Bonne question ! Pour un **OrderService**, au-delà des méthodes CRUD classiques (create, read, update, delete), tu peux intégrer des méthodes plus métier qui correspondent aux besoins spécifiques du système. En voici quelques-unes qui pourraient enrichir ton service :

📄 Méthodes métier utiles pour un OrderService

✅ Validation & traitement

- `validateOrder(Order order)` – Vérifie que toutes les conditions sont réunies pour passer la commande (stock dispo, client valide, etc.).
- `processOrder(Long orderId)` – Lance le processus de traitement de la commande : paiement, réservation de stock, etc.
- `confirmOrder(Long orderId)` – Confirme la commande après traitement (état : "Confirmé").

💳 Paiement

- `payOrder(Long orderId, PaymentDetails paymentDetails)` – Gère le paiement de la commande.
- `refundOrder(Long orderId)` – Traite le remboursement (partiel ou total).

🚚 Livraison

- `shipOrder(Long orderId)` – Lance l'expédition.
- `trackOrder(Long orderId)` – Retourne l'état d'avancement de la livraison.
- `updateDeliveryStatus(Long orderId, String status)` – Permet de mettre à jour manuellement le statut de la livraison.

🔄 Gestion du cycle de vie

- `cancelOrder(Long orderId)` – Permet d'annuler une commande avant son expédition.
- `archiveOrder(Long orderId)` – Archive une commande finalisée.

📊 Analyse / Suivi

- `getOrdersByStatus(String status)` – Retourne toutes les commandes selon un état (en cours, livrée, annulée...).
- `getOrdersByCustomer(Long customerId)` – Historique des commandes d'un client.
- `calculateOrderTotal(Long orderId)` – Calcule le total (avec ou sans TVA, frais de port...).

Si tu veux encore plus de granularité, je peux t'aider à organiser ton service en plusieurs sous-services spécialisés (par ex. : `PaymentService`, `ShippingService`, etc.).

Tu veux que je te montre un exemple de classe `OrderService` avec ces méthodes ?