PATRONS ET MODÈLES

TP2 : Introspection

UQAC

LEYE Mame Ramatoulaye

RAHARIJAONA Anja

29/02/2020

Exercice 1 :

Dans cet exercice, nous avons décidé de créer 3 classes pour créer un encodeur Json. Nous citons :

MyJson : cette classe contient 2 fonctions principales : *convertToDictionnary()* qui fabrique le dictionnaire et *formatDictionnary()* qui s'occupe de la conversion des données en Json et l'afficher.

Ensuite, on différencie 3 cas : les objets primitifs (Int, String,...) , les objets et les listes qui ont le même contenu que les IEnumerables.

Pour gérer les listes et les tableaux, on a décidé d'appeler la fonction de manière récursive pour gérer les objets et les sous objets.

ObjetQuelconque et SousObjetQuelconque : sont des classes sont dépendantes l'une de l'autre, c'est à dire que la classe SousObjetQuelconque possède parmis ses attributs un type sousObjet. Cette dernière peut contenir n'importe quel attribut. Notre programme peut donc stocker dans un dictionnaire et afficher sous forme d'un json tous les attributs d'un objet quelconque.

Exercice 2 :

Notre architecture se décompose en 3 parties essentielles :

- ***Capteurs***

Nous y créons nos capteurs à l'aide du patron Factory. On y retrouve également notre gestionnaire de capteurs qui va permettre l'ajout, suppression, mise à jour de ces derniers ainsi que l'historique de nos convertisseurs existants à l'aide de dictionnaires.

La méthode *changeSystemeImperial*() va nous permettre de simuler un changement d'unités dans la vraie vie et ainsi d'introduire les convertisseurs.

- ***Conversion***

Nous y retrouvons notre classe Convertisseur qui est défini par la classe Attribute ConvertisseurAttribute.

- ***Visualisation***

Ici, nous allons pouvoir créer les représentations de nos capteurs à l'aide du pattern Builder.

Par souci de design, les énumérations des unités et des types n'ont pas été ajoutées dans l'architecture UML. Nous avons personnalisé 2 classes d'attributs pour la conversion et pour la création d'un capteur. A l'aide de la réflexion, nous pouvons accéder à ces derniers et ainsi construire les visualisations et conversions nécessaires.

Nous avons pensé à modifier l'attribut lors de la création d'un convertisseur durant le runTime() , dans le cas où on changerait les unités impériales pendant l'exécution. Cependant, les attributs sont des méta-données qui sont chargées au moment de la compilation et sont donc fixées. C'est une perspective intéressante dont nous pourrions évaluer avec les possibilités que nous offre C#.