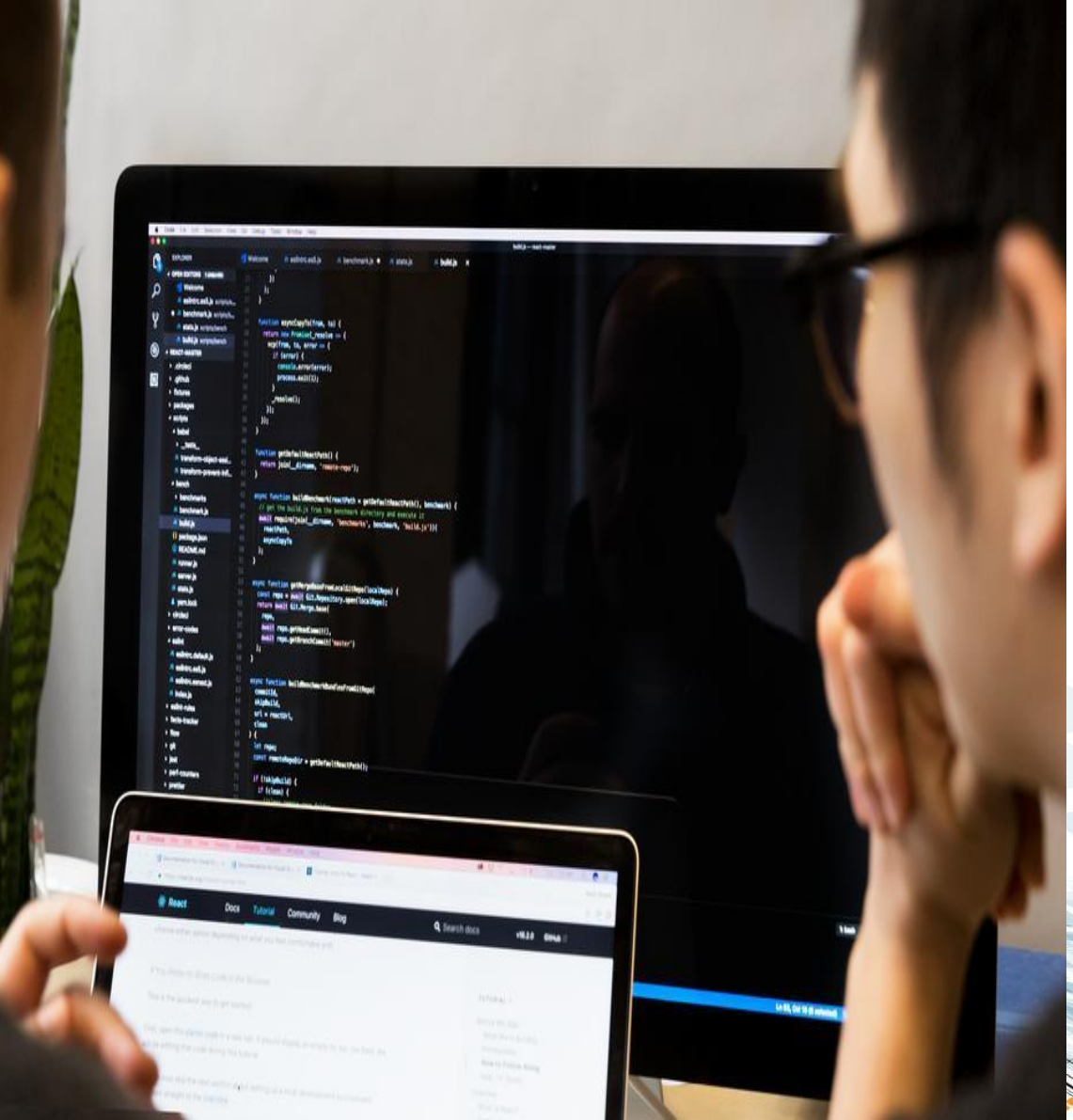
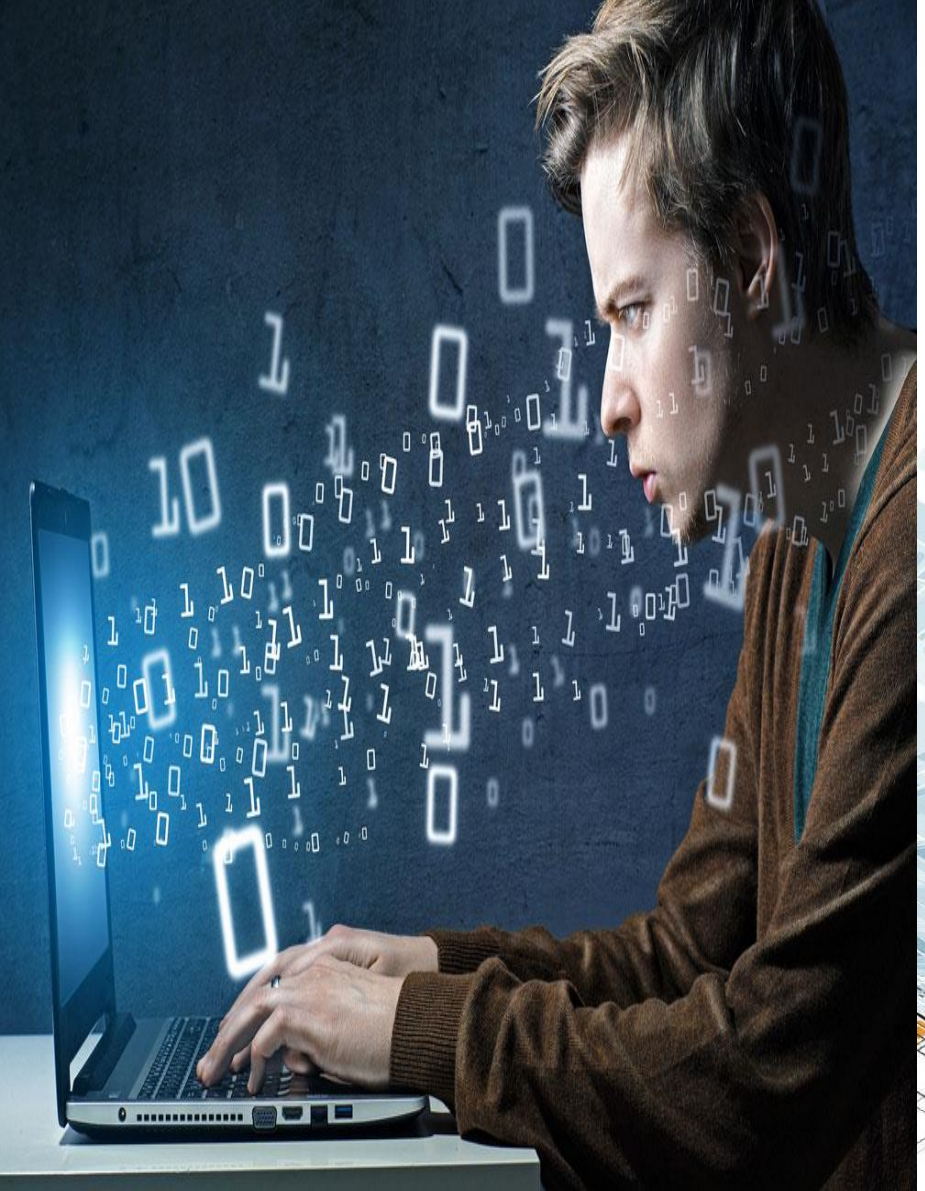


Spring MVC

Initiation
Approfondissement

Woodson JUSTE

wjuste@dawan.fr



Les objectifs

Construire des applications web en utilisant Spring Web, Spring Data JPA et Spring Boot - Implémentation de services web REST

Pré-requis

Maîtrise la programmation orientée objets

Connaissance de JPA-Hibernate

Notions HTML/CSS/JS

Plan

- **Comprendre les frameworks Spring** :
  - Spring Framework : inversion de contrôle et injection de dépendances
  - Spring boot – Spring Data JPA – Spring Web

- **Construire une application Spring Web MVC**
  - Utilisation d'un framework de vues : JSP ou Thymeleaf
  - Validation de formulaires

- **Spring Security**

- **Réaliser un mapping des données avec Spring Data JPA**

- **Implémenter des web services REST (Representational State Transfer)**
  - Architecture REST
  - *Spring RestController*

Comprendre les frameworks Spring

Exigences d'un projet informatique

- Exigences fonctionnelles:
  - Une application est créée pour répondre , tout d'abord, aux besoins fonctionnels des entreprises.
- Exigences Techniques :
  - Les performances:
    - Temps de réponse
    - Haute disponibilité et tolérance aux pannes
    - Eviter le problème de montée en charge
  - La maintenance:
    - Une application doit évoluer dans le temps.
    - Doit être fermée à la modification et ouverte à l'extension
  - Sécurité
  - Portabilité
  - Distribution
  - Capacité de communiquer avec d'autres applications distantes.
  - Capacité de fournir le service à différents type de clients (Desk TOP, Mobile, SMS, http…)
  - …..
  - Coût du logiciel

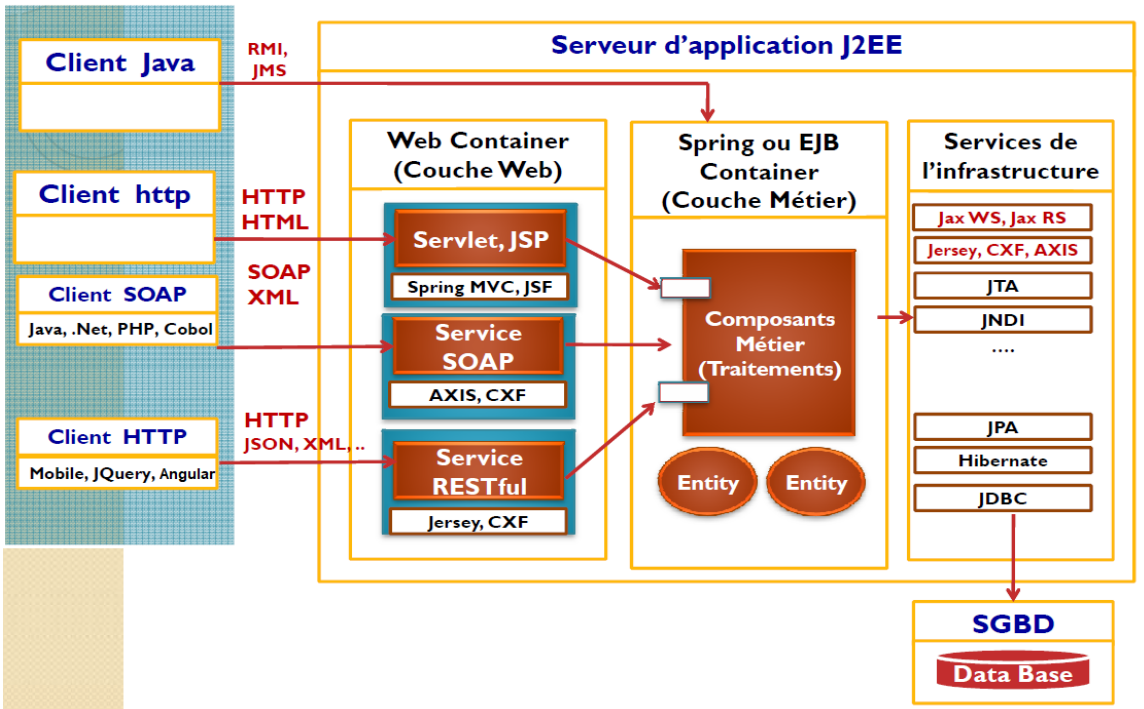
Comprendre les frameworks Spring

Constat

- Il est très difficile de développer un système logiciel qui respecte ces exigences sans utiliser l'expérience des autres :
  - Serveur d'application JEE:
    - JBOSS, Web Sphere,
    - GlassFish, Tomcat,
    - …
  - Framework pour l'Inversion de contrôle:
    - Spring (Conteneur léger)
    - EJB (Conteneur lourd)
  - Frameworks :
    - Mapping objet relationnel (ORM) : JPA, Hibernate, Toplink, …
    - Applications Web : Struts, JSF, SpringMVC
    - ….
  - Middlewares :
    - RMI, CORBA : Applications distribuées
    - JAXWS pour Web services SOAP
    - JAXRS pour les Web services RESTful
    - JMS : Communication asynchrone entre les application
    - …

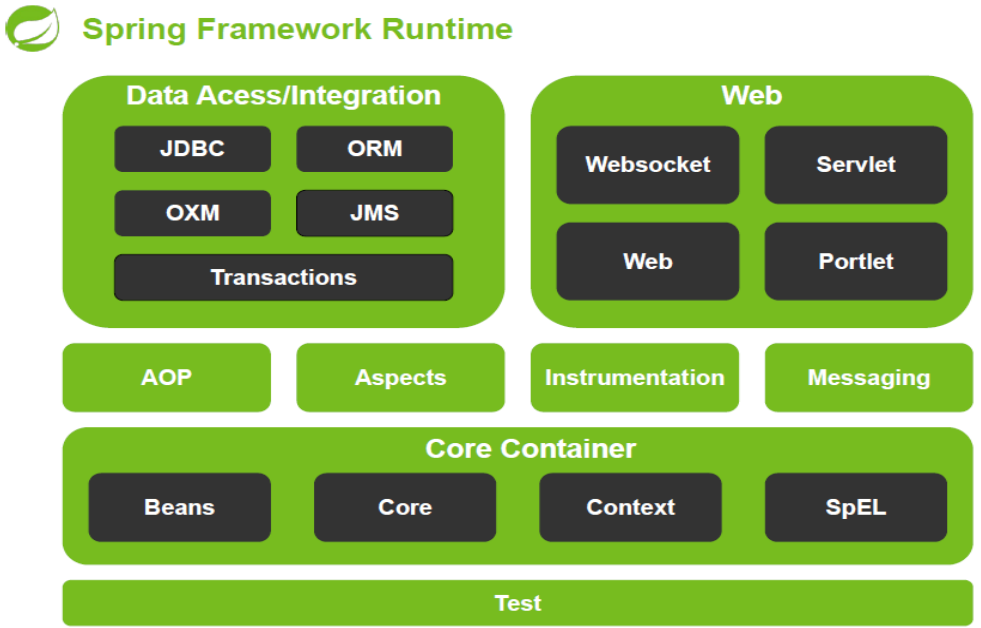
Comprendre les frameworks Spring

Architecture Java EE (JEE)



Comprendre les frameworks Spring

Spring Framework et Architecture



Comprendre les frameworks Spring

Spring Framework et Architecture

Spring Security:

- C’est un module de Spring Framework qui fournit des fonctionnalités de sécurité pour les applications web.
- Il s'agit d'un module qui fournit les fonctionnalités de sécurité pour les applications, comme l'authentification, l'autorisation, la gestion des rôles

Spring Boot:

- Il s'agit d'un module qui facilite la création d'applications Spring en fournissant une configuration automatique et une configuration minimale pour les applications, ainsi qu'un ensemble de fonctionnalités pour les tâches courantes de développement d'application.

Comprendre les frameworks Spring

Spring Framework et Architecture

Spring Data:

- Il s'agit d'un module qui fournit les fonctionnalités pour la gestion des données dans les applications, comme Spring Data JPA pour la gestion des données avec JPA, Spring Data MongoDB pour la gestion des données avec MongoDB

Spring Data JPA

- C’ est un module de Spring Framework qui fournit une abstraction pour l'accès aux bases de données relationnelles en utilisant l'API de persistance Java (JPA).
- Il permet aux développeurs de créer des applications qui accèdent aux bases de données de manière simple et efficace en utilisant les fonctionnalités de Spring Framework, telles que la gestion des transactions, la gestion des erreurs, etc. Il fournit des fonctionnalités pour la gestion des entités JPA, la gestion des requêtes, la gestion des relations entre entités, etc.

Comprendre les frameworks Spring

Spring Framework et Architecture

Spring Data JPA

Il offre plusieurs fonctionnalités pour l'accès aux bases de données, y compris :

- **Repository** : Il s'agit d'un modèle de programmation qui permet aux développeurs de créer des objets DAO (Data Access Object) de manière simple en utilisant les fonctionnalités de Spring Data JPA. Il fournit des méthodes pour les opérations courantes, comme la sélection, l'insertion, la mise à jour et la suppression de données.

- **JpaRepository** : Il s'agit d'une interface qui étend les fonctionnalités de la Repository pour les cas d'utilisation JPA. Il fournit des méthodes pour les opérations courantes, comme la recherche, l'insertion, la mise à jour et la suppression de données.

- **CrudRepository** : Il s'agit d'une interface de base qui fournit des méthodes pour les opérations CRUD courantes.

Comprendre les frameworks Spring

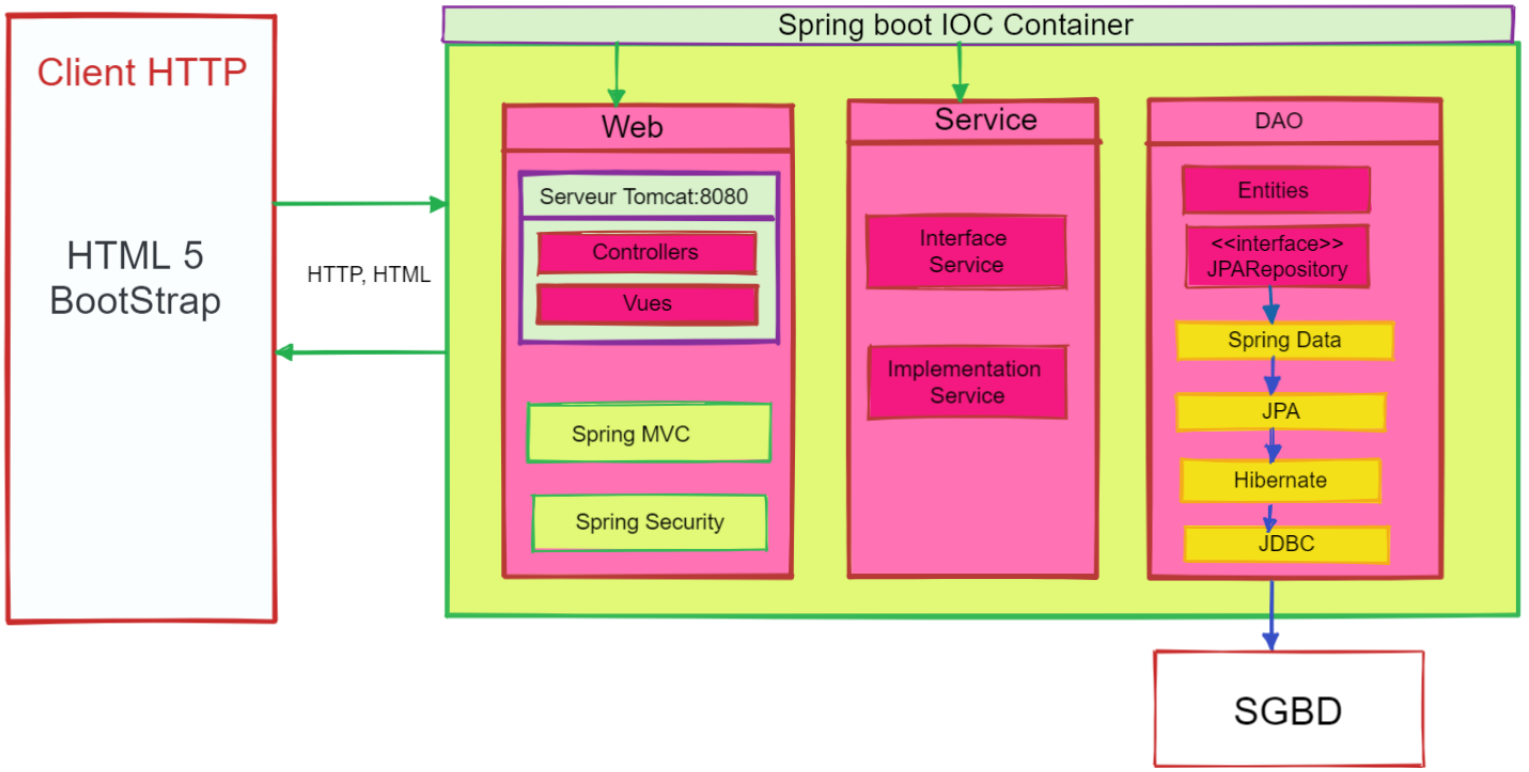
Spring Framework et Architecture

L'architecture Java EE avec Spring est un modèle d'architecture pour créer des applications web robustes et évolutives en utilisant les technologies Java EE et Spring.

Il utilise les avantages de Java EE pour la gestion des transactions, la sécurité, les services web, etc. et les avantages de Spring pour la gestion des dépendances, la gestion des transactions, la gestion de la sécurité, etc.

Comprendre les frameworks Spring

Spring Framework et Architecture



Comprendre les frameworks Spring

Spring Framework et Architecture

Il y a plusieurs couches dans une architecture Java EE avec Spring, ces couches incluent :

- **Couche de présentation** : Il s'agit de la couche qui gère les interactions avec l'utilisateur final, elle peut être implémentée en utilisant des technologies comme JSF, Spring MVC, etc.
- **Couche de service** : Il s'agit de la couche qui gère les règles métier de l'application, elle peut être implémentée en utilisant des technologies comme EJB, Spring Beans, etc.
- **Couche de données** : Il s'agit de la couche qui gère les données de l'application, elle peut être implémentée en utilisant des technologies comme JPA, Hibernate, Spring Data, etc.

Comprendre les frameworks Spring

Spring Framework et Architecture

Il y a plusieurs couches dans une architecture Java EE avec Spring, ces couches incluent :

- **Couche de persistance** : Il s'agit de la couche qui gère la persistance des données dans une base de données, elle peut être implémentée en utilisant des technologies comme JDBC, JPA, Hibernate, etc.

- **Couche de sécurité** : Il s'agit de la couche qui gère la sécurité de l'application, elle peut être implémentée en utilisant des technologies comme Java EE Security, Spring Security

Comprendre les frameworks Spring

Inversion de contrôle et injection de dépendances

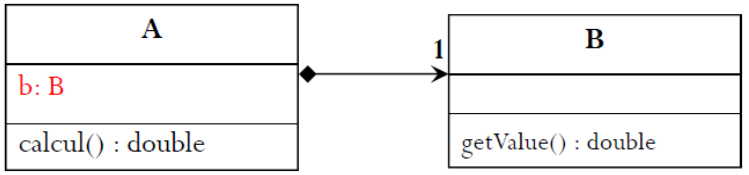
Quelques principes de conception

- Une application qui n'évolue pas meurt.
- Une application doit être fermée à la modification et ouverte à l'extension.
- Une application doit s'adapter aux changements
- Efforcez-vous à coupler faiblement vos classes.
- Programmer une interface et non une implémentation
- Etc..

Comprendre les frameworks Spring

Couplage fort

- Quand une classe A est lié à une classe B, on dit que la classe A est fortement couplée à la classe B.
- La classe A ne peut fonctionner qu'en présence de la classe B.
- Si une nouvelle version de la classe B (soit B2), est crée, on est obligé de modifier dans la classe A.
- Modifier une classe implique:
  - Il faut disposer du code source.
  - Il faut recompiler, déployer et distribuer la nouvelle application aux clients.
  - Ce qui engendre un cauchemar au niveau de la maintenance de l'application



Comprendre les frameworks Spring

Exemple de couplage fort

Exemple de couplage fort


```
package metier;
import dao.DaoImpl;
public class MetierImpl {
  private DaoImpl dao;
 public MetierImpl() {
  dao=new DaoImpl();
 }
public double calcul(){
 double nb=dao.getValue();
 return 2*nb;
}
}
```

```
package dao;
public class DaoImpl {
 public double getValue(){
 return(5);
 }
}
```

```
package pres;
import metier.MetierImpl;
public class Presentation {
private static MetierImpl metier;
public static void main(String[]
   args) {
metier=new MetierImpl();
System.out.println(metier.calcul());
}
}
```

Comprendre les frameworks Spring

Conclusion : Problème du couplage fort

- Dans l'exemple précédent, les classes MetierImpl et DaoImpl sont liées par un couplage fort. De même pour les classe Présentation et MetierImpl

- Cette conception nous ne a pas permis de créer une application fermée à la modification et ouverte à l'extension.

- En effet, la création d'une nouvelle version de la méthode getValue() de la classe DaoImpl, va nous obliger d'éditer le code source de l'application aussi bien au niveau de DaoImpl et aussi MetierImpl.

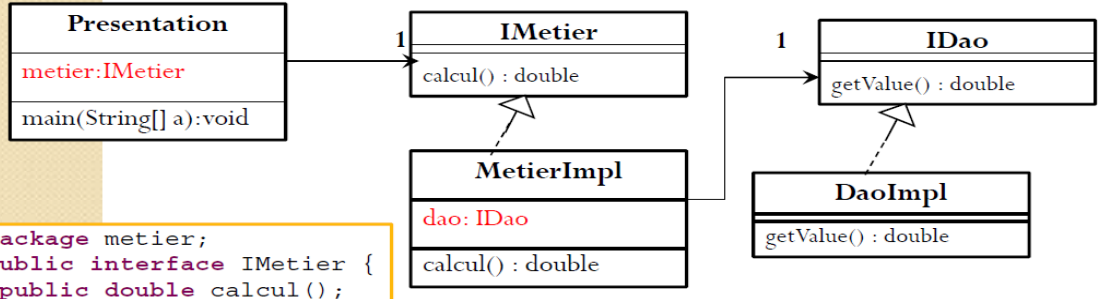
- De ce fait nous avons violé le principe « une application doit être fermée à la modification et ouverte à l'extension»

- Nous allons voir que nous pourrons faire mieux en utilisant le couplage faible.

Comprendre les frameworks Spring

Exemple Couplage faible

Exemple de coupage faible




```
package metier;
public interface IMetier {
 public double calcul();
}
```

```
package metier;
import dao.IDao;
public class MetierImpl
   implements IMetier {
 private IDao dao;
public double calcul() {
double nb=dao.getValue();
 return 2*nb;
}
// Getters et Setters
}
```

```
package dao;
public interface IDao {
public double getValue();
}
```

```
package dao;
public class DaoImpl implements IDao {
public double getValue() {
 return 5;
}
}
```

Thymeleaf

- Thymeleaf est un moteur de templates pour les applications web. Il est utilisé pour générer des pages HTML, XHTML ou d'autres types de documents à partir de modèles et de données.

- Il est conçu pour être utilisé avec les frameworks web tels que Spring Framework, Spring MVC

- Thymeleaf est basé sur la syntaxe XML et possède de nombreuses fonctionnalités pour la création de modèles, notamment :

    - Expressions: Thymeleaf permet d'inclure des expressions dans les modèles pour afficher les données de l'application

    - Instructions conditionnelles et de boucle: Thymeleaf permet de créer des instructions conditionnelles et de boucle dans les modèles pour afficher ou masquer des éléments en fonction des données de l'application.

    - Prise en charge des fragments: Thymeleaf permet de créer des fragments de modèles qui peuvent être réutilisés dans plusieurs pages.

Thymeleaf

- Thymeleaf est basé sur la syntaxe XML et possède de nombreuses fonctionnalités pour la création de modèles, notamment :
  - Validation de formulaire : Thymeleaf prend en charge la validation de formulaire côté serveur. Il permet de vérifier les erreurs de validation et de les afficher dans la page HTML
  - Insertion de scripts et de liens: Thymeleaf permet d'ajouter des scripts et des liens dans des pages HTML
  - Sécurité : Thymeleaf prend en charge les protocoles de sécurité courants, tels que la sécurité CSRF, pour renforcer la sécurité de l'application.

Thymeleaf

- La déclaration xmlns:th="http://www.thymeleaf.org" dans une page HTML est utilisée pour définir un espace de noms pour les attributs de Thymeleaf.
- Cela permet de déclarer un préfixe pour les attributs de Thymeleaf, en général "th" pour les utiliser dans la page.
- Thymeleaf propose de nombreux attributs pour ajouter des fonctionnalités aux éléments HTML. Voici quelques exemples courants d'attributs Thymeleaf :
  - th:text : cet attribut est utilisé pour définir le contenu texte d'un élément HTML.
  - Permet d'afficher la valeur d'un élément du model

```
<div>
     <label>Date de création:</label> <label
     th:text="${compte.dateCreation}"></label>
</div>
```

Thymeleaf

- Thymeleaf propose de nombreux attributs pour ajouter des fonctionnalités aux éléments HTML. Voici quelques exemples courants d'attributs Thymeleaf :
    - th:each : cet attribut est utilisé pour répéter un élément HTML pour chaque entrée d'une liste. Il prend une expression Thymeleaf qui définit la liste à parcourir et une variable qui contient chaque entrée de la liste à chaque itération

```
<tr th:each="op:${listOperations}">
     <td th:text="${op.numOperation}"></td>
     <td th:text="${op.class.simpleName}"></td>
     <td th:text="${op.dateOperation}"></td>
     <td th:text="${op.montant}"></td>
</tr>
```

Thymeleaf

- Thymeleaf propose de nombreux attributs pour ajouter des fonctionnalités aux éléments HTML. Voici quelques exemples courants d'attributs Thymeleaf :
    - th:if : cet attribut est utilisé pour afficher ou masquer un élément HTML en fonction de la valeur d'une expression Thymeleaf. Il prend une expression qui doit évaluer à true pour afficher l'élément, sinon il sera masqué

```
<!- Si l'utilisateur rentre un compte qui n'existe pas, on afficher le message de l'exception-->
 <div class="text-danger" th:if="${compteIntrouvableException}"
         th:text="${compteIntrouvableException}">
 </div>
```

Thymeleaf

- Thymeleaf propose de nombreux attributs pour ajouter des fonctionnalités aux éléments HTML. Voici quelques exemples courants d'attributs Thymeleaf :
    - th:href : cet attribut est utilisé pour définir l'attribut href d'un élément a ou link. Il permet de créer des liens vers une autre page ou vers des des routes définies dans une application web

```
<ul class="navbar-nav ms-auto">
    <!-- Avec Spring Security /login?logout permet de se connecter -->
    <li class="nav-item"><a class="nav-link" th:href="@{/login?logout}">Logout</a></li>
</ul>
```

Thymeleaf

- Thymeleaf propose de nombreux attributs pour ajouter des fonctionnalités aux éléments HTML. Voici quelques exemples courants d'attributs Thymeleaf :
  - L'attribut th:action de Thymeleaf est utilisé pour définir l'attribut action d'un élément form.
  - Cela permet de lier le formulaire à une méthode du contrôleur
  - l'attribut th:method est utilisé pour définir la méthode HTTP utilisée pour soumettre les données (GET, POST, etc.)

```
 <!– Le formulaire est lié à la méthode « consultercompte » du contrôleur-->
<form th:action="@{/consultercompte}" method="get">
   <div>
        <label>Code Compte: </label>
        <input type="text" name="numCompte" th:value="${numCompte}">
        <button type="submit" class="btn btn-primary">Ok</button>
   </div>
</form>
```

Thymeleaf

- Thymeleaf propose de nombreux attributs pour ajouter des fonctionnalités aux éléments HTML. Voici quelques exemples courants d'attributs Thymeleaf :
  - th:value : cet attribut est utilisé pour définir la valeur d'un élément du formulaire. Cela permet de lier les données de l'application aux champs de formulaire.
  - la variable est liée à la valeur du champ de saisie, ce qui signifie que si la variable change, la valeur du champ de saisie sera automatiquement mise à jour.

```
<form th:action="@{/consultercompte}" method="get">
   <div>
        <label>Code Compte: </label>
        <!-- th:value affiche dans la zone de texte la valeur de l'attribut numCompte qu'on a stocké
             dans le modele
             La zone de texte va garder le numero de compte entrer par l'utilisateur
         -->
        <input type="text" name="numCompte" th:value="${numCompte}">
        <button type="submit" class="btn btn-primary">Ok</button>
   </div>
</form>
```

Thymeleaf

- Thymeleaf propose de nombreux attributs pour ajouter des fonctionnalités aux éléments HTML. Voici quelques exemples courants d'attributs Thymeleaf :
  - th:fragment : cet attribut est utilisé pour définir un fragment du template qui peut être inclus dans d'autres templates. Il prend le nom du fragment comme valeur

```
<header>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        …
    </nav>
</header>

<!-
     Le rendu de chaque vue (vues enfants) se fera dans section grâce à layout:fragment="content"
     C'est la partie qui va changer tandis que le header et le footer se sont statiques
 -->
<section layout:fragment="content"></section>

<footer>
   <div class="navbar fixed-bottom">
      <small>@L'adresse de votre Banque</small>
   </div>
</footer>
```

Thymeleaf

- Thymeleaf propose de nombreux attributs pour ajouter des fonctionnalités aux éléments HTML. Voici quelques exemples courants d'attributs Thymeleaf :
  - #objects.simpleName permet d'obtenir le nom simple (sans le nom de package) de la classe d'un objet.
  - ${obj.class.simpleName} permet d'obtenir le nom simple de la classe de l'objet 'obj' et l'afficher dans la vue

```
<!-- Si compte est  un CompteCourant il affiche le découvert Sinon si c'est un CompteEpargne il affiche le taux
-->
<div th:if="${compte.class.simpleName == 'CompteCourant'}">
<label>Découvert:</label> <label th:text="${compte.decouvert}"></label>
</div>
```

Thymeleaf

- Thymeleaf propose de nombreux attributs pour ajouter des fonctionnalités aux éléments HTML. Voici quelques exemples courants d'attributs Thymeleaf :
    - L'attribut layout:decorate de Thymeleaf est utilisé pour inclure un fragment de template dans une autre page, cela permet de définir un modèle de page ou un gabarit qui peut être réutilisé dans plusieurs pages

```
<html lang="en"
xmlns:th="http://www.thymeLeaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-sprVinsecurity5"
layout:decorate="~{template1}"
>

 <head>…</head>
 <body>
    <!-- Le contenu de notre page va être à ce niveau -->
    <div layout:fragment="content">
       …
       …
    </div>
 </body>
</html>
```

Voici un exemple d'utilisation de l'attribut layout:decorate pour inclure un fragment de template nommé "template1" dans une page HTML

Thymeleaf

- La déclaration xmlns:layout dans une page HTML est utilisée pour définir un espace de noms pour les attributs de mise en page de Thymeleaf.
- Cela permet de déclarer une préfixe pour les attributs de mise en page de Thymeleaf, en général "layout" pour les utiliser dans la page.

```
<html lang="en"
xmlns:th="http://www.thymeLeaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-sprVinsecurity5"
layout:decorate="~{template1}"
>

 <head>…</head>
 <body>
    <!-- Le contenu de notre page va être à ce niveau -->
    <div layout:fragment="content">
       …
       …
    </div>
 </body>
</html>
```

Thymeleaf

- La déclaration xmlns:th="http://www.thymeleaf.org" dans une page HTML est utilisée pour définir un espace de noms pour les attributs de Thymeleaf.
- Cela permet de déclarer un préfixe pour les attributs de Thymeleaf, en général "th" pour les utiliser dans la page.

Introduction

Mise en pratique

Maven rappel :

- *Maven* utilise un paradigme connu sous le nom de ***Project Object Model* (POM)** afin de :
  - Décrire un projet logiciel,
  - Ses dépendances avec des modules externes
  - et l'ordre à suivre pour sa production.
- Il est livré avec un grand nombre de tâches (**GOLS**) prédéfinies, comme la compilation du code Java ou encore sa modularisation.