



Frameworks Spring & Hibernate

- Introduction au choix et à l'utilisation d'un framework
- Présentation de Spring et de ses modules
- Les Design Patterns de Spring
- Hibernate : un outil de mapping objet-relationnel
- Combiner Hibernate & Spring

- **Introduction au choix et à l'utilisation d'un framework**
- Présentation de Spring et de ses modules
- Les Design Patterns de Spring
- Hibernate : un outil de mapping objet-relationnel
- Combiner Hibernate & Spring

Introduction aux frameworks

- Framework : ensemble de composants qui **structure** une application et **contraint** la manière de développer
 - Lié à un langage de programmation
 - Plutôt générique (contrairement à une librairie)

Introduction aux frameworks

Avantages d'un framework :

- Gain de temps et d'efficacité
- Meilleure structuration du code
- Maintenance / Evolution simplifiée

Introduction aux frameworks

Inconvénients d'un framework :

- Limite les fonctionnalités complexes du langage
- Niveau d'abstraction supplémentaire
- Nouvelle technologie à supporter / intégrer / maintenir / ...

Introduction aux frameworks

Conclusion :

- Framework == Abstraction de fonctionnalités et d'architecture pour un langage au détriment de certaines possibilités

Introduction aux frameworks

Comment choisir un framework ?

- Définir le cadre du projet (besoins, fonctionnalités, ...)
- Ratio avantages/inconvénients technologiques
- ***Quel impact au niveau projet ?***

Introduction aux frameworks

Exemple de métriques techniques :

- + Facilité d'utilisation
- + Fonctionnalités déjà intégrées
- + Besoins client “standards”
- Utilisation de fonctionnalités avancées du langage de programmation
- Impact sur l'architecture / difficulté à changer de framework

Introduction aux frameworks

Exemple de métriques projet :

- + Stabilité
- + Maturité (techno et/ou paradigme)
- + Communauté
- Maintenance / évolutions du framework non garanties
- Difficulté de formation

- Introduction au choix et à l'utilisation d'un framework
- **Présentation de Spring et de ses modules**
- Les Design Patterns de Spring
- Hibernate : un outil de mapping objet-relationnel
- Combiner Hibernate & Spring

Présentation de Spring

“Fundamentally, what is Spring? We think of it as a Platform for your Java code” (docs.spring.io)

- Spring est un framework Java open-source composé de nombreux modules offrant des fonctionnalités haut niveau pour une application classique.
- Le design pattern d'injection de dépendances utilisé dans toutes les couches du framework permet d'ajouter des services à des POJOs de manière non invasive

Présentation de Spring

Les modules de Spring :

- Spring core
- Spring context
- Spring dao
- Spring orm
- Spring web
- Spring web mvc

Présentation de Spring

Spring core : le noyau, qui contient à la fois

- un ensemble de classes utilisées par toutes les briques du framework
- le conteneur léger
`org.springframework.context.ApplicationContext`

Présentation de Spring

Spring Context : ce module supporte

- l'internationalisation (I18N)
- Enterprise Java Beans (EJB)
- Java Message Service (JMS)
- Basic Remoting

Présentation de Spring

Spring DAO : constitue le socle de l'accès aux dépôts de données

- Fournit une implémentation pour JDBC
- D'autres modules fournissent des abstractions pour l'accès aux données (solutions de mapping objet-relationnel, LDAP) qui suivent les mêmes principes que le support JDBC
- La solution de gestion des transactions de Spring fait aussi partie de ce module

Présentation de Spring

Spring ORM : propose une intégration avec des outils populaires de mapping objet-relationnel

- Hibernate
- JPA
- EclipseLink
- iBatis
- ...

Présentation de Spring

Spring WEB : le module comprenant le support de Spring pour les applications Web

- contient notamment Spring Web MVC, la solution de Spring pour les applications Web
- propose une intégration avec de nombreux frameworks Web
- propose une intégration avec des technologies de vue

Présentation de Spring

Spring Web MVC : implémentation Model-
View-Controller (MVC) pour applications
Spring Web

- Introduction au choix et à l'utilisation d'un framework
- Présentation de Spring et de ses modules
- **Les Design Patterns de Spring**
- Hibernate : un outil de mapping objet-relationnel
- Combiner Hibernate & Spring

Spring design patterns

Rappel : la programmation orientée objet est basée sur les principes **SOLID**

- Single Responsibility
 - Open / Closed
 - Liskov Substitution
 - Interface segregation
 - Dependency injection
-
- Spring utilise massivement les principes de **Single Responsibility** et **Dependency Injection**

Spring design patterns

Open / Closed

Ouvert à l'extension, fermé à la modification

Liskov Substitution

L'héritage ne doit pas changer le comportement

Interface Segregation

Interfaces minimalistes

Spring design patterns

Single Responsibility :

1 classe ou 1 méthode => 1 responsabilité

- Spring utilise des interfaces simples ayant peu de fonctionnalités
- Le framework permet de séparer facilement les responsabilités au sein de différentes classes

Spring design patterns

- ***Dependency injection :***

Les liens de dépendances sont résolus dynamiquement en injectant les dépendances dans les classes à l'exécution (et non pas statiquement à la compilation)

- Spring utilise massivement l'injection pour fournir une implémentation des interfaces lorsque nécessaire

Spring design patterns

Le framework Spring utilise principalement les design pattern suivants :

- ***Inverse de contrôle***
- ***Singleton***
- ***Programmation par template***
- ***Modèle MVC***

Spring design patterns

Inversion de contrôle : le flot d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même mais du framework ou de la couche logicielle sous-jacente. (Wikipédia)

- Spring implémente une inversion de contrôle par le principe d'***injection de dépendances***

Spring design patterns

Exemple d'injection de dépendance :
maDépendance est instanciée par Spring

```
@Component
public class MaDépendance {
    [...]
}

@Component
public class MaClasseAvecDépendance {

    @Autowired
    private MaDépendance maDépendance;

}
```

Spring design patterns

Singleton : design pattern dont l'objectif est de restreindre l'instanciation d'une classe à un seul objet (Wikipédia)

- Exemple : Les “enum” en Java sont des singletons
- En Spring, par défaut ***chaque interface est implémentée par un singleton***

Spring design patterns

Singleton
- <u>singleton : Singleton</u>
- Singleton() + <u>getInstance() : Singleton</u>

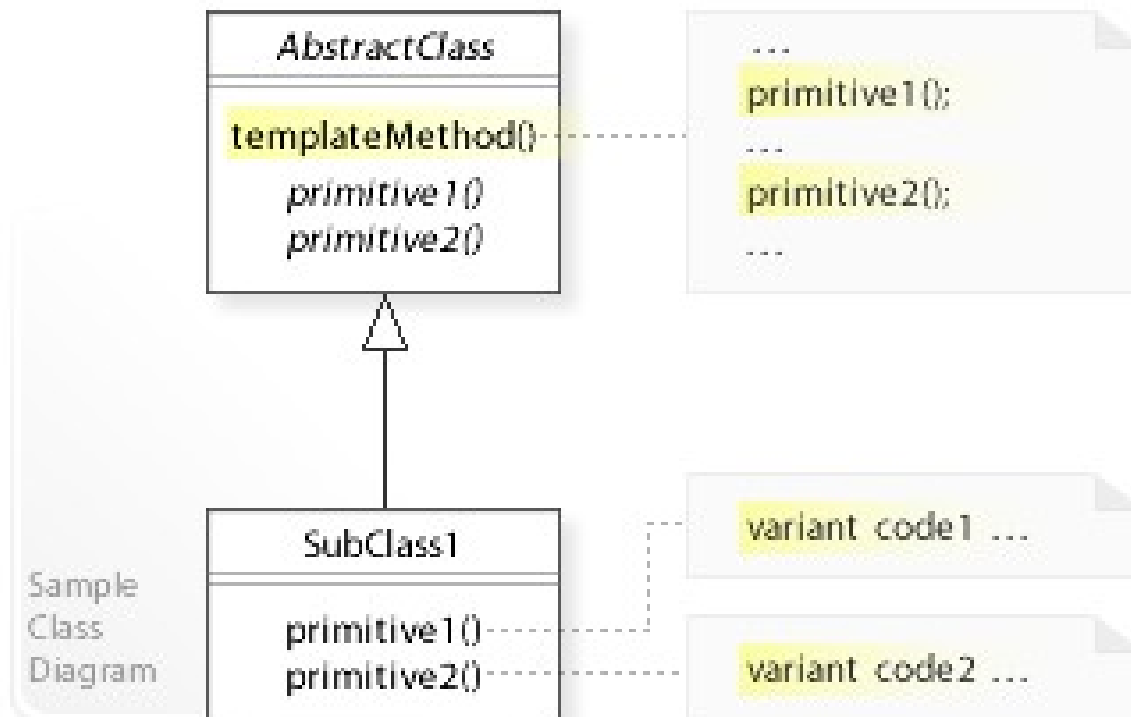
Singleton pattern (Wikipedia)

Spring design patterns

Programmation par template : en POO, design pattern qui laisse libre l'implémentation d'une partie de l'algorithme à l'héritage, sans changer la structure de cet algorithme

- Exemple : Les classes abstraites en Java
- ***Attention*** : la programmation par template ne doit pas pour autant invalider le principe de Liskov !
 - ***L'implémentation ne doit pas changer le comportement de la classe parente !***

Spring design patterns



Template method pattern (Wikipedia)

Spring design patterns

Modèle/View/Contrôleur (MVC) : Design pattern populaire dans les applications web, permettant :

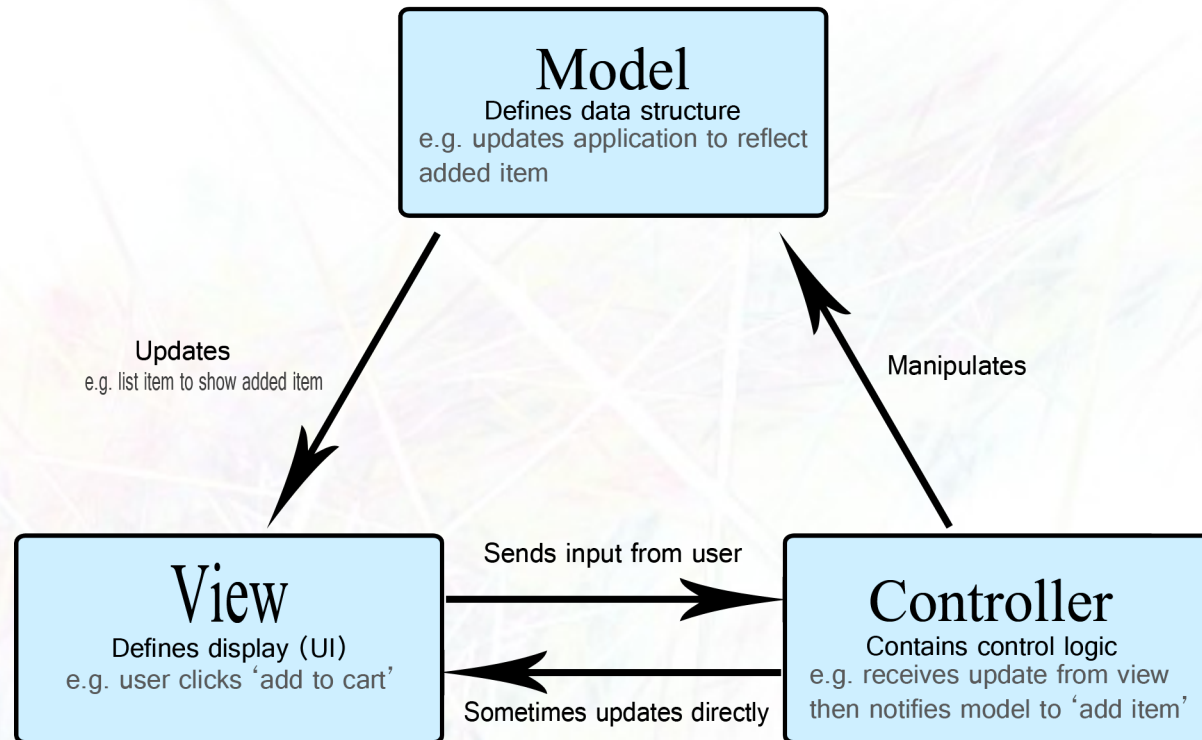
- De séparer la présentation des données de leur modélisation et de leur manipulation
- La mise à jour de la couche de représentation lorsque les données changent

Spring design patterns

MVC est composé de 3 modules

- **Modèle** : contient les données ainsi que de la logique en rapport avec les données (validation, ...)
- **Vue** : partie visible d'une interface graphique. Une vue contient des éléments visuels ainsi que la logique nécessaire pour afficher les données provenant du modèle
- **Contrôleur** : module qui traite les actions de l'utilisateur, modifie les données du modèle et de la vue.

Spring design patterns



The MVC pattern (developer.mozilla.org)

Spring design patterns

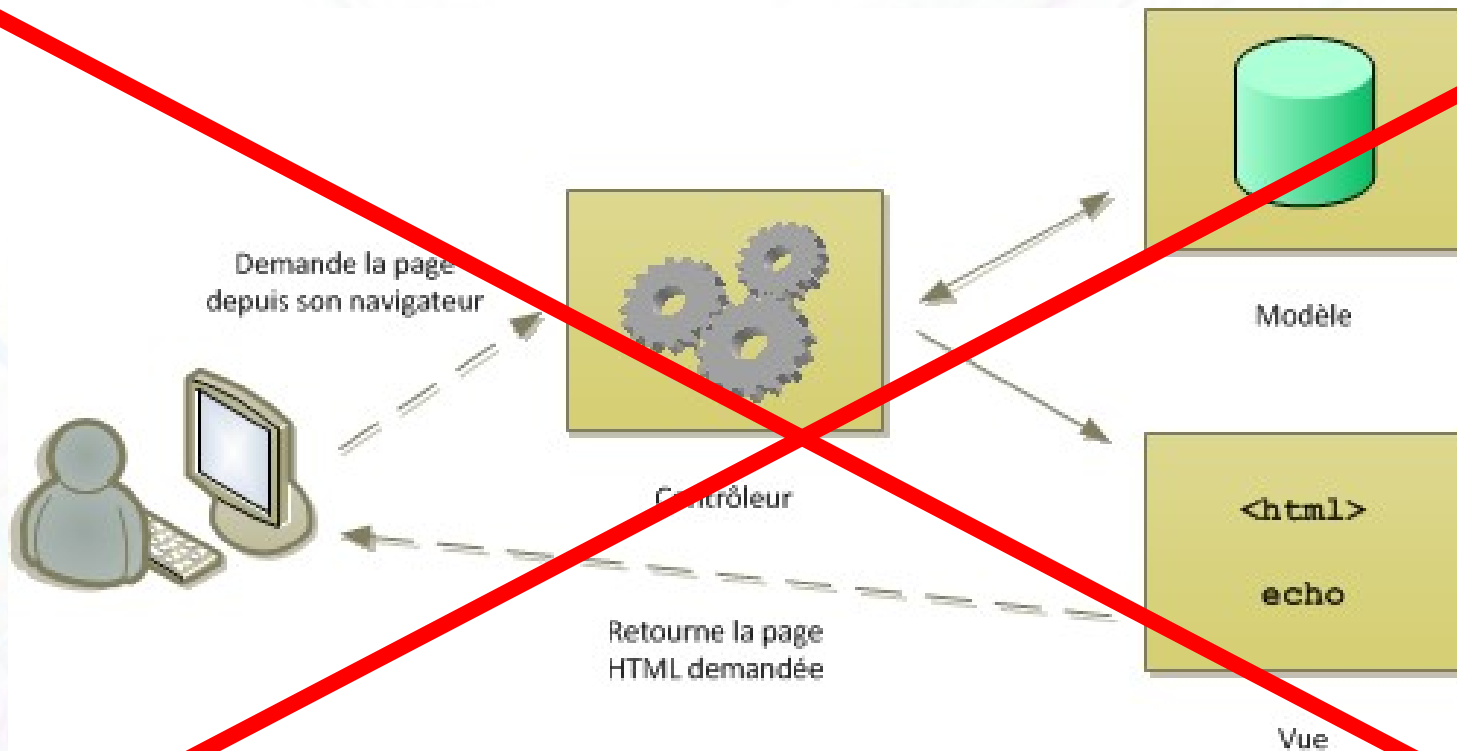
“Architecture of an application is all about its intent.” (Robert Martin)

!!! MVC N'EST PAS UNE ARCHITECTURE !!!

MVC est un mécanisme de livraison de données (par exemple, par le Web) : l'architecture et la logique métier de l'application sont décorélés de ce pattern

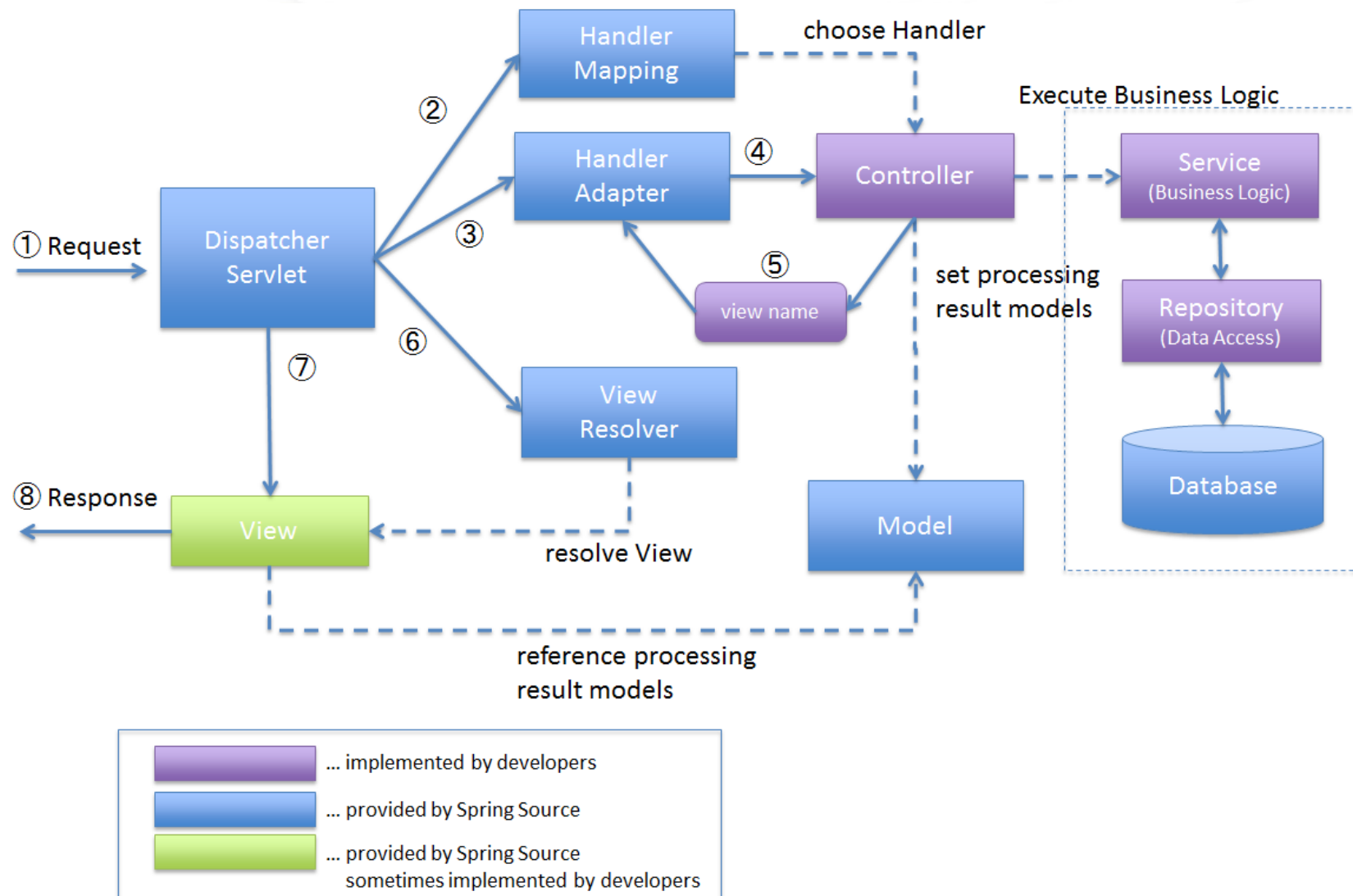
i.e. les exemples (nombreux) d'application MVC avec persistance directe de la couche “Model” en base de données sans logique métier sont des anti-patterns !!!

Spring design patterns



Ceci n'est PAS le pattern MVC !!!

Spring design patterns



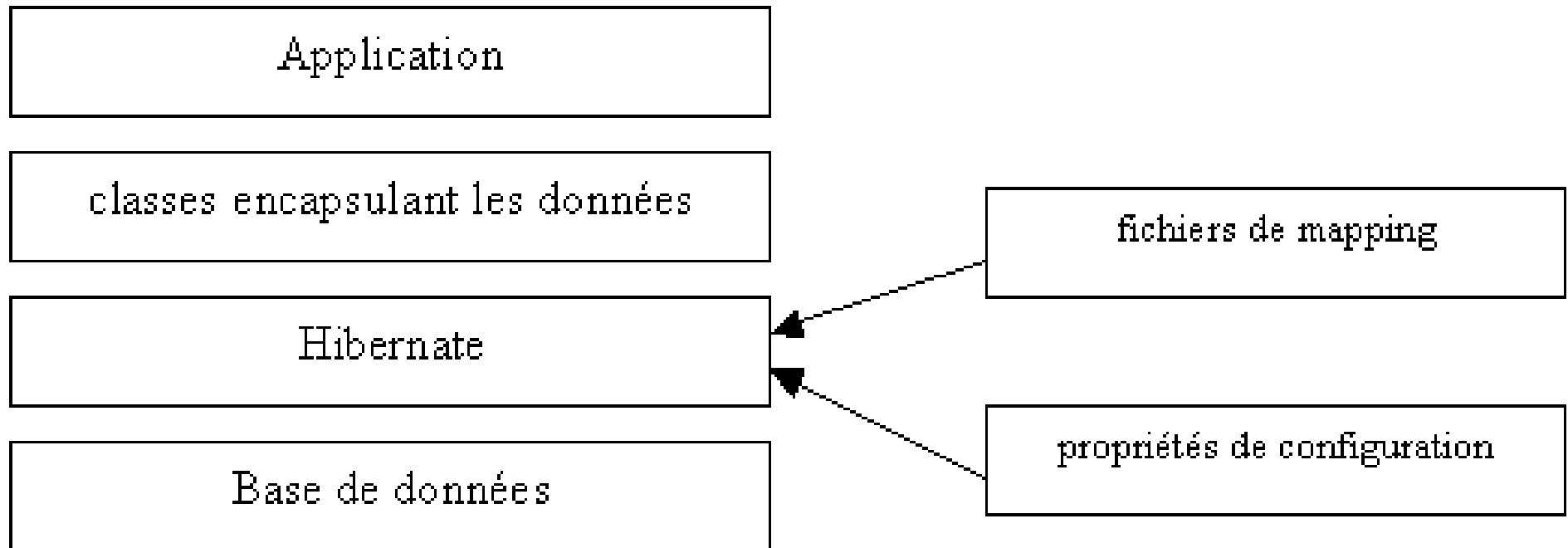
*Une implémentation MVC : Spring MVC Request Lifecycle
(terasolunaorg.github.io)*

- Introduction au choix et à l'utilisation d'un framework
- Présentation de Spring et de ses modules
- Les Design Patterns de Spring
- **Hibernate : un outil de mapping objet-relationnel**
- Combiner Hibernate & Spring

Hibernate

- Hibernate est une solution open source de type ORM (Object Relational Mapping) qui permet de faciliter le développement de la couche persistance d'une application.
- Hibernate permet donc de **représenter une base de données en objets Java et vice versa**.
 - Ceci afin d'abstraire l'implémentation de la base de données du code
 - La plupart des BD sont supportées
- Note d'architecture : *"The database is a detail"* (Robert Martin)

Hibernate - architecture

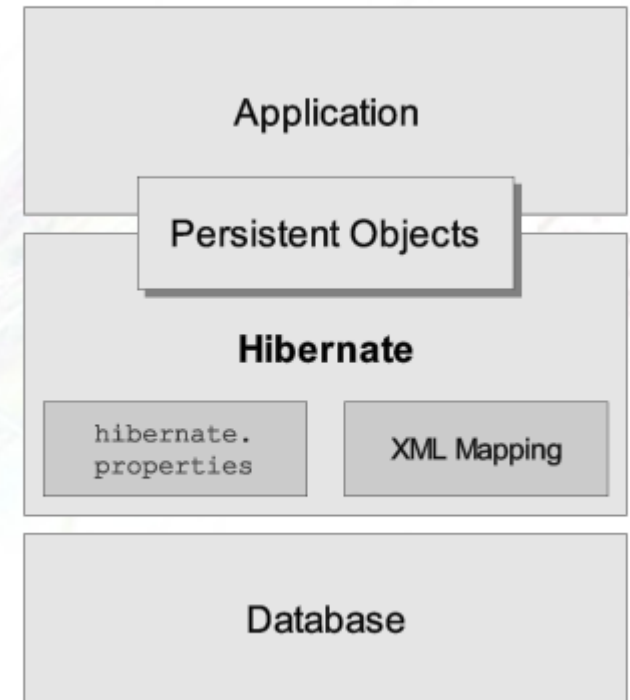


Architecture haut niveau d'une application utilisant Hibernate

Hibernate - architecture

Persistent Objects :

- Objets mono-threadés à vie courte
- Contenant :
 - Un état persistant
 - Du code métier
- Objets ordinaires (JavaBean, POJO)
- Associés avec une (et une seule) Session.
- Dès que la Session est fermée, ils sont détachés et libres d'être utilisés par n'importe quelle couche de l'application (par ex. de et vers la présentation).



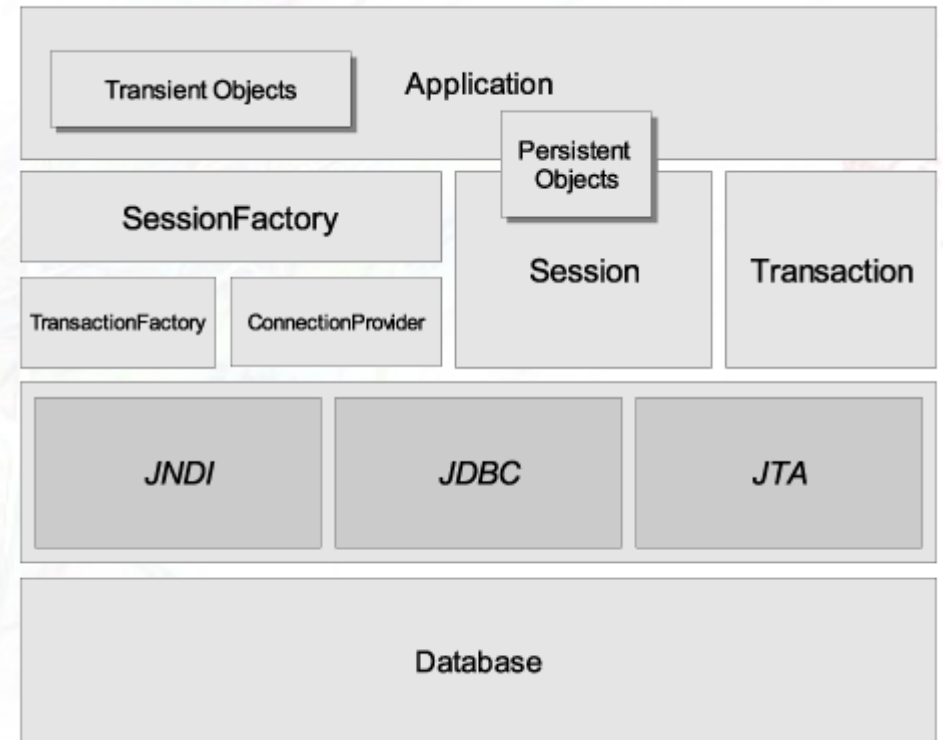
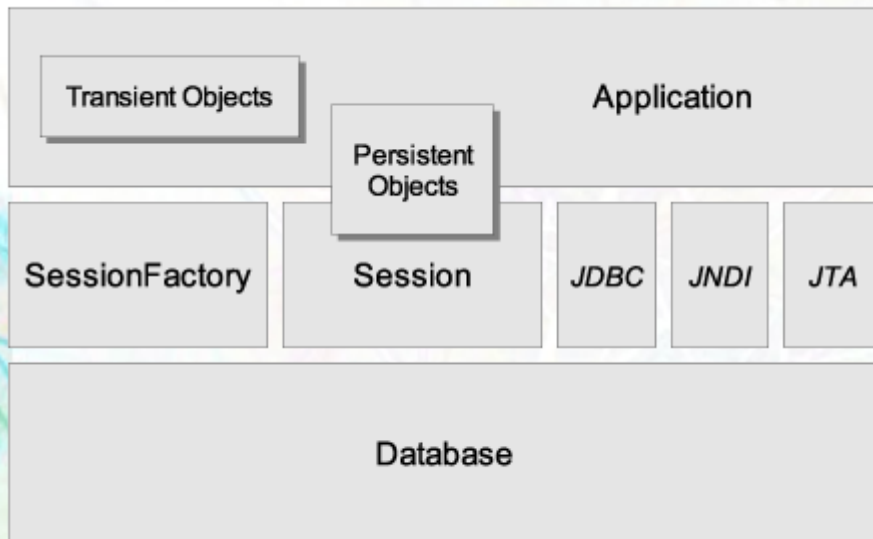
Architecture Haut Niveau

Hibernate - architecture

Exemple de POJO :

```
public class MyEntity {  
  
    private Integer myId;  
    private String myString;  
  
    public Integer getMyId() {  
        return myId;  
    }  
  
    public String getMyString() {  
        return myString;  
    }  
}
```

Hibernate - architecture



*Exemples d'architectures Hibernate :
Architectures "légères" (gauche) et "complètes" (droite)*

Hibernate - architecture

- **Session :**
 - objet mono-threadé, à durée de vie courte
 - représente une conversation entre l'application et l'entrepôt de persistance
 - encapsule une connexion JDBC
 - fabrique des objets Transaction
 - la Session contient un cache (de premier niveau) des objets persistants, qui sont utilisés lors de la navigation dans le graphe d'objets ou lors de la récupération d'objets par leur identifiant.

Hibernate - architecture

- **Unité de travail :**
 - **Séquences** de requêtes à la BD pour effectuer une opération atomique dans l'application
 - => 1 session par opération métier, et pas 1 session par requête BD !
 - En web, souvent 1 session par requête
`SessionFactory.getCurrentSession()`
 - Hibernate supporte aussi d'autres patterns beaucoup plus complexes
 - Toute opération (lecture et/ou modification) doit avoir lieu dans une **transaction**

Hibernate - architecture

- ***SessionFactory*** :
 - Responsable du cycle de vie des sessions
 - Objet complexe, coûteux et thread-safe
 - Prévu pour n'être instancié qu'une seule fois via une instance Configuration en général au démarrage de l'application.
- Au contraire, la ***Session*** :
 - N'est pas coûteuse, non-threadsafe
 - Ne devrait être utilisé qu'une seule fois pour une requête unique, une conversation, une unité de travail unique et devrait être relâché ensuite.

Hibernate – résumé architecture

La SessionFactory génère des Session

- Par exemple :
 SessionFactory.getCurrentSession()
- Les Session modélisent une utilisation atomique de la base de données
 - Par exemple : recherche d'un produit et mise à jour d'un de ses champs
- Les Session génèrent des Transaction (obligatoires pour tout dialogue avec la BD)
 - Session.beginTransaction()
 - Transaction.commit(), Transaction.rollback()

Hibernate - configuration

- Hibernate supporte de nombreux modes de configuration (programmatische, fichier `hibernate.properties`, fichier `hibernate.cfg.xml`, ...)
- Il est recommandé de :
 - Configurer les propriétés générales (JDBC, ...) dans un fichier `hibernate.properties` ou `hibernate.cfg.xml`
 - Déclarer les classes d'objets à persister dans le fichier `hibernate.cfg.xml`
 - Initialiser hibernate et ses fichiers de configuration lors de la génération de la `SessionFactory`

Hibernate - configuration

hibernate.properties : principales propriétés JDBC

- hibernate.connection.driver_class => driver JDBC
- hibernate.connection.url => URL JDBC
- hibernate.connection.username
- hibernate.connection.password
- hibernate.connection.pool_size => nombre maximum de connexions dans un pool

Hibernate - configuration

Initialisation d'Hibernate et génération d'une SessionFactory classique :

```
new Configuration().configure().buildSessionFactory()
```

Hibernate fournit également de nombreuses possibilités d'intégration dans un serveur J2EE :
réutilisation des connexions JDBC (par JNDI),
binding JNDI et Java Transaction API (JTA)
automatique, JMX déploiement, ...

Hibernate – classes de persistance

- Les classes persistées par Hibernate sont des entités décrites par des POJO (Plain Old Java Object) :
 - Constructeur public sans paramètre
 - Les attributs sont de type simple (int, float, String, Date, ...)
 - Chaque attribut doit avoir un getter et un setter
 - Penser à redéfinir les méthodes equals() et hashCode() pour éviter les mauvaises surprises
 - Les POJO peuvent contenir des méthodes métier non utilisées par hibernate

Hibernate - ORM

Bases du Mapping Objet-Relationel (ORM) :

- Les entités persistées sont en principe enregistrées en BD selon un schéma très proche :
 - ***1 table par entité***
 - ***1 colonne par attribut***
- Les relations entre classe/table et attribut/colonne sont définies soit dans un fichier XML dédié `nomDeLaClasse.hbm.xml` dans le même package, soit par annotation dans la classe Java

Hibernate - ORM

POJO à persister :

```
public class MyEntity {  
  
    private Integer myId;  
    private String myString;  
  
    public Integer getMyId() {  
        return myId;  
    }  
  
    public String getMyString() {  
        return myString;  
    }  
}
```


Hibernate - ORM

Exemple de mapping MyEntity.hbm.xml

```
<hibernate-mapping>
    <class name="MyEntity" table="MY_ENTITY">
        <id name="id">
            <generator class="native"/>
        </id>
        <property name="myString" type="string"
not-null="true" />
    </class>
</hibernate-mapping>
```


Hibernate - ORM

Exemple d'annotation Java :

```
@Entity // il s'agit d'une classe à persister
@Table(name = "MY_ENTITY") // la table à utiliser en BD pour cette classe
public class MyEntity {

    private Integer myId;

    private String myString;

    @Id // @Id indique une clé primaire
    public Integer getMyId() {
        return myId;
    }

    @Basic(optional = false) // @Basic désigne un type simple (int, float,
    String, ...) automatiquement mappé en BD pour une colonne par Hibernate.
    public String getMyString() {
        return myString;
    }
}
```

Hibernate - ORM

Utilisation d'Hibernate ORM :

- L'ORM permet de directement sauver / éditer / supprimer / lister les entités persistées sans avoir à écrire de requête SQL (ou à utiliser JDBC).
- L'objet Session possède un ensemble de méthodes (save(), delete(), update(), ...) qui prennent directement un objet en paramètre et réalisent le mapping et la persistance en BD
 - Exemple : `currentSession.save(myEntity);`

- Introduction au choix et à l'utilisation d'un framework
- Présentation de Spring et de ses modules
- Les Design Patterns de Spring
- Hibernate : un outil de mapping objet-relationnel
- **Combiner Hibernate & Spring : TP**

References

- Tutoriels Spring de baeldung.com :
<https://www.baeldung.com/spring-mvc-tutorial>
- Tutoriel Hibernate de J.M.Doudoux :
<https://www.jmdoudoux.fr/java/dej/chap-hibernate.htm>
- Architecture Hibernate :
<https://docs.jboss.org/hibernate/orm/3.5/reference/fr-FR/html/architecture.html>
- Hibernate ORM mapping :
<https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/mapping.html>
- Bon usage de MVC : Ruby Midwest 2011 - Keynote: Architecture the Lost Years by Robert Martin
<https://www.youtube.com/watch?v=WpkDN78P884>