Jules COHEN *20032498*
Olivier LAMY-CANUEL *1034976*

IFT3913

Rapport TP#2

7 novembre 2018

Université de Montréal

Arborescence :

- IFT3913_TP2_COHEN_LAMY-CANUEL :
  - Artefacts :
    - IFT3913_Rapport_TP2.pdf
    - Parcer.ucd
    - Parcer_complet.csv
    - parcer.jar
  - Src :
    - ParseurView.java
    - Controller :
      - ParseurController.java
      - parseurView.fxml
    - Metrics :
      - CalcMetrics.java
      - CSV.java
      - Metrics.java
    - Model :
      - Parcer.java
      - UMLAggregate.java
      - UMLAssoAggrega.java
      - UMLAttributes.java
      - UMLClass.java
      - UMLComponent.java
      - UMLMethode.java
      - UMLRelation.java
    - Test :
      - CalcMetricTest.java
  - Bin : contient les fichiers .class

Nous avons tous les deux passé environ 25 heures sur ce devoir.

1. INTRODUCTION

La conception d'un parseur s'est faite selon le modèle MVC avec une extension permettant de calculer les métriques des classes parsées. Cette structure nous semble la meilleure puisqu'elle facilite la division du travail et l'encapsulation. Les différentes composantes se trouvent dans les packages aux noms peu originaux de *Model, Controller, Metrics et Test.*

2. MODEL

Le *model* reprend la même structure que pour le TP1. Cependant nous y avons fait quelques petites modifications que voici :

=== Changement sur Parcer ===

- Modification *parcing UMLClass* (voir *UMLClass*)
- Modification de la *Regex* pour évaluer les classes pour accommoder le cas du constructeur.
- Modification de la présentation pour rendre cette *Regex* légèrement plus maintenable.
- Ajout identification parent pour sous-classe (*parcing generalization*)
- Ajout erreur si héritage multiple
- Ajout erreur en cas d'héritage en boucle (permet l'utilisation de récursivité pour les calculs des métriques)
- Ajout d'une méthode pour réinitialiser les marks de toute les classes.

=== *UMLClass* ===

- Ajout attribut parent + méthode pour gérer.
- Ajout mark pour validation héritage + méthode pour gérer.
- Ajout second constructeur pour parser information interne.

=== UMLMethod ===

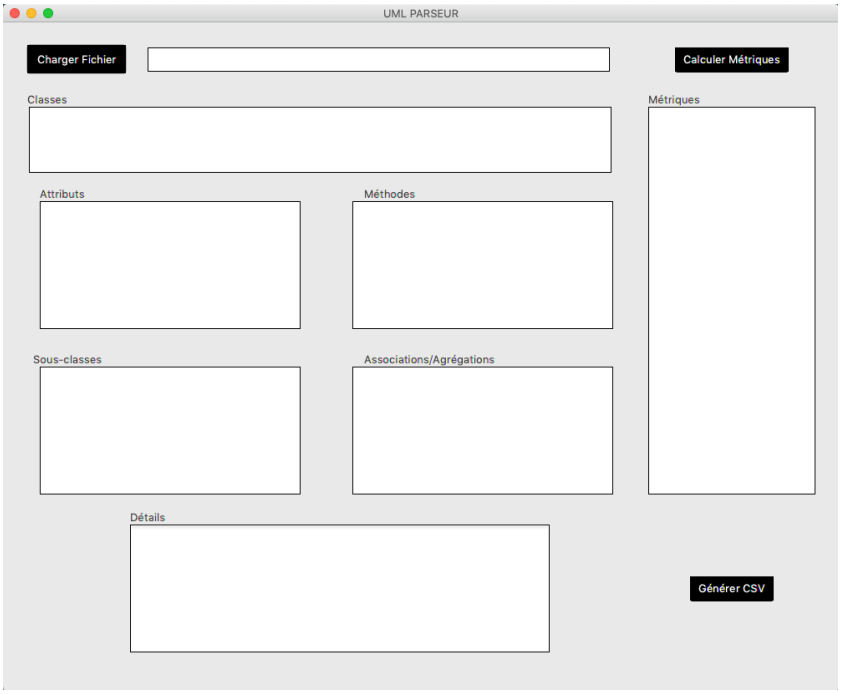
- Ajout d'un ArrayList contenant les arguments de la méthode. Ajout de méthode pour parser ces arguments.

=== UMLAttribute ===

- Ajout d'attributs type et nom + méthode pour parser ces attributs.

3. VUE ET CONTROLLER

3.1. parseurView.fxml et ParseurView.java (Default package) :



L'interface grapgique du parcer

L'interface graphique du parseur est chargée à partir du fichier *parseurView.fxml* .

Le fichier a été créé grâce au logiciel *Scene Builder* qui a généré le *XML*. Cela nous a apporté un réel gain de temps et nous a permis de consacrer le temps restant pour implémenter le *controller ParseurController.java* ainsi que *Parcer.java* et toutes les autres classes du modèle.
L'interface se compose d'un bouton « Charger Fichier » qui permet comme son nom l'indique de charger le fichier et de débuter le parsing. Une fois le parsing effectué, les noms des classes vont être affichés dans la ListView à l'horizontale *Classes.* Le fait que mettre cette liste à l'horizontale donne l'impression que les classes sont disposées en onglets.
Nous avons également un bouton « Calculer métriques », qui va permettre d'effectuer le calcul des métriques pour toutes les classes parsées. Une fois les calculs effectués, lors d'un clic sur une des classes, le résultat des métriques sera affiché dans la ListView « Métriques ». De plus lors d'un clic sur une de ces métriques, sa définition sera affichée dans la section détails.
Le bouton « Générer CSV » va, comme son nom l'indique, générer un fichier CSC à la demande l'utilisateur.

3.2. ParseurController.java (Package Controller) :

Tous les *listenners* des boutons et des ListView sont initialisés dans *ParseurController.java.* Toute les classes sont chargées dans un *ArrayList arrayClass* une fois le fichier parsé. Chacune des ListView correspondent à un objet UML (ex : *ListView<UMLClass>, ListView<UMLAttribute>*…). De plus, chacun des objets UML du package Model possédant une redéfinition *de toString(),* l'affichage dans les listes est simplifié et évite un certain nombre de boucle grâce à la méthode *ListView.addAll()* permettant de charger directement à partir des *ArrayList* d'objet UML de la classe *UMLClass*.

On a choisi également de pouvoir utiliser la touche *Enter* pour charger/parser le fichier en plus du bouton. Nous trouvons ça quand même plus pratique et plutôt intuitif.

En ce qui concerne les métriques, une fois qu'elles sont calculées, elles sont stockées dans *ArrayList<CalcMetrics> listMetrics*. Chaque élément de cette liste contient toutes les métriques pour une classe données que l'on récupère en *faisant listMetrics.get(i).getMetrics*. Cela va nous retourner un *ArrayList<Metrics>* contenant toutes les résultats des calculs pour une classe donnée.
Pour la création du CSV, nous récupérons le nom du fichier parsé et nous le passons en paramètre du constructeur de CSV.
Des *popups* sont mis en place pour avertir l'utilisateur lorsqu'il clique sur le bouton Calculer Métrique ou Générer CSV mais l'opération n'est pas possible (ex : fichier non parsé ou calcul non effectué).

4. METRIQUES

4.1. Méthodes de calcul : CalcMetrics.java

Cette classe permet de calculer chacun des 10 métriques pour une classe donnée.
Les résultats retournés vont permettre d'instancier des objet Metrics contenant le nom, la valeur ainsi que la définition de la métrique, qui vont être stocker dans *une ArrayList<Metrics>.* Cette ArrayList servira pour l'affichage dans les ListView ainsi que pour la création du CSV.

Voici des définitions de nos fonctions de calculs :

- *calcANA(UMLClass ci)* : Renvoie la somme des d'arguments de chaque méthode de ci (obtenue à l'aide de la taille de l'*ArrayList* arguments) et divise le tout par le nombre de méthodes de ci.

- *CalcNOM(UMLClass ci)* : Renvoie la taille de l'A*rrayList* fournie par la méthode getUniqueMethod.

- *calcNOA(UMLClass ci) :* Renvoie la taille de *l'ArrayList attributes* de ci additionné au NOA de son parent (s'il existe).

- *calcITC(UMLClass ci, ArrayList<UMLClass> modelClass) :* Renvoie la valeur donnée par l'additionne des résultats de la fonction *calcFreqClassAsArgsType* en passant comme paramètre chaque méthodes de ci et chaque *UMLClass de modelClass*.

- *calcETC(UMLClass ci, ArrayList<UMLClass> modelClass)* : Renvoie la valeur donnée par l'addition des résultats de la fonction *calcFreqClassAsArgsType* en passant comme paramètre ci pour chaque méthodes de chaque membre de *modelClass.*

- *calcCAC(UMLClass ci) :* Renvoie la taille de *l'arrayList aggregate* de ci additionné ; de l'arrayList relations de ci ; additionné du CAC de la classe parent de ci (s'il existe).

- *calcDIT(UMLClass ci) :* Monte récursivement dans l'arbre d'héritage à partir de ci jusqu'à atteindre une racine et incrémente la valeur retournée à chaque saut.

- *calcCLD(UMLClass ci) :* Utilise la méthode *getLeafs()* pour obtenir *l'ArrayList* de toute les *UMLClass* feuilles pouvant être atteinte à partir de ci. Calcule le DIT de chaque classe feuille pour déterminer celle qui est la plus loin de la racine. Soustrait le DIT de ci pour obtenir la valeur recherchée.

- *CalcNOC(UMLClass ci) :* Renvoie la taille de *l'ArrayList* contenant les sous-classes de ci.

- *calcNOD(UMLClass ci) :* Renvoie NOD de ci additionné du NOD de chaque sous-classe de ci.

4.1.1. Autres méthodes

- *getUniqueMethod () :* Utilise la méthode *getUniqueMethod* pour *l'ArrayList* de toutes les méthodes de la classe (locale et hérité). Les méthodes redéfinies ne sont ajoutées qu'une seule fois dans *l'ArrayList*.
- *CalcFreqClassAsArgsType () :* Calcul la fréquence à laquelle la classe est un type d'une méthode
- *getLeafs() :* Renvoie l'ensemble des sous-classes feuilles pouvant être obtenu à partir de *umlClass.*

4.2. Metrics.java

Cette classe permet de définir une métrique selon son nom, sa valeur (integer ou float), et sa définition. La définition de la métrique lui ai donné par la fonction setDef(), permet de récupéré à l'aide d'un switch, la définition de la métrique selon son nom.

4.3. CSV.java

La classe CSV a un constructeur prenant en paramètre un *ArrayList* de *CalcMetric* (chaque élement contient toutes les métriques pour une classe donnée), ainsi que le nom du fichier que l'on a auparavant parsé, pour nommer le CSV avec le même nom.
La méthode *writeCsvFile ()* va parcourir le *ArrayList* mentionné plus haut, récupérer les métriques et, classe par classe, ajouter les résultats des calculs, une classe par ligne.

5. ASSURANCE QUALITE

Nous avons utilisé trois méthodes distinctes pour effectuer les tests soit : une inspection visuelle, l'utilisation de UCD pour valider le parcing et l'utilisation de tests unitaires pour valider le résultat du calcul des métriques.

Le GUI a été inspecté visuellement tant pour valider le fonctionnement du parcer du point de vue de l'utilisateur que d'assurer une utilisation plaisante du logiciel.

Le parcing a été validé à l'aide de plusieurs fichier .ucd représentant des cas distincts devant être pris en charge. Ces tests ont été effectués en démarrant le logiciel et en chargeant des fichiers contenant une erreur spécifique. Nous avons noté le comportement du logiciel par rapport à ces fichiers et effectué les correctifs au besoin de sorte à ce que le logiciel renvoie le bon message d'erreur.

En ce qui concerne le calcul des métriques, nous avons jugé bon de faire des tests unitaires pour assurer la validité des résultats. Chacun de ses tests a été effectué en créant une structure fictive composée des objets pertinents au calcul pour ensuite valider la valeur obtenue avec la valeur attendue.

6. ANALYSE DES METRIQUES

	NCLOC	CLOC	ANA	NOM	NOA	ITC	ETC	CAC	DIT	CLD	NOC	NOD
max	243	115	1.04	20	16	17	19	12	1	1	3	3
min	7	10	0	1	0	0	0	0	0	0	0	0
moyenne	77	41	0,66	7	5	2	2	3	0	0	0	0
médiane	47	24	0,75	5	4	0	0	1	0	0	0	0
mode		62	1	3	5	0	0	1	0	0	0	0
Ecart-type	84	38		6	4	5	5	3	1	0	1	1

La faible valeur de ANA (maximum de 1.04) s'explique en bonne partie par l'utilisation très fréquente de l'objet *UMLClass* comme argument. Puisqu'il s'agit de l'objet central de notre

structure de donné, cette utilisation est justifiée selon nous. Il est par contre a noté que les modifications de cet objet sont risquées étant donné cette utilisation très fréquente. L'aspect centrale de la classe UMLClass dans la structure de donné est par ailleurs mis en évidence par la valeur très élevée de ETC (19) et de CAC (12) ce qui correspond à ce qui est observé sur le diagramme UML. Le recours à l'ensemble de la structure de donnée par *CalcMetrics* est quant à lui mis en évidence par sa valeur de ITC (17) et de NOM (20) très élevée, en particulier comparativement aux autres classes.En ce qui concerne le *controller*, son NOA très élevé (17) comparativement aux autres classes dénote clairement son rôle de pont entre la Vue et le modèle du logiciel.Finalement, les valeurs très basses de DIT, CLD, NOC et NOD indique que nous avons un arbre d'héritage très peu profond ce qui correspond à la structure représentée sur le diagramme UML.

Sommes toutes, nous croyons que ces métriques valide les observations faites sur le diagramme UML et sont conséquentes avec l'architecture, tel que planifier, du logiciel.

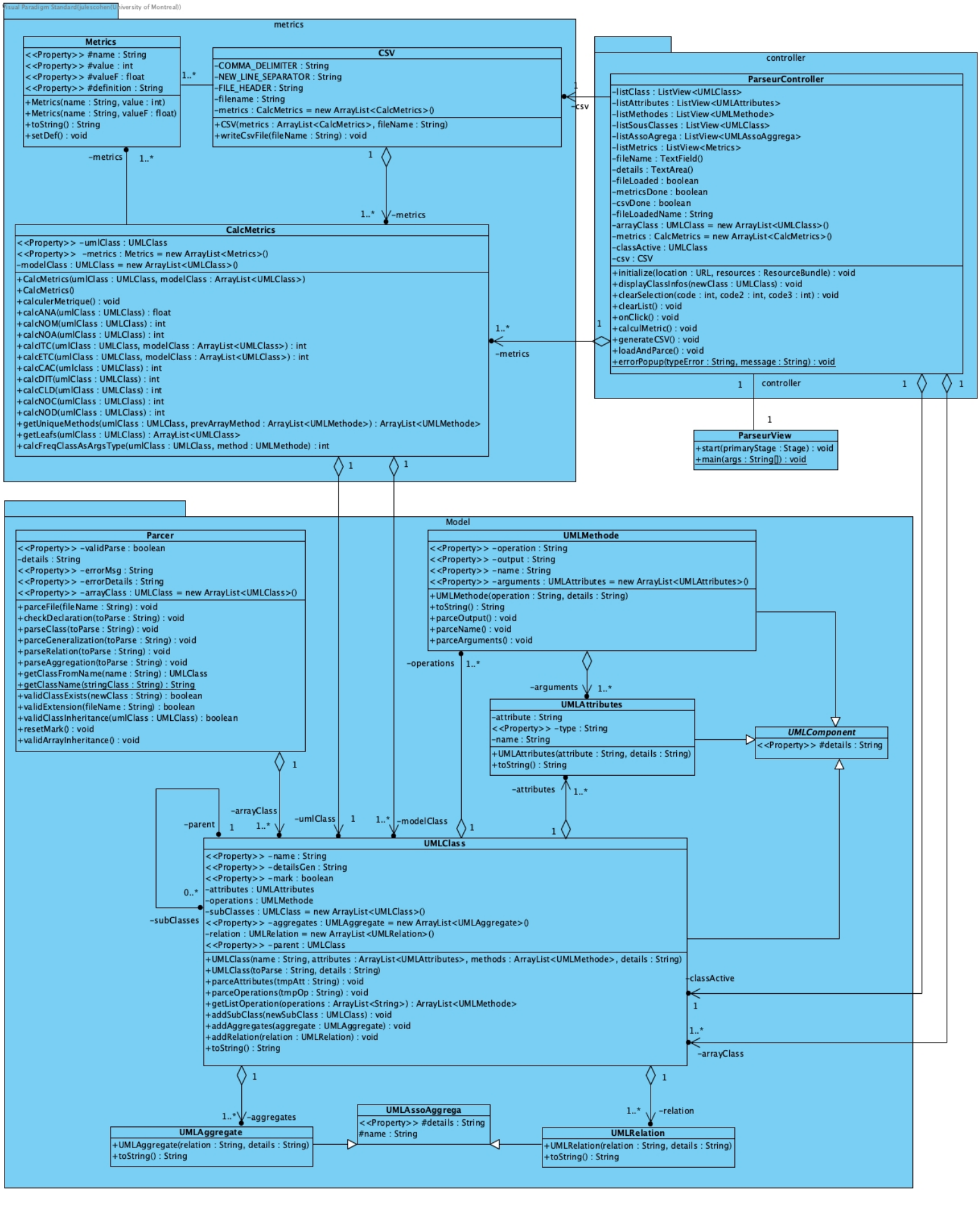

Visual Paradigm Standard(julescohen(University of Montreal))
metrics
Metrics
<<Property>> #name : String
<<Property>> #value : int
<<Property>> #valueF : float
<<Property>> #definition : String
+Metrics(name : String, value : int)
+Metrics(name : String, valueF : float)
+toString() : String
+setDef() : void
1..*
CSV
-COMMA_DELIMITER : String
-NEW_LINE_SEPARATOR : String
-FILE_HEADER : String
-filename : String
-metrics : CalcMetrics = new ArrayList<CalcMetrics>()
+CSV(metrics : ArrayList<CalcMetrics>, fileName : String)
+writeCsvFile(fileName : String) : void
1
-csv
-metrics
1..*
1
1..*
-metrics
CalcMetrics
<<Property>> -umlClass : UMLClass
<<Property>> -metrics : Metrics = new ArrayList<Metrics>()
-modelClass : UMLClass = new ArrayList<UMLClass>()
+CalcMetrics(umlClass : UMLClass, modelClass : ArrayList<UMLClass>)
+CalcMetrics()
+calculerMetrique() : void
+calcANA(umlClass : UMLClass) : float
+calcNOM(umlClass : UMLClass) : int
+calcNOA(umlClass : UMLClass) : int
+calcITC(umlClass : UMLClass, modelClass : ArrayList<UMLClass>) : int
+calcETC(umlClass : UMLClass, modelClass : ArrayList<UMLClass>) : int
+calcCAC(umlclass : UMLClass) : int
+calcDIT(umlClass : UMLClass) : int
+calcCLD(umlClass : UMLClass) : int
+calcNOC(umlClass : UMLClass) : int
+calcNOD(umlClass : UMLClass) : int
+getUniqueMethods(umlClass : UMLClass, prevArrayMethod : ArrayList<UMLMethode>) : ArrayList<UMLMethode>
+getLeafs(umlClass : UMLClass) : ArrayList<UMLClass>
+calcFreqClassAsArgsType(umlClass : UMLClass, method : UMLMethode) : int
1..*
-metrics
1
1
controller
ParseurController
-listClass : ListView<UMLClass>
-listAttributes : ListView<UMLAttributes>
-listMethodes : ListView<UMLMethode>
-listSousClasses : ListView<UMLClass>
-listAssoAgrega : ListView<UMLAssoAggrega>
-listMetrics : ListView<Metrics>
-fileName : TextField()
-details : TextArea()
-fileLoaded : boolean
-metricsDone : boolean
-csvDone : boolean
-fileLoadedName : String
-arrayClass : UMLClass = new ArrayList<UMLClass>()
-metrics : CalcMetrics = new ArrayList<CalcMetrics>()
-classActive : UMLClass
-csv : CSV
+initialize(location : URL, resources : ResourceBundle) : void
+displayClassInfos(newClass : UMLClass) : void
+clearSelection(code : int, code2 : int, code3 : int) : void
+clearList() : void
+onClick() : void
+calculMetric() : void
+generateCSV() : void
+loadAndParce() : void
+errorPopup(typeError : String, message : String) : void
1
1
controller
1
1
1
ParseurView
+start(primaryStage : Stage) : void
+main(args : String[]) : void
Model
Parcer
<<Property>> -validParse : boolean
-details : String
<<Property>> -errorMsg : String
<<Property>> -errorDetails : String
<<Property>> -arrayClass : UMLClass = new ArrayList<UMLClass>()
+parceFile(fileName : String) : void
+checkDeclaration(toParse : String) : void
+parseClass(toParse : String) : void
+parceGeneralization(toParse : String) : void
+parseRelation(toParse : String) : void
+parseAggregation(toParse : String) : void
+getClassFromName(name : String) : UMLClass
+getClassName(stringClass : String) : String
+validClassExists(newClass : String) : boolean
+validExtension(fileName : String) : boolean
+validClassInheritance(umlClass : UMLClass) : boolean
+resetMark() : void
+validArrayInheritance() : void
1
UMLMethode
<<Property>> -operation : String
<<Property>> -output : String
<<Property>> -name : String
<<Property>> -arguments : UMLAttributes = new ArrayList<UMLAttributes>()
+UMLMethode(operation : String, details : String)
+toString() : String
+parceOutput() : void
+parceName() : void
+parceArguments() : void
-operations
1..*
-arguments
1..*
UMLAttributes
-attribute : String
<<Property>> -type : String
-name : String
+UMLAttributes(attribute : String, details : String)
+toString() : String
UMLComponent
<<Property>> #details : String
-attributes
1..*
1
-parent
1
-arrayClass
1..*
-umlClass
1
1..*
-modelClass
1
UMLClass
<<Property>> -name : String
<<Property>> -detailsGen : String
<<Property>> -mark : boolean
-attributes : UMLAttributes
-operations : UMLMethode
-subClasses : UMLClass = new ArrayList<UMLClass>()
<<Property>> -aggregates : UMLAggregate = new ArrayList<UMLAggregate>()
-relation : UMLRelation = new ArrayList<UMLRelation>()
<<Property>> -parent : UMLClass
+UMLClass(name : String, attributes : ArrayList<UMLAttributes>, methods : ArrayList<UMLMethode>, details : String)
+UMLClass(toParse : String, details : String)
+parceAttributes(tmpAtt : String) : void
+parceOperations(tmpOp : String) : void
+getListOperation(operations : ArrayList<String>) : ArrayList<UMLMethode>
+addSubClass(newSubClass : UMLClass) : void
+addAggregates(aggregate : UMLAggregate) : void
+addRelation(relation : UMLRelation) : void
+toString() : String
0..*
-subClasses
-classActive
1
1..*
-arrayClass
1
1..*
-aggregates
1
1..*
-relation
UMLAggregate
+UMLAggregate(relation : String, details : String)
+toString() : String
UMLAssoAggrega
<<Property>> #details : String
#name : String
UMLRelation
+UMLRelation(relation : String, details : String)
+toString() : String

Diagramme de Classe :

Nous avons opté pour un modèle MVC étant donné qu'il s'agissait, selon-nous, de la meilleure approche pour découper les différentes fonctionnalités du programme et ainsi répartir les tâches. Chaque *package* correspond donc à un aspect de ce modèle à l'exception du package *Metric* étant donné que le calcul devait être conçu comme une extension.
Nous avons ainsi pu limiter au maximum le couplage entre les classes des différents packages. De plus, nous avons opté pour l'utilisation d'une classe centrale à chaque package en vertu de leur fonctionnalité ce qui nous a permis davantage le couplage entre les classes appartenant à différent packages. On note aisément, à l'aide du diagramme de classe, que la classes *UMLClass* joue un rôle central dans la structure de donnée sous-jacente du package Model. En ce qui concerne le package *Metric*, nous avons opté pour l'utilisation de deux classes distinctes pour chaque fonctionnalité soit *CalcMetrics* qui contient l'ensemble des méthodes permettant le calcul des métriques et CSV qui génère le fichier CSV.
Nous croyons que cette sous-division du logiciel facilite l'analyse de son fonctionnement et par le fait même rend l'ajout de fonctionnalité nettement plus aisé.

Manuel Utilisateur

Pour exécuter le .jar : (en se plaçant dans le dossier de parcer.java sinon donner le path en entier)

java - jar parcer.jar

Pour compiler puis exécuter (dans le dossier /src):

javac ParseurView.java

java ParseurView

Compilation au DIRO:

En essayant de compiler avec : javac parseurView.java

On a obtenu l'erreur suivante :

ParseurView.java:3: error: package javafx.application does not exist

En essayant d'executer parcer.jar

Jar: Error: A JNI error has occurred, please check your installation and try again
Exception in thread "main" java.lang.UnsupportedClassVersionError: ParseurView has been compiled by a more recent version of the Java Runtime (class file version 53.0), this version of the Java Runtime only recognizes class

Utilisation :

- Entrer le nom du fichier à parser, soit le avec le *path* en entier du fichier ou juste le nom du fichier s'il se trouve dans le même dossier
- Appuyer sur *Enter* ou cliquer sur le bouton *Charger fichier*
  - Si popup d'erreur apparait :
    - Le nom du fichier n'est pas valide
    - Il y a une erreur de syntaxe dans le fichier
  - Entrer un nouveau nom de fichier
- La liste des classes apparait.
  - En choisissant une classe, ses informations apparaissent dans les différentes views de la fenêtre.
  - En cliquant sur un attribut, une méthode, une sous-classe ou une association/agrégation, la partie correspondante dans le fichier parsé apparait dans la *view* Détails
- Pour calculer et afficher les résultats des calculs de métriques, cliquer sur *Calculer Métriques.*
  - Un popup d'erreur apparait si aucun fichier n'a été parsé.
  - Sinon les résultats des calculs de métriques vont être affichés.
- Pour générer le CSV, cliquer sur *Générer CSV* :
  - Un popup d'erreur va apparaitre si aucune métrique n'a été calculée.
  - Sinon un .csv va être généré, portant le même nom que le fichier en .ucd

Si vous souhaitez parser un nouveau fichier, il suffit juste de taper son nom à la place du présent fichier.