DEF (DÉVELOPPEMENT/ENCADREMENT/FORMATION)

FORMATION JAVA/J2EE

Java/J2EE

JAVA/J2EE: Objet/Connexion JDBC/Hibernate/JPA/Servlet/JSP/Spring MVC /Serveur d'application Tomcat

2014/2015

Plan du Cours

1. Mise en place de l'environnement de travail
2. Création et manipulations des objets en Java
3. TP1 : Connexion JDBC à la base de données
4. TP2 : Framework HIBERNATE
5. TP3 : Application web J2EE
6. TP4 : Implémentation et Utilisation de spring MVC
7. TP5 : Développement d'un site de commerce électronique en J2EE avec Spring MVC, Hibernate, JPA, JSP , MySql…

QU'EST CE QUE JAVA? (1)

- Langage de programmation orienté objet (Classe, Objet, Héritage, Encapsulation et Polymorphisme)
- Avec java on peut créer des application multiplateformes. Les applications java sont portables. C'est-à-dire, on peut créer une application java dans une plateforme donnée et on peut l'exécuter sur n'importe quelle autre plateforme.
- Le principe de java est : Write Once Run Every Where
- Open source: On peut récupérer le code source de java. Ce qui permet aux développeurs, en cas de besoin, de développer ou modifier des fonctionnalités de java.

QU'EST CE QUE JAVA? (2)

- Java est utilisé pour créer :
  - Des applications Desktop
  - Des applets java (applications java destinées à s'exécuter dans une page web)
  - Des applications pour les smart phones
  - Des applications embarquées dans des cartes à puces
  - Des application JEE (Java Entreprise Edition)
- Pour créer une application java, il faut installer un kit de développement java
  - JSDK : Java Standard Developpement Kit, pour développer les application DeskTop
  - JME : Java Mobile Edition, pour développer les applications pour les téléphones potables
  - JEE : Java Entreprise Edition, pour développer les applications qui vont s'exécuter dans un serveur d'application JEE (Web Sphere Web Logic, JBoss).
  - JCA : Java Card Edition, pour développer les applications qui vont s'éxécuter dans des cartes à puces.

Méthode Orienté Objet

- La méthode orientée objet permet de concevoir une application sous la forme d'un ensemble d'objets reliés entre eux par des relations
- Lorsque que l'on programme avec cette méthode, la première question que l'on se pose plus souvent est :
  - **«qu'est-ce que je manipule ? »,**
  - Au lieu de **« qu'est-ce que je fait ? ».**
- L'une des caractéristiques de cette méthode permet de concevoir de nouveaux objets à partir d'objets existants.
- On peut donc réutiliser les objets dans plusieurs applications.
- La réutilisation du code fut un argument déterminant pour venter les avantages des langages à objets.
- Pour faire la programmation orientée objet il faut maitriser les fondamentaux de l'orienté objet à savoir:
  - Objet et classe
  - Héritage
  - Encapsulation (Accessibilité)
  - Polymorphisme

Objet Java

- Un objet est une structure informatique définie par un état et un comportement
- Objet=état + comportement
  - L'état regroupe les valeurs instantanées de tous les attributs de l'objet.
  - Le comportement regroupe toutes les compétences et décrit les actions et les réactions de l'objet. Autrement dit le comportement est défini par les opérations que l'objet peut effectuer.
- L'état d'un objet peut changer dans le temps.
- Généralement, c'est le comportement qui modifie l'état de l'objet

Les classes

- Les objets qui ont des caractéristiques communes sont regroupés dans une entité appelé classe.
- La classe décrit le domaine de définition d'un ensemble d'objets.
- Chaque objet appartient à une classe
- Les généralités sont contenues dans les classes et les particularités dans les objets.
- Les objets informatique sont construits à partir de leur classe par un processus qui s'appelle l'instanciation.
- Tout objet est une instance d'une classe.

CARACTÉRISTIQUES D'UNE CLASSE

- Une classe est définit par: ses attributs & ses méthodes
- Les attributs permettent de décrire l'état de des objets de cette classe. Chaque attribut est défini par:
  - Son nom
  - Son type
  - Éventuellement sa valeur initiale
- Les méthodes permettent de décrire le comportement des objets de cette classe.
  - Une méthode représente une procédure ou une fonction qui permet d'exécuter un certain nombre d'instructions.
- Parmi les méthode d'une classe, existe deux méthodes particulières:
  - Une méthode qui est appelée au moment de la création d'un objet de cette classe. Cette méthode est appelée **CONSTRUCTEUR**
  - Une méthode qui est appelée au moment de la destruction d'un objet. Cette méthode s'appelle le **DESTRUCTEUR**

Problème

- On souhaite créer un site web de commerce électronique qui se compose de deux parties :
  - Une partie back office qui nécessite une authentification et qui permet de gérer les produits et les catégories
  - L'administrateur a la possibilités de gérer les catégories et les produits (Ajout, suppression, Edition, Modification et consultation) ainsi que les droits d'accès
- Une partie front office qui représentent la boutique virtuelle qui ne nécessite pas d'authentification. Dans cette partie l'utilisateur a la possibilités de :
  - Consulter toutes les catégories
  - Consulter les produits d'une catégorie
  - Consulter les produits sélectionnés
  - Chercher des produits par mot clé
  - Ajouter un produit avec une quantité au panier
  - Supprimer un produit du panier
  - Enregistrer le client et la commande des produits de son panier.

ARCHITECTURE

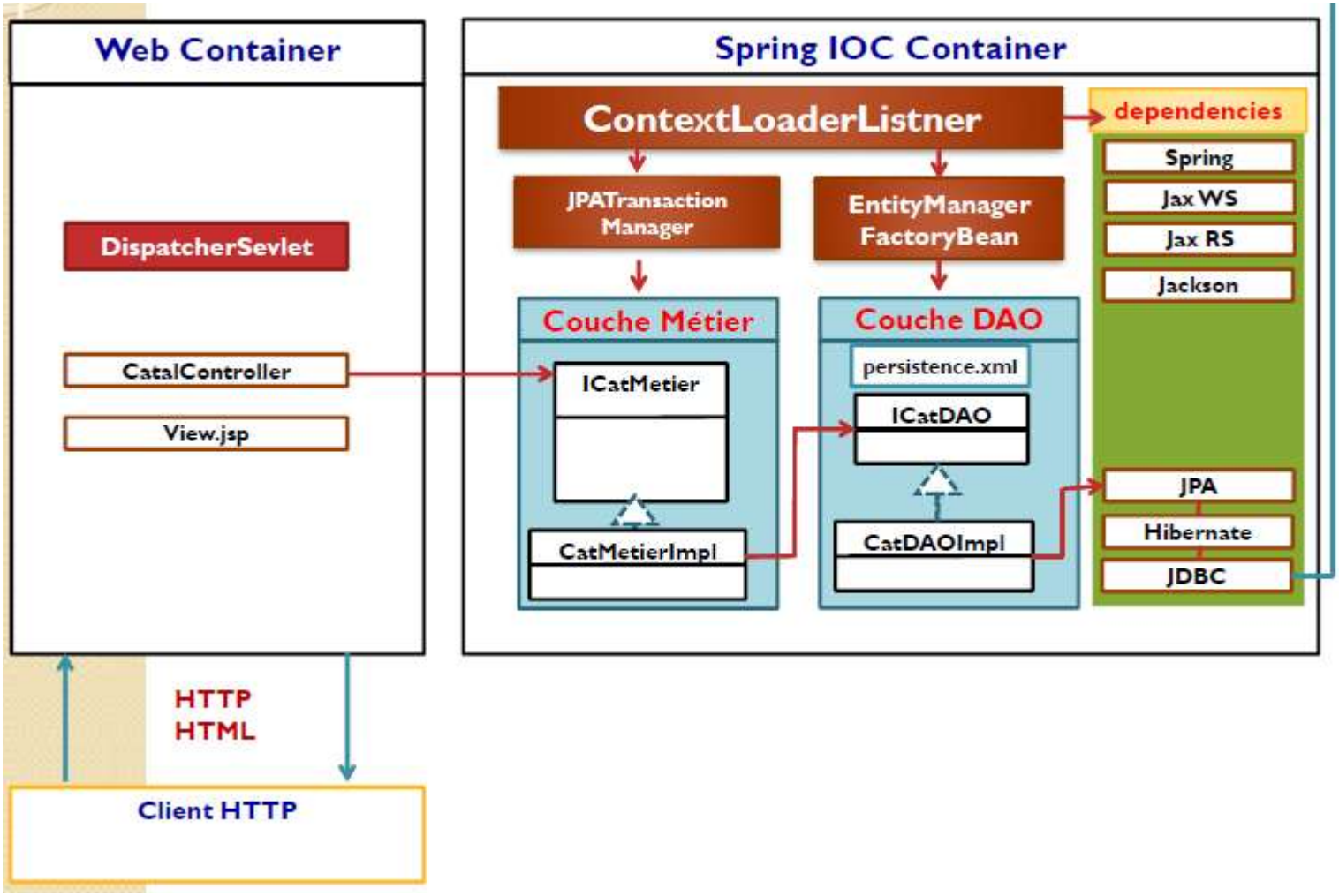
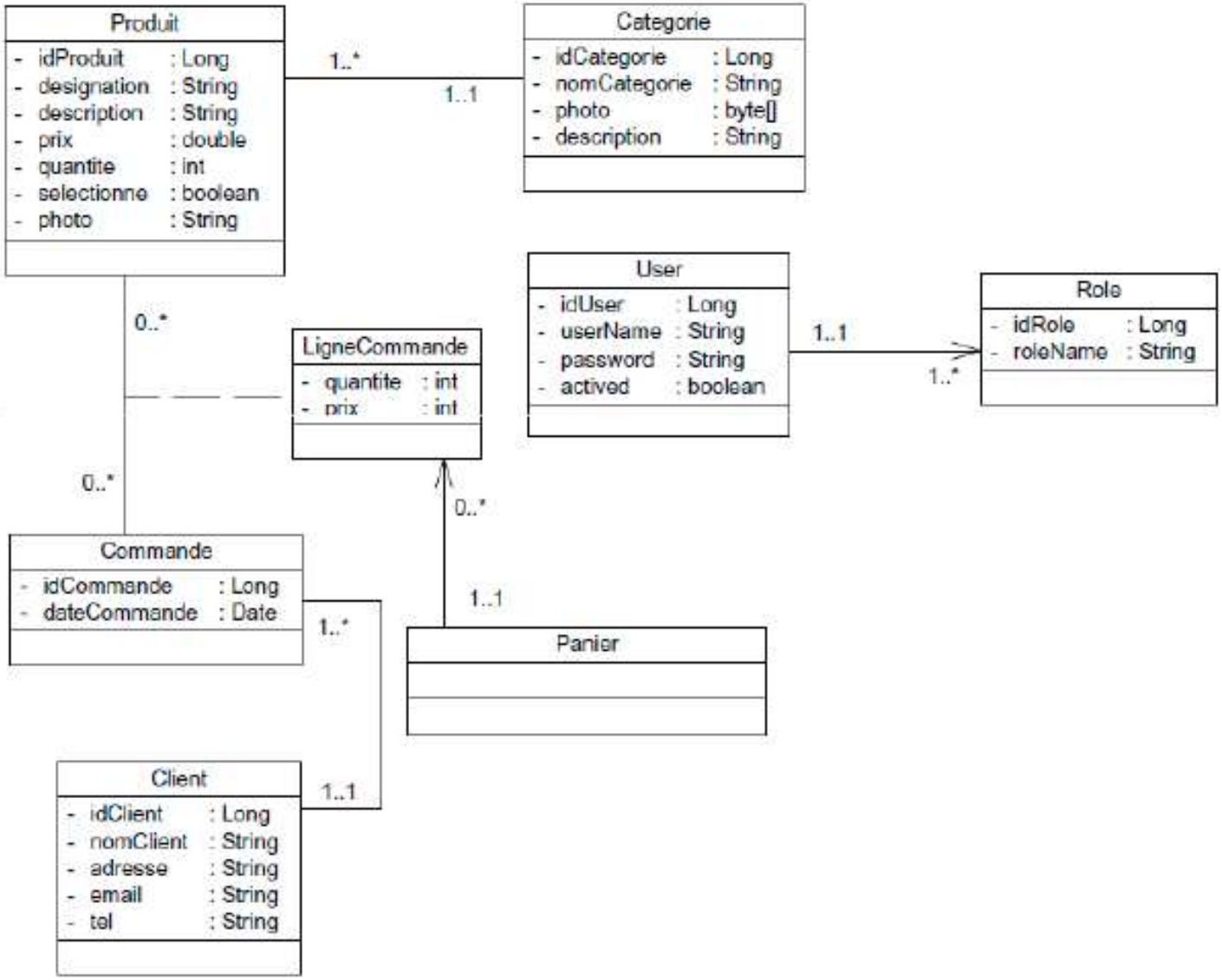


DIAGRAMME DE CLASSE



Mapping Objet/relationnel (1)

- Dans la pratique, on cherche toujours à séparer la logique de métier de la logique de présentation.
- On peut dire qu'on peut diviser une application en 3 couches:
  - La couche d'accès aux données: DAO
    - Partie de l'application qui permet d'accéder aux données de l'applications . Ces données sont souvent stockées dans des bases de données relationnelles .
  - La couche Métier:
    - Regroupe l'ensemble des traitements que l'application doit effectuer.
  - La couche présentation:
    - S'occupe de la saisie des données et de l'affichage des résultats;

Mapping Objet/relationnel (2)

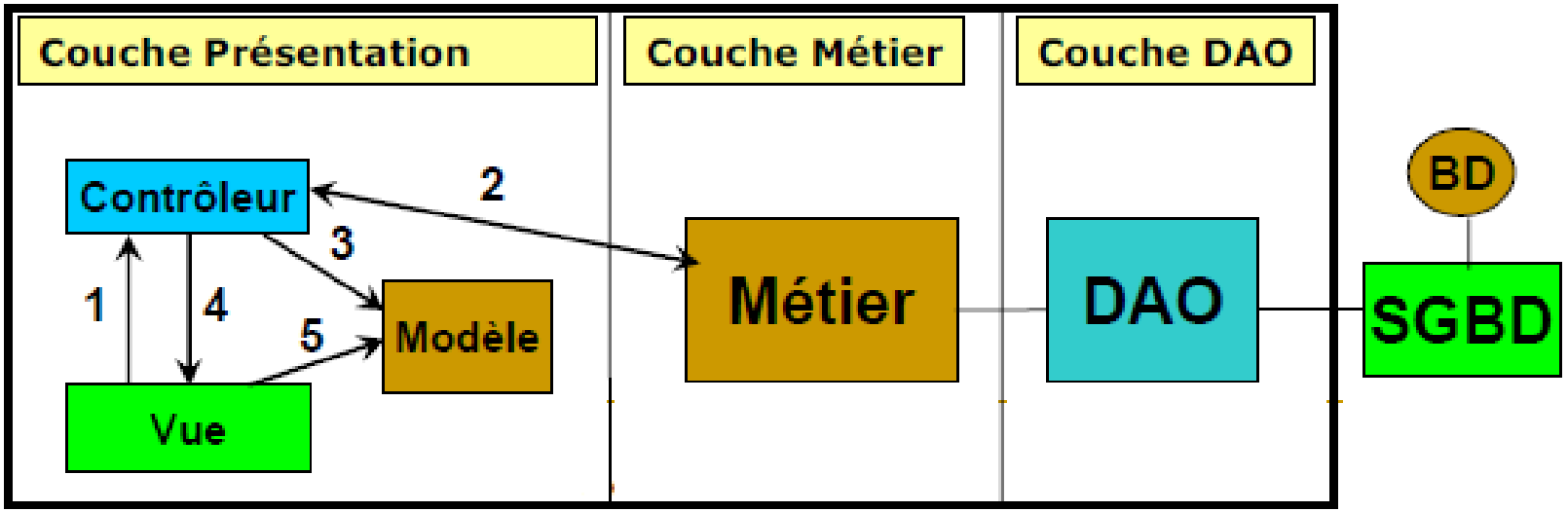
- D'une manière générale les applications sont orientée objet :
  - Manipulation des objet et des classes
  - Utilisation de l'héritage et de l'encapsulation
  - Utilisation du polymorphisme
- D'autres part les données persistantes sont souvent stockées dans des bases de données relationnelles.
- Le mapping Objet relationnel consiste à faire correspondre un enregistrement d'une table de la base de données à un objet d'une classe correspondante.
- Dans ce cas on parle d'une classe persistante.
- Une classe persistante est une classe dont l'état de ses objets sont stockés dans une unité de sauvegarde (Base de données, Fichier, etc..)

Architecture d'une application

- Une application se compose de plusieurs couches:
  - La couche DAO qui s'occupe de l'accès aux bases de données.
  - La couche métier qui s'occupe des traitements.
  - La couche présentation qui s'occupe de la saisie, le contrôle et l'affichage des résultats. Généralement la couche présentation respecte le pattern MVC qui fonctionne comme suit:

1. La vue permet de saisir les données, envoie ces données au contrôleur
2. Le contrôleur récupère les données saisies. Après la validation de ces données, il fait appel à la couche métier pour exécuter des traitements.
3. Le contrôleur stocke le résultat de le modèle.
4. Le contrôleur fait appel à la vue pour afficher les résultats.
5. La vue récupère les résultats à partir du modèle et les affiche.

ARCHITECTURE D'UNE APPLICATION

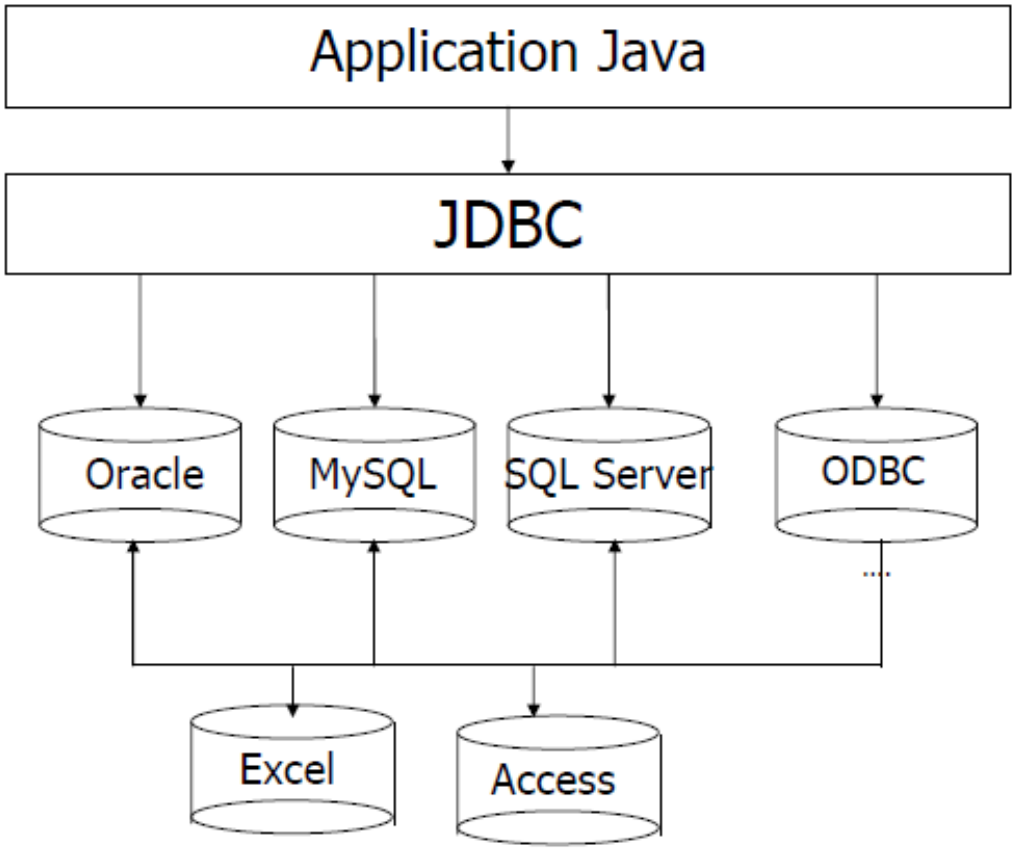


Accès aux bases de données via JDBC

PILOTES JDBC

- Pour qu'une application java puisse communiquer avec un serveur de bases de données, elle a besoin d'utiliser les pilotes JDBC (Java Data Base Connectivity)
- Les Pilotes JDBC est une bibliothèque de classes java qui permet, à une application java, de communiquer avec un SGBD via le réseau en utilisant le protocole TCP/IP
- Chaque SGBD possède ses propres pilotes JDBC.
- Il existe un pilote particulier « JdbcOdbcDriver » qui permet à une application java communiquer avec n'importe quelle source de données via les pilotes ODBC (Open Data Base Connectivity)
- Les pilotes ODBC permettent à une application Windows de communiquer une base de données quelconque (Access, Excel, MySQL, Oracle, SQL SERVER etc…)
- La bibliothèque JDBC a été conçu comme interface pour l'exécution de requêtes SQL.Une application JDBC est isolée des caractéristiques particulières du système de base de données utilisé.

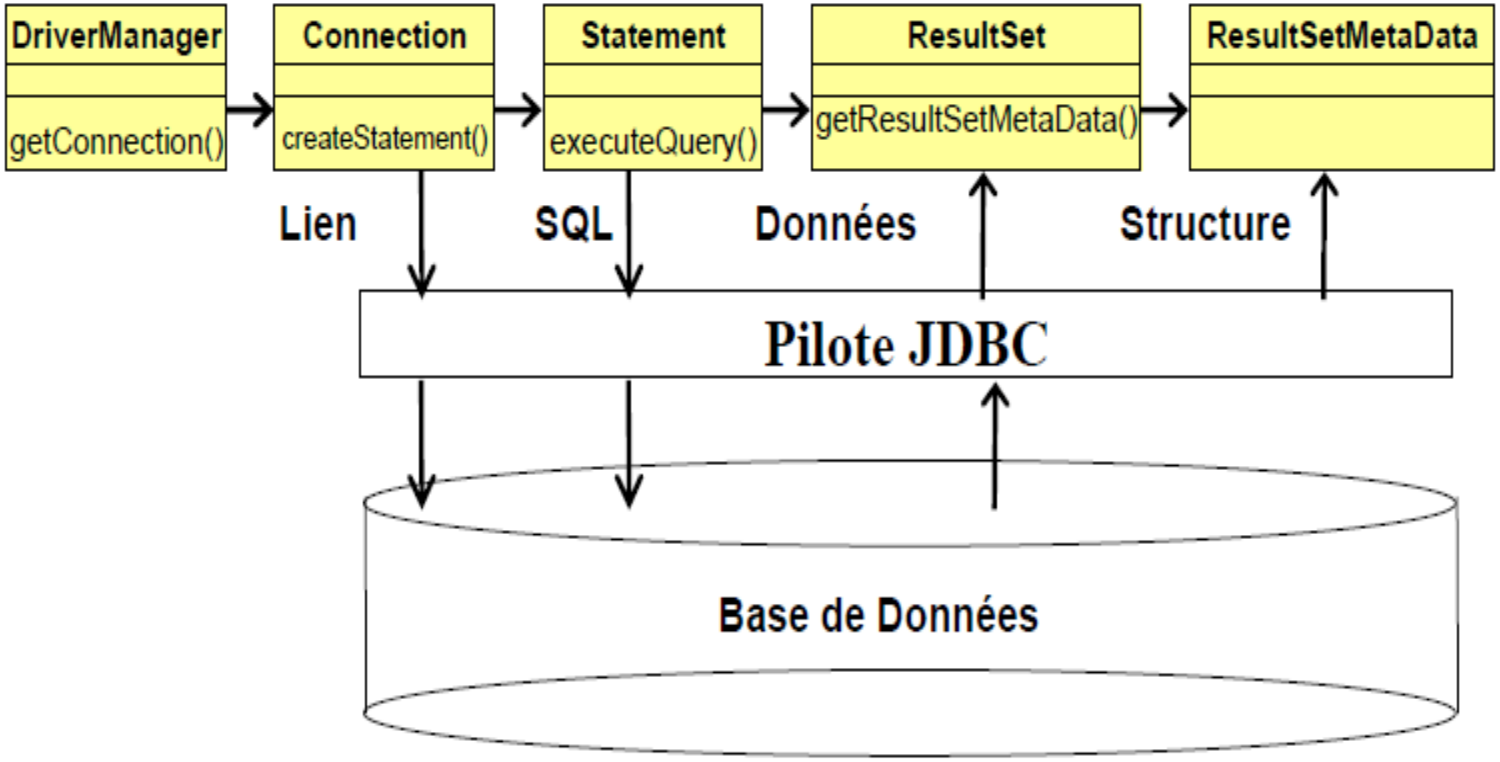
JAVA & JDBC


Application Java
JDBC
Oracle
MySQL
SQL Server
ODBC
....
Excel
Access

CRÉER UNE APPLICATION JDBC

- Pour créer une application élémentaire de manipulation d'une base de données il faut suivre les étapes suivantes :
  - Chargement du Pilote JDBC ;
  - Identification de la source de données ;
  - Allocation d'un objet ***Connection***
  - Allocation d'un objet Instruction ***Statement*** *ou* **PreparedStatement**
  - Exécution d'une requête à l'aide de l'objet Statement ;
  - Récupération de données à partir de l'objet renvoyé ***ResultSet ;***
  - Fermeture de l'objet ResultSet ;
  - Fermeture de l'objet Statement ;
  - Fermeture de l'objet Connection.

Créer une application JDBC



Démarche JDBC

- Charger les pilotes JDBC :
  - Utiliser la méthode forName de la classe Class, en précisant le nom de la classe pilote.
- Exemples:
  - Pour charger le pilote JdbcOdbcDriver:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver") ;
```

  - Pour charger le pilote jdbc de MySQL:

```
Class.forName("com.mysql.jdbc.Driver") ;
```

Créer une connexion

- Pour créer une connexion à une base de données, il faut utiliser la méthode statique getConnection() de la classe DriverManager. Cette méthode fait appel aux pilotes JDBC pour établir une connexion avec le SGBDR, en utilisant les sockets.
- Pour un pilote com.mysql.jdbc.Driver :

```
Connection conn = DriverManager.getConnection
("jdbc:mysql://localhost:3306/DB", "user", "pass" );
```

- Pour un pilote sun.jdbc.odbc.JdbcOdbcDriver :

```
Connectio conn= DriverManager.getConnection
("jdbc:odbc:dsnSCO", "user", "pass" );
```

OBJETS STATEMENT & RESULTSET

- Pour exécuter une requête SQL, on peut créer l'objet Statement en utilisant la méthode createStatement() de l'objet Connection.

```
Statement st=conn.createStatement();
```

- Pour exécuter une requête SQL de type select, on peut utiliser la méthode executeQuery() de l'objet Statement. Cette méthode exécute la requête et stocke le résultat de la requête dans l'objet ResultSet:

```
ResultSet rs=st.executeQuery("select * from produit");
```

- Pour exécuter une requête SQL de type insert, update et delete on peut utiliser la méthode executeUpdate() de l'objet Statement :

```
st.executeUpdate("insert into produit(…) values(…)");
```

OBJET PREPAREDSTATEMENT

- Pour exécuter une requête SQL, on peut également créer l'objet PreparedStatement en utilisant la méthode prepareStatement() de l'objet Connection.

```
PreparedStatement ps=conn.prepareStatement("select *
from PRODUITS where NOM_PROD like ? AND PRIX<?");
```

- Définir les valeurs des paramètres de la requête:

```
ps.setString(1,"%"+motCle+"%");
ps.setString(2, p);
```

- Pour exécuter une requête SQL de type select, on peut utiliser la méthode executeQuery() de l'objet Statement. Cette méthode exécute la requête et stocke le résultat de la requête dans l'objet ResultSet:

```
ResultSet rs=ps.executeQuery();
```

- Pour exécuter une requête SQL de type insert, update et delete on peut utiliser la méthode executeUpdate() de l'objet Statement :

```
ps.executeUpdate();
```

LES DONNÉES D'UN RESULTSET

- Pour parcourir un ResultSet, on utilise sa méthode next() qui permet de passer d'une ligne à l'autre. Si la ligne suivante existe, la méthode next() retourne true. Si non elle retourne false.
- Pour récupérer la valeur d'une colonne de la ligne courante du ResultSet, on peut utiliser les méthodes getInt(colonne), getString(colonne), getFloat(colonne), getDouble(colonne), getDate(colonne), etc… colonne représente le numéro ou le nom de la colonne de la ligne courante.
- **Syntaxte:**

```
while(rs.next()){
    System.out.ptintln(rs.getInt(1));
    System.out.ptintln(rs.getString("NOM_PROD"));
    System.out.ptintln(rs.getDouble("PRIX"));
}
```

LE FRAMEWORK HIBERNATE

INTRODUCTION

- Travailler dans les deux univers que sont l'orienté objet et la base de données relationnelle peut être lourd et consommateur en temps dans le monde de l'entreprise d'aujourd'hui.
- Hibernate est un outil de mapping objet/relationnel pour le monde Java.
- Le terme mapping objet/relationnel (ORM) décrit la technique consistant à faire le lien entre la représentation objet des données et sa représentation relationnelle basée sur un schéma SQL.

HIBERNATE?

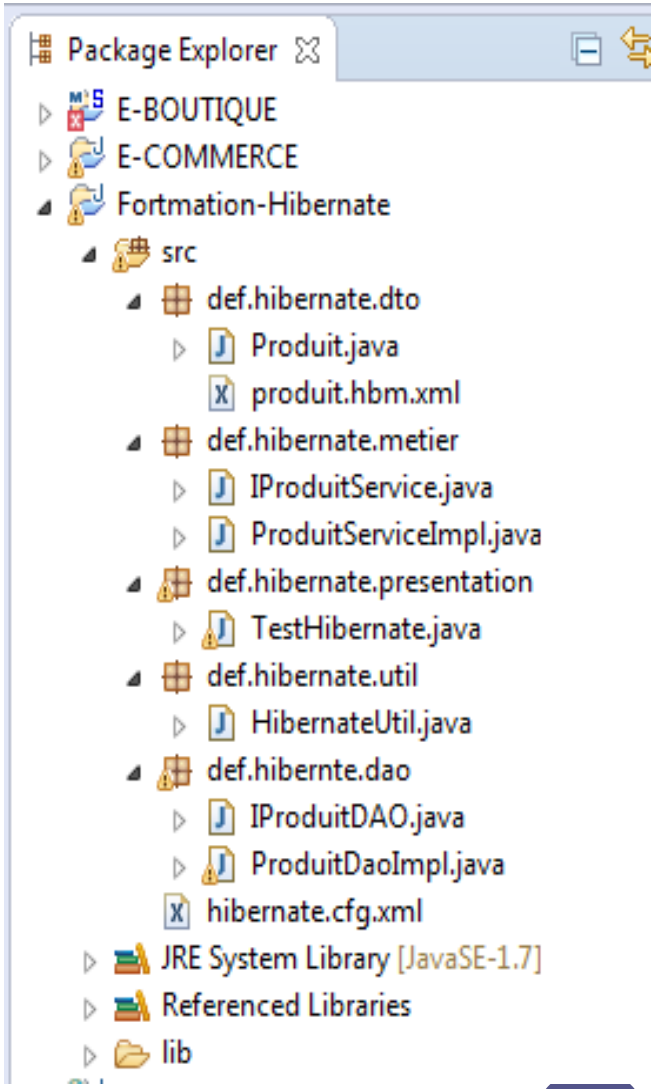
- Hibernate s'occupe du transfert des objets Java dans les tables de la base de données
- En plus, il permet de requêter les données et propose des moyens de les récupérer.
- Il peut donc réduire de manière significative le temps de développement qui aurait été autrement perdu dans une manipulation manuelle des données via SQL et JDBC
- Le but d'Hibernate est de libérer le développeur de 95% des tâches de programmation liées à la persistance des données communes.
- Hibernate assure la portabilité de votre application si vous changer de SGBD.
- Hibernate propose au développeur des méthodes d'accès aux bases de données plus efficace ce qui devrait rassurer les développeurs.

ARCHITECTURE HIBERNTE

- Hibernate permet d'assurer la persistance des objets de l'application dans un entrepôt de données.
- Cet entrepôt de données est dans la majorité des cas une base de données relationnelle, mais il peut être un fichier XML.
- Le mapping des objets est effectuée par Hibernate en se basant sur des fichiers de configuration en format texte ou souvent XML.

Arborescence du projet

- Package def.hibernate.dto
  - Produit.java
  - Produit.hbm.xml (fichier de mapping xml)
- Package def.hibernate.presentation
  - Classe principal main de test
- Package def.hibernate.metier
  - IProduitService (interface)
  - ProduitServiceImpl (implémentation)
- Package def.hibernate.dao
  - IProduitDAO
  - ProduitDAOImpl
- Package def.hibernate.utils
  - HibernateUtil



Le fichier de configuration Hibernate hibernate.cfg.xml

Mapper la classe Produit

- Chaque classe persistante doivent être mapper à l'aide d'un fichier hbm.xml.
- Créer un fichier Produit.hbm.xml pour mapper la classe Produit.java
- Le fichier doit définir le nom de la table ainsi que les champs correspondants.

Produit.hbm.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
3          "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
4  <hibernate-mapping package="def.dto">
5      <class name="Produit" table="produit">
6          <id name="idProduit" column="ID_PRODUIT">
7          <generator class="native"></generator>
8          </id>
9          <property name="nomProduit" column="NOM_PRODUIT"></property>
10         <property name="prix" column="PRIX"></property>
11         <property name="quantite" column="QUANTITE"></property>
12     </class>
13
14 </hibernate-mapping>
```

FICHIER DE CONFIGURATION

- Création du fichier hibernate.cfg.xml sous la racine du projet.
- Configuration des données liées à la base:driver, url, le login et le mot de passe.
- Définir l'ensemble des fichiers de Mapping hbm.xml

hibernate.cfg.xml

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <!DOCTYPE hibernate-configuration PUBLIC
3            "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4            "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5  <hibernate-configuration>
6    <session-factory>
7      <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
8      <property name="connection.url">jdbc:mysql://localhost/e-commerce</property>
9      <property name="connection.username">root</property>
10     <property name="connection.password"></property>
11
12    <property name="transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</property>
13     <property name="current_session_context_class">thread</property>
14
15     <property name="connection.pool_size">1</property>
16     <property name="show_sql">true</property>
17
18     <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
19     <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
20     <mapping resource="def/hibernate/dto/produit.hbm.xml" />
21     <!--  <mapping class="def.hibernate.dto.Produit"/>-->
22
23   </session-factory>
24
25 </hibernate-configuration>
26
```

Classe d'accès aux données DAO (1)

- Définir les méthodes à implémenter dans une interface: IProduitDAO (pas d'implémentation)
  - Méthode d'ajout insertProduit (objet chargé en paramètre d'entré)
  - Méthode de modification updateProduit
  - Méthode de Suppression deleteProduit
  - Méthode de recherche exemple: listProduitByMC (type de retour liste de produit)

IProduitDAO.java

```
1
2  package def.dao;
3
4  import java.util.List;
7
8  public interface IProduitDAO {
9
10     public void insertProduit (Produit produit);
11     public void updateProduit (Produit produit);
12     public void deleteProduit (int idProduit);
13     public Produit getProduitById (int idProduit);
14     public List<Produit> listProduitByMC (String mc);
15
16 }
```

Classe d'accès aux données DAO (2)

- Implémenter la méthode d'insertion d'un produit:
  - Ouvrir une session de sessionFactory d'Hibernate
  - Commencer une transaction
  - Enregistrer l'objet produit avec la méthode save d'hibernate
  - Commiter la transaction
  - Fermer la session d'hibernate.

```
@Override
public void insertProduit(Produit produit) {
    // TODO Auto-generated method stub
    Session session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();
    session.saveOrUpdate(produit);
    tx.commit();
    session.close();
}
```

LE FRAMEWORK SPRING MVC

Framework Spring

- Spring est un framework open source JEE pour les applications ntiers, dont il facilite le développement et les tests.
- Il est considéré comme un conteneur dit « léger », c'est-à-dire une infrastructure similaire à un serveur d'application J2EE.
- Il prend donc en charge la création d'objets et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration qui décrit les objets à fabriquer et les relations de dépendances entre ces objets.
- Il permet de séparer le code métier du code technique Spring s'appuie principalement sur l'intégration de trois concepts clés:
  - l'inversion de contrôle ou injection de dépendance (IoC).
  - la programmation orientée aspect (AOP).
  - une couche d'abstraction.
- Ce framework, grâce à sa couche d'abstraction, ne concurrence pas d'autres frameworks dans une couche spécifique d'un modèle architectural MVC mais s'avère un framework multi-couches pouvant s'insérer au niveau de toutes les couches.

Injection des dépendances avec Spring (1)

- L'injection des dépendance, ou l'inversion de contrôle est un concept qui intervient généralement au début de l'exécution de l'application.
- Spring IOC commence par lire un fichier XML qui déclare quelles sont différentes classes à instancier et d'assurer les dépendances entre les différentes instances.
- Quand on a besoin d'intégrer une nouvelle implémentation à une application, il suffirait de la déclarer dans le fichier xml de beans spring.

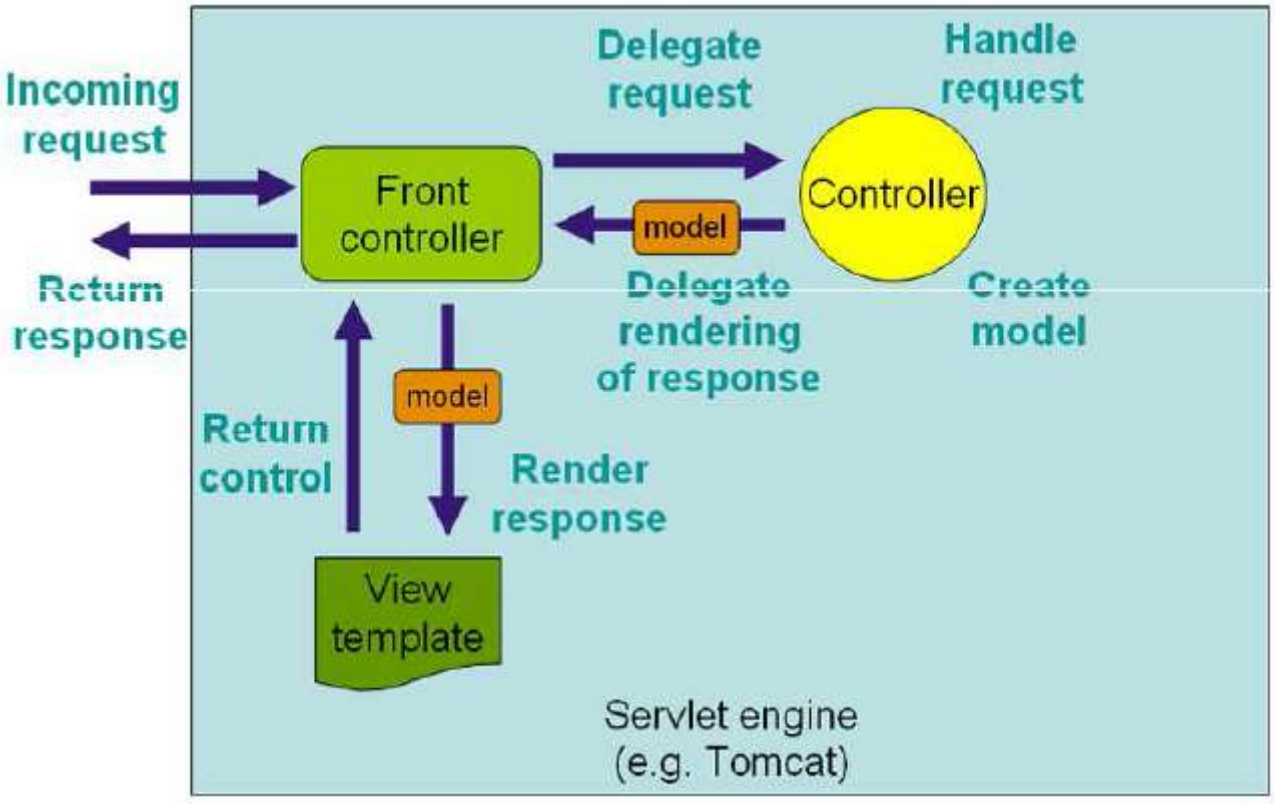
Injection des dépendances avec Spring (2)

- Dans une application web, SpringIOC est appelé démarrage du serveur en déclarant le listener ContextLoaderListener dans le fichier web.xml

```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-beans.xml</param-value>
</context-param>
<listener>
        <listener-class>
   org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>
```

- Dans cette déclaration, CotextLoaderListener est appelé par Tomcat au moment du démarrage de l'application. Ce listener cherchera le fichier de beans spring « spring-beans.xml » stocké dans le dossier WEB-INF. ce qui permet de faire l'injection des dépendances entre MetierImpl et DaoImpl

Architecture Spring MVC



SPRING MVC (1)

1. le client fait une demande au contrôleur. Celui-ci voit passer toutes les demandes des clients. C'est la porte d'entrée de l'application. C'est le C de MVC. Ici le contrôleur est assuré par une servlet générique :

   org.springframework.web.servlet.DispatcherServlet
2. le contrôleur principal [DispatcherServlet] fait exécuter l'action demandée par l'utilisateur par une classe implémentant l'interface : org.springframework.web.servlet.mvc.Controller
   - A cause du nom de l'interface, nous appellerons une telle classe un contrôleur secondaire pour le distinguer du contrôleur principal [DispatcherServlet] ou simplement contrôleur lorsqu'il n'y a pas d'ambiguïté.
3. le contrôleur [Controller] traite une demande particulière de l'utilisateur. Pour ce faire, il peut avoir besoin de l'aide de la couche métier. Une fois la demande du client traitée, celle-ci peut appeler diverses réponses. Un exemple classique est :
   - une page d'erreurs si la demande n'a pu être traitée correctement
   - une page de confirmation sinon

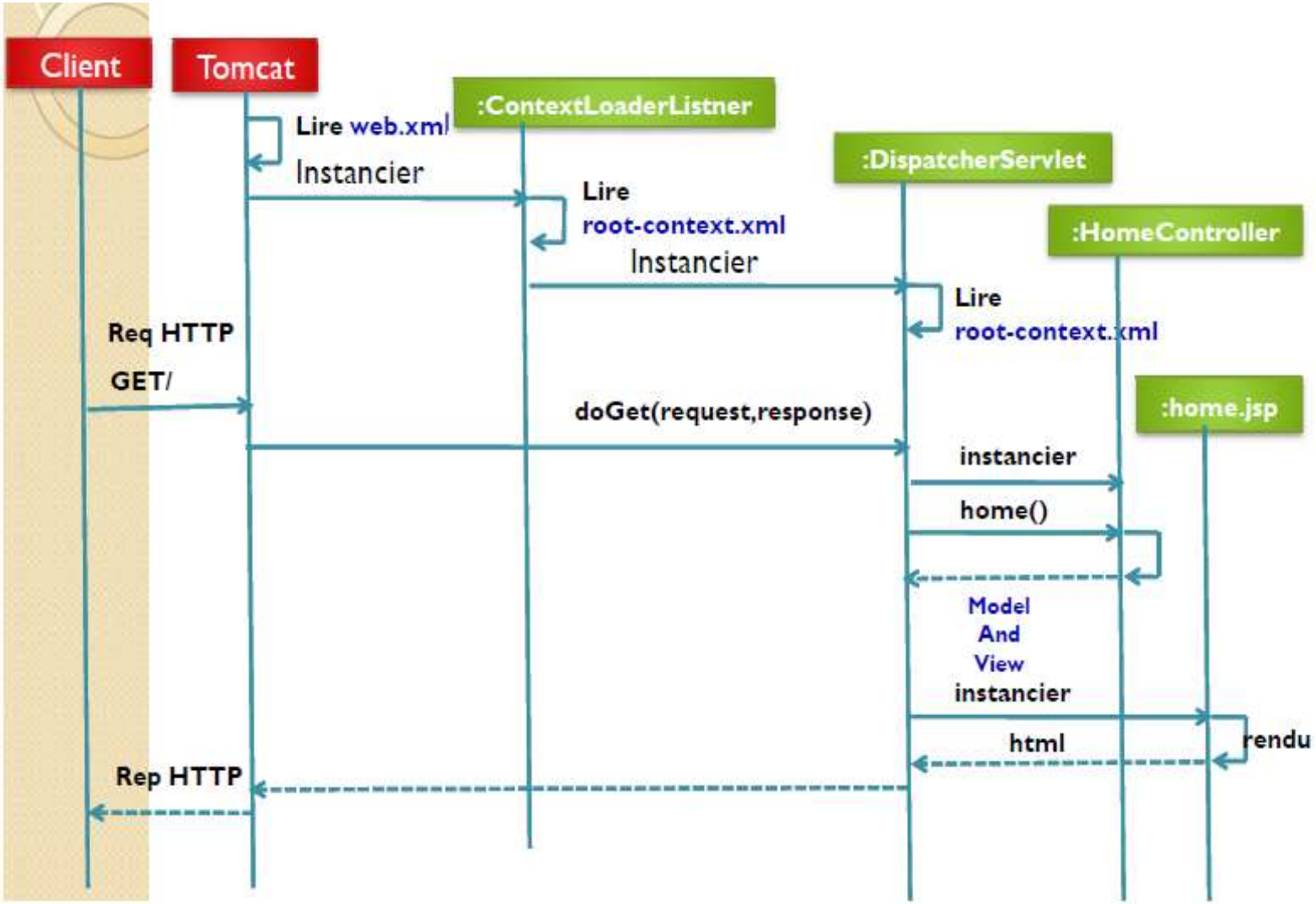
SPRING MVC (2)

4. Le contrôleur choisit la réponse (= vue) à envoyer au client. Choisir la réponse à envoyer au client nécessite plusieurs étapes :
   - choisir l'objet qui va générer la réponse. C'est ce qu'on appelle la vue **V, le V de** MVC. Ce choix dépend en général du résultat de l'exécution de l'action demandée par l'utilisateur.
   - lui fournir les données dont il a besoin pour générer cette réponse. En effet, celle-ci contient le plus souvent des informations calculées par la couche métier ou le contrôleur lui-même. Ces informations forment ce qu'on appelle le modèle **M de la** vue, le **M de MVC.**
   - Cette étape consiste donc en le choix d'une vue **V et la construction du modèle M** nécessaire à celle-ci.

SPRING MVC (3)

5. Le contrôleur **DispatcherServlet demande à la vue choisie de s'afficher.** Il s'agit d'une classe implémentant l'interface **org.springframework.web.servlet.View**
   - Spring MVC propose différentes implémentations de cette interface pour générer des flux HTML, Excel, PDF, ...
6. le générateur de vue View utilise le modèle Map préparé par le contrôleur Controller pour initialiser les parties dynamiques de la réponse qu'il doit envoyer au client.
7. la réponse est envoyée au client. La forme exacte de celle-ci dépend du générateur de vue. Ce peut être un flux HTML, XML, PDF, Excel, ...

FONCTIONNEMENT



LE FICHIER ROOT-CONTEXT.XML

- Ce fichier est lu par ContextLoaderListener, au démarrage du serveur .
- C'est un fichier dans lequel contexte de l'application sera construit ContextLoaderListener représente Spring IOC
- C'est donc un fichier pour l'injection des dépendances
- Pour le moment, il est vide

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/sc
  hema/beans
  http://www.springframework.org/schema/beans/spring-
  beans.xsd">

<!-- Root Context: defines shared resources visible
  to all other web components -->

</beans>
```