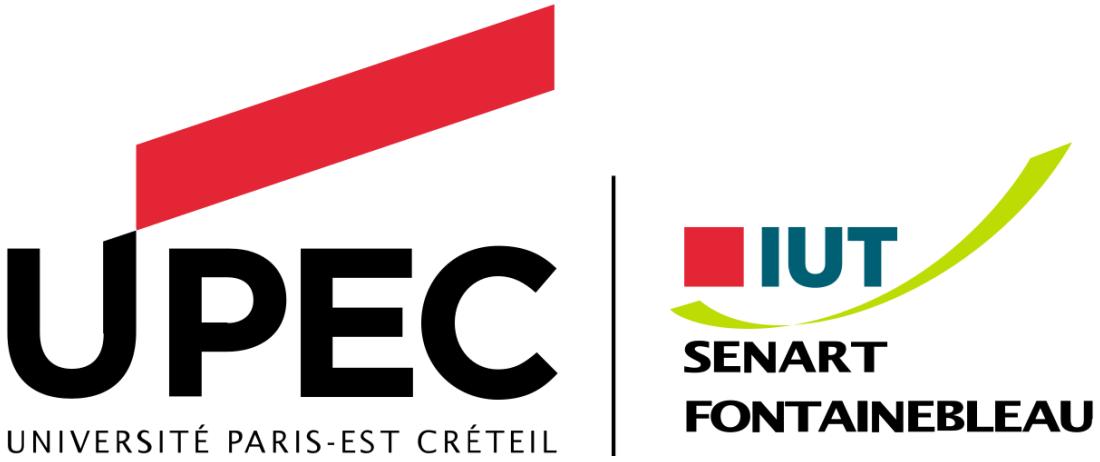
15/01/2023

SAÉ 3.2 : Inspecteur de JSON



Département informatique

Formation : BUT2 FI

Rapport de projet

Réalisé par :
Youcef SAYAD
Yanis BOUARROUDJ
Joffrey FOUCHE

Responsable de projet :
Luc HERNANDEZ

Sommaire

Introduction

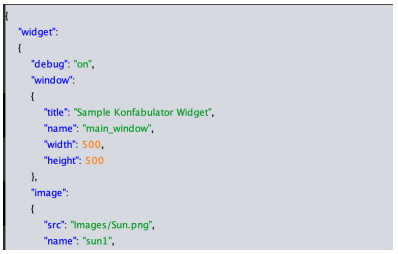
Le but de ce projet a été de programmer une application à l'aide du langage de programmation Java, l'application a pour but de représenter les données d'un fichier json dans une interface très simple pour faciliter l'analyse de ces données.
Notre programme est composé de deux parties.
La première partie est le modèle. On lit dans cette partie le fichier json et on construit un arbre avec les clés, les valeurs et les ouverture et fermeture d'objets et de tableau. On applique ensuite quelques filtres pour obtenir la chaine dans le format que l'on souhaite.
La seconde partie est la vue.
Celle-ci crée une interface dans laquelle on peut voir le fichier passé à la ligne de commande. De plus on peut rafraichir la page pour réafficher le texte. On peut tout déplier, tout replier ou switcher entre le mode PHP et Json. On peut également ouvrir un nouveau fichier en parcourant nos dossiers pour choisir le bon fichier.

Nous avons fait au mieux pour séparer les deux parties pour éviter qu'elles soient trop dépendantes l'une de l'autre et nous avons essayé de répartir le travail pour finir le projet.

Fonctionnalités

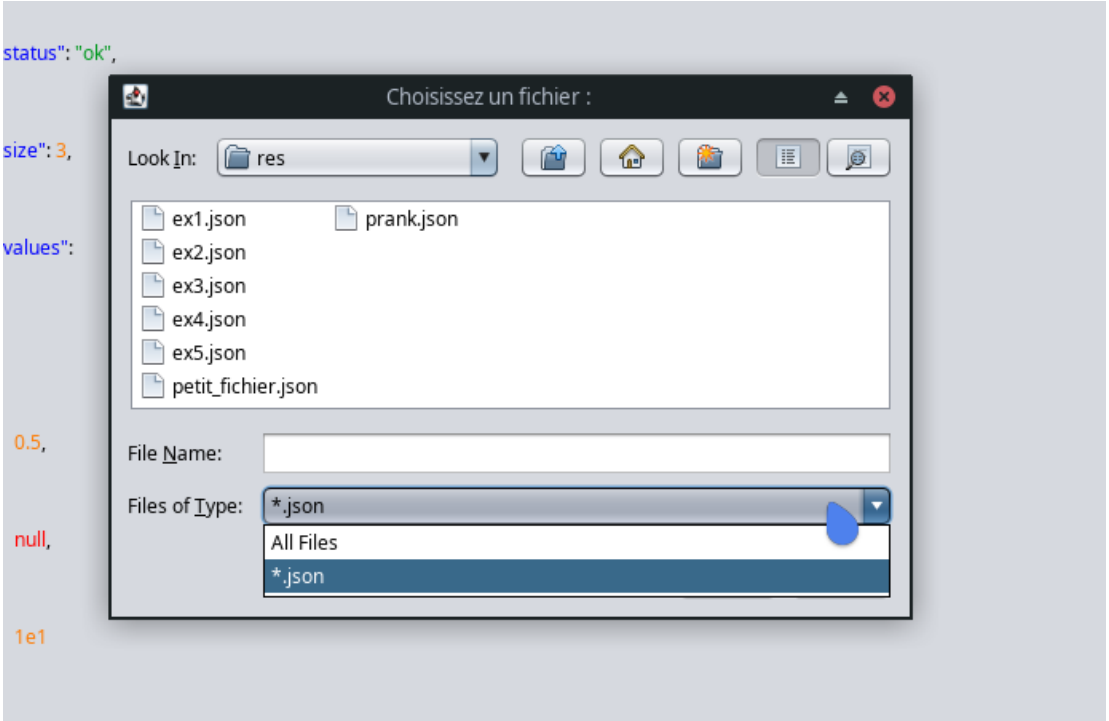
- Un bouton permettant de déplier tous les noeuds

- Un bouton permettant de plier tout les noeuds



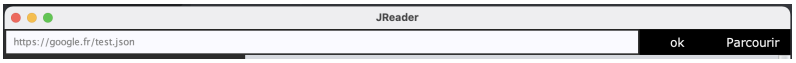
- Un bouton qui permet l'affichage au format PHP

- Il est possible de rechercher un fichier via un explorateur de fichier lorsque l'on clique sur le bouton parcourir



- Rechercher un fichier en tapant simplement son chemin dans le champ de texte en haut de page

- Rechercher un fichier en ligne en saisissant son lien dans le champ de texte en haut de page



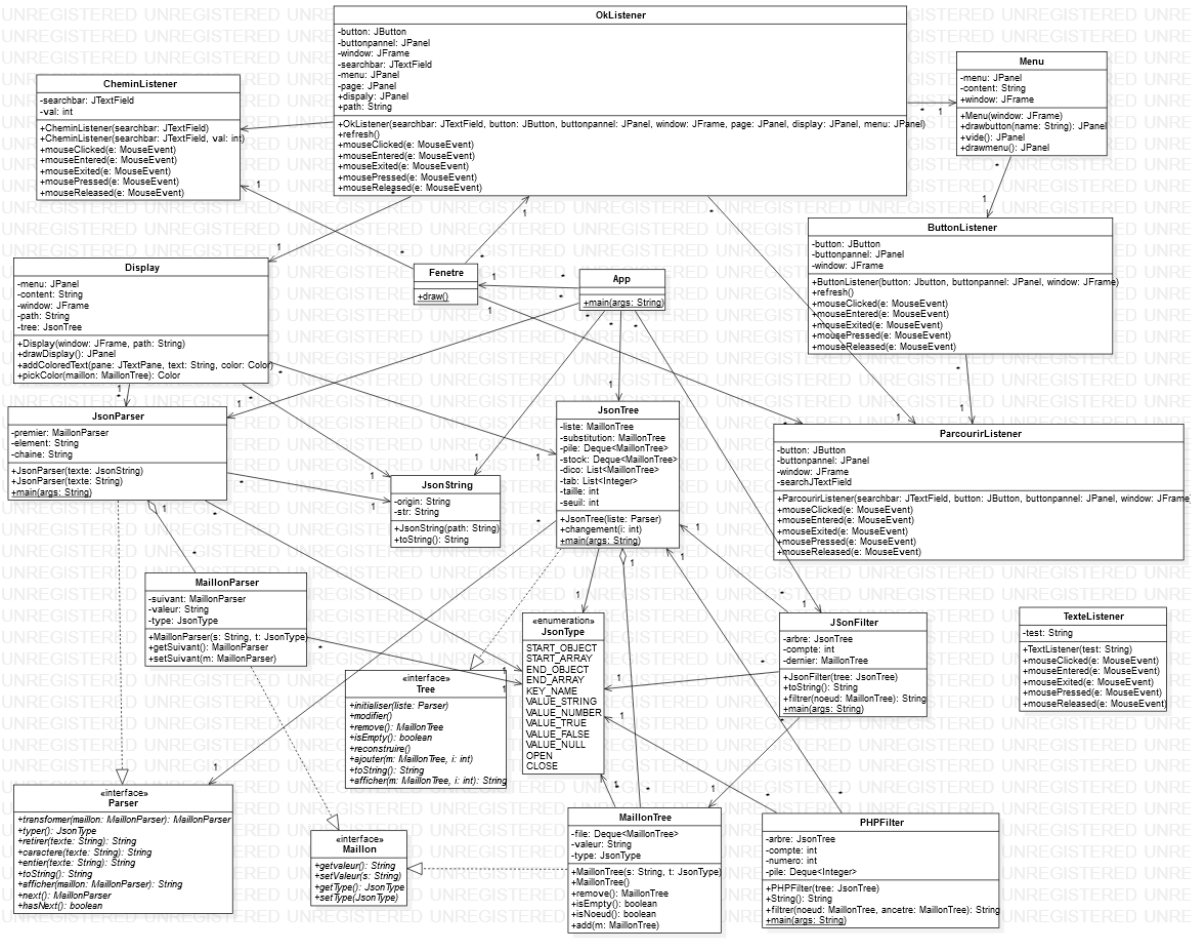
Structure du programme

Lorsqu'un fichier doit être ouvert, on vérifie qu'il existe puis on récupère l'intégralité du contenu en une chaîne de caractères. Ensuite on transforme les différents éléments de cette chaine en maillon d'une liste. La liste s'appelle JsonParser car elle s'inspire de l'interface du même nom mais ne l'implémente pas. On donne des types différents aux éléments de cette liste en fonction de leur contenu.
Les différents types sont :
-Les noms de clés,
-Les valeurs représentant une chaîne de caractère,
-Les valeurs représentant un nombre,
-Les valeurs représentant des booléens(true,false),
-Les valeur nulles,
-Les ouvertures d'objets,
-Les fermetures d'objets,
-Les ouvertures de tableaux,
-Les fermetures de tableaux.
Les virgules et les deux points ne sont pas pris en compte dans l'arbre car ils n'ont pas de type dans le JsonParser.
On transforme ensuite cette liste en un arbre.
Cet arbre est plus détaillé dans la partie "Arbre Syntaxe". Il récupère les éléments de la liste et crée un arbre avec.
Ensuite on peut utiliser deux filtres pour manier l'arbre.
Le premier est le filtre Json. L'objectif de ce filtre sera d'obtenir un visuel comme dans un éditeur de texte. On a rajouté les deux points, les virgules, les espaces après les deux points, les retour à la ligne après les virgules et les ouvertures de tableaux et d'objets et l'indentation après chaque retour à la ligne.
Le deuxième est le filtre PHP. L'objectif de ce filtre est d'afficher le Json comme la fonction print_r dans l'affichage PHP. Chaque tableau et chaque objet est remplacé par des parenthèses avec le mot "Array" devant, les deux points sont remplacés par des flèches. On ajoute des valeurs pour chaque élément d'un tableau et on entoure ces valeurs de crochets. Les noms des clés sont également entourés de crochets. Il y n'y a plus de virgules mais les retours à la ligne et les indentations sont les mêmes.
La liste et l'arbre sont composés de maillons qui prennent un type et une valeur. On peut récupérer ou modifier ce type et cette valeur quand on le veut.

Du côté de la vue, l'affichage est géré via différents panel, un pour la recherche qui contient un JTextfield pour le champ de texte et 2 JButton un qui ouvre un explorateur de fichier et l'autre pour valider l'URL saisi dans le champ de texte, un autre panneau est utilisé pour le menu latéral contenant 4 boutons un permettant l'affichage du fichier au format php un pour plier les noeuds, un pour déplier les noeuds et enfin un bouton qui permet de quitter l'application, et pour finir il y a un panel principal qui gère la conversion de l'arbre en éléments JLabel. Cette conversion s'effectue de la même manière que le filtre et applique un Listener sur les objets et listes pour pouvoir les déplier ou replier.

Diagramme de classe

Voici le diagramme de classe de notre programme :



Arbre de syntaxe

L'arbre de syntaxe est construit dans la partie Modèle. Les éléments de l'arbre restent dans le même ordre que ceux du fichier Json mais peuvent se trouver à des niveaux différents. En effet, chaque tableau et chaque objet se transforme en un nœud dans l'arbre. Chaque nœud d'un objet a pour première valeur une accolade ouvrante("{") et se termine par une accolade fermante("}"). Chaque nœud d'un tableau a pour première valeur un crochet ouvrant("[") et se termine par un crochet fermant("]"). On a rajouté deux types dans les types de Json. Ce sont des types pour les nœuds pour savoir s'ils sont ouverts(OPEN) ou fermés (CLOSE).
Pour voir comment se construit un arbre on peut lancer la classe JsonTree. Cela va nous afficher les éléments avec leur types à côté ainsi qu'une indentation pour voir la structure de l'arbre. Par exemple pour lancer l'ex5.json on fait les commandes suivantes :
javac -cp build -d build src/fr/iutfbleau/projetJson/ainspecteur/Modele/*.java
java -cp build src/fr/iutfbleau/projetJson/ainspecteur/Modele/JsonTree "file:res/ex5.json"

On obtient :

```
{       START_OBJECT
"status"        KEY_NAME
"\"ok\"" VALUE_STRING
"size"  KEY_NAME
3       VALUE_NUMBER
"values"        KEY_NAME
[...]
"object"        KEY_NAME
{...}
}       END_OBJECT
```

Sans changement, par défaut le premier noeud est ouvert et les autres sont fermés
Cet exemple représente la visualisation de l'intérieur de l'arbre, ce n'est absolument pas cela qui sera fournit à la partie Vue pour voir la partie concrète on peut lancer :

Pour le filtre Json :
java -cp build src/fr/iutfbleau/projetJson/ainspecteur/Modele/JsonFilter "file:res/ex5.json"

```
{
        "status": "\"ok\"",
        "size": 3,
        "values":
        [...],
        "object":
        {...}
}
```

Pour le filtre PHP :
java -cp build src/fr/iutfbleau/projetJson/ainspecteur/Modele/PHPFilter "file:res/ex5.json"

```
Array
(
```

```
        [status] => \"ok\"
        [size] => 3
        [values] => Array
        (
        [1] => Array
                (
                [1] => Array
                (
                        [1] => 0.5
                        [2] => null
                        [3] => 1e1
                )
                [2] => 0.5
                [3] => null
                [4] => 1e1
                )
        [2] => Array
                (
                [1] => 0.5
                [2] => null
                [3] => 1e1
                )
        [3] => Array
                (
                [status] => \"ok\"
                [size] => 3
                [values] => Array
                (
                        [1] => 0.5
                        [2] => null
                        [3] => 1e1
                )
                )
        [4] => 0.5
        [5] => null
        [6] => 1e1
        )
        [object] => Array
        (
        [object] => Array
                (
                [object] => Array
                (
                        [status] => \"ok\"
                )
                )
        )
)
```

Structures de données abstraites

Les structures de données que nous avons utilisé sont de plusieurs types, il y a celles que l'on a créé puis d'autres que l'on utilise de classes de la javadoc. Celle que l'on a créé sont une liste qui créé des maillons et un arbre qui s'occupe de l'agencement de la syntaxe.

En plus de cela nous avons appelé des structures de données existantes. Deux piles sont utilisés dans la classe JsonTree, la première sert à récupérer les noeuds antérieurs pour mettre les bonnes valeurs au bon endroit. La seconde est similaire à la première mais est utile lorsque l'on reconstruit l'arbre avec la valeur substitution. Il y a également une pile dans le PHPFilter.java qui récupère les numéro des tableaux en cours d'affichage.

Conclusions

Youcef :
J’ai trouvé cette SAE très pertinente car on a l’habitude de coder des mini-jeux et non des applications concrètes, cette fois-ci c’est différent on a vraiment eu l’impression de créer un programme que l'on aurait pu reproduire en entreprise.
De plus le travail de groupe a été très important car les notions abordé n’étaient pas forcément les plus simples et le fait de s’entraider nous a permis de se débloquer les uns les autres et donc d'avancer beaucoup plus rapidement

Yanis :
Je reprends les propos de Youcef j'ai trouvé ce projet très concret aussi,
de plus il est complémentaire avec le projet IHM qui avait aussi pour but de créer une application et non un jeu comme on a l'habitude de le faire, m'ayant occupé principalement de la partie graphique j'ai appliqué les méthodes utilisées dans le projet IHM, cela appuie sur le fait que ce genre de travail nous apporte beaucoup de connaissances pour la suite !

Joffrey :
J’ai apprécié faire cette SAE, pour une fois que j’arrive à rendre un travail qui me semble totalement fini. Mes deux partenaires ont été géniaux dans leur travail. Cependant cela aurait pu être bien plus complexe, nous avons commencé le travail trop tard selon moi et j’avais peur de ne pas pouvoir rendre ce travail. De plus, une vision différente des choses peut nous amener à des désaccords les premières semaines.Mais nous avons fait des compromis pour pouvoir rendre notre travail pour le projet.