Java Enterprise Edition

Noël De Palma
UGA
noel.depalma@imag.fr

Remerciements

Lionel Seinturier
Fabienne Boyer
Daniel Hagimont

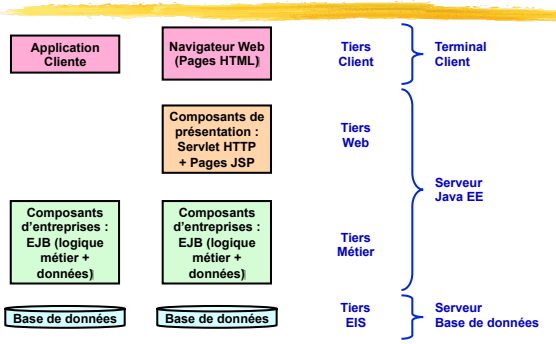
Plan

- Intro JEE
- EJB
  - Annotations
  - Session
  - Entity
  - MDB
- Conclusion

Java EE pour quoi faire ?

- Infrastructure « serveur » pour le support d'applications Web ou d'entreprise
  - E-commerce, Système d'information, telecoms, etc
  - Support de propriétés non-fonctionelles : persistance, transaction, sécurité ...
  - Connexion standard à des systèmes d'information externes (accès au « legacy »)
- Architecture multi-tiers
  - Présentation, Métier, BD
  - Architecture client léger (basée « browser »)
  - Architecture client lourd (GUI avancées)

Architectures multi-tiers



Intègre de nombreux standards pour l'entreprise

- Standard de communication
  - RMI : synchrone (TCP + serialisation java)
  - JAX-WS : synchrone ou pas (HTTP+SOAP+XML)
  - JMS : asynchrone (queue de messages)
- Standard pour les services système
  - JNDI (annuaire), JTA (transaction), JPA (Persistance) ,JAAS (Sécurité), JMX (management)
- Standard de connexion à des *legacy*
  - *JCA*
- *Standard de développement*
  - *Servlet, JSP, JFS, EJB*

Composants EJB

- Enterprise Java Bean (EJB)
  - *Composant métier coté serveur*
  - les propriétés non-fonctionnelles sont fournies par le conteneur
  - la logique de présentation est du ressort du client
- Vocabulaire dans ce cours : *bean* = EJB = composant
- 3 types d'EJB gérés par le container
  - Session : *performs a task for a client*
  - *Message-Driven : listener processing messages asynchronously*
  - Entity : *represents a business entity object that exists* in persistent storage
- Accès Local/RMI/JMS/WS

Annotations Java 5

- Mécanisme standard dans le langage Java depuis version 5 (1.5)
- Idée similaire aux commentaires Javadoc
  - informations attachées à des éléments de programme (classe, méthode, attributs, …)
- @*Identificateur*
- Eventuellement des paramètres : @*Identificateur*(*name*=*value*,...)
  - types autorisés
    - primitifs, String, Class, annotation
    - tableaux de primitifs, String, Class, annotation
- Eventuellement plusieurs annotations par éléments

Exemple

```
@Resource(name="myDB", type=javax.sql.DataSource.class)
@Stateful
public class ShoppingCartBean implements ShoppingCart { ... }
```


Session Bean (définition)

- Session Bean : représente un traitement (services fournis à un client)
- Stateless session bean
  - sans état
  - ne conserve pas d'information entre 2 appels successifs
  - 2 instances qqconques d'un tel *bean* sont équivalentes
- Stateful session bean
  - avec un état (en mémoire)
  - similaire session servlet/JSP
  - même instance pendant toute la durée d'une session avec un client
  - 1 instance par client

Session Bean (développement)

- 1 interface (éventuellement 2 : Local + Remote) + 1 classe
- Interface
  - annotations @javax.ejb.Local ou @javax.ejb.Remote

```
import javax.ejb.Remote;
 @Remote
 public interface CalculatriceItf {
     public double add(double v1,double v2);
     public double sub(double v1,double v2);
     public double mul(double v1,double v2);
     public double div(double v1,double v2);
 }
```


Session Bean (développement)

- Classe
  - annotation @javax.ejb.Stateless ou @javax.ejb.stateful

```
import javax.ejb.Stateless;
 @Stateless
 public class CalculatriceBean implements CalculatriceItf {
     public double add(double v1,double v2) {return v1+v2;}
     public double sub(double v1,double v2) {return v1-v2;}
     public double mul(double v1,double v2) {return v1*v2;}
     public double div(double v1,double v2) {return v1/v2;}
 }
```


Session Bean (Nommage)

- **Possibilité de nommer les *beans* :**
- @Stateless(**name="foobar"**) :
  - Un nom local unique dans l'ejb-jar
  - **Nom de la classe du bean par defaut**

Ex : **@Stateless(name="foobar")**

```
public class CalculatriceBean implements CalculatriceItf {...}
```

Session Bean (Reference)

- Injection dans les clients (servlets ou beans) par annotations des attributs
  - @EJB
    - Nom local par défaut utilisé
  - @EJB(beanName= "foobar")
    - Client et Bean sont dans le même serveur
- Lookup explicite dans l'annuaire JNDI
    - Client et Bean ne sont pas forcement dans le même serveur, il faut forcement une remote interface !!!

Session Bean (ref exemple)

- Exemple Client local
  - typiquement une servlet **colocalisée** sur le même serveur que le *bean*

```
public class ClientServlet extends HttpServlet {
 @EJB(name="foobar")  //ou @EJB si nom par défaut
 private CalculatriceItf myBean;
 public void service( HttpServletRequest req, HttpServletResponse resp ) {
     resp.setContentType("text/html");
     PrintWriter out = resp.getWriter();
    double result = myBean.add(12,4.75);
     out.println("<html><body>"+result+"</body></html>");
 } }
```


Session Bean (ref exemple)

- Exemple client distant
  - Récupération de la référence vers l'annuaire JNDI
  - Recherche du *bean* dans l'annuaire

```
public class Client {
 public static void main(String args[]) throws Exception {
     javax.naming.Context ic = new javax.naming.InitialContext();
     CalculatriceItf bean = (CalculatriceItf) ic.lookup(jndiName);
     double res = bean.add(3,6);
 } }
```

!!! : init du context et jndiName : pas normalisé dans ejb 3.0
------> non portable, voir la doc du serveur

Statefull Session Bean

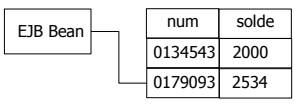
- Instance du *bean* reste en mémoire tant que le client est présent
  - Expiration au bout d'un délai d'inactivité
  - Similaire session JSP/servlet
- Utilisation type
  - gestion d'un panier électronique sur un site de commerce en ligne
  - rapport sur l'activité d'un client
- 2 annotations principales
  - @Stateful : déclare un *bean* avec état
  - @Remove
    - définit la méthode de fin de session
    - la session expire à l'issu de l'exécution de cette méthode

Statefull Session Bean

```
@Stateful
public class CartBean implements CartItf {
    private List items = new ArrayList();
    private List quantities = new ArrayList();

    public void addItem( int ref, int qte ) { ... }
    public void removeItem( int ref ) { ... }
    @Remove
    public void confirmOrder() { ... }
}

// ! ne pas utiliser l'injection mais le lookup jndi, stocker la référence
    comme donnée de session (de la servlet par ex).
```


Entity Bean (définition)

- Représentation d'une donnée manipulée par l'application
  - Donnée typiquement stockée dans un SGBD (ou tout autre support accessible en JDBC)
  - Correspondance objet – tuple relationnel (*mapping O/R*)
  - Possibilité de définir des clés, des relations, des recherches
- Avantage : manipulation d'objets Java plutôt que de requêtes SQL
- Mis en œuvre à l'aide
  - d'annotations Java 5
  - de la généricité Java 5
  - de l'API JPA (Java Persistence API)

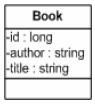


Entity Bean (développement)

- POJO avec getter/setter + annotations
  - Annotation **@Entity** : déclare une classe correspondant à un *entity bean* (EB)
  - Annotation **@Id** : définit une clé primaire
- Chaque classe de EB est mis en correspondance avec une table
  - par défaut table avec même nom que la classe
  - sauf si annotation **@Table(name="...")**
- 2 modes (exclusif) de définition des colonnes des tables
  - ***property-based access*** : on annote les méthodes *getter*
  - ***field-based access*** : on annote les attributs
    - par défaut colonne avec même nom que *field/property*
    - sauf si annotation **@Column(name="...")**

Entity Bean (développement)

```
@Entity
public class Book {
    private long id;
    private String author;
    private String title;
    public Book() {}
    public Book(String author, String title) {
        this.author = author;
        this.title = title; }
    @Id
    public long getId() { return id; }
    public void setId(long id) { this.id = id; }
    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; } }
```



Entity Bean (développement)

- Possibilité de définir des champs auto-incrémentés
  - Annotation @GeneratedValue

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
public long getId() { return id; }
```

  - GenerationType.AUTO : les numéros de séquence sont choisis automatiquement
  - GenerationType.SEQUENCE : un générateur de numéros de séquence est à fournir
- Possibilité pour un entity d'implémenter serializable
  - Permet de **détacher** l'entity pour le manipuler en dehors du container EJB
  - L'entity doit être **rattaché** pour être de nouveau persistant

Entity Bean (Gestionnaire d'entités)

- *Entity Manager*
  - assure la correspondance entre les objets Java et les tables relationnelles
    - point d'entrée principal dans le service de persistance
    - permet de faire persister les beans
    - permet d'exécuter des requêtes
  - accessible via une injection de dépendance
    - attribut de type javax.persistence.EntityManager
    - annoté par @PersistenceContext

*Void **persist**(Object o)*
*Void **remove**(Object o)*
*<T> T **find**(Class<T> aClass, Object o)*
*Query **createQuery**(String query)*
*<T> T **merge**(T t) // pour ratacher l'entity au container ejb (s'il a été sérializé)*

Entity Bean (Gestionnaire d'entités)

- Exemple
  - création de trois enregistrements dans la table des livres

```
@Stateless
public class MyBean implements MyBeanItf {
    @PersistenceContext
    private EntityManager em;
     public void init() {
         Book b1 = new Book("Honore de Balzac","Le Pere Goriot");
         Book b2 = new Book("Honore de Balzac","Les Chouans");
         Book b3 = new Book("Victor Hugo","Les Miserables");
         em.persist(b1);
         em.persist(b2);
         em.persist(b3);
     } }
```

  - de façon similaire em.remove(b2) retire l'enregistrement de la table

Entity Bean (Gestionnaire d'entités)

- Recherche par clé primaire
  - méthode find du gestionnaire d'entités

```
Book myBook = em.find(Book.class,12);
```

  - retourne null si la clé n'existe pas dans la table
  - IllegalArgumentException
    - si 1er paramètre n'est pas une classe d'EB
    - si 2ème paramètre ne correspond pas au type de la clé primaire

Entity Bean (Mise à jour)

- Mise à jour d'un entity après recherche dans un session bean

```
Book b = em.find(Book.class, 12);
b.setAuthor("tutu");
b.setTitle("mybook");
```

- Mise à jour d'un entity sérializé

```
// ex methode d'un stateless bean :
public void updateBook(Book b) { // b serializé par une servlet
    em.merge(b);
}
```

- Mise à jour à partir d'un DTO (voir plus loin)

Entity Bean (Gestionnaire d'entités)

- Recherche par requête
  - requêtes SELECT dans une syntaxe dite EJB-QL étendue
  - paramètres nommés (préfixés par :) pour configurer la requête

```
Query q = em.createQuery("select OBJECT(b) from Book b where b.author = :au");
String nom = "Honore de Balzac";
q.setParameter("au",nom);
List<Book> list = (List<Book>) q.getResultList();
```

  - méthode getSingleResult() pour récupérer un résultat unique
    - NonUniqueResultException en cas de non unicité

Entity Bean (Gestionnaire d'entités)

- Recherche par requête pré-compilée
  - création d'une requête nommée attachée à l'EB

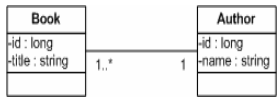
```
@Entity
@NamedQuery(name="allbooks",query="select OBJECT(b) from Book b")
public class Book { ... }

Query q = em.createNamedQuery("allbooks");
List<Book> list = (List<Book>) q.getResultList();
```

  - paramètres peuvent être spécifiés (voir transparent précédent)
  - plusieurs requêtes nommées peuvent être définies

```
@Entity
@NamedQueries(value={ @NamedQuery("q1","..."), @NamedQuery("q2","...") }
public class Book { ... }
```

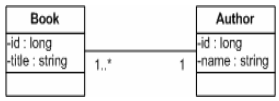

Entity Bean (Relation)

- 2 catégories principales : 1-n et n-n




```
@Entity
public class Author {
    private long id;
    private String name;
    private Collection<Book> books;
    public Author() { books = new ArrayList<Book>(); }
    public Author(String name) { this.name = name; }
    @OneToMany(mappedBy="author")
    public Collection<Book> getBooks() { return books; }
    public void setBooks(Collection<Book> books) { this.books=books; }
    ...
}
```

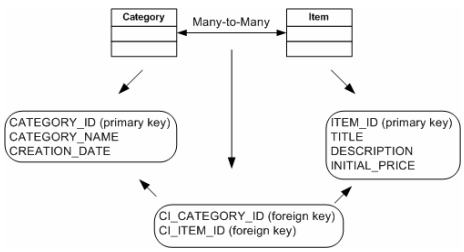

Entity Bean (Relation 1-n)




```
@Entity
public class Book {
    private long id;
    private Author author;
    private String title;
    public Book() {}
    public Book(Author author, String title) {
        this.author = author;
        this.title = title; }
    @ManyToOne
    public Author getAuthor() { return author; }
    public void setAuthor(Author author) { this.author = author; }
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }
    ...
}
```


Entity Bean (Relation n-n)

- Notion de table de jointure



Entity Bean (Relation n-n)

```
@Entity
public class Category {
    @Id
    @Column(name="CATEGORY_ID")
    protected long categoryId;
    @ManyToMany
    protected Set<Item> items;
```

```
@Entity
public class Item {
    @Id
    @Column(name="ITEM_ID")
    protected long itemId;
    @ManyToMany
    protected Set<Category> categories;
```


Entity Bean (annotations liées aux relations)

- **Mode de chargement d'une relation**
  - Attribut Fetch sur l'annotation d'une relation
  - EAGER ou LAZY
- **Persistance ou suppression en cascade pour une relation**
  - Attribut Cascade sur l'annotation d'une relation
  - CascadeType.ALL, CascadeType.PERSIST ...

ex :

```
@OneToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
```


Transactions

- Assure des propriétés **ACID** pour des transactions plates
- Exemple classique : un transfert bancaire (débit, crédit)
  - atomicité soit les 2 opérations s'effectuent complètement, soit aucune
  - cohérence le solde d'un compte ne doit jamais être négatif
  - isolation des transferts // doivent fournir le même résultat qu'en séq.
  - durabilité les soldes doivent être sauvegardés sur support stable
- Support complètement intégré au serveur EJB

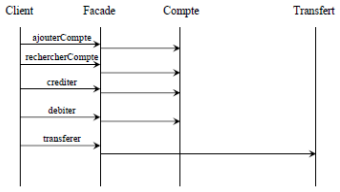
Design Pattern

- Problèmes de codage récurrents

  =>Design pattern (DP) : solutions reconnues d'organisation du code
- But
  - améliorer la clarté, la compréhension du code
  - mettre en avant des éléments d'architecture logicielle

Design Pattern : DP Session facade

- Pb : nombreuses dépendances entre les clients et les beans
- Solution : présenter aux clients **une seule interface façade** (*stateless session bean*)
  - Traitements effectués par la façade minimaux
  - Eventuellement plusieurs façades sur un même ensemble de beans

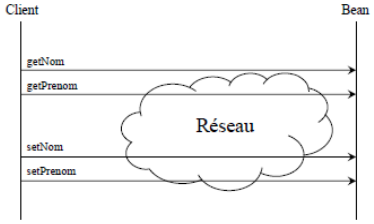


Design Patterns : DTO

Data Transfert Object

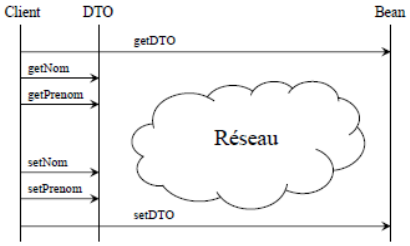
Aussi connu sous le terme : Value Object

Pb : nombreux échanges réseaux pour de simples get/set



Design Patterns : DTO

Solution : transmettre une instance par valeur



Conclusion

- Applications complexes "facile" à écrire
  - gestion de la persistance
  - gestion des transactions de manière déclarative
  - gestion intégré de la sécurité
  - gestion de la répartition
- Indépendance entre applications et plate-formes
- Mais
  - Evolution permanente ...