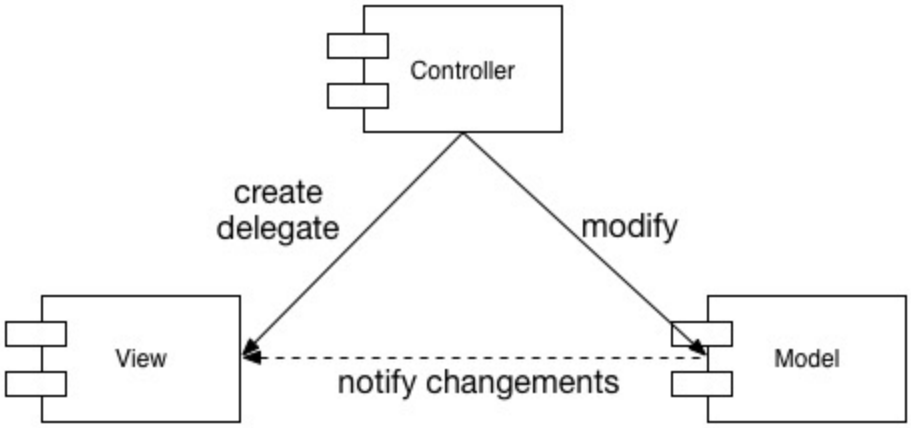
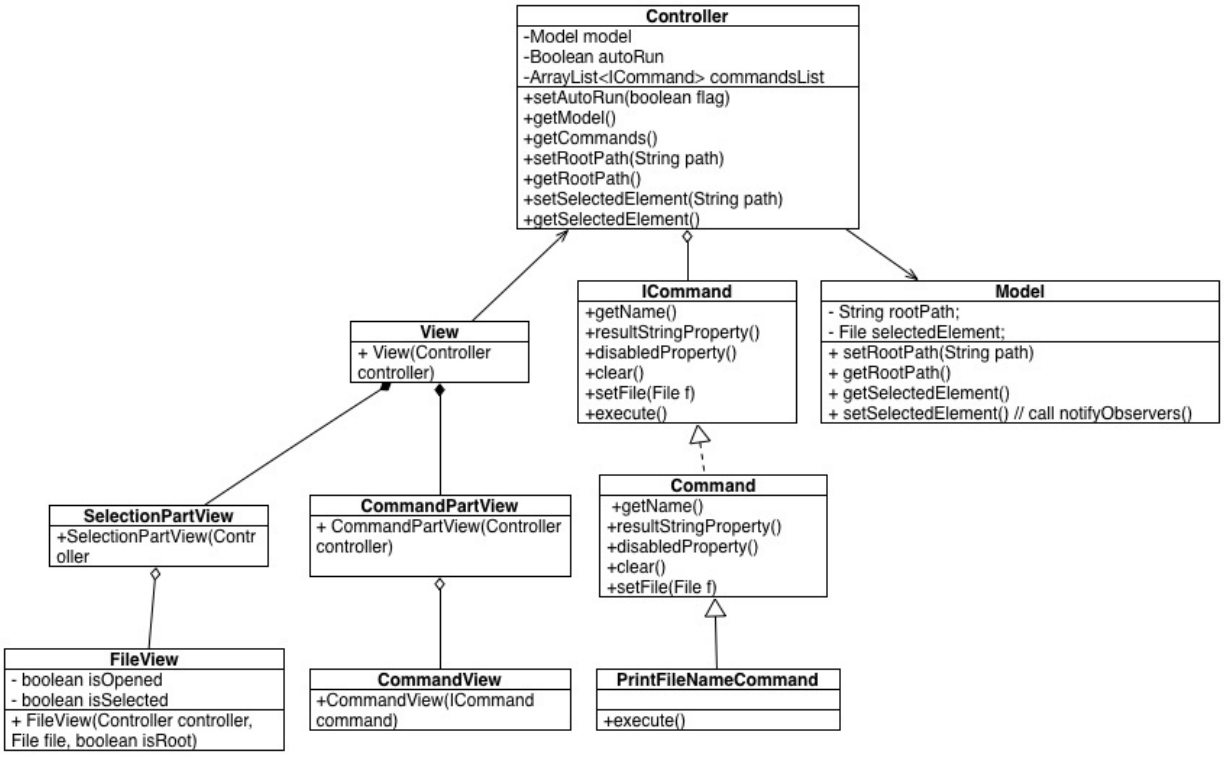
Guide du développeur

P1)

Nous avons choisi une architecture de type MVC. Cela permet de bien séparer la vue (interface utilisateur), du modèle (racine et fichier séléctionnés) ainsi que des controleurs (séléction de fichier, execution de commandes etc..)
L'architecture est détaillée dans le diagramme UMl suivant :





Dans les différents composants, nous avons aussi utilisé divers patrons de conception :

- La modele et les vues utilisent le patron de conception "Observer". En effet, le champ de texte de la vue observe le résultat de la commande, afin de se mettre à jour

automatiquement (par exemple quand la commande est executée, ou que la commande est “cleared” ou que le fichier est changer)

- Nous avons implementer le “Command” patron les commands. Les commandes a une fonction setFile(File f) pour set le *receiver*, et une fonction execute() pour executer la commandes.
- Nous avons pensé d’utiliser la patron de conception “composite” et “visitor” pour construire l’arbre de fiches et effectuer les commands. Mais finalement, nous n’avons pas les implementé et nous sauvegarde qu’une rootPath et une selectedFile dans la modèle. L’arbre de fichier est construite dans la vue seulement.
- Nous avons pensé d’utiliser le “Singleton” pour le modèle. Ça permet de créer qu’une modèle. Finalement, pour cette petite application, ce n’est pas necessaire et nous avons retiré le Singleton.

P2)

Nous avons défini l’interface *ICommand* et une abstract class qui est commune à toutes les commandes :

ICommand

- public void setFile(File f); // set le fiche pour executer
- public void execute(); // execute la command
- public void clear(); //remettre le resultat
- public String getName(); // get the name of the command
- public StringProperty resultStringProperty(); // pour update la vue les resultat
- public BooleanProperty disabledProperty(); // pour update la vue pour disable le button

P3)

Nous avons creé une class “Controller”, qui a un attribut private ArrayList<ICommand> commandsList, pour stocker tous les commands; Nous pouvons executer les commandes à partir de ce “Controller”.

On a créé une abstract classes “Command”, pour mettre les methodes commune, et 3 concrete commandes classes:

- PrintPathCommand : retourne le “filepath” du fichier
- PrintFileNameCommand : retourne le nom du fichier, et “error” si c’est un dossier
- PrintFolderNameCommand : retourne le nom du dossier, et “error” si c’est un fichier

P4)

Le premier problème que la distinction fichier/dossier pose est le suivant : les boutons pour exécuter les commandes doivent être grisés si celles-ci ne sont pas compatibles avec le fichier sélectionné.
Pour cela nous modifions la méthode set File à l'interface ICommand et ajoutons une methode "public BooleanProperty disabledProperty()" :

- Quand la method "public void setFile(File f)", est appelé, on vérifie d'abord la compabilité de méthod avec l'élément sélectioné. S'ils ne sont pas compatible, nous mettons l'attribut "disabledProperty" en false;

- public BooleanProperty disabledProperty(); Elle est permet d'observer le booléen "disabledProperty" qui indique si oui ou non la commande est compatible. Les boutons des commandes, dans l'IHM, pourront donc observer cette variable.

P5) Done

P6) Done

P7) Done

P8) P9) La charge dynamique n'est pas implementé jusqu'au maintenant.