Guide du développeur

LOG8430: TP2

Par:

Guillaume Rivest (1438615)

Clément Duffau (1699323)

Mohammed Benbachir (1484785)

Présentation du guide du développeur

Ce guide est à l'attention des développeurs. Il y est expliqué les choix de d'architecture et l'implémentation.

Présentation du logiciel

Le logiciel propose un outils de sélection de fichier ou dossier et 3 commandes applicables sur ce fichier ou dossier. Certaines de ces commandes sont applicables sur un fichier ou dossier, d'autres exclusivement sur un des deux types précédents.

L'interface proposée permet de lancer ces 3 commandes à l'aide des 3 boutons ainsi que voir leurs effets. Suite à la sélection d'un fichier, un panneau indique dans le cas d'un fichier, son nom tandis que pour un dossier, la liste des fichiers ou dossiers qu'il contient

3 commandes sont fournies à titre d'exemple, elles sont chargées dynamiquement, il est donc possible d'en ajouter ou enlever. Les commandes se trouvent dans le dossier « commandes » à la racine du projet

Explication de l’architecture

L’architecture choisie pour la partie métier de l'application est basée sur un patron de conception commande. Le choix a été fait de privilégier ce schéma de conception car il permet d'avoir une commande générique qui agrège un invocateur chargé de récupérer les paramètres des commandes et le modèle de données manipuler. Ne reste plus qu'à étendre cette interface pour créer des commandes concrètes et leurs actions.

L'architecture choisie pour la partie interface homme-machine est basée sur le patron de conception MVC. Le choix a été fait de privilégier ce schéma de conception car il permet de bien séparer les différentes partie de l'IHM. Une partie pour le modèle de données (chargement dans un base de données, structures de données, …). Une autre partie qui gère la vue en total indépendance du modèle. Un

contrôleur qui manipule et créer les interaction avec le modèle de données et la vue.

Pour implémenter le chargement dynamique, une structure de données CommandeDynamic a été créé et permet de stocker les informations. Ceci permet l'exécution d'une commande dynamiquement. Toutes les commandes situées dans le dossier « commandes »doivent hérité de CommandeFichier et implémenter sa propre méthode executer()

Après chaque exécution d'une commande, il était demandé de mettre à jour les commandes disponibles. Pour se faire, nous avons mis en place un patron Observer-Observable, ceci permet lors de la fin de l'exécution d'une commande de notifier notre chargeur dynamique qu'un rafraîchissement est demandé. Celui-ci recharge les classes et demande la mise à jour de la vue. Ce schéma de conception semble le plus approprié car il permet essaiment de rajouter une vue tout en ne perdant aucun fonctionnalité et avec un coût de développement faible.

Description sommaire des classes

« Invocateur » est responsable des entrés utilisateurs.

« Commande » est l'interface qui définit les méthodes génériques d'une commande. Ici, on se restreint à executer() mais on peut très bien imaginé rajouter desexecuter() et estReversible() pour étendre les fonctionnalités du logiciel.

« CommandeFichier » est une classe abstraite chargé de représenter une commande générique sur un fichier ou dossier. Elle sert de modèle de données dans notre cas

« CommandeVide1 », « CommandeVide2 » et « CommandeVide3 » représente les 3 commandes de notre système.

« CommandeFrame » est la vue de notre application

« CommandeManager » est le controleur de notre application qui charge dynamique ment les commandes

« CommandeDynamic » est le modèle de données pour le chargement dynamique de classe

« CommandeView » est le modèle de données pour le vue d'une commande (bouton d'exécution et label de résultat)

Diagramme de classes

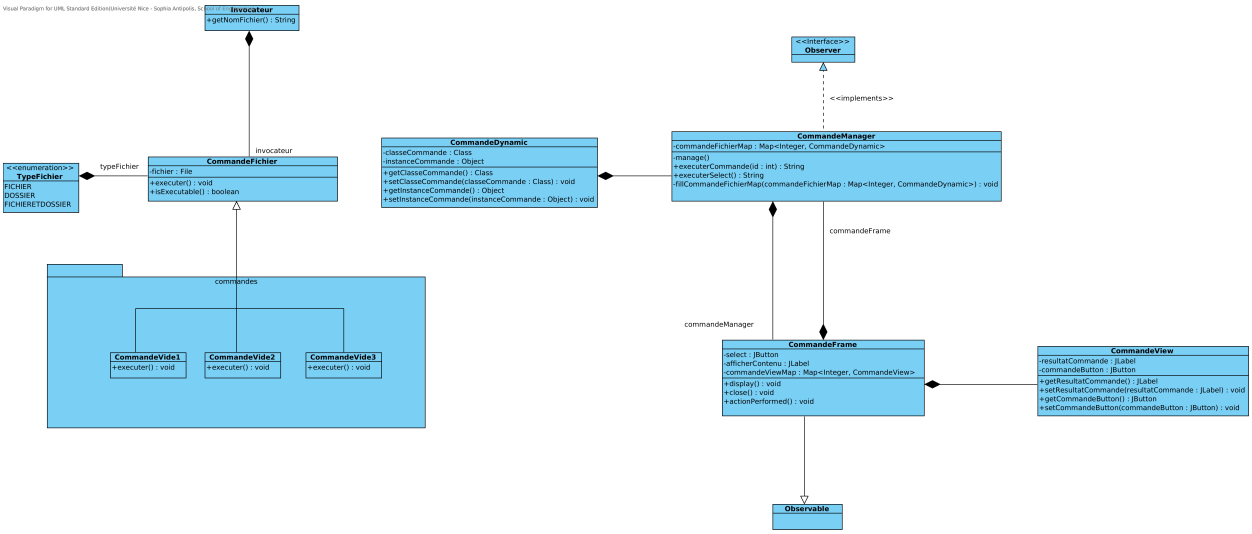

Illustration 1: Diagramme de classe du TP2

(Pour plus de lisibilité, le diagramme est aussi disponible dans l'archive au format png)