Séance 9 – Rendre l’application dynamique

Positionnement de la séance

La séance 9 intervient **après la structuration complète de l’architecture** :

- métier isolé,
- instanciation centralisée,
- accès unifié via Facade,
- modules et responsabilités clairs.

L’objectif n’est plus de structurer, mais de **rendre le comportement du système flexible et évolutif**, sans modifier le cœur du métier à chaque nouvelle règle.

Cette séance prépare directement la **soutenance de la séance 10**, où vous devrez justifier vos choix architecturaux et montrer que votre système peut évoluer sans être refondu.

Objectif projet

Rendre l’application **dynamique**, c’est-à-dire capable de :

- faire varier ses règles métiers,
- enrichir son comportement,
- s’adapter à des contextes différents,

sans modifier les classes métier existantes.

Problème architectural de départ

Dans beaucoup de projets, les règles métiers sont :

- codées en dur,
- dispersées dans plusieurs services,
- difficiles à faire évoluer.

Conséquences :

- chaque nouvelle règle implique une modification du code existant,
- risque de régression,
- difficulté à tester des variantes de comportement.

La séance 9 répond à cette question :

Comment faire évoluer les règles métiers sans modifier le cœur du système ?

1. Strategy – Variabiliser les règles métiers

Intention

Le pattern Strategy permet de **déléguer une règle métier à un composant interchangeable**.

Au lieu d'avoir :

- une règle codée dans un service,

on obtient :

- plusieurs stratégies possibles,
- sélectionnées selon le contexte.

Exemple conceptuel

Contexte générique : calcul, validation, éligibilité, tarification.

Au lieu de :

- `if / else` multiples,

on définit :

- une interface de règle,
- plusieurs implémentations.

Le service métier :

- applique une stratégie,
- sans connaître son implémentation concrète.

Le métier reste stable, la règle devient variable.

Ce que Strategy apporte architecturalement

- ouverture à l'évolution sans modification du service,
- meilleure testabilité,
- règles métiers explicites et isolées.

2. Decorator – Enrichissement dynamique du comportement

Intention

Le pattern Decorator permet **d'ajouter un comportement à un objet existant**, sans modifier sa classe.

Il est utilisé lorsque :

- le comportement doit être optionnel,
- combinable,
- activable ou non selon le contexte.

Exemple conceptuel

Un traitement métier de base peut être enrichi par :

- du logging,
- de la sécurité,
- de la validation supplémentaire,
- de la traçabilité.

Chaque enrichissement :

- enveloppe le comportement précédent,
- sans casser l'existant.

On empile des responsabilités, au lieu de multiplier les classes.

Ce que Decorator apporte architecturalement

- respect du principe Open/Closed,
- comportement modulaire,
- suppression des classes "fourre-tout".

3. Moteur de règles métier (livrable)

Intention du livrable

Le livrable de la séance 9 n’est **pas un moteur technique complexe**.

Il s’agit d’une **structure conceptuelle claire** montrant :

- où sont définies les règles,
- comment elles sont sélectionnées,
- comment elles peuvent être enrichies.

Attendus minimaux

Votre moteur de règles doit montrer :

- au moins une règle métier implémentée via Strategy,
- au moins un enrichissement via Decorator,
- une intégration cohérente avec votre Facade et vos services.

Aucun code complexe n’est attendu :

- pseudo-code,
- schéma,
- description structurée.

4. Lien direct avec la soutenance (séance 10)

Cette séance vous prépare explicitement à défendre votre architecture.

À l’oral, vous devrez être capables d’expliquer :

- pourquoi vos règles métiers sont isolées,
- comment vous les faites évoluer sans refactorisation,
- en quoi Strategy et Decorator sont des choix raisonnés.

Ce sont **des arguments de soutenance**, pas juste des patterns.

Livrables attendus

Un document contenant :

- la description du moteur de règles métier,
- l’usage de Strategy,
- l’usage de Decorator,
- un ou plusieurs schémas explicatifs,
- une justification claire des choix.

Message clé à retenir

> Une architecture mature ne code pas ses règles en dur.
> Elle les rend configurables, remplaçables et composables.

La séance 9 marque le passage d’une architecture structurée à une architecture **vivante et évolutive**.