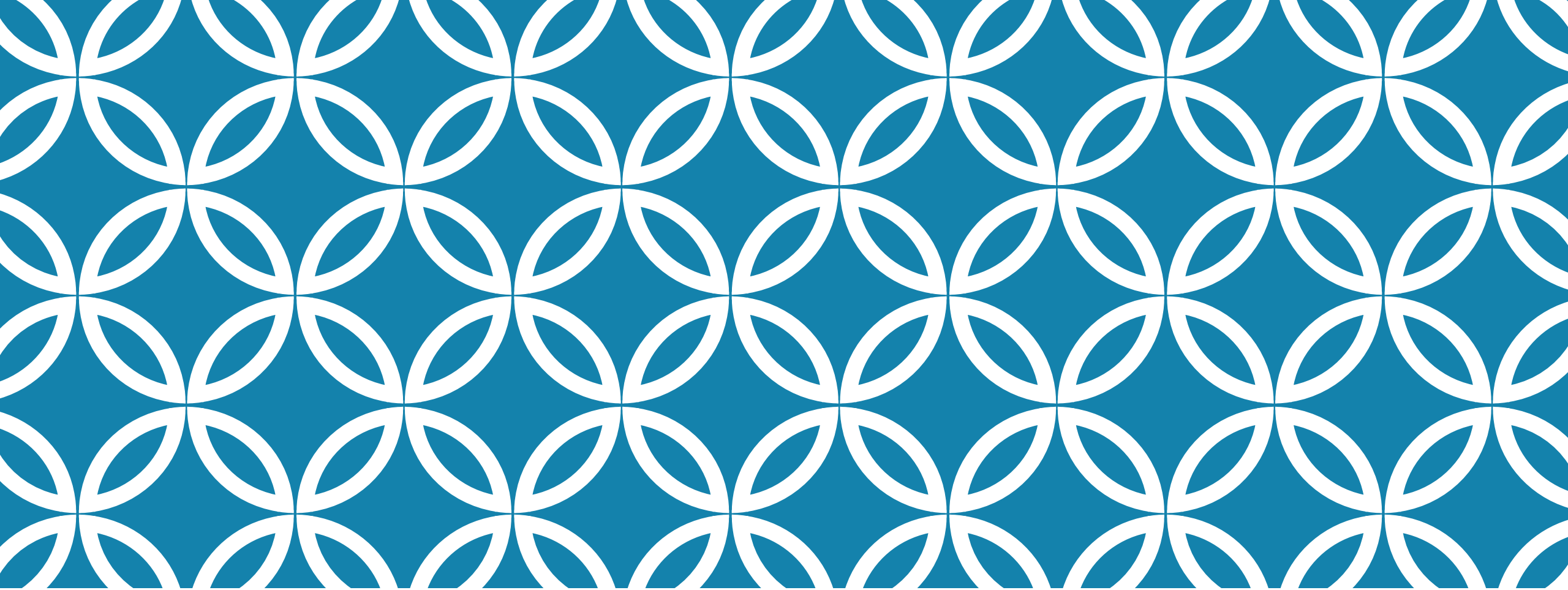
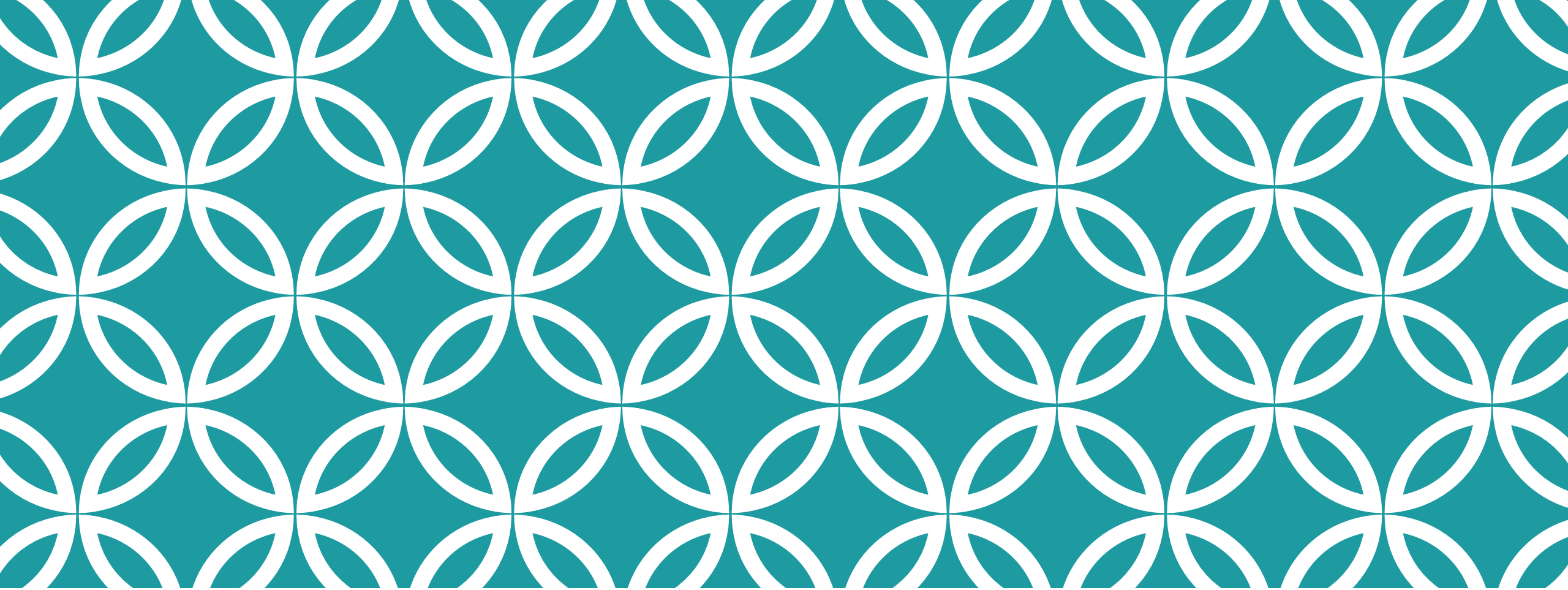


JAVA EE

Jérémy PERROUAULT



SPRING ET HIBERNATE

Spring & Hibernate
Combot gagnant !

PRÉSENTATION SPRING JPA

Spring va prendre le pas

- Sur la déclaration JNDI du DataSource
- Sur la configuration JPA (*persistence.xml*)
- Configuration dans "application-context.xml"

Les objets EJB DAO deviennent des *@Repository*

Les transactions sont gérées par service *@Transactional*

CONFIGURATION

Utilisation de 3 beans

- DataSource
- EntityManagerFactory
  - Utilise la référence du DataSource
  - Précise les options comme le Provider et les options du Provider
- TransactionManager
  - Utilise la référence de l' EntityManagerFactory

Activation des annotations @Transactional

- Utilise la référence du TransactionManager

CONFIGURATION

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <!-- ... -->
</bean>

<bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <!-- ... -->
</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>

<tx:annotation-driven transaction-manager="transactionManager" />
```

CONFIGURATION

```
<!-- On crée le DataSource -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <property name="url" value="jdbc:mysql://localhost:3306/tp_jpa" />
  <property name="username" value="root" />
  <property name="password" value="" />
  <property name="maxActive" value="10" />
</bean>
```

Création de la DataSource gérée par Spring

- Avec le pilote à utiliser
- L'URL de connexion à la base de données
- Les identifiants de connexion
- Le maximum de connexions simultanées actives

CONFIGURATION

```
<!-- On crée un entityManagerFactory local à partir de la dataSource -->
<bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <property name="packagesToScan" value="fr.ascadis.model" />

  <!-- On précise le provider ... -->
  <property name="jpaVendorAdapter">
    <!-- ... Hibernate ... -->
    <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
    <!-- ... Et ses options -->
      <property name="databasePlatform" value="org.hibernate.dialect.MySQL5InnoDBDialect" />
      <property name="showSql" value="true" />
      <property name="generateDdl" value="true" />
    </bean>
  </property>
</bean>
```

Création de l'EntityManagerFactory géré par Spring

- En lui passant la DataSource gérée par Spring
- En lui précisant le provider (implémentation JPA, Hibernate dans notre cas)
- En lui précisant les options du provider (les options de Hibernate)

CONFIGURATION

```
<!-- On crée le transactionManager pour JPA avec entityManagerFactory -->
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>
```

Création du JpaTransactionManager géré par Spring

- En lui passant l'EntityManagerFactory géré par Spring

CONFIGURATION

```xml
<!-- On active les annotations @Transactional avec transactionManager -->
<tx:annotation-driven transaction-manager="transactionManager" />

<!-- On active la translation d'exception -->
<bean class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor" />
```

Activation des annotations @Transactional

- Qu'on utilisera sur nos DAO
- Permet d'écouter toutes les méthodes de nos DAO grâce à AOP

Activation de la gestion des Exceptions Spring DAO

- Permet de traduire les Exceptions levées par Spring TX en Exception JPA/Hibernate
- Fonctionne sur les classes annotées de @Repository

UTILISATION

Déclaration du *bean DAO Spring*

- @Repository

Déclaration de l'utilisation des transactions Spring

- @Transactional

Injection de la dépendance

- @Autowired

CHARGEMENT LAZY

Utilisation d'un filtre OpenEntityManagerInViewFilter

- Garder EntityManager actif tout au long du chargement de la vue
- Utile pour les chargements Lazy
  - Plus besoin des requêtes intermédiaires de chargement avec « fetch »
- Filtre à mapper sur les URL nécessitant Spring JPA
  - Peut être toutes les ressources

CHARGEMENT LAZY

```
<!-- Utilisation d'un filtre qui permet de garder un EntityManager actif dans la vue -->
<filter>
  <filter-name>openEntityManagerInViewFilter</filter-name>
  <filter-class>org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter</filter-class>

  <init-param>
    <param-name>singleSession</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>openEntityManagerInViewFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Déclaration du filtre dans le fichier web.xml

- Initialisation du paramètre « singleSession » à vrai
- Mappé sur toutes les ressources (actif sur toutes les URL)

EXERCICE

Récupérer les classes Model et DAO du projet TP-JPA

Configurer Hibernate avec Spring

Modifier les DAO

Créer une Servlet

- Afficher la liste des clients et fournisseurs
- Afficher la liste des produits
- Afficher la liste des produits pour un fournisseur

- S'aider de http://formations.ascadis.fr/m2i/jee/05-03__configuration-spring-jpa.html

EXERCICE

Manipulation Spring

EXERCICE

Acquérir et se familiariser avec

- Spring
- Spring beans
- Injection de Dépendances

EXERCICE

Implémenter Spring en partant du projet SQL

- Voir la page http://formations.ascadis.fr/m2i/jee/05-03__tp-tetris.html