TP SPRING

STÉPHANE RIGAUT

Montreuil, le 07/10/2019

BNP PARIBAS

La banque d'un monde qui change

Sommaire

- Prérequis
- Application fil rouge
  - Spring IO
  - Intégration IDE
  - Vérification du squelette
  - Swagger
  - Lancement de l'application
  - On développe 🔄
    - Caractéristiques d'une bibliothèque
    - Ecriture de l'objet bibliothèque
    - Ecriture de l'API
    - Ecriture de la persistance
    - Ecriture du service (use-case)
    - Test et vérification manuel de l'API
- Architecture en couche par ordre de priorité (DDD)
  - Couche Domaine
  - Couche Application
  - Couche Exposition
  - Couche Infrastructure
- Gestion des exceptions
- Relation OneToMany
- Amélioration de la tolérance au changement
  - Mapping IHM
  - Isolation du domaine - Inversion des dépendances
- Validation
- Let's finish !

Prérequis

- IntelliJ
- JDK 8
- Maven

Application fil rouge

- Référentiel de bibliothèques
  - Créer une bibliothèque
    - En retournant l'identité de la nouvelle bibliothèque
  - Afficher le détail d'une bibliothèque
    - Gérer la non existence de la bibliothèque avec une exception
  - Lister l'ensemble des bibliothèques
  - Mettre à jour une bibliothèque
    - Mise à jour directement par l'IHM. Problème avec la méthode save qui va créer une nouvelle entrée en base de données par défaut si l'objet n'existe pas
    - Contrôle d'existence de la bibliothèque avant la mise à jour
  - Supprimer une bibliothèque
    - En utilisant deleteById
    - En utilisant delete

Spring IO

- Initialiser l'application sur https://start.spring.io/
- Choisir les critères
  - Project Maven
  - Langage JAVA
  - Spring Boot 2.1.8 (ou dernière version stable)
  - Group com.bnpparibas.itg.mylibraries
  - Artifact libraries
- Choisir les dépendances
  - Spring Web Starter
  - Spring Data JPA
  - H2 Database

Intégration IDE

- Extraire dans un répertoire le contenu
- Ouvrir l'IDE
- File -> Open
- Choisir le dossier contenant le squelette

Vérification du squelette

- Vérifier que le fichier pom.xml existe
  - Les dépendances sont correctes
  - Le plugin spring-boot-maven-plugin est présent. Il permet de créer un livrable exécutable (jar, war …). Il se trouve par défaut dans le répertoire du projet dans /target
- Vérifier le fichier de configuration de l'application
  - Répertoire -> main/resources/application.properties
  - Copier le contenu du fichier de l'url github (https://github.com/Stephhhh/mylibraries)

Swagger

- Ca sert à quoi ?
  - Offre une interface web permettant d'explorer et tester les différentes APIs
  - Permet de générer une documentation des APIs

- Dans le squelette
  - Ajouter les dépendances de swagger dans le pom.xml du projet
    - springfox-swagger2
    - springfox-swagger-ui
  - Copier la classe SwaggerConfig de github et la mettre au même niveau que la classe LibraryApplication

Lancement de l'application

- Lancer l'application
  - Aller dans la classe principale (LibrariesApplication)
  - Clique droit -> debug
- Ouvrir un navigateur et aller sur l'url http://localhost:8080
- Vérifier les endpoints (interfaces disponibles)
  - http://localhost:8080/swagger-ui.html
  - http://localhost:8080/h2

Caractéristiques d'une bibliothèque

- Caractéristiques (attributs)
  - Identifiant unique [Long]
  - Type (Associative, nationale, publique, scolaire, universitaire) [enum]
  - Adresse (numéro, rue, code postal, ville) [int, String, int, String]
  - Directeur (prénom, nom) [String, String]

- Bonnes pratiques
  - Penser à découper en plusieurs objets plutôt que de mettre tous les attributs dans une seule classe
  - Encapsulation
    - Les attributs d'une classe ne peuvent pas être directement manipulés de l'extérieur de la classe
      - Utiliser le mot clé private pour définir les attributs de la classe
      - Définir des méthodes d'accès à ces attributs (getter) avec le mot clé public

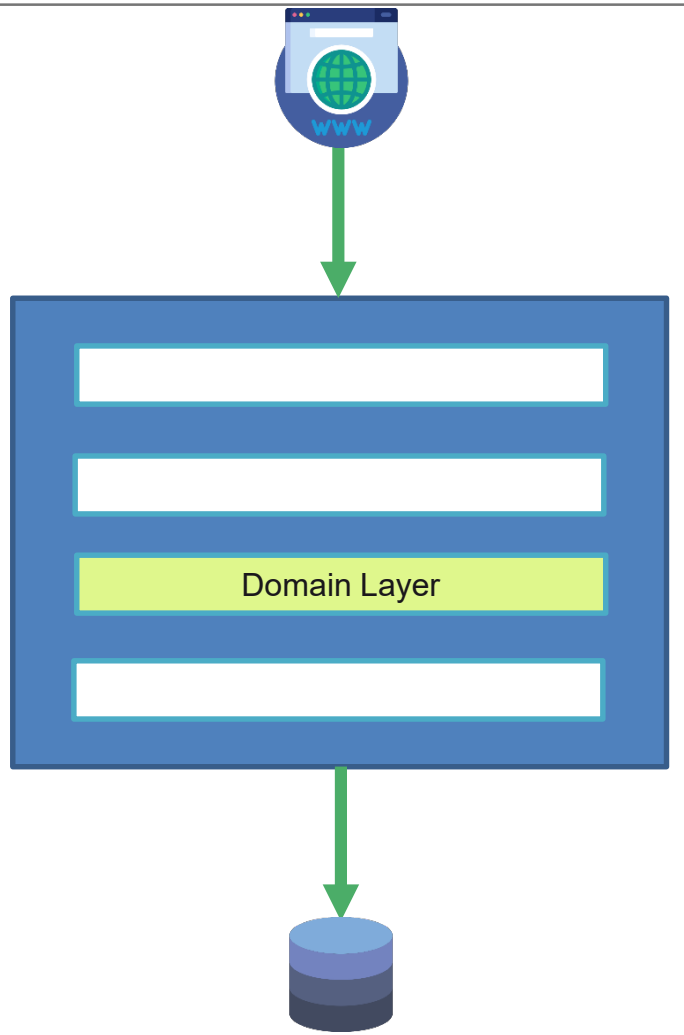
Ecriture de l'objet bibliothèque

- Penser objet et encapsulation

```java
public class Library {
    public String id;
    public Type type;

    public int numberAddress;
    public String streetAddress;
    public int postalCodeAddress;
    public String cityAddress;

    public String surnameDirector;
    public String nameDirector;
}
```

```java
public class Library {
    private String id;
    private Type type;
    private Address address;
    private Director director;

    public Library(String id, Type type, Address address, Director director) {
        this.id = id;
        this.type = type;
        this.address = address;
        this.director = director;
    }

    public String getId() {
        return id;
    }

    public Type getType() {
        return type;
    }

    public Address getAddress() {
        return address;
    }

    public Director getDirector() {
        return director;
    }
}
```

Architecture en couche

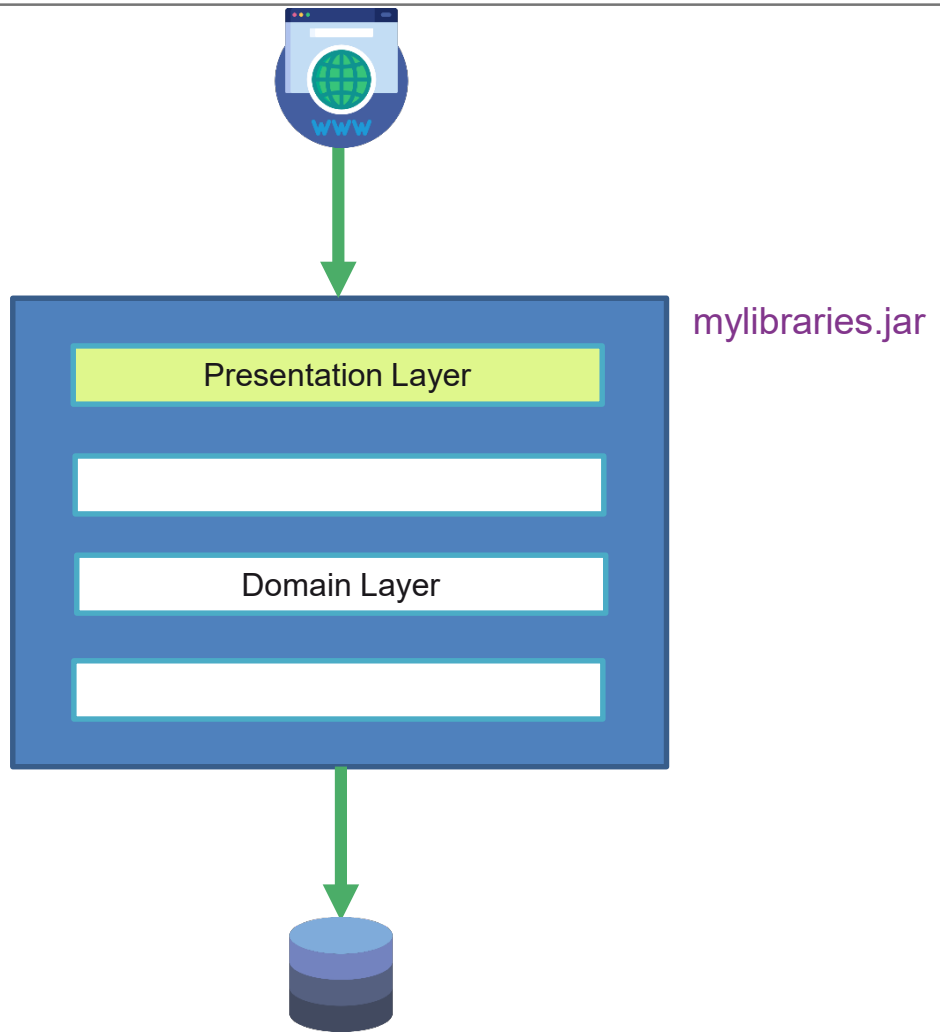


mylibraries.jar

Ecriture de l'API

- Bonnes pratiques : https://blog.octo.com/designer-une-api-rest/
  - KISS
  - Granularité moyenne
  - Noms > verbes
  - Pluriel > singulier
  - Casse (spinal-case)
  - CRUD (POST, GET, PUT, DELETE)
- Annotations
  - @RestController (@Controller + @ResponseBody)
  - @RequestMapping (GET, POST, PUT, DELETE …)
  - @RequestBody, @PathVariable, @RequestParam
  - @ResponseStatus
  - @Valid

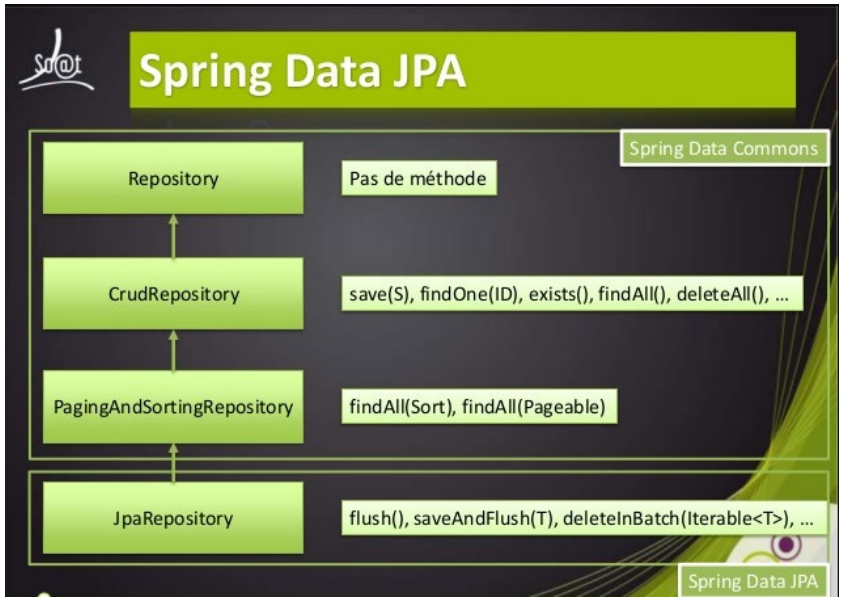
Architecture en couche



mylibraries.jar

Ecriture de la persistance

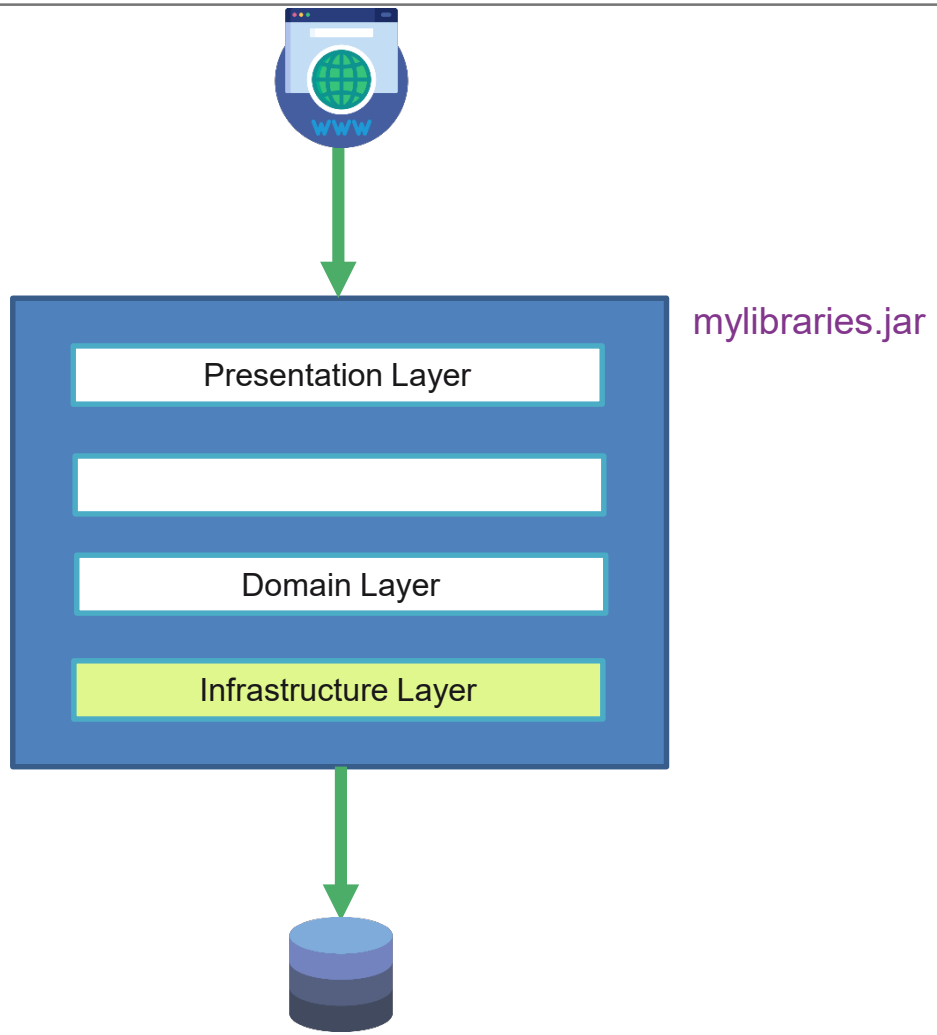
- Utilisation de spring-data-jpa
  - CrudRepository (CRUD opérations)
  - PagingAndSortingRepository (fournit en plus la possibilité de paginer et d'ordonner)
  - JpaRepository (fournit en plus la possibilité de forcer la synchronisation avec la base de données -> flush)



Ecriture de la persistance

- Choisir le nom de la table associée à la classe avec @Entity
- Utiliser l'attribut @Column pour définir le nom de la colonne dans la table associé à l'attribut
- Identifier l'attribut unique et choisir le type de génération
  - @Id
  - @GeneratedValue
- Pour notre type de bibliothèque qui utilise un enum
  - @Enumerated(EnumType.STRING)
- Définir les relations avec les autres objets
  - Si les données des objets en relations sont stockées dans la même table
    - @Embedded sur l'attribut
    - @Embeddable sur la classe fille
  - Si les données des objets en relations sont stockées dans une table à part
    - @OneToOne
    - @OneToMany
    - @ManyToOne

Architecture en couche

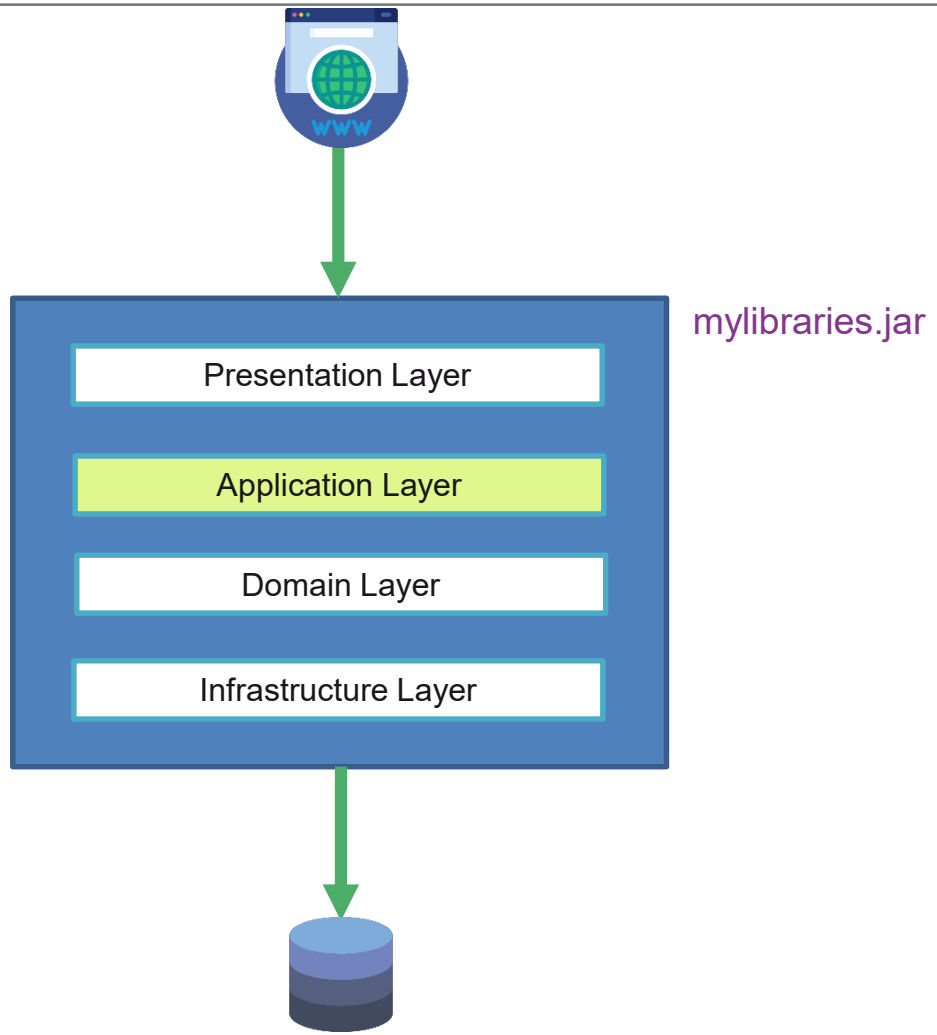


mylibraries.jar

Ecriture du service (use-case)

- Implémentation de/des la fonctionnalité(s) métier
- Déclare les méthodes qui doivent s'exécuter dans un contexte transactionnel
- Reçoit les requêtes/appels des entrées publiques (APIs, web, etc)
- Annotations
  - @Service
  - @Transactional

Architecture en couche



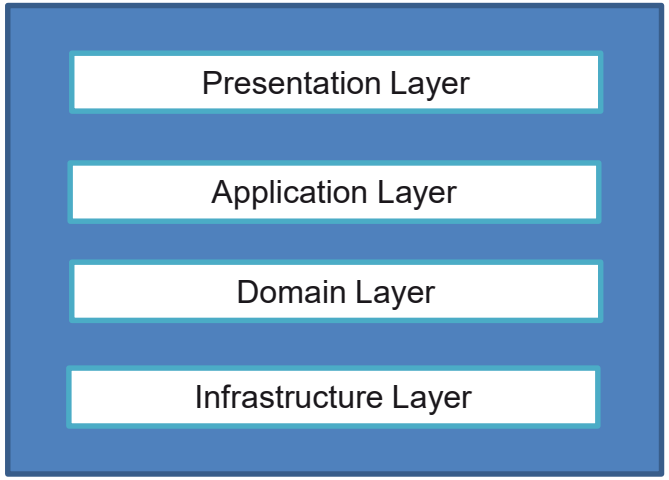
mylibraries.jar

Test et vérification manuel de l'API (tp-spring-0)

- http://localhost:8080/swagger-ui.html
- http://localhost:8080/h2

Architecture en couche (tp-spring-1)

- Chaque couche à une responsabilité
- Permet d'isoler le code métier du reste du code
- Le code métier est représenté par la couche application et la couche domaine
- La définition de l'API se retrouve dans la couche présentation
- L'implémentation du code d'accès à la base de données se trouve dans la couche infrastructure



Couche Domaine

- Protégée du reste du monde (aucune dépendance vers d'autres modules)
- Contient la logique métier (règles métier)
- Utilisation de pattern tactiques
  - Entities (Objets ayant une identité qui reste la même au cours de l'application indépendamment de leurs attributs)
  - Value Objects (Objects sans identité définis seulement par leurs attributs)
  - Repositories (abstraction de la persistance – agnostique des détails techniques du stockage)
  - ...


Domaine
Framework
Domaine
Persistance
REST
La logique métier
protégée du reste du monde

Couche Application

- Implémentation des use cases (fonctionnalités métier)
- Coordinateur (appelle d'autres application services si besoin)
- S'occupe du contexte transactionnel
- Reçoit les requêtes/appels des entrées publiques (APIs, web, etc) via la couche exposition
- Annotations
  - @Service
  - @Transactional


Application
L'orchestration des use cases
Exposition
Domaine
Infrastructure
Application

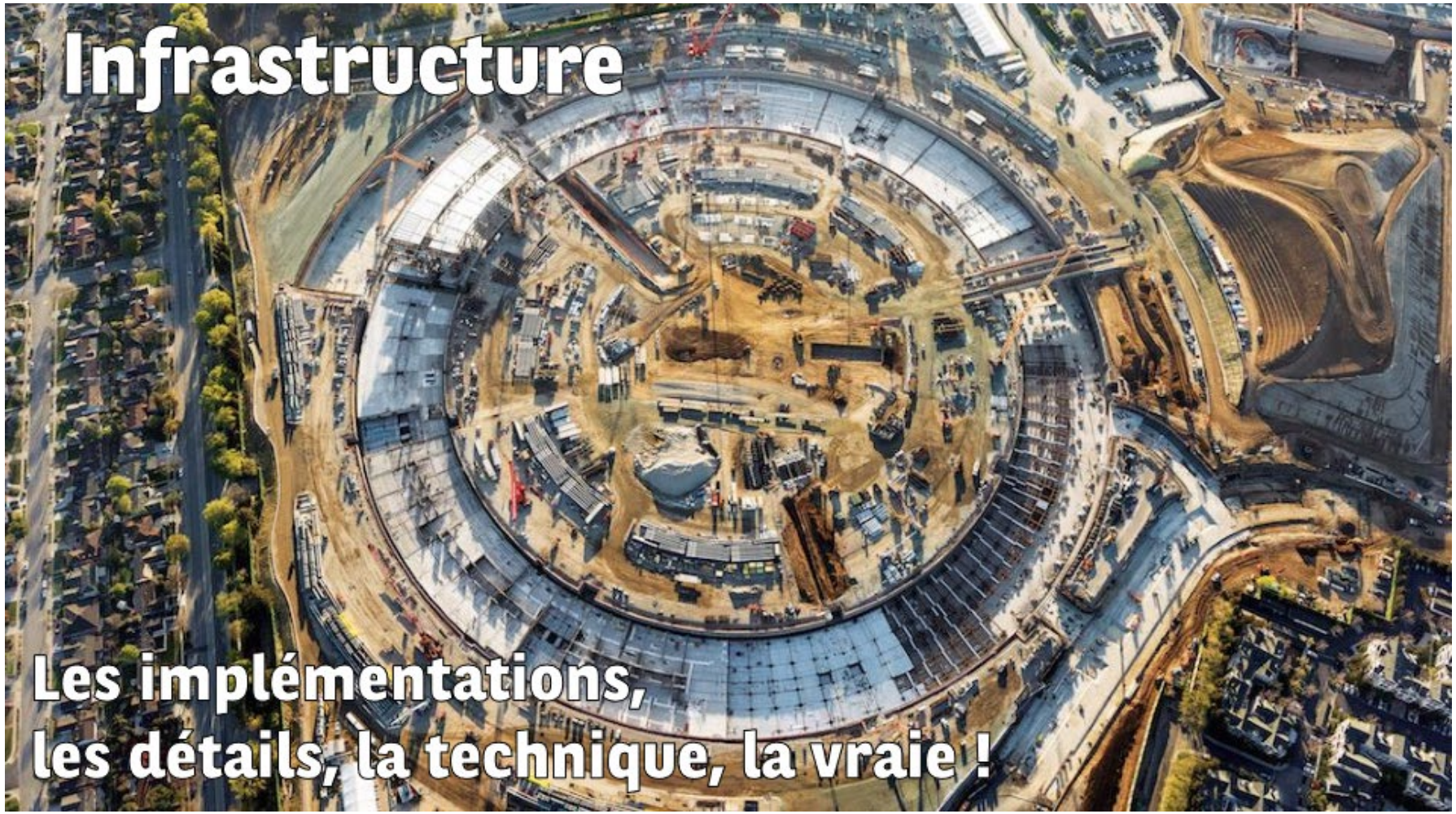
Couche Exposition

- Ouvert au monde extérieur
- Exposition des APIs REST
- Adaptateurs
- DTO (Data Transfer Object)
- Annotations
  - @RestController (@Controller + @ResponseBody)
  - @RequestMapping (GET, POST, PUT, DELETE …)
  - @RequestBody, @PathVariable, @RequestParam
  - @Valid
  - ResponseEntity


Exposition
JURASSIC WORLD
La porte ouverte
sur le monde extérieur

Couche Infrastructure

- Implémentation des détails techniques
  - Configuration et accès aux stockages de données
  - Envoi de mails
  - Audit
  - Appels à des services externes à l'application
- Couche d'anti-corruption
  - Adapter les résultats des appels externes aux objets du domaine


Infrastructure
Les implémentations,
les détails, la technique, la vraie !

Gestion des exceptions (tp-spring-2)

- Pouvoir gérer les exceptions de manière centralisé
  - Utilisation d'exception de type unchecked (runtime)
  - Créer sa propre exception qui hérite de RuntimeException
- Annotations
  - @ControllerAdvice
  - @ExceptionHandler
  - @ResponseStatus

Relation OneToMany (tp-spring-3)

- Nous voulons pouvoir créer une bibliothèque avec des livres
- Créer l'objet livre
  - Caractéristiques
    - Id [Long]
    - Titre [String]
    - Auteur [String]
    - Nombre de page [int]
    - Genre littéraire [enum]
- Définir le mapping avec la base de données
  - @Entity
  - @Id
  - @GeneratedValue
  - @Column
  - @Enumerated(EnumType.STRING)
- Mettre en relation la bibliothèque avec les livres
  - @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
  - @JoinColumn(name="LIBRARY_ID", referencedColumnName = "ID")

Amélioration de la tolérance au changement

- L'objet Library :
  - Représente notre objet métier ainsi que sa logique (règles métier)
  - Mapping avec le monde extérieur (REST / JSON)
  - Mapping avec la base de données (JPA)

Que se passe-t-il si nous modifions le modèle Library ?

- Impact direct sur l'IHM et sur la base de données

Solution ?

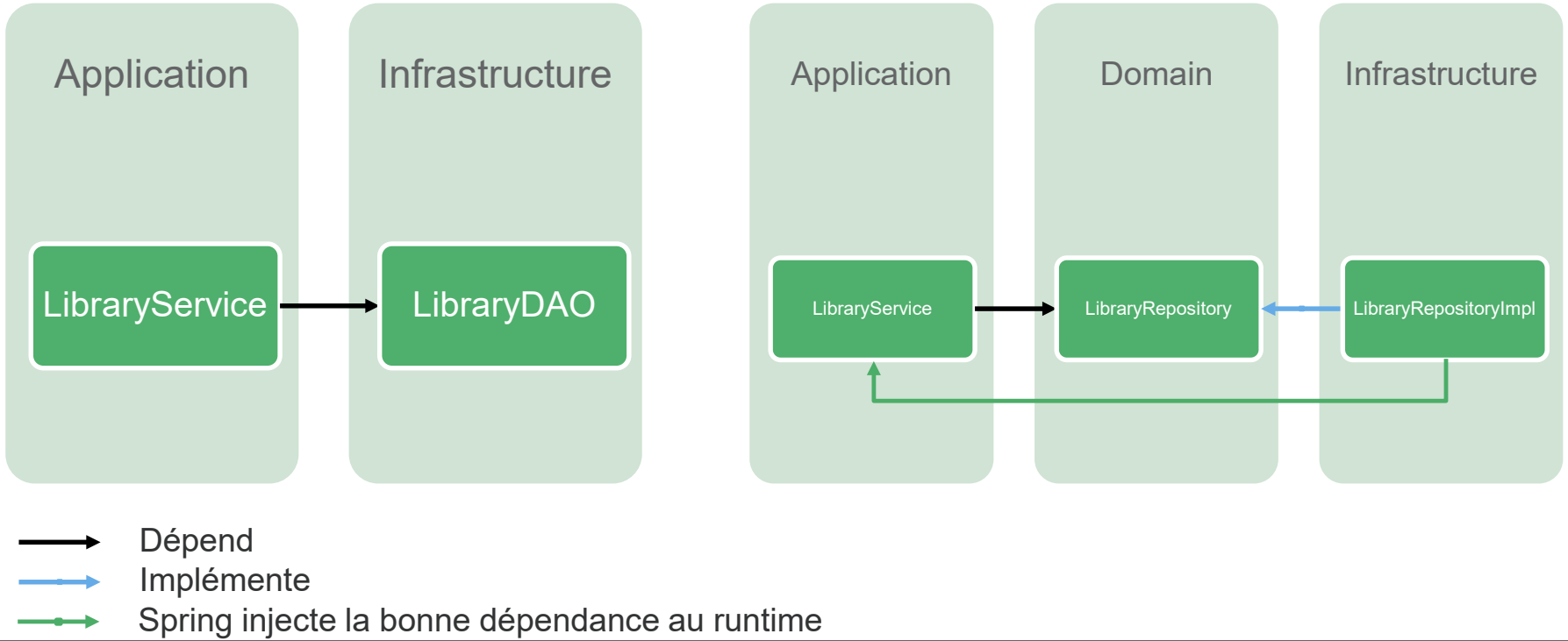
- L'idée est d'isoler l'objet métier Library et de créer deux nouveaux objets
  - Un objet LibraryDTO (représentant les données de l'IHM) (tp-spring-4)
  - Un objet LibraryJPA (représentant les données à persister en base) (tp-spring-5)

Mapping IHM (tp-spring-4)

- Créer un objet LibraryDTO qui correspond aux données entrante reçu de l'IHM
  - Utiliser l'annotation JsonProperty pour le mapping JSON vers un objet JAVA
  - Veillez à respecter les relations entre les objets pour une meilleure lisibilité
- Créer une classe permettant de transformer les données de l'IHM vers le domaine et du domaine vers l'IHM (LibraryAdapter)

Isolation du domaine – Inversion des dépendances (tp-spring-5)

- Pour isoler complétement le domaine et le rendre agnostique de toute technologie sous-jacente
  - Inversion des dépendances
  - Détacher le mapping de l'entité métier avec la base de données
- Inversion des dépendances



Validation (tp-spring-6)

- Nous voulons vérifier qu'un directeur est bien présent
- Où mettre le code associé à cette règle ?

- Plusieurs possibilités
  - Au niveau des adapteurs avec la vérification des contraintes
    - On utilise les annotations de javax.validation.constraints.* et @Valid dans la signature des APIs
    - Permet d'effectuer les contrôles de surface simple
    - Ne permet pas de protéger le domaine, on peut toujours créer une bibliothèque sans contrôle
  - Au niveau du domaine, on effectue les différents contrôles souhaités
    - Le domaine est ainsi protégé de l'extérieur

Let’s finish ! (tp-spring-7)

- DDD références pratiques
  - @DDD.Entity
  - @DDD.ValueObject
  - @DDD.Repository
  - @DDD.RepositoryImpl
  - @DDD.ApplicationService
- Egalité entre les entités
- Egalité entre les objets valeurs
- Test unitaire pour vérifier ces égalités