# 3 – Système de notification utilisateur

## PALIER 3 — Polymorphisme & Interfaces (Analyse orientée objets)

---

🎯

## Objectif global

Modéliser un système simple capable d'**envoyer des notifications** à un utilisateur,

par **plusieurs moyens différents** (tu choisiras lesquels).

L'utilisateur ne doit **pas dépendre directement** du moyen de notification.

Tu dois réfléchir à une conception qui **sépare les rôles** et **minimise les dépendances**.

---

🧠

## Contexte métier

Une application interne doit informer ses utilisateurs à différents moments :

- Lorsqu'un nouvel utilisateur est créé,
- Lorsqu'une action spécifique est réussie ou échoue,
- Lorsqu'un rappel ou une alerte doit être envoyée.

Actuellement, chaque développeur ajoute son propre "print message" dans le code, ce qui crée du désordre et du code dupliqué.

Ton rôle est de proposer une conception **orientée objets propre et extensible** pour gérer ces notifications.

---

🧩

## Attentes fonctionnelles

1. Le système doit pouvoir **envoyer un message** à un utilisateur.
2. Il doit être **possible de changer le canal de notification** (par exemple passer d'un envoi "par email" à un envoi "par autre chose") **sans modifier le code de l'utilisateur**.
3. On doit pouvoir **ajouter un nouveau type de notification** (par exemple via une messagerie instantanée) **sans casser le reste du système**.
4. L'utilisateur doit **conserver une trace minimale de son identité** (nom, adresse, ou ce que tu juges pertinent).

---

## Contraintes de conception

- Ton modèle doit exprimer **une dépendance faible** entre les entités : l'expéditeur du message ne doit **pas connaître les détails** du canal d'envoi.
- Tu dois pouvoir **étendre le système** sans modifier les classes existantes (principe *Open/Closed*).
- Certains objets devront **avoir une relation de type "utilise" ou "dépend de"** d'autres objets — à toi de les identifier.
- Pense à la **communication entre objets**, pas seulement à leurs données.

---

## Travail demandé

1. **Analyse**
   - Lis le contexte et identifie les **objets métiers** (exemples : acteur principal, rôles secondaires, canaux, messages, etc.).
   - Décris leur **responsabilité** principale (une phrase par objet).
   - Évalue leurs **relations** (composition, agrégation, association simple ?).
2. **Conception UML**
   - Dessine un **diagramme de classes** représentant ta vision du système.
   - Indique les attributs et méthodes essentiels, **mais sans entrer dans le détail du code**.
   - Représente clairement les **liens d'héritage ou d'implémentation** si tu en vois la nécessité.

- Représente les **interactions clés** dans un **diagramme de séquence** ou un **cas d'usage**.

3. **Préparation au code**
   - Identifie les points du système où le **polymorphisme** pourrait intervenir.
   - Liste les classes qui, selon toi, devront **changer de comportement selon leur type**.
   - Réfléchis à **comment tu pourrais injecter** ce comportement dans les objets qui en ont besoin.

---

💬

## Questions de réflexion (à faire avant tout UML)

1. Quels sont les **acteurs** du scénario ?
2. Qui **initie** la notification ?
3. Qui **exécute** l'envoi concret du message ?
4. Qui **reçoit** le message ?
5. Si demain on ajoute un nouveau canal d'envoi (ex: Discord, Slack, push mobile), que faut-il changer dans ton modèle ?
6. Où pourrais-tu exploiter **le polymorphisme** pour éviter de dupliquer du code ?

---

⚡

## Variante bonus (interactions multiples)

Imagine que ton application gère **plusieurs utilisateurs**.

Elle doit pouvoir **envoyer un même message à tous**, mais **via leur canal préféré**.

> → Ajoute alors un nouvel objet "coordonnateur" responsable d'envoyer les notifications en boucle.

> Tu choisiras toi-même si ce rôle appartient à une classe déjà existante ou à une nouvelle entité.

---

🧩

## Livrable attendu

- Un **diagramme UML clair et cohérent**, montrant :
  - Les classes principales que tu as identifiées.
  - Les relations entre elles (associations, implémentations, héritage).
  - Le sens des dépendances.
- Une **brève justification écrite** de ton choix de conception (2–3 phrases).

---

🧠

## Objectif de cette séance

> Être capable de **penser comme un concepteur**, pas comme un codeur.

> Identifier et relier les “acteurs objets” avant de toucher à la syntaxe Java.