Université de Montréal

**Devoir 2**

Par  
Marc Laliberté

Et

Ounissa Nait Amer

Bacc. en Informatique

Travail présenté à Edouard Batot

Dans le cadre du cours IFT-3913

Qualité du logiciel et métriques

Octobre 2018

**Conception :**

Pour mettre à jour notre programme nous avons commencé par définir les nouvelles fonctionnalités qui doivent être ajoutées, et quelles parties de notre programme allaient être touchées par ces modifications :

Les nouvelles fonctionnalités qu’on avait à coder sont les suivantes :

* Calcul de métriques pour chacune des classes du diagramme de classe.
* Affichage des résultats des métriques via l’interface graphique
* Visualisation des définitions des métriques.
* Pouvoir produire un fichier csv contenant les différentes métriques.
* Batterie de tests pour le programme

Pour pouvoir mettre en œuvre ces nouvelles fonctionnalités, nous avons essayé de ne pas apporter trop de modifications à la structure du programme. Pour cela on a eu à ajouter des attributs et méthodes à nos classes déjà définies, nous avons ainsi défini les métriques comme des méthodes de la *Class\_dec* qui est la classe qui instancie les classes du diagramme de classe parsés. Nous avons aussi modifié la classe de l’interface graphique pour ajouter un bouton qui permet de générer le fichier CSV, et une section qui affiche les résultats des différentes métriques.

Notez que les méthodes, attributs, commentaires et javadoc sont tous en anglais.

**Diagramme de classes :**

Pour ce travail nous avons eu a aporter des modifications à notre diagramme de classes. Les modifications que nous avons ajouté sont les suivantes :

* Nous avons supprimé la classe *PrinterHelper* , qui nous servait pour le premier travail comme point de repers pour verifier le résultat du parseur, et supprimé la classe *Get\_ClassList.*
* Nous avons aussi défini de nouvelles classe :

*\* ParserTest* qui englobe les tests unitaires.

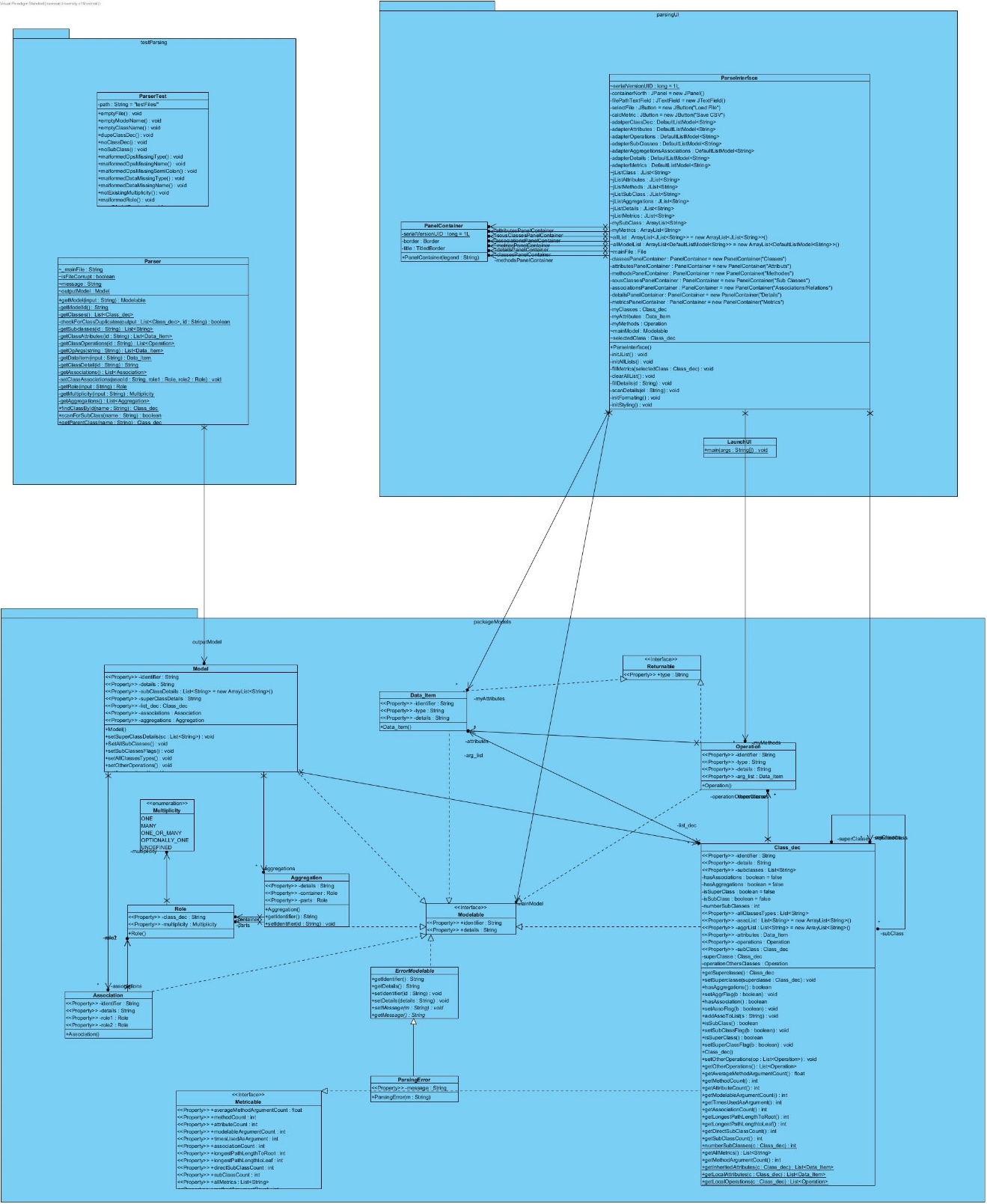
\* *ParsingError* qui va être applée en cas d’erreur de parsing.

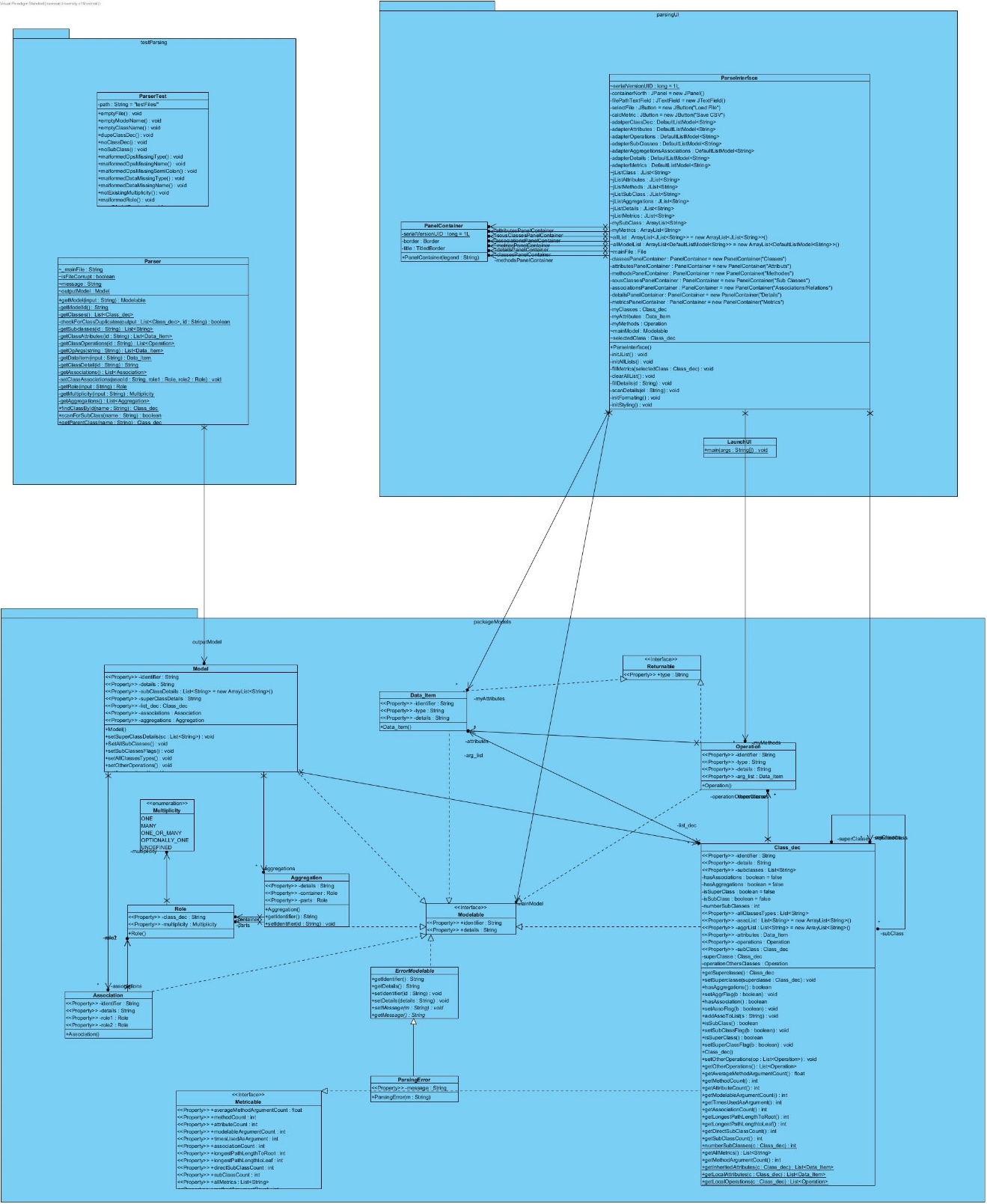
\* la classe abstraite *ErrorModelable* pour mettre une abstraction pour la classe *ParsingError* et la simplifier étant donné que la majorité des méthodes de *ErrorModelable* ne font rien du tout.

\*L’interface *Metricable* qui definies les differentes metriques. Et cette derniere est implemantée par la classe *Class\_dec*, là oû les differntes methodes relatives au métriques sont implémenté.

* De nouveau attributs sont apparus dans la classe *Class\_dec* pour nous permettre notamment de garder la liste des sous\_classes, des **booleen** pour vérifier si la classe est une sous\_classe, une classe parent ou si elle a des aggregations ou pas.
* Nous avons ajouter des methodes à la classe *Model* notamennt pour setter quelques attibuts de *Class\_dec*.
* Nous avons ajouté a la classe *ParsInterface* des méthodes pour obtnir les résultats des métriques et une méthode pour afficher les définition de chacune d’elles dans détails.

Le diagramme de classe au complet est illustré dans les deux figures qui suivent :

****

****

Lancement du programme :

La commande Linux pour lancer le programme est la suivante :

Naviguez dans le dossier remise/code source et lancer la commande :

Java -cp . parsingUI/LaunchUI

De plus, un fichier *shell script* est fourni dans le dossier remise/code source/classes pour un lancement plus facile des classes précompilées.

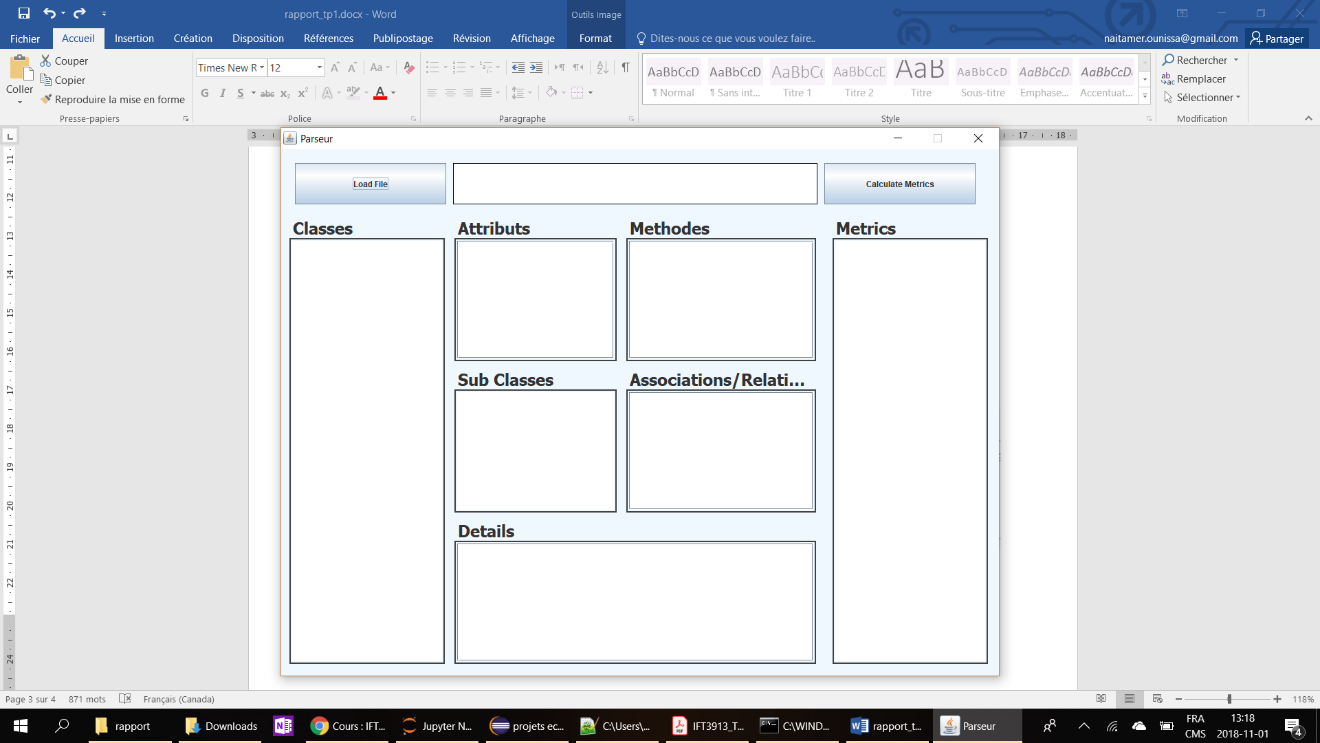
Un ficher .jar exécutable est aussi fourni dans le dossier racine remise.

Manuel utilisateur

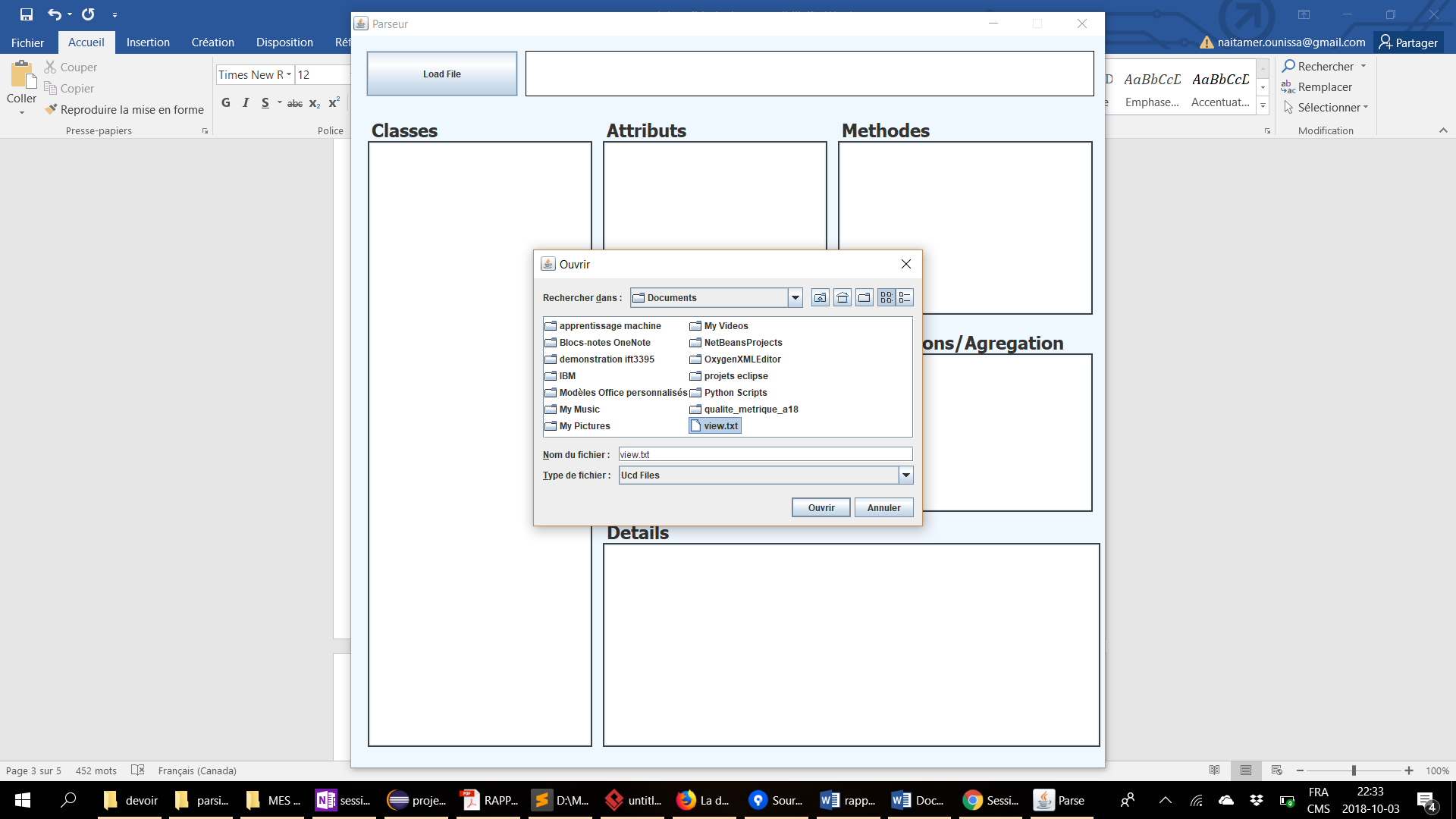
L’application consiste à construire un diagramme de classe à partir d’une représentation sous format texte et permet de le visualiser.

Son utilisation se fait comme suit :

**Présentation de l’interface graphique** : après avoir lancer le programme on obtient cette interface graphique :



1

**1 :** en cliquant sur le bouton *load file* une fenêtre de dialogue s’ouvre sur le répertoire *mes documents*, pour nous permettre de choisir un fichier a parser.

**A noter** que le choix de fichier et filtré de sorte qu’on ne puisse charger que des fichiers qui sont sous format ucd ou txt.

**2 :** après chargement du fichier, la barre de texte nous permet de voir le chemin entier (path) du fichier sélectionné.

**3 :** ce sont des sections qui permettent un affichage dynamique des différentes composantes du diagramme de classes elle regroupe les onglet *Classes*, *Attributs*, *Methodes*, *Sub Classes*, *Associations/Relations*.

Après chargement du fichier (ci celui-ci n’est pas corrompu), a la section *Classes* on aura une liste de toutes les classes avec la possibilité de cliquer sur chacune d’entre elle pour afficher les informations qui lui sont relatives : ainsi si la classe choisie a 2 attributs et une seule méthode, on va voir les 2 attributs affichés dans la section *Attributs* et la méthode affichée dans la section *Méthodes.*

**4** : cette partie nous permet de visualiser le detail relatif a chaque élèment des sections 3.

En cliquant par exemple sur une sous\_classe on va voir la partie du fichier texte où cet élèment a été mentionné.

Ici aussi vont s’afficher les definition relatif aux metriques : quand une metrique de la section 6 est selectionnee sa definition s’affiche dans la section 4.

**5** : en cliquant sur le bouton 5, les 10 metriques vont être calculées pour l’élèment qui a été selectionné et on pourras visualiser le resutat dans la section 6.

**6** : sette section sert a afficher le resultat des metriques.

**Gestion des fichiers erronés** : lorque le fichier choisi ne correspond pas a la grammaire BNF ou qui’il est vide, une boite de dialogue avec un message d’erreur s’affiche en indiquand la nature de l’erreur.

Analyse de notre code

Nous trouvons que notre classe *Classe\_dec* est beaucoup trop grosse et aurait pu être optimisée d’avantage, notamment pour ce qui est en lien avec les métriques. Il y a une petite lacune au niveau du Parser et beaucoup d’éléments auraient pu être mis dans les Class\_dec au niveau du parsing pour simplifier la tâche des calculs de métriques. Il y a malheureusement un peu dédoublement d’effort à ce niveau. Par exemple, garder une liste de string et *class\_dec* pour les *subclasses* est définitivement une faiblesse de conception ici.

De plus, il est mentionné dans l’énoncé d’utiliser les fonctions statiques avec parcimonie. Nous avons définitivement utilisée ce mot clé un peu plus que l’on aurait du, notamment au niveau du *Parser*. La logique derrière notre raisonnement était que le *parser* n’est pas un objet nécessaire à la conception ou la visualisation du diagramme de classe. Nous l’avons donc considéré un peu comme une base de donnée ou on peut lui faire des requêtes quelconque.

Aussi, notre classe d’interface graphique est un peu lourde. Cependant, nous avons essayé de garder un certain ordre ou structure pour rendre la visibilité et le maintien plus facile. La partie dont nous sommes le moins fier a ce niveau est ce qui nous permet d’afficher les détails des associations et agrégations. Cette partie est très lourde même si les commentaires essaient d’expliquer au maximum. Voir les commentaires à même la classe pour plus de détails.

Extraction des métriques de notre programme

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Chemin | classe | #classes | NLOC | CLOC | ANA | NOM | NOA | ITC | ETC | CAC | CLD | DIT | NOC | LOD |
| n.d | Aggregation | 1 | 35 | 60 | 0.5 | 8 | 4 | 2 | 0 | 0 | 1 | 0 | 0 | 0 |
| n.d | Association | 1 | 37 | 58 | 0.5 | 8 | 5 | 2 | 0 | 0 | 1 | 0 | 0 | 0 |
| n.d | Class\_dec | 1 | 580 | 366 | 0.684 | 31 | 8 | 0 | 8 | 0 | 1 | 0 | 0 | 0 |
| n.d | Data\_Item | 1 | 6 | 70 | 0.6 | 7 | 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| n.d | ErrorModelable | 1 | 20 | 39 | 0.5 | 6 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| n.d | Metricable | 1 | 16 | 90 | 0 | 12 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| n.d | Modelable | 1 | 7 | 28 | 0.5 | 4 | 1 | 0 | 0 | 0 | 0 | 2 | 7 | 8 |
| n.d | Model | 1 | 152 | 130 | 0.5 | 12 | 7 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| n.d | Multiplicity | 1 | 8 | 14 | 0 | 1 | 5 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| n.d | Operation | 1 | 40 | 55 | 0.4285 | 9 | 5 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| n.d | ParseInterface | 1 | 493 | 249 | 0,4545 | 11 | 35 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| n.d | Parser | 1 | 410 | 200 | 0.8947 | 19 | 4 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| n.d | ParsingError | 1 | 11 | 23 | 0 | 7 | 2 | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| n.d | Returnable | 1 | 5 | 17 | 0.5 | 2 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 2 |
| n.d | Role | 1 | 19 | 36 | 0.5 | 4 | 2 | 1 | 6 | 3 | 0 | 0 | 0 | 0 |

quelques statistiques :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | mean | std | max | min | total |
| chemin |  |  |  |  | 6 |
| #classes |  |  |  |  | 18 |
| NLOC | 122.6 | 198.3319008 | 580 | 5 | 1839 |
| CLOC | 95.66666667 | 101.1291018 | 366 | 14 | 1435 |
| ANA | 0.436228571 | 0.262798725 | 0.8947 | 0 | - |
| NOM | 9.4 | 7.490946917 | 31 | 1 | - |
| NOA | 5.666666667 | 8.4148395 | 35 | 1 | - |
| ITC | 0.6 | 0.91025899 | 2 | 0 | - |
| ETC | 1 | 2.478478796 | 8 | 0 | - |
| CAC | 0.266666667 | 0.798808637 | 3 | 0 | - |
| CLD | 0.6 | 0.632455532 | 2 | 0 | - |
| DIT | 0.333333333 | 0.6172134 | 2 | 0 | - |
| NOC | 0.733333333 | 1.830950833 | 7 | 0 | - |
| LOD | 0.8 | 2.077085871 | 8 | 0 | - |

L’analyse de ces chiffres montre qu’on a une grande disparité dans la taille des classe. En effet, les valeur moyenne de LOC sont au allonteur de 315 mais avec une variation allant jusqu’à 300 ce qui montre qu’on a des classes volumineuse et d’autres beaucoup plus petite. A constater aussi que le rapport des commentaire/ligne\_de\_code est assez important. Par contre avec un ITC autour de 0.6 on pet dire qu’on a un faible coublage entre les classes. L’heritage est peu profond la moyenne des sous\_classe étant de 0.8 avec un maximum de 8 classes. Nous devons ici noter que la valeur de CLOC utilisée est Ligne de Code totale de la classe – LoC, pour des raisons de temps, les lignes vides seront donc considérées dans cette valeure.

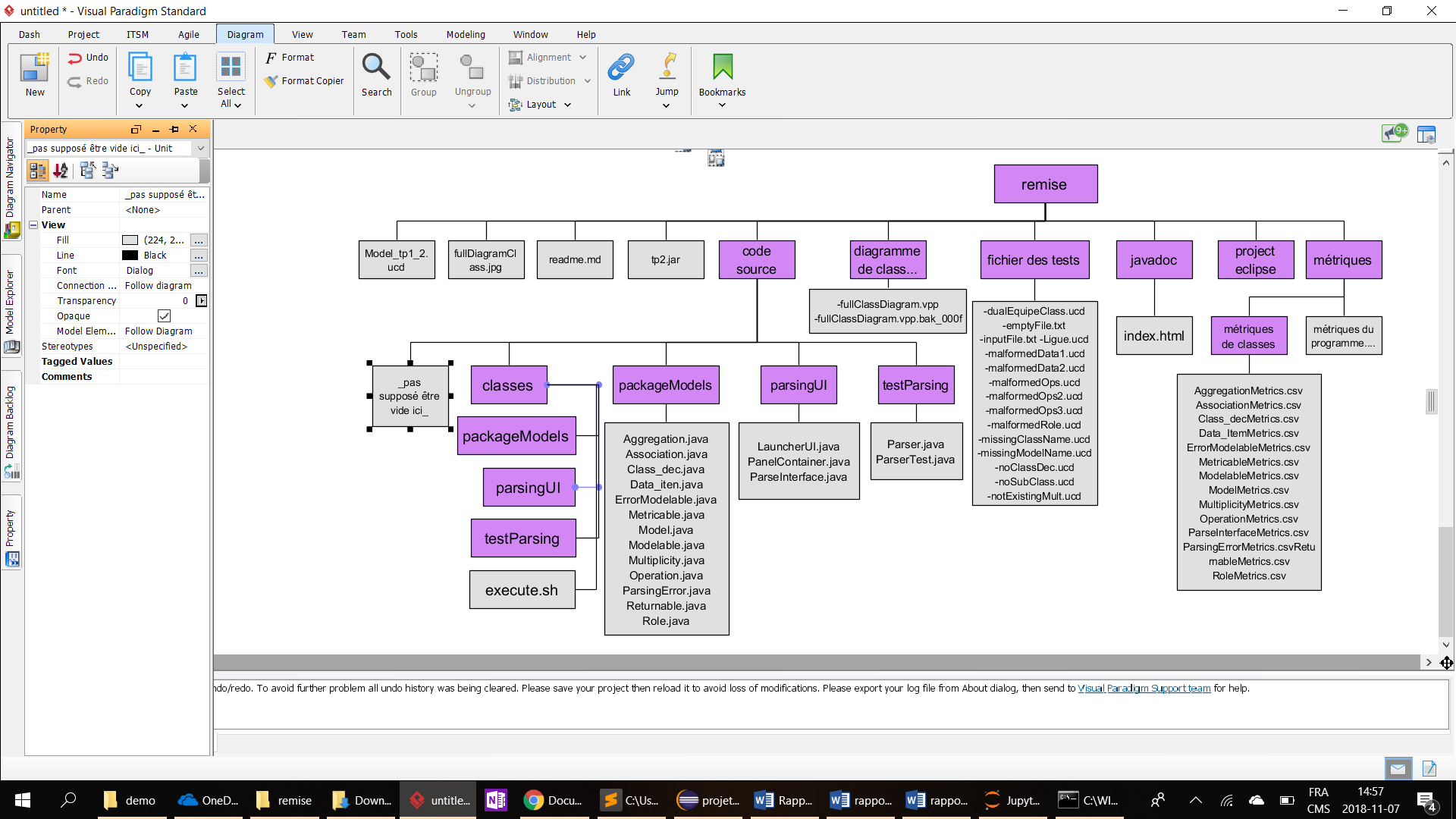
Analyses et Tests effectués :

Pour notre programme nous avons 15 tests unitaires qui correspondent chacun à un fichier *.ucd* avec un nom correspondant au test.

Les 14 premiers sont tous corrompus et contiennent dans l’ordre les erreurs suivantes:

* Fichier avec un nom du model vide
* Fichier avec un nom de la classe vide
* Fichier avec une class vide
* Fichier avec une classe dupliquée
* Fichier avec un model qui ne possède aucune classe
* Fichier avec les noms de sous-classes vide
* Fichier avec le nom ou le type d’une opération vide
* Fichier avec une opération contenant juste le type de retour.
* Fichier avec une opération mal déclarée (sans les : entre le nom de la méthode et le type de retour).
* Fichier avec un attribut sans type.
* Fichier avec un attribut sans nom
* Fichier avec une multiplicité erronée (n’existe pas)
* Fichier avec une relation sans les rôles de lasses

L’exécution de chacun de ces tests revoit une erreur et l’.

* Fichier non corrompu (ligue.ucd)
* Arborescence de notre fichier de remise :
* 

Les limites de notre programme

Nous avons pu relever quelques limites de notre programme qui sont :

* S’il y a deux ou plus espaces après une déclaration de méthode les attributs ne sont pas *parser*.
* Si un attribut est déclaré avec 2 types le fichier sera *parsé* mais le 2eme type ne sera pas pris en compte.
* Si une méthode est déclarée avec 2 type le fichier sera *parser* et les deux types seront affichés
* Si une classe ne contient aucun attribut le parseur lance une erreur de classe vide est lancée.
* On aurait pu mettre plus de tests unitaires on estimes que 15 ce n’est pas assez. En plus de tester les cas d’erreurs, on aurait pu tester les cas de succès (nombres de méthodes, arguments, attributs, etc.)
* On aurait du mettre des tests pour les métriques. L’ensemble des tests de nos métriques est donc la comparaison des résultats du programme et ceux que nous effectués à la main.

Le temps passé sur le devoir :

On estime le temps total passé sur le devoir de 20 heures chacun.