

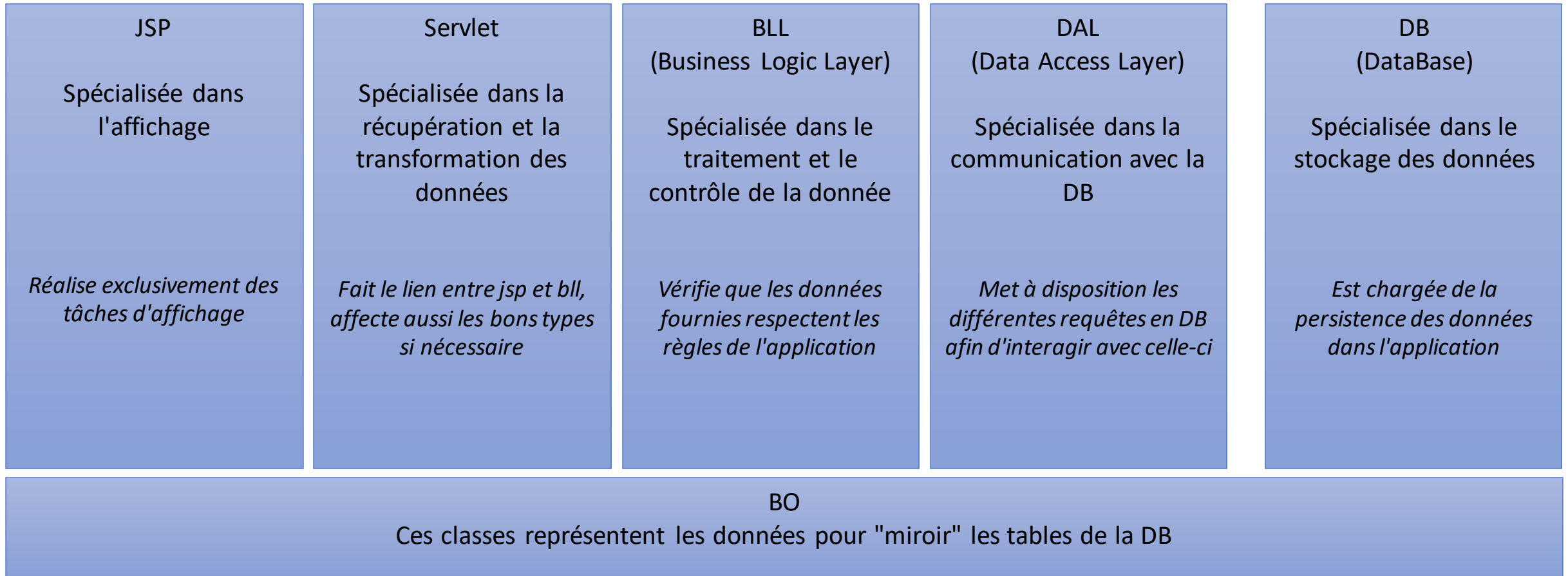
# Créer un projet JEE

De A à Z

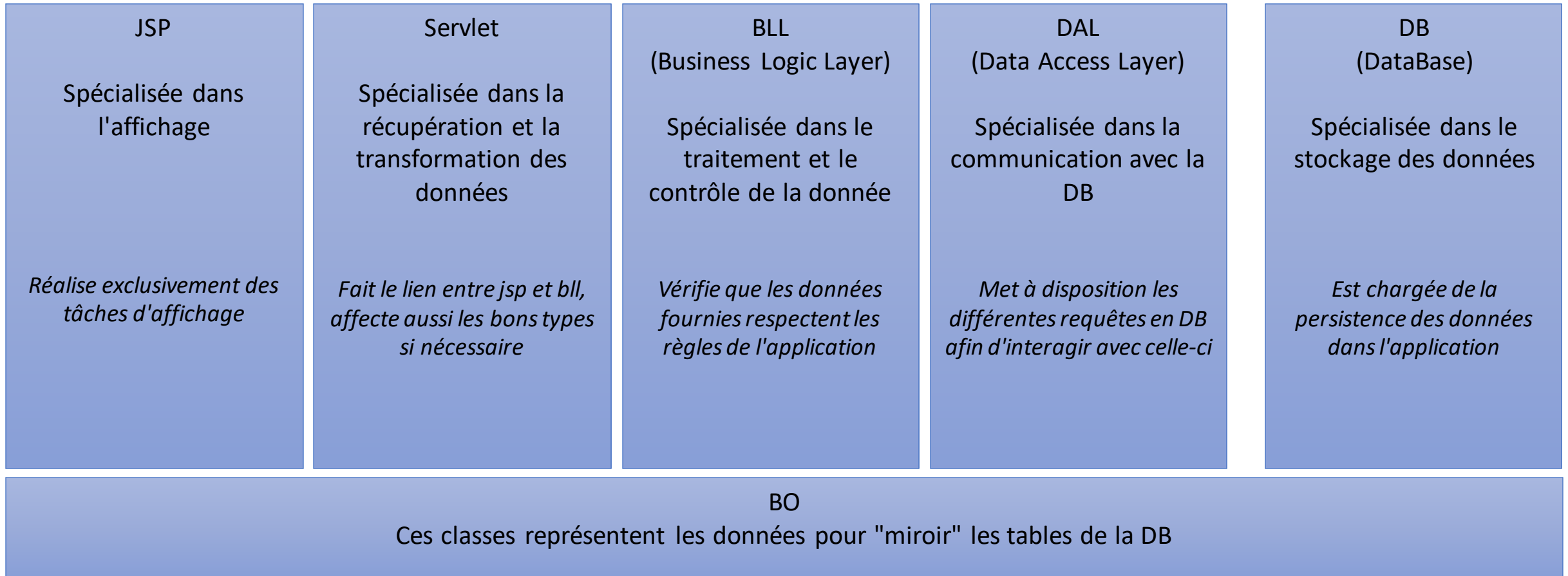


# Objectif :

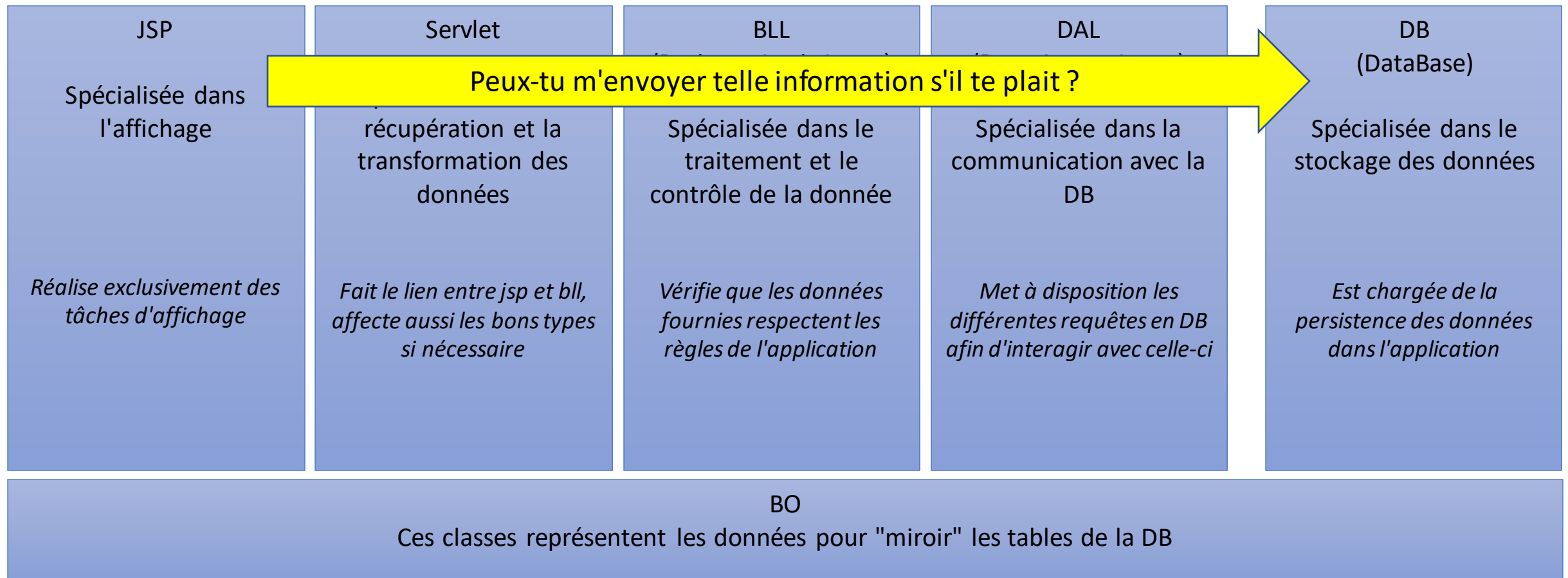
- Construire un projet respectant l'architecture JEE standard



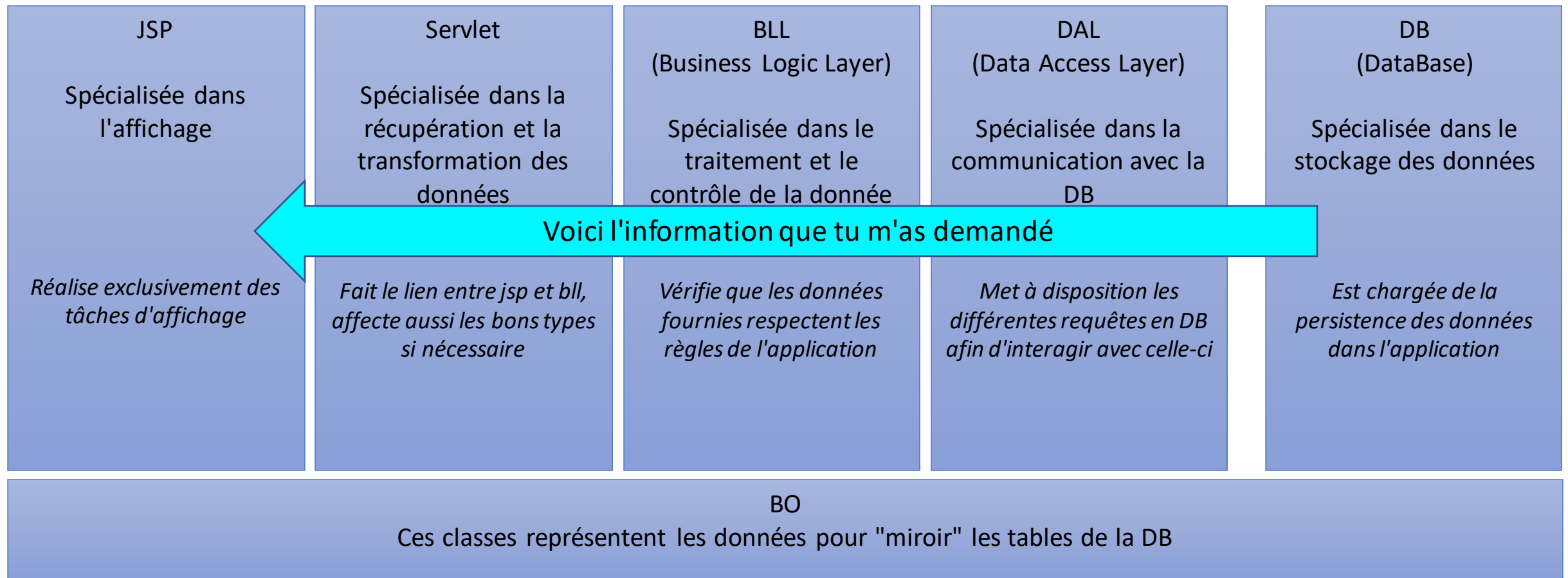
- La circulation des informations est simple :
  - Soit on cherche à afficher de l'information contenue en base. Dans ce cas, l'information circule depuis la base de données vers la JSP.
  - Soit on cherche à modifier de l'information contenue en base. Dans ce cas, l'information circule depuis la JSP vers la base de données.
- Dans tous les cas, c'est la JSP qui initie la demande



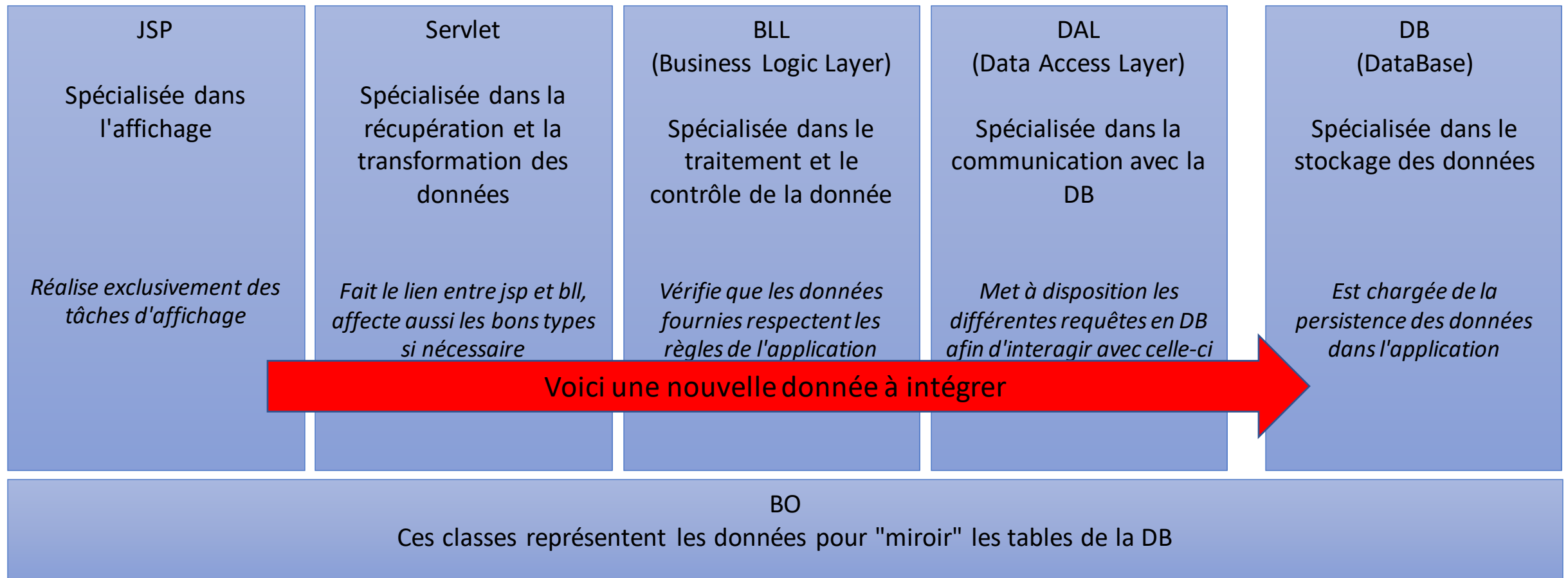
- La circulation des informations est simple :
  - Soit on cherche à afficher de l'information contenue en base. Dans ce cas, l'information circule depuis la base de données vers la JSP.
  - Soit on cherche à modifier de l'information contenue en base. Dans ce cas, l'information circule depuis la JSP vers la base de données.
- Dans tous les cas, **c'est la JSP qui initie la demande**



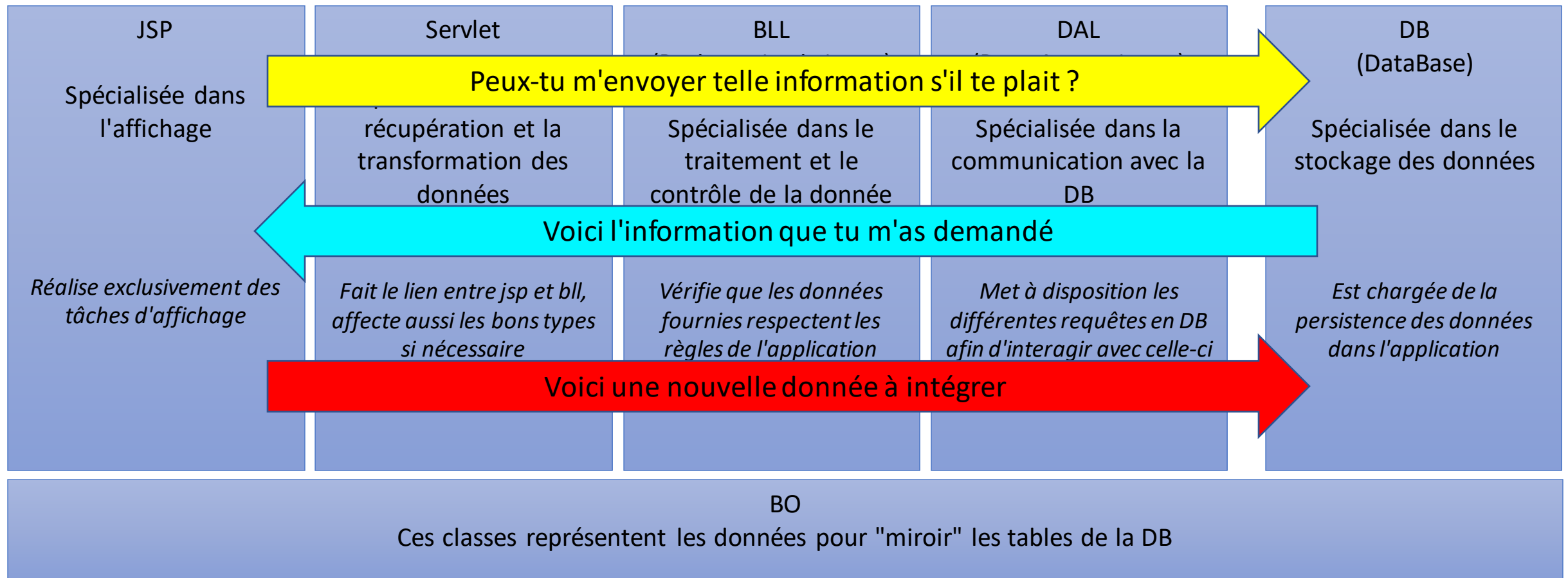
- La circulation des informations est simple :
  - Soit on cherche à afficher de l'information contenue en base. Dans ce cas, l'information circule depuis la base de données vers la JSP.
  - Soit on cherche à modifier de l'information contenue en base. Dans ce cas, l'information circule depuis la JSP vers la base de données.
- Dans tous les cas, c'est la JSP qui initie la demande



- La circulation des informations est simple :
  - Soit on cherche à afficher de l'information contenue en base. Dans ce cas, l'information circule depuis la base de données vers la JSP.
  - Soit on cherche à modifier de l'information contenue en base. Dans ce cas, **l'information circule depuis la JSP vers la base de données.**
- Dans tous les cas, c'est la JSP qui initie la demande



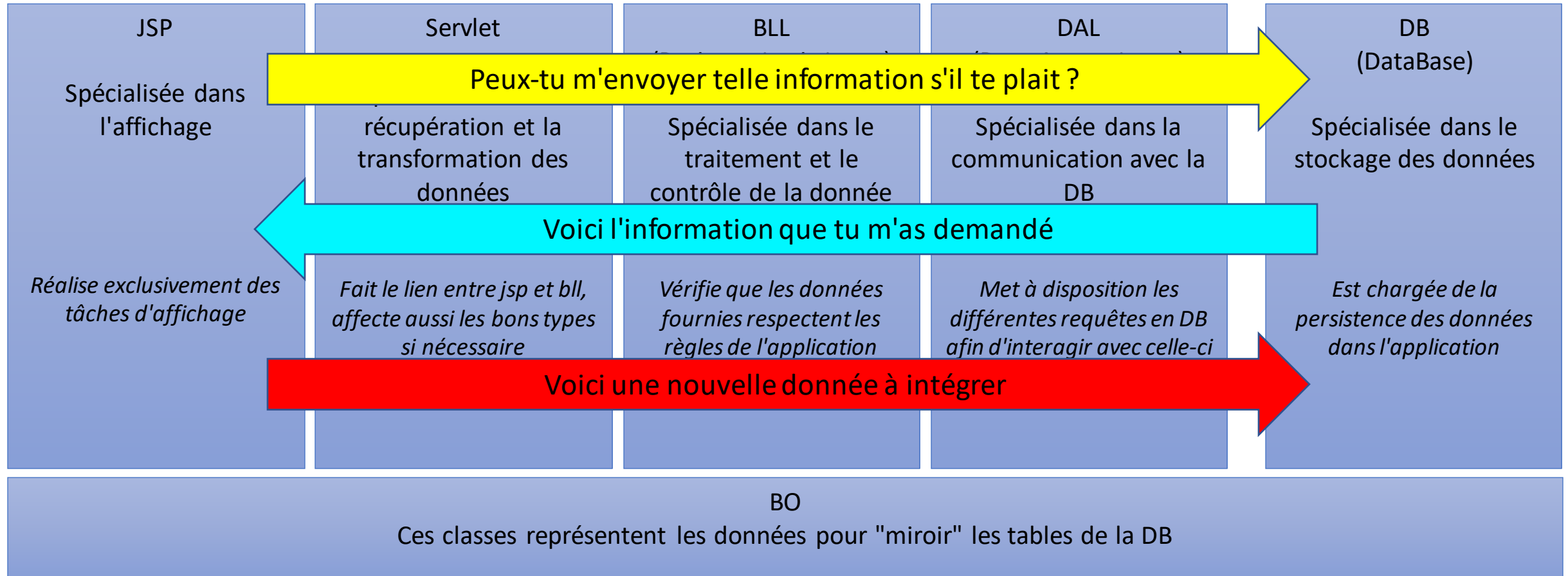
- La circulation des informations est simple :
  - Soit on cherche à afficher de l'information contenue en base. Dans ce cas, l'information circule depuis la base de données vers la JSP.
  - Soit on cherche à modifier de l'information contenue en base. Dans ce cas, l'information circule depuis la JSP vers la base de données.
- Dans tous les cas, c'est la JSP qui initie la demande



Lors du développement, nous suivrons le même ordre !

JSP -> Servlet -> BLL -> Dal -> DB

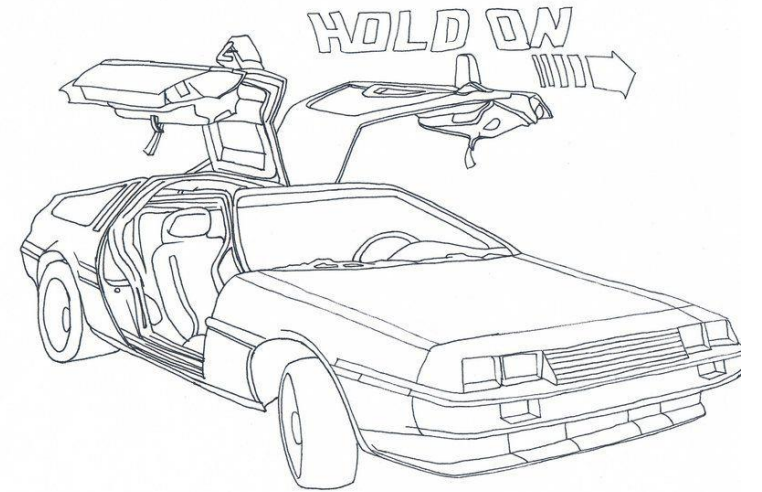
Il est souvent judicieux de commencer par nos objets BO





# Concrètement, qu'est-ce que ça donne ?

Le retour de la voiture



# Objectif :

- Reproduire un CRUD pour notre voiture

# Objectif :

- Reproduire un CRUD pour notre voiture



Notre point de départ sera notre objet BO

Autrement dit, nous allons commencer par répondre à la question "quels sont les attributs qui caractérisent une voiture ?"

BO

Ces classes représentent les données pour "miroir" les tables de la DB

# Objectif :

- Reproduire un CRUD pour notre voiture

```
@Entity
@Table(name="voitures")
public class Voiture {
    @Id @GeneratedValue (strategy=GenerationType.IDENTITY)
    @Column(name="id", insertable=false, updatable=false)
    private Integer id;

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

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

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

    @Column(name="kilometrage")
    private Integer kilometrage;
}
```

Réponse :

Une voiture est constituée d'un identifiant unique, d'un modèle, d'une marque, d'une immatriculation et d'un kilométrage.  
N'oubliez pas votre constructeur sans argument + les getters setters !



BO

Ces classes représentent les données pour "miroir" les tables de la DB

# Objectif :

- Reproduire un CRUD pour notre voiture

Première étape :  
Validée !



BO

Ces classes représentent les données pour "miroir" les tables de la DB

# Objectif :

- Reproduire un CRUD pour notre voiture

JSP

Spécialisée dans  
l'affichage

*Réalise exclusivement des  
tâches d'affichage*

Première étape :  
Validée ! Au tour de  
notre JSP



BO

Ces classes représentent les données pour "miroir" les tables de la DB

# Objectif :

- Reproduire un CRUD pour notre voiture

JSP

Spécialisée dans  
l'affichage

*Réalise exclusivement des  
tâches d'affichage*



Cette fois-ci, nous chercherons à répondre à la question suivante :

"Quelles informations est-ce que je souhaite afficher, et comment ?"

# Objectif :

- Reproduire un CRUD pour notre voiture

JSP

Spécialisée dans  
l'affichage

*Réalise exclusivement des  
tâches d'affichage*

```
<c:forEach var="current" items="${ voitures }">
  <ul>
    <li>${ current.modele }</li>
    <li>${ current.marque }</li>
    <li>${ current.immatriculation }</li>
    <li>${ current.kilometrage }</li>
  </ul>
</c:forEach>
```

Réponse :

Je veux afficher une liste de voitures ! Pour l'heure, je me fiche de comment l'information est récupérée : je fais confiance à la Servlet pour m'envoyer une liste de voitures qui s'appelle "voitures"





# Objectif :

- Reproduire un CRUD pour notre voiture

JSP

Spécialisée dans  
l'affichage

*Réalise exclusivement des  
tâches d'affichage*

Deuxième étape :  
Validée !



# Objectif :

- Reproduire un CRUD pour notre voiture

## JSP

Spécialisée dans  
l'affichage

*Réalise exclusivement des  
tâches d'affichage*

## Servlet

Spécialisée dans la  
récupération et la  
transformation des  
données

*Fait le lien entre jsp et bll,  
affecte aussi les bons types  
si nécessaire*

Deuxième étape :

Validée ! On peut  
désormais s'attaquer à  
notre Servlet



# Objectif :

- Reproduire un CRUD pour notre voiture

## Servlet

Spécialisée dans la  
récupération et la  
transformation des  
données

*Fait le lien entre jsp et bll,  
affecte aussi les bons types  
si nécessaire*



Deux questions cette fois :

"Quelle information récupérer pour  
notre JSP ? Quelle information récupérer  
de notre JSP ?"

# Objectif :

- Reproduire un CRUD pour notre voiture

## Servlet

Spécialisée dans la récupération et la transformation des données

*Fait le lien entre jsp et bll, affecte aussi les bons types si nécessaire*

```
// La servlet récupère une information pour la JSP
List<Voiture> voitures = vm.selectAll();
request.setAttribute("voitures", voitures);
```

```
// La servlet récupère une information de la JSP
// Première étape : recuperation des donnees
String modele = request.getParameter("modele");
String marque = request.getParameter("marque");
String immatriculation = request.getParameter("immatriculation");
String kilometrage = request.getParameter("kilometrage");
```

# Objectif :

- Reproduire un CRUD pour notre voiture

## Servlet

Spécialisée dans la  
récupération et la  
transformation des  
données

*Fait le lien entre jsp et bll,  
affecte aussi les bons types  
si nécessaire*

## Réponses :

Je récupère la liste de voitures à  
afficher (GET)

OU

Je récupère les informations me  
permettant de créer une voiture  
(POST)



# Objectif :

- Reproduire un CRUD pour notre voiture

## Servlet

Spécialisée dans la  
récupération et la  
transformation des  
données

*Fait le lien entre jsp et bll,  
affecte aussi les bons types  
si nécessaire*

Troisième étape validée !



# Objectif :

- Reproduire un CRUD pour notre voiture

**Servlet**

Spécialisée dans la récupération et la transformation des données

*Fait le lien entre jsp et bll, affecte aussi les bons types si nécessaire*

**BLL**  
(Business Logic Layer)

Spécialisée dans le traitement et le contrôle de la donnée

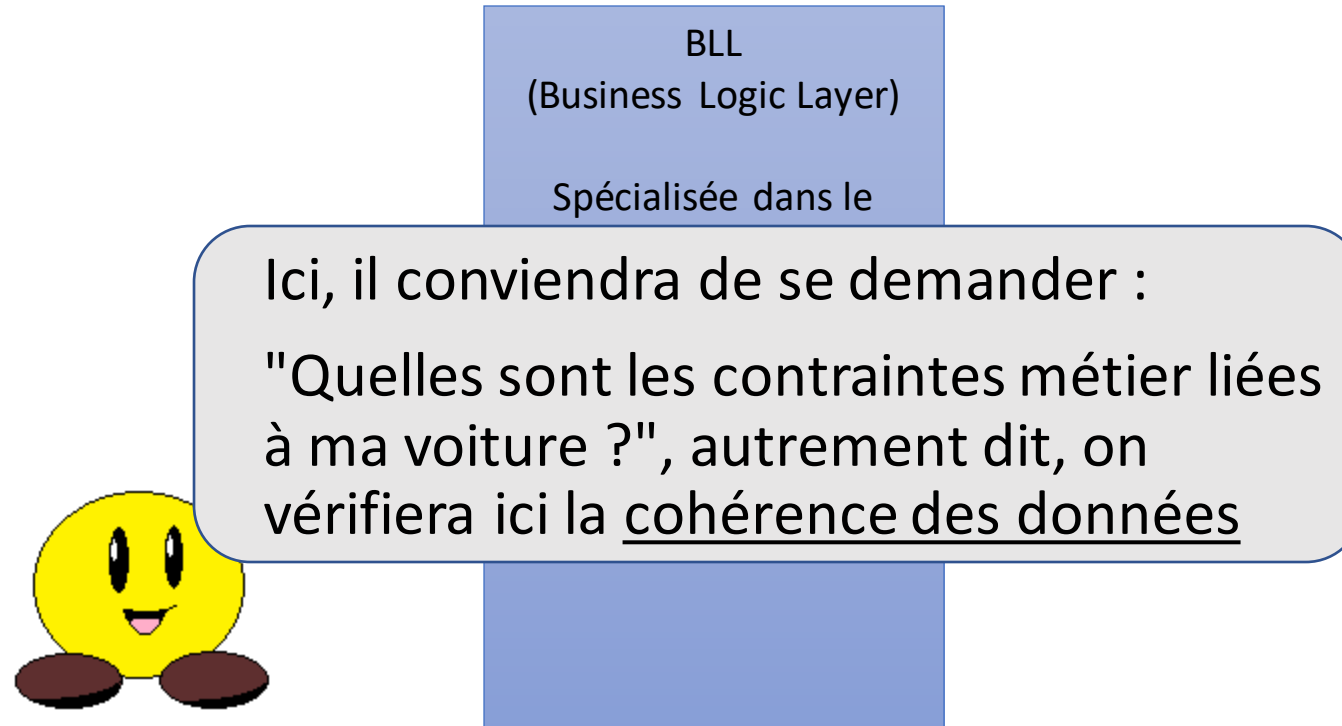
*Vérifie que les données fournies respectent les règles de l'application*

Troisième étape validée !  
Au tour du manager  
(couche BLL)



# Objectif :

- Reproduire un CRUD pour notre voiture





# Objectif :

- Reproduire un CRUD pour notre voiture

BLL  
(Business Logic Layer)

Spécialisée dans le

```
// Une immatriculation française doit faire 9 caractères
if (immatriculation.length() == 9) {
    // Un kilométrage négatif n'aurait aucun sens
    if (kilometrageReel >= 0 ) {
        dao.insert(modele, marque, immatriculation, kilometrageReel);
    }
}
```

# Objectif :

- Reproduire un CRUD pour notre voiture

Une fois les contraintes métier validées, je peux attaquer la dernière étape : la communication avec la base de données

BLL  
(Business Logic Layer)

Spécialisée dans le traitement et le contrôle de la donnée

*Vérifie que les données fournies respectent les règles de l'application*

DAL  
(Data Access Layer)

Spécialisée dans la communication avec la DB

*Met à disposition les différentes requêtes en DB afin d'interagir avec celle-ci*



# Objectif :

- Reproduire un CRUD pour notre voiture

Objectif de cette dernière étape :  
mettre à disposition les méthodes  
nous permettant de requêter notre  
base de données !



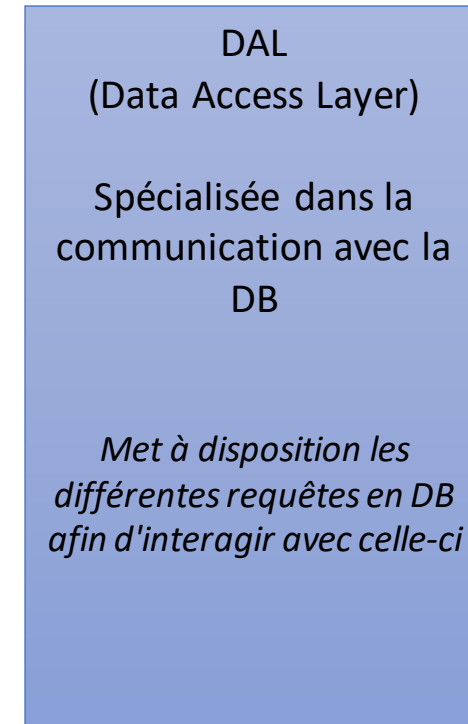
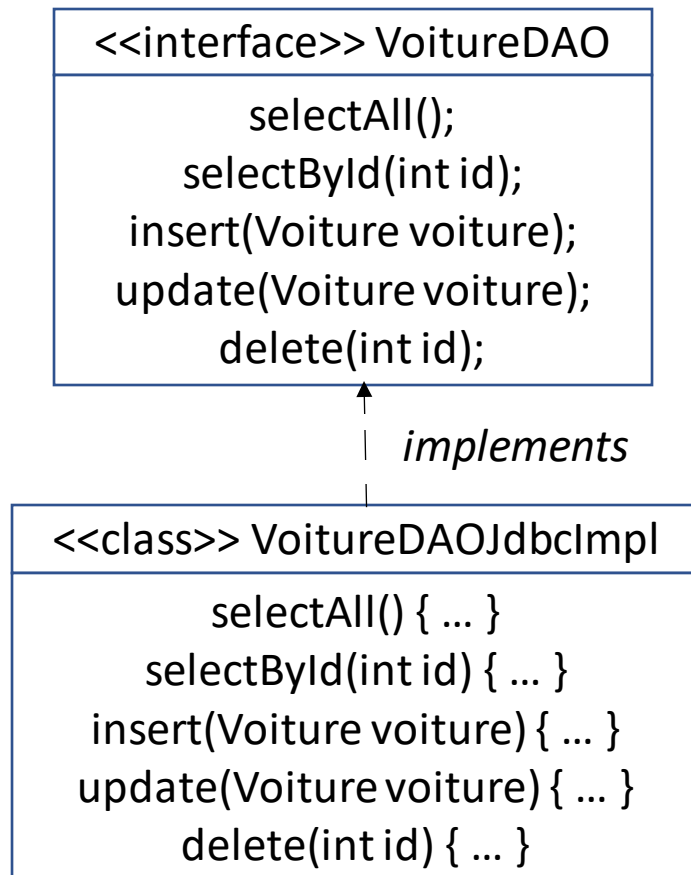
DAL  
(Data Access Layer)

Spécialisée dans la  
communication avec la  
DB

*Met à disposition les  
différentes requêtes en DB  
afin d'interagir avec celle-ci*

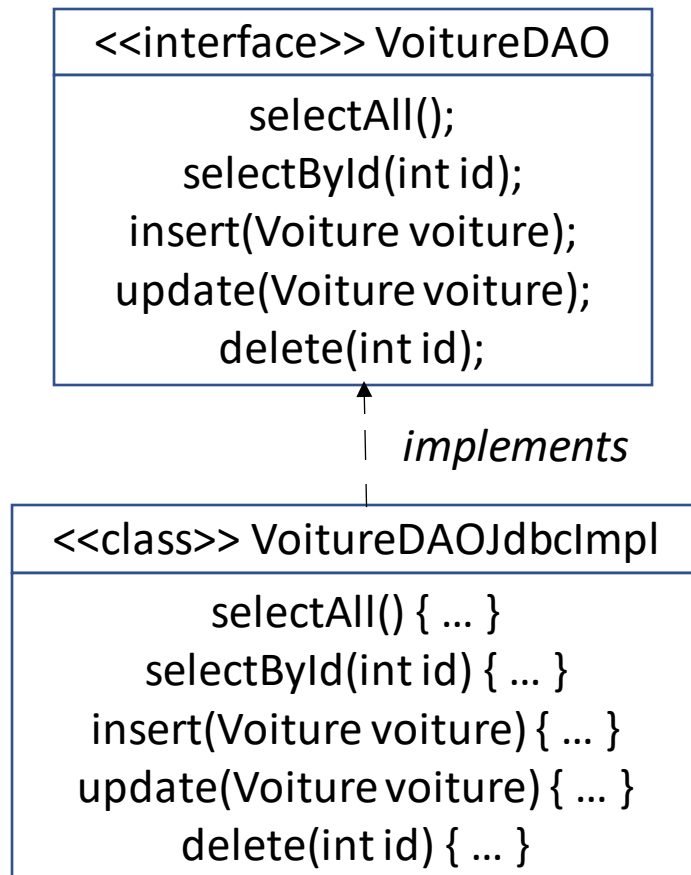
# Objectif :

- Reproduire un CRUD pour notre voiture



# Objectif :

- Reproduire un CRUD pour notre voiture



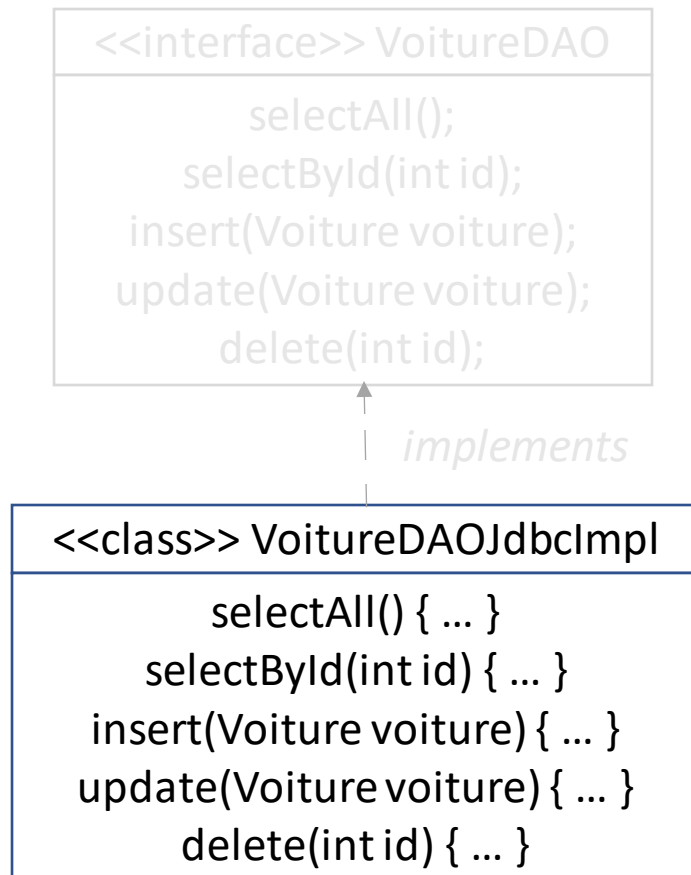
N'oubliez pas !

Une interface permet de définir quelles méthodes devront être implémentées. C'est le cas de VoitureDAO.

Une classe implémentant une interface doit proposer une implémentation pour les méthodes de l'interface. Ainsi, VoitureDAOJdbcImpl doit disposer des 5 méthodes définies dans VoitureDAO

# Objectif :

- Reproduire un CRUD pour notre voiture



C'est donc bien dans `VoitureDAOJdbcImpl` que nous écrirons le code permettant de requêter la base de données

# Objectif :

- Reproduire un CRUD pour notre voiture

```
@Override
public List<Voiture> selectAll() {
    return session.createQuery("from Voiture").list();
}
```

```
Transaction t = session.beginTransaction();
session.save(voiture);
t.commit();
```

etc

DAL  
(Data Access Layer)

Spécialisée dans la  
communication avec la  
DB

*Met à disposition les  
différentes requêtes en DB  
afin d'interagir avec celle-ci*

# Objectif :

- Reproduire un CRUD pour notre voiture

Le DAO réalisé, nous avons fini de mettre en place les différentes couches



DAL  
(Data Access Layer)

Spécialisée dans la  
communication avec la  
DB

*Met à disposition les  
différentes requêtes en DB  
afin d'interagir avec celle-ci*



# Objectif :

- Reproduire un CRUD pour notre voiture



On a fi...ni ?

Pourquoi il reste un bloc  
bleu ?

## JSP

Spécialisée dans  
l'affichage

*Réalise exclusivement des  
tâches d'affichage*

## Servlet

Spécialisée dans la  
récupération et la  
transformation des  
données

*Fait le lien entre jsp et bll,  
affecte aussi les bons types  
si nécessaire*

## BLL (Business Logic Layer)

Spécialisée dans le  
traitement et le  
contrôle de la donnée

*Vérifie que les données  
fournies respectent les  
règles de l'application*

## DAL (Data Access Layer)

Spécialisée dans la  
communication avec la  
DB

*Met à disposition les  
différentes requêtes en DB  
afin d'interagir avec celle-ci*

## DB (DataBase)

Spécialisée dans le  
stockage des données

*Est chargée de la  
persistance des données  
dans l'application*

## BO

Ces classes représentent les données pour "miroir" les tables de la DB

# Objectif :

- Reproduire un CRUD pour notre voiture

Ce bloc bleu, c'est la base de données. Si, si, souvenez-vous, le fameux système SQLServer ou MySQL que nous avons passé une demie-journée à configurer.

Heureusement, maintenant que nous l'avons connecté une fois, les prochaines utilisations seront plus simples !

Il nous suffit d'y créer les tables dont nous aurons besoin dans notre projet. En principe, Hibernate va même s'en charger pour nous : nous n'avons donc même pas besoin de nous en préoccuper !

A partir de ce point, vous n'avez pas encore vu les notions : vous les découvrirez en Février avec Spring/Hibernate :)

DB  
(DataBase)

Spécialisée dans le  
stockage des données

*Est chargée de la  
persistance des données  
dans l'application*

# Objectif :

- Reproduire un CRUD pour notre voiture

DB  
(DataBase)

Spécialisée dans le  
stockage des données

*Est chargée de la  
persistance des données  
dans l'application*

# Objectif :

- Reproduire un CRUD pour notre voiture

En parlant de Hibernate...

Il nous manque deux petits quelque chose pour que notre application fonctionne.

DB  
(DataBase)

Spécialisée dans le  
stockage des données

*Est chargée de la  
persistance des données  
dans l'application*

# Objectif :

- Reproduire un CRUD pour notre voiture

En parlant de Hibernate...

Il nous manque deux petits quelque chose pour que notre application fonctionne.

 `hibernate.cfg.xml`



 `springContext.xml`

# Objectif :

- Reproduire un CRUD pour notre voiture

En parlant de Hibernate...

Il nous manque deux petits quelque chose pour que notre application fonctionne.

 `hibernate.cfg.xml`  
 `springContext.xml`

"Nooon, pas eux, je n'ai toujours pas compris à quoi ils servent !"

# Objectif :

- Reproduire un CRUD pour notre voiture

Commençons par le applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/b
xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
xsi:schemaLocation="http://www.springframework.or

<bean id="dao" class="dal.VoitureDAOJdbcImpl" />

<bean id="vm" class="bll.VoitureManager" >
  <property name="dao" ref="dao"></property>
</bean>

</beans>
```

Un "bean" dans le jargon Spring désigne une classe (n'importe laquelle)

Mon fichier applicationContext.xml a pour rôle d'instancier les beans définis à l'intérieur afin de rendre mes instances disponibles dans toute l'application.

# Objectif :

- Reproduire un CRUD pour notre voiture

Commençons par le applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/b
xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
xsi:schemaLocation="http://www.springframework.or

<bean id="dao" class="dal.VoitureDAOJdbcImpl" />

<bean id="vm" class="bll.VoitureManager" >
  <property name="dao" ref="dao"></property>
</bean>

</beans>
```

Ici, je suis donc en train de créer une instance de VoitureManager et une instance de VoitureDAOJdbcImpl.

Par ailleurs, mon VoitureDAOJdbcImpl, d'id "dao", est passé en référence au VoitureManager. Ainsi, "dao" est considéré comme une propriété de "vm"



# Objectif :

- Reproduire un CRUD pour notre voiture

Commençons par le springContext.xml

```
<beans xmlns="http://www.springframework.org/schema/b
xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
xsi:schemaLocation="http://www.springframework.or

<bean id="dao" class="dal.VoitureDAOJdbcImpl" />

<bean id="vm" class="bll.VoitureManager" >
  <property name="dao" ref="dao"></property>
</bean>

</beans>
```

L'attribut "dao" de la classe VoitureManager prendra l'instance de dao telle que définie dans le springContext.xml

```
public class VoitureManager {
  private VoitureDAO dao;
```

# Objectif :

- Reproduire un CRUD pour notre voiture

Commençons par le springContext.xml

```
<beans xmlns="http://www.springframework.org/schema/b
xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
xsi:schemaLocation="http://www.springframework.or

<bean id="dao" class="dal.VoitureDAOJdbcImpl" />

<bean id="vm" class="bll.VoitureManager" >
  <property name="dao" ref="dao"></property>
</bean>

</beans>
```

Récupération du VoitureManager créé par Spring

```
ApplicationContext context =
    new ClassPathXmlApplicationContext("springContext.xml");
vm = context.getBean("vm", VoitureManager.class);
```

# Objectif :

- Reproduire un CRUD pour notre voiture

Et enfin, le hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
    <property name="hbm2ddl.auto">none</property>
    <property name="dialect">org.hibernate.dialect.SQLServerDialect</property>
    <property name="connection.url">jdbc:sqlserver://localhost;databasename=MyDB</property>
    <property name="connection.username">USER_BDD</property>
    <property name="connection.password">password</property>
    <property name="connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>

    <mapping class="bo.Voiture"/>
  </session-factory>
</hibernate-configuration>
```

# Objectif :

- Reproduire un CRUD pour notre voiture

Et enfin, le hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
    <property name="hbm2ddl.auto">none</property>
    <property name="dialect">org.hibernate.dialect.SQLServerDialect</property>
    <property name="connection.url">jdbc:sqlserver://localhost;databasename=MyDB</property>
    <property name="connection.username">USER_BDD</property>
    <property name="connection.password">password</property>
    <property name="connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>

    <mapping class="bo.Voiture"/>
  </session-factory>
</hibernate-configuration>
```

À laisser à "none" pour ne pas créer / modifier les tables existantes  
À mettre à "create" ou "update" pour que les tables soient créées de façon dynamique

# Objectif :

- Reproduire un CRUD pour notre voiture

Et enfin, le hibernate.cfg.xml

Le dialecte à utiliser pour discuter avec notre Système de Gestion de Base de Données (SGBD). SQLServerDialect ou MySQL8Dialect dans notre cas

```
<hibernate-configuration>
  <session-factory>
    <property name="hbm2ddl.auto">none</property>
    <property name="dialect">org.hibernate.dialect.SQLServerDialect</property>
    <property name="connection.url">jdbc:sqlserver://localhost;databasename=MyDB</property>
    <property name="connection.username">USER_BDD</property>
    <property name="connection.password">password</property>
    <property name="connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>

    <mapping class="bo.Voiture"/>
  </session-factory>
</hibernate-configuration>
```

# Objectif :

- Reproduire un CRUD pour notre voiture

Et enfin, le hibernate.cfg.xml

L'adresse complète permettant d'accéder à notre BDD.

```
<hibernate-configuration>
  <session-factory>
    <property name="hbm2ddl.auto">none</property>
    <property name="dialect">org.hibernate.dialect.SQLServerDialect</property>
    <property name="connection.url">jdbc:sqlserver://localhost;databasename=MyDB</property>
    <property name="connection.username">USER_BDD</property>
    <property name="connection.password">password</property>
    <property name="connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>

    <mapping class="bo.Voiture"/>
  </session-factory>
</hibernate-configuration>
```

# Objectif :

- Reproduire un CRUD pour notre voiture

Et enfin, le hibernate.cfg.xml

L'utilisateur et le mot de passe utilisé pour se connecter à la BDD

```
<hibernate-configuration>
  <session-factory>
    <property name="hbm2ddl.auto">none</property>
    <property name="dialect">org.hibernate.dialect.SQLServerDialect</property>
    <property name="connection.url">jdbc:sqlserver://localhost;databasename=MyDB</property>
    <property name="connection.username">USER_BDD</property>
    <property name="connection.password">password</property>
    <property name="connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>

    <mapping class="bo.Voiture"/>
  </session-factory>
</hibernate-configuration>
```

# Objectif :

- Reproduire un CRUD pour notre voiture

Et enfin, le hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
    <property name="hbm2ddl.auto">none</property>
    <property name="dialect">org.hibernate.dialect.SQLServerDialect</property>
    <property name="connection.url">jdbc:sqlserver://localhost;databasename=MyDB</property>
    <property name="connection.username">USER_BDD</property>
    <property name="connection.password">password</property>
    <property name="connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>

    <mapping class="bo.Voiture"/>
  </session-factory>
</hibernate-configuration>
```

Le driver utilisé pour se connecter à la base de données. A titre d'information, "jdbc" est un acronyme pour "Java DataBase Connectivity"



# Objectif :

- Reproduire un CRUD pour notre voiture

Et enfin, le hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
    <property name="hbm2ddl.auto">none</property>
    <property name="dialect">org.hibernate.dialect.SQLServerDialect</property>
    <property name="connection.url">jdbc:sqlserver://localhost;databasename=MyDB</property>
    <property name="connection.username">USER_BDD</property>
    <property name="connection.password">password</property>
    <property name="connection.driver_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>

    <mapping class="bo.Voiture"/>
  </session-factory>
</hibernate-configuration>
```

Enfin, les classes BO nécessitant d'être mappées à une table dans la base de données

# Objectif :

- Reproduire un CRUD pour notre voiture

Et voilà !

Avant de tester, assurez-vous d'avoir bien ajouté toutes vos librairies dans /WEB-INF/lib. Librairies Sprint, librairies Hibernate, jdbc, jstl...

A vous de jouer !

