Université de Montréal

**Devoir 1**

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

**Diagramme de classes :**

Nous avons tout d’abord commencé à lire la grammaire BNF fournie avec l’énoncé. Nous nous sommes basé sur celle-ci pour créer nos classes modèles.

Nous avons opter pour une hiérarchie simple ou chaque partie du programme (classe, attributs, opérations, data item) est un objet en soit et tout est encapsulés dans la classes Model (voir diagramme de classe).

Pour minimiser les efforts de codage, nous avons aussi choisis de créer deux classes interfaces java, soient *Modelable* et *Returnable*, ce qui nous a permis de générer rapidement des méthodes semblables sur plusieurs classes.

Par ce choix on estime avoir une meilleure robustesse et facilité l’entretient (exemple si jamais on doit modifier beaucoup de choses pour le tp2), nous avons préférés utilisé des interfaces et ne pas utiliser des classes parents.

Nous avons donc créé le diagramme de classe en se basant sur ces choix. De plus, avec une certaine structure, nous avons pu commencer a séparé le travail qui devait être fait. Ce fut donc facile de travailler une personne sur le *parsing* et l’autre sur l’interface graphique et ce, en évitant au maximum les *merge conflicts*.

Pour ce qui est du couplage, nous avons essayé de le minimiser au maximum. Le *Parser* est la classe qui interagit surtout avec les modèles et c’est cette classe qui crée le Model qui contient toute l’arborescence. D’ailleurs, nous sommes conscient que le *Parser* et *ParsInterface* sont de grosses classes, mais nous n’avons pas vu l’utilité les séparer en plusieurs objets ou classes.

La classe Parser par instancier des objet\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C’est donc les classes de l’interface graphique qui demande un Model au *Parser* lorsque l’on charge un fichier. Ce sont ces classes qui s’occupent de remplir l’interface graphique.  
  
Pour nous aider avec les tests de *Parsing* et vérifier si tout s’est bien déroulé lors du chargement du fichier, nous avons créé la classe *PrinterHelper* qui nous a aidé a afficher les données qui sont chargées et retournés par nos expressions régulières.

On a separe notre travail en 3 package. On a deux package principaux ParsingUI et PArsingTest dans le premier on fait un afficger graphique

Faire

Un diagramme de sequence pour mieux shematiser ce qui se passe

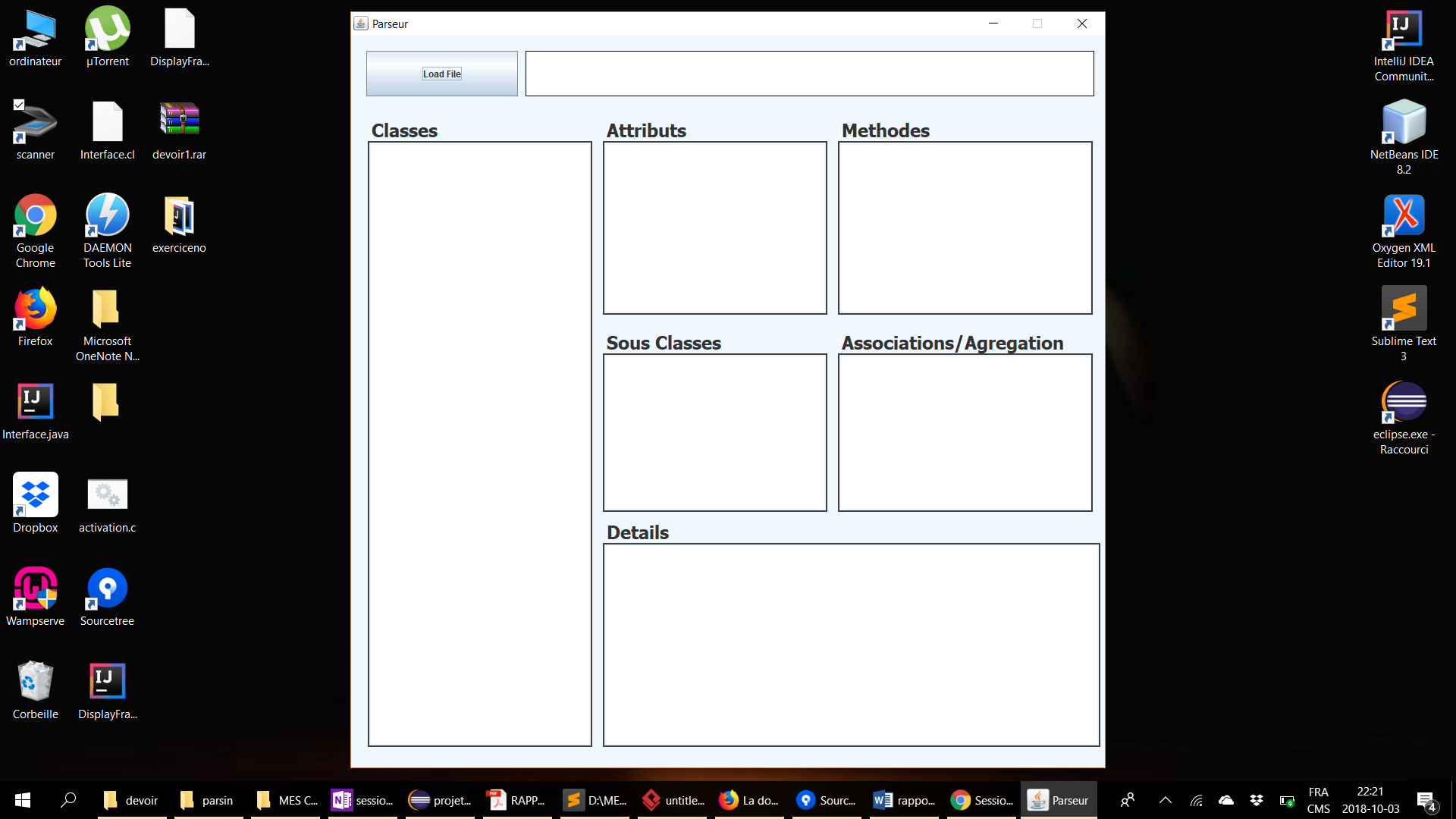
Manuel utilisateur

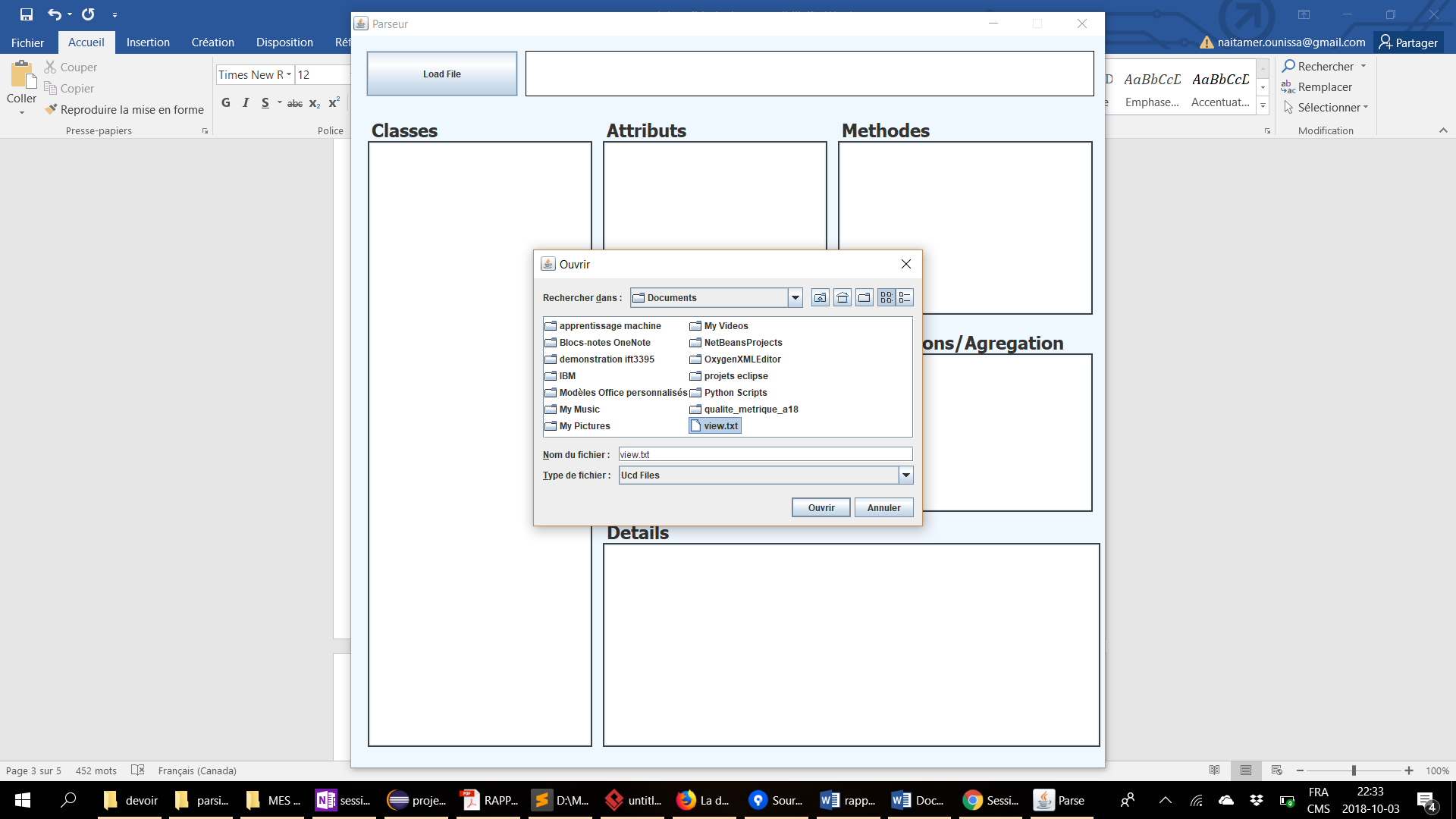
L’application consiste à construire un diagramme de classe à partir d’une représentation sous format texte et permet de le visualiser.

Son utilisation se fait comme suit :

**Lancer le programme** : on peut lancer le programme en cliquant sur le fichier *tp1.jar* ou en utilisant la commande suivant en ligne de commande : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

1



1 : en cliquant sur le bouton *charger ficher* une fenêtre de dialogue s’ouvre sur le répertoires 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 cette barre 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.

Après chargement du fichier 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 element des sections 3.

En cliquant par exemple sur une sous\_classe on va voir la partie du fichier texte ou cet element a été mentionné. Noter que cette partie est dotté d’un scrol bar.

**Gestion des fichiers erronés** : lorque le fichier choisi ne correspond pas a la grammaire BNF ou vide un message d’erreur s’affiche en indiquand la nature de l’erreur. Comme mentionné precedement on ne procede pas a la verification de l’extension du fichier puisque lors du chargement on applique un filtre qui permet uniqument les fichier de la bonne extention.