25/04/2023
Version 4

SPRING

JÉRÉMY PERROUAULT

SPRING ASPECT

INTRODUCTION À SPRING

PRÉSENTATION AOP

- **A**spect-**O**riented **P**rogramming
- **POO** est limité
  - On trouve souvent du code technique dans du code métier
    - Journalisation, sécurité, transaction, …
  - Ce code technique est dit « préoccupation transversale »
  - La maintenance et la réutilisabilité des composants s'en trouve diminuée

PRÉSENTATION AOP

- **AOP** permet de séparer le code métier et le code technique
- C'est une surcouche à **POO**
- Un `Aspect` correspond donc une préoccupation transversale

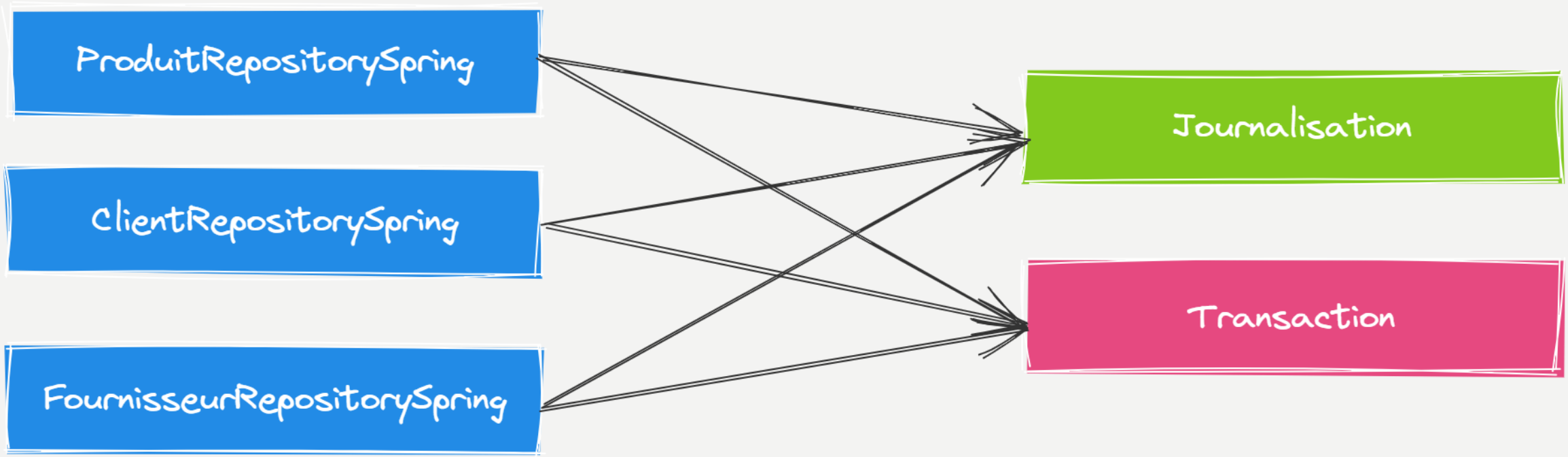
PRÉSENTATION AOP

- Intercepter les méthodes métier et appliquer un aspect associé
  - Intercepter l'ajout d'un utilisateur en base de données pour journaliser cette action

- **AOP** fonctionne grâce à un « tisseur d'aspect »
  - Dans le cas de **Spring**, il s'agit de **Spring AOP**
  - **Spring AOP** s'appuie sur **AspectJ**

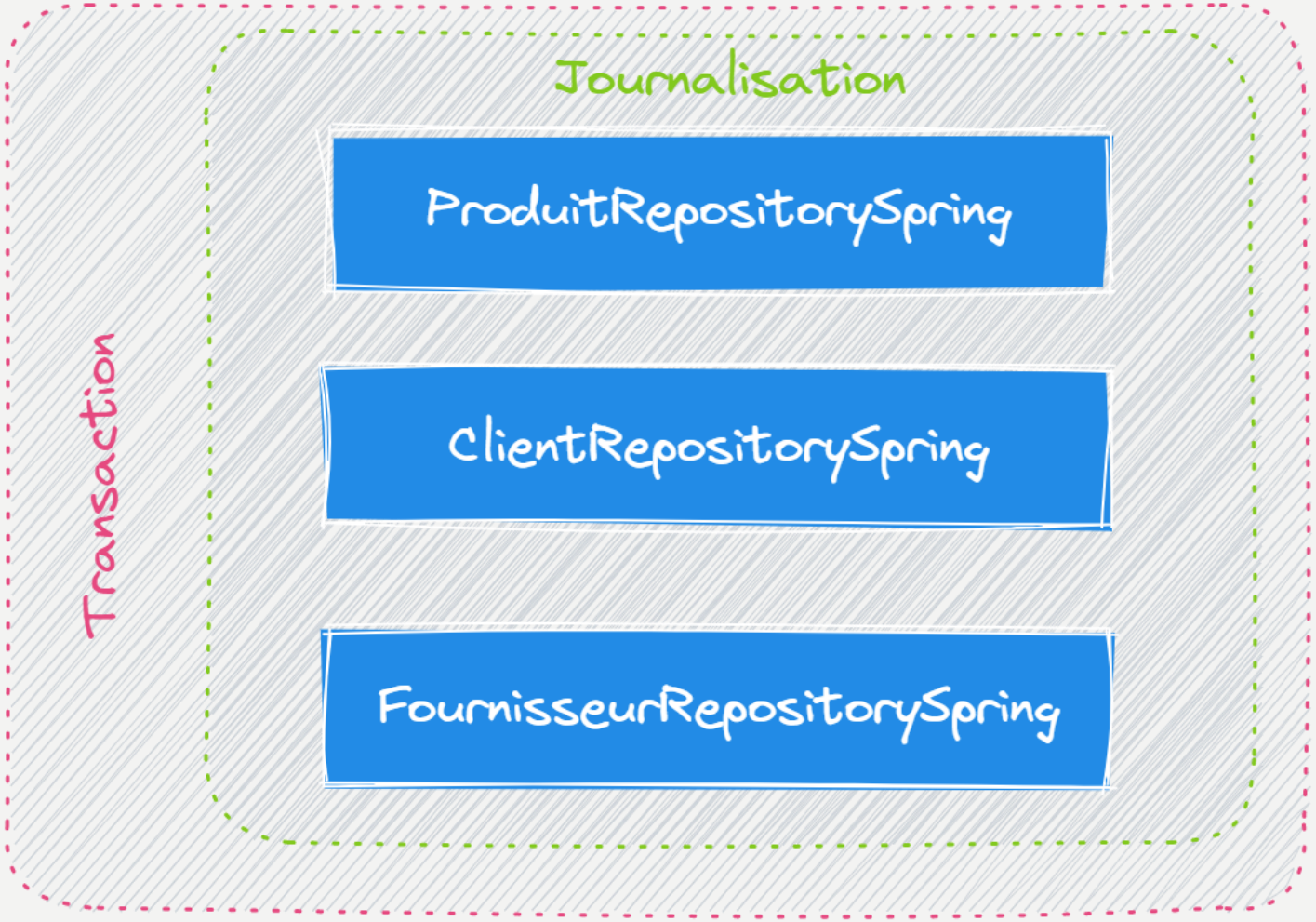
PRÉSENTATION AOP

Sans AOP



PRÉSENTATION AOP

Avec AOP



PRÉSENTATION AOP

- Sans **AOP**

```
public class ProduitRepository {
  public Produit save(Produit produit) {
    log.debug("On sauvegarde un produit ...");
    em.getTransaction().begin();
    em.persist(produit);
    em.getTransaction().commit();
    return produit;
  }
}
```

PRÉSENTATION AOP

- Avec **AOP**

```
public class ProduitRepositorySpring {
  public Produit save(Produit produit) {
    em.persist(produit);
    return produit;
  }
}
```

La méthode **debug** de Log va écouter
la méthode **save** de **ProduitRepositorySpring**
et effectuer son action soit avant, soit après

Un autre objet
écoutera la méthode **save** de **ProduitRepositorySpring**
Il démarrera la transaction avant son exécution
Il commitera la transaction après son exécution

PRÉSENTATION AOP

- Attention, Spring **AOP** ne fonctionne qu'avec des **beans Spring**
- Dépendances
  - **spring-aop**
  - **aspectjrt**
  - **aspectjweaver**

VOCABULAIRE

AOP	Définition
Aspect	Service, préoccupation transversale
Point de jonction (JoinPoint)	Méthode spécifique (Expression) pour laquelle il est possible d'insérer un greffon (avant ou après la méthode, pas pendant)
Coupe (Pointcut)	Ensemble de points de jonction
Greffon (Advice)	Méthode qui sera activée à un certain point d'exécution, précisé par un point de jonction
Cible (Target)	Objet sur lequel appliquer Aspect
Tissage	Application d'un aspect à une cible

VOCABULAIRE

Greffons	Définition
Before	Exécution avant le point de jonction
After	Exécution après le point de jonction
After-returning	Exécution après, si succès
After-throwing	Exécution après, si échec (Exception levée)
Around	Exécution autour du point de jonction

POINT DE JONCTION

- Les expressions d'un point de jonction

```
execution([<Scope>] <TypeRetour> <NomClasse>.<Methode> ([<Args>]) [throws <TypeException>])
```

  - Toutes les méthodes publiques de `ProduitRepositorySpring`

```
execution(public * fr.formation.repo.ProduitRepositorySpring.*(..))
```

  - N'importe quelle méthode « find », publique ou privée

```
execution(* *.find (..))
```

POINT DE JONCTION

- Pour une expression
  - `execution` — Exécution d'une méthode d'une classe
  - `within(<package>)` — Exécution dans un package
  - `within(@<annotation> *)` — Exécution d'une méthode dont la classe est annotée
  - `@within(<annotation>)` — Exécution d'une méthode annotée
  - `@annotation(<annotation>)` — Exécution d'une méthode annotée

- Dans une expression
  - `*` — N'importe quoi, un seul élément
  - `..` — N'importe quoi, de 0 à plusieurs éléments
- Dans une expression, on peut associer plusieurs points de jonction
  - `&&`
  - `||`

CONFIGURATION

CONFIGURATION DE SPRING

CONFIGURATION

- Activer la configuration `Aspect` par annotation (sur la classe de configuration)

```
@EnableAspectJAutoProxy
```

CONFIGURATION

- Annoter les classes et les méthodes

Annotation	Description
@Aspect	Configuration de la classe comme Aspect
@Pointcut	Méthode exécutée au point de jonction
@Before	Méthode exécutée avant
@After	Méthode exécutée après
@AfterReturning	Méthode exécutée après, si succès
@AfterThrowing	Méthode exécutée après, si échec
@Around	Méthode exécuter avant et/ou après, ou remplace

CONFIGURATION

```java
@Component
@Aspect
public class MonAspect {
  @Before("execution(* fr.formation.repo.*.save())")
  public void beforeSave() {
    System.out.println("Un save va être appelé !!");
  }

  @AfterReturning(pointcut = "execution(* fr.formation.repo.*.save(..))", returning = "result")
  public void afterReturningSave(Object result) {
    System.out.println("Le save a retourné : " + result);
  }
}
```

CONFIGURATION

```java
@Component
@Aspect
public class MonAspect {
  @Pointcut("execution(* fr.formation.repo.*.save(..))")
  public void intercept() { }

  @Before("intercept()")
  public void interceptToString() {
    System.out.println("Un save va être appelé !!");
  }

  @AfterReturning(pointcut = "intercept()", returning = "result")
  public void interceptToStringReturning(Object result) {
    System.out.println("Le save a retourné : " + result);
  }
}
```

CONFIGURATION

```
@Before("intercept()")
public void interceptToString(JoinPoint joinPoint) {
  if (joinPoint.getTarget() instanceof ProduitRepository) {
    System.out.println("C'est un produit qui va être sauvegardé");
  }
  System.out.println("Un save va être appelé !!");
}
```

CONFIGURATION

```java
@Around("execution(* *.*(..))")
public void aroundAll(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
  //Avant la méthode

  //Exécuter la méthode
  //Donc ne pas indiquer cette ligne n'exécutera pas la méthode
  proceedingJoinPoint.proceed();

  //Après la méthode
}
```

INJECTER LES ARGUMENTS

- Il est possible d'injecter les arguments d'une méthode dans un greffon
  - Préciser la liste des arguments dans le point de jonction
  - Ajouter le paramètre au greffon

```
@Around("execution(* fr.formation.repo.*.save(..)) && args(entity)")
public void aroundSave(ProceedingJoinPoint proceedingJoinPoint, Object entity) throws Throwable {
  //... partie est utilisable ici ...
}
```

INJECTER LES ARGUMENTS

- Il est possible d'injecter la cible d'une méthode dans un greffon
  - Préciser la cible dans le point de jonction
  - Ajouter le paramètre au greffon

```
@Around("execution(* fr.formation.repo.*.save(..)) && target(repo)")
public void aroundSave(ProceedingJoinPoint proceedingJoinPoint, Object repo) throws Throwable {
  //... partie est utilisable ici ...
}
```

EXERCICE

- Exécuter une méthode
  - Après qu'un `Guitariste` ait exécuté sa méthode `jouer`
  - La méthode fait un simple `System.out`

EXERCICE

- Créer une méthode qui
  - Retourne une chaine de caractères
  - Est capable de lever une `Exception`
- Configurer l'aspect qui intercepte sa valeur de retour
- Configurer l'aspect qui intercepte son `Exception`
- Tester

EXERCICE

- Créer une annotation personnalisée
- Créer une méthode annotée dans `Guitariste`
- Créer une méthode qui écoute cette annotation

EXERCICE

- Créer une méthode avec un paramètre
- Intercepter la méthode avec un `@Around`
- Récupérer le paramètre et sa valeur
  - Tester avec un `System.out` par exemple

EXERCICE

- Mise en musique (avec 2 musiciens : un pianiste et un trompettiste)
  - Le premier musicien joue un air
    - Avant, le public s'installe
    - Après, le public applaudit
    - S'il y a une fausse note, le public siffle
    - Lorsque le premier musicien a terminé, le public demande au deuxième de jouer : il joue alors

- Implémenter cette histoire avec **Spring AOP**