

# Framework Java EE

François Robert  
TOP

1

## De quoi parle t'on ?

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

2

## Java EE 7

Un standard

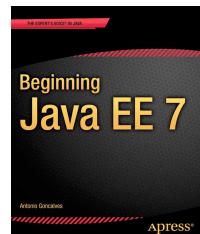
C'est un ensemble de plus de 30 spécifications  
Coiffées par la JSR 342

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

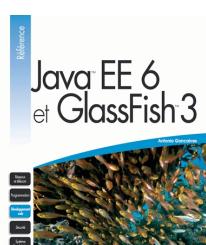
<http://www.oracle.com/technetwork/java/javaee/tech/index.html>



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

## 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

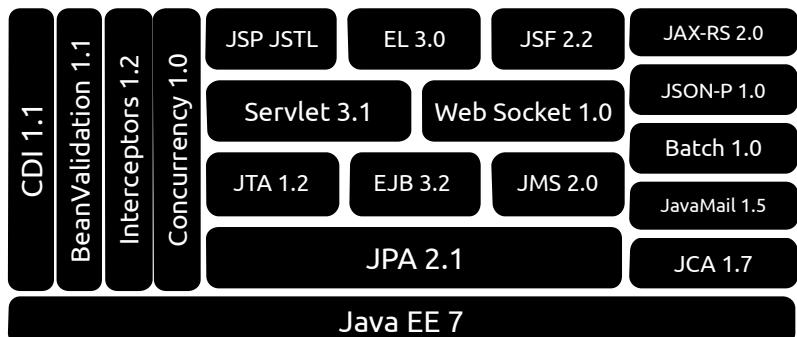
<http://www.oracle.com/technetwork/java/javaee/tech/index.html>

3

4

# Java EE 7

## Les principaux composants

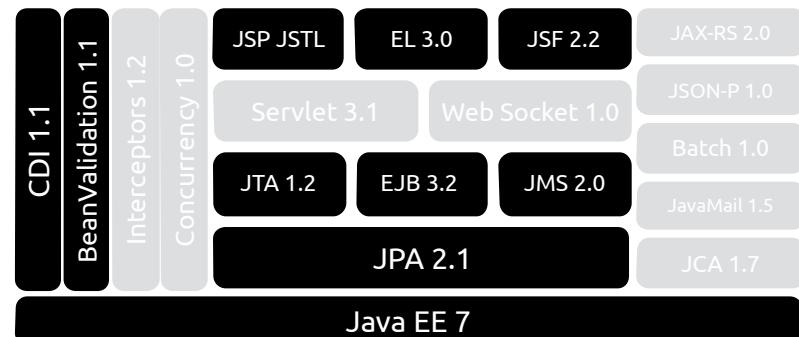


5

# Java EE 7

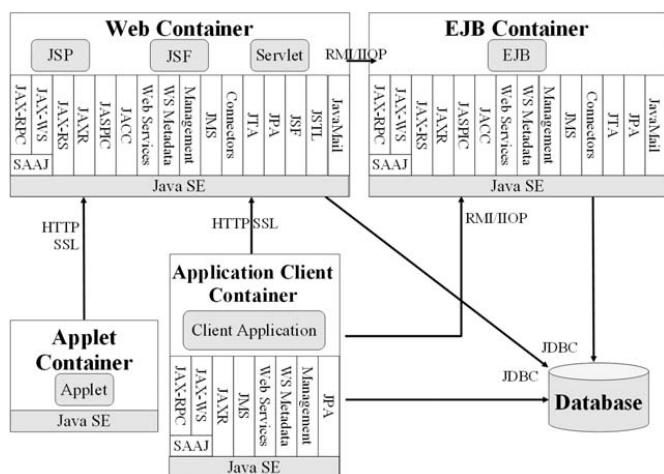
## Les principaux composants

Abordés pendant ce cours



6

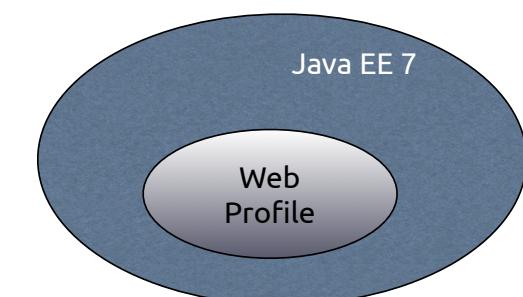
# Java EE 7



7

# Java EE profil web

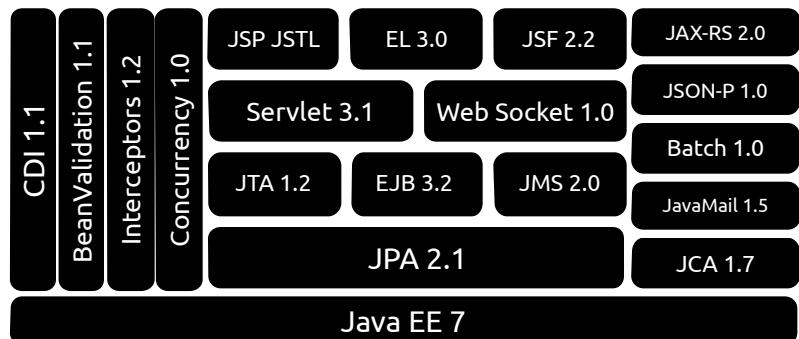
## Version légère



8

## Java EE 7

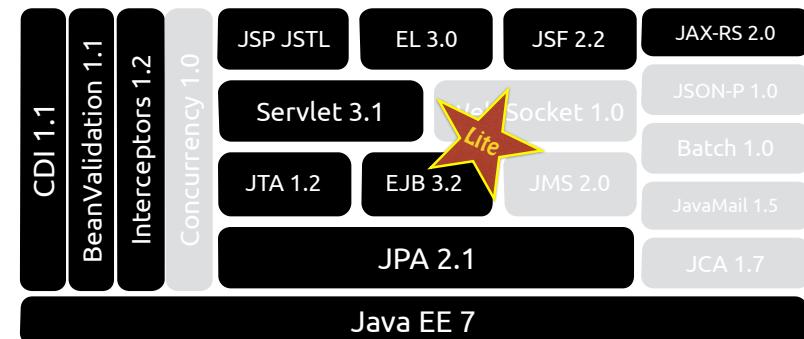
### Composant Full profile



9

## Java EE 7

### Composant profil web



10

## 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)

11

## Java EE profil web

### ce qu'il y a en plus dans EJB “full”

- 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

12

## Java EE 7

### Composant profil web

- 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

13

## Architecture d'entreprise

Architecture distribuée

Architecture multi-tiers

Architecture SOA

Redondance & Cluster

14

## Architecture distribuée

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

15

## Architecture multitiers

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

Architecture 3 tiers

Architecture 4 tiers

16

## Architecture multitiers

Architecture 3 tiers

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

17

## Architecture multitiers

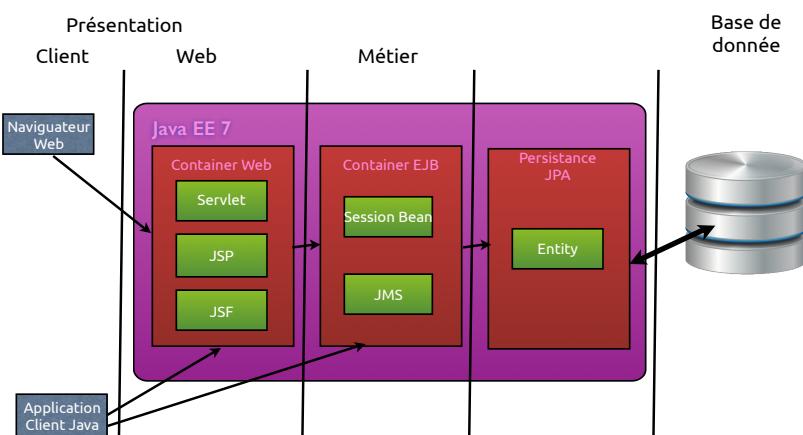
Architecture 4 tiers

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

18

## Architecture multitiers

Architecture 4 tiers



19

## Architecture SOA

Services Oriented Architecture

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

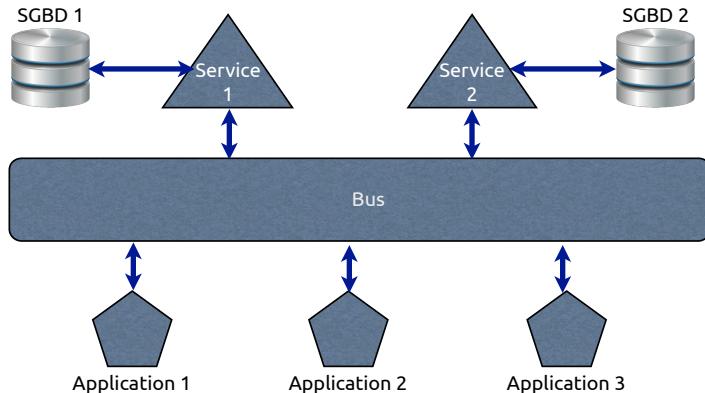
Une partie métier

Un accès aux données

20

## Architecture SOA

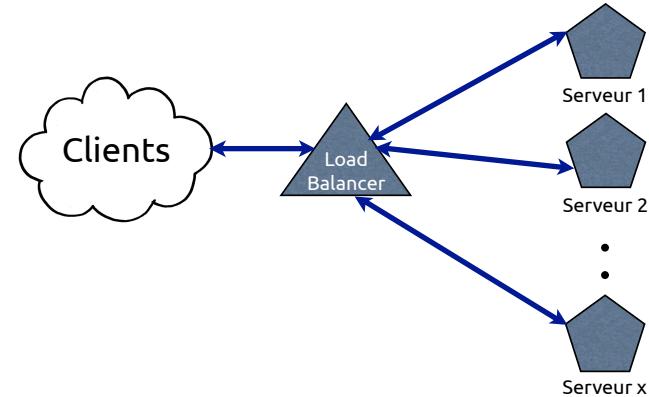
Services Oriented Architecture



21

## Redondance & Cluster

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



22

## 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
  - ...

23

## Les serveurs d'application

Fournissent aussi des services

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

24

## Les outils du cours

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

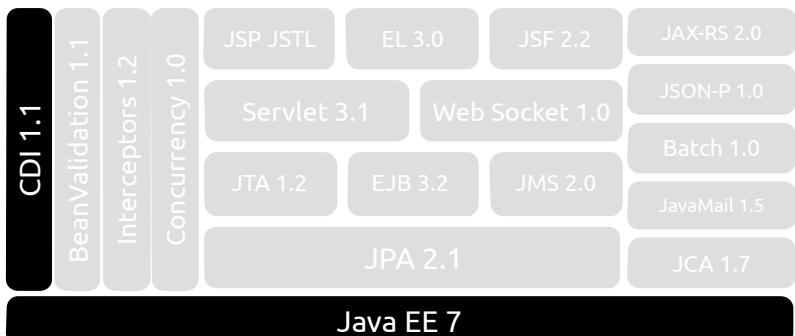
25

## 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

26

## Java EE 7 Contexts and Dependency Injection



27

## 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

28

# Java EE 7

## Contexts and Dependency Injection

« I am not a blues singer. I am not a jazz singer.  
I am not a country singer.  
But I am a singer who can sing the blues, who  
can sing jazz, who can sing country. »

*Ray Charles*

29

# Java EE 7

## Contexts and Dependency Injection

lookup                      Inject  
  
« Don't call us, we'll call you »

*Principe d'hollywood*

30

## CDI

- Apparu avec JavaEE 6 (1998)
  - #annotationeverywhere
  - #noxml (ou presque)
- Aucune limite
  - Si vous pouvez l'imaginer, vous pouvez le faire
- Implémentation de référence : JBoss weld
  - Il en existe deux autres
    - Apache open webBeans
    - Cauchy Candi
- Limité à JavaEE ??????
  - Pas forcément !

31

## CDI

- Contrôle de
  - Resolution d'injection
  - Cycle de vie
  - Configuration
- Généré par un composant externe (ex : container)
  - ... et pas par le composant lui-même
- Il apporte le « loose coupling »

32

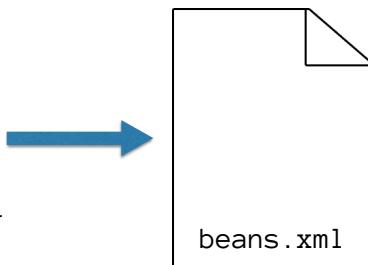
## CDI

Le bean

En JavaEE 7 tout est managed Bean



Pensez au fichier



33

## CDI

beans.xml

```
<beans
  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/beans_1_1.xsd" bean-discovery-mode="all">
</beans>
```

JAR    src/main/java/resources/META-INF/beans.xml

WAR    src/main/java/resources/WEB-INF/beans.xml

34

## CDI

Le bean

En JavaEE 7 tout est managed Bean

- Le managed bean est le composant de base
- Il a un cycle de vie
- Il est interceptable
- Il est injectable
- Il est accessible via JNDI

35

## CDI

Injection de dépendance

@Inject

36

## CDI

Injection de dépendance

```
public class GreetingService {  
  
    public GreetingService() {  
    }  
  
    public String hello() {  
        return "Hello";  
    }  
  
}
```

37

## CDI

Injection de dépendance - via le constructeur

```
public class SayHello {  
  
    private GreetingService greetingService;  
  
    @Inject  
    public SayHello(GreetingService greetingService) {  
        this.greetingService = greetingService;  
    }  
  
    public String sayHello() {  
        return greetingService.hello();  
    }  
}
```

38

## CDI

Injection de dépendance - via une propriété

```
public class SayHello {  
  
    private GreetingService greetingService;  
  
    @Inject  
    public void setGreetingService(GreetingService greetingService) {  
        this.greetingService = greetingService;  
    }  
  
    public String sayHello() {  
        return greetingService.hello();  
    }  
}
```

39

## CDI

Injection de dépendance - via le champ

```
public class SayHello {  
  
    @Inject  
    private GreetingService greetingService;  
  
    public String sayHello() {  
        return greetingService.hello();  
    }  
}
```

40

# CDI

## Injection de dépendance

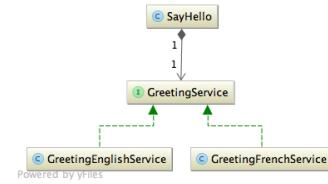
```
public interface GreetingService {  
    public String hello();  
}  
  
public class GreetingFrenchService implements GreetingService {  
    @Override  
    public String hello() {  
        return "Bonjour";  
    }  
}  
  
public class GreetingEnglishService implements GreetingService {  
    @Override  
    public String hello() {  
        return "hello";  
    }  
}
```

41

# CDI

## Injection de dépendance

```
public class SayHello {  
  
    @Inject  
    private GreetingService greetingService;  
  
    public String sayHello() {  
        return greetingService.hello();  
    }  
}
```



42

# CDI

## Le bean @Default

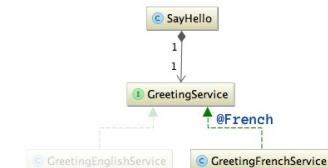
```
public interface GreetingService {  
    public String hello();  
}  
  
@Default  
public class GreetingFrenchService implements GreetingService {  
    @Override  
    public String hello() {  
        return "Bonjour";  
    }  
}
```

43

# CDI

## Les qualifiers

```
public class SayHello {  
  
    @Inject @French  
    private GreetingService greetingService;  
  
    public String sayHello() {  
        return greetingService.hello();  
    }  
}
```



44

# CDI

## Les qualifiers

```
@Qualifier  
@Retention(RUNTIME)  
@Target({FIELD, TYPE, METHOD, PARAMETER})  
public @interface French { }
```

```
@Qualifier  
@Retention(RUNTIME)  
@Target({FIELD, TYPE, METHOD, PARAMETER})  
public @interface English { }
```

45

# CDI

## Les qualifiers

```
@Qualifier  
@Retention(RUNTIME)  
@Target({FIELD, TYPE, METHOD, PARAMETER})  
public @interface Language {  
  
    Languages value();  
  
    @Nonbinding  
    String description() default "";  
  
    public enum Languages {  
        FRENCH, ENGLISH  
    }  
}
```

47

# CDI

## Les qualifiers

```
public interface GreetingService {  
    public String hello();  
}  
  
@French  
public class GreetingFrenchService implements GreetingService {  
    @Override  
    public String hello() {  
        return "Bonjour";  
    }  
}  
  
@English  
public class GreetingEnglishService implements GreetingService {  
    @Override  
    public String hello() {  
        return "hello";  
    }  
}
```

46

# CDI

## Les qualifiers

```
public interface GreetingService {  
    public String hello();  
}  
  
@Language(FRENCH)  
public class GreetingFrenchService implements GreetingService {  
    @Override  
    public String hello() {  
        return "Bonjour";  
    }  
}  
  
@Language(ENGLISH)  
public class GreetingEnglishService implements GreetingService {  
    @Override  
    public String hello() {  
        return "hello";  
    }  
}
```

48

# CDI

## Les qualifiers

```
public class SayHello {  
  
    @Inject @Language(FRENCH)  
    private GreetingService greetingService;  
  
    public String sayHello() {  
        return greetingService.hello();  
    }  
}
```



49

# CDI

## Les qualifiers

```
public class SayHello {  
  
    @Inject @Language(ENGLISH)  
    private GreetingService greetingService;  
  
    public String sayHello() {  
        return greetingService.hello();  
    }  
}
```



50

# CDI

## Les qualifiers

```
@French @Validated @User  
public class GreetingFrenchService implements GreetingService {  
    @Override  
    public String hello() {  
        return "Bonjour";  
    }  
}  
  
public class SayHello {  
  
    @Inject @French  
    private GreetingService greetingService;  
  
    public String sayHello() {  
        return greetingService.hello();  
    }  
}
```

51

# CDI

## Les qualifiers

```
@French @Validated @User  
public class GreetingFrenchService implements GreetingService {  
    @Override  
    public String hello() {  
        return "Bonjour";  
    }  
}  
  
public class SayHello {  
  
    @Inject @French @User @Validated  
    private GreetingService greetingService;  
  
    public String sayHello() {  
        return greetingService.hello();  
    }  
}
```

52

# CDI

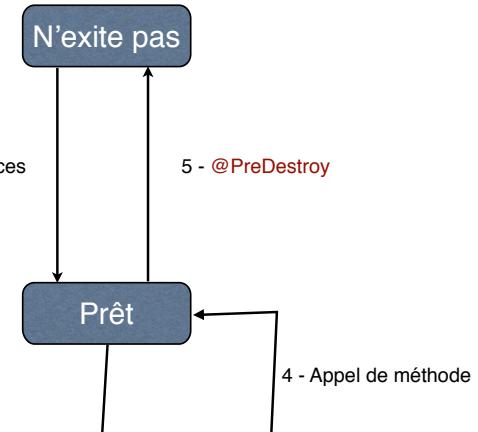
## Les qualifiers

```
@French @Validated  
public class GreetingFrenchService implements GreetingService {  
    @Override  
    public String hello() {  
        return "Bonjour";  
    }  
  
    public class SayHello {  
  
        @Inject @French @User @Validated  
        private GreetingService greetingService;  
  
        public String sayHello() {  
            return greetingService.hello();  
        }  
}
```

53

# CDI

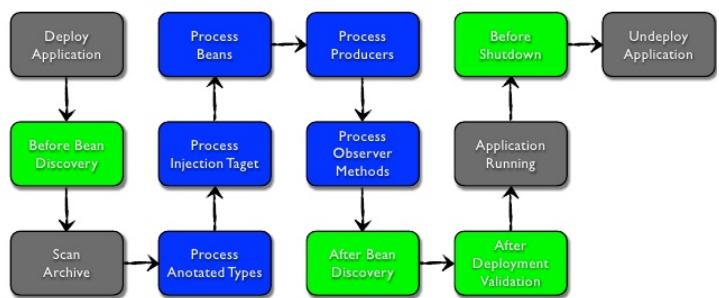
## Le cycle de vie



54

# CDI

## Le cycle de vie



55

# CDI

## Le cycle de vie - Les callback

```
@Language(FRENCH)  
public class GreetingFrenchService implements GreetingService {  
  
    @PostConstruct  
    public void setup() {  
        // do something smart  
    }  
  
    @PreDestroy  
    public void teardown() {  
        // do something...  
    }  
  
    @Override  
    public String hello() {  
        return "Bonjour";  
    }  
}
```

56

# CDI

## les contextes

- Gestion des cycle de vie des beans
- Choix du moment de création & destruction des beans
  - Singleton pour un contexte donné
- Contextes
  - Session
  - Requête
  - Conversation
  - Application
  - Singleton
- Possibilité de créer des scope personnalisés
  - Mais ça c'est une autre histoire

57

# CDI

## les contextes

```
@Language(FRENCH)
@SessionScoped
public class GreetingFrenchService implements GreetingService,
Serializable {

    @Override
    public String hello() {
        return "Bonjour";
    }
}
```

58

# CDI

## les contextes

```
@Language(FRENCH)
@ApplicationScoped
public class GreetingFrenchService implements GreetingService,
Serializable {

    @Override
    public String hello() {
        return "Bonjour";
    }
}
```

*Failed !!!!*

59

# CDI

## les contextes

```
@Language(FRENCH)
@RequestScoped
public class GreetingFrenchService implements GreetingService,
Serializable {

    @Override
    public String hello() {
        return "Bonjour";
    }
}
```

60

# CDI

les contextes

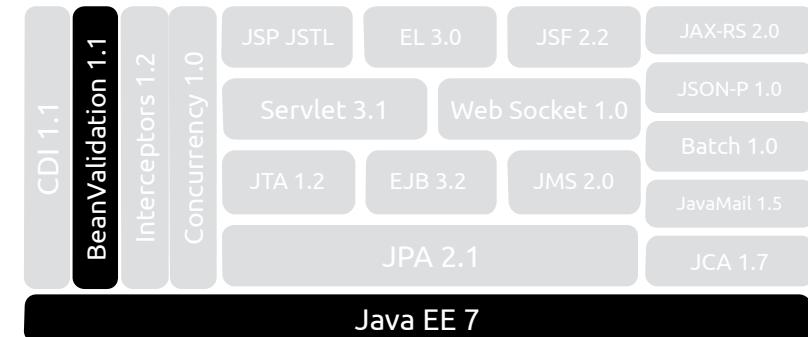
```
@Language(FRENCH)
@ConversationScoped
public class GreetingFrenchService implements GreetingService,
Serializable {

    @Override
    public String hello() {
        return "Bonjour";
    }
}
```

61

# Java EE 7

## Bean validation



62

## Bean validation

Vous avez dit contraintes ?

- Contrainte de votre modèle : bean ou attribut
  - L'adresse est elle valide ?
  - l'email est il correctement formaté ?
  - Le nom est non null ?
  - La date de naissance est bien dans le passé ?
- S'assurer que toutes les données sont valides
- S'assurer que le service va avoir le comportement attendu
- Donner à l'utilisateur un retour cohérent

63

## Bean validation

Ou utiliser les contraintes



64

## Bean validation

Constraint once, Validate everywhere !

- Moyen unique d'exprimer une contrainte
- Standardisation de la validation
- Utilisation de contrainte déjà définies
- Développer ses propres contraintes
- Réutilisation des contraintes à travers les couches

65

## Bean validation

Les contraintes standard

- @AssertFalse, @AssertTrue
- @DecimalMax, @DecimalMin, @Digit
- @Future, @Past
- @Max, @Min
- @NotNull, @Null
- @Pattern
- @Size

67

## Bean validation

Contrainte sur POJO

```
public class Address {  
    @NotNull  
    private String name;  
    @NotNull  
    @Size(max = 30, message = "Ne doit pas excéder {max} caractères")  
    private String street1;  
    @NotNull @Valid  
    private Country country;  
    ...  
}  
  
public class Country {  
    @NotNull @Size(max = 30)  
    private String name;  
}
```

66

## Bean validation

Valider un POJO

Hors container CDI

```
public void validateAddress(Address address) {  
    ValidatorFactory validatorFactory =  
        Validation.buildDefaultValidatorFactory();  
    Validator validator =  
        validatorFactory.getValidator();  
    validator.validate(address);  
}
```

68

## Bean validation

Valider un POJO  
container CDI

```
@Inject  
private Validator validator;  
  
public void validateAddress(Address address) {  
    validator.validate(address);  
}
```

69

## Bean validation

Ecrire une contrainte  
Avec code

```
@Target({METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER})  
@Retention(RUNTIME)  
@Documented  
@Constraint(validatedBy = NotNullValidator.class)  
public @interface NotNull {  
    String message() default "L'objet ne doit pas être null";  
    Class<?>[] groups() default {};  
    Class<? extends Payload>[] payload() default {};  
}
```

70

## Bean validation

Ecrire une contrainte  
Avec code

```
@Target({METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER})  
@Retention(RUNTIME)  
@Documented  
@Constraint(validatedBy = NotNullValidator.class)  
public @interface NotNull {  
    String message() default "{siad.constraints.notnull}";  
    Class<?>[] groups() default {};  
    Class<? extends Payload>[] payload() default {};  
}
```

71

## Bean validation

Ecrire une contrainte  
Avec code

```
public class NotNullValidator  
    implements ConstraintValidator<NotNull, Object> {  
    @Override  
    public void initialize(NotNull notNull) {  
    }  
  
    @Override  
    public boolean isValid(Object object, ConstraintValidatorContext constraintValidatorContext) {  
        return object != null;  
    }  
}
```

72

## Bean validation

Ecrire une contrainte  
Sans code

```
@NotNull  
 @Size(min = 7)  
 @Pattern(regexp = "[a-zA-Z0-9!#$%&'*+/=?^_`{|}~-]+(?:\\.[a-zA-Z0-9!#$%&'*+/=?^_`{|}~-]+)*|(?:@[a-zA-Z0-9]([a-zA-Z0-9-]*[a-zA-Z0-9])?\\.)+[a-zA-Z0-9]([a-zA-Z0-9-]*[a-zA-Z0-9])?")  
 @Target({METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER})  
 @Retention(RUNTIME)  
 @Documented  
 @Constraint(validatedBy = {})  
 public @interface EMail {  
     String message() default "l'adresse email n'est pas correctement  
     formatée";  
  
     Class<?>[] groups() default {};  
     Class<? extends Payload>[] payload() default {};  
 }
```

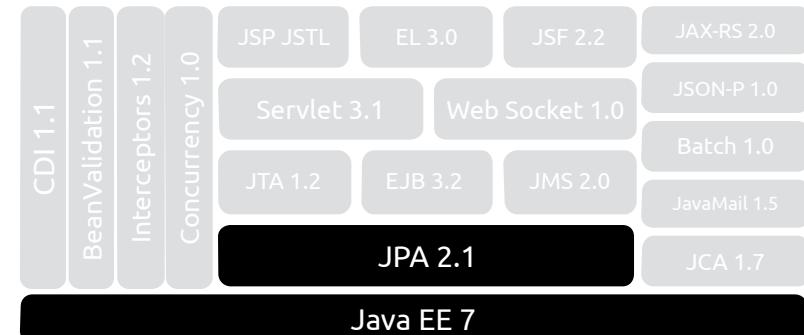
73

## 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

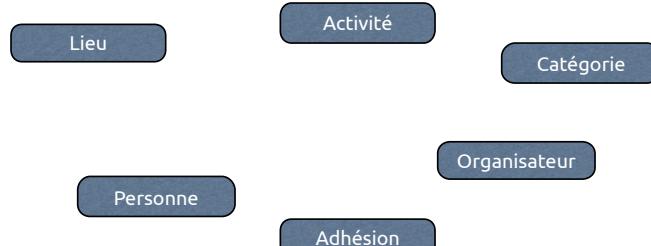
75

## Java EE 7 Persistance



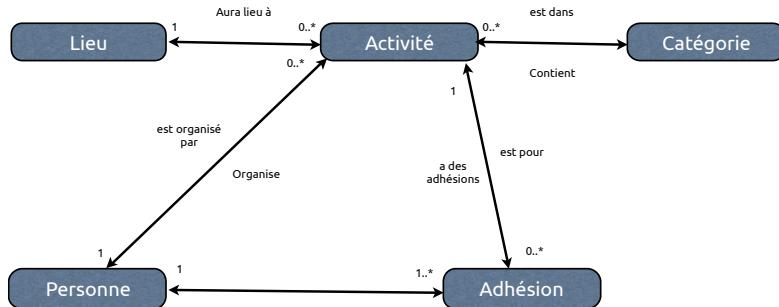
74

## Exemple



76

# Exemple



77

## 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

79

## 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

78

## 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  
}
```

80

# 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() {  
    }  
}
```

81

# 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

83

# Anatomie d'une entité

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

82

# 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() {  
    }  
}
```

84

# 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

85

# Les annotations de mapping

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

87

# 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	classes annotées @Embeddable

86

# @Table

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

88

## @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

89

## @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;
}
```

90

## @Column

- **optionnelle**, si omise, la colonne à le nom de l'attribut ou propriété
- **name** : nom de la colonne
- **table** : en cas de mapping sur deux tables
- **insertable, updatable** : permet d'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'unicité
  - **length** : taille de la colonne
  - **precision, scale** : pour les décimales
  - **columnDefinition** : Spécifie le SQL exact de la création

91

## @Enumerated

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

92

## @Enumerated

- Si on utilise ORDINAL, les valeurs sauvegardées en base sont de 0 à n selon le nombre d'éléments dans l'énumération
- Si c'est STRING, c'est le nom de la valeur de l'énuméré qui est stocké
- Si l'annotation est omise, l'attribut ou propriété est sauvegardée de façon ordinal

93

## @Lob

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

94

## @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

95

## @Temporal

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

96

## @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

97

## @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() {}  
}
```

99

## @Transient

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

98

## @SecondaryTables

Il est parfois nécessaire de stocker les valeurs dans différentes tables

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

100

## @ 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;
```

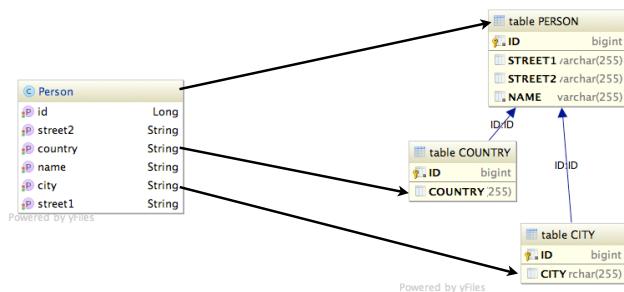
101

## @ 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;
```

102

## @ SecondaryTables



103

## @ SecondaryTables

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

104

## 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

105

## 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;  
    }  
}
```

107

## 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() {}
```

106

## 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() {}
```

108

## 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

109

## @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

110

## @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

111

## @EmbeddedId

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

    public Category() {
    }
    ...
}

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

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

112

## @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

113

## @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() {}  
}
```

114

## 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

115

## 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;
```

116

## 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;
```

117

## @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;  
}
```

118

## 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;
```

119

## Table de séquence

- La table SEQUENCE\_GENERATOR\_TABLE

SEQUENCE_NAME	SEQUENCE_VALUE
USER_SEQUENCE	1
«autre séquence»	23

120

## @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 {};
}
```

121

## @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;
```

123

## 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»

122

## @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;
```

124

# @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() {  
    }
```

125

# @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;  
}
```

126

# @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

127

# @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;
```

128

# 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

129

# @ 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 "";
}
```

131

# @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() {
    }
```

130

# @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() {
    }
```

132

## @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 "";
}
```

133

## 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;
```

134

## @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() {
    }
```

135

## @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() {
    }
```

136

## @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

137

## @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;
```

139

## @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;
```

138

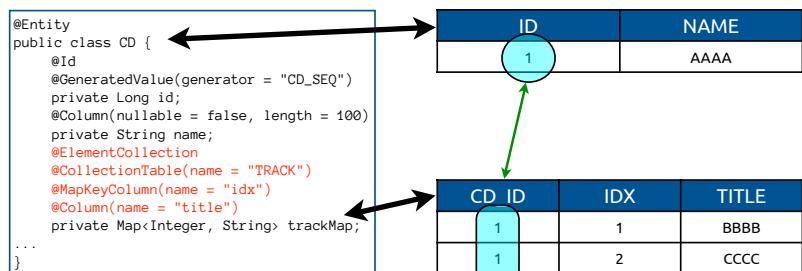
## Relation et FETCH

Relation	Stratégie FETCH
<code>@OneToOne</code>	EAGER
<code>@ManyToOne</code>	EAGER
<code>@OneToMany</code>	LAZY
<code>@ManyToMany</code>	LAZY

140

# Et les Map ?...

Petit rappel, une map = liaison clé/valeur



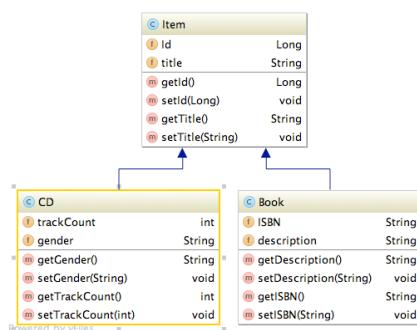
141

# L<sup>e</sup>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

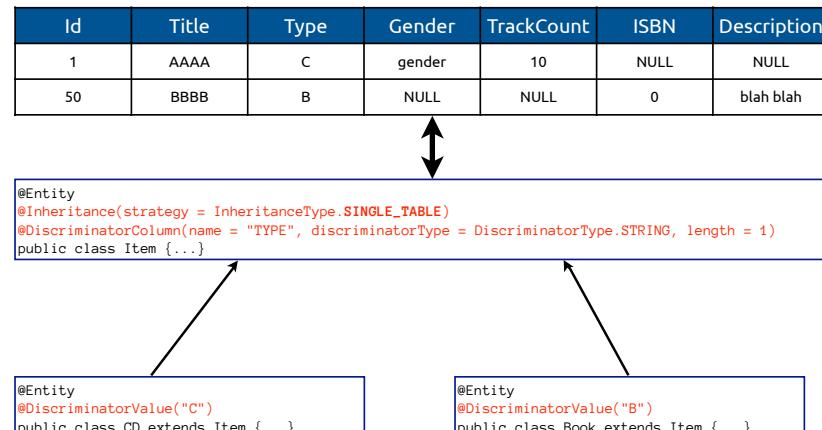
142

## Le modèle



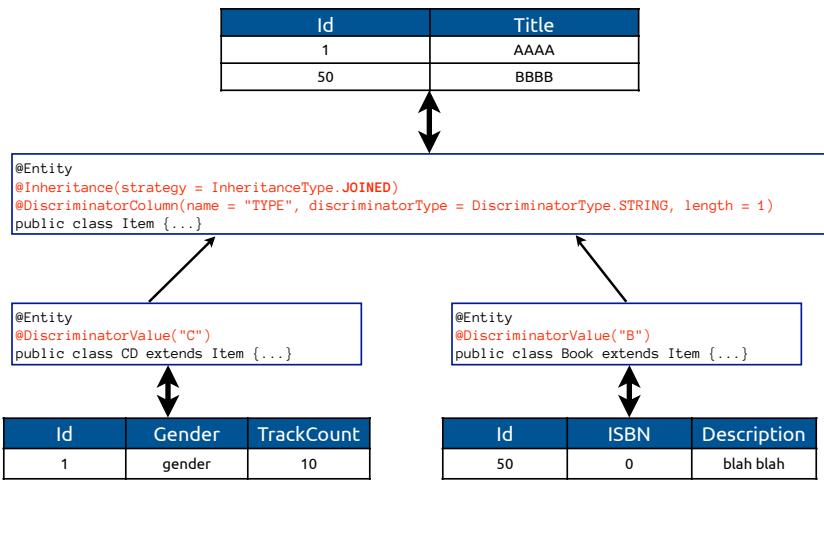
143

## Une table



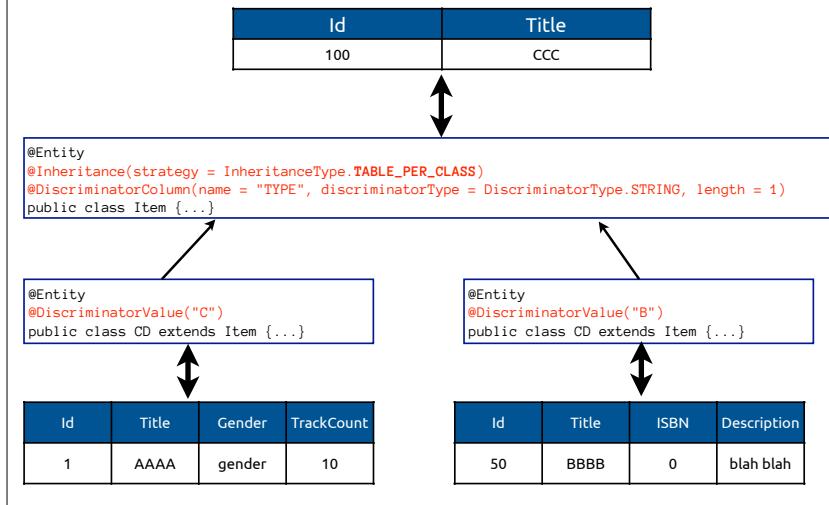
144

## Tables jointes



145

## Une table par classe

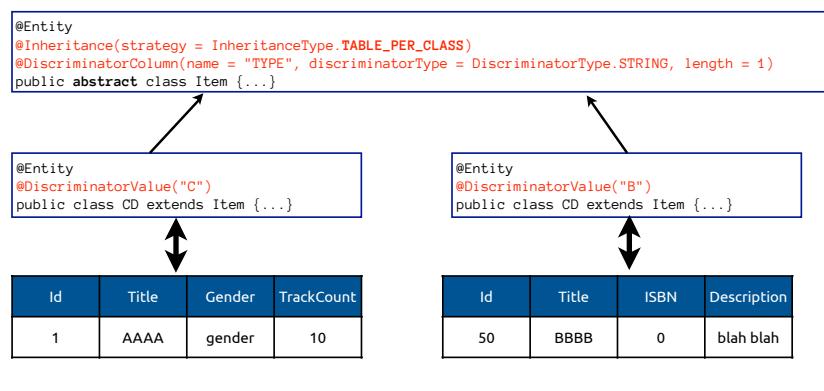


146

## Une table par classe



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



147

## 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

148

# Persistance & JPA

149

## 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

151

## 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

150

## Comment persister en java

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

152

# Les avantages de JPA

## 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.

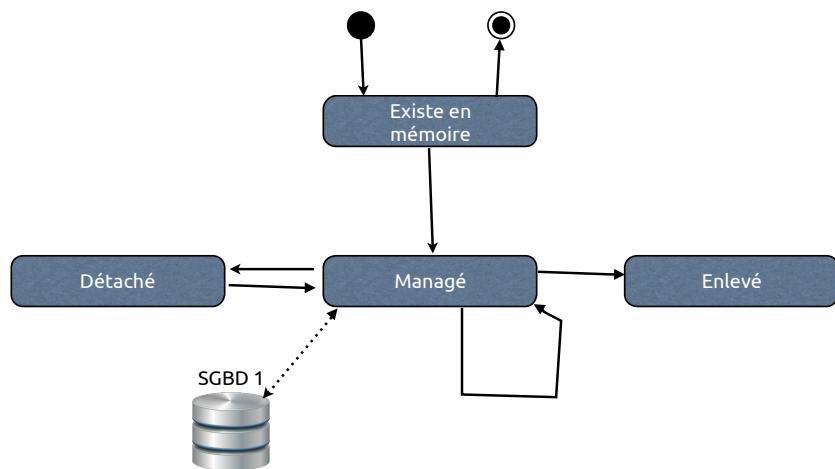
153

# 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

154

# Cycle de vie



155

# 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();  
}
```

156

# 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>
```

157

## 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 definir le scope transaction ou extended.
Extended, ne peut être activer qu'avec des StateFullSessionBean (SFSB)
```

159

# 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)

158

## @PersistenceContex

```
@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

160

# 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

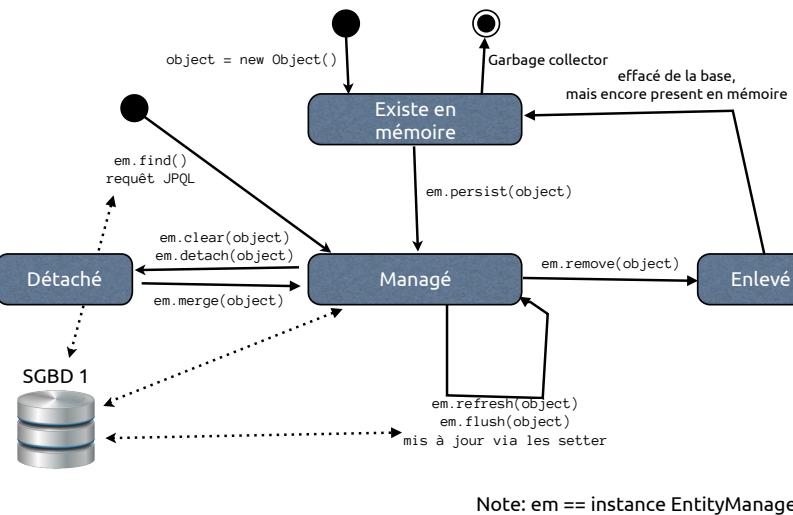
161

# Persister 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

163

# Cycle de vie



162

# Persister 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());

164

# 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

165

# 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)

167

# 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

166

# Modifications

- Entité managé
  - 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)

168

# 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.assertNotNull(person);
Person person1 = entityManager.find(Person.class, 1L);
Assert.assertNull(person1);
```

Les données `person` ont été supprimées de la base  
Cependant l'instance reste en mémoire jusqu'au  
passage du GarbageCollector

169

# Forcer les mises à jour

- l'`EntityManager` ne transfert pas les mises à jour immédiatement
- On peut cependant la forcer avec `flush`

170

# 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*

171

# 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());
```

172

## 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));
```

173

## 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 associations sont soumises en même temps aux opérations de rattachement
REFRESH	Classes et associations sont rafraîchies au même moment
DETACH	Classes et associations sont détachées au même moment
ALL	Association de toutes les conditions précédentes

175

## 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();
```

174

## 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();
```

176

## 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();
```

177

## 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*

178

## 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

179

## 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));
```

180

## 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.**

181

## 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");
```

183

## 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));
```

182

## JPQL pour charger

- Charger une liste d'entités

```
List personList = query.getResultList();  
• aucun résultat == liste vide  
• Possibilité de pagination via l'API Query
```

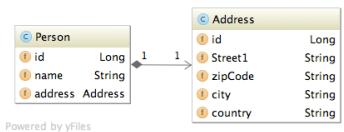
- Charger un résultat unique

```
Person personFound = (Person)query.getSingleResult();  
• Lève des exceptions  
NonUniqueResultException et  
NoResultException qui sont des exceptions  
RunTime
```

184

# JPQL et les relations

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



Powered by yFiles

```
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();
```

185

# 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

187

# 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

186

# 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 »>]
```

188

## Jointure

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

189

## Opérations de masse

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

190

## 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

191

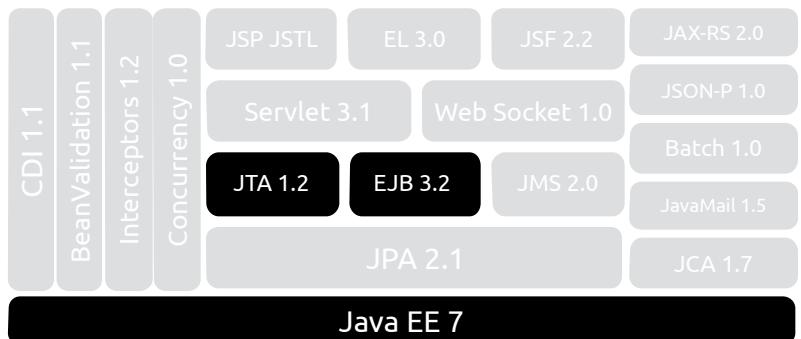
## Hands on !!

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

192

## Java EE 7

### Enterprise Java Bean



193

## Les types d'EJB

- Stateless: Gestion sans état. **Session bean**
- Stateful: Gestion conversationnelle
- Singleton
- Message Driven bean: interaction avec JMS

195

## 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 !

194

## un EJB...

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

196

## 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

197

## 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

199

## Comment appeler un EJB

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

198

## 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.

200

## 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.

201

## 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

203

## 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

202

## 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

204

## Transaction

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

205

## Sécurité

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

206

## intercepteur

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

207

## 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

208

## Web services

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

209

## Lookup JNDI

```
@Stateless  
@JBoss(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  
    }  
}
```

211

## Invocation d'un EJB

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

210

## 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);  
    }  
}
```

212

## @EJB

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

213

## 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

215

## @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.

214

## Les Bean de session

216

## 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

217

## Les 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.

219

## 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

218

## EJB et interfaces

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

220

# EJB et interfaces

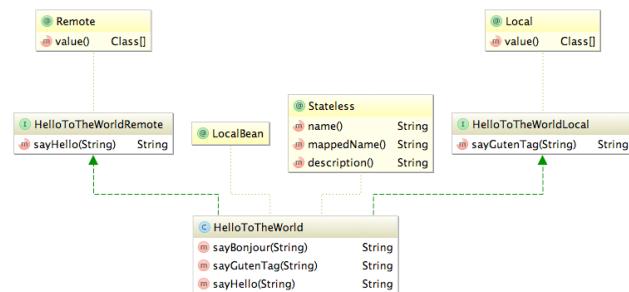
```
@Stateless @LocalBean
public class HelloToTheWorld implements HelloToTheWorldLocal {
    public String sayBonjour(String name) {
        return "bonjour " + name;
    }
    @Override
    public String sayGutenTag(String name) {
        return "guten tag " + name;
    }
}
```

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

221

# 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

223

# 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);
}
```

222

# 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;
    }
}
```

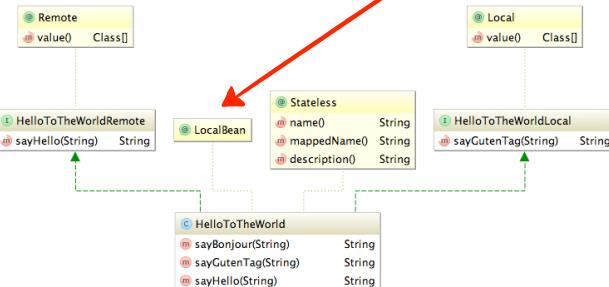
```
public interface HelloToTheWorldLocal {
    String sayGutenTag(String name);
}
```

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

224

# EJB, interfaces & appel

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



Powered by yfiles

225

# 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 REST";
    }
}
```

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

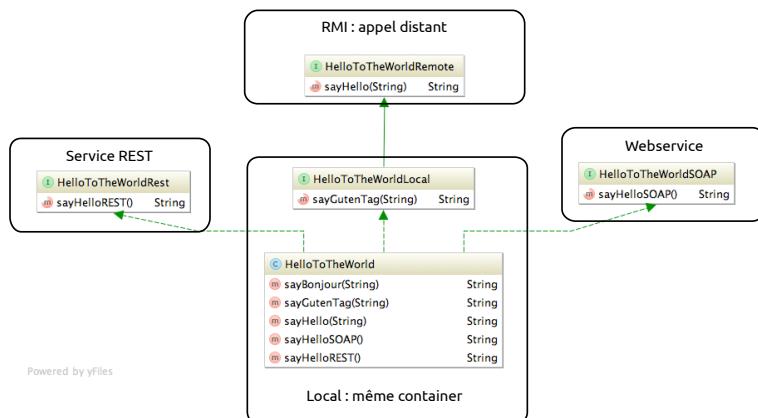
```
@WebService
public interface HelloToTheWorldSOAP {
    String sayHelloSOAP();
}
```

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

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

226

# EJB et interfaces (bonus)



Powered by yfiles

227

# 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

228

# Règles de programmation

- Héritage des annotations pour DI et lifecycle callback
- Les méthodes business ne doivent pas démarrer par "ejb"
- 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`

229

## 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 "";
}
```

231

# 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 la session HTTP

230

## 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;
    }
}
```

232

# 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

233

## 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.

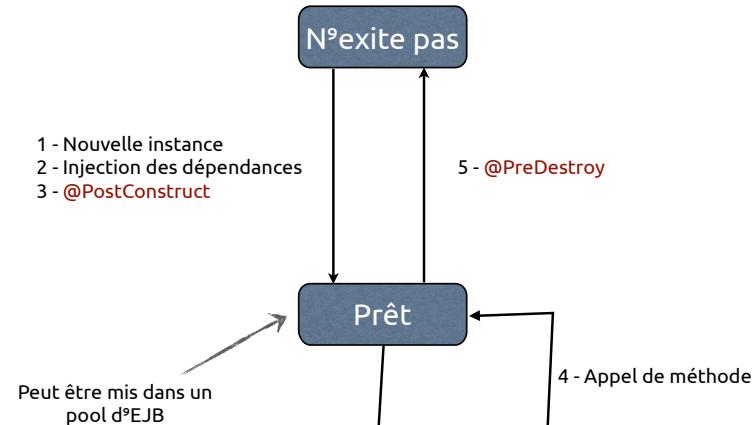
Vous remarquerez une organisation différente des sources:

- 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.

235

# Stateless

## Cycle de vie



234

## 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 créé, 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 récupérer toutes les personnes enregistrées
- 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.

236

# 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 "";
}
```

237

# Singleton

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

238

# 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) {}
}
```

239

# Singleton Enchainement

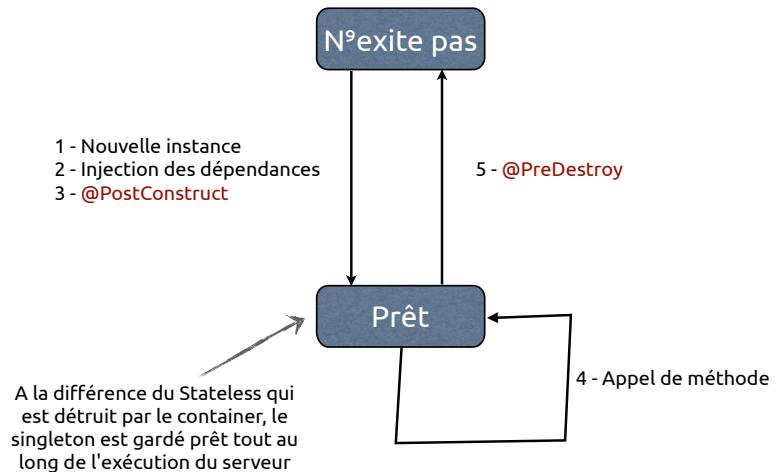
```
@Singleton
public class ZipcodeCache {
}
@Singleton
@Startup
@DependsOn("CityCache", "ZipcodeCache")
public class AddressCache {
    @PostConstruct
    private void addressCacheInitialization() {
        // Initialisation longue
    }
    public Address getAddressFromCache(long id) {return address}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

@Singleton
public class CityCache {
}
```

240

# Singleton

## Cycle de vie



241

# 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**

242

# 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

243

```
@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) {  
    }  
}
```

244

# Singleton

Concurrence gérée par le bean

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

245

## 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 ;)

247

# 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) {  
    }  
}
```

246

## 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;  
    }  
}
```

248

# Stateful

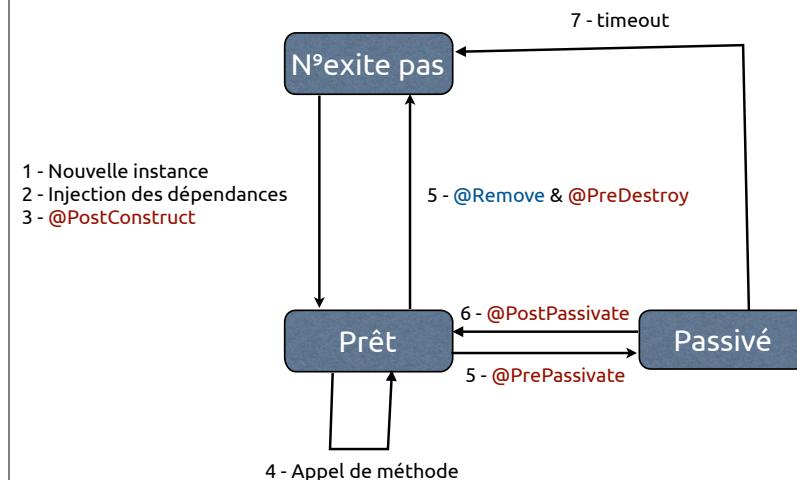
## Règles de programmation

- 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)

249

# Stateful

## Cycle de vie



250

# Stateful

## Passivation

- Sérialisation 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`

251

# 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

252

# 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

253

# 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

254

# Déférence 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

255

# EJB & Transaction

256

# 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

257

# 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)

259

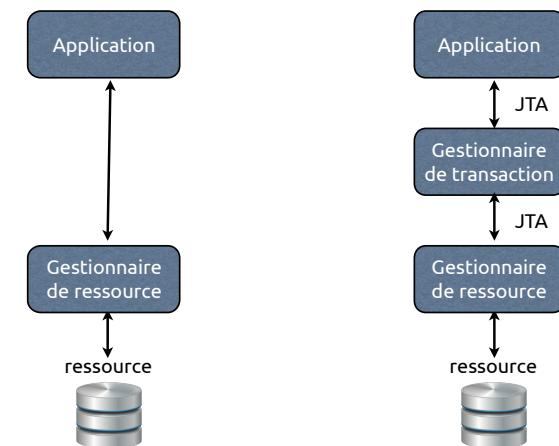
# ACID

- **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 commitées sont permanentes et survivent à un crash système

258

# Transaction locale

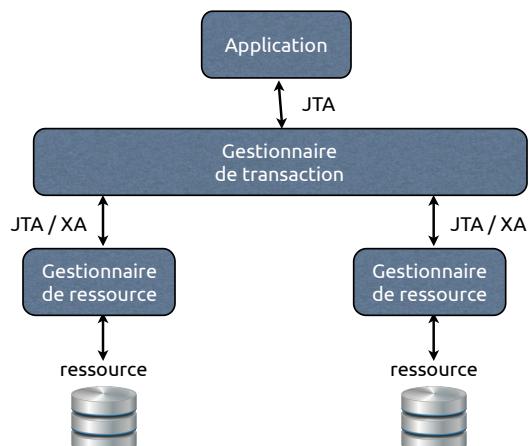
une seule ressource transactionnelle



260

# Transactions Distribuées

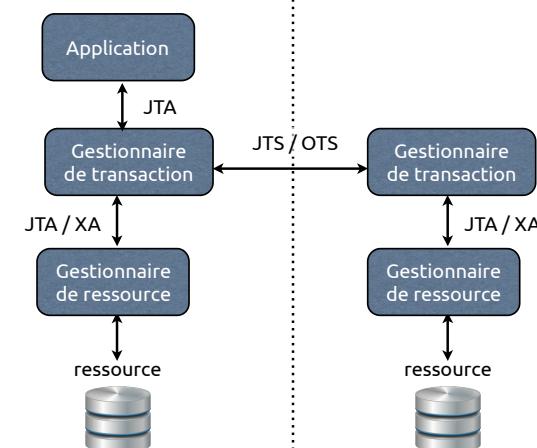
XA



261

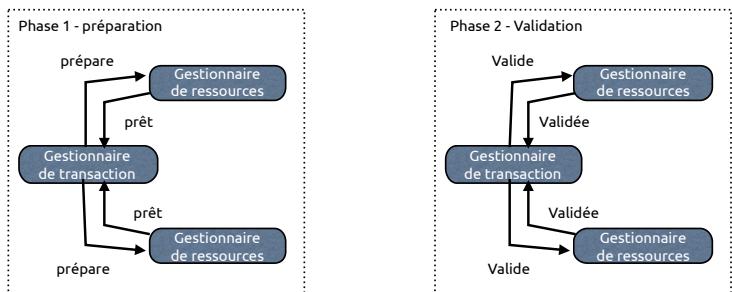
# Transactions Distribuées

XA via le reseau



262

# Validation en 2 phases



263

# 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;  
    }  
}
```

264

## 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;  
    }  
}
```

265

## Annulation de transaction

- On peut noter un 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;  
@TransactionAttribute(TransactionAttributeType.MANDATORY)  
public Task createTask(Task task) {  
    entityManager.persist(task);  
    toDoListEJB.addTask(task);  
    sessionContext.setRollbackOnly();  
    return task;  
}
```

267

## Attributs de transaction

Gestion des transactions par le container  
@TransactionAttribute

@TransactionAttribute	Dans une transaction	hors Transaction
MDB REQUIRED	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
MDB NOT_SUPPORTED	La transaction du client est suspendue et aucune transaction est créée	Pas de transaction
NEVER	déclenche une EJBException	Pas de transaction

266

## Transactions et exceptions

- par défaut toutes les <sup>8</sup>checked Exception<sup>8</sup> sont gérées par le client et se voit ajouter le @ApplicationException
- les <sup>8</sup>runtime Exception<sup>8</sup> 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 TaskAlreadyDoneException  
extends Exception {  
}
```

268

## 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;  
}
```

269

## @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;  
}
```

270

## @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

271

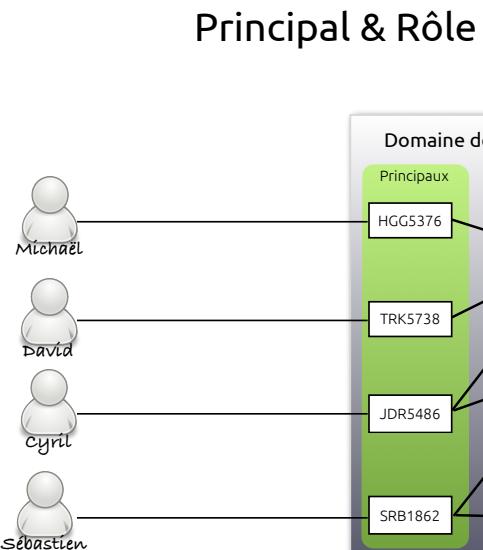
## Avantages & inconvénients BMT

- 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

272

# EJB & Sécurité

273



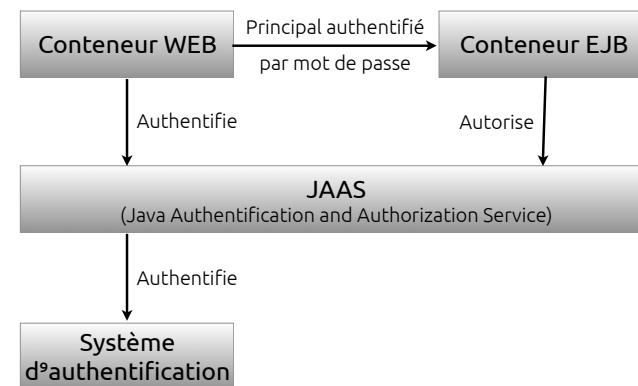
275

## 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 un politique de sécurité
- JAAS : Java Authentication and Authorization Service

274

## Principe de fonctionnement



276

## Support de la sécurité

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

277

## 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

279

## Sécurité déclarative

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

278

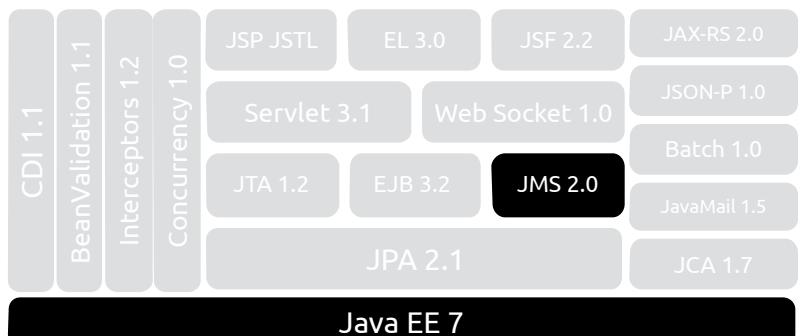
## Sécurité programmatique

```
@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  
    }  
}
```

280

# Java EE 7

## Enterprise Java Bean



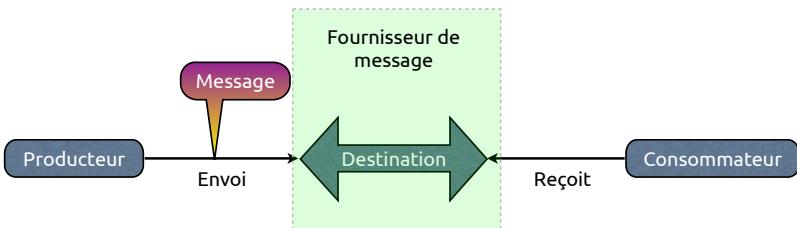
281

# Message

- Communication asynchrone
- Faiblement couplée
- Basé sur MOM

282

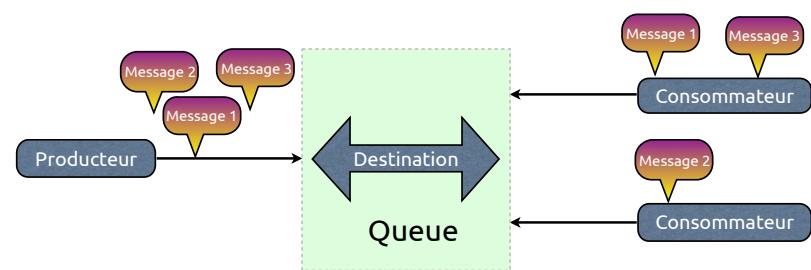
## MOM



- 2 stratégies
  - Point-To-Point (PTP)
  - Publish-Subscribe (pub-sub)
- ActiveMQ, SonicMQ, IBM Websphere MQ, etc...

283

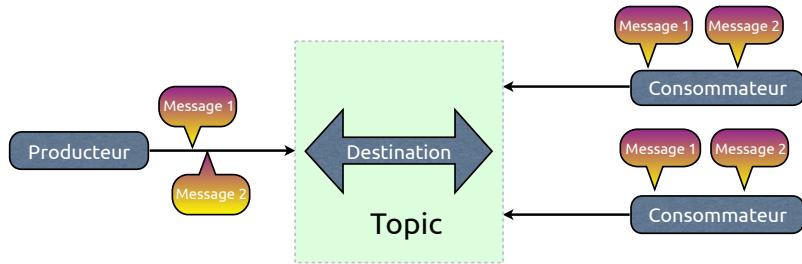
## Point-To-Point



- Pas de garantie d'ordre de délivrance
- Si il y a plusieurs récepteurs, le choix du récepteur est aléatoire

284

## 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)

285

## 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
    }
}
```

287

## 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

286

## 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

288

## Intérêt des MDBs

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

289

## 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
        }
    }
}
```

291

## 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)

290

## 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
        }
    }
}
```

292

## @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

293

## 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.

295

## @ActivationConfigProperty

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

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

294

## 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)

296

## 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

297

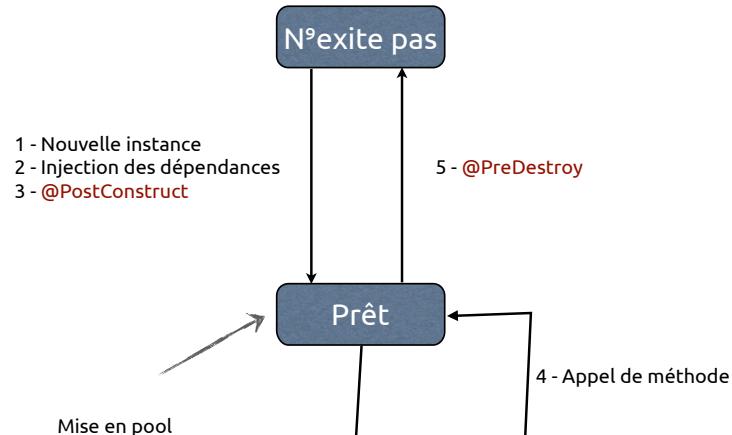
## MDB producteur (1)

```
@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();
    }
}
```

299

## Cycle de vie

Semblable au Stateless



298

## 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();
    }
}
```

300

## 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

301

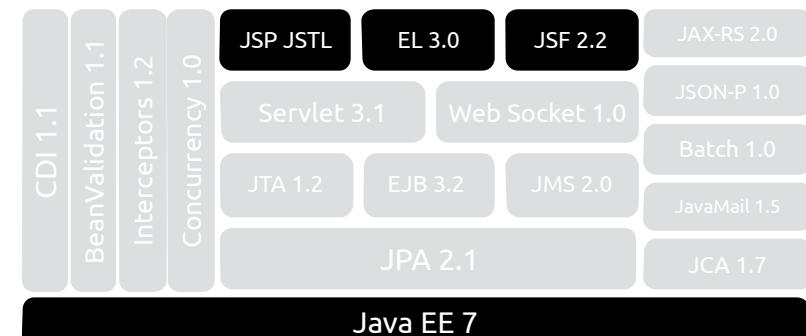
## Quelques rappels

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

303

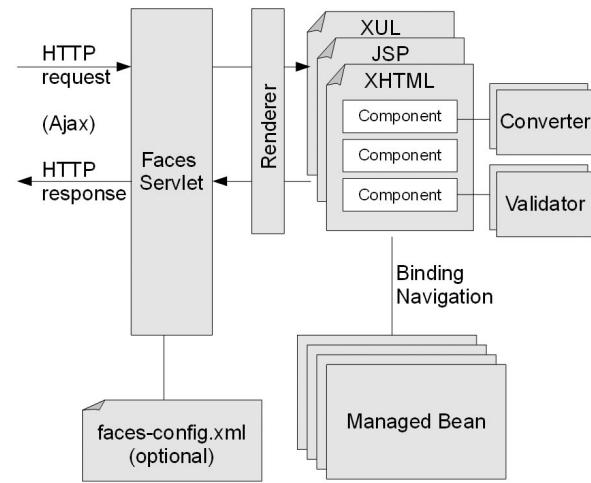
## Java EE 7

### La couche de présentation



302

## JSF : les briques...



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

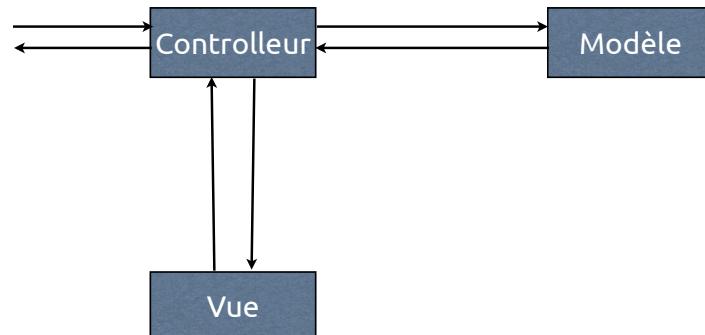
304

## 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

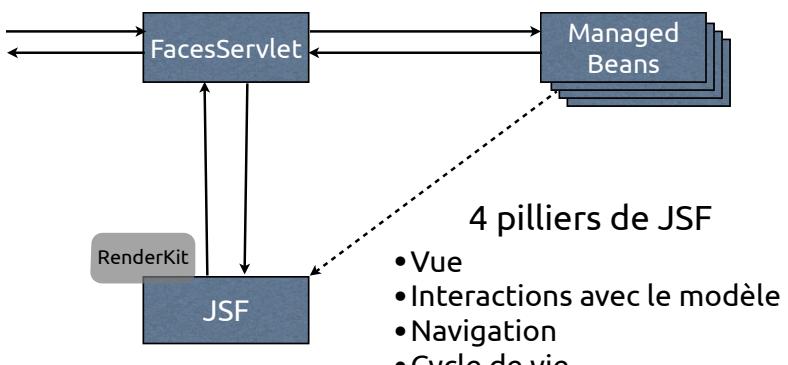
305

## MVC : rappel



306

## MVC : JSF



307

## 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)



308

## Exemple JSP non XML

login   
 password

```
<%@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>
```

309

## Exemple JSP XML

login   
 password

```
<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>
```

310

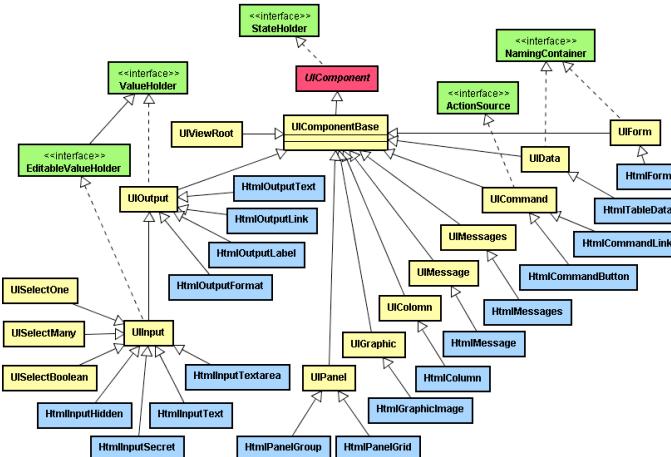
## Exemple Facelets XHTML

login   
 password

```
<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>
```

311

## Les Composant (UIComponent)



312

## 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 <sup>8</sup> custom <sup>8</sup> indépendante de tout kit de rendu (ex: f:selectItem, f:validateLength, etc...)
http://xmlns.jcp.org/jsf/facelets	<b>ui</b>	Support de template
http://xmlns.jcp.org/jsf/composite	<b>composite</b>	Support pour les composants <sup>8</sup> personnalisé <sup>8</sup>

313

## 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

315

## 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

314

## 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 <sup>8</sup>img<sup>8</sup>

316

## 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**

317

## Composant : message

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

319

## 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**)

318

## 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)

320

## 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

321

## tag JSF : les attributs DHTML

- onBlur : l'élément perd le focus
- onClick : clique souris sur l'élément
- onDoubleClick : double clique souris sur l'élément
- onFocus : l'élément gagne le focus
- onKeyDown, onKeyUp : touche clavier enfoncée et relâchée
- onKeyPress : touche clavier pressée puis relâchée
- onMouseDown, onMouseUp : bouton souris enfoncé et relâché
- onMouseOut, onMouseOver : curseur souris sort et entre
- onReset : formulaire réinitialisé
- onSelect : texte sélectionné dans un champs texte
- onSubmit : formulaire soumis

323

## 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

322

## Interactions avec le modèle

- Utilisation de EL pour associer les UIComponent à des propriétés du modèle
- La conversion se fait automatiquement
- Une 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

324

## 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 objets implicites
- S'appuie sur les conventions de nommage JavaBeans
- EL peut aussi pointer vers des méthodes

325

## 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  
}
```

327

## Les managed beans

- C'est l'<sup>9</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)

326

## 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>
```

```
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">  
<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>
```

328

## 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

329

## Managed bean exemple (jsf) binding Old flavor

```
<h:form>                                         login.xhtml
  <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.xml
<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>
```

331

## 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
}
```

330

## 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**

332

## 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.

333

## 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>`

334

## 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>
```

335

## Validateur spécialisé

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

336

## Validateur spécialisé Backing bean

```

<h:form>                                         login.xhtml
    <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>

public class Login {                               login.java
.....
    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"));
        }
    }
.....
}

```

337

## 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"));
        }
    }
}

```

338

## Validateur spécialisé Classe spécialisée - old flavor

```

<h:form>                                         login.xhtml
    <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>

<?xml version='1.0' encoding='UTF-8'?>          faces-config.xml
<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>

```

339

## 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 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"));
        }
    }
}

```

340

## 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:panelGrid>
    <h:commandButton id="btnLogIn" value="Connection" action="#{login.checkLogin}" />
</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">...
<validator>
    <validator-id>loginLength</validator-id>
    <validator-class>org.bn.biois.web.LoginValidator</validator-class>
</validator>...
</faces-config>
```

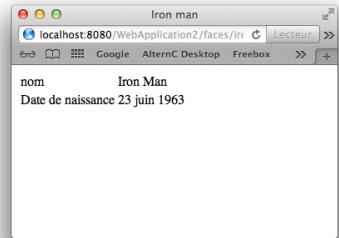
341

## convertisseur explicite

### Exemple

```
ironman.xhtml
<h:panelGrid columns="2">
    <h:outputLabel for="username" value="nom"/>
    <h:outputLabel id="username" value="#{ironman.name}">
        <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() {}
    ....
}
```



343

## 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 sont :  
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>

342

## 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 ... />

344

## convertisseur spécialisé old flavor

```
ironman.xhtml
<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>
```

```
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">
...
<converter>
    <converter-id>phoneNumberConverter</converter-id>
    <converter-class>org.univ.blois.web.PhoneConverter</converter-class>
</converter>
...
</faces-config>
```

345

## 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

347

## 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);
    }
}
```

346

## 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>
```

348

## 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";  
        }  
    }  
....  
}
```

349

## 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>
```

350

## 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>
```

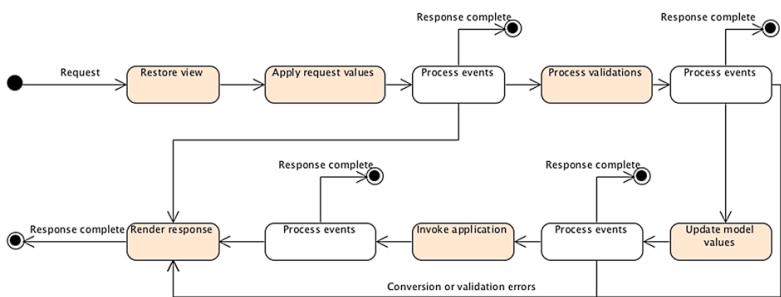
351

## 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

352

## Cycle de vie JSF



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

353

## 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

354

## 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()`

355

## 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()`

356

## 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()`

357

## 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

359

## 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`

358

## Cycle de vie JSF

### Immediate

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

360

## 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

361

That's all folks...

Enfin presque !

363

## Librairies additionnelles

Il existe d'autres librairies propriétaires ou open- source qui permettent d'apporter d'autres composant 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>

362

## Bonnes pratiques

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

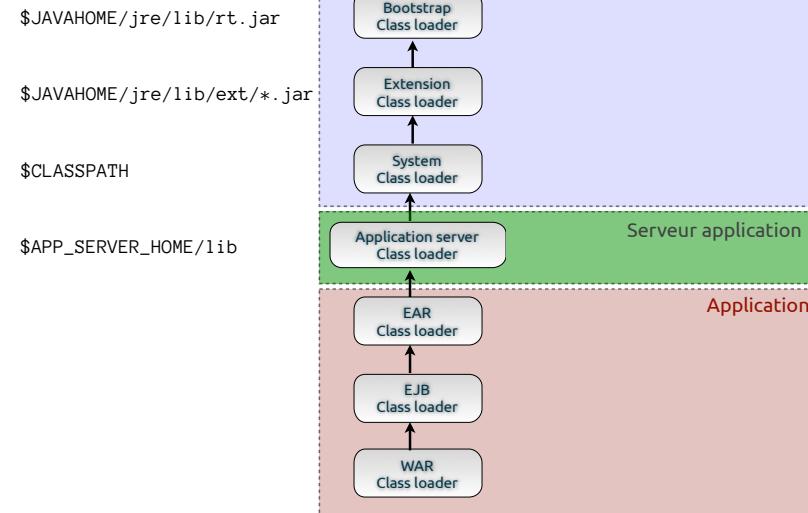
364

## 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

365

## Class loading



366

## 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 inexisteante

367