

## 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 sélectionnés) ainsi que des contrôleurs (sélection de fichier, exécution 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 modèle 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 exécutée, ou que la commande est "cleared" ou que le fichier est changé)
- Nous avons pensé d'utiliser le patron de conception "composite" et "visitor" pour construire l'arbre de fichiers et effectuer les commandes. Mais finalement, nous n'avons pas les implémentés et nous sauvegarde qu'une `rootPath` dans la modèle. L'arbre est construit dans la vue.

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 fichier pour exécuter
- `public void execute();` // exécute la commande
- `public void clear();` // remettre le résultat
- `public String getName();` // get the name of the command
- `public StringProperty resultStringProperty();` // pour update la vue les résultats
- `public BooleanProperty disabledProperty();` // pour update la vue pour disable le bouton

P3)

Nous avons créé une classe "Controller", qui a un attribut `private ArrayList<ICommand> commandsList`, pour stocker tous les commandes;

On a créé 3 commandes basiques :

- `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 ajoutons deux méthode à l'interface ICommand :

- public void setFile(File f); // quand on set file dans les command, on set une disabledProperty
- public BooleanProperty disabledProperty(); // pour update la vue pour disable le button

la deuxième méthode permet d'observer un booléen 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.

