Génie logiciel CB2
TP4 Noté et individuel
Extension du projet de génération

Jean-Claude Royer

12 janvier 2011

1 Le projet existant

Le projet fourni comme sujet est la solution du TP3 précédent, y compris la question subsidiaire permettant d'imbriquer des listes. Il va s'agir d'étendre le modèle UML puis de modifier le projet en conséquence. Voici pour rappel le diagramme des classes, le fichier `tp3.zargo` vous est également fourni.

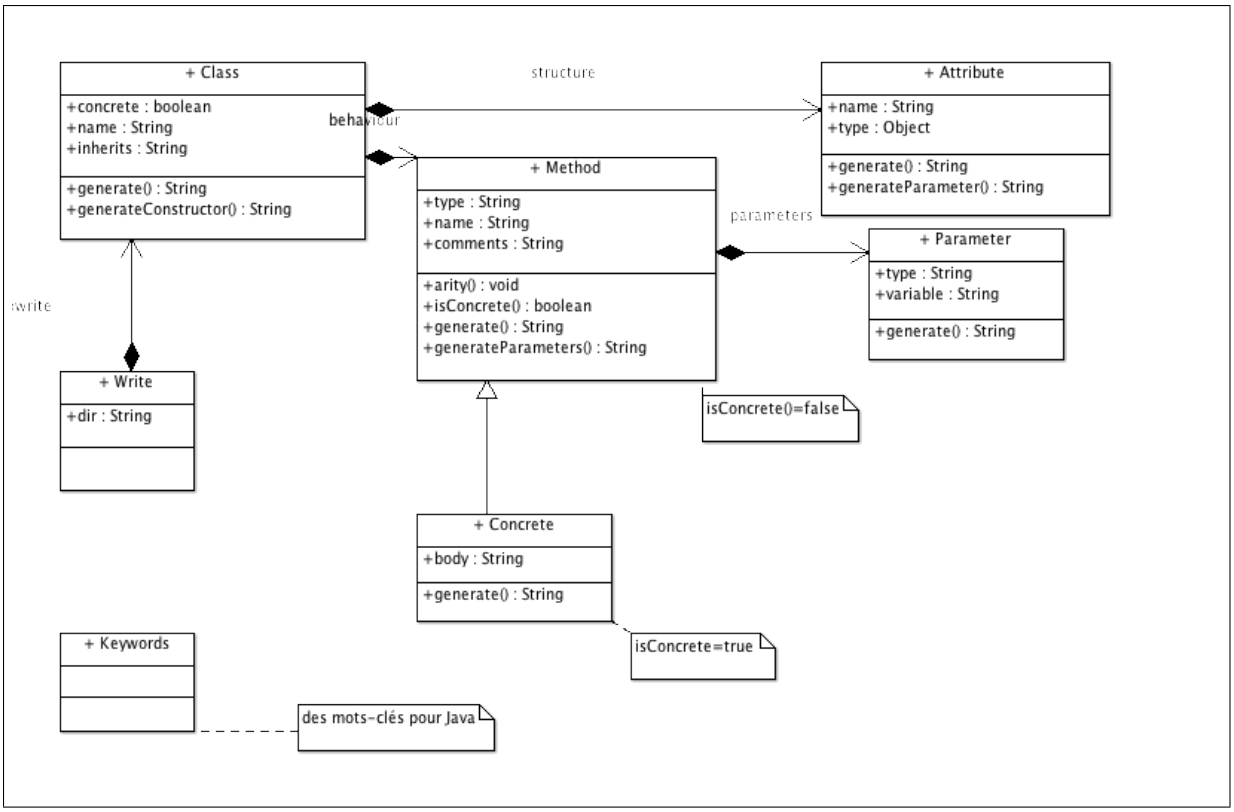

FIGURE 1 – Diagramme des classes du TP3

2 Extension du projet

2.1 Le modèle

Lancer `ArgoUML` avec le .zargo et résoudre les points suivants.

1. Placer dans le diagramme une méthode `generateCopyConstructor`.
2. Enrichir le diagramme pour qu'un attribut puisse avoir un `scope` et un `gener`. Le `scope` indique la visibilité qui sera `private`, `protected` ou `public`. Le `gener` peux prendre quatre valeurs : `no`, `get`, `set`, `getset`.
3. Ajouter aussi le `scope` pour les méthodes.
4. Placer dans le diagramme une méthode `generateEquals`.

Dans votre diagramme vous devez placer les champs, les méthodes et leurs redéfinitions.

2.2 L'implémentation

Reprendre le projet Eclipse du TP3 qui vous est fourni comme sujet.

1. Définir la génération d'un constructeur de recopie superficielle. Si la classe n'est pas abstraite et si elle a des attributs le générateur doit créer un nouveau constructeur. Celui-ci prends un argument du même type que la classe et crée une nouvelle instance en initialisant les champs avec les valeurs correspondantes dans l'objet argument du constructeur de recopie. Dans cette version nous utiliserons un appel au constructeur complet. Le principe est illustré sur l'exemple de la classe `NotEmpty` :

   ```
   public NotEmpty(NotEmpty other) {
          this(other.head, other.tail);
   }
   ```

2. Prendre en compte la génération du scope pour les attributs et les méthodes.
3. Implémenter la sémantique du `gener` pour les attributs qui ne sont pas `public`. Le principe est le suivant : `no` signifie rien à générer, `get` signifie générer un getter publique, `set` générer un setter publique et `getset` les deux. Bien sûr il faudra suivre les conventions de nommage en usage en Java, *i.e.* générer les noms des accesseurs à partir des noms des attributs et en majusculisant la première lettre.
4. Ecrire une méthode de génération pour `equals`, une égalité profonde. Le format général est illustré avec l'exemple de la classe `NotEmpty` :

   ```
   public boolean equals(Object other) {
          return this.head.equals(((NotEmpty) other).head) &&
                 this.tail.equals(((NotEmpty) other).tail);
   }
   ```

   Toutefois il convient d'ajouter `(other != null) && (other instanceof NotEmpty)`. Reste un dernier problème, quel est-il ? comment le résoudre ?

3 Travail à rendre

Archive .zip résultant **d'un export du projet** [1] à déposer sur campus et contenant :

1. IF (!zipProjetCorrect()) THEN note = 0 ;

1. le code source du projet avec la même structure que le projet initial
2. le diagramme UML étendu, nommé diagrammeV2.png et placé dans `ressources`
3. un fichier `COMMENTS` contenant vos remarques au sujet de votre travail et placé dans `ressources`