

# MODULE 12

# ENTITY BEANS

# TABLE OF CONTENT

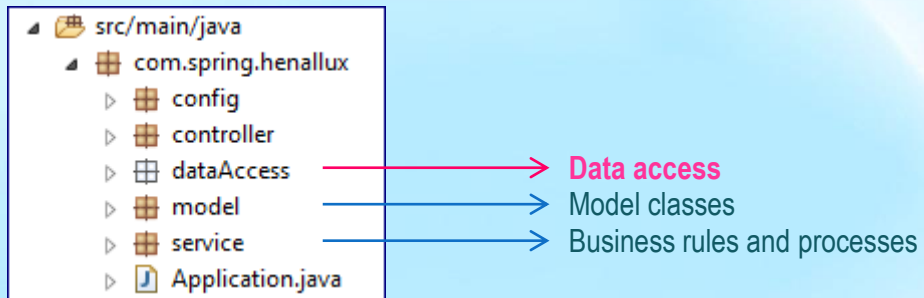
- Entity Bean
- Entity Bean Annotations
- Entity Bean States
- Data Repository
- Entity Class <> Model Class
- DAO Class
- Customized Query

# Entity Bean

- ▶ Server-side component representing persistent data maintained in a database
- ▶ The link between the application and the database
- ▶ Entity beans mirror the data model
  - Each entity bean (or entity class) is **the mirror of one table**
- ⇒ Access these entities rather than directly accessing the DB

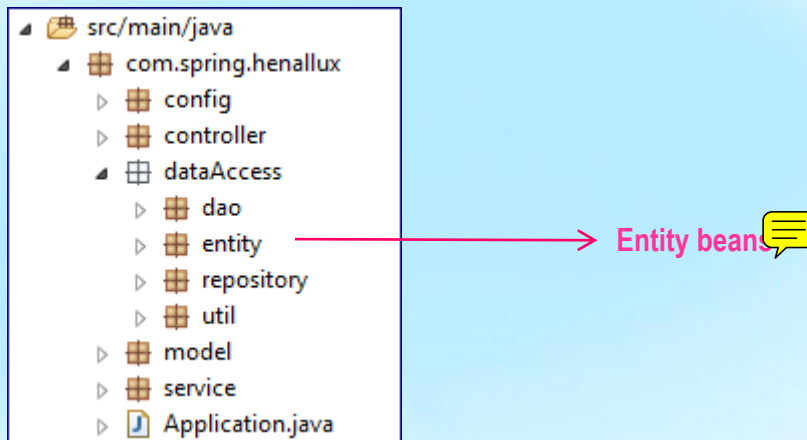
# Entity Bean

- ▶ Structure of the application including data access



# Entity Bean

- ▶ Create *entity* package in *dataAccess* package



# Entity Bean

- ▶ E.g, The MySQL **Book** table

book	
<u>isbn</u>	→ String
nbpages[0..1]	→ Number
title[0..1]	→ String
editiondate[0..1]	→ Date

# Entity Bean





Table Name:  Schema: **bookdb**

Collation:  Engine:

Comments:

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
isbn	VARCHAR(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
nbpages	INT(6)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
title	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
editiondate	DATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	



```
CREATE TABLE `book` (  
  `isbn` varchar(20) NOT NULL,  
  `nbpages` int(6) DEFAULT NULL,  
  `title` varchar(100) DEFAULT NULL,  
  `editiondate` date DEFAULT NULL,  
  PRIMARY KEY (`isbn`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```



# Entity Bean

- ▶ A Entity Bean (or Entity class) is a POJO with
  - **@Entity** annotation
  - No-argument default constructor
    - But may have other constructors
  - **Private** persistent properties
    - ↳ Accessed through public getters and setters
  - Neither the class nor its methods nor its persistent instance variables declared final
  - The class declared **Serializable** if entities are transferred
    - E.g. through remote call



# Entity Bean Annotations

- ▶ Annotation on Entity class
  - **@Entity**
  - **@Table** : table in the database corresponding to the entity class
    - **name** attribute
- ▶ If no **@Table**
  - The table name will be the class name

# Entity Bean Annotations

- ▶ Annotation on instance variable (property)
  - **@Id** : identifier
  - **@Column** : column of the table corresponding to the property
    - **name** attribute

# Entity Bean Annotations

► E.g,

```
import javax.persistence.*;

@Entity
@Table(name="book")
public class BookEntity {

    @Id
    @Column(name="isbn")
    private String isbn;

    @Column(name="title")
    private String title;

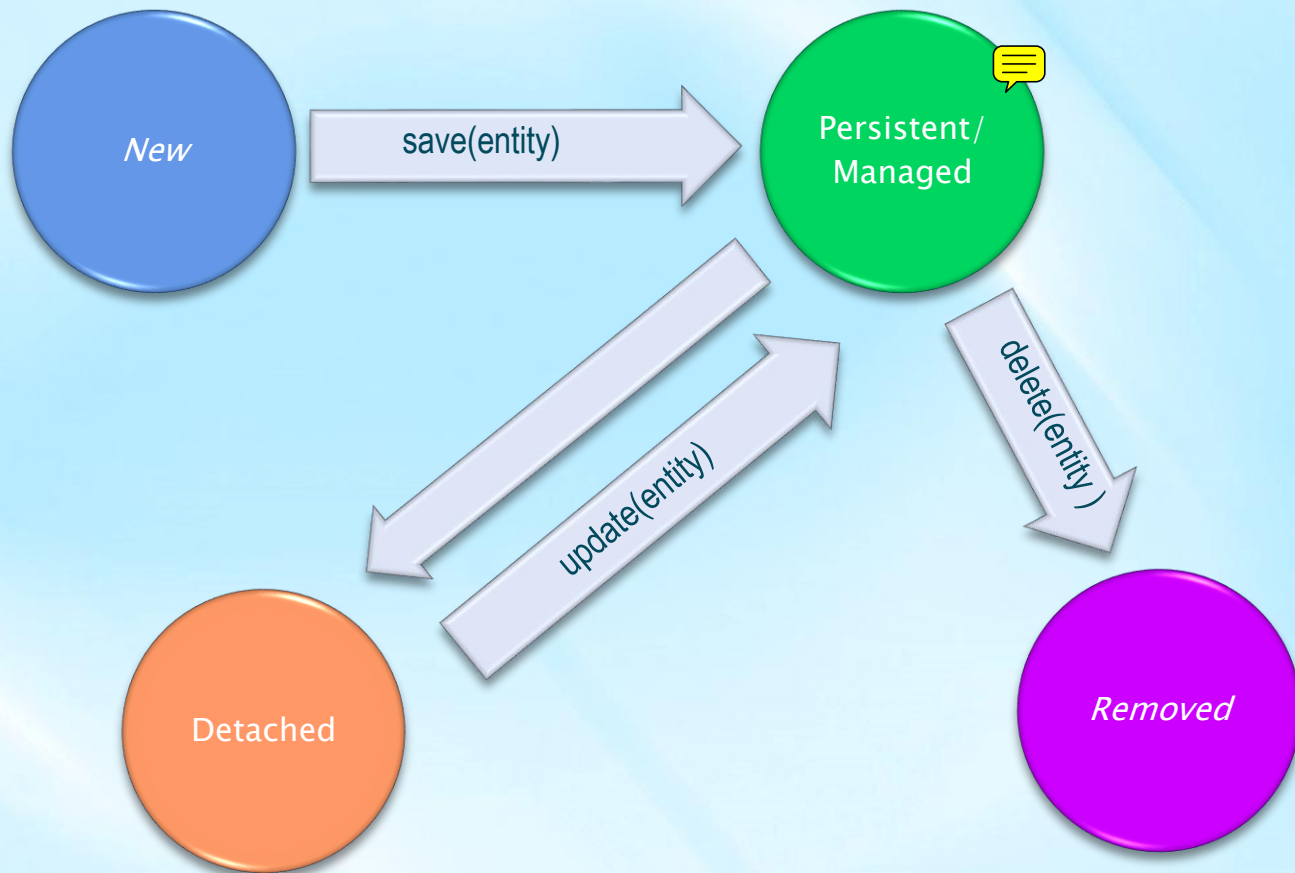
    @Column(name="nbpages")
    private Integer nbPages;

    @Column(name="editiondate")
    private java.util.Date editionDate;
```

# Entity Bean States

- ▶ Entity instances are managed by invoking operations on them
- ▶ Entity instances are in one of four states
  - New (transient)
  - Persistent (Managed)
  - Detached
  - Removed

# Entity Bean States



# Entity Bean States

- ▶ New (transient)
  - New entity instances created (after call of the **new** operator)
  - New entity instance not mapped to any database table row
- ▶ Managed (persistent)
  - After call of the **save** method
  - Entity instance associated with database table row
- ▶ Detached
  - Entity instances manipulated by the client
  - Entity instances not currently mapped to any database table row
- ▶ Removed
  - After call of the **delete** method
  - Entity instance mapped to a database table row and scheduled for removal from the data store

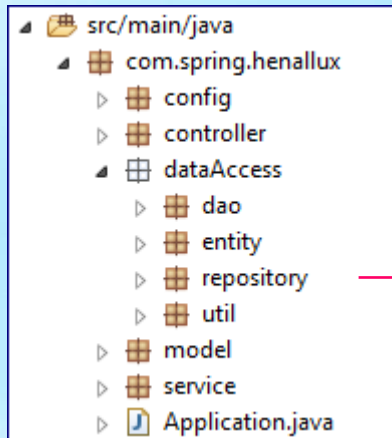
# Data Repository

- ▶ Spring Data repository abstraction
  - To significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores
- ▶ Interface created for each entity class containing access methods
  - **CRUD**
  - To create, locate, modify and delete entities in the database



# Data Repository

- ▶ Create *repository* package in *dataAccess* package



Repository interfaces



# Data Repository

Interface générique

- ▶ Create an interface extending ***JpaRepository<T,ID>***

- *org.springframework.data.jpa.repository.JpaRepository*

- ▶ Arguments

- T : The entity class to manage
  - ID : The id type of the entity class

Les 2 arguments :

T => entity classe

ID => type de l'identifiant

- ▶ ***@Repository*** annotation

# Data Repository

- ▶ E.g,

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.spring.henallux.entity.BookEntity;

@Repository
public interface BookRepository extends JpaRepository<BookEntity, String>{
}
```

- ▶ N.B. *No implementation to write for the repository interface!*

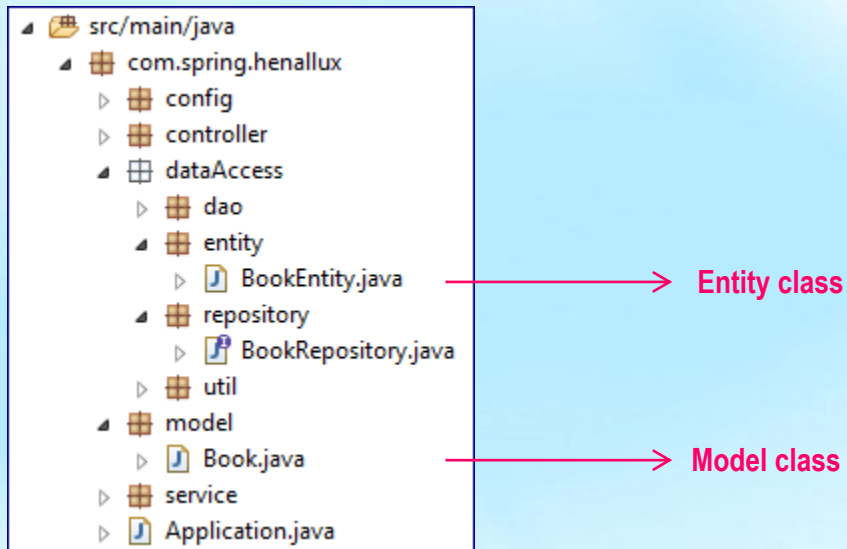
# Data Repository

- ▶ Some of available methods (no need to implement!)
  - `<S extends T> S save(S entity)` soit un objet de bookEntity ou une sous classe
  - `void delete(T entity)`
  - `T findOne(ID primaryKey)`
  - `List<T> findAll()` retourne une arrayList
  - `Long count()` nbr ligne dans la table
  - `boolean exists(ID primaryKey)`

# Entity Class <> Model Class

- ▶ In order to be reused in other applications and contexts, model classes should not be dependant of persistence choice
  - E.g, in another application, data could be persisted in xml files instead of relational databases
- ▶ Neither **@Table** nor **@Column** annotations on Model classes
  - ⇒ **Model classes ≠ Entity classes**

# Entity Class <> Model Class



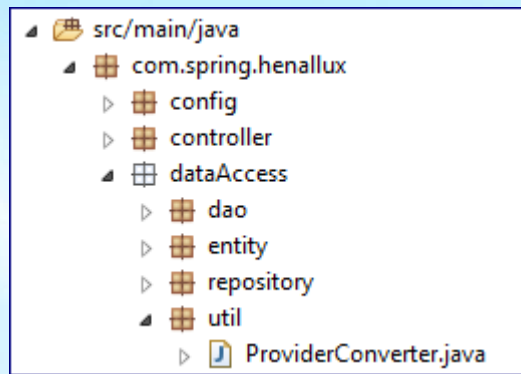
# Entity Class <> Model Class

Model classes  $\neq$  Entity classes

⇒ Create a *util* package in *dataAccess* package

- ▶ Containing Converter classes

- To convert model objects to entity objects
- To convert entity objects to model objects
- Use *@Component* annotation



# Entity Class <> Model Class

► E.g,

```
@Component
public class ProviderConverter {

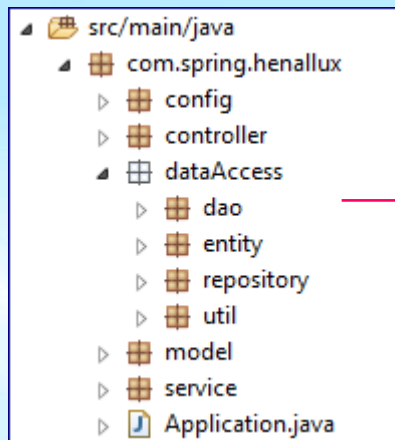
    public BookEntity bookModeltoBookEntity(Book book) {
        BookEntity bookEntity = new BookEntity();
        bookEntity.setIsbn(book.getIsbn());
        bookEntity.setTitle(book.getTitle());
        bookEntity.setNbPages(book.getNbPages());
        bookEntity.setEditionDate(book.getEditionDate());
        return bookEntity;
    }

    public Book bookEntityToBookModel(BookEntity bookEntity) {
        Book book = new Book();
        book.setIsbn(bookEntity.getIsbn());
        book.setTitle(bookEntity.getTitle());
        book.setNbPages(bookEntity.getNbPages());
        book.setEditionDate(bookEntity.getEditionDate());
        return book;
    }
}
```



# DAO Class

- ▶ Create a **dao** package in **dataAccess** package
- ▶ DAO classes contain methods **to persist Model classes**
  - Call of corresponding repository
    - Through bean injection
    - Use **@Autowired**



→ DAO classes



# DAO Class

► E.g,

interface

```
@Service
public class BookDAO {

    @Autowired // Récupération des référence de ce qui est auto généré par spring
    private BookRepository bookRepository;

    @Autowired
    private ProviderConverter providerConverter;

    public Book save(Book book) {
        BookEntity bookEntity = providerConverter.bookModelToBookEntity(book);
        bookEntity = bookRepository.save(bookEntity);
        return providerConverter.bookEntityToBookModel(bookEntity);
    }

    public ArrayList<Book> getAllBooks()
    {
        List<BookEntity> bookEntities = bookRepository.findAll();
        ArrayList<Book> books = new ArrayList<>();
        for (BookEntity entity : bookEntities)
        {
            Book book = providerConverter.bookEntityToBookModel(entity);
            books.add(book);
        }
        return books;
    }
}
```

# Customized Query

- ▶ Additional queries can be generated in repository interface
  - By combining logical keywords and attribute names

Logical keyword	Keyword expressions
AND	And
OR	Or
AFTER	After, IsAfter
BEFORE	Before, IsBefore
CONTAINING	Containing, IsContaining, Contains
BETWEEN	Between, IsBetween
ENDING_WITH	EndingWith, IsEndingWith, EndsWith
EXISTS	Exists
FALSE	False, IsFalse
GREATER_THAN	GreaterThan, IsGreaterThan
GREATER_THAN_EQUALS	GreaterThanEqual, IsGreaterThanEqual
IN	In, IsIn
IS	Is, Equals, (or no keyword)

<http://docs.spring.io/spring-data/jpa/docs/1.4.1.RELEASE/reference/html/jpa.repositories.html>

# Customized Query

## ► Continue

IS_NOT_NULL	NotNull, IsNotNull
IS_NULL	Null, IsNull
LESS_THAN	LessThan, IsLessThan
LESS_THAN_EQUAL	LessThanEqual, IsLessThanEqual
LIKE	Like, IsLike
NEAR	Near, IsNear
NOT	Not, IsNot
NOT_IN	NotIn, IsNotIn
NOT_LIKE	NotLike, IsNotLike
REGEX	Regex, MatchesRegex, Matches
STARTING_WITH	StartingWith, IsStartingWith, StartsWith
TRUE	True, IsTrue
WITHIN	Within, IsWithin

<http://docs.spring.io/spring-data/jpa/docs/1.4.1.RELEASE/reference/html/jpa.repositories.html>

# Customized Query

► E.g,

And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull	findByAgeIsNull	... where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1

<http://docs.spring.io/spring-data/jpa/docs/1.4.1.RELEASE/reference/html/jpa.repositories.html>

# Customized Query

► E.g (continue),

OrderBy	<code>findByAgeOrderByLastnameDesc</code>	<code>... where x.age = ?1 order by x.lastname desc</code>
Not	<code>findByLastnameNot</code>	<code>... where x.lastname &lt;&gt; ?1</code>
In	<code>findByAgeIn(Collection&lt;Age&gt; ages)</code>	<code>... where x.age in ?1</code>
NotIn	<code>findByAgeNotIn(Collection&lt;Age&gt; age)</code>	<code>... where x.age not in ?1</code>
True	<code>findByActiveTrue()</code>	<code>... where x.active = true</code>
False	<code>findByActiveFalse()</code>	<code>... where x.active = false</code>

<http://docs.spring.io/spring-data/jpa/docs/1.4.1.RELEASE/reference/html/jpa.repositories.html>

# Customized Query

- ▶ E.g (continue),

StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	... where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	... where x.firstname like ?1 (parameter bound wrapped in %)

<http://docs.spring.io/spring-data/jpa/docs/1.4.1.RELEASE/reference/html/jpa.repositories.html>