Génie logiciel CB2
TP0
Un petit projet

Jean-Claude Royer

9 décembre 2010

Ce premier TP ne sera pas noté mais vous permettra de pendre contact avec des outils utiles dans l'environnement d'Eclipse. Les objectifs sont :

- Créer un projet à partir d'une archive.
- Se remettre dans le bain d'Eclipse et créer un petit projet standard.
- Utiliser la généricité de Java 5.
- Utiliser la `javadoc`.
- Faire une archive jar d'un projet avec les sources et la javadoc.
- Recréer un projet depuis une archive avec les sources

1 Le sujet

Il s'agit de réaliser un programme qui lit un texte en entrée et produit un index en sortie. Le programme doit réaliser le calcul d'un index pour un texte donné et à partir d'une liste de mots (pour les curieux c'est le rôle de la commande `makeindex` dans le traitement de texte LaTeX). Le texte en entrée sera constitué de chaque occurrence de mot dans le texte et de la page où il apparaît.

```
%%%%% exemple de texte
% le % débute un commentaire
% ensuite j'ai des lignes
% et pour chaque ligne: un mot puis sa page

Bidon 4
Bidon 5
Chaussure 6
Etudiant 8
CRS 12
Nicolas 4
Machin 8
Chaussure 5
Bidon 10
Nicolas 6
```

Le fichier de sortie sera l'index sous une forme simple comme ci-dessous :

```
% fichier des mots de resources/montexte
Bidon 4, 5, 10
CRS 12
Chaussure 5, 6
Etudiant 8
Machin 8
Nicolas 4, 6
```

Tous les fichiers sont supposés être des fichiers textes. Le texte en entrée sera dans un fichier suffixé `.idx` et votre programme doit créer un fichier de même nom mais avec le suffixe `.ind`. Le fichier de sortie contiendra une table des mots à référencer et pour chacun la liste des pages en ordre strictement croissant où le mot apparaît.

2 Le travail

1. La première étape consiste donc à récupérer l'archive `tp0sujet.zip` sur campus.
2. La seconde est de créer le projet initial à partir de cette archive. Rien de plus simple if suffit de faire `import > Existing Projects into Workspace` puis de choisir `select archive file`, sélectionner l'archive voulue et puis faire `finish`.
3. Vous remarquerez que la structure du projet est la suivante : `resources` contient des fichiers utiles pour l'application, `src` contient les classes de l'application et `tests` les classes principales ou de tests. Les fichiers binaires exécutables sont dans `bin` en général non visible dans le projet avec le `package explorer` [1].
4. Pour la partie programmation il faut savoir lire un fichier texte et en récupérer les mots. Une solution simple vous est fournie dans l'archive et est décrite dans la classe `Loader`. Vous pouvez d'ailleurs la tester grâce à la classe `TestLoader`. Cette classe lit le fichier d'entrée et le réécrit mais sans traitement et les mots (y compris) les nombres apparaissent les uns à la suite des autres dans le fichier produit.
5. Pour réaliser l'implémentation plusieurs approches sont possibles. L'implémentation que nous vous proposons suit le diagramme UML qui apparaît dans la figure 1. Vous devez compléter les classes dont les squelettes vous sont fournies.

2.1 `SortedListOfInteger`

La classe `SortedListOfInteger` permet de gérer les listes de pages (entiers positifs) en ordre strictement croissant. Les services à définir sont :

- `void add(Integer page)` [2] : ajoute une page dans la liste, la page est un entier strictement positif, la liste est triée en ordre strictement croissant et cet ordre doit être conservé après l'insertion.
- `String toString()` : représentation sous forme de chaîne d'une liste de pages séparées par des virgules.

1. Par contre visible avec la vue `navigator`
2. Remarque : cette méthode existe dans `Vector` si vous souhaitez en hériter faites attention à deux choses : 1) vous devez définir `boolean add(Integer page)` et 2) attention à `this` et `super`.

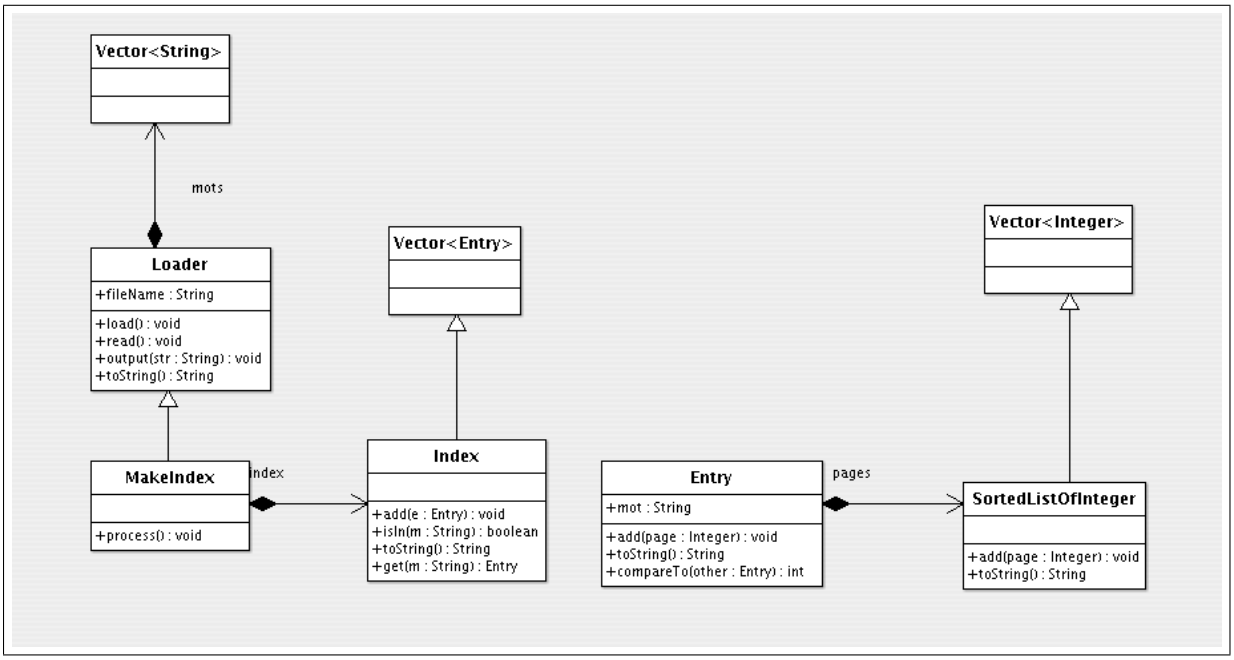

FIGURE 1 – Diagramme des classes

2.2 `Entry`

La classe `Entry` permet de définir une entrée de l'index. Les services à définir sont :

- `void add(Integer page)` : ajoute une page dans la liste de l'entrée.
- `int compareTo(Entry other)` : comparaison des mots de deux entrées, le résultat est 0 en cas d'égalité, `<` 0 si `this` est inférieur à `other` et `>` 0 sinon. L'idée est d'utiliser `compareTo` de la classe `String`.
- `String toString()` : représentation sous forme de chaîne d'une entrée.

Rappel : si vous ne savez pas ce qu'est la définition d'une classe ou d'une méthode en faisant `google: Java 1.5 String` vous devriez obtenir une réponse utile.

2.3 `Index`

La classe `Index` représente la table des index du texte, il n'est pas inutile de s'inspirer de la classe `SortedListOfInteger`. Les services à définir sont :

- `void add(Entry e)` : ajoute une entrée dans l'index, les entrées sont classées en ordre strictement croissant des mots (ordre dit lexicographique et calculable par la méthode `compareTo`).
- `boolean isIn(String m)` : teste si une entrée de mot `m` existe déjà dans l'index.
- `Entry get(String m)` : récupère l'entrée de mot `m` dans l'index et suppose donc que celle-ci existe (pré-condition `isIn(m)`).
- `String toString()` : représentation sous forme de chaîne d'un index.

2.4 `MakeIndex`

La classe `MakeIndex` permet de faire le traitement du texte initial en se basant sur la classe `Loader` et la classe `Index`. La classe `Loader` a juste pour rôle de lire le fichier d'entrée ligne par ligne et de produire

les mots sous forme d'un vecteur (`getMots()`).

- `process` : l'algorithme est simplement : lire une entrée du fichier d'entrée et la placer correctement dans l'index.

Une fois toutes ces fonctions implémentées vous devez créer une classe de test, sur le même principe que `TestLoader` et la compléter.

3 Structure du projet

Normalement votre polycopié de cours contient toutes les indications concernant les différents outils utiles durant ce TP. En cas de panne ou de problème inattendu contacter votre enseignant.

1. **Dans tous vos projets**, présent et futurs, vous devrez avoir au moins un répertoire `src` pour les sources Java et un `doc` pour la `javadoc`.
2. Vous devez générer la documentation avec `javadoc`.
3. Vous rendrez une archive `.zip` avec vos sources et la `javadoc`.

4 Addendum : questions sur les archives en vrac

En général l'opération de création d'une archive se fait très simplement mais il arrive que cela ne se passe pas bien. Comme vous aurez besoin de savoir faire cela notamment pour rendre les TPs notés, je vous conseille de passer un peu de temps là-dessus. Le polycopié du cours contient certaines des réponses aux questions ci-dessus, pour les autres faire quelques essais avec Eclipse devrait vous donnez la réponse. Pour créer une archive `.jar` de votre projet il faut : sélectionner le projet voulu, utiliser le menu contextuel "droit" et choisir "exporter" puis "Java" puis "Jar file".

1. Comment mettre les sources dans un .jar ?
   Lors de la création il faut cocher la casse "Export Java source files and ressources".
2. Comment mettre la javadoc dans un .jar ?
   Il suffit de la générer dans le projet, elle fait partie des ressources généralement exportées par défaut.
3. Est-ce que je peux faire un autre type d'archive pour les sources ? et comment ?
   Il faut procéder comme pour un ".jar" mais on choisit "General" puis "Archive file".
4. Si je fais un `import` de mon `.jar` comme une archive que se passe-t'il ?
   Si je l'importe comme une archive il va être mémorisé sous un répertoire et sûrement me causer quelques ennuis de compilation.
5. Est-ce que je peux recréer un projet sans faire une archive ? Comment ?
   Oui en utilisant la création d'un projet Java mais en cochant la casse "from existing sources".