Université de Montréal

Rapport Travail Pratique #1

Par

Charbel Kassis, p0976458

Département d’informatique et de recherché opérationnelle

Faculté des arts et sciences

Travail présenté à Edouard BATOT et Lechanceux LUHUNU KAVUYA dans le cadre du cours IFT3913,

Qualité du logiciel et métriques

28 Septembre 2016

# Les packages :

* uml.exception : ce package contient les exceptions qui sont lancés par le programme lors qu’il y a une erreur durant le parsing causé par une erreur de syntaxe.
* *uml.extraction :* ce package contient l’ensemble des classes qui sont utilisé durant le parsing pour extraire une ligne spécifique selon l’état du parsing. En y trouve 3 type de classes différentes : La classe Extractor qui est la superclasse de toutes les classes d’extraction, la classe ExtractorBuilder qui est une classe servant à construire des objets de type Extractor et une enum qui sert à stocker des valeurs sémantiques tel que les noms des éléments UML (Classes, Agrégation, Arguments, etc.)
* uml.gui : ce package contient les objets des interfaces graphiques : la fenêtre principale et les panneaux où sera afficher la liste des éléments extraits du texte.
* uml.gui.controller: ce package contient les listeners et la classe qui relie les listeners à chacun des composantes graphiques de la fenêtre.
* uml.parser : ce package contient la fonction de parsing
* uml.struct : ce package contient les objets formant une structure arborescente de la grammaire uml. La racine étant UML\_Model. Chacune de ces classes hérite de la classe AbstractUML.
* uml.main : le main du programme.

# Les classes importantes :

* UML\_Exception : La classe abstraite héritant de RuntimeException, les 7 autres exceptions du même package héritent de UML\_Exception
* Extractor : La classe abstraite qui demande d’implémenter la fonction extract() et prenant en paramètre un enum UML\_Element. Selon le type d’élément à extraire et le type de l’Extractor, une string est retournée. Généralement, ceci retourne un élément particulier dans le fichier texte à parser, sinon il retourne null si une action s’est effectuée sans erreur. L’Extractor s’occupe principale d’extraire de l’information sur une seule ligne.
* ExtractorBuilder : La classe qui sert à construire des objets de type Extractor. Il sert uniquement pour rendre le code plus visible lorsque les classes Extractor vont être utilisées dans la classe UML\_Parser. le constructeur peut être vide ou bien il prend en paramètre une chaîne de caractères. Pour construire un objet Extractor, il suffit d’appeler la méthode .build() en lui passant en paramètre un enum UML\_Element. Exemple : new ExtractorBuilder().build(ARGUMENT) retournera un objet de type ExtractArgument
* UML\_Element : L’enum qui contient des noms d’éléments UML. Ils serviront pour la création d’un Extractor et pour effectuer des switch/case.
* UML\_Frame : C’est la classe qui représente la fenêtre du programme. En y trouve directement le boutons ainsi que les zones où des listes de mots s’y trouveront.
* UML\_Panel : Classe abstraite qui décrit la forme d’un panneau dans la fenêtre. Chaque panneau aura un titre et un JScrollPane(scroll bar si nécessaire) dans lequel s’y trouvera une JList (la liste des mots à afficher).
* UML\_List : Classe abstraite qui hérite de UML\_Panel. Cette classe sert spécifiquement à afficher la liste. Il faut implémenter les fonctions qui servent à lui ajouter une liste d’AbstractUML puis une autre fonction qui devrait itérer dans cette liste pour produire des strings qui serviront à être ajouté à la liste.
* UML\_InformationPanel : Classe qui représente le panneau ayant 4 panneaux interne : La liste des attributs, la liste des opérations, les sous-classes et les agrégations/relations.
* UML\_ClassListListener : C’est le listener qui écoute les changements d’état dans la liste des classes. Une fois le parsing réussit, on peut sélectionner (avec la souris ou les clavier) le nom des classes, cette action engendrera la fonction interne de la UML\_ClassListListener. Cette fonction affichera les listes dans UML\_InformationPanel (attributs, opérations, etc)
* UML\_FileChooserListener : C’est le listener du bouton de chargement de fichier. Lorsqu’on clique sur le bouton, UML\_FileChooserListener ouvrira un JFileChooser puis si un fichier a été sélectionner en succès, le fichier sera immédiatement parser. Lorsque le parsing est réussi, une boîte de message apparaîtra pour indiquer le succès. Ensuite, la liste des classes sera affichée dans la liste à gauche.
* UML\_GUI\_Controller : C’est la classe qui s’occupe de démarrer la fenêtre et d’ajouter les listeners à chacune des composantes respectives.
* UML\_Parser : C’est la classe qui s’occupe de parser le fichier. Elle crée des Extractor spécifique selon la situation. Si l’Extractor s’occupe d’extraire les informations sur une ligne, alors le UML\_Parser s’occupe de relier logiquement chacune des lignes ensembles. Les informations extraites avec les Extractor sont placées dans des objets AbstractUML dont une référence vers la racine (de type UML\_Model) est gardée en mémoire comme propriété de la classe UML\_Parser.
* AbstractUML : Classe abstraite qui représente un élément UML. Chacune de ses sous-classes pointent vers un élément interne. Exemple : un UML\_Model pointe vers une liste de UML\_Class, UML\_Generalization, UML\_Aggregation, UML\_Association. Cette structure en arborescence servira pour stocker les informations extraites pour les afficher ensuite dans la GUI.

# Déroulement du programme :

Le programme débutera par lancer le main : Le main crée un objet UML\_Frame et UML\_GUI\_Controller. Le controller lancera la fenêtre tout en ajoutant des listeners aux composantes de la fenêtre.

**public** **class** Main {

**public** **static** **void** main(String[] args) {

UML\_Frame frame = **new** UML\_Frame();

UML\_GUI\_Controller controller = **new** UML\_GUI\_Controller(frame);

controller.start();

}

}

Lorsque l’utilisateur pèse sur « Charger fichier », le UML\_FileChooserListener qui lui est associé exécutera sa fonction interne :

Il débutera le parsing de la fonction en appelant un objet de type UML\_Parser puis lui passera le fichier choisit en paramètre. le parser lui a été fournis par le controller. ce parseur va être partagé par la suite par d’autres listeners.

...

File file = chooser.getSelectedFile();

**try** {

/\* parser le fichier \*/

**this**.parser.addFile(file);

**this**.parser.parse();

...

 À l’aide des streams, la classe UML\_Parser lira le fichier ligne par lire une fois que la méthode .parse() sera appelée. La fonction parse() regardera l’état du parsing pour exécuter une fonction de parsing spécifique. Chacune de ces fonctions crées des Extractor spécifiques pour chercher les données dans le texte. Une fois ces données sont extraites, un objet de la sous-classe de AbstractUML sera créé pour stocker cette information, puis l’objet lui-même sera placé dans un autre objet de la sous-classe de AbstractUML pour les relier ensemble : Un UML\_Attribute crée sera ajouté à la liste des attributs de la dernière UML\_Class crée par exemple.

Lorsque le parsing est terminée et qu’aucune exception est lancée due à une erreur de syntaxe, alors la méthode interne de UML\_FileChooserListener continuera par afficher le chemin du fichier à droite du bouton puis une fenêtre s’ouvrira pour nous affirmer que le parsing est réussi.

Finalement, les méthode setList() et showList() de UML\_ClassList (hérités de UML\_List) seront appelées pour afficher les classes.

**this**.frame.getClassList().setList(umlClasses);

**this**.frame.getClassList().showList();

/\* mettre le chemin du fichier dans le label \*/

**this**.frame.getFileChooser().getLabel().setText(file.getAbsolutePath());

Lorsqu’on sélectionne les nom de classes avec la souris ou le clavier, l’écouteur de changement déclenche sa fonction interne pour appeler à son tour les méthode setList() et showList() des 4 panneaux se trouvent dans les UML\_InformationPanel.

**public** **void** valueChanged(ListSelectionEvent e) {

JList<String> list = (JList<String>) e.getSource();

String selectedItem = list.getSelectedValue();

**if**(selectedItem != **null**)

**this**.showResults(selectedItem);

}

**private** **void** showResults(String selectedItem) {

UML\_Class selectedClass = **this**.frame.getClassList().getClassMap().get(selectedItem);

**this**.showAttributes(selectedClass);

**this**.showMethods(selectedClass);

**this**.showSousClasses(selectedItem);

...

}

showAttributes, showMethods, etc, appeleront les setList() et showList()

setList() et showList() ont des implémentations différentes selon le type d’information. Le nom des classes par exemples sont affiché directement a partir de la liste des classes, mais pour les sous-classe par exemple, il faut parcourir la listes des UML\_Generalization puis chercher celles dont l’identifiant est égale au nom de la classe sélectionner, puis utiliser cette UML\_Generalization comme paramètre pour setList().

# Diagramme de classe de l’interface graphique (graphe partiel, il manque des classes par faute d’espace) :

