Document de spécifications des classes constituant le TP-Héritage

Kilian Ollivier, Álvaro González

20 janvier 2015

Table des matières

1 Lexique **2**

2 Choix généraux de conception **2**

3 Spécification des classes **2**
3.1 Application . . . 2
3.1.1 Rôle de la classe . . . 2
3.1.2 Constructeur, Destructeur & opérateurs . . . 2
3.1.3 Types personnalisés . . . 2
3.1.4 Attributs . . . 2
3.1.5 Méthodes . . . 2

4 Description et justification des structures de données **2**
4.1 Nœuds & Arcs . . . 2

5 Diagramme de classes **3**

6 Description des tests **3**

1 Lexique

Termes et définitions :
–

2 Choix généraux de conception

–

3 Spécification des classes

3.1 Application

3.1.1 Rôle de la classe

C'est la classe principale du programme. Elle gère les entrées fichier et ligne de commande, ainsi que leur interprétation et déclenche les opération associées à réaliser sur la figure.

3.1.2 Constructeur, Destructeur & opérateurs

3.1.3 Types personnalisés

Les types de la classe Application :

```
    struct CmdType
{   string toString;
    int nbArgs;
};
```

Description : destiné à contenir les caractéristiques de chaque commande : la chaîne de caractère ("C", "R", "DELETE", etc), et le nombre de paramètre qu'elle prend (2, 3, ∞ pour "DELETE" et "POLYLIGNE", etc).

3.1.4 Attributs

Les attributs de la classe Application :
– *filename* (string), chemin relatif ou absolu du fichier.

3.1.5 Méthodes

Les méthodes de la classe application :

```
Launch();
```

Description : boucle principale de récupération des entrées clavier.

```
void interpret(string cmdLine);
```

Description : prend en paramètre une ligne de commande et lance les bonnes actions sur la figure en fonction de cette commande.

```
bool parseLine(Cmd & cmd, stringstream & line, int nbArgs);
```

Description :

```
bool Write(string str);
```

Description : écrit dans le fichier la chaîne *str* passée en paramètre.

```
ifstream * FileStream();
```

Description : retourne un pointeur vers le flux de lecture du fichier.

4 Description et justification des structures de données

4.1 Nœuds & Arcs

Les nœuds et les arcs sont stockés séparément dans deux arbres rouge noir dont les clés sont le label du nœud ou la représentation sérialisée d'un arc. Cela permet de stocker chaque nœud de manière unique et la recherche et l'insertion sont rapides (complexité $O(log(n))$). Cela permet aussi de générer de manière efficace le fichier *GraphViz* ainsi que les statistiques (complexité $O(n)$).

FIGURE 1 – Implentation en mémoire des données du graphe

5 Diagramme de classes

FIGURE 2 – Diagramme UML classes

6 Description des tests

Identifiant du test	Nom du test	Description synthétique du test
1	Test des statistiques	Vérification du bon fonctionnement du programme en terme d'élaboration de statistiques
2	Test du générateur Dot	Vérification du bon fonctionnement du programme en terme de génération du fichier GraphViz (*.dot)
3	Test du filtre sur fichiers	Vérification du bon fonctionnement du programme en terme de filtrage des entrées de logs sur le type de fichier cible (*.png, *.css, *.js)
4	Test du filtre sur horaires	Vérification du bon fonctionnement du programme en terme de filtrage des entrées de logs sur l'horaire
5	Test traitement log vide	Vérification du bon fonctionnement du programme en terme de traitement d'un fichier de log vide
6	Test traitement entrée de log	Vérification du bon fonctionnement du programme en terme de traitement d'une entrée de log réduite au strict minimum i.e. les paramètres non utilisés sont réduits à '-'
7	Test traitement entrée de log	Vérification du bon fonctionnement du programme en terme de traitement d'une entrée de log réduite au strict minimum i.e. les paramètres non utilisés sont réduits à '-'