

# Spring & Hibernate : Concepts Fondamentaux

## 1. ORM (Object-Relational Mapping)

Définition : Technique de programmation permettant de : - Convertir des objets Java ! Tables SQL - Gérer les relations (1-1, 1-N, N-N) - Exécuter des opérations CRUD via des objets

## 1. ORM (Object-Relational Mapping)

Avantages : - Abstraction de la base de données - Réduction du code SQL manuel - Gestion automatique des transactions - Mapping des types complexes (JSON, ENUM, etc)

## 2. Hibernate en Bref

Principales fonctionnalités : - Implémentation JPA (Java Persistence API) - Lazy Loading - Cache de 1er et 2nd niveau - HQL (Hibernate Query Language) - Migrations avec Flyway/Liquibase

## 2. Hibernate en Bref

Architecture :

```
[Application] ! [SessionFactory] ! [Database]
    |
    +----- [Session]
```

## 3. Modularité de Spring

Principes : - Inversion de Contrôle (IoC) - Injection de Dépendances (DI) - Composants modulaires interchangeables

## 3. Modularité de Spring

Modules clés : - Core : Beans, DI, Context - DAO : Transaction management - ORM : Intégration Hibernate/JPA - Web : MVC, REST

## 3. Modularité de Spring

```
<!-- Exemple de dépendances modulaires -->
```

```

<dependency>
Ê   <groupId>org.springframework</groupId>
Ê   <artifactId>spring-orm</artifactId>
Ê   <version>5.3.30</version>
</dependency>

```

## 4. Integration Spring + Hibernate

### Configuration Type (applicationContext.xml)

```

<!-- DataSource -->
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource">
Ê   <property name="jdbcUrl" value="jdbc:postgresql://localhost/mydb"/>
Ê   <property name="username" value="user"/>
Ê   <property name="password" value="pass"/>
</bean>

<!-- SessionFactory -->
<bean id="sessionFactory"
Ê   class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
Ê   <property name="dataSource" ref="dataSource"/>
Ê   <property name="packagesToScan" value="com.example.model"/>
Ê   <property name="hibernateProperties">
Ê       <props>
Ê           <prop key="hibernate.dialect">
org.hibernate.dialect.PostgreSQLDialect</prop>
Ê           <prop key="hibernate.hbm2ddl.auto">update</prop>
Ê       </props>
Ê   </property>
</bean>

<!-- Transaction Manager -->
<bean id="transactionManager"
Ê   class="org.springframework.orm.hibernate5.HibernateTransactionManager">
Ê   <property name="sessionFactory" ref="sessionFactory"/>
</bean>

```

## 5. Declaration d'Entités

### Exemple : Entité Book

```

@Entity
@Table(name = "books")
public class Book {

```

```

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

    @Column(nullable = false, length = 100)
    private String title;

    @ManyToOne
    @JoinColumn(name = "author_id")
    private Author author;

    // Getters/Setters
}

```

## Exemple : Entité Book

Annotations clés : - **@Entity** : Déclare une classe comme entité - **@Table** : Personnalise le nom de table - **@Id** + **@GeneratedValue** : Clé primaire - **@Column** : Mapping champ ! colonne - **@ManyToOne**, **@OneToMany** : Relations

## 6. Opérations de Base

### Repository Spring

```

@Repository
@Transactional
public class BookRepository {

    @Autowired
    private SessionFactory sessionFactory;

    public void save(Book book) {
        sessionFactory.getCurrentSession().save(book);
    }

    public Book findById(Long id) {
        return sessionFactory.getCurrentSession().get(Book.class, id);
    }
}

```

### Utilisation dans un Service

```

@Service
public class BookService {

```

```

@Autowired
private BookRepository repository;

@Transactional
public Book createBook(String title, Author author) {
    Book book = new Book();
    book.setTitle(title);
    book.setAuthor(author);
    repository.save(book);
    return book;
}
}

```

## 7. Bonnes Pratiques

- ✧ Utiliser `@Transactional` au niveau service
- ✧ Éviter les `Open Session In View`
- ✧ Préférer le chargement paresseux (Lazy Loading)
- ✧ Valider les entités avec Bean Validation
- ✧ Utiliser DTO pour les couches d'exposition

## 8. Pièges Courants

- ✧ N+1 Problem : Résolu par `JOIN FETCH` en HQL
- ✧ Transaction Not Active : Vérifier les annotations `@Transactional`
- ✧ `LazyInitializationException` : Initialiser les relations dans la transaction
- ✧ Versionning : Gérer les conflits avec `@Version`

## 9. Alternatives

- ✧ Spring Data JPA : Abstraction supplémentaire
- ✧ `JdbcTemplate` : Pour les opérations SQL directes
- ✧ MyBatis : Mapper XML ! SQL