Rapport - SAE32$_2$022

Bilal Boudjemline & Romain Besson

16 janvier 2023

Résumé

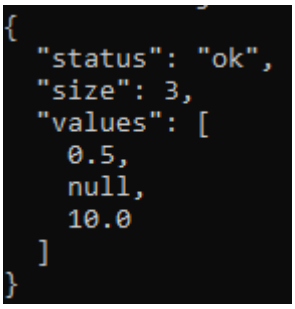
Ce projet avait pour but de parser une ligne JSON avec la documentation officiel de Java pour en faire un affichage avec de la coloration syntaxique, avec une simplification visuelle permettant de fermer des tableaux et objets. La fin du projet nous demandait de convertir la ligne parse en tableau PHP. Tout ça bien sur a l'aide des automates présents sur ce site.

1 Fonctionnalités

Commencons par l'execution. Le programme se lance avec 0 ou 1 argument. Quand il est lancé avec 1 argument, un affichage console se fera. Si il n'y a pas d'arguement, une interface graphique se lancera et vous pourrez y entrer le chemin vers le fichier json désiré.

Le parseur entame son travail en, si ce n'est pas fait, mettre une seul ligne le fichier json selectionné. Il procede ensuite a une verification qui genere l'exception **JsonSyntaxException** si la syntaxe du fichier json n'est pas correcte. Il va ensuite stocker les clés valeurs présente dans le fichier a l'aide d'un conteneur **Tree** et de noeux **Node** contenu dans **Storage**. Les types sont bien entendu detecté et converti par les énumerations contenues dans **Type** contenu dans le meme package.

Passons maitenant a la version sans interface graphique. Comme dit plus haut, il faut mettre un argument qui correspond au chemin du fichier json desiré. Une fois fais, le code avec les bonnes indentation, les bons saut de lignes apparaitra dans la console.



Pour la partie graphique, aucun argument n'est nécessaire. Cette partie embarque avec elle de la coloration syntaxique, un pliment-dépliment des conteneurs de donnée et une traduction de l'objet en tableau PHP.

2 Structure

La structure du programme est simple et visuel, l'exception concocté par nos soins se trouve dans **Exception/**. Les parametres tel que les colorations par type sont stocke dans **Parameters/**. Les codes correspondant aux stockages relatif dans notre AST (**Tree**) dans **Storage/**.Les elements graphiques comme **Frame** dans **Graphics/**.

Le diagramme de classe se trouve dans le pdf : **DiagramJSonInspector.pdf**.

3 AST (Arbre de Syntaxe Abstraite)

Cet arbre **Tree** est composé de noeux **Node** qui sont eux meme composé valeurs **Value** de type d'une des enumerations de **Type**.

Pour la partie graphique, l'arbre est construit a l'aide de la fonction *validationAction(JTextField)* contenue dans **Frame**.

Pour la partie non-graphique, l'arbre est directement construit dans le **Main**.

3.1 Construction de l'AST

— Creation de l'AST via l'arbre **Tree**

```
Tree ast = new Tree(Fichier\_JSON)
```

— L'AST s'occupe de creer l'arbre entier a l'aide d'une de ces fonctions recurssives *parseElement(Fichier_JSON_en_une_ligne_string)*

```
parseElement(fichier_toString)
```

— La boucle contenue dans *parseElement(fichier_toString)* s'occupe de creer des noeuds qui seront stocké dans la variable

```
Node element
```

— Les types sont definis par les fonctions appeller conditionnelement par *whichType()*

```
parseArray() pour les tableaux
parseObject() pour les objets
parsePair() pour le reste
```

— Pour finir, *parseElement()* retourne un noeud qui contient tous les autres et l'affecte a l'attribut de **Tree**.

```
private Node firstNode;
```

— Exemple :

```
String fichier_en_string = {"groupe":1,"iut":"Fontainebleau","nbEleve":60"}
Tree ast = new Tree(fichier_en_string);

/* ici, ast.parseElement(fichier_en_string) vas creer
    "List elements" qui s'initialise avec splitList(fichier_en_string) :
    elements.get(0) = "groupe":1
    elements.get(1) = "iut":"Fontainebleau"
    elements.get(2) = "groupe":60

    puis la boucle contenue dans ast.parseElement(fichier_en_string) va appeller
```

```
        ast.whichType(element_dune_iteration)
        pour y determiner le type et renvoyer un noeud
        pour qu'il soit enregistré dans le noeud "element"
        de la fonction parseElement(fichier_en_string)
        a l'aide de element.add(le_prochain_noeud)
    */
```

3.2 Structures abstraites utilisées

La seul structure abstraite qu'on a utilisé est l'interface List qui instancie ArrayList

4 Conclusion

Ca a été un projet incroyablement enrichissant pour notre culture en Java. Le travail d'equipe et l'ensemble de nos connaissances en Java ont été réuni dans ce projet pour réalisé cette application.

4.1 Bilal Boudjemline

J'ai trouvé ce projet intéressant car il nous a demandé de mobilisé toutes les connaissances qu'on a eu a ce jour. N'aimant pas les travaux de groupe en général, je me suis beaucoup amusé a faire ce projet.

4.2 Romain Besson

J'ai trouvé ce projet intéressant car dans un premier tant c'est la première fois que je créé un parser de fichier et dans un deuxième tant je trouve que ce projet mobilise tous les concept vu en cours comme les arbres ou la généricité

5 Usage

Se positionner dans la racine du repertoire et faire la commande **make run** (n'oubliez pas de **make clean** avant de refaire un **make run**)

6 Annexes

— Repertoire du projet

— Automates du JSON

FIN