RAPPORT DE PROJET

Programme objet oriente et Java

Checker & Beautifier in Java for Python language

Pires Hervé ANAGONOU
Bouaouni Mohamed Samy
Groupe D

Sommaire

Introduction

Le but de ce projet est de créer un programme qui va être capable de lister tous les fichiers avec l'extension “.py” qui sont disponible dans un répertoire donné, détecter tout type d’erreur mentionné, modifier le contenu du fichier en fonction des détections, afficher le statistique des détections sur l'ensemble des fichiers. Il a été demandé d’avoir une version console et une version interface, dans laquelle une fenêtre secondaire va être affichée montrant les fichiers et les sous répertoire disponible. Dans ce projet nous allons expliquer comment nous avons faire pour finir ce projet, en expliquant d’abord les grandes ligne du projet, puis les opérations liée à la recherche des fichiers de façons récursive, la détection du shebang, la détection des erreurs de typage, détection des commentaires pydoc, la modification du contenu du fichier en cas de détection positive, avant de terminer par le façon dont nous avons codé la console et les interfaces graphique visuelle.

Plan de développement

Au départ, le plan pour le programme était de le créer au cours du mois de novembre à la moitié du mois décembre avant les examens, puis de travailler sur les autres éléments du projet à compléter, comme ce rapport et la soutenance, en janvier.Cela semblait fonctionner durant les premières semaines. Des progrès notables ont été réalisés, et le programme a rapidement pris forme. Le codage du mode console et les points d'avancement se sont bien passées sans commentaire négatif. L'interface visuelle était en cours, et il ne manquait que la modification, que l'on croyait être une tâche facile. Il y avait sept classes différentes pour les recherches de fichier, détection des erreurs, modification et statistique, une pour cli, et une pour l'application visuelle. Cependant, les progrès ont ralenti à cause des révisions pour l'examen. L'interface visuelle s'est avérée difficile à coder. Ainsi, après l'examen de décembre, les jours qui ont suivi ont été consacrés principalement à l'interface visuelle, au rapport, et à la découverte de la façon d'optimiser le code. La partie visuelle devenant de plus en plus complexe, elle a été faite plusieurs fois par chacun d'entre nous pour qu'à la fin on puisse juste créer une union d'idées pour créer une version définitive.

Diagramme UML

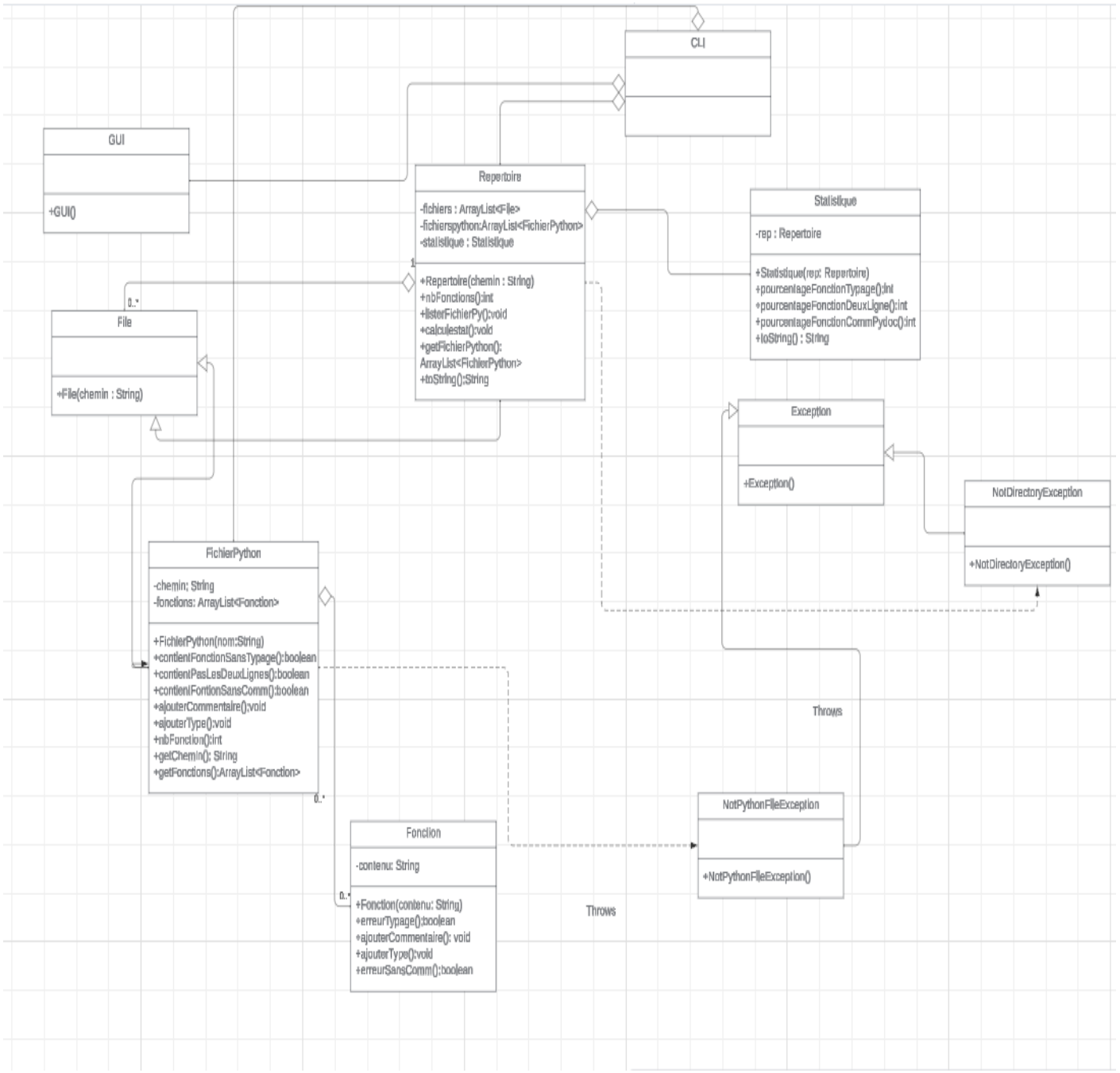

figure 1: Diagramme de classe du de logiciel

Recherche des fichiers Python

Tout d'abord pour pouvoir trouver les fichiers python dans un répertoire, on doit parcourir récursivement celle ci, plus précisément chaque fois que l'on tombe sur dossier on fait appelle à la même méthode ou si on tombe sur un nom de fichier ce finissant avec ".py", on le stock. C'est ainsi qu'est née la classe Répertoire qui contient la méthode listerFicherPy. Les fichiers python trouver sont stockés dans une liste de type File (ArrayList) qui nous permet de stocker au fur et à mesure un nombre illimité de fichier finissant avec l'extension « .py ». Mais notons que l'argument passer en paramètres qui est de type String doit mener vers un dossier sinon y aura un arrêt immédiat du déroulement du programme qui signifie pour nous une Exception levée. Pour cette première base de notre projet, on utilise nos propres connaissances (duo) tout en essayant de comprendre davantage les différentes fonctions que peut nous procurer la classe File qui est la classe mère de la notre (Répertoire) .

Opération sur les fichiers trouvés

- **Détection des fichiers qui ont pas le shebang**

A cette étape, tous les fichiers trouvés qui était stocké dans la liste nommée « fichierPython » vont être examinés, plus précisément les deux premières lignes. On peut soit compter le nombre total de fichiers n'ayant pas un bon shebang (qui est composée à la première ligne par " #! + chemin du dossier parent de notre fichier + python3" et à la deuxième ligne par (commençant par " #-*- " et finissant par " -*- ")) grâce à l'utilisation du BufferedReader et du FileReader qui est un outil de lecture d'un fichier donner, ou soit identifier ceux qui ne possèdent pas en appliquant sur eux une action de modification qui vas être expliqué dans une partie ci dessous.

- **Lister les fonctions qui sont présentes dans le fichier**

Pour avoir toutes les fonctions disponibles dans un fichiers de type Python donner, on a eu à faire appel à BufferedReader et du FileReader pour la classe FichierPython. Cet appel nous permet de vérifier à chaque fois si l'on tombe dans un fichier python ,une ligne commençant par « def », si oui ,on stock dans une variable de type String qui est déclarée et initialisée à vide, cette ligne et en plus un « \n » qui signifie à la ligne et on refais ce stockage tant que qu'une nouvelle commençant par « def » n'est pas trouver. Et si on retombe en plus sur une ligne commençant encore pas « def », on stocke dans une liste (ArrayList) de type Fonction une nouvelle instance du type Fonction qui est une classe qui prend en argument un String. Notre String ici est représenté par toutes les lignes

stocker après avoir vu un « def » et la boucle et refais ça tant que on n'arrive pas à la fin de lecture qui est la fin du fichier .

➔ **Lister les fonctions qui sont présentes dans le fichier**

★ *Opération sur les fonctions trouvés*

Détection des fonctions qui ont des erreurs de typage

La variable fonctions qui est une liste et qui dans la classe FichierPython, contient que des objets de type Fonction qui est lui même composé du corps d'une fonction ou procédure présent dans un fichier. Pour chaque objet Fonction on doit vérifier si le typage est bon, ce qui est gérée par la méthode « existeAnnotationType ». Dans cette méthode on détecte la signature de la fonction en utilisant le Pattern et le matcher pour la trouver dans la fonction précise.

Détection des fonctions qui ont pas les commentaires pydoc

De plus cette variable définie ci dessus contient en plus une autre méthode noter « existePyDoc » qui grâce à l'index de là où commence les commentaires et finis les commentaires, vérifie si il contient bien tous les détails d'un commentaire pydoc

★ *Modification sur les fonctions trouvées*

Ajouts des commentaires en cas d'absence

Par rapport à l'ajout , il se fait par là construction total du squelette du pydoc en fonction du nombre de paramètres et si c'est de type fonction ou procédure, la taille de cette structure dépend de donc du nombre de paramètres et si c'est une fonction ou une procédure.

Modification sur les fichiers trouvés

Correction du shebang en cas de problème

À ce niveau on modifie totalement le contenu du fichier en copiant le reste du contenu fichier à part les deux lignes où il y a eu l'erreur et la fin après avoir corrigé les deux lignes , on procède à l'ajout du reste du code qui était détaché..

Statistique sur les fichiers trouvés sur la base des détections

Le statistiques se fait sur plusieurs fichiers où à partir du nombre de fichier de type python trouver , on calcule le pourcentage de tous les détection faite

Affichage

- Console et Interface graphique

Le code de l'interface de la console appelle différentes méthodes en fonction des arguments qu'il reçoit. C'était assez facile à coder. Nous localisons l'endroit où l'utilisateur a saisi l'option, et nous vérifions que l'argument qui le suit est bien un chemin valide. Initialement, une boucle 'for' était utilisée, mais comme il y a au maximum quatre arguments dans une exécution valide du jar, elle était vraiment surchargée. À sa place, nous avons une méthode qui localise où se trouvent les arguments et d'opérer les différents types d'action de détection et de modification .Concernant le code de l'interface graphique, il a été compliqué parce qu' on avait du mal.a ce décider sur une bonne version et on a faire plusieurs test pour être sur . Lors du lancement de l'interface vous avez la possibilité de tester sur un fichier en particulier ou tester sur un dossier. Mais la différence est que l'analyse de dossiers nous donne en plus comme option la statistique complète en fonction des détections . Les modifications aussi sont directes sur l'interface ce qui est décidé par une vérification d'abord.

Conclusion

Pour conclure, lorsque nous avons commencé le projet, nous avons largement sous-estimé le temps qu'il nous faudrait pour le terminer. Nous avons correctement réparti les tâches, choisissant plutôt de faire la plupart du code ensemble . Ce projet a permis une union de nos connaissances et d'apprendre l'une de l'autre sur la manière de coder.