Génie logiciel CB2
TP1 individuel
implémentation du patron composite

Jean-Claude Royer

12 décembre 2010

Vous avez vu le patron composite en cours, le sujet de ce TP est d'en faire une implémentation. On suppose la description de ce patron pour des listes de chaîne de caractères donnée dans la figure 1. Dans cette figure il y a trois classes, la classe `ListeDeMots` est abstraite et toutes ses méthodes le sont également. La classe `ListeNonVideDeMots` possèdent trois méthodes concrètes.

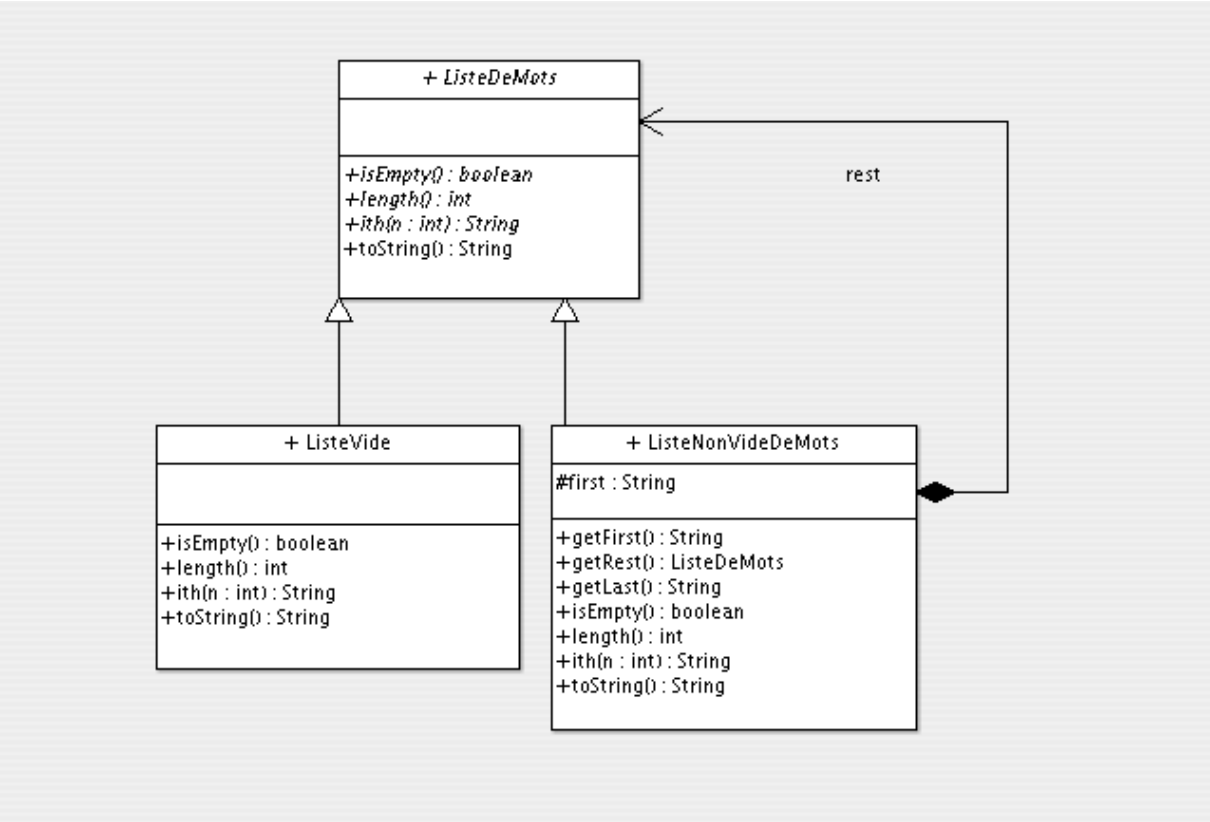

FIGURE 1 – Diagramme des classes pour les listes

1 Utilisation d'ArgoUML

Dans un premier temps vous allez utilisez ArgoUML pour dessiner ce diagramme de classe. En principe cet outil doit déjà avoir été installé par vos soins et vous aurez lu la documentation dans le polycopié. Une documentation est disponible avec l'installation et dans le répertoire OUTILS sur campus. Cette partie d'apprentissage peut être un peu pénible, n'y passer pas trop de temps, une heure tout au plus.

1. Lancer ArgoUML.
2. Créer un nouveau diagramme des classes.
3. Penser à sauver votre projet.
4. Générer le code des classes du diagramme.

Noter que la cardinalité de la composition est 1 dans ce diagramme. Rappelez vous que pour UML une composition ou un attribut c'est la même chose. Plus votre diagramme sera précis, complet et cohérent plus la génération de code sera efficace et donc les modifications manuelles du code plus limitées. Remarque : pour voir les visibilités des opérations et attributs, ainsi que d'autres caractéristiques, ouvrir le panneau des propriétés et aller à l'onglet notations.

2 Compléter le code

Dans un second temps vous allez compléter le code produit et vous le testerez. Vous devez donc passer sous Eclipse et créer un projet `from existing source code`. L'utilisation de l'option `add unimplemented methods` peut vous facilitez la vie. Vous devez implémenter les méthodes suivantes, évidemment inspirez vous du cours et utilisez un peu de réflexion.

1. `isEmpty()` : teste si l'objet représente la liste vide ou non.
2. `length()` : calcule la longueur de la liste.
3. `toString()` : fabrique une string avec les mots séparés par des blancs.
4. `ith(int n)` : si `1 <= n <= length()` alors retourne le n^{e} mot de la liste sinon cette fonction n'est pas définie.
5. `getFirst()` : produit le premier mot.
6. `getRest()` : produit le reste de la liste sans le premier mot.
7. `getLast()` : produit le dernier mot.

Pour gérer le cas où une fonction n'est pas définie vous pouvez rendre `null`. Toutefois l'utilisation des exceptions est fortement conseillée. Faites des tests : créer des listes et appliquer leur les opérations précédentes. En particulier tester l'application des trois dernières va demander un `cast` que vous devez savoir réaliser.

Il y a généralement deux façons de programmer avec un langage, soit avec une approche dite `purement fonctionnelle`, soit une approche `impérative`. Les deux approches permettent les mêmes choses mais dans un style différent plus ou moins élégant ou efficace suivant les cas. Dans l'approche fonctionnelle pure on travaille avec des valeurs et systématiquement on calcule ou on construit de nouveaux objets ou de nouvelles valeurs. Dans l'approche impérative, qui vous est plus familière, on modifie l'état des objets. Dans ce travail il est beaucoup plus facile d'implémenter les méthodes proposées dans un style fonctionnel.

3 Étendre votre code

Une fois l'étape précédente réalisée vous êtes en mesure de faire des fonctions un peu plus complexes. Je vous propose donc les étapes suivantes à réaliser :

1. Faire apparaître et justifier l'utilisation du patron singleton dans votre implémentation.
2. `append: ListeDeMots ListeDeMots -> ListeDeMots` : concatène simplement deux listes en une seule. Par exemple soient `l1 = "a" "b" "c"` et `l2 = "titi" "toto"` alors `l1.append(l2)` sera la liste `"a" "b" "c" "titi" "toto"`.
3. `putlast: ListeDeMots String -> ListeDeMots` : ajoute le mot à la fin de la liste. Par exemple `l1.putlast("roro")` sera `"a" "b" "c" "roro"`.
4. `reverse: ListeDeMots -> ListeDeMots` : produit une nouvelle liste inversée. Par exemple `l1.reverse()` produira une nouvelle liste représentant `"c" "b" "a"`.

Implémenter de façon récursive les fonctions suivantes : `append`, `putlast`, `reverse`.

4 Le travail à rendre

Vous devez produire un projet Java bien structuré et contenant le résultat de votre conception. Nous espérons que les conventions d'écriture (les principales) seront vérifiées et que la javadoc sera générée. Vous fournirez une archive `.zip` de votre projet contenant les sources que vous déposerez sur campus. Vérifiez votre archive est souvent une bonne précaution, certains outils comme `emacs`, `winzip` sont capables de la lire, mais vous pouvez aussi essayer de recréer votre projet (**attention** à ne pas écraser celui existant) à partir de l'archive.