TP - Création d'un générateur de DAO personnalisé

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-11-01

Table des matières

0.1 Objectifs : 1
0.2 Contexte : 1
0.3 Extensions possibles : 2

0.1 Objectifs :

- **Comprendre** le principe des processeurs d'annotations et leur utilisation dans la génération de code.
- **Maîtriser** les bases de Jakarta Persistence pour la persistance d'objets.
- **Développer** un outil de productivité permettant de générer automatiquement du code DAO.
- **Apprendre** à utiliser un moteur de templates pour personnaliser la génération de code.

0.2 Contexte :

La programmation de DAO (Data Access Object) est une tâche récurrente dans le développement d'applications Java. Ces classes permettent d'isoler la logique d'accès à la base de données du reste de l'application. Pour gagner en productivité, il est intéressant d'automatiser la génération de ces classes.

0. **Définition d'une interface de DAO générique:**

   Par exemple (vous pouvez ajouter un paramètre pour le type de l'Id),

```java
public interface GenericDao<T> {
    T save(T entity);
    T findById(Long id);
    List<T> findAll();
```

```
    void delete(T entity);
}
```

1. **Définition d'une annotation `@DAO`:**
   - Définir une annotation personnalisée pour marquer les classes représentant des entités à persister.
   - Ajouter des attributs à l'annotation pour spécifier le nom de la table, la clé primaire, etc.
2. **Création d'un processeur d'annotations:**
   - Implémenter un processeur qui analyse les classes annotées `@DAO`.
   - Pour chaque classe annotée, générer une interface DAO avec les méthodes CRUD de base (create, read, update, delete).
3. **Intégration de Jakarta Persistence:**
   - Utiliser les annotations de Jakarta Persistence (e.g., `@Id`, `@Column`, `@OneToMany`, etc.) pour définir les propriétés des entités.
   - Générer les requêtes JPQL correspondantes pour les méthodes CRUD.
4. **(optionnel) Utilisation d'un moteur de templates:**
   - Choisir un moteur de templates (StringTemplate, FreeMarker, Thymeleaf, etc.) pour générer le code Java du DAO.
   - Créer des templates pour définir la structure du code généré.
   - Personnaliser les templates pour ajouter des fonctionnalités spécifiques (par exemple, des méthodes de recherche personnalisées).
5. **Test unitaire:**
   - Écrire des tests unitaires pour vérifier le bon fonctionnement du générateur de DAO.
   - Tester la génération du code, la compilation du code généré, et l'exécution des méthodes du DAO.

0.3 Extensions possibles :

- **Personnalisation avancée:**
  - Permettre de définir des templates personnalisés pour chaque méthode du DAO.
  - Ajouter des options de configuration pour le générateur.
- **Gestion des relations:**
  - Gérer les relations entre les entités (One-to-One, One-to-Many, Many-to-Many).
  - Générer les méthodes de navigation en conséquence.

Conseils:

- **Démarrer par un cas simple:** Commencez par générer des DAO pour des entités simples sans relations.