intégrer dans un langage OO
  - Manipule des objets
  - Encapsule des états
  - Constructeur et GC mais ne perdure pas
  - Utilise alors un Object-Relation Mapping
- Les Frameworks
  - Hibernate
  - TopLink
  - …
- Utilisation Java EE 7 et JPA 2.1
- IR = EclipseLink 2.5 (avec XML Binding)

Java Persistence API

- JPA 2.1
  - JPA depuis Java EE 5
  - JDBC sans SQL
- Composé de :
  - ORM (manipuler les données)
  - EntityManager / JPQL (requêtage)
  - JTA (transactions)
  - Callback / listener
- Historique
  - Avant hibernate
  - Hibernate
  - Java EE 5 : JPA 1.0
  - Java EE 6 : JPA 2.0
  - Java EE 7 : JPA 2.1

JPA - Entity

- Objets à persister
- Mapping via annotations
  - @Entity - Les entités à sauvegarder
  - @Id - Les identifiants de ces entités
  - @Table - Les tables du SGBD
  - @Column - Les colonnes des tables

```java
@Entity
@Table(name = "car")
public class Voiture {

    @Id
    private Long id;

    @Column(name = "color")
    private String couleur;

    // ========================================
    // =          Constructors, Getters & Setters              =
    // ========================================
}
```

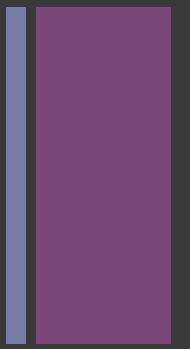
JPA - Entity

- Clés primaires :
  - ID Auto
- Tables secondaires
- Clés composées :
  - Embeddable (EmbeddedId)
  - IdClass
- Basic et Large Object

```java
@Entity
public class Voiture {

    @Id
    @GeneratedValue(strategy =
            GenerationType.AUTO)
    private Long id;

    //...
}
```

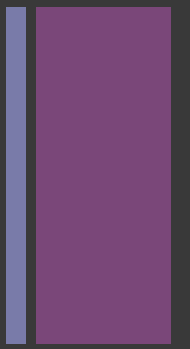
```java
@Entity
@Table(name = "address")
@SecondaryTables({
        @SecondaryTable(name = "city"),
        @SecondaryTable(name = "country")
})
public class Adresse {

        @Id
        private Long id;

        private String rue;

        @Column(table = "city")
        private String ville;

        // ...
}
```

```java
@Embeddable
public class NouvellesId {

    private String titre;

    private String langage;

     // ...
}
```



```java
@Entity
@Table(name = "news")
public class Nouvelles {

    @EmbeddedId
    private NouvellesId id;

    private String contenu;
     // ...
}
```

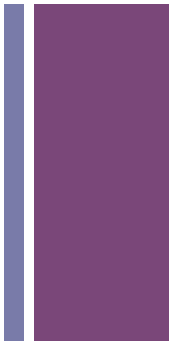
```java
@Embeddable
public class NouvellesId {

    private String titre;

    private String langage;

    // ...
}
```



```java
@Entity
@Table(name = "news")
@IdClass(NouvellesId.class)
public class Nouvelles {

    @Id
    private String titre;

    @Id
    private String langage;

    private String contenu;

    // ...
}
```

```java
@Entity
public class Musique {

    @Basic(fetch = FetchType.LAZY)
    @Lob
    private byte[] wav;

    //...
}
```

JPA - Entity

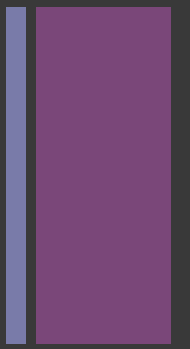


TP 1 à 9

JPA - Entity

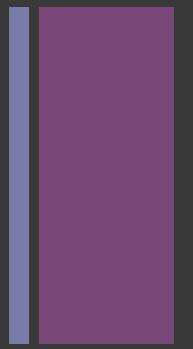
- Temporal
- Transient
- Enumeration
- Acces Type (FIELD / PROPERTY)
- Collection / Map of Basic
- XML Mapping

```java
@Entity
public class Voiture {

    @Temporal(TemporalType.DATE)
    private Date dateCreation;

    @Transient
    private Integer puissance;

    @Temporal(TemporalType.TIMESTAMP)
    private Date dateImmatriculation;


    //...
}
```



```java
public enum Couleur {

    ROUGE,
    JAUNE,
    BLANC

}
```

```java
@Entity
public class Voiture {

    @Enumerated(EnumType.STRING)
    private Couleur couleur;

    //...
}
```



```java
@Entity
public class Livre {

    @ElementCollection(fetch = FetchType.LAZY)
    @CollectionTable(name = "tag")
    @Column(name = "tag_value")
    private List<String> tags = new ArrayList<>();

    //...
}
```

```java
@Entity
public class CD {

    @ElementCollection
    @CollectionTable(name = "cd")
    @MapKeyColumn(name = "position")
    @Column(name = "titre")
    private Map<Integer, String> chanson = new HashMap<>();


    //...
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation=
                         "http://xmlns.jcp.org/xml/ns/persistence/orm http://xmlns.jcp.org/xml/ns/persistence/orm_2_1.xsd"
                 version="2.1">

  <entity class="org.kearis.formation.javaee7.chapter01.ex25.Book25">
    <table name="livre"/>
    <attributes>
      <basic name="titre">
        <column name="titre_livre" nullable="false" updatable="false"/>
      </basic>
      <basic name="description">
        <column length="2000"/>
      </basic>
      <basic name="nbPage">
        <column name="nb_page" nullable="false"/>
      </basic>
    </attributes>
  </entity>

</entity-mappings>
```

```java
@Entity
@Table(name = "pas_pris_en_compte")
public class Book25 {

        @Id
        private Long id;
        private String title;
        @Column(length = 500)
        private String description;
        private Integer nbOfPage;

}
```

JPA - Entity

TP 14 à 25

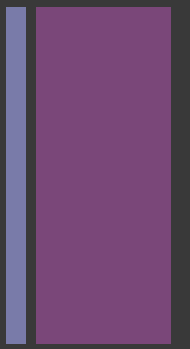
JPA - Relationnel

- Relation entre objets et tables
- Orienté pour les BDD relationnelles
- Plusieurs type de relations :
  - OneToOne
  - OneToMany
  - ManyToOne
  - ManyToMany
- Uni / Bi directionelle
- Définir qui porte la relation
- Stratégie de FETCH, CASCADE
- OrderBy >> OrderColumn

```java
@Entity
public class Client {

    @OneToOne (fetch = FetchType.LAZY)
    @JoinColumn(name = "address_fk", nullable = false)
    private Adresse adresse;


    //...
}
```

```java
@Entity
public class Commande {

    @OneToMany
    @JoinTable(name = "ord_line",
        joinColumns = @JoinColumn(name = "order_fk"),
        inverseJoinColumns = @JoinColumn(name = "order_line_fk"))
        private List<LigneCommande> lignesCommande;

        //...
}
```

```java
@Entity
public class Commande {
        @OneToMany(fetch = FetchType.EAGER)
        @JoinColumn(name = "order_fk")
        @OrderBy("numeroLigne DESC")
        private List<LigneCommande> lignesCommande;

        //...
}
```



```java
@Entity
public class CD {

    @ManyToMany(mappedBy = "apparitionSurCd")
    private List<Artist> artistes;

}
```

```java
@Entity
public class Artiste {

    @ManyToMany
    @JoinTable(name = "jnd_art_cd",
        joinColumns = @JoinColumn(name = "artist_fk"),
        inverseJoinColumns = @JoinColumn(name = "cd_fk"))
    private List<CD> apparitionSurCd;

}
```

JPA - Relationnel

TP 34 à 51

JPA - Héritage

- Stratégie:
  - SINGLE TABLE (défaut, une seule table)
  - JOINED (une table par classe)
  - TABLE PER CLASS (une table par classe concrète)

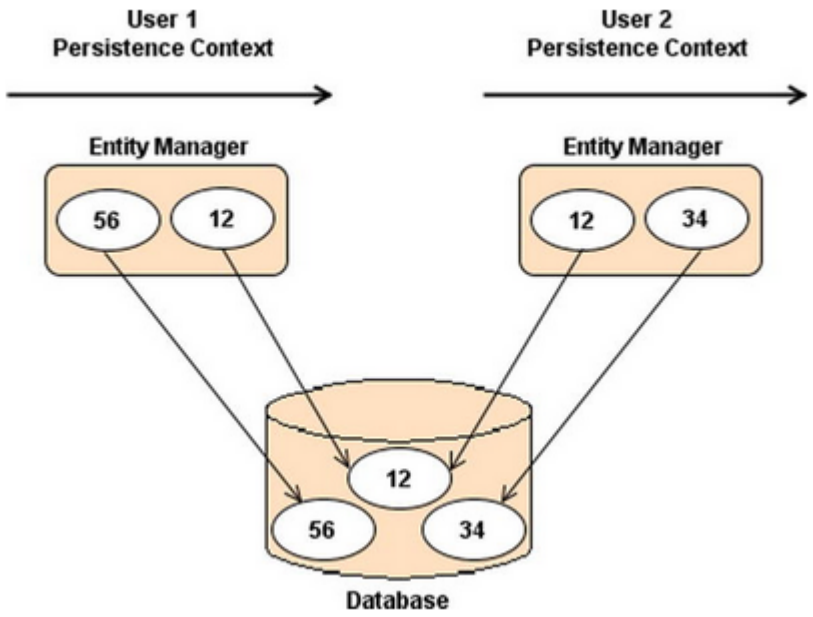
JPA - Héritage

TP 53 à 61

JPA - Objet persistant

- EntityManager
  - Gestion état et cycle de vie d'un Entity
  - Recherche par requete / critère d'un Entity
  - cf *AbstractPersistentTest.java*
- Persistence Context
  - Instance d'entity unique à l'instant T
  - L'entityManager ne gère que les entités du contexte
  - PU : Pont entre PersitenceContexte et DB

JPA - Objet persistant



JPA - Objet persistant

- JPQL
  - SELECT
  - UPDATE
  - DELETE
- Exemples :
  - SELECT c.nom FROM Client c WHERE c.age > 18
  - SELECT c FROM Client WHERE c.nom = **:nom**

JPA - Objet persistant

- Queries
  - Dynamic
    - em.createQuery(« SELECT c FROM Client c » + param)
  - Named
    - @NamedQuery sur les Entity
    - em.createNamedQuery(« findAll », Client.class)
  - Criteria API
    - criteriaBuiler.createQuery(Client.class).from(…).select(…) .where(…)
    - desc, asc, avg, sum, lowerThan, …

JPA - Objet persistant

- Queries
  - Natives
    - Idem NativeQuery mais en SQL
    - Non portable DB
  - Procédures stockées
    - em.createStoredProdureQuery(« proc »)
    - query.registerStoredProcedureParameter(…)
    - query.setParameter()

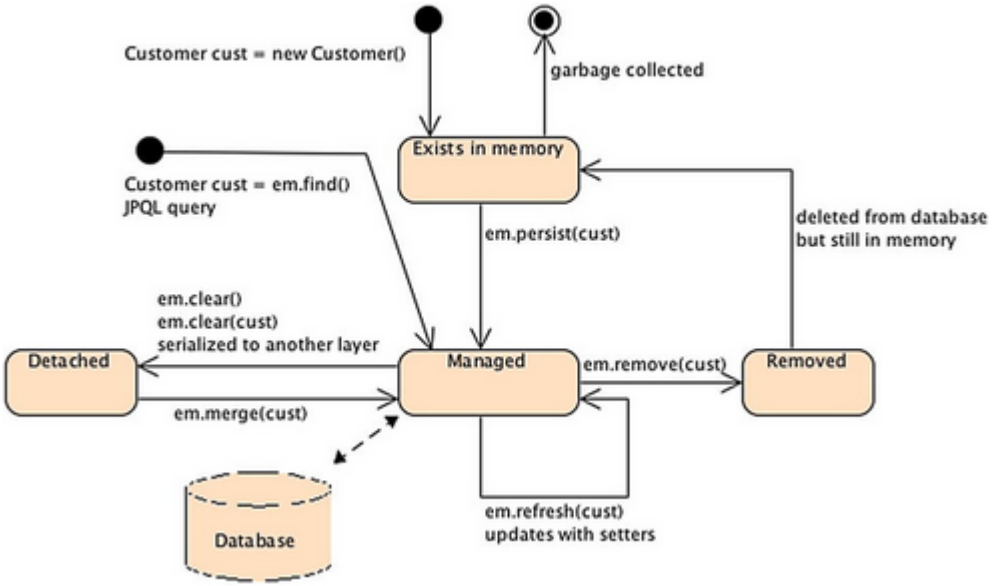
JPA - Objet persistant

TP 3 à 29

JPA - Avancé

- Cache
- Concurrence
- Cycle de vie

JPA - Avancé

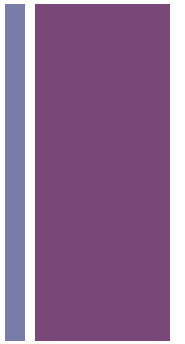

Customer cust = new Customer()
garbage collected
Exists in memory
Customer cust = em.find()
JPQL query
em.persist(cust)
deleted from database
but still in memory
em.clear()
em.clear(cust)
serialized to another layer
Detached
Managed
em.remove(cust)
Removed
em.merge(cust)
Database
em.refresh(cust)
updates with setters

JPA - Avancé

- Callback
  - Pre / Post
  - Persist / Load / Remove / Update
- Entity Listeners
  - Callback dans classe séparée

JPA - Avancé

TP 34 à 42



Bean Validation

Bean Validation

- But : Valider les données
  - Bean Validation 1.0 : Java EE6
  - Bean Validation 1.1 : Java EE7
- RI : Hibernate Validator
- Utilise des Contraintes
  - @NotNull
  - @Size
  - @Min
  - @Past
  - @Constraint(validatedBy = {MyValidator.class})

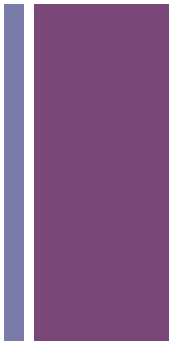
Bean Validation

- Type (Level)
  - Attributs
  - Classes
    - Valide plusieurs paramètres
    - Valide métier
  - Méthodes
    - Valide résultats
    - Valide les paramètres
- Message de validation
- Héritage
- Contexte

Bean Validation

- Utiliser les validations
  - Validation.buildDefaultValidatorFactory().getValidator()
  - validate(bean)
  - validateProperty(bean, « myProperty »)
  - validateValue
  - forExecutables().validateParameters()

Bean Validation



TP 1 à 21

EJB

EJB

- Entreprise JavaBeans 3.2
  - Couche métier
  - Coté serveur
  - Facile à utiliser
- 3 types (session bean):
  - Stateless
  - Stateful
  - Singleton
- Message-driven beans (Chapitre Messaging)
- Serveur embarqué (depuis EJB 3.1)

EJB

- Fonctionnalités
  - Communication distante
  - Injection de dépendance (CDI)
  - Gestion des états de sessions
  - Pooling : File d'attente partagées / réutilisées
  - Gestion du cycle de vie
  - Messaging
  - Transaction
  - Concurrence
  - Intercepteur
  - Mode asynchrone

EJB

- Fonctionnalités EJB Lite
  - Session beans
  - Pas d'interface
  - Interface locale
  - Intercepteur
  - Transaction
  - Securité
  - Mode embarqué
- En plus dans Full EJB
  - Appel asynchrone
  - Messaging
  - Interface distance
  - Web Service
  - Timer service
  - RMI/IIOP

EJB

- Glassfish v4
  - Implémentation de référence
  - Installation et démarrage
    - as start-domain
    - as start-database
    - déploy

EJB

- Session : Stateful / Stateless / Singleton
- Visibilité : Remote / Local / No-Interface
- JNDI
  - name : java:<scope>[/<app-name>]/<module-name>/<bean-name>[!<fully-qualified-interface-name>]
  - scope : global / app /module / comp
  - app-name : nom de EAR ou WAR
  - module-name : nom du module
  - interface-name : nom de l'interface (avec pkg)

EJB

- Stateless
  - Pas d'état pour le client
  - Pooling (instances multiples)
- Stateful
  - Un bean par client
  - Gestion timeout session / @Remove session
  - Pooling (instances multiples)
- Singleton
  - Une seule instance / gestion concurrence
  - Chargé au démarrage avec @Startup / Possibilité ordre

+

EJB

TP 1

Créer un EJB remote sans état

+

EJB

TP 1

Créer un EJB remote sans état

+

EJB

TP 7

Créer un EJB remote avec état

+

EJB

TP 9

Créer un EJB remote singleton

EJB

- Injection de dépendance
  - @Inject
  - @EJB
  - @Resource
  - …
- Contexte de session
  - API SessionContext gestion de session
- Appel Asynchrone

```
@Resource
private SessionContext context;

@Override
public Book14 createBook(Book14 book) {
  if (!context.isCallerInRole("admin"))
    throw new SecurityException("Only admins can create");

  em.persist(book);

  if (inventoryLevel(book) == TOO_MANY_BOOKS)
    context.setRollbackOnly();

  return book;
}
```

+

EJB

TP 7

Créer un EJB remote avec des méthodes asynchrones

EJB

- Invocation
  - Injection
    - @EJB
    - Param lookup = « java:global/… »
  - CDI
    - @Inject

Web Service

REST

WS REST

- Architecture RESTful
  - REpresentational State Transfer
  - Protocole HTTP
  - Pas de contrat de service
  - WEB expérience
  - Ressources et URI
- Méthodes HTTP
  - GET
  - POST
  - PUT
  - DELETE
  - HEAD, TRACE, OPTIONS, CONNECT

WS REST

- Content types
  - type/sous-type
- Status codes
  - 1xx : INFO
  - 2xx : SUCCES
  - 3xx : REDIRECT
  - 4xx : CLIENT ERROR
  - 5xx : SERVER ERROR
- Cache

WS REST

- Contrat
  - Web Application Description Language
  - Non Standard
- JAX-RS 2.0
  - RI : Jersey
- Produit / Consomme
  - XML
  - JSON
  - TEXT
  - …

WS REST

- Bonnes pratiques
  - Keep it simple, stupid
  - Versioning
  - Granularité moyenne
  - Nommage / Ressource
  - Pagination
  - Tri
  - Méthode HTTP
  - Stateless
  - Cache (Etag)

WS REST

- Structure du WS
  - Définir le PATH
  - Définir les méthodes HTTP
  - Définir les paramètres
  - Définir les types à produire
  - Définir les types à consommer
  - Renvoyer un code de retour
- URI
  - http://host:port/path?queryString#fragment

WS REST

- Structure du WS
  - Définir le PATH
  - Définir les méthodes HTT
  - Définir les paramètres
  - Définir les types à produi
  - Définir les types à consor
  - Renvoyer un code de reto
- URI
  - http://host:port/path?que

```
@Path("/books")
@Produces({MediaType.APPLICATION_JSON})
@Consumes({MediaType.APPLICATION_XML})
class Book {

@GET
public Response getBook(@PathParam("id") String id) {
 Book b = …
  return Response.ok(book).build();
}

@POST
public Response createBook(Book book) {
  URI uri = …
  return Response.created(uri).build();
}
```

WS REST

- Récupération
  - @GET
  - uriInfo.getAbsolutePathBuilder().path(bookId).toString()).build()
- Création
  - @POST
- Mise à jour
  - @PUT
  - ~~@PATCH~~
- Suppression
  - @DELETE

WS REST

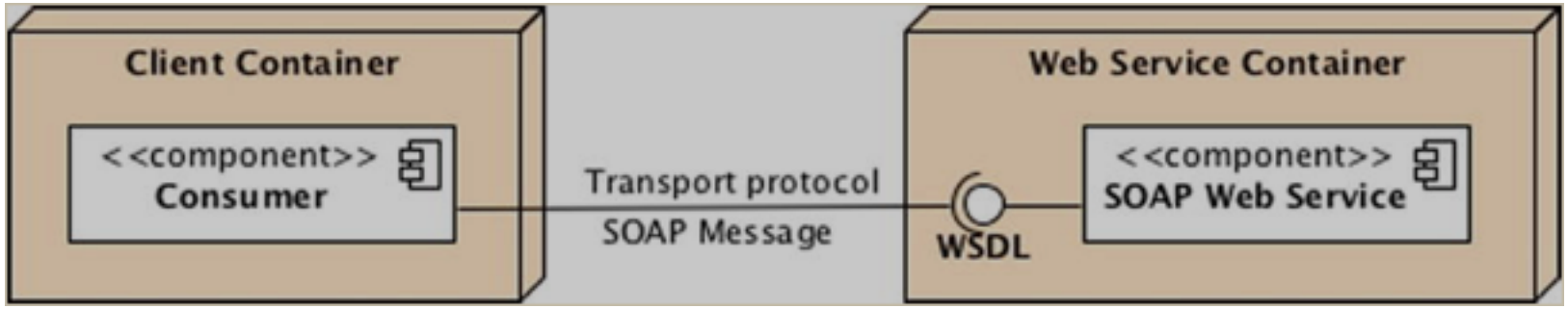
- URI Application
  - @ApplicationPath(String)
- Erreur
  - JAX-RS 2 Exception
  - Code HTTP
- EJB
  - Stateless / Singleton
- Injection du @Context

Web Service

SOAP

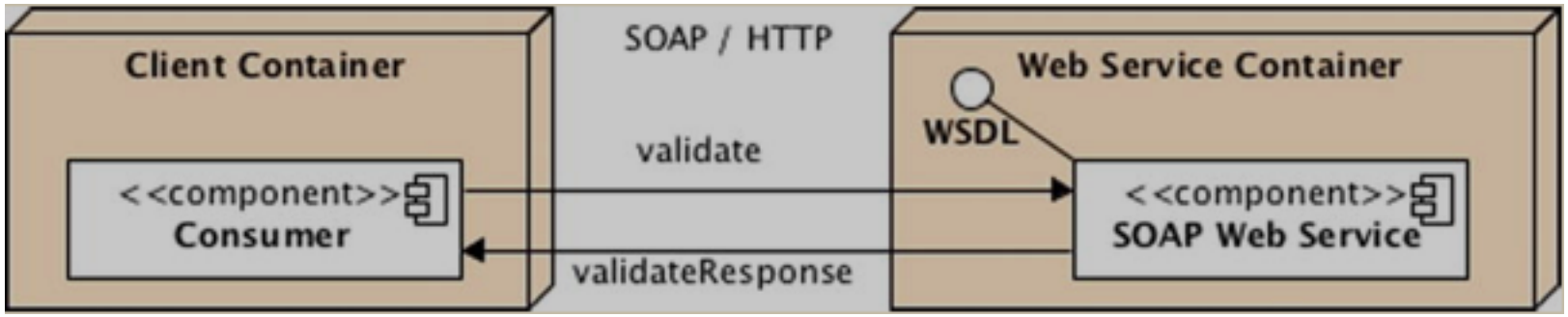
WS SOAP

- Architecture SOA
  - Service Oriented Architecture
  - WS Simple Object Access Protocol
  - Contrat de service
- WSDL
  - Web Services Description Language
  - XML/XSD
  - Basé sur HTTP (mais pas seulement)



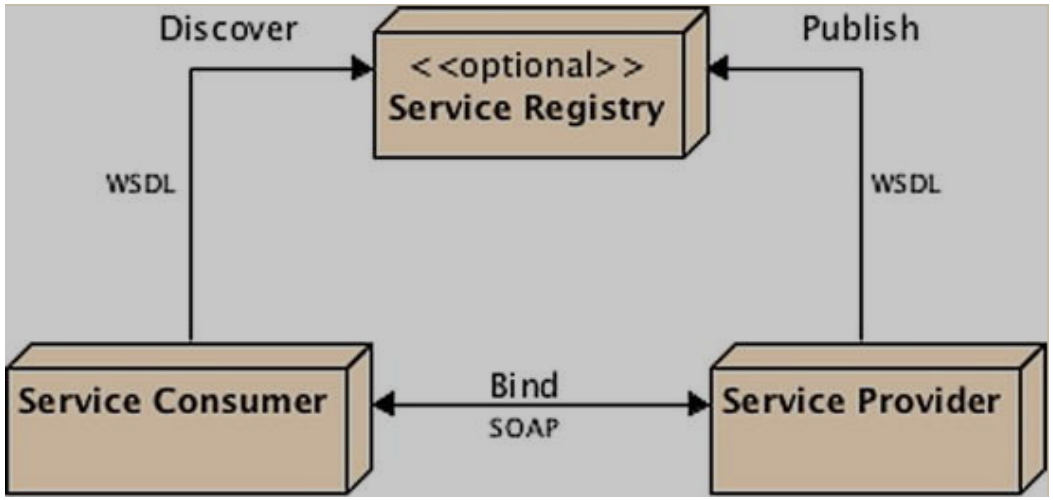
WS SOAP

- Protocole SOAP
  - Implémentation de la WSDL
  - Messages au format XML
  - Différents éléments
    - Envelope : Root element / namespace / message
    - Header : Optionnel / securité/ …
    - Body : Corps du message
    - Fault : Optionnel / Définition des erreurs possibles



WS SOAP

- Transport
  - HTTP / HTTPS (par défaut)
  - Mais aussi FTP / SMTP / …
- UDDI
  - Annuaire de services
  - Pas le succès escompté



WS SOAP

- Différents type de format de binding
  - Document/Literal (par défaut)
  - Document/Encoded (non WS-* compliant)
  - RPC/Literal
  - RPC/Encoded

WS SOAP

- JAX-WS 2.2.a
  - API pour produire et consommer des WS
- Web Services 1.3
  - Comportement dans le container
- WS-Metadata 2.3
  - Pour simplifier le dév et deploy
- Implémentation
  - IR : Métro
  - Autres : CXF / Axis / …

WS SOAP

- Produire un WS SOAP
  - Déclaration @WebService
  - Classe ou interface
  - Si interface : définir endpoint
- Déclarer les méthodes
  - Utilisation de @WebMethod
  - Comportement par défaut
  - Pas obligatoire
  - Possibilité d'exclure

```
@WebService
public class BookService {

  @WebMethod
  public Book getBook(String id) {
     Book b = …
     return b;
  }

}
```

WS SOAP

- Définir les paramètres
  - Déclaration @WebParam
  - Comportement par défaut
- Définir le type de réponse
  - Utilisation de @WebResult
  - Comportement par défaut
  - Pas obligatoire
- Méthode sans retour
  - Utilisation de @OneWay
  - Exemple : Mode asynchrone

```
@WebMethod
 public Book getBook(@WebParam(name=« livreId »
String id) {
    Book b = …
    return b;
 }

@WebMethod
@OneWay
 public void wakeUp() {
    // -- asynchronous logic
    // -- … no response / no exception
 }
}
```

WS SOAP

- Gérer les exceptions
  - Définition dans la signature
  - Utilisation de @WebFault
  - SOAPFactory pour détails

```
@WebService
class Book {

 @WebMethod
  public void foo() throw MyException {
     throw new MyException(« erreur !!! »);
  }
}

 @WebFault(name=«  MyFault»)
  class MyException  extends Exception {

    // -- constructors
  }
```

WS SOAP

- Gérer le conexte
  - Injection via @Resource
  - Utilisation de @WebFault
  - SOAPFactory pour détails

```
@WebService
class Book {

 @Resource
 private WebServiceContext context;

 public boolean validate(CreditCard creditCard) {
    if (! context .isUserInRole(Admin)) {
        throw new SecurityException("Only Admin");
    }

    // -- Business logic
 }
}
```

+

WS SOAP

TP 1

Créer un web service SOAP et le déployer dans Glassfish

+

WS SOAP

TP 2

Consommer un web service SOAP via SOAPUI

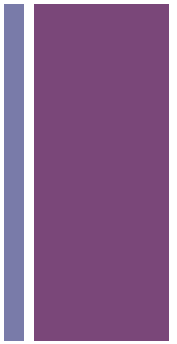
WS SOAP

- Invoquer un WS SOAP
  - Invocation manuelle
  - Injection via @WebServiceRef
  - ~~@Inject~~ pas d'update dans Java EE 7
  - Utiliser les @Producers pour utiliser CDI

- Génération du client
  - Utilisation de plugin maven
  - jaxws-maven plugin -> wsimport

+

WS SOAP



TP 3

Injecter un web service SOAP dans un Bean

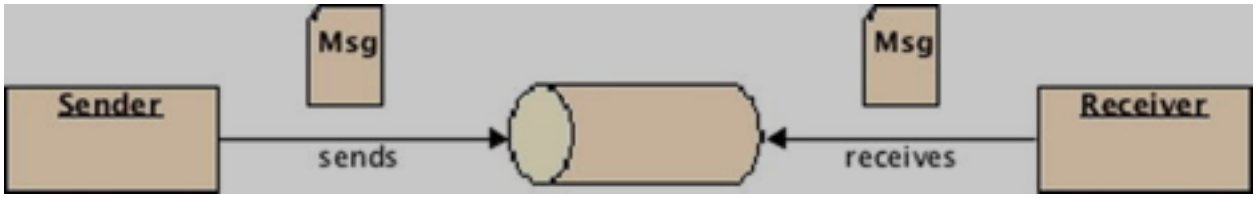
Messaging

Messaging

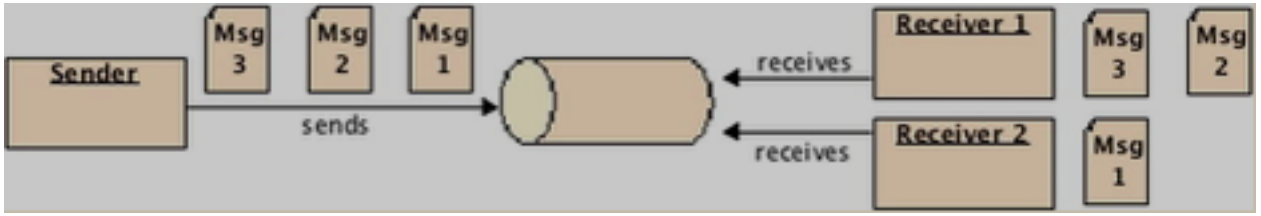
- Asynchronisme
  - Appel sans réponse, ni résultat
  - Producteur → Destination ← Consommateurs
- Message
  - Message-Oriented Middleware
  - Lu par un ou plusieurs consommateurs
  - Envoyer un n'importe quel producteur

Messaging

- Point-to-Point : Queue
  - Envoi par un producteur
  - Reçu par un seul consommateur



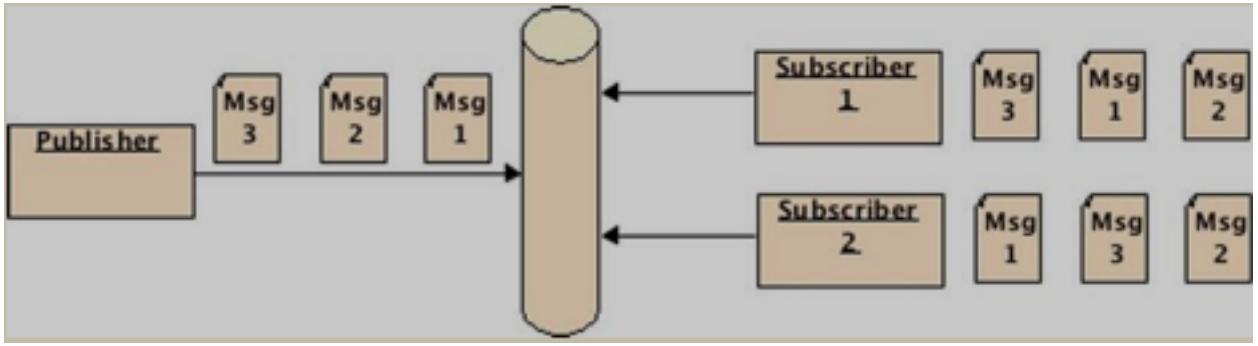
  - …même s'il y en a plusieurs



  - Pas d'ordre dans les messages

Messaging

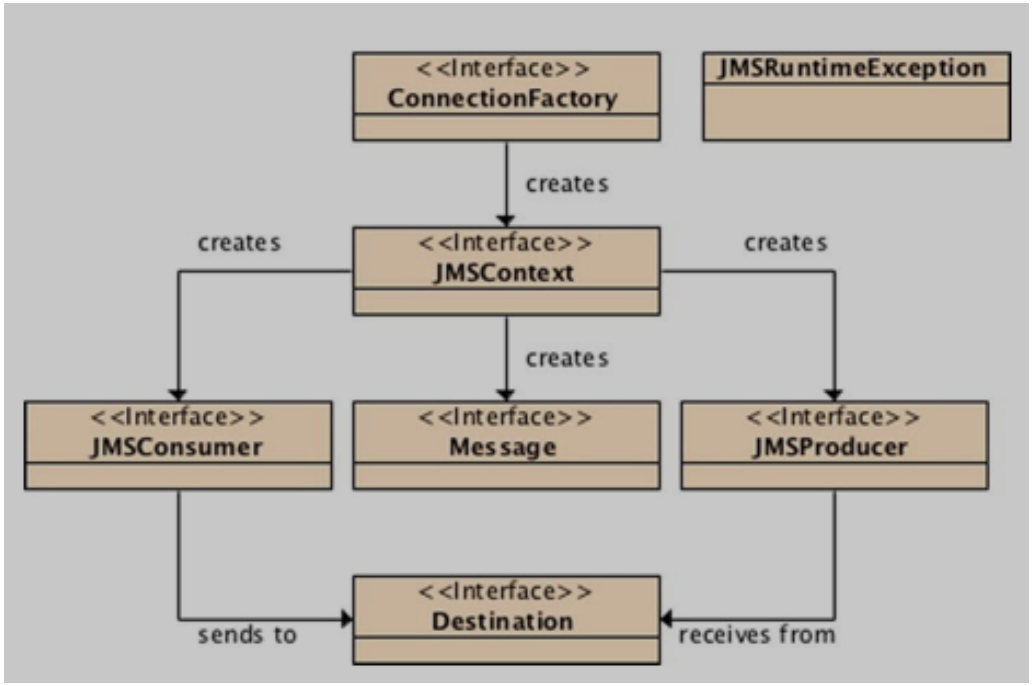
- Publish-Subscribe : Topic
  - Envoi par un producteur
  - Reçu par un plusieurs souscripteurs



  - Ne voit que les messages arrivés après la souscription
  - Souscripteur peut être inactif, il reçoit message quand redevient actif

Messaging

- JMS 2.0
  - IR : OpenMQ
  - Java EE 7 : Simplification de l'API
- EJB 3.2
  - MDB pour consommateur
  - @Asynchronous

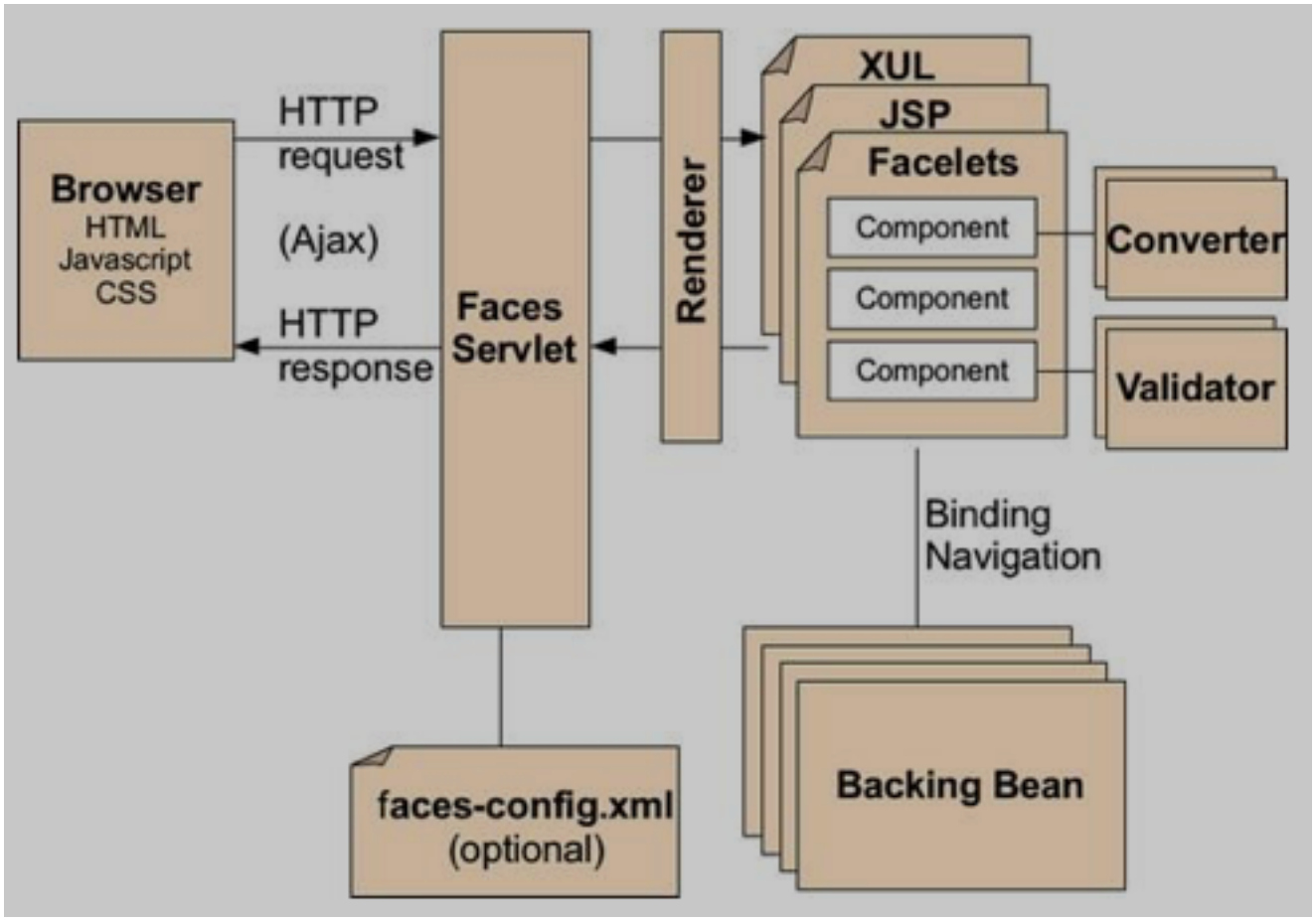


JSF

JSF

- Framework pattern MVC
  - API Servlet
  - Basé sur les composants (!= action)
- Structure
  - Vue : Facelet (.xhtml) ou JSP
  - Modèle : Entités / JavaBeans
  - Contrôleur unique FacesServlet + ManagedBean
  - Pas de servlets spécifiques par page

JSF


Browser
HTML
Javascript
CSS
HTTP
request
(Ajax)
HTTP
response
Faces
Servlet
Renderer
XUL
JSP
Facelets
Component
Component
Component
Converter
Validator
Binding
Navigation
Backing Bean
faces-config.xml
(optional)

JSF

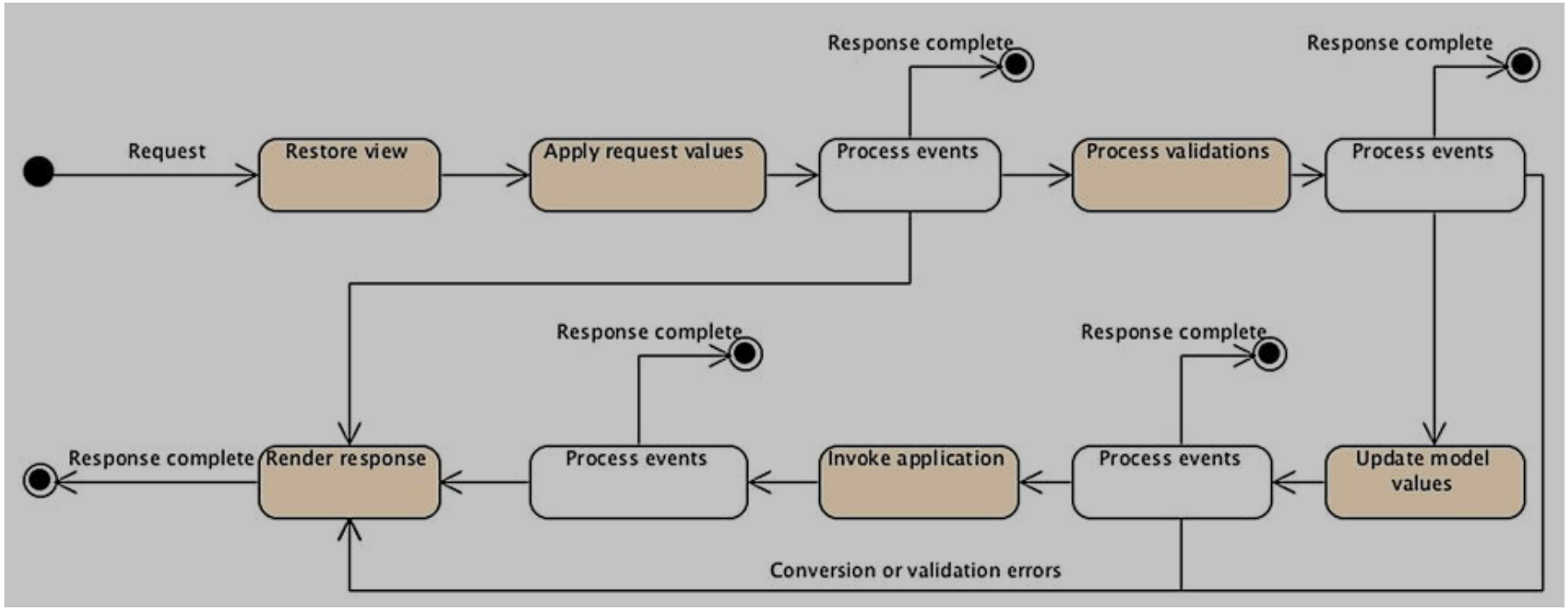
- Pages et composants
  - JSF permet de multiple PDLs
  - Mais Facelets est recommandé depuis JSF 2.0
- Renderer
  - Affichage des composants
  - Traduction des saisies utilisateurs
- Converter
- Validator

JSF

- FacesServlet
  - Point d'entrée de toute requête
  - Contrôleur unique du cycle de vie du traitement des requêtes
- Backing beans
  - Couche métier
- EL (Expression Language)
  - Liaison des variables/actions entre composant et backing bean
  - Syntaxe : #{…}

JSF

- Cycle de vie d'une page JSF



JSF

- Les bases d'une vue
  - Standard XHTML
  - Bibliothèque et balise

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
    ...
</html>
```

  - Bibliothèque JSTL (HTML / templating / core / …)

```
<h:outputLabel for="confirmation">Confirmation du mot de passe <span
class="requis">*</span></h:outputLabel>
```

  - EL

```
#{inscrireBean.utilisateur.motDePasse}
```

JSF

- Les bases d'une vue
  - Appel d'un controleur
  - Appel d'un EJB (avec @Named sur EJB)

```
<h:dataTable value=" #{bookEJB.findAllBooks()} " var="book">
  <h:column>
    <h:outputText value="#{book.title}"/>
  </h:column>
</h:dataTable>
```

```
@Named
@Stateless
public class BookEJB {

  @Inject
  private EntityManager em;

  public List<Book> findAllBooks() {
    return em.createNamedQuery("findAllBooks", Book.class).getResultList();
  }
}
```

JSF

- URI bibliothèques
  - h: http://xmlns.jcp.org/jsf/html
  - f: http://xmlns.jcp.org/jsf/core
  - ui: http://xmlns.jcp.org/jsf/facelets
  - composite: http://xmlns.jcp.org/jsf/composite
  - c: http://xmlns.jcp.org/jsp/jstl/core
  - fn: http://xmlns.jcp.org/jsp/jstl/functions

JSF

- JSF HTML Components Tags
  - Bases

```
<h:body>
<h:head>
<h:form>
<h:doctype>
<h:outputScript>
<h:outputStylesheet>
```

JSF

- JSF HTML Components Tags
  - Command
    - Bouton <h:commandButton/>

```
<h:commandButton value="A submit button"/>
<h:commandButton type="reset" value="A reset button"/>
<h:commandButton image="book.gif" title="A button with an image"/>
```

    - Lien <h:commandLink/>

```
<h:commandLink>A hyperlink</h:commandLink>
```

```
<h:commandLink action="#{bookController.doNew}" >
  Create a new book
</h:commandLink>
```

  - Target (idem mais sans action)

```
<h:button outcome="newBook.xhtml" value="A bookmarkable button link"/>
<h:link outcome="newBook.xhtml" value="A bookmarkable link"/>
```

JSF

- JSF HTML Components Tags
  - Input

```
<h:inputHidden value="Hidden data"/>
<h:inputSecret maxlength="8" />
<h:inputText value="An input text"/>
<h:inputText size="40" value="A longer input text"/>
<h:inputTextarea rows="4" cols="20" value="A text area"/>
<h:inputFile/>
```

JSF

- JSF HTML Components Tags
  - Output

<h:outputLink> is that the latter displays the link but doesn't invoke any backing bean method when clicked. It just creates an external link or anchor.

```
<h:outputLabel value=" #{bookController.book.title} "/>
<h:outputText value="A text"/>
<h:outputLink value=" http://www.apress.com/">A link</h:outputLink>
<h:outputFormat value="Welcome {0}. You have bought {1} items">
  <f:param value="#{user.firstName}" />
  <f:param value="#{user.itemsBought}" />
</h:outputFormat>
```

The preceding code doesn't have any special graphical representation, just text. Following is the HTML rendering:

```
<label>The title of the book</label>
A text
<a href=" http://www.apress.com/">A link</a>
Welcome Paul. You have bought 5 items
```

JSF

- JSF HTML Components Tags
  - Sélection

```
<h:selectOneMenu>
  < f:selectItem itemLabel="History"/>
  <f:selectItem itemLabel="Biography"/>
  <f:selectItem itemLabel="Literature"/>
  <f:selectItem itemLabel="Comics"/>
  <f:selectItem itemLabel="Child"/>
  <f:selectItem itemLabel="Scifi"/>
</h:selectOneMenu>
```

JSF

- JSF HTML Components Tags
  - Images

```
<h:graphicImage value="book.gif" height="200" width="320"/>
```

  - Table

```
<h:panelGrid columns="3" border="1">
  <f:facet name="header">
    <h:outputText value="Header"/>
  </f:facet>
  <h:outputLabel value="One"/>
  <h:outputLabel value="Two"/>
  <h:outputLabel value="Three"/>
  <h:outputLabel value="Four"/>
  <h:outputLabel value="Five"/>
  <h:outputLabel value="Six"/>
  <f:facet name="footer">
    <h:outputText value="Footer"/>
  </f:facet>
</h:panelGrid>
```

JSF

- JSF HTML Components Tags
  - Messages d'erreur

```
< h:messages style="color:red"/>
<h:form>
  Enter a title:
  <h:inputText value="#{bookController.title}" required="true"/>
  <h:commandButton action="#{bookController.save}" value="Save"/>
</h:form>
```

JSF

- JSF Core Tags

```
<f:facet>
<f:attribute>
<f:param>

<f:actionListener><f:valueChangeListener><f:propertyActionListener>

<f:phaseListener>

<f:converter><f:convertDateTime><f:convertNumber>


<f:validator><f:validateDoubleRange><f:validateLongRange><f:validateLength><f:validateRegex>

<f:validateBean>
<f:loadBundle>
<f:selectItem><f:selectItems>
<f:ajax>
```

JSF

- JSTL Tags
  - Core actions

JSF

```
<h:body>
 <h1>Checking numbers</h1>
 <hr/>
 <c:set var="upperLimit" value="20"/>
 <c:forEach var="i" begin="3" end="#{upperLimit - 5}">

  <c:choose>
   <c:when test="#{i%2 == 0}">
    <h:outputText value="#{i} is even"/><br/>
   </c:when>
   <c:otherwise>
    <h:outputText value="#{i} is odd"/><br/>
   </c:otherwise>
  </c:choose>

 </c:forEach>
 <hr/>
 <h:outputText value="APress - Beginning Java EE 7"
style="font-style: italic"/>
</h:body>
```

JSF

- JSF Templating Tags
  - Composition : Définir des templates (layout) reutilisables

JSF

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

<h:head>
 <title>
  <ui:insert name="title">Default title</ui:insert>
 </title>
</h:head>
<h:body>
 <h1>
  <ui:insert name="title">Default title</ui:insert>
 </h1>
 <hr/>

 <ui:insert name="content">Default content</ui:insert>

 <hr/>
 <h:outputText value="…" />

</h:body>
</html>
```

JSF

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

<ui:composition template="layout.xhtml">

 <ui:define name="title">Create a new book</ui:define>

 <ui:define name="content">

  <h:form>
   <h:panelGrid columns="2">
    <h:outputLabel value="Title :"/>
    <h:inputText value="#{bookController.book.title}"/>

    <h:outputLabel value="Price : "/>
    <h:inputText value="#{bookController.book.price}"/>

    <h:outputLabel value="Description  : "/>
    <h:inputTextarea value="#{bookController.book.description}" cols="20" rows="5"/>

   </h:panelGrid>
   <h:commandButton value="Create a book" action="#{bookController.doCreateBook}"/>

  </h:form>

 </ui:define>

</ui:composition>
</html>
```

JSF

- Ajax

```
<h:commandButton value="Create a book" action="#{bookController.doCreateBook}">
  < f:ajax execute="@form" render=":booklist"/>
</h:commandButton>
```

```
<h:commandButton value="Create a book" onclick =" jsf.ajax.request ( this , event,
        { execute :'isbn title price description nbOfPage illustrations',
         render :'booklist'}); return false;"
        actionListener="#{bookController.doCreateBook}" />
```