

# Framework Java EE

François Robert  
APSIDe  
TOP

1

## Le plan !

- Java EE, definition
- Les applications d'entreprise
- Entités
- Persistance (JPA)
- Entreprise Java Bean
- Java Server Faces

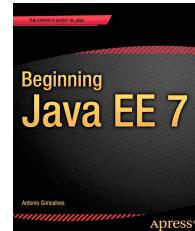
3

## De quoi parle t'on ?

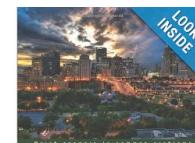
- Java EE (version 7)
- Architecture d'entreprise
- Développement

2

## Références



Beginning Java EE 7  
Antonio Goncalves  
aPress 2013



EJB 3.1 Cookbook  
Richard M. Reese  
Packt Publishing 2011



Java EE 6 & Glassfish 3  
Antonio Goncalves  
Pearson 2010

4

# Java EE 7

Un standard

C'est un ensemble de plus de 30 spécifications  
Coiffées par la JSR 342

Et réparties dans 5 domaines

Web Application Technologies   Enterprise Application Technologies   Web Services Technologies  
Management and Security Technologies   Java EE-related Specs in Java SE  
<http://www.oracle.com/technetwork/java/javase/tech/index.html>

5

# Java EE 7

Nouveautés

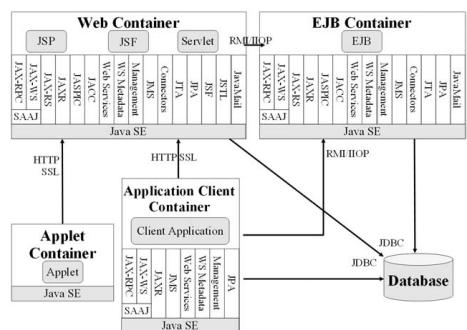
- JSON-P
- Batch
- Websocket
- Multithread sur serveur

Mise à jour

- JAX-RS
- Bean validation
- Interceptor
- CDI
- etc...

6

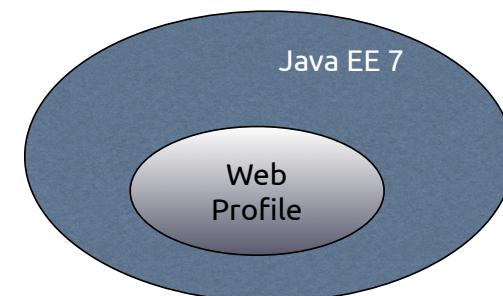
# Java EE 7



7

# Java EE profil web

Version légère



8

## Java EE profil web

Version légère

- Servlet 3.1
- JavaServer Pages (JSP) 2.2
- Expression Language (EL) 3.0
- Debugging Support for Other Languages (JSR-45) 1.0
- Standard Tag Library for JavaServer Pages (JSTL) 1.2
- JavaServer Faces (JSF) 2.2
- Java API for RESTful Web Services (JAX-RS) 2.0
- Common Annotations for the Java Platform (JSR-250) 1.1
- Enterprise JavaBeans (EJB) 3.2 **Lite**
- Java Transaction API (JTA) 1.2
- Java Persistence API (JPA) 2.1
- Bean Validation 1.1
- Managed Beans 1.0
- Interceptors 1.1
- Contexts and Dependency Injection for the Java EE Platform 1.1
- Dependency Injection for Java 1.0

9

## Java EE profil web

EJB Lite

- Support des EJB de type session (stateless, stateful, et singleton)
- Support des EJB avec interface local ou sans interface
- L'injection
- Les intercepteurs
- La sécurité et les transactions (Container Managed Transactions et Bean Managed Transactions)

10

## Java EE profil web

ce qu'il y a en plus dans EJB <sup>8full<sup>8</sup></sup>

- Les EJB 2.x
- L'invocation via RMI/IOP
- Les session bean avec interface Remote
- Les EJB de type MDB
- Le support des endpoints pour les services web
- Le service Timer
- CMP / BMP

11

## Architecture d'entreprise

Architecture distribuée

Architecture multi-tiers

Architecture SOA

Redondance & Cluster

12

## Architecture distribuée

- Architecture dont les composants sont répartis entre plusieurs systèmes logiques ou physiques
- Les différents composants communiquent à travers le réseau

13

## Architecture multitiers

Architecture dont les composants sont séparés selon leurs fonctions au sein de l'application

Architecture 3 tiers

Architecture 4 tiers

14

## Architecture multitiers

Architecture 3 tiers

- IHM (présentation)
- Métier
- Données (SGBD, Fichier, LDAP...)

15

## Architecture multitiers

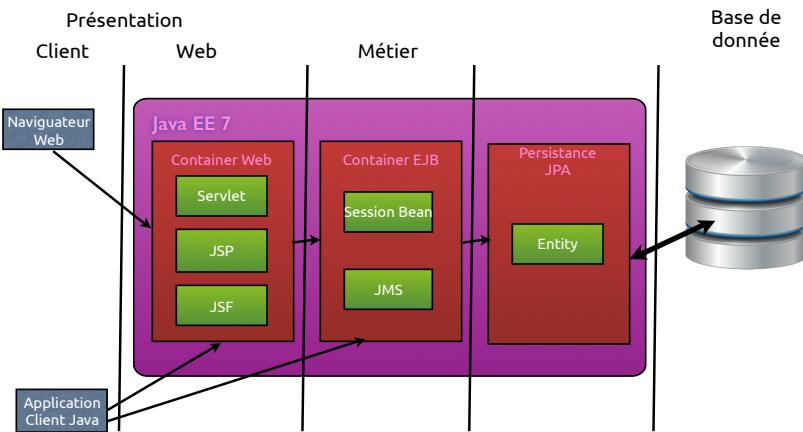
Architecture 4 tiers

- IHM (présentation)
- Métier
- Persistance
- Données (SGBD)

16

## Architecture multitiers

Architecture 4 tiers



17

## Architecture SOA

Services Oriented Architecture

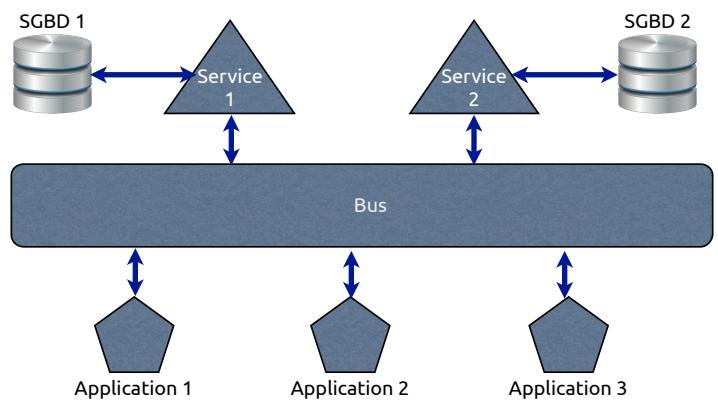
Architecture avec découpage en composant présentant un sens au niveau métier. Chaque composant regroupe :

Une partie métier

Un accès aux données

## Architecture SOA

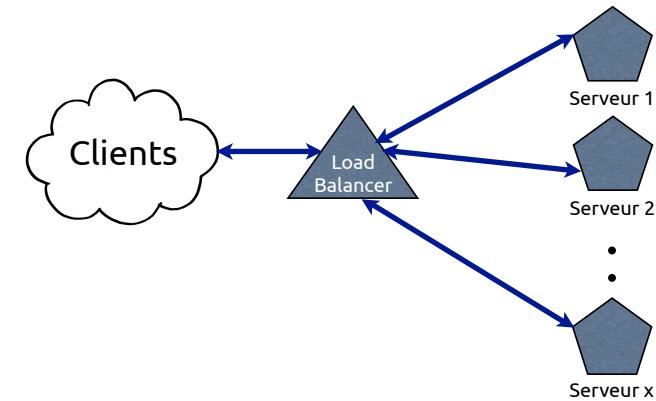
Services Oriented Architecture



19

## Redondance & Cluster

Réplication des ressources et distribution des accès afin de conserver des performances au dessus du nombre



20

## Les serveurs d'application

- hébergeant et exécutant le code des applications
- fournissant des services standards permettant aux développeurs de se concentrer sur la partie métier de l'application
- proposant un ensemble de composants :
  - Serveur web statique
  - Moteur de servlet
  - Container d'EJBs
  - Serveur de messagerie
  - ...

21

## Les serveurs d'application

Fournissent aussi des services

- Sécurité
- Nomage (JNDI)
- Transaction
- Accès aux données
- Répartition de charge (pool, Cluster)

22

## Les outils du cours

- Un éditeur de code (Netbeans ou IntelliJ)
- Maven
- Tests (TDD ?)
- Base de données (Derby)
- Serveur d'entreprise (Glassfish)

23

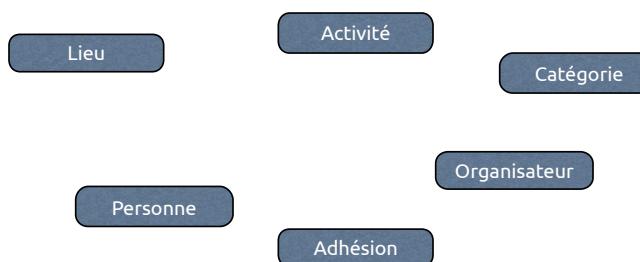
## Runtime vs Embedded container

- Runtime EJB Container
  - Glassfish, JBoss, etc...
  - VM séparé
  - Bonne tenue en production
  - «not good for» développement et test.
- Embedded EJB Container
  - Unique VM
  - Ajoute un jar au classpath
  - Peut être utilisé dans JavaSE, batch, conteneur web

24

## Les entités

25

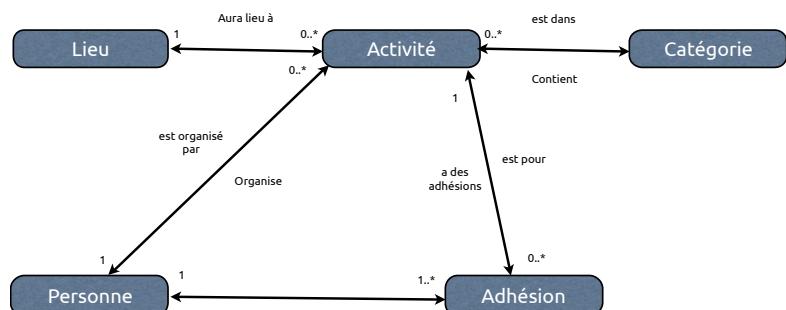


## Modèle du domaine

- Image conceptuelle du problème que l'application essaie de résoudre
- Ensemble d'objets (physique ou concept) et de relations ou associations entre ces objets

26

## Exemple



27

28

## Acteurs du modèle

- Objets : traduit par les classes java dont les attributs seront les données (Personne contient nom, prénom,...)
- Relations : traduit par des classes ayant des références sur d'autre. Ces relations peuvent être unidirectionnelles ou bidirectionnelles
- Multiplicité ou cardinalité : les relations ne sont pas forcément un pour un , on trouve donc one-to-one, one-to-many, many-to-one et many-to-many

29

## Objet du domaine

```
public class Activity {  
    private Long id;  
    private String name;  
    private String description;  
    private Date dateEvent;  
    private Date dateCreation;  
  
    public Activity() {  
    }  
    ... getter et setter  
}
```

31

## Modèle riche ou anémique

- un modèle anémique est un modèle qui n'encapsule que les données en se contentant de mapper directement les classes sur les tables dans une relation un pour un.
- un modèle riche est un modèle qui encapsule les données mais aussi le comportement en utilisant des notions comme l'héritage, le polymorphisme et l'encapsulation
- Un modèle anémique est plus simple à réaliser, mais un modèle riche est plus pratique pour la programmation de l'application

30

## Entité

```
@Entity  
public class Activity {  
    @Id  
    private Long id;  
    private String name;  
    private String description;  
    private Date dateEvent;  
    private Date dateCreation;  
    @Transient  
    private int registeredPersonCount;  
  
    public Activity() {  
    }  
}
```

32

## Anatomie d<sup>e</sup>une entité

- Un simple POJO
- Une annotation @Entity
- Une clé primaire @Id
- Un constructeur sans arguments
- Une classe non finale
- Attention, des enumérés ou interface ne peuvent être des entités

33

## Entité (bis)

```
@Entity  
public class Activity {  
    @Id  
    private Long id;  
    private String name;  
    private String description;  
    private Date dateEvent;  
    private Date dateCreation;  
    @Transient  
    private int registeredPersonCount;  
  
    public Activity() {  
    }  
}
```

35

## Entité ou Pojo ?

- Une entité est un POJO
- Mais un POJO n'est pas une entité
- Les entités sont managées par EntityManager
  - Persiste les entités
  - Leur état est synchronisé avec la base
- Quand ils ne sont pas managés, ce sont de simple POJO

34

## Les annotations

- Les annotations peuvent être mises sur l'attribut lui même (Field-based persistence) ou sur le getter de l'attribut (Property-based persistence)
- Le choix exclusif
- L'utilisation du property-based respecte mieux l'encapsulation pronée en POO

36

# Type de donnée

Types	Exemple
Primitives	int, double, long
Wrapper de primitive	Integer, Double, Long
Chaine	String
Type serialisable	BigInteger, java.sql.Date
Classe serialisable (utilisateur)	Implementant java.io.Serializable
Tableau	Byte[]
Type enuméré	Tous les énumérés
Collection (entité)	Set
Type rattaché	class annotées @Embeddable

37

# Les annotations de mapping

- `@Table`
- `@Column`
- `@Enumerated`
- `@Lob`
- `@Temporal`
- `@Embeddable`
- `@Transient`

38

## `@Table`

```
@Target({TYPE}) @Retention(RUNTIME)
public @interface Table {
    String name() default AB;
    String catalog() default AB;
    String schema() default AB;
    UniqueConstraint[] uniqueConstraints()
default {};
}
```

39

## `@Table`

- **Optionnelle**, si elle est omise la table a le nom de la classe, dans le schéma par défaut
- **name** : permet de préciser le nom de la table
- **schema** : permet de préciser le schéma dans la base
- **catalog** : permet de préciser le catalogue dans la base
- **uniqueConstraints** permet de préciser une contrainte d'unicité sur une ou des colonnes en cas d'utilisation de la génération de schéma

40

## @Column

```
Target({METHOD, FIELD}) @Retention(RUNTIME)
public @interface Column {
    String name() default FG;
    boolean unique() default false;
    boolean nullable() default true;
    boolean insertable() default true;
    boolean updatable() default true;
    String columnDefinition() default FG;
    String table() default FG;
    int length() default 255;
    int precision() default 0;
    int scale() default 0;
}
```

41

## @Column

- **optionnelle**, si omise, la colonne à le nom de l<sup>e</sup>attribut ou propriété
- **name** : nom de la colonne
- **table** : en cas de mapping sur deux tables
- **insertable, updatable** : permet d<sup>e</sup>exclure le champs des requêtes INSERT ou UPDATE, utiliser pour les champs en lecture seul comme les clés primaires générées
- Les autres sont principalement pour la création de table :
  - nullable : si la colonne supporte les valeurs nulles
  - unique : si la colonne à une contrainte d<sup>e</sup>unicité
  - length : taille de la colonne
  - precision, scale : pour les décimales
  - columnDefinition : Spécifie le SQL exact de la création

42

## @Enumerated

```
@Target({METHOD, FIELD}) @Retention(RUNTIME)
public @interface Enumerated {
    EnumType value() default ORDINAL;
}
public enum EnumType{
    ORDINAL,
    STRING }
```

43

## @Enumerated

- Si on utilise ORDINAL, les valeurs sauvegardées en base sont de 0 à n selon le nombre d<sup>e</sup>éléments dans l<sup>e</sup>enumération
- Si c<sup>e</sup>est STRING, c<sup>e</sup>est le nom de la valeur de l<sup>e</sup>enuméré qui est stocké
- Si l<sup>e</sup>annotation est omise, l<sup>e</sup>attribut ou propriété est sauvegardée de façon ordinal

44

## @Lob

```
@Target({METHOD, FIELD}) @Retention(RUNTIME)
public @interface Lob {
}
```

45

## @Lob

- permet de désigner un champs comme BLOB ou CLOB
- Si le champs est un char[] ou String c'est un CLOB, sinon c'est BLOB
- Généralement utilisé en conjonction avec @Basic (fetch=FetchType.LAZY) permettant d'en différer le chargement

46

## @Temporal

```
@Target({METHOD, FIELD}) @Retention(RUNTIME)
public @interface Temporal {
    TemporalType value();
}
public enum TemporalType{
    DATE,
    TIME,
    TIMESTAMP
}
```

47

## @Temporal

- Cette annotation est redondante avec les types java.sql.Date, java.sql.Time, java.sql.Timestamp
- Permet de préciser si on veut persister les types java.util.Date et java.util.Calendar sur des champs de type DATE, TIME ou TIMESTAMP
- Sans annotation le persistence provider utilise le type TIMESTAMP

48

## @Transient

```
@Target({ElementType.METHOD, ElementType.FIELD})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface Transient {  
}
```

49

## @Transient

- Permet de ne pas persister une donnée qui pourrait être calculée

```
@Entity  
@Table(name = "PERSON")  
public class Person {  
    @Id  
    @GeneratedValue(generator = "USER_SEQ")  
    private Long id;  
    @Column(nullable = false)  
    private String name;  
    @Temporal(TemporalType.DATE)  
    private Date dateOfBirth;  
    @Transient  
    private int age;  
  
    public Person() {  
    }  
}
```

50

## @SecondaryTables

Il est parfois nécessaire de stocker les valeurs dans différentes tables

```
@Target({TYPE})  
@Retention(RUNTIME)  
public @interface SecondaryTables {  
    javax.persistence.SecondaryTable[] value();  
}
```

51

## @ SecondaryTables

```
@Entity  
@Table(name = "PERSON")  
@SecondaryTables({  
    @SecondaryTable(name = "CITY"),  
    @SecondaryTable(name = "COUNTRY")  
})  
public class Person {  
    @Id  
    @GeneratedValue(generator = "USER_SEQ")  
    private Long id;  
    @Column(nullable = false)  
    private String name;  
    private String Street1;  
    private String Street2;  
    @Column(table = "CITY")  
    private String city;  
    @Column(table = "COUNTRY")  
    private String country;
```

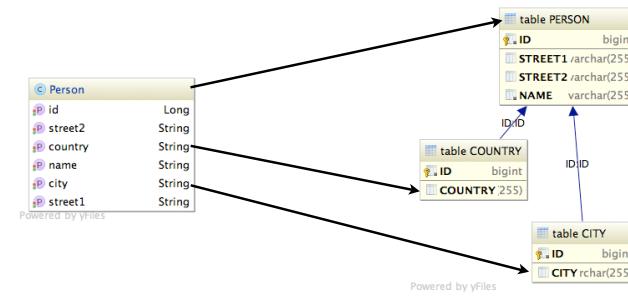
52

## @SecondaryTables

```
@Entity  
@Table(name = "PERSON")  
@SecondaryTables({  
    @SecondaryTable(name =  
    "CITY", pkJoinColumns=@PrimaryKeyJoinColumn(name= "ID")),  
    @SecondaryTable(name =  
    "COUNTRY", pkJoinColumns=@PrimaryKeyJoinColumn(name= "ID"))  
})  
public class Person {  
    @Id  
    @GeneratedValue(generator = "USER_SEQ")  
    private Long id;  
    @Column(nullable = false)  
    private String name;  
    private String Street1;  
    private String Street2;  
    @Column(table = "CITY")  
    private String city;  
    @Column(table = "COUNTRY")  
    private String country;  
}
```

53

## @ SecondaryTables



54

## @SecondaryTables

- Permet d'associer des tables secondaires
- soit une, utilisation directe de @SecondaryTable
- soit plusieurs (@SecondaryTables & @SecondaryTable)
- Risque non négligeable sur les performances

55

## Les classes «embarquées» Embedded Class

- L'inverse des tables secondaires
- Entité et embeddedClass == 1 seule table
- La classe embarquée n'est pas une entité
- Annoté @Embeddable
- N'a pas d'@Id
- Utilisable dans plusieurs entités

56

## EmbeddedClass

### Exemple

```
@Entity  
@Table(name = "PERSON")  
public class Person {  
    @Id  
    @GeneratedValue(generator = "USER_SEQ")  
    private Long id;  
    @Column(nullable = false)  
    private String name;  
    @Embedded  
    private Address address;  
  
    public Person() {}
```

57

## EmbeddedClass

### Exemple - réutilisation

```
@Entity  
@Table(name = "CLUB")  
public class Club {  
  
    @Id  
    private long id;  
    @Column(nullable = false, length = 200)  
    private String name;  
    @Column(nullable = true)  
    private String motto;  
    @Embedded  
    private Address address;  
  
    public Club() {}
```

59

## EmbeddedClass

### Exemple

```
@Embeddable  
public class Address {  
  
    private String street1;  
    private String street2;  
    private String zipCode;  
    private String city;  
    private String country;  
  
    public Address() {}  
    @Column(length=50, nullable=false)  
    public getStreet1() {  
        return this.street1;  
    }
```

58

## Identifiants

- L'identifiant d'une entité est la clé primaire
- Ce peut être un champs de type primitif, un ensemble de champs ou un objet
- Il existe 3 moyens de le définir
  - `@Id`
  - `@EmbeddedId`
  - `@IdClass`

60

## @Id

- Permet de marquer un champs comme étant l'identifiant de l'objet
- Peut être un type primitif, un wrapper de primitif ou un Serializable (java.lang.String, java.util.Date, java.sql.Date)
- Il est recommandé d'éviter les types float, double et leur wrapper, à cause de la précision de la base
- De même il faut éviter le type TimeStamp

61

## @EmbeddedId

- Utilisation d'un objet embarqué comme Identifiant
- La classe doit redéfinir hashCode et equals
- Permet une maintenance du code simple
- Modifie le modèle du domaine

62

## @EmbeddedId

```
@Entity
public class Category {
    @EmbeddedId
    private CategoryPK categoryPK;

    public Category() {
    }
    ...

    @Embeddedable
    public class CategoryPK implements Serializable {
        private String Name;
        private Date createDate;

        public CategoryPK() {}
        @Override public boolean equals(Object o) {}
        @Override public int hashCode() {}
    }
}
```

63

## @IdClass

- Permet d'utiliser plus d'une annotation @Id dans une Entité
- Nécessite la définition d'une classe supplémentaire qui implémente java.io.Serializable et fournit des implémentations correctes de **equals** et **hashCode**
- Peut poser des problèmes de maintenance mais maintient l'état du modèle du domaine

64

## @IdClass

```
@Entity  
@IdClass(CategoryPK.class)  
public class Category implements Serializable {  
    @Id  
    private String Name;  
    @Id  
    private Date createDate;  
  
    public Category() {}  
    ...  
}  
  
public class CategoryPK implements Serializable {  
    private String Name;  
    private Date createDate;  
  
    public CategoryPK() {}  
    @Override public boolean equals(Object o) {}  
    @Override public int hashCode() {}  
}
```

65

## Identité de colonne

- Supporté par certaine base comme MS SQL Server
- Quand on l'utilise, la valeur générée peut ne pas être disponible avant sauvegarde en base

```
@Id  
@GeneratedValue(strategy=GenerationType.IDENTITY)  
@Column(name=>USER_ID)  
private long id;
```

67

## Clé primaire

- 2 types de clé primaires
  - Clé naturelle (clé INSEE), constituée de donnée(s) métier
  - Clé substituée (Surrogate ou Substitute)
- Les clés substituées sont préférables aux clés composées
- 3 manières de les générer
  - Identity de colonne
  - Séquence SGBD
  - Table de séquence

66

## Séquence SGBD

Nécessite la création d'une séquence en base de données et d'un SequenceGenerator (pas nécessairement dans le même entity)

```
@Id  
@SequenceGenerator(name="USER_SEQUENCE_GENERATOR",  
    sequenceName="USER_SEQUENCE", initialValue=1,  
    allocationSize=10)  
@GeneratedValue(strategy=GenerationType.SEQUENCE,  
    generator="USER_SEQUENCE_GENERATOR")  
@Column(name="USER_ID")  
private long id;
```

68

## @SequenceGenerator

```
@Target({TYPE, ElementType.METHOD, ElementType.FIELD})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface SequenceGenerator {  
    java.lang.String name();  
    java.lang.String sequenceName() default "";  
    java.lang.String catalog() default "";  
    java.lang.String schema() default "";  
    int initialValue() default 1;  
    int allocationSize() default 50;  
}
```

69

## Table de séquence

- Nécessite la création d'une table respectant un modèle précis

```
@Id  
@TableGenerator(name="USER_TABLE_GENERATOR",  
    table="SEQUENCE_GENERATOR_TABLE",  
    pkColumnName= "SEQUENCE_NAME",  
    valueColumnName= "SEQUENCE_VALUE",  
    pkColumnValue= "USER_SEQUENCE")  
@GeneratedValue(strategy=GenerationType.TABLE,  
    generator= "USER_TABLE_GENERATOR")  
@Column(name= "USER_ID")  
private long id;
```

70

## Table de séquence

- La table SEQUENCE\_GENERATOR\_TABLE

SEQUENCE_NAME	SEQUENCE_VALUE
USER_SEQUENCE	1
«autre séquence»	23

71

## @TableGenerator

```
@Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface TableGenerator {  
    java.lang.String name();  
    java.lang.String table() default "";  
    java.lang.String catalog() default "";  
    java.lang.String schema() default "";  
    java.lang.String pkColumnName() default "";  
    java.lang.String valueColumnName() default "";  
    java.lang.String pkColumnValue() default "";  
    int initialValue() default 0;  
    int allocationSize() default 50;  
    javax.persistence.UniqueConstraint[] uniqueConstraints() default {};  
    javax.persistence.Index[] indexes() default {};  
}
```

72

# Les relations

- Les entités peuvent être reliées à d'autres
- La cardinalité est gérée par annotations
  - @OneToOne (unidirectionnelle, défaut)
  - @OneToMany
  - @ManyToOne
  - @ManyToMany
- Elles peuvent être unidirectionnelles ou bidirectionnelles
- Les relations bidirectionnelles ont «Owning side» & «Inverse side»

73

## @OneToOne

```
@Entity  
@Table(name = "PERSON")  
public class Person {  
    @Id  
    @GeneratedValue(generator = "USER_SEQ")  
    private Long id;  
    @Column(nullable = false)  
    private String name;  
    @OneToOne  
    private Address address;  
  
    @Entity  
    public class Address {  
        @Id  
        @GeneratedValue(generator = "ADDRESS_SEQ")  
        private Long id;  
        private String Street1;  
        private String Street2;  
        private String zipCode;  
        private String city;  
        private String country;
```

75

## @OneToOne

```
@Entity  
@Table(name = "PERSON")  
public class Person {  
    @Id  
    @GeneratedValue(generator = "USER_SEQ")  
    private Long id;  
    @Column(nullable = false)  
    private String name;  
    private Address address;  
  
    @Entity  
    public class Address {  
        @Id  
        @GeneratedValue(generator = "ADDRESS_SEQ")  
        private Long id;  
        private String Street1;  
        private String Street2;  
        private String zipCode;  
        private String city;  
        private String country;
```

74

## @OneToOne

```
@Entity  
@Table(name = "PERSON")  
public class Person {  
    @Id  
    @GeneratedValue(generator = "USER_SEQ")  
    private Long id;  
    @Column(nullable = false)  
    private String name;  
    @OneToOne  
    @JoinColumn(name = "ADDRESS_FK", nullable = false)  
    private Address address;  
  
    public Person() {  
    }
```

76

# @OneToOne

```
@Target({ElementType.METHOD, ElementType.FIELD})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface OneToOne {  
    java.lang.Class targetEntity() default void.class;  
    javax.persistence.CascadeType[] cascade() default {};  
    javax.persistence.FetchType fetch() default FetchType.EAGER;  
    boolean optional() default true;  
    java.lang.String mappedBy() default "";  
    boolean orphanRemoval() default false;  
}
```

77

# @OneToOne

Bidirectionnel

```
@Entity  
@Table(name = "PERSON")  
public class Person {  
    @Id  
    @GeneratedValue(generator = "USER_SEQ")  
    private Long id;  
    @Column(nullable = false)  
    private String name;  
    @OneToOne  
    private Address address;  
}  
  
@Entity  
public class Address {  
    @Id  
    @GeneratedValue(generator = "ADDRESS_SEQ")  
    private Long id;  
    private String Street1;  
    ....  
    private String country;  
    @OneToOne(mappedBy = "address", optional = false)  
    private Person person;
```

79

# @OneToOne

- target Entity : type de classe pointée par la relation. Par défaut le “persistence provider“ utilise celui de l’attribut ou la valeur de retour du getter
- cascade : ce qu’il advient de la donnée pointée en cas de modification ou suppression de la relation
- fetch : façon dont les données sont peuplées
- optionnal : si la donnée référencée peut être nulle
- mappedBy : pour déclarer une relation bidirectionnelle, permet de positionner le coté “propriétaire“ de la relation

78

# mapping OneToOne

- Deux cas se présentent selon la table où se trouve la clé étrangère.
- Si A possède une référence sur B :
  - Cas 1 : la clé primaire de B est clé étrangère dans la table A
  - Cas 2 : la clé primaire de A est clé étrangère dans la table B

80

# @OneToOne

## Cas 1

```
@Entity  
@Table(name = "PERSON")  
public class Person {  
    @Id  
    @GeneratedValue(generator = "USER_SEQ")  
    private Long id;  
    @Column(nullable = false)  
    private String name;  
    @OneToOne  
    @JoinColumn(name = "ADDRESS_FK", nullable = false)  
    private Address address;  
  
    public Person() {  
    }
```

81

# @OneToOne

## Cas 2

```
@Entity  
@Table(name = "PERSON")  
public class Person {  
    @Id  
    @GeneratedValue(generator = "USER_SEQ")  
    private Long id;  
    @Column(nullable = false)  
    private String name;  
    @OneToOne  
    @PrimaryKeyJoinColumn(name="ID",referencedColumnName="ADRESS_PERSONN_ID")  
    private Address address;  
  
    public Person() {  
    }
```

83

# @ JoinColumn

```
@Target({ElementType.METHOD, ElementType.FIELD})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface JoinColumn {  
    java.lang.String name() default "";  
    java.lang.String referencedColumnName() default "";  
    boolean unique() default false;  
    boolean nullable() default true;  
    boolean insertable() default true;  
    boolean updatable() default true;  
    java.lang.String columnDefinition() default "";  
    java.lang.String table() default "";  
}
```

82

# @PrimaryKeyJoinColumn

```
@Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface PrimaryKeyJoinColumn {  
    java.lang.String name() default "";  
    java.lang.String referencedColumnName() default "";  
    java.lang.String columnDefinition() default "";  
}
```

84

# Les collections

```
@Entity  
@Table(name = "PERSON")  
public class Person {  
    @Id  
    @GeneratedValue(generator = "PERSON_SEQ")  
    private Long id;  
    .....  
    private Set<Adhesion> adhesionSet;  
  
    public Person() {  
    }  
  
    @Entity  
    @Table(name = "ADHESION")  
    public class Adhesion {  
        @Id  
        @GeneratedValue(generator = "ADHESION_SEQ")  
        private Long id;  
        @Temporal(TemporalType.TIMESTAMP)  
        private Date date;
```

85

# @OneToMany

```
@Entity  
@Table(name = "PERSON")  
public class Person {  
    @Id  
    @GeneratedValue(generator = "PERSON_SEQ")  
    private Long id;  
    .....  
    @OneToMany  
    @JoinTable(name = "ADHESIONJOINPERSON", joinColumns =  
    @JoinColumn(name = "PERSON_FK"), inverseJoinColumns =  
    @JoinColumn(name = "ADHESION_FK"))  
    private Set<Adhesion> adhesionSet;  
  
    public Person() {  
    }
```

86

# @OneToMany

Sans table de jointure

```
@Entity  
@Table(name = "PERSON")  
public class Person {  
    @Id  
    @GeneratedValue(generator = "PERSON_SEQ")  
    private Long id;  
    .....  
    @OneToMany(mappedBy = "person")  
    private Set<Adhesion> adhesionSet;  
    public Person() {  
    }  
  
    @Entity  
    @Table(name = "ADHESION")  
    public class Adhesion {  
        @Id  
        @GeneratedValue(generator = "ADHESION_SEQ")  
        private Long id;  
        @Temporal(TemporalType.TIMESTAMP)  
        private Date date;  
        @ManyToOne  
        private Person person;  
  
        public Adhesion() {  
        }
```

87

# @OneToMany & @ManyToOne

- Le One-to-many unidirectionnel n'est pas supporté dans la spécification. Bien que des “persistence provider” le supportent, il faut utiliser une table de relation avec `@JoinTable`
- Pas de `mappedBy` dans le `@ManyToOne` car c'est toujours cette table qui contient la clé étrangère
- Le `JoinColumn` peut pointer deux colonnes de la même table

88

# @ManyToMany

BiDirectionnel

```
@Entity  
@Table(name = "CATEGORY")  
public class Category {  
    @Id  
    @GeneratedValue(generator = "CATEGORY_SEQ")  
    @Column(name = "CATEGORY_ID")  
    private Long id;  
    @Column(nullable = false)  
    private String name;  
    @ManyToMany  
    private List<Event> eventList;  
  
@Entity  
@Table(name = "EVENTS")  
public class Event {  
    @Id  
    @GeneratedValue(generator = "EVENT_SEQ")  
    @Column(name = "EVENT_ID")  
    private Long id;  
    @Column(nullable = false)  
    private String name;  
    @ManyToMany  
    private List<Category> category;
```

89

# @ManyToMany

Unidirectionnel

```
@Entity  
@Table(name = "CATEGORY")  
public class Category {  
    @Id  
    @GeneratedValue(generator = "CATEGORY_SEQ")  
    @Column(name = "CATEGORY_ID")  
    private Long id;  
    @ManyToMany  
    @JoinTable(  
        name = "CATEGORY_EVENT",  
        joinColumns = @JoinColumn(  
            name = "CE_CATEGORY_ID",  
            referencedColumnName = "CATEGORY_ID"  
        ),  
        inverseJoinColumns = @JoinColumn(  
            name = "CE_EVENT_ID",  
            referencedColumnName = "EVENT_ID"  
        )  
    )  
    private List<Event> eventList;
```

90

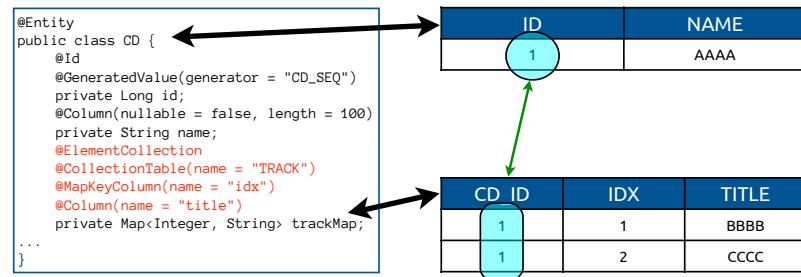
# Relation et FETCH

Relation	Stratégie FETCH
@OneToOne	EAGER
@ManyToOne	EAGER
@OneToMany	LAZY
@ManyToMany	LAZY

91

# Et les Map ?...

Petit rappel, une map = liaison clé/valeur

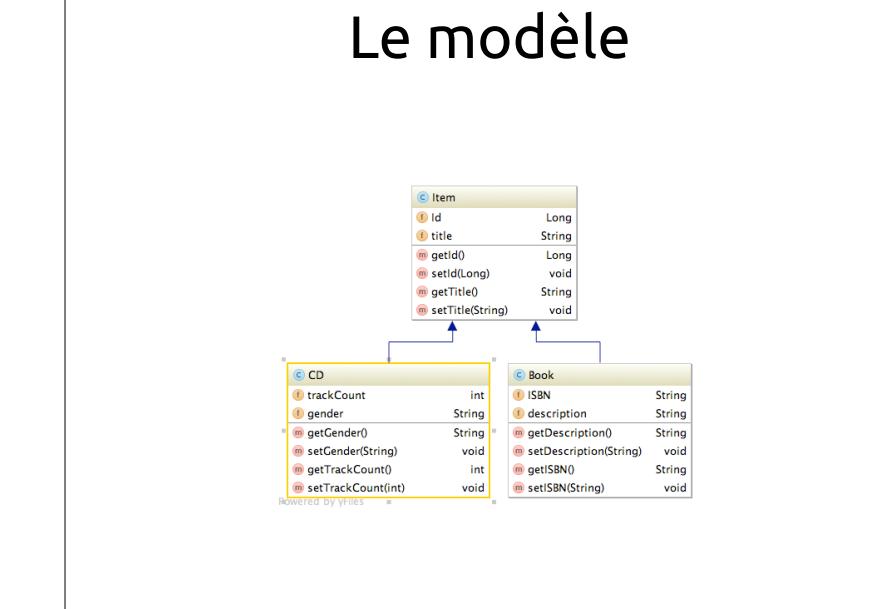


92

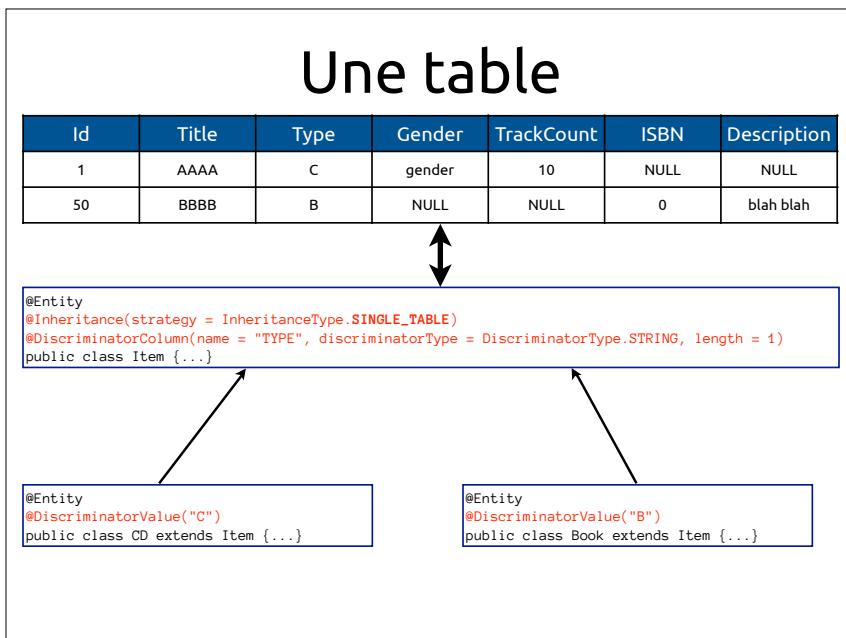
# L'héritage

- Le concept d'héritage n'existe pas dans le monde relationnel
- La transposition peut s'effectuer selon 3 stratégies
  - Une seule table
  - Tables jointes
  - Une table par classe

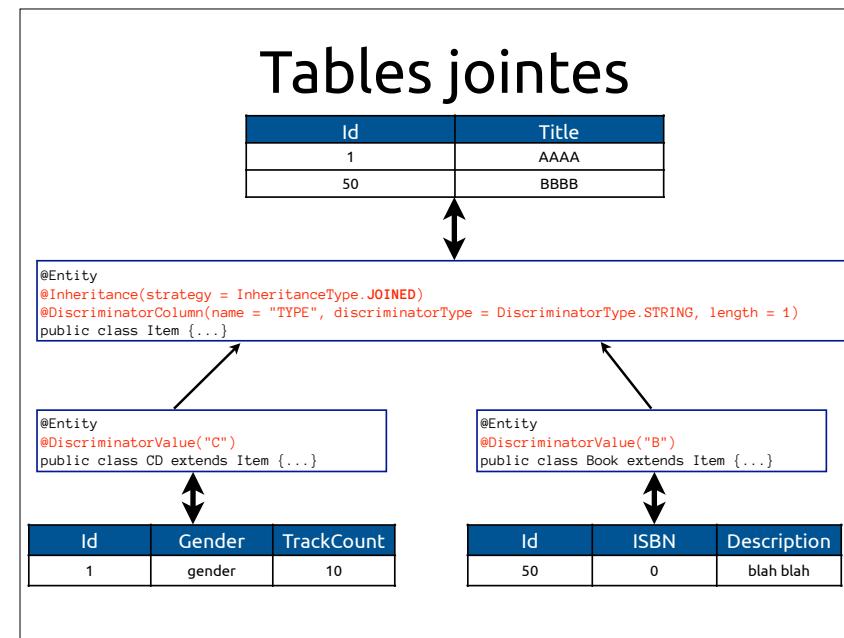
93



94

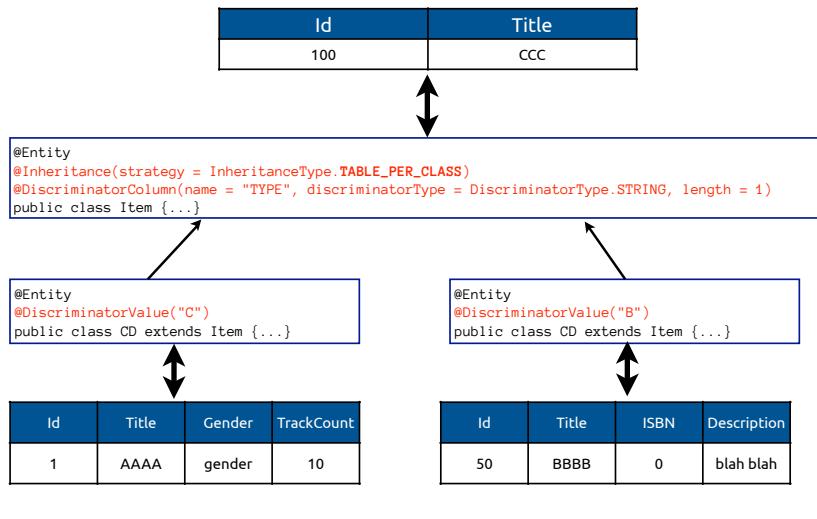


95



96

## Une table par classe

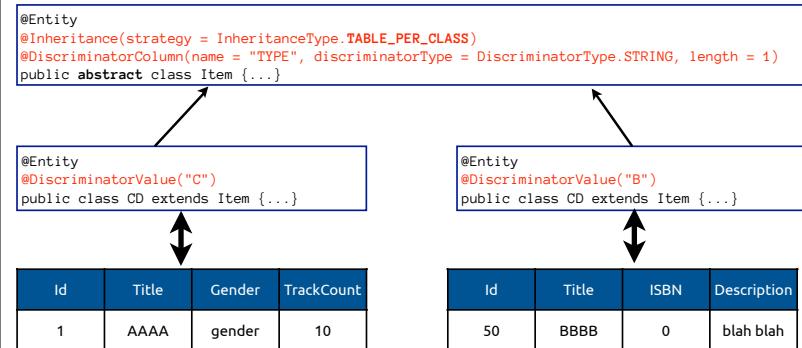


97

## Une table par classe



Classe parente abstraite, la table Item ne sera pas créée.



98

## L'héritage en bref

Fonction	Une table	Tables jointes	Une table par classe
Tables	*une seule table pour toute la hiérarchie *colonnes obligatoires peuvent être nullables *la table est modifiée en cas d'ajout de sous classes	*une pour la classe parent et une par sous-classe *les tables sont normalisées	une table par classes concrète dans la hiérarchie
utilise colonne discriminante	OUI	OUI	NON
SQL généré pour récupérer un entity	Select simple	Select avec Jointures	Select complexe (1 select par classe et union)
SQL généré pour insert et update	simple	Multiple en cascade	simple, par classe
Relation polymorphe	Bon	Bon	Mauvais
requête polymorphe	Bon	Bon	Mauvais
Implémentation JPA	Obligatoire	Obligatoire	Optionnel

99

## JPA



### Exercice 3 - Système de blog simpliste... (I)

- Le blog a deux type d'utilisateurs : redacteur et modérateur (ce choix est exclusif), décrit par l'enum UserRules.
- Un utilisateur possède un système d'identification (login/password) enregistré dans la table USER.
- Un utilisateur possède une adresse, stockée dans la table USER\_ADDRESS et qui porte la relation à l'utilisateur dans la table USER\_ADDRESS.
- Un rédacteur peut écrire 1 ou plusieurs articles.
- Un modérateur a différent niveau de modération (un seul possible).

100

# JPA



## Exercice 3 - Système de blog simpliste... (2)

- La table des utilisateur : USER
- Déterminer le choix de la gestion de l'héritage en prenant en compte la contrainte d'espace disque utilisé. Expliquez votre choix.
- Gérer les relation OneToOne, OneToMany et ManyToMany.
- Réalisez toutes ces opération dans la classe de test de l'exercice 3 (sauvegardez vos objets et vérifiez dans la base leur persistance).

101

# Persistance & JPA

102

## Persistance

- Les applications regroupent logique métier, IHM et Données
- Les données doivent être persistées
  - Fichier
  - base de données
- JPA permet de relier des objets à des bases relationnelles

103

## Pourquoi persister

- Les objets ne sont accessibles que lorsque la JVM fonctionne
- Si la JVM s'arrête, le GC nettoie la mémoire et les objets s'y trouvant
- Quelques objets doivent être persistés
- Stockés de manière permanente sur support magnétique, mémoire flash

104

## Comment persister en java

- Serialisation
- JDBC
- Object Relational Mapping (ORM)
  - JPA (EclipseLink )
  - Hibernate
  - Toplink

105

## EntityManager

- Il gère le cycle de vie des entités
- C'est la pièce centrale de JPA
- Il gère les requêtes sur les entités
- Il est garant des opérations CRUD

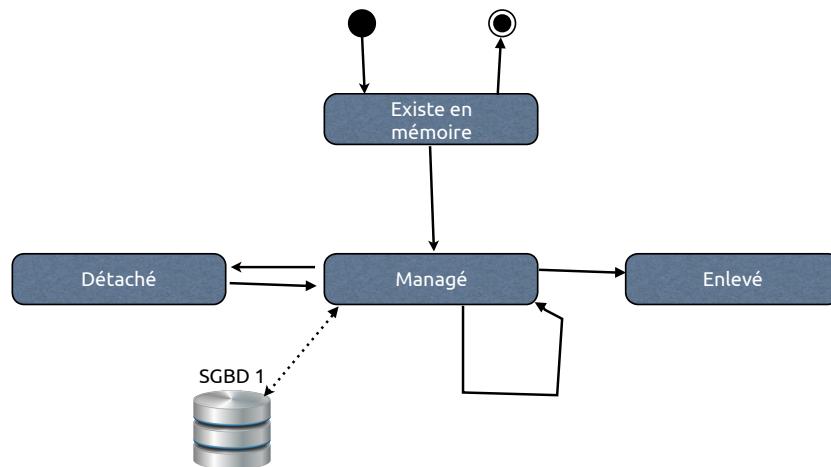
107

## Les avantages de 2.1

- ORM : Mapping des objets
- Entity Manager...
- Langage de requête basé sur l'ORM (JPQL)
- Mécanisme de lock basé sur JTA
- Callback et listeners pour injecter une logique métier dans le cycle de vie.

106

## Cycle de vie



108

# EntityManager

```
public interface EntityManager{  
    public void persist(Object entity);  
    public <T> T merge(T entity);  
    public void remove(Object entity);  
    public <T> T find(Class<T> entityClass, Object primaryKey);  
    public void flush();  
    public void setFlushMode(FlushModeType flushMode);  
    public FlushModeType getFlushMode();  
    public void refresh(Object entity);  
    public Query createQuery(String jpqlString);  
    public Query createNamedQuery(String name);  
    public Query createNativeQuery(String sqlString);  
    public Query createNativeQuery(String sqlString, Class resultClass);  
    public Query createNativeQuery(String sqlString, String  
        resultSetMapping);  
    public void close();  
    public boolean isOpen();  
    public EntityTransaction getTransaction();  
    public void joinTransaction();  
    public void clear();  
}
```

109

# persistence.xml

- persistence-unit bloc de configuration
- provider, défini le vendeur (eclipseLink)
- class, défini les entités
- les propriétés
  - url d'accès à la base
  - driver de la base
  - Utilisateur / mot de passe
  - stratégie de génération (create, create and drop, none)

111

# EntityManager

Il nécessite un fichier de configuration

## META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://  
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">  
    <persistence-unit name="blois-unitName" transaction-type="RESOURCE_LOCAL">  
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>  
        <class>fr.blois.jee.jpa.td1.Activity</class>  
        <properties>  
            <property name="javax.persistence.jdbc.url" value="jdbc:derby://localhost:1527/jpa"/>  
            <property name="javax.persistence.jdbc.driver"  
                value="org.apache.derby.jdbc.ClientDriver"/>  
            <property name="javax.persistence.jdbc.user" value="blois"/>  
            <property name="javax.persistence.jdbc.password" value="blois"/>  
            <property name="eclipselink.ddl-generation" value="drop-and-create-tables"/>  
        </properties>  
    </persistence-unit>  
</persistence>
```

110

# Obtenir le manager

- Géré par l'application

```
EntityManagerFactory entityManagerFactory =  
Persistence.createEntityManagerFactory("blois-unitName");  
EntityManager entityManager =  
entityManagerFactory.createEntityManager();  
...  
entityManager.close();  
entityManagerFactory.close();
```

- Géré par le container

```
@PersistenceContext(UnitName = "blois-unitName")  
private EntityManager entityManager;  
  
On peut aussi définir le scope transaction ou extended.  
Extended, ne peut être activer qu'avec des StateFullSessionBean (SFSB)
```

112

# @PersistenceContext

```
@Target({TYPE, METHOD, FIELD}) @Retention(RUNTIME)
public @interface PersistenceContext {
    String name() default "";
    String unitName() default "";
    PersistenceContextType type default TRANSACTION;
    PersistenceProperty[] properties() default {};
}
```

Attention, l'EntityManager n'est pas Thread safe, donc pas d'injection dans les classes comme les servlets

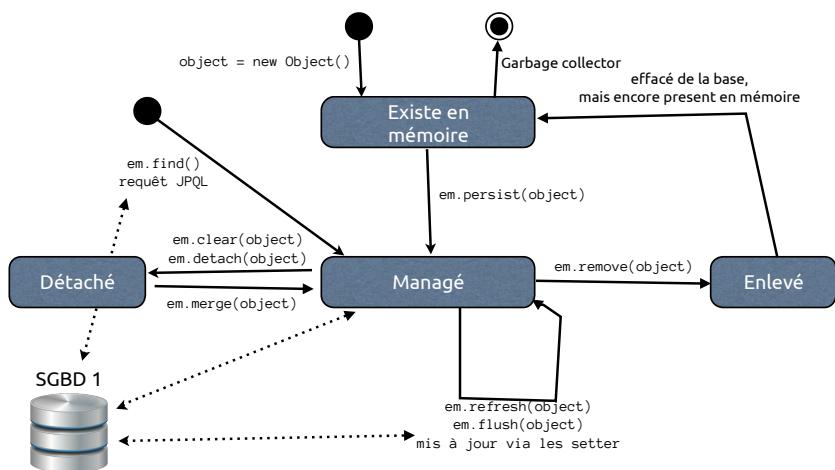
113

# Entité managée ?

- signifie que l'EntityManager s'assure que les données de l'entité sont synchronisées avec la base
- Quand l'entité devient managée, l'EntityManager synchronise son état avec la base de données
- Quand l'entité devient non-managée, l'EntityManager s'assure que les changements de données de l'entité sont répercutés en base

114

# Cycle de vie



115

# Persistir des entités

- `entityManager.persist(entity);`
- Si violation d'une contrainte d'intégrité : `PersistenceException` qui wrappe l'exception de la base de données
- Gestion automatique des clés selon la stratégie
- Problème pour la persistance de graphe d'objet : par défaut pas de persistance des objets en relation

116

# Persist des entités

```
EntityTransaction entityTransaction =  
entityManager.getTransaction();  
Person person = new Person("toto");  
Address address = new Address("rue", "ville", "00000", "pays");  
person.setAddress(address);  
entityTransaction.begin();  
entityManager.persist(address);  
address.setStreet("ma rue");  
entityManager.persist(person);  
entityTransaction.commit();
```

Le manager cache toute action après le début de la transaction (entityTransaction.begin());

117

# Détachement

```
Person person = new Person("toto");  
Address address = new Address("rue", "ville", "00000", "pays");  
person.setAddress(address);  
entityTransaction.begin();  
entityManager.persist(address);  
entityManager.persist(person);  
entityTransaction.commit();  
entityManager.detach(person);  
Assert.assertNotNull(person);  
Person person1 = entityManager.find(Person.class, 1L);  
Assert.assertNotNull(person1);  
Assert.assertEquals(person, person1);
```

Utilisation de entityManager.detach pour que l'entité person ne soit plus Managé

Le 2 objets person et person1 sont égaux au niveau des données, mais il s'agit bien de 2 instances différentes

119

# Charger une entité

- entityManager.find(Entity.class, primaryKey);
- Si les données ne sont pas trouvées en base, l'EntityManager renvoi null (ou un objet vide).
- Si Aucune transaction n'est démarrée, l'Objet renvoyé l'est en mode détaché
- Le chargement des objets (field) dépend de la stratégie de chargement (FETCH)
- Utilisation d'une requête JPQL

118

# JPA

- Crédation du projet "JPA" sous netbeans
  - Ajouter les librairies EclipseLink(JPA 2.1) et Java DB Driver
- Crédation de classe:
  - Person (nom, prénom)
  - Crédation de la base (utilisateur : ce que vous voulez)
  - Crédation du fichier Persistence.xml à l'aide de Netbeans
  - Transformation de la classe en Entité et enregistrement dans le fichiers de persistance (*pensez aux identifiants générés*)
  - Crédier 2 personnes et les persister
    - Ceci via une fonction pour factoriser
  - Ecrire une fonction qui recherche un personne en fonction de son id

Attention, pensez à initialiser vos projets en local sur les machines

N'oubliez pas, les commentaires, la javadoc peuvent être en français,  
le code est en anglais.

120

# Stratégies de chargement

- Détermine à quel moment les données seront récupérées
- 2 stratégies : LAZY & EAGER
  - EAGER: chargement immédiat (utilisation de requête join)
  - LAZY: chargement différé, exécuté à chaque demande de get (1 + N requêtes)

121

# Suppression

```
Person person = new Person("toto");
Address address = new Address("rue", "ville", "00000", "pays");
person.setAddress(address);
entityTransaction.begin();
entityManager.persist(address);
entityManager.persist(person);
entityTransaction.commit();
entityTransaction.begin();
entityManager.remove(person);
entityTransaction.commit();
Assert.assertNull(person);
Person person1 = entityManager.find(Person.class, 1L);
Assert.assertNull(person1);
```

Les données <sup>8</sup>person<sup>8</sup> ont été supprimées de la base  
Cependant l<sup>9</sup>instance reste en mémoire jusqu<sup>9</sup>au  
passage du GarbageCollector

123

# Modifications

- Entité managée
  - Etat synchronisé via l'EntityManager (utilisation des setter, au sein d'une transaction si EM transactionnel)
- Entité non managée
  - Les données seront sauvegardées lors du rattachement
  - Si l'entité n'existe pas, une exception est levée (IllegalArgumentException)

122

# Forcer les mises à jour

- l<sup>9</sup>EntityManager ne transfert pas les mises à jour immédiatement
- On peut cependant la forcer avec flush

124

## Forcer les mises à jour

```
Person person = new Person("toto");
Address address = new Address("rue", "ville", "00000", "pays");
person.setAddress(address);
entityTransaction.begin();
entityManager.persist(person);
entityManager.flush();
entityManager.persist(address);
entityTransaction.commit();
```

*Attention, cet exemple lève une IllegalStateException car la clé étrangère n'a pas été créée*

125

## Manipuler le PC

```
Person person = new Person("toto");
Address address = new Address("rue", "ville", "00000", "pays");
person.setAddress(address);
entityTransaction.begin();
entityManager.persist(person);
entityManager.persist(address);
entityTransaction.commit();
Assert.assertTrue(entityManager.contains(person));
entityManager.detach(person);
Assert.assertFalse(entityManager.contains(person));
```

127

## Refresh data...

```
Person person = new Person("toto");
Address address = new Address("rue", "ville", "00000", "pays");
person.setAddress(address);
entityTransaction.begin();
entityManager.persist(person);
entityManager.persist(address);
entityTransaction.commit();
person.setName("titi");
entityManager.refresh(person);
Assert.assertEquals("toto", person.getName());
```

126

## Hang on !!

- Suite du projet...
- Création d'une méthode de suppression basé sur l'id (et sur l'instance)
- Création d'un classe **Address** (rue, ville, code postal, pays)
  - Relation OneToOne avec **Person**.
- Assigner une adresse différente pour 2 personnes
- Persister le tout
- Créer une méthode qui modifie le prénom d'une personne, et persister.
- Détacher une personne, puis changer son nom, puis la recharger (sans persist). Vérifier que le nom n'a pas changé sur la base (via le code). De plus mettre en évidence la non égalité des instances

128

# Rattachement

```
Person person = new Person("toto");
Address address = new Address("rue", "ville", "00000", "pays");
person.setAddress(address);
entityTransaction.begin();
entityManager.persist(person);
entityManager.persist(address);
entityTransaction.commit();
entityManager.clear();
person.setName("titi");
entityTransaction.begin();
entityManager.merge(person);
entityTransaction.commit();
```

129

# Evenements en cascade

```
@Entity
@Table(name = "PERSON")
public class Person {
    @Id
    @GeneratedValue(generator = "PERSON_SEQ")
    @Column(name = "PERSON_ID")
    private Long id;
    @Column(nullable = false)
    private String name;
    @OneToOne
    @JoinColumn(name = "ADDRESS_FK", nullable = false)
    private Address address;
    .....
}

entityTransaction.begin();
entityManager.persist(person);
entityManager.persist(address);
entityTransaction.commit();
```

131

# Evenements en cascade

Type	Description
PERSIST	Les relations sont persistées en même temps que la classe parent
REMOVE	Les relations sont supprimées en même temps que la classe parent
MERGE	La classe parent et les association sont soumises en même temps aux opérations de rattachement
REFRESH	Classes et associations sont rafraîchis au même moment
DETACH	Classes et associations sont détachées au même moment
ALL	Association de toutes les conditions précédentes

130

# Evenements en cascade

```
@Entity
@Table(name = "PERSON")
public class Person {
    @Id
    @GeneratedValue(generator = "PERSON_SEQ")
    @Column(name = "PERSON_ID")
    private Long id;
    @Column(nullable = false)
    private String name;
    @OneToOne(
        fetch = FetchType.LAZY,
        cascade = {CascadeType.PERSIST, CascadeType.MERGE}
    )
    @JoinColumn(name = "ADDRESS_FK", nullable = false)
    private Address address;

    entityTransaction.begin();
    entityManager.persist(person);
    entityTransaction.commit();
```

132

## Requeter une entité

- EntityManager gère les opérations de type CRUD
- Les opérations CRUD ne suffisent pas forcément
- Besoin d'un langage de requête complexe
- avec une vue objet (pas de vue SGDB)
- Utilisation de **JPQL**
- *On peut aussi utiliser le SQL*

133

## Utilisation JPQL

```
Person person = new Person("toto");
Address address = new Address("rue", "ville", "00000", "pays");
person.setAddress(address);
entityTransaction.begin();
entityManager.persist(person);
entityTransaction.commit();
entityManager.clear();
Query query =
    entityManager.createQuery("select p from Person p");
List<Person> personList = query.getResultList();
Assert.assertTrue(personList.size() == 1);
Assert.assertEquals(person, personList.get(0));
```

135

## JPQL

- Langage de requête JPA
- Converti ses requêtes en SQL via le JPQL Processor Query



Si les requêtes sont exécutées hors transaction, les entités sont détachées

134

## Requêtes nommées

- Centraliser les requêtes, garder la logique métier claire
- Maintenance plus aisée
- Amélioration des performances



Attention, une requête nommée est liée au scope du PersistenceUnit, **son nom doit être unique.**

136

# Utilisation JPQL

```
@Entity  
@Table(name = "PERSON")  
@NamedQueries(  
    @NamedQuery(name = "Person_findAll", query = "select p from Person  
p")  
)  
public class Person { ... }  
  
Person person = new Person("toto");  
Address address = new Address("rue", "ville", "00000", "pays");  
person.setAddress(address);  
entityTransaction.begin();  
entityManager.persist(person);  
entityTransaction.commit();  
entityManager.clear();  
Query query =  
    entityManager.createNamedQuery("Person_findAll");  
List<Person> personList = query.getResultList();  
Assert.assertTrue(personList.size() == 1);  
Assert.assertEquals(person, personList.get(0));
```

137

# Requêtes paramétrées

- Possibilité de passer un paramètre par son index

```
Query query = entityManager.createQuery(  
    "select p from Person p where p.name = ?1");  
query.setParameter(1, "toto");
```

- Possibilité de passer un paramètre par un alias

```
Query query = entityManager.createQuery(  
    "select p from Person p where p.name = :nameParameter");  
query.setParameter("nameParameter", "toto");
```

138

# JPQL pour charger

- Charger une liste d'entités

```
List personList = query.getResultList();

- aucun résultat == liste vide
- Possibilité de pagination via l'API Query

```

- Charger un résultat unique

```
Person personFound = (Person)query.getSingleResult();

- Lève des exceptions
  - NonUniqueResultException et
  - NoResultException qui sont des exceptions RunTime

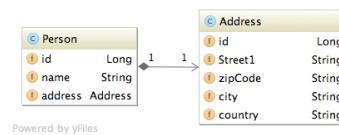
```



139

# JPQL et les relations

Sélection d'un élément via l'arbre de relation objet...



```
Person person = new Person("toto");  
Address address = new Address("rue", "ville", "00000", "pays");  
person.setAddress(address);  
...  
Query query = entityManager.createQuery("select p from Person p  
where p.address.country = :countryParameter");  
query.setParameter("countryParameter", "pays");  
Person personFound = (Person)query.getSingleResult();
```

140

## Mots clé

- **Statement et clause :** SELECT, UPDATE, DELETE, FROM, WHERE, GROUP, HAVING, ORDER, BY, ASC, DESC
- **Jointures :** JOIN, OUTER, INNER, LEFT, FETCH
- **Conditions et opérateurs :** DISTINCT, OBJECT, NULL, TRUE, FALSE, NOT, AND, OR, BETWEEN, LIKE, IN, AS, UNKNOWN, EMPTY, MEMBER, OF, IS, NEW, EXISTS, ALL, ANY, SOME
- **Fonctions :** AVG, MAX, MIN, SUM, COUNT, MOD, UPPER, LOWER, TRIM, POSITION, CHARACTER\_LENGTH, CHAR\_LENGTH, BIT\_LENGTH, CURRENT\_TIME, CURRENT\_DATE, CURRENT\_TIMESTAMP

141

## Structure d'une requête

```
SELECT [DISTINCT] <expression de selection> [[AS] <alias>]  
FROM <clause from (classe)>  
[WHERE <conditions>]  
[ORDER BY <clause de tri>]  
[GROUP BY <clause de regroupement>]  
[HAVING <clause de « possession »>]
```

143

## Opérateurs

Type	Opérateurs
Navigation	.
Signe Unaire	+ -
Arithmétique	/ * - +
Relationnel	<, >, =, <=, >=, <>, [NOT] BETWEEN, [NOT] LIKE, [NOT] IN, IS [NOT] NULL, IS [NOT] EMPTY, [NOT] MEMBER [OF]
Logique	NOT, AND, OR

142

## Jointure

- Les jointures suivent la même logique qu'en SQL
- INNER JOIN
- LEFT OUTER JOIN

144

# Opérations de masse

- Les actions en masse sont possibles
- Mise à jour globale (ou sur critères)
- Effacement globale (ou sur critères)

145

## Hands on !!

- Suite du projet...
- Crédation d'une fonction d'effacement selon le pays (avec requête nommée et paramétrée) - Quelle erreur dans notre cas ? (intégrité de la base ?...)
- Crédation d'un fonction renvoyant la liste des personnes habitant un pays

147

# Utilisation SQL

## Requête native

- Possibilité d'utiliser des requêtes SQL

```
Query query = entityManager.createNativeQuery("SELECT * FROM PERSON");
```

- Utilisation de particularité de la base
- Portabilité moyenne
- Utilisation de ResultSetMapping pour le résultat
- Accès aux procédures stockées, par exemple

146

EJB

148

## EJB == Logique métier

- La couche de persistance n'est pas appropriée pour la logique métier
- La couche de présentation ne doit pas exécuter la logique métier
- Besoin de sécurité et de transaction
- Interaction avec des services extérieurs

Besoin d'une couche métier !

149

## Les types d'EJB

- Stateless: Gestion sans état. [Session bean](#)
- Stateful: Gestion conversationnelle
- Singleton
- Message Driven bean: interaction avec JMS

150

## un EJB...

```
@Stateless  
public class UserEJB {  
  
    @PersistenceContext  
    EntityManager entityManager;  
  
    public User findUserById(Long id) {  
        return entityManager.find(User.class, id);  
    }  
}
```

151

## Anatomie d'un EJB

- C'est un POJO
- Annotation `@Stateless`, `@Statefull`, `@Singleton`, `@MessageDriven` (ou un descripteur XML)
- Il peut avoir plusieurs interfaces.
- Constructeur sans arguments
- Classe non finale, ni abstraite

152

## Comment appeler un EJB

```
public class Main {  
  
    @EJB  
    private static UserEJB userEJB;  
  
    public static void main(String... args) {  
        User user = userEJB.findUserById(1L);  
    }  
}
```

153

## Intégration

- Session Beans, MDBs, Entities
- Aide à mettre les composants en relation par la configuration plutôt que le code, par Injection de dépendance (Dependency Injection DI) et Lookup.

155

## Les services de la couche métier

- Intégration
- Sécurité
- intercepteurs
- Cycle de vie et pooling
- Accès distant
- Gestion de l'état
- Messaging
- Transaction
- Multi thread
- Web service
- Invocation asynchrone
- Injection de dépendances

154

## Pooling

- SLSBs, MDBs
- Création d'un lot d'instance des Beans gérés par le container. Le container n'autorise qu'un seul client à accéder à une instance. Après utilisation l'instance retourne dans le pool.

156

## Threads

- Session Beans, MDBs
- Le container assure l'accès à une instance par un seul client. Pas de gestion de la concurrence.
- Depuis JEE 7, dans certains cas le multitread est possible, mais interdit dans les versions antérieures

157

## Gestion de l'état

- SFSB (Statefull Session Bean)
- Gestion automatique de l'état par le container. Il gère la persistance des SFSB et l'association des instances avec les utilisateurs

158

## Messaging

- MDBs (Message Driven Beans)
- Simplification extrême de la création de composants <sup>8</sup>messaging-aware<sup>8</sup> : écoute un canal de communication et réagit à l'arrivée de messages

159

## Transaction

- Session Beans, MDBs
- Possibilité d'utiliser la configuration pour déléguer la gestion des transactions au container d'EJB.

160

## Sécurité

- Session Beans
- Possibilité d'utiliser la configuration pour interagir avec JAAS (Java Authentication and Authorization Service)

161

## intercepteur

- Session Beans, MDBs
- Externalise la gestion des problèmes transverses des applications (logging, auditing...)
- Version simplifiée de l'AOP (Aspect Oriented Programming)

162

## Accès distant

- Session Beans
- Accès distant (RMI) sans avoir de code à écrire
- Possibilité d'injection de dépendance permettant une utilisation comme si on était dans le container

163

## Web services

- SLSBs
- Minimalise les changements de code pour rendre un Stateless Session Bean accessible par Service Web

164

## Invocation d'un EJB

- JNDI Lookup permet la récupération d'objet dans l'arbre JNDI du serveur
  - Problème : couplage du code avec le serveur, <sup>8</sup>boilerplate code<sup>8</sup>
- Injection Dépendance aussi simple qu'une annotation

165

## Injection de dependance

```
@Stateless  
public class UserEJB {  
    @PersistenceContext  
    EntityManager entityManager;  
    public User findUserById(Long id) {  
        return entityManager.find(User.class, id);  
    }  
}  
  
public class Main {  
    @EJB  
    private static UserEJB userEJB;  
    public static void main(String... args) {  
        User user = userEJB.findUserById(1L);  
    }  
}
```

167

## Lookup JNDI

```
@Stateless  
@EJB(name = "UserEJBSERVICE")  
public class UserEJB {  
    @PersistenceContext  
    EntityManager entityManager;  
    public User findUserById(Long id) {  
        return entityManager.find(User.class, id);  
    }  
}  
  
public void MainString... args() {  
    try {  
        InitialContext initialContext = new InitialContext();  
        UserEJB userEJB = (UserEJB) initialContext.lookup("java:comp/env/UserEJBSERVICE");  
        User user = userEJB.findUserById(1L);  
    } catch(NamingException e) {  
        // gestion exception  
    }  
}
```

166

## @EJB

```
@Target({TYPE, METHOD, FIELD})  
@Retention(RUNTIME)  
public @interface EJB{  
    String name() default IJ;  
    Class beanInterface() default Object.class;  
    String beanName() default IJ;  
}
```

168

## @EJB

- name : Nom utilisé pour lier à l'arbre JNDI.
- beanInterface : Interface business utilisée pour accéder à l'EJB
- beanName : Distinction si plusieurs EJBs implémentent la même interface business.

169

## Les Bean de session

171

## Annotation ou XML

- Question de goût.
- Annotations : concis, proche du code
  - Beaucoup ont des valeurs par défaut
- XML : verbeux mais modifiable sans recompilation
- XML permet un surcharge des annotations

170

## Quelques définitions

- Session : connexion entre un client et un serveur qui dure une période de temps finie
- Client : ligne de commande, composant web (servlet, JSP, JSF,...), application Desktop, voir application .NET via des web services

172

## Les outils des bean de session

- Les EJB fournissent des services aux développeurs
  - Concurrence et Thread Safety
  - Remoting et web services
  - Transaction et sécurité
  - Timers et intercepteurs

173

## EJB et interfaces

```
@Stateless  
public class HelloToTheWorld {  
    public String sayBonjour(String name) {  
        return "bonjour " + name;  
    }  
}
```

175

## Les <sup>8</sup>session beans<sup>8</sup>

- Stateless: littéralement sans état conversationnel, Utilisé pour réaliser des tâches avec un simple appel de méthode.
- Statefull: Maintient un état (et les données associées) avec un client spécifique. Utilisé pour réaliser des actions en plusieurs étapes.
- Singleton: design pattern du singleton. C'est le container qui s'assure de l'unicité de cet EJB.

174

## EJB et interfaces

```
@Stateless @LocalBean  
public class HelloToTheWorld implements HelloToTheWorldLocal {  
    public String sayBonjour(String name) {  
        return "bonjour " + name;  
    }  
    @Override  
    public String sayGutenTag(String name) {  
        return "guten tag " + name;  
    }  
}
```

```
@Local  
public interface HelloToTheWorldLocal {  
    String sayGutenTag(String name);  
}
```

176

# EJB et interfaces

```
@Stateless @LocalBean
public class HelloToTheWorld implements HelloToTheWorldLocal, HelloToTheWorldRemote {
    public String sayBonjour(String name) {
        return "bonjour " + name;
    }
    @Override
    public String sayGutenTag(String name) {
        return "guten tag " + name;
    }
    @Override
    public String sayHello(String name) {
        return "hello " + name;
    }
}
```

```
@Local
public interface HelloToTheWorldLocal {
    String sayGutenTag(String name);
}
```

```
@Remote
public interface HelloToTheWorldRemote {
    String sayHello(String name);
}
```

177

# EJB et interfaces (alt)

```
@Stateless @LocalBean
@Local(value = HelloToTheWorldLocal.class)
@Remote(value = HelloToTheWorldRemote.class)
public class HelloToTheWorld implements HelloToTheWorldLocal, HelloToTheWorldRemote {
    public String sayBonjour(String name) {
        return "bonjour " + name;
    }
    @Override
    public String sayGutenTag(String name) {
        return "guten tag " + name;
    }
    @Override
    public String sayHello(String name) {
        return "hello " + name;
    }
}
```

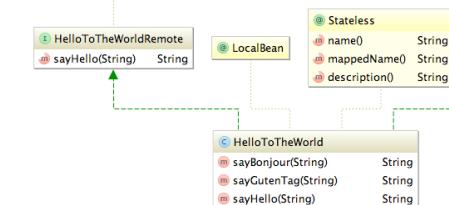
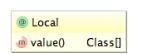
```
public interface HelloToTheWorldLocal {
    String sayGutenTag(String name);
}
```

```
public interface HelloToTheWorldRemote {
    String sayHello(String name);
}
```

179

# EJB et interfaces

```
@Stateless @LocalBean
public class HelloToTheWorld implements HelloToTheWorldLocal, HelloToTheWorldRemote {
    public String sayBonjour(String name) {
        return "bonjour " + name;
    }
    @Override
    public String sayGutenTag(String name) {
        return "guten tag " + name;
    }
    @Override
    public String sayHello(String name) {
    }
}
```

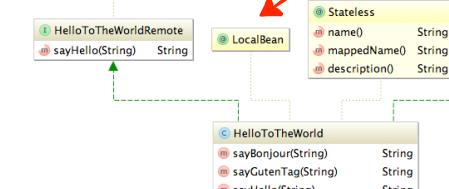
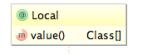


Powered by yfiles

178

# EJB, interfaces & appel

```
public class CallHelloToTheWorld {
    @EJB
    private HelloToTheWorld helloToTheWorld;
    @EJB
    private HelloToTheWorldLocal helloToTheWorldLocal;
    @EJB
    private HelloToTheWorldRemote helloToTheWorldRemote;
}
```



Powered by yfiles



180

## EJB et interfaces (bonus)

```
@Stateless @LocalBean  
public class HelloToTheWorld  
    implements HelloToTheWorldRemote, HelloToTheWorldLocal, HelloToTheWorldSOAP, HelloToTheWorldRest {  
    public String sayBonjour(String name) {  
        return "bonjour " + name;  
    }  
    @Override  
    public String sayGutenTag(String name) {  
        return "guten tag " + name;  
    }  
    @Override  
    public String sayHello(String name) {  
        return "hello " + name;  
    }  
    @Override  
    public String sayHelloSOAP() {  
        return "hello through SOAP";  
    }  
    @Override  
    public String sayHelloREST() {  
        return "hello through SOAP";  
    }  
}
```

```
@Remote  
public interface HelloToTheWorldRemote {  
    String sayHello(String name);  
}
```

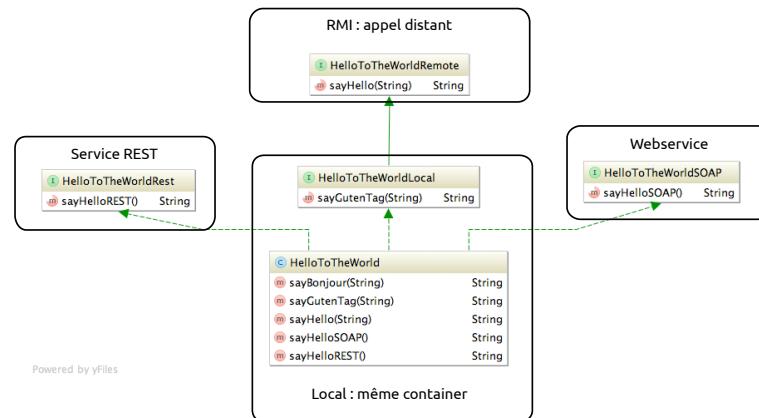
```
@Local  
public interface HelloToTheWorldLocal  
    extends HelloToTheWorldRemote {  
    String sayGutenTag(String name);  
}
```

```
@WebService  
public interface HelloToTheWorldSOAP {  
    String sayHelloSOAP();  
}
```

```
@Path("/hello")  
public interface HelloToTheWorldRest {  
    String sayHelloREST();  
}
```

181

## EJB et interfaces (bonus)



182

## Règles de programmation

- Au moins une interface métier (business)
- Classe concrète
- Constructeur sans argument
- Plusieurs annotations sur une interface, impossible, utilisation de l'héritage
- Les règles d'héritage OO s'appliquent

183

## Règles de programmation

- Héritage des annotations pour DI et lifecycle callback
- Les méthodes business ne doivent pas démarrer par `ejb8`
- Les méthodes business doivent être public, non final et non static
- Si les méthodes sont dans l'interface remote, les arguments et le type de retour doivent implémenter `java.io.Serializable`

184

## Stateless ou Stateful ?

- Cinématique ayant besoin d'un état conversationnel (gestion de caddie, formulaire sur plusieurs pages...)
- Stateful bean permet de gérer cette conservation des données en session sur le serveur, plus simplement que le session HTTP

185

## Stateless

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface Stateless {
    java.lang.String name() default "";
    java.lang.String mappedName() default "";
    java.lang.String description() default "";
}
```

186

## Stateless

```
@Stateless @LocalBean
public class HelloToTheWorld implements HelloToTheWorldLocal {
    public String sayBonjour(String name) {
        return "bonjour " + name;
    }
    @Override
    public String sayGutenTag(String name) {
        return "guten tag " + name;
    }
}
```

187

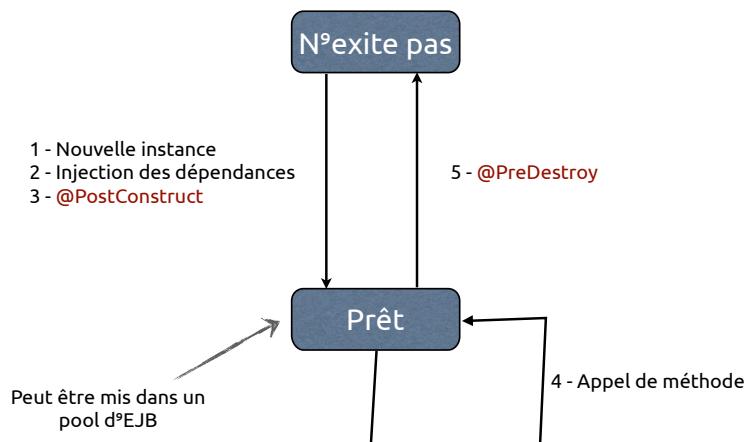
## Stateless

- name : Nom du Bean. Certains containers l'utilisent pour lier l'EJB dans l'arbre JNDI. Par défaut le nom de la classe
- mappedName : vendeur spécifique. Utilisé par certains containers pour lier l'EJB dans l'arbre JNDI
- description : pour la console d'administration

188

# Stateless

## Cycle de vie



189

# Hands on !! (1)

Il s'agit de mettre en place un EJB sans état pour manipuler une personne et logger dans la console les événements de création de l'EJB et de destruction.

Récupérez le source EJBModule et copiez le en local sur votre PC.

Vous remarquerez une organisation différente des sources:

- Ce sont des sources de type «EJB»
- Observez le `persistence.xml`
  - Il ne contient plus les informations de connexion mais une référence à un datasource
  - La datasource est défini dans la configuration du serveur.

190

# Hands on !! (2)

Il s'agit de mettre en place un EJB sans état pour manipuler une personne et logger dans la console les événements de création de l'EJB et de destruction.

- Création de l'EJB (Stateless) - il est déjà créer, mais il faut le transformer
- Création d'une méthode de création de personne
- Création d'une méthode pour effacer une personne
- Création d'une méthode pour trouver une personne selon son identifiant
- Création d'une méthode pour recuperer toutes les personnes enregistrés
- Logger avec la console (System.out...) la création de l'EJB (interdit de placer cela dans le constructeur !!!!)
- Utiliser la classe Main (en créant une instance via le main) pour tester tout cela.

191

# Singleton

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface Singleton {
    java.lang.String name() default "";
    java.lang.String mappedName() default "";
    java.lang.String description() default "";
}
```

192

# Singleton

```
@Singleton  
public class AddressCache {  
    public Address getAddressFromCache(long id) {return address}  
    public void putAddressToCache(Address address) {}  
    public void removeAddressFromCache(long id) {}  
}
```

193

# Singleton

```
@Singleton  
@Startup  
public class AddressCache {  
  
    @PostConstruct  
    private void addressCacheInitialization() {  
        // Initialisation longue  
    }  
    public Address getAddressFromCache(long id) {return address}  
    public void putAddressFromCache(Address address) {}  
    public void removeAddressFromCache(long id) {}  
}
```

194

# Singleton

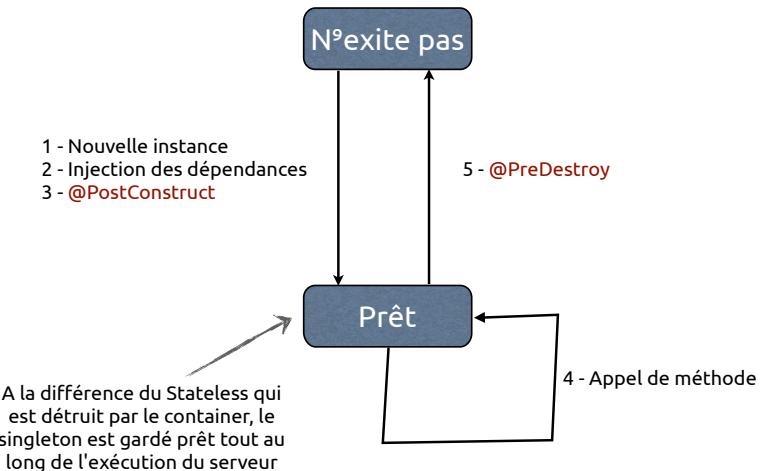
## Enchainement

```
@Singleton  
public class ZipcodeCache {  
}  
@Singleton  
@Startup  
@DependsOn("CityCache", "ZipcodeCache")  
public class AddressCache {  
    @PostConstruct  
    private void addressCacheInitialization() {  
        // Initialisation longue  
    }  
    public Address getAddressFromCache(long id) {return address}  
    public void putAddressFromCache(Address address) {}  
    public void removeAddressFromCache(long id) {}  
}
```

195

# Singleton

## Cycle de vie



196

# Singleton

## Concurrence

- Une instance, plusieurs clients
  - Problèmes d'accès concurrent
  - `@ConcurrencyManagement`
  - Concurrence gérée par le container
    - Container-Management Concurrency **CMC**
  - Concurrence gérée par le bean
    - Bean-Managed Concurrency **BMC**

197

# Singleton

## Concurrence gérée par le container

- Le verrouillage peut être dirigé `@Lock`
  - `@Lock(LockType.READ)` - verrou partagé
  - `@Lock(LockType.WRITE)` - verrou exclusif
- Déclaration au niveau de la classe et des méthodes

198

# Singleton

## Concurrence gérée par le container

```
@Singleton  
@Lock(LockType.READ)  
public class AddressCache {  
    public Address getAddressFromCache(long id) {  
        return address;  
    }  
    @Lock(LockType.WRITE)  
    @AccessTimeout(2000)  
    public void putAddressToCache(Address address) {  
    }  
    public void removeAddressFromCache(long id) {  
    }  
}
```

199

# Singleton

## Concurrence gérée par le bean

- Le développeur doit gérer les accès concurrentiels
- Utilisation de `synchronized` & `volatile`

200

# Singleton

## Concurrence gérée par le bean

```
@Singleton  
@ConcurrencyManagement(ConcurrencyManagementType.BEAN)  
public class AddressCache {  
    public synchronized Address getAddressFromCache(long id) {  
        return null;  
    }  
    public synchronized void putAddressToCache(Address address) {  
    }  
    public void removeAddressFromCache(long id) {  
    }  
}
```

201

# Hands on !!

Pour ce TD, il faut mettre en place un singleton qui au lancement va remplir la base avec les données que nous avions injectées avec la classe main.

- Création de l'EJB (singleton) - Il s'agit de remplir la base, exemple de nom : PersonPopulate.
- Faire en sorte que ce singleton se lance au démarrage du conteneur d'EJB...
- Mise en place de la méthode qui permettra l'exécution des méthodes d'initialisation de la base (petit indice, il sera créé automatiquement au lancement, repensez au cycle de vie ;)
- Depuis l'initialisation de la base, dans l'ordre
  - Supprimer tous les enregistrements de personne et adresse (soyez malin)
  - Remplir la base (soyez fainéant :: respectez le principe de responsabilité)
- Utiliser la classe Main (en créant une instance via le main) pour tester tout cela. Pour cette fois, il suffira de supprimer du code ;)

202

# Stateful

```
@Stateful  
@StatefulTimeout(value = 30, unit = TimeUnit.SECONDS)  
public class HelloToWorld implements HelloToWorldLocal {  
    public String sayBonjour(String name) {  
        return "bonjour " + name;  
    }  
    @Override  
    public String sayGutenTag(String name) {  
        return "guten tag " + name;  
    }  
}
```

203

# Stateful

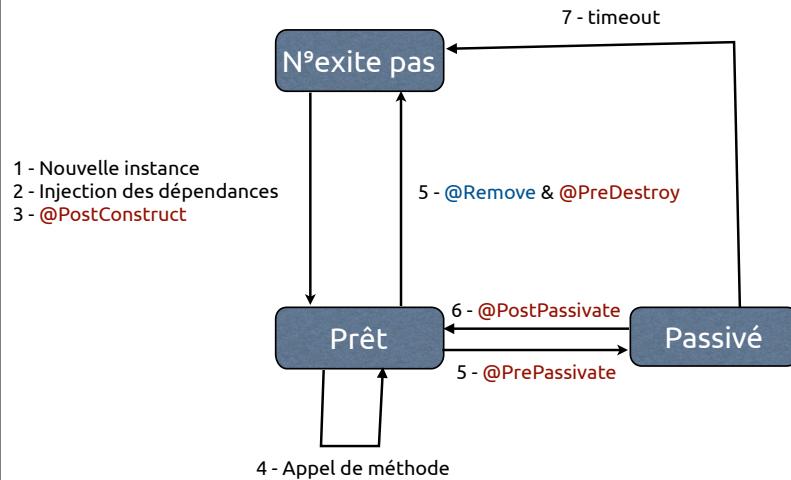
## Règles de programmation

- Les attributs doivent être des primitives ou implémenter java.io.Serializable pour permettre la passivation
- Prévoir une méthode de destruction pour libérer les ressources
- Prendre en compte du cycle de vie différent (callback)

204

# Stateful

## Cycle de vie



205

# Stateful

## Passivation

- Serialisation de l'EJB sur le disque lorsqu'il est trop longtemps sans servir
- Le sens contraire : Activation
- deux callbacks : @PrePassivate @PostActivate, il peut s'agir des mêmes méthodes que pour @PostConstruct et @PreDestroy

206

# Stateful

## @Remove

- Annotation sur une ou plusieurs méthodes de l'EJB
- Indique que l'EJB doit être détruit après exécution de la méthode
- Permet libération des ressources de l'EJB sans attente du timeout et évite passivation/activation excessives

207

# Stateful

## Règle pour l'appel d'un EJB Stateful

- Pas d'injection de dépendance d'un SFSB dans un SLSB
- Injection d'un SFSB dans une servlet, même instance pour tous les clients

208

# Stateful

## Performance

- Coût d'un EJB Statefull
  - Occupation mémoire / disque
  - Problème de réplication en réseau (cluster) et donc coût réseau
- Attention aux objets référencés (si possible utiliser des identifiants)
- Attention à la configuration (nom de SFSB max actif et timeout)
- Ne pas oublier @Remove

209

# EJB & Transaction

211

# Différence SFSB / SLSB

Fonctionnalité	Stateless	Stateful
État conversationnel	NON	OUI
Problème de performance	Selon les méthodes	Possible
lifecycle callback	@PostConstruct & @PreDestroy	@PostConstruct, @PreDestroy, @PrePassivated, @PostPassivated
Timer	OUI	NON
Synchronisation de la session	NON	OUI
WebService	OUI	NON
Pool	OUI	NON
Extended Persistence Context	NON	OUI

210

# Transaction

- Les données sont fondamentales
- Elles doivent être précises quelles que soit les opérations effectuées
- Y compris lors d'un accès concurrentiel
- Les transactions assurent un état consistant
- Une série d'opérations a besoin d'être exécutée comme une simple opération

212

## ACID

- **Atomicity** : tout ou rien, la transaction se termine soit par un commit soit un rollback
- **Consistency** : le système est dans un état consistant avant la transaction et dans un état consistant après, quel que soit le devenir de la transaction.
- **Isolation** : Les changements d'une transaction ne sont pas visibles à une autre transaction avant le commit
- **Durability** : les données commises sont permanentes et survivent à un crash système

213

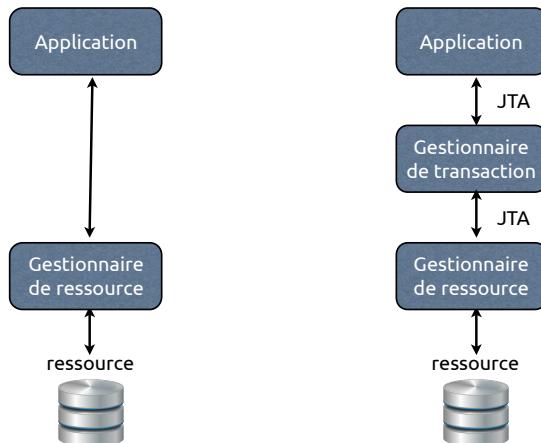
## support transaction

- Les EJB sont transactionnelles par défaut
- Les outils du framework vous aident :
  - Transaction manager
  - Resource manager
- Utilisation JTA
- 2 gestions des transactions
  - Container-Managed-Transaction (CMT)
  - Bean-Managed-Transaction (BMT)

214

## Transaction locale

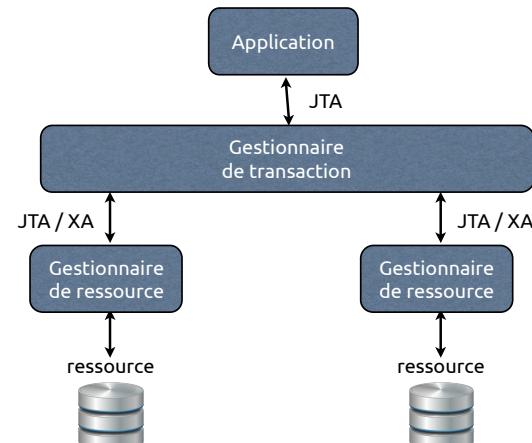
une seule ressource transactionnelle



215

## Transactions Distribuées

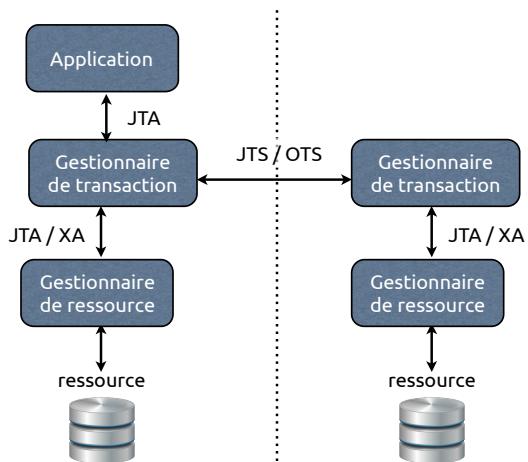
XA



216

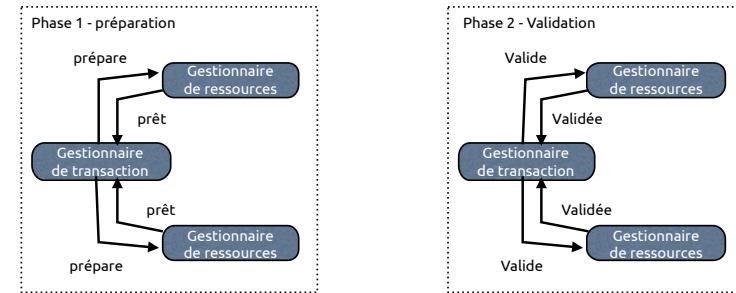
# Transactions Distribuées

XA via le réseau



217

# Validation en 2 phases



218

## Container-Managed-Transaction

```
@Stateless  
@TransactionManagement(TransactionManagementType.CONTAINER)  
public class TaskToDoEJB {  
    @PersistenceContext  
    private EntityManager entityManager;  
  
    @EJB  
    private ToDoListEJB toDoListEJB;  
  
    public Task createTask(Task task) {  
        entityManager.persist(task);  
        toDoListEJB.addTask(task);  
        return task;  
    }  
}
```

219

## Container-Managed-Transaction

```
@Stateless  
@TransactionManagement(TransactionManagementType.CONTAINER)  
public class TaskToDoEJB {  
    @PersistenceContext  
    private EntityManager entityManager;  
  
    @EJB  
    private ToDoListEJB toDoListEJB;  
  
    public Task createTask(Task task) {  
        entityManager.persist(task);  
        toDoListEJB.addTask(task);  
        return task;  
    }  
}
```

220

# Attributs de transaction

Gestion des transactions par le container  
@TransactionAttribute

@TransactionAttribute	Dans une transaction	hors Transaction
REQUIRED <small>MDB</small>	Rattrape la transaction du client (appelant)	Création d'une nouvelle transaction
REQUIRES_NEW	Création d'une nouvelle transaction et suspend la transaction du client	Création d'une nouvelle transaction
SUPPORTS	Rattrape la transaction du client (appelant)	Pas de transaction
MANDATORY	Rattrape la transaction du client (appelant)	déclenche une EJBTransactionRequiredException
NOT_SUPPORTED <small>MDB</small>	La transaction du client est suspendue et aucune transaction est créée	Pas de transaction
NEVER	déclenche une EJBException	Pas de transaction

221

# Transactions et exceptions

- par défaut toutes les `checked Exception` sont gérées par le client et se voit ajouter le `@ApplicationException`
- les `runtime Exception` sont wrappées dans des `EJBException`
- Par défaut, toutes les exceptions entraînent un rollback
- par contre, la gestion du rollback peut être modérée

```
@ApplicationException(rollback = false)
public class TaskAlrearyDoneException
extends Exception { }
```

223

# Annulation de transaction

- On peut noter une transaction en rollback en utilisant la méthode `SessionContext.setRollbackOnly()`
- l'appel de cette méthode hors contexte transactionnel ou dans un contexte BMT lève une `IllegalStateException`

```
@PersistenceContext
private EntityManager entityManager;
@Resource
private SessionContext sessionContext;
@TransactionalAttribute(TransactionAttributeType.MANDATORY)
public Task createTask(Task task) {
    entityManager.persist(task);
    toDoListEJB.addTask(task);
    sessionContext.setRollbackOnly();
    return task;
}
```

222

# Bean-Managed-Transaction

```
@Resource
private UserTransaction userTransaction;
public Task createTask(Task task) {
    try {
        userTransaction.begin();
        entityManager.persist(task);
        toDoListEJB.addTask(task);
        if (noError) {
            userTransaction.commit();
        } else {
            userTransaction.rollback();
        }
    } catch (Exception e) {
        try {
            userTransaction.setRollbackOnly();
        } catch (SystemException e1) {
            // Gestion erreur
        }
        // gestion erreur
    }
    return task;
}
```

224

## @UserTransaction

```
package javax.transaction;

public interface UserTransaction {
    void begin() throws NotSupportedException, SystemException;

    void commit() throws RollbackException, HeuristicMixedException, HeuristicRollbackException,
SecurityException, IllegalStateException, SystemException;

    void rollback() throws IllegalStateException, SecurityException, SystemException;

    void setRollbackOnly() throws IllegalStateException, SystemException;

    int getStatus() throws SystemException;

    void setTransactionTimeout(int i) throws SystemException;
}
```

225

## Avantages & inconvénients BMT

- Avantages
  - gestion plus fine des limites de la transaction
  - permet maintien de transactions entre les appels pour un SFSB
- Inconvénients
  - code plus verbeux et susceptible d'erreur
  - ne peut joindre une transaction courante

227

## @UserTransaction et status

- Interface regroupant les valeurs retournées par UserTransaction.getStatus()
- Les valeurs possibles :
  - STATUS\_ACTIVE
  - STATUS\_MARKED\_ROLLBACK
  - STATUS\_PREPARED
  - STATUS\_COMMITTED
  - STATUS\_ROLLEDBACK
  - STATUS\_UNKNOWN
  - STATUS\_NO\_TRANSACTION
  - STATUS\_PREPARING
  - STATUS\_COMMITTING
  - STATUS\_ROLLING\_BACK

226

## Hands on !!

Simulation de Statefull bean.

- Création d'un bean de session Statefull
  - Permettant d'ajouter une série de personnes sans les persister
  - la méthode permettant de les persister tous relâche le bean
  - Utiliser les callback pour suivre le cycle de vie du bean

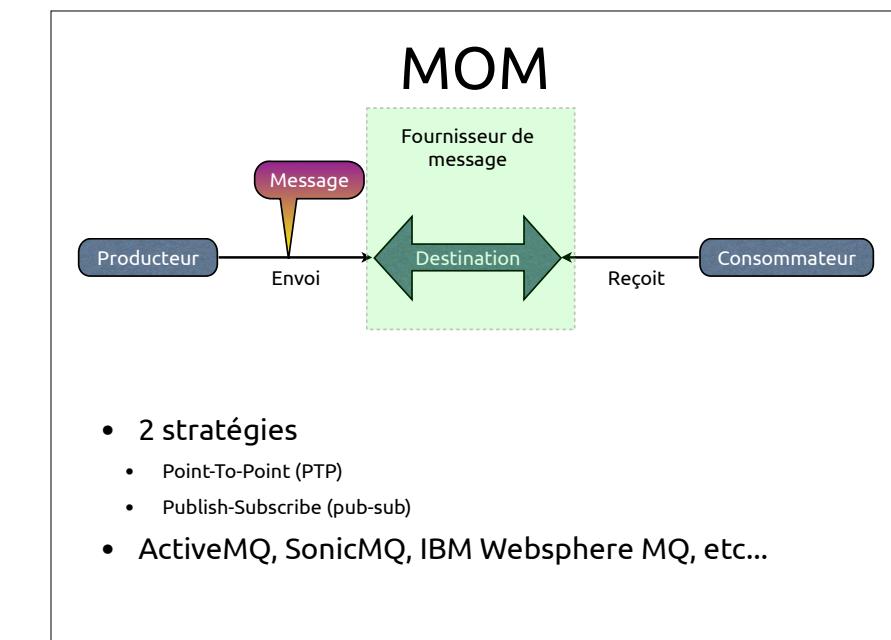
228

# Message Driven Bean

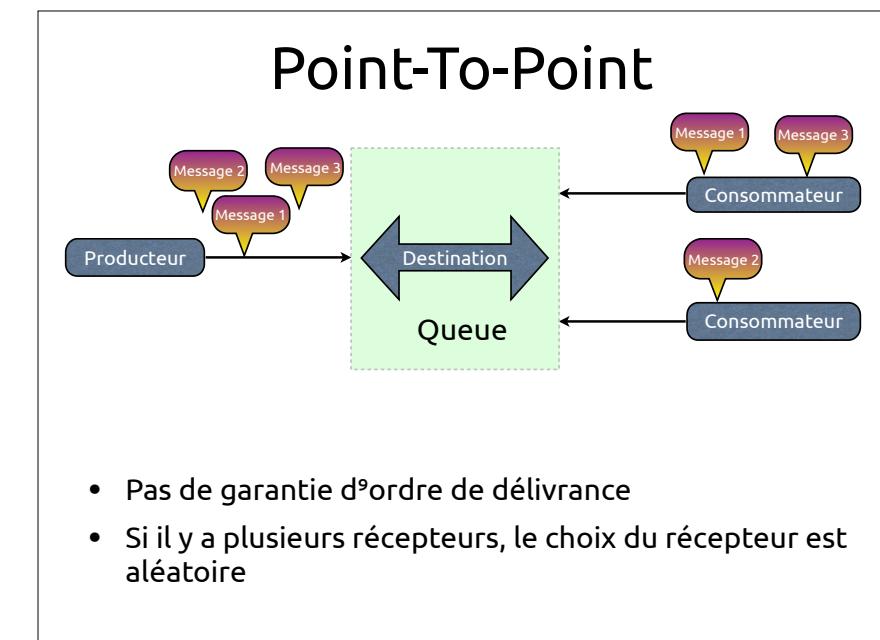
229

## Message

- Communication asynchrone
- Faiblement couplée
- Basé sur MOM

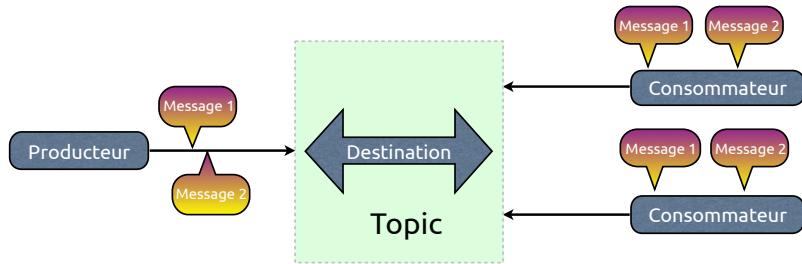


231



232

## Publish-Subscribe



- Timer entre Publisher & Subscriber sont liés
- Si un abonné est inactif au delà d'une période donnée, il ne recevra pas le(s) message(s)

233

## Request-Reply

- permet d'avoir un accusé réception des Consommateurs
- S'ajoute au dessus des deux précédents
- Pour permettre la réponse, ajout d'informations dans le message comme un identifiant unique et une Queue de destination pour la réponse en PTP

234

## Emetteur : exemple

```
@Resource(name = "jms/QueueConnectionFactory")
private ConnectionFactory connectionFactory;
@Resource(name="jms/MessageQueue")
private Destination destination;
public void sendMessage() {
    try {
        Connection connection = connectionFactory.createConnection();
        Session session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
        MessageProducer messageProducer = session.createProducer(destination);
        textMessage.setJMSPriority(1);
        textMessage.setStringProperty("key", "value");
        textMessage.setText("Hello from Blois's university");
        messageProducer.send(textMessage);
    } catch (JMSException e) {
        // TRAITEMENT DES ERREURS
    }
}
```

235

## Interface Message

- Semblable à un mail composé de trois parties
  - Headers : couples standards nom-valeur (JMSCorrelationID, JMSReplyTo, JMSMessageID, JMSTimeStamp...)
  - Properties : couples libre nom-valeur (primitives, String ou Objet)
  - Body : Contenu du message
- ByteMessage, MapMessage, StreamMessage, TextMessage, ObjectMessage

236

## Intérêt des MDBs

- Multithreading : pool de MDB dont une instance est extraite lors de l'arrivée d'un message
- Simplification du code du consumer

237

## Receveur (interface)

```
@MessageDriven(name = "receiverMDB"
    , activationConfig = {
        @ActivationConfigProperty(
            propertyName = "DestinationType", propertyValue = "javax.jms.Queue"
        ),
        @ActivationConfigProperty(
            propertyName = "DestinationName", propertyValue = "jms/Queue"
        )
    }
)
public class Receiver implements MessageListener {
    @Resource
    private MessageDrivenContext messageDrivenContext;

    @Override
    public void onMessage(Message message) {
        if (message instanceof TextMessage) {
            TextMessage textMessage = (TextMessage) message;
            // do some stuff with message
        }
    }
}
```

239

## Règles de programmation

- doit implémenter une interface MessageListener soit directement (implements) soit indirectement (annotation)
- doit être une classe concrète (ni final, ni abstract)
- doit être un POJO et pas sous classe de MDB
- doit être public
- Constructeur sans argument
- Ne pas lever de RuntimeException (termine l'instance du MDB)

238

## Receveur (annotation)

```
@MessageDriven(name = "receiverMDB"
    , messageListenerInterface = javax.jms.MessageListener.class
    , activationConfig = {
        @ActivationConfigProperty(
            propertyName = "DestinationType", propertyValue = "javax.jms.Queue"
        ),
        @ActivationConfigProperty(
            propertyName = "DestinationName", propertyValue = "jms/Queue"
        )
    }
)
public class Receiver {
    @Resource
    private MessageDrivenContext messageDrivenContext;

    public void onMessage(Message message) {
        if (message instanceof TextMessage) {
            TextMessage textMessage = (TextMessage) message;
            // do some stuff with message
        }
    }
}
```

240

## @ MessageDriven

```
Target({TYPE}) @Retention(RUNTIME)
public @interface MessageDriven {
    String name() default DE;
    Class<?> messageListenerInterface() default Object.class;
    ActivationConfigProperty[] activationConfig() default {};
    String mappedName();
    String description();
}
```

- name : nom du MDB dans l'arbre JNDI
- messageListenerInterface : type de message listener implémenté (sinon, il faut utiliser l'interface)
- activationConfig : propriétés de configuration

241

## Propriété acknowledgeMode

- Confirmation du container d'EJB au serveur JMS que le message a été bien reçu.
- AUTO\_ACKNOWLEDGE : confirmation dès réception du message (mode par défaut)
- DUPS\_OK\_ACKNOWLEDGE : la confirmation peut être envoyée en différé. Le MDB doit être capable de gérer les doublons.

243

## @ActivationConfigProperty

```
public @interface ActivationConfigProperty {
    String propertyName();
    String propertyValue();
}
```

- Les plus communs sont :
  - destinationType : Queue ou Topic
  - connectionFactoryJndiName
  - destinationName
  - acknowledgMode
  - messageSelector
  - subscriptionDurability

242

## Propriété subscriptionDurability

- Pour les MDB écoutant des Topics, on précise la durée de la souscription, c'est à dire si le MOM doit conserver une copie du message en l'absence du Consumer
  - Durable : conservation des messages
  - NonDurable : non conservation des messages (valeur par défaut)

244

## Propriété messageSelector

permet de déclarer un filtre pour le MDB avec une syntaxe proche du WHERE du SQL sur les Headers et Properties du message

Type	Description	Exemple
Littéral	String, numérique ou booléen	Chaine, 100,TRUE
Identifiant	message property ou header name	RECIPIENT, JMSTimestamp, Fragile, ...
<sup>8</sup> Whitespace <sup>8</sup>	cf. JLS (space, tab, form feed, line terminator)	
Opérateurs de comparaison	>, >= , = , < , <= , <>	RECIPIENT= <sup>8</sup> MonMDB <sup>8</sup> NumOfBids>=100
Opérateurs logique	NOT,AND, OR	Condition1 AND Condition2
Comparaison à NULL	IS [NOT] NULL	FirstName IS NOT NULL
Comparaison à TRUE/FALSE	IS [NOT] TRUE, IS [NOT] FALSE	Fragile IS TRUE

245

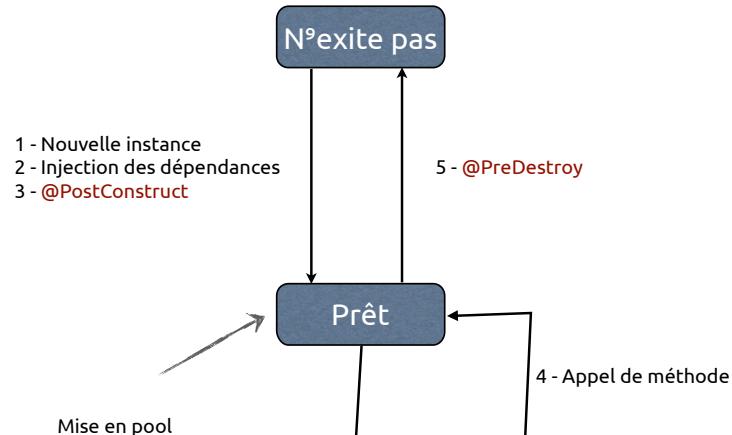
## MDB producteur (1)

```
@MessageDriven(name = "receiverMDB"
    , activationConfig = {@ActivationConfigProperty(propertyName = "messageSelector", propertyValue
    = "newItem > 10"})
public class Receiver implements MessageListener {
    @Resource
    private MessageDrivenContext messageDrivenContext;
    @Resource(name = "jms/QueueConnectionFactory")
    private ConnectionFactory connectionFactory;
    @Resource(name="jms/MessageQueue")
    private Destination destination;
    private Connection connection;
    @Override
    public void onMessage(Message message) {
        if (message instanceof TextMessage) {
            TextMessage textMessage = (TextMessage) message;
            printItemList();
        }
    }
    @PostConstruct
    private void initialization() throws JMSException {
        connection = connectionFactory.createConnection();
    }
    @PreDestroy
    private void tearDown() throws JMSException {
        connection.close();
    }
}
```

247

## Cycle de vie

Semblable au Stateless



246

## MDB producteur (2)

```
public class Receiver implements MessageListener {
    .....
    private void printItemList() throws JMSException {
        Session session = connection.createSession();
        MessageProducer messageProducer = session.createProducer(destination);
        TextMessage textMessage = session.createTextMessage();
        textMessage.setText("Print last items");
        messageProducer.send(textMessage);
        session.close();
    }
}
```

248

## MDBs Recommandations

- Choisir avec attention le modèle de message (PTP ou Pub-Sub)
- Découpler la logique du traitement dans une autre méthode que onMessage voir dans un Session Bean
- Choisir entre multiplier les destinations ou utiliser les filtres
- Choisir le type de message (XML permet le découplage mais augmente la charge)
- Attention aux messages empoisonnés (les MOM permettent de compter les <sup>8</sup>redelivery<sup>8</sup> et basculer sur une <sup>8</sup>dead message queue<sup>8</sup>)
- Dimensionner correctement le pool de MDBs

249

## Quelques définitions

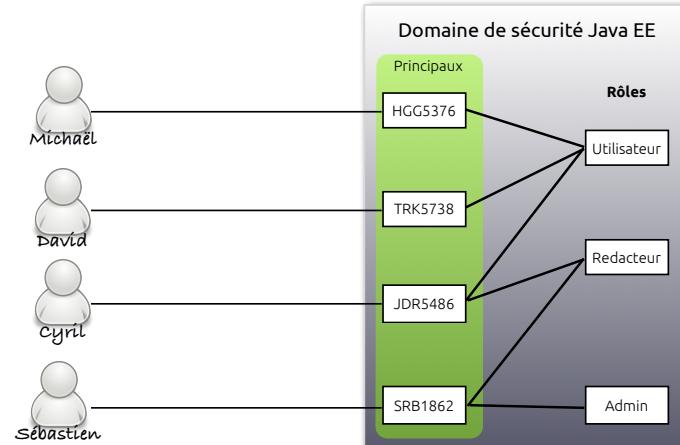
- Authentication : Vérification de l'identité de l'utilisateur
- Authorization : détermination si un utilisateur est autorisé à accéder à une ressource ou méthode
- User : personne reconnue par Authentication et associée à un Principal
- Groupe : regroupement logique de User au niveau du serveur
- Role : regroupement logique de User au niveau de l'application (peut être équivalent des groupes)
- Realm : scope sur lequel s'applique une politique de sécurité
- JAAS : Java Authentication and Authorization Service

251

## EJB & Sécurité

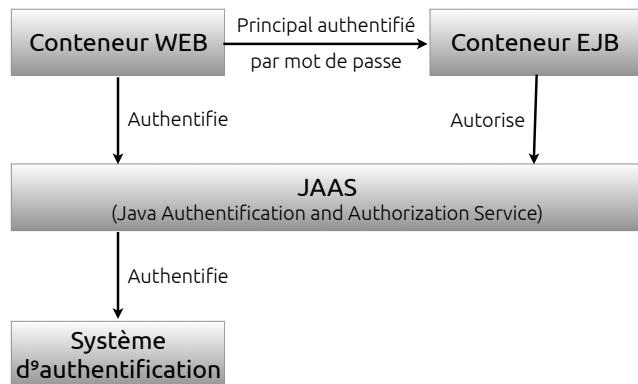
250

## Principal & Rôle



252

## Principe de fonctionnement



253

## Support de la sécurité

- Les EJB gèrent les autorisations
- Les clients gèrent l'authentification
  - Principal et rôle transmis aux EJB
- Sécurité déclarative
- Sécurité programmatique

254

## Sécurité déclarative

```
@Stateless  
@RolesAllowed({"admin", "writer", "user"})  
public class ActionEJB {  
  
    @RolesAllowed("admin")  
    public void restrictedAction() {  
        // some stuff  
    }  
  
    @PermitAll  
    public void action() {  
        // some stuff  
    }  
}
```

255

## Annotation de sécurité

Annotation	Bean	Méthode	Description
@PermitAll	✓	✓	Le bean ou la méthode est accessible à tous les rôles déclarés
@DenyAll	✓	✗	Aucun rôle n'est autorisé à exécuter
@RolesAllowed	✗	✓	Donne la liste des rôles autorisés
@DeclaredRoles	✓	✗	Définition des rôles de sécurité
@RunAs	✗	✓	Affecte temporairement un nouveau rôle au principal

256

## Sécurité programmatique

```
@Stateless  
public class ActionEJB {  
  
    @Resource  
    private SessionContext sessionContext;  
  
    public void restrictedAction() {  
        if (sessionContext.isCallerInRole("admin")) {  
            // do some stuff  
        } else {  
            throw new SecurityException("Only admin can do that");  
        }  
    }  
  
    public void action() {  
        if (sessionContext.isCallerInRole("writer")) {  
            // do some writer stuff  
        } else if (sessionContext.getCallerPrincipal().getName().equals("Cyril")) {  
            // do some stuff for Cyril  
        }  
        // do global stuff  
    }  
}
```

257

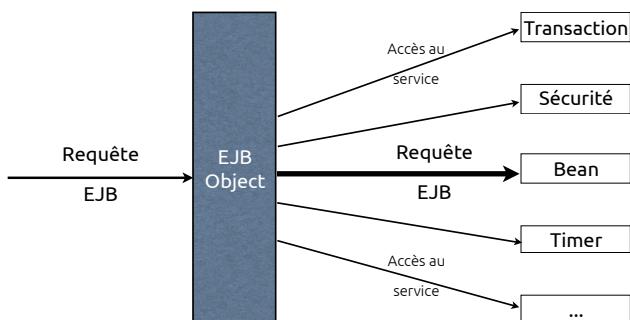
## EJB <sup>8</sup>forward<sup>9</sup>

EJB Context  
Managed bean  
CDI et @Resource  
Intercepteur

258

## EJB en coulisse

Au déploiement, le container génère un proxy appelé <sup>8</sup>EJB Object<sup>8</sup> qui accède à toutes les fonctionnalités du container



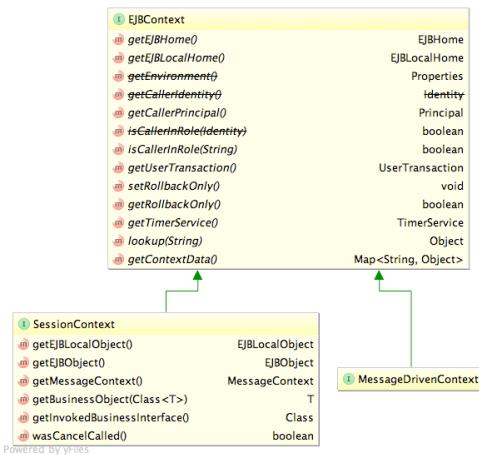
259

## EJB Context

```
public interface EJBContext {  
    EJBHome getEJBHome() throws IllegalStateException;  
    EJBLocalHome getEJBLocalHome() throws IllegalStateException;  
    Principal getCallerPrincipal() throws IllegalStateException;  
    boolean isCallerInRole(String s) throws IllegalStateException;  
    UserTransaction getUserTransaction() throws IllegalStateException;  
    void setRollbackOnly() throws IllegalStateException;  
    boolean getRollbackOnly() throws IllegalStateException;  
    TimerService getTimerService() throws IllegalStateException;  
    Object lookup(java.lang.String s) throws IllegalArgumentException;  
    Map<String, Object> getContextData();  
}
```

260

## EJB Context



261

## EJB Context - Utilisation

```
@Stateless  
public class MonSessionBean implement MonSession{  
    @Resource SessionContext sessionContext;  
    ..... }  
  
@MessageDriven  
public class MonMDB implements MessageListener {  
    @Resource MessageDrivenContext messageDrivenContext;  
    .... }
```

262

## Managed bean

- Les <sup>8</sup>bean managé<sup>8</sup> sont des POJO gérés par le container
- Modèle léger de composant
- Support d<sup>9</sup>un petit ensemble de services
  - Injection (@Resource, @Inject)
  - Contrôle de cycle de vie (@PostConstruct & @PreDestroy)
  - Intercepteur (@Interceptor, @AroundInvoke)

263

## Managed bean Vs EJB

Services	EJB	Managed bean
Injection de dépendance		
Intercepteur		
Cycle de vie		
Client distant		
Gestion d <sup>9</sup> état et pool		
Message		
Transaction & sécurité		
Concurrence		
Invocation Asynchrone		

264

## Managed bean - Exemple

```
@ManagedBean  
public class RandomId {  
    private Random randomGenerator;  
    @PostConstruct  
    private void initialization() {  
        randomGenerator = new Random(System.currentTimeMillis());  
    }  
    @PreDestroy  
    private void tearDown() {  
        // some stuff before destroy  
    }  
    @Interceptors(LoggingInterceptor.class)  
    public String generateRandomId() {  
        return "ID" + randomGenerator.nextInt();  
    }  
}
```

265

## Managed bean - Utilisation

```
public class CallRandomId {  
    @Resource  
    private RandomId randomId;  
  
    public void call() {  
        String id = randomId.generateRandomId();  
    }  
}
```

266

## CDI

- Contrôle de
  - Résolution de dépendance (DI)
  - Cycle de vie
  - Configuration
- A destination d'un composant externe
- Couplage faible

267

## Ce bon vieux new

```
public class CallRandomId {  
    private RandomId randomId;  
    public CallRandomId() {  
        randomId = new RandomId();  
    }  
    public UserId call() {  
        UserId userId = new UserId();  
        userId.setId(randomId.generateRamdomId());  
        return userId;  
    }  
}  
  


- Couplage fort
  - Impossibilité de changer l'implémentation
  - Mock difficile
- Cycle de vie géré par le composant
  - new par forcement suffisant
  - Création d'une instance, ouverture, fermeture

```

268

## Pourquoi DI

- Découpler la dépendance entre composant
  - <sup>8</sup>loose coupling<sup>8</sup>
- Principe d<sup>9</sup>Hollywood
  - <sup>8</sup>don<sup>9</sup>t call us, we<sup>9</sup>ll call you...<sup>8</sup>
- Injecter tout, partout
- Fonctionne dans un container EE
- ... et aussi ailleurs :)

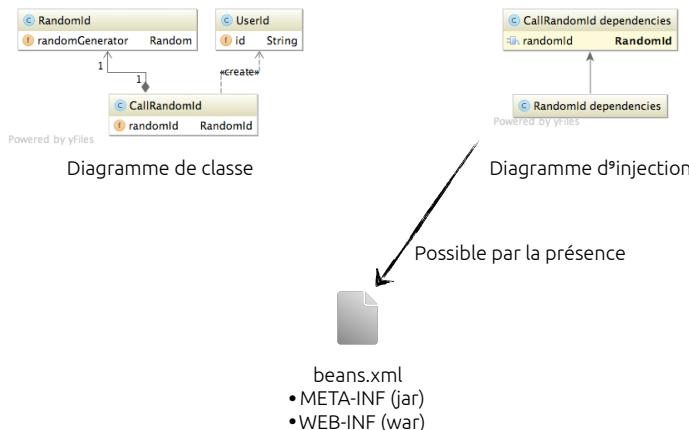
269

## Exemple CDI

```
public class CallRandomId {  
    @Inject  
    private RandomId randomId;  
  
    public UserId call() {  
        UserId userId = new UserId();  
        userId.setId(randomId.generateRandomId());  
        return userId;  
    }  
  
    public class RandomId {  
        private Random randomGenerator;  
        @PostConstruct  
        private void initialization() {  
            randomGenerator = new Random(System.currentTimeMillis());  
        }  
        @PreDestroy  
        private void tearDown() {  
            // some stuff before destroy  
        }  
        @Interceptors(LoggingInterceptor.class)  
        public String generateRandomId() {  
            return "ID" + randomGenerator.nextInt();  
        }  
    }  
}
```

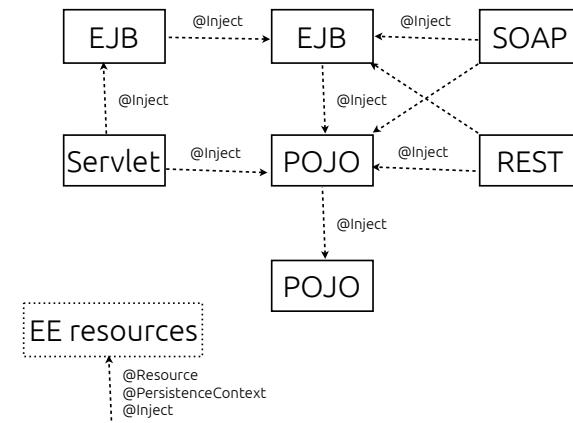
270

## CDI diagramme classe et injection



271

## Injection de dépendance



272

## Injection de ressource

```
@Target({TYPE, METHOD, FIELD})
@Retention(RUNTIME)
public @interface Resource {
    public enum AuthenticationType {CONTAINER, APPLICATION}
    String name() default LM;
    Class type() default Object.class;
    AuthenticationType authenticationType() default AuthenticationType.CONTAINER;
    boolean shareable() default true;
    String mappedName() default LM;
    String description () default LM;
}



- name : nom pour l'arbre JNDI
- type : type de la ressource. Par défaut c'est celle de l'attribut, pour une méthode c'est celle de la propriété, pour une classe il faut préciser
- authenticationType : type d'authentication pour utiliser la ressource, peut être précisée pour une Connection Factory, sinon ne pas préciser
- shareable : indique si la ressource peut être partagée entre composant, peut être préciser pour une Connection Factory, sinon ne pas préciser

```

273

## Injection de ressource

- L'annotation @Resource peut être positionnée sur l'attribut ou sur le setter s'il respecte la convention de nommage des JavaBeans
- l'utilisation sur les setters peut paraître plus de travail mais elle peut simplifier l'écriture de tests unitaires et permettre de se passer de méthode annotée @PostConstruct
- Utilisé pour injecter des ressources JavaEE
  - Ressource JMS (Queue, Topic, QueueConnectionFactory, TopicConnectionFactory)
  - EJB Context
  - E-mail ressource (javax.mail.Session)
  - TimerService
  - Paramétrage

274

## @Resource et paramétrage

- @Resource peut être utilisée pour injecter des variables d'environnement
- dans le code :

```
@Resource
private boolean isProduction
```
- Paramétrage dans le fichier ejb-jar.xml

```
<env-entry>
<env-entry-name>isProduction</env-entry-name>
<env-entry-type>java.lang.Boolean</env-entry-type>
<env-entry-value>true</env-entry-value>
</env-entry>
```

275

## Intercepteur & AOP

- Objectif de l'AOP : gestion des aspects transverses du code comme le logging, profiling, auditing dans un module indépendant
- EJB3 permet d'intercepter les appels aux méthodes business

```
@Stateless
public class UserEJB {
    @Interceptors(UserLogger.class)
    public User getUser(long userId) {
        // retrieve user by Id
        return user;
    }
}

public class UserLogger {
    @AroundInvoke
    public Object logSomething(InvocationContext invocationContext) throws Exception {
        System.out.println(invocationContext.getMethod().getName() + " appel");
        return invocationContext.proceed();
    }
}
```

276

## @Interceptors

- Peut être utilisé au niveau de la classe ou des méthodes
- possibilité de cumuler les intercepteurs
- Exécution du scope le plus large au plus fin  
@Interceptors({ApplicationLogger.class, ApplicationStatistic.class})
- Définition d'un intercepteur par défaut:  

```
<assembly-descriptor>
    <interceptor-binding>
        <ejb-name></ejb-name>
        <interceptor-class>
            org.organisme.application.DefaultInterceptor
        </interceptor-class>
    </interceptor-binding>
</assembly-descriptor>
```

277

## JavaServerFaces

279

## Exclusion

- Deux annotations permettent d'exclure des intercepteurs
- @ExcludeDefaultInterceptor au niveau des classes ou des méthodes pour exclure l'intercepteur par défaut
- @ExcludeClassInterceptor au niveau des méthodes pour exclure l'intercepteur déclaré au niveau de la classe

## Implémentation

- L'intercepteur doit toujours avoir une méthode annotée @AroundInvoke avec la signature :  

```
public Object methodName(InvocationContext invocationContext) throws Exception
```
- On doit appeler la méthode interceptée en retournant son résultat  

```
return invocationContext.proceed();
```
- On peut lever une exception dans le corps de la méthode et ne pas appeler la méthode ciblée

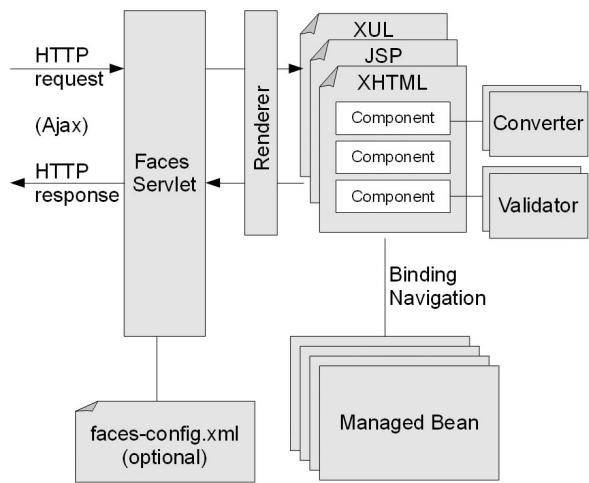
278

## Quelques rappels

- HTML (Hyper Text Markup Language) protocole déconnecté
- Session et cookies
- 4 scopes :
  - application
  - session
  - request
  - page

280

## JSF : les briques...



Beginning Java™ EE 6 Platform with GlassFish™ 3 - A.Goncalves - Ed APRESS

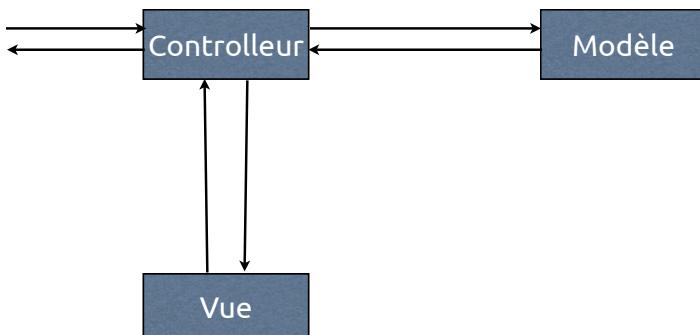
281

## JSF : définition

- API décrivant des composants graphiques (UIComponent) et permettant :
  - gestion d'état
  - gestion d'événements
  - validation
  - conversion
  - gestion de la navigation
- Il est possible d'étendre ses fonctionnalités

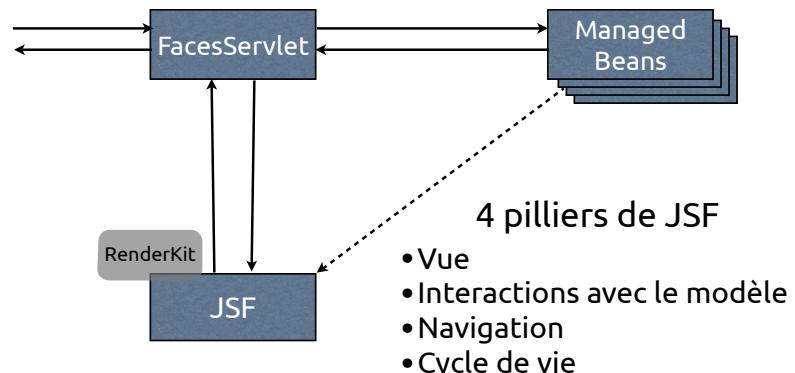
282

## MVC : rappel



283

## MVC : JSF



284

- 4 piliers de JSF
- Vue
  - Interactions avec le modèle
  - Navigation
  - Cycle de vie

## Vue

- Pages :
  - JSP avec syntaxe HTML ou XML
  - Facelets avec syntaxe XHTML
- Imbrication de tags représentant les UIComponents
- Permet la construction d'un arbre de composants (component tree)



285

## Exemple JSP non XML



```
<%@page contentType="text/html" %>
<%@taglib uri="http://xmlns.jcp.org/jsf/core" prefix="f" %
<%@taglib uri="http://xmlns.jcp.org/jsf/html" prefix="h" %
<f:view>
    <html><body>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel for="username" value="Login" />
                <h:inputText id="username" size="12" required="true" />
                <h:outputLabel for="password" value="Password" />
                <h:inputText id="password" size="8" required="true" />
                <h:commandButton id="btnLogIn" value="Connection" />
            </h:panelGrid>
        </h:form>
    </body></html>
</f:view>
```

286

## Exemple JSP XML



```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
    xmlns:h="http://xmlns.jcp.org/jsf/html">
    <jsp:directive.page contentType="text/html;charset=UTF-8" />
    <f:view>
        <html><body>
            <h:form>
                <h:panelGrid columns="2">
                    <h:outputLabel for="username" value="Login" />
                    <h:inputText id="username" size="12" required="true" />
                    <h:outputLabel for="password" value="Password" />
                    <h:inputText id="password" size="8" required="true" />
                    <h:commandButton id="btnLogIn" value="Connection" />
                </h:panelGrid>
            </h:form>
        </body></html>
    </f:view></jsp:root>
```

287

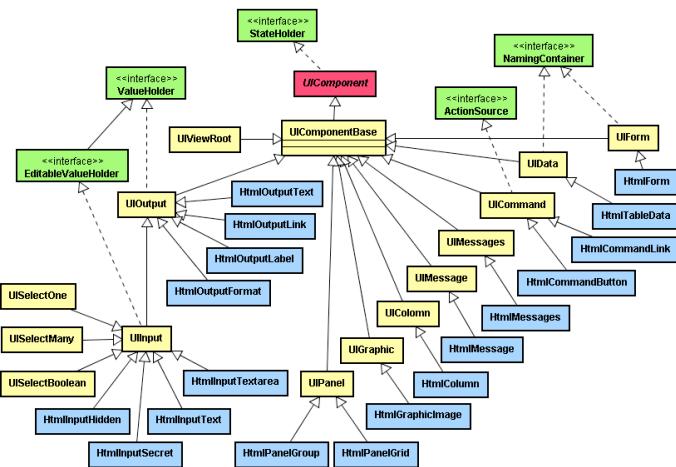
## Exemple Facelets XHTML



```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xml:lang="en" lang="en">
    <body>
        <f:view>
            <h:form>
                <h:panelGrid columns="2">
                    <h:outputLabel for="username" value="Login" />
                    <h:inputText id="username" size="12" required="true" />
                    <h:outputLabel for="password" value="Password" />
                    <h:inputText id="password" size="8" required="true" />
                    <h:commandButton id="btnLogIn" value="Connection" />
                </h:panelGrid>
            </h:form>
        </f:view>
    </body>
</html>
```

288

## Les Composant (UIComponent)



289

## Composant : Action

- **h:commandButton** génère un élément HTML input de type submit qui soumet le formulaire et déclenche une action
- **h:commandLink** génère un élément HTML Link qui se comporte comme un commandButton
- **h:link** génère un lien
- **h:button** génère un input de type bouton

291

## Namespace des composants

URI	Prefixe (commun)	Description
http://xmlns.jcp.org/jsf/html	<b>h</b>	Composant HTML (ex: h:commandButton, h:inputText, etc...)
http://xmlns.jcp.org/jsf/core	<b>f</b>	Action *custom* indépendante de tout kit de rendu (ex: f:selectItem, f:validateLength, etc...)
http://xmlns.jcp.org/jsf/faceslets	<b>ui</b>	Support de template
http://xmlns.jcp.org/jsf/composite	<b>composite</b>	Support pour les composants *personnalisés*

290

## Composant : Input

- **h:inputHidden** envoie un élément HTML input de type hidden
- **h:inputSecret** envoie un élément HTML input de type password
- **h:inputText** envoie un élément HTML input de type texte
- **h:inputTextarea** renvoie un élément HTML input de type TextArea

292

## Composant : Output

- **h:outputLabel** affiche sa valeur sous la forme d'un élément label ou permet l'affichage de la valeur d'un composant donné en paramètre
- **h:outputText** affiche un texte simple
- **h:outputLink**
- **h:outputFormat** affiche un texte paramétré
  - **f:param** paramètres passés au format (dans l'ordre de ceux-ci)
- **h:graphicImage** affiche une image en renvoyant une balise `<img>`

293

## Composant : Table

- **h:panelGrid** retourne un élément HTML table
- **h:dataTable** retourne un élément HTML de type table rempli en itérant sur un DataModel où chaque objet de la collection représente une ligne
- **h:column** représente une colonne à l'intérieur d'un composant UIData (comme h:dataTable)

295

## Composant : Selection

- **h:selectOneMenu** génère une selectBox
- **h:selectBooleanCheckbox** UIInput pour les booléens qui génère un élément HTML input de type checkbox
- **h:selectOneListBox**, **h:selectOneMenu**, **h:selectManyListBox**, **h:selectManyMenu** renvoient des listes d'options à choix unique ou multiple et sous forme de liste ou de menu
- **h:selectOneRadio** renvoie un input de type radio
- **h:selectManyCheckbox** renvoie une liste de checkbox
- hormis **h:selectBooleanCheckbox** tous ces tags utilisent en enfants des **f:selectItem** ou **f:selectItems**

294

## Composant : message

- **h:message** affiche le premier message lié à son composant parent
- **h:messages** affiche tous les messages du FacesContext courant

296

## Composant : divers

- **h:body** équivalent du body HTML
- **h:head** équivalent du head HTML
- **h:form** équivalent du formulaire HTML
- **h:panelGroup** permet de regrouper plusieurs UIComponent dans un parent qui ne peut recevoir qu'un enfant (colonne de tableau, facet)

297

## tag JSF : les attributs HTML

- alt : texte alternatif si le composant ne s'affiche pas
- border : bordure du composant
- disabled : booléen permettant la désactivation d'un composant de saisie ou bouton
- maxlength : maximum de caractères pour un composant texte
- readonly : mode de lecture uniquement
- size : taille d'un champs texte
- style : information du style
- target : nom de la frame où le document est ouvert

299

## tag JSF : les attributs

- **id : identifiant du composant**
- binding : association à un backing bean
- rendered : booléen déterminant si le composant s'affiche
- styleClass : classe CSS à appliquer au composant
- value : valeur du composant
- valueChangeListener : associe une méthode appelée en cas de changement de valeur
- converter : classe utilisée pour la conversion de valeur
- validator : classe utilisée pour la validation de donnée
- required : booléen indiquant si la valeur est obligatoire

298

## tag JSF : les attributs DHTML

- onblur : l'élément perd le focus
- onClick : clique souris sur l'élément
- ondblClick : double clique souris sur l'élément
- onfocus : l'élément gagne le focus
- onkeydown, onkeyup : touche clavier enfoncée et relachée
- onkeypress : touche clavier pressée puis relachée
- onmousedown, onmouseup : bouton souris enfoncé et relaché
- onmouseout, onmouseover : curseur souris sort et entre
- onreset : formulaire réinitialisé
- onselect : texte sélectionné dans un champs texte
- onsubmit : formulaire soumis

300

## Interactions avec le modèle

- Utilisation de EL pour associer les UIComponent à des propriétés du modèle
- La conversion se fait automatiquement
- Un validation côté serveur peut être faite
- Le modèle ne contient que des données converties et validées
- Vous êtes responsable de la persistance du modèle
- Le modèle est constitué de POJO, managed-bean ou backing-beans
- EL fonctionne dans les deux sens : lecture et écriture

301

## Les managed beans

- C'est l'<sup>98</sup>Inversion of Control<sup>8</sup> de JSF
- Les managed beans sont créés au besoin lors de l'accès et placé dans le scope approprié
- On peut utiliser l'injection Java EE dans les managed beans (@Inject, @Resource, @PostConstruct, @PreDestroy)

303

## Interactions avec le modèle

- EL dispose de quelques objets implicites : cookie, facesContext, header, headerValues, param, paramValues, request, requestScope, view, application, applicationScope, initParam, session, sessionScope
- L'accès est simple : #{propriété}
  - #{requestScope.user.name}
- On peut facilement créer de nouveaux objet implicites
- S'appuie sur les conventions de nommage JavaBeans
- EL peut aussi pointer vers des méthodes

302

## Managed bean exemple (java)

```
public class Login {  
    private String login;  
  
    private String password;  
  
    public Login() {  
    }  
  
    public String checkLogin() {  
        if ("guest".equals(login) && "guest".equals(password)) {  
            return "login_success";  
        } else {  
            return "login_failed";  
        }  
    }  
    // getter et setter  
}
```

304

## Managed bean exemple (jsf)

```
login.xhtml
<h:form>
    <h:panelGrid columns="2">
        <h:outputLabel for="username" value="login"/>
        <h:inputText id="username" size="2" required="true" value="#{login.login}" />
        <h:outputLabel for="password" value="password" />
        <h:inputSecret id="password" size="4" required="true" value="#{login.password}" />
        <h:commandButton id="btnLogIn" value="Connection" action="#{login.checkLogin}" />
    </h:panelGrid>
</h:form>
```

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/
web-facesconfig_2_2.xsd">
<managed-bean>
    <managed-bean-name>login</managed-bean-name>
    <managed-bean-class>org.univ.blois.web.Login</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
</faces-config>
```

305

## Managed bean exemple (java) binding

```
public class Login {
    private HtmlInputText login;
    private HtmlInputSecret password;

    public Login() {
    }

    public String checkLogin() {
        if ("guest".equals(login.getValue()) && "guest".equals(password.getValue())) {
            return "login_success";
        } else {
            return "login_failed";
        }
    }

    // getter et setter
}
```

307

## Composant ou binding

- On n'utilise plus l'attribut value mais l'attribut binding
- Les attributs du Managed Bean ne sont plus des objets du domaine mais des UIComponent
- Recommandé lors d'une utilisation de type un Backing Bean par page
- *Ne déclarer un UIComponent que lorsque l'on a besoin d'y accéder*
- L'origine est une similarité avec ASP.NET et Visual Basic pour attirer ces développeurs

306

## Managed bean exemple (jsf) binding Old flavor

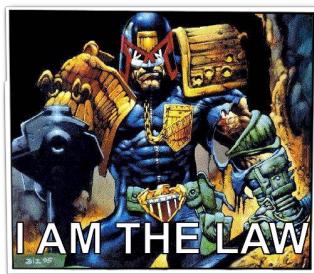
```
login.xhtml
<h:form>
    <h:panelGrid columns="2">
        <h:outputLabel for="username" value="login"/>
        <h:inputText id="username" size="2" required="true" binding="#{login.login}" />
        <h:outputLabel for="password" value="password" />
        <h:inputSecret id="password" size="4" required="true" binding="#{login.password}" />
        <h:commandButton id="btnLogIn" value="Connection" action="#{login.checkLogin}" />
    </h:panelGrid>
</h:form>
```

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/
web-facesconfig_2_2.xsd">
<managed-bean>
    <managed-bean-name>login</managed-bean-name>
    <managed-bean-class>org.univ.blois.web.Login</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
</faces-config>
```

308

## Petite remarque sur les IDs

- Les identifiants sont facultatifs car facelet les génèrent automatiquement
- Mais ils sont mouvants en fonction de l'arbre DOM
- C'est un vrai problème pour la mise en place des tests d'interface.
- La bonne pratique : fixer un identifiant



**RESPONSABLE DES  
TESTS**

309

## Les validateurs

- Permet de s'assurer que les données répondent à des contraintes
- Il existe des validateurs standards mais on peut aussi créer des validateurs répondant à des contraintes métier et/ou fonctionnelles
- **Les validateurs standards sont :**  
LongRangeValidator, DoubleRangeValidator, LengthValidator, "required"
- **Les erreurs de validation s'affichent dans les balises**  
`<h:messages> OU <h:message>`

311

## Hands on !!

1. Création d'une calculatrice
2. En utilisant (`panelGrid` optionnel), `inputText`, `selectOneMenu` faire en sorte de calculer les 4 opérations sur 2 valeurs en affichant le résultat dans un `input box` en lecture seule.
  1. Pensez à respecter les règles de base des mathématique :)
  3. Puis Passer en Ajax.

310

## Validation du login Exemple

```
login.xhtml
<h:form>
  <h:panelGrid columns="2">
    <h:outputLabel for="username" value="login"/>
    <h:inputText id="username" size="2" required="true" value="#{login.login}" >
      <f:validateLength maximum="10" minimum="5" />
    </h:inputText>
    <h:outputLabel for="password" value="password" />
    <h:inputSecret id="password" size="4" required="true" value="#{login.password}" />
    <h:commandButton id="btnLogIn" value="Connection" action="#{login.checkLogin}" />
    <h:message for="username" />
  </h:panelGrid>
</h:form>
```

312

## Validateur spécialisé

- Deux possibilités
  - Implémentation dans le backing bean
  - Dans une classe dédiée et déclarée en tant que validateur dans le face-config.xml

313

## Validateur spécialisé Classe spécialisée - old flavor

```
....  
import javax.faces.validator.Validator;  
import javax.faces.validator.ValidatorException;  
...  
public class LoginValidator implements Validator {  
  
    @Override  
    public void validate(FacesContext context, UIComponent component, Object value) throws  
    ValidatorException {  
        String componentValue = value.toString();  
        if (componentValue.length() < 5) {  
            throw new ValidatorException(new FacesMessage("le login est trop court by  
LoginValidator"));  
        } else if (componentValue.length() > 10) {  
            throw new ValidatorException(new FacesMessage("le login est trop long by  
LoginValidator"));  
        }  
    }  
}
```

315

## Validateur spécialisé Backing bean

```
login.xhtml  
....  
<h:form>  
    <h:panelGrid columns="2">  
        <h:outputLabel for="username" value="login"/>  
        <h:inputText id="username" size="2" required="true" value="#{login.login}"  
            validator="#{login.validateLogin}"/>  
    </h:inputText>  
    <h:outputLabel for="password" value="password" />  
    <h:inputSecret id="password" size="4" required="true" value="#{login.password}" />  
    <h:commandButton id="btnLogIn" value="Connection" action="#{login.checkLogin}" />  
    <h:message for="username" />  
</h:panelGrid>  
</h:form>
```

```
login.java  
....  
public class Login {  
....  
    public void validateLogin(FacesContext facesContext, UIComponent component, Object value)  
        throws ValidatorException {  
        ....  
        String componentValue = value.toString();  
        if (componentValue.length() < 5) {  
            throw new ValidatorException(new FacesMessage("le login est trop court"));  
        } else if (componentValue.length() > 10) {  
            throw new ValidatorException(new FacesMessage("le login est trop long"));  
        }  
    }  
....  
}
```

314

## Validateur spécialisé Classe spécialisée - old flavor

```
login.xhtml  
....  
<h:form>  
    <h:panelGrid columns="2">  
        <h:outputLabel for="username" value="login"/>  
        <h:inputText id="username" size="2" required="true" value="#{login.login}">  
            <f:validator validatorId="loginLength"/>  
        </h:inputText>  
        <h:outputLabel for="password" value="password" />  
        <h:inputSecret id="password" size="4" required="true" value="#{login.password}" />  
    </h:panelGrid>  
</h:form>
```

```
faces-config.xml  
....  
<?xml version='1.0' encoding='UTF-8'?>  
<faces-config version="2.2"  
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/  
    web-facesconfig_2_2.xsd">  
....  
    <validator>  
        <validator-id>loginLength</validator-id>  
        <validator-class>org.univ.blois.web.LoginValidator</validator-class>  
    </validator>  
....  
</faces-config>
```

316

## Validateur spécialisé Classe spécialisée - annotation

```
...
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;
...
@FacesValidator("loginLength")
public class LoginValidator implements Validator {
    ...
    @Override
    public void validate(FacesContext context, UIComponent component, Object value) throws ValidatorException {
        String componentName = value.toString();
        if (componentName.length() < 5) {
            throw new ValidatorException(new FacesMessage("le login est trop court by LoginValidator"));
        } else if (componentName.length() > 10) {
            throw new ValidatorException(new FacesMessage("le login est trop long by LoginValidator"));
        }
    }
}
```

317

## Les convertisseurs

- Chaque valeur envoyée ou reçue est une String
- Les objets du domaine contiennent des nombres, dates....
- Il existe des convertisseurs standards mais on peut aussi en créer.
- Les convertisseurs implicites standards :
   
BigDecimalConverter, BigIntegerConverter, BooleanConverter, ByteConverter, CharacterConverter, DoubleConverter, FloatConverter, IntegerConverter, LongConverter, ShortConverter
- Deux convertisseurs explicites standards :
   
DateTimeConverter et NumberConverter
- Les erreurs de conversion s'affichent dans les balises
   
`<h:messages> OU <h:message>`

319

## Validateur spécialisé Classe spécialisée - annotation

```
login.xhtml
<h:form>
    <h:panelGrid columns="2">
        <h:outputLabel for="username" value="login"/>
        <h:inputText id="username" size="2" required="true" value="#{login.login}">
            <f:validator validatorId="loginLength"/>
        </h:inputText>
        <h:outputLabel for="password" value="password" />
        <h:inputSecret id="password" size="4" required="true" value="#{login.password}" />

        <h:commandButton id="btnLogIn" value="Connection" action="#{login.checkLogin}" />
    </h:panelGrid>
</h:form>

faces-config.xml
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">
    ...
    <validator>
        <validator-id>loginLength</validator-id>
        <validator-class>org.javblois.web.LoginValidator</validator-class>
    </validator>
    ...
</faces-config>
```

318

## convertisseur explicite Exemple

```
ironman.xhtml
<h:panelGrid columns="2">
    <h:outputLabel for="username" value="nom"/>
    <h:outputLabel id="username" value="#{ironman.name}">
        <h:outputLabel for="username" value="Date de naissance"/>
        <h:outputLabel id="dateOfBirth" value="#{ironman.dateOfBirth}">
            <f:convertDateTime pattern="dd MMM YYYY" locale="fr"/>
        </h:outputLabel>
    </h:panelGrid>
```

```
IronMan.java
public class IronMan {
    private String name;
    private String phoneNumber;
    private Date dateOfBirth;
    public IronMan() {}
    ...
}
```



320

## convertisseur spécialisé

- Implémentation d'un classe dédié :
- Classe implémentant javax.faces.convert.Converter
- Surcharger les méthodes
  - getAsObject & getAsString
- Identification
  - Old flavor : dans le face-config.xml
  - Par annotation
    - @[javax.faces.convert]FacesConverter
- Lier dans la page jsf via <f:converter . . . />

321

## convertisseur spécialisé Annotation

```
import javax.faces.convert.Converter;
import javax.faces.convert.FacesConverter;

@FacesConverter("phoneNumberConverter")
public class PhoneConverter implements Converter{

    @Override
    public Object getAsObject(FacesContext context, UIComponent component, String value) {
        // conversion inverse si (page web vers bean)
    }

    @Override
    public String getAsString(FacesContext context, UIComponent component, Object value) {
        String realPhone = value.toString();
        return "(" + realPhone.substring(0, 3) + " ) " +
            realPhone.substring(3, 6) + "-" +
            realPhone.substring(7);
    }
}
```



323

## convertisseur spécialisé old flavor

```
<h:panelGrid columns="2">
    <h:outputLabel for="username" value="nom"/>
    <h:outputText id="username" value="#{ironman.name}"/>
    <h:outputLabel for="username" value="Date de naissance"/>
    <h:outputText id="dateOfBirth" value="#{ironman.dateOfBirth}">
        <f:convertDateTime pattern="dd MMM YYYY" locale="fr"/>
    </h:outputText>
    <h:outputLabel for="" value="téléphone" />
    <h:outputText id="phoneNumber" value="#{ironman.phoneNumber}">
        <f:converter converterId="phoneNumberConverter" />
    </h:outputText>
</h:panelGrid>
```

ironman.xhtml

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/
    web-facesconfig_2_2.xsd">
    ...
    <converter>
        <converter-id>phoneNumberConverter</converter-id>
        <converter-class>org.univ.blois.web.PhoneConverter</converter-class>
    </converter>
    ...
</faces-config>
```

faces-config.xml

322

## La navigation

- JSF propose un système de navigation flexible basé sur des **outcome** et décrit dans le fichier de configuration (faces- config.xml ou autre)
- Chaque composant ActionSource donne un **outcome** qui peut être en dur (navigation statique) ou renvoyé par une méthode appelée via EL (navigation dynamique)
- Il n'y a pas de limite au nombre d'ActionSource dans la page
- un **outcome** null renvoie la même page
- Si aucun outcome correspond, la même page est renvoyée

324

## La navigation Exemple

```
<h:form>
<h:panelGrid columns="2">
    <h:outputLabel for="username" value="login"/>
    <h:inputText id="username" size="2" required="true" value="#{login.login}">
        <f:validator validatorId="loginLength"/>
    </h:inputText>
    <h:outputLabel for="password" value="password" />
    <h:inputSecret id="password" size="4" required="true" value="#{login.password}" />
    <h:commandButton id="btnLogin" value="Connection" action="#{login.checkLogin}" />
    <h:commandLink id="goodbye" value="m'en aller..." action="goodbye" />
    <h:message for="username" />
</h:panelGrid>
</h:form>
```



325

## La navigation Exemple

```
public class Login {
    private String login;
    private String password;

    public Login() {
    }

    public String checkLogin() {
        if ("guest".equals(login) && "guest".equals(password)) {
            return "login_success";
        } else {
            return "login_failed";
        }
    }
    ....
}
```

326

## La navigation Exemple

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/
    web-facesconfig_2_2.xsd">
...
<navigation-rule>
    <from-view-id>/login.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>login_success</from-outcome>
        <to-view-id>ironman.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>login_failed</from-outcome>
        <to-view-id>loginfailed.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>goodbye</from-outcome>
        <to-view-id>goodbye.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
...
</faces-config>
```

## La navigation Lien globaux

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/
    web-facesconfig_2_2.xsd">
...
<navigation-rule>
    <from-view-id>*</from-view-id>
    <navigation-case>
        <from-outcome>loginOut</from-outcome>
        <to-view-id>logout.xhtml</to-view-id>
    </navigation-case>
    .....
</navigation-rule>
...
</faces-config>
```

327

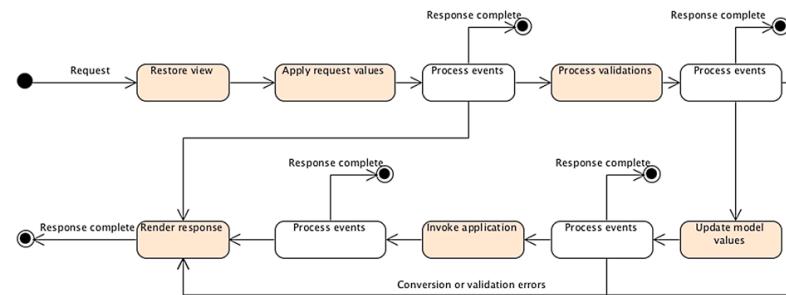
328

## Cycle de vie JSF

- Il définit comment les requêtes sont gérées et les réponses renvoyées
- en rapport avec :
  - trouver la vue concernée
  - associer les valeurs aux composants
  - s'assurer de la conversion et validation des données
  - s'assurer que les listeners sont appelés
  - mettre à jour le modèle
  - sélectionner et rendre la vue réponse

329

## Cycle de vie JSF



Beginning Java™ EE 7 - A.Goncalves - Ed APRESS - 2013

330

## Cycle de vie JSF

### Restore view

- Construction d'une vue de la page sous forme d'arbre de composants et branchement des validateurs et event listeners, l'ensemble est ensuite sauvé dans le FacesContext
- deux cas :
  - requête initiale : la vue est laissée vide, et l'on saute en phase 6
  - postback : la vue est peuplée à partir des données sauvees chez le client ou le serveur

331

## Cycle de vie JSF

### Apply Requests

- Applique les valeurs de la requête sur l'arbre de composant
- Les éventuels messages d'erreur sont placés dans le FacesContext et entraîne un passage en phase 6
- Les éventuels événements sont stockés pour être traités avant la phase suivante
- Une redirection vers une autre application ou page sans composant JSF peut être faite par `FacesContext.responseComplete()`

332

## Cycle de vie JSF

### Process Validations

- Phase durant laquelle les validateurs sont appelés
- Les éventuels messages d'erreur de validation sont placés dans le FacesContext et entraîne un passage en phase 6
- Les éventuels événements sont stockés pour être traités avant la phase suivante
- Une redirection vers une autre application ou page sans composant JSF peut être faite par `FacesContext.responseComplete()`

333

## Cycle de vie JSF

### Update Model Values

- Copie des valeurs de l'arbre de composants vers les objets du modèle
- Seule les propriétés pointées par des champs de type INPUT sont mises à jour
- Les éventuels messages d'erreur de conversions sont placés dans le FacesContext et entraîne un passage en phase 6
- Les éventuels événements sont stockés pour être traités avant la phase suivante
- Une redirection vers une autre application ou page sans composant JSF peut être faite par `FacesContext.responseComplete()`

334

## Cycle de vie JSF

### Invoke Application

- Les événements applicatifs tels que soumission de formulaire ou clic sur un lien sont traités
- C'est ici qu'est faite la résolution de la page suivante
- Les événements sont de deux natures :
  - issus de l'attribut `action` de `<h:commandLink>` ou `<h:commandButton>`
  - issus de l'attribut `actionListener`

335

## Cycle de vie JSF

### Render Response

- JSF délègue le rendu de la page au container de JSP si les pages sont des JSP
- Si c'est une requête initiale, l'arbre est vide et les composants sont ajoutés au fur et à mesure
- Après rendu de la vue, l'état de l'arbre est sauvegardé pour le prochain cycle

336

## Cycle de vie JSF

### Immediate

- Lorsqu'il est positionné sur un composant de type ActionSource, la phase 5 <sup>8</sup>invoke Application<sup>8</sup> est appelée directement durant pendant la phase 2 <sup>8</sup>Apply requests<sup>8</sup>
- Lorsqu'il est positionné sur un composant de type UIInput, la phase 3 <sup>8</sup>Process Validations<sup>8</sup> est appelée directement durant pendant la phase 2 <sup>8</sup>Apply requests<sup>8</sup>
- Permet d'implémenter les fonctions type Cancel sur des formulaires

337



1. Créer une application de carnet d'adresse...
2. Avec nos entités Person, Address et les EJB liés
  1. Créer une page de visualisation de liste
    1. Avec pour chaque ligne la possibilité de
      1. visualiser en détail
      2. Editer
      3. Supprimer la ligne
    2. Créer une personne (pensez à l'adresse)
  3. Gérer la navigation global vers une page home
  4. Page home : lien vers la visualisation des personnes et des adresses.
  4. Création d'une page de liste des adresses permettant d'ouvrir le détail de la personne lié.

339

## Debuggage page JSF

- Utilisation de FacesTrace
  - Présentation des objets dans les différents scopes
  - Temps d'exécution des différentes phases
  - Arbre de composants
  - une taglib et une balise dans la page

338

## Librairies additionnelles

Il existe d'autres librairies propriétaires ou open-source qui permettent d'apporter d'autres composants ou fonctionnalités

- Primefaces **JSF 2.2**  
Ajax, Push, mobile, skin, HTML5  
(<http://primefaces.org>) 
- RichFaces **JSF 2.2**  
Ajax, Push, la plus répandue  
(<http://www.jboss.org/richfaces>) 
- IceFaces **JSF 2.1 (?)**  
Ajax, Push  
(<http://www.icesoft.org/java/home.jsf>) 

comparaison des 3 : <http://www.mastertheboss.com/richfaces/primefaces-vs-richfaces-vs-icefaces>

340

## La sécurité petit rappel

- Permettre de s'assurer de l'identité de la personne utilisatrice et des ressources auxquelles elle accède
- Principal : couple login-password
- Realms : Base de données des principaux. Il existe plusieurs types de Realm : mémoire, JDBC, JNDI
- Authentification : action de l'utilisateur pour s'identifier auprès d'une plateforme J2EE qui vérifie alors dans le Realm

341

## La sécurité

- Sécurité déclarée : définie dans les descripteurs de déploiement, elle est mise en place avant exécution
- Sécurité programmée : définie dans le code, elle utilise les services du container
- Groupe : ensemble d'utilisateurs ayant un même type de droit. Un utilisateur peut appartenir à plusieurs groupes
- Role : représentation logique d'un type de droits

342

## Authentification

- BASIC : popup du navigateur avec transmission en cryptage base64
- DIGEST : idem avec cryptage par ajout de valeur et hashage
- FORM : Utilisation d'un formulaire avec page et champs imposé.

343

## Déploiement

- Livrer l'application
- Embarquer la configuration (descripteur)
- Embarquer les parties statiques (web)

344

## Déploiement Packaging

Type	Description	Descripteur	Contenu
CAR	Client Application Archive	application-client.xml	Client lourd pour EJB
EAR	Entreprise Application Archive	application.xml	Autres modules Java EE composition
EJB-JAR	EJB- Java Archive	ejb-jar.xml	Session beans, MDB et optionnellement les Entités
RAR	Resource Adapter Archive	ra.xml	Adapteurs de ressources
WAR	Web Application Archive	web.xml faces-config.xml	Application web et optionnellement les Entités

345

## Déploiement Descripteur : application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    version="7" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/application_7.xsd">

    <application-name>application</application-name>
    <initialize-in-order>true</initialize-in-order>
    <module>
        <web>
            <web-uri>test-web.war</web-uri>
            <context-root>application</context-root>
        </web>
    </module>
    <module>
        <ejb>application-ejb.jar</ejb>
    </module>
    <library-directory>lib</library-directory>
</application>
```

347

## Déploiement Packaging - EAR

- Il s'agit d'un zip contenant :
  - le ou les ejb-jar
  - le ou les war
  - un éventuel dossier lib contenant des librairies
  - un dossier META-INF contenant le fichier application.xml

346

## Déploiement Packaging - EJB-JAR

- C'est un zip contenant :
  - les sources compilées des EJB
  - un répertoire META-INF avec un fichier ejb-jar.xml et accessoirement persistence.xml et/ou un fichier MANIFEST.MF

348

## Déploiement

### Descripteur : ejb-jar.xml (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    version="3.2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/
javaee/ejb-jar_3_2.xsd">

    <enterprise-beans>
        <session>
            <ejb-name>EmailEjb</ejb-name>
            <env-entry>
                <env-entry-name>java:app/env/smtp.host</env-entry-name>
                <env-entry-type>java.lang.String</env-entry-type>
                <env-entry-value>smtp.mandrillapp.com</env-entry-value>
            </env-entry>
            <remote>com.company.appli.ejb.EmailEjb</remote>
            <ejb-class>com.company.appli.ejb.EmailEjb</ejb-class>
            <session-type>stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>

    <!-- to be continued -->
```

Optionnel

## Déploiement

### Descripteur : ejb-jar.xml (2)

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    version="3.2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/
javaee/ejb-jar_3_2.xsd">

    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>MyEJB</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
        <security-role>
            <role-name>user</role-name>
        </security-role>
    </assembly-descriptor>

</ejb-jar>
```

Optionnel

349

350

## Déploiement

### Packaging - WAR

- Fichier war (c'est un simple zip)
- Structure :
  - \*.html, \*.jsp, \*.js, \*.css, ...
  - /WEB-INF
    - web.xml, faces-config.xml, ...
    - classes
    - lib

351

## Déploiement

### Descripteur : faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://
xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">

    <render-kit></render-kit>

    <managed-bean></managed-bean>

    <validator></validator>

    <converter></converter>

    <navigation-rule></navigation-rule>

</faces-config>
```

352

## Déploiement

Descripteur : web.xml

- Permet aussi la configuration
  - de paramètres pour les TagLib
  - des filtres
  - des servlets
  - de la gestion d'erreur
  - de la sécurité

353

That's all folks...

Enfin presque !

355

## Déploiement

Descripteur : web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        < servlet-name>Faces Servlet</servlet-name>
        < url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        < session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

354

## Bonnes pratiques

- Comprendre son application et ses dépendances (librairies, ressources...)
- Eviter les API et annotations propriétaires
- Impliquer le DBA

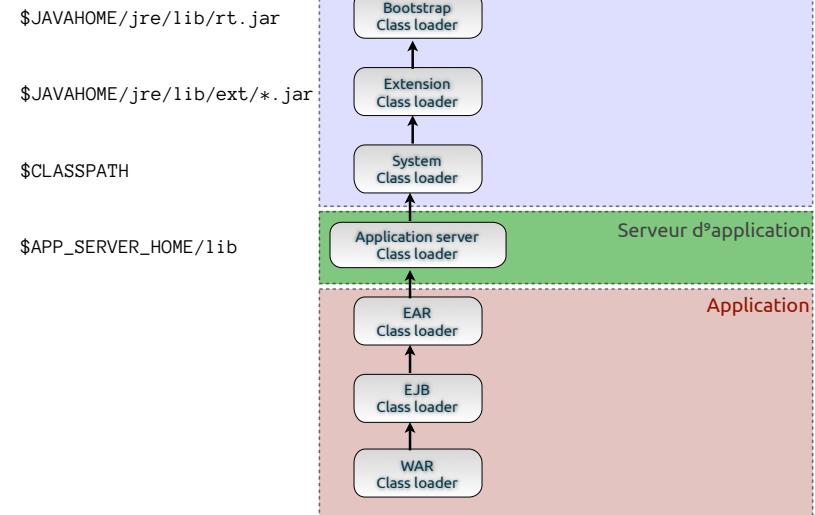
356

## Class loading

- Les classes sont chargées sur une base du besoin
- Concept d'héritage de ClassLoader :
  - Chaque ClassLoader hérite d'un autre
  - le premier ClassLoader est le Bootstrap ClassLoader
- Concept de délégation au parent :
  - Le ClassLoader regarde dans son cache
  - Le ClassLoader demande à son parent
  - Le ClassLoader cherche dans son classpath

357

## Class loading



358

## éviter quelques chausse trappes

- ClassNotFoundException : Chargement dynamique d'une classe : Librairie absente ou dans le mauvais ClassLoader. Le chargement de ressources se fait par : Thread.currentThread().getContextClassLoader().getResourceAsStream()
- NoClassDefFoundException : Librairie absente ou dans le mauvais ClassLoader
- ClassCastException : Duplication de classes : cast d'une instance d'une classe chargée avec ClassLoader 1 avec une classe chargée par ClassLoader 2
- NamingException : Echec lors d'un lookup JNDI (injection ou manuel)
- Echec de déploiement : généralement XML non valide ou appelant des ressources inexistantes

359