Examen de C# - Juin 2018

Calendau GUQUET

Le projet sera réalisé seul ou en binôme, chez vous, et correspond à un temps de travail d'environ une demi-journée.

Vous rendrez ce projet par mail (cguquet@synexie.fr et copie à Gregory DELAFOSSE gregory.delafosse@yncrea.fr). Le projet devra être publié sur un repository public de votre choix au format Git (GitHub, GitLab…), et vous indiquerez simplement l'url du repo dans le mail.

Vous publierez des tags git pour marquer le rendu correspondant à chaque question.

Question 1 (1 pt)

En utilisant l'interface en ligne de commande (CLI) dotnet, vous devez créer la structure de projet suivante :

- Racine : dossier Isen.<VotreNom> et ce dossier contient un fichier solution (sln) nommé Isen.<VotreNom>.sln
- Dossier Isen.<VotreNom>.Library, avec un projet de type librairie de classes. Ce projet sera ajouté au sln.
- Dossier Isen.<VotreNom>.ConsoleApp, avec un projet de type console. Ce projet sera ajouté au sln. En outre, ce projet devra référencer le projet Library
- Le SDK cible sera netstandard2.0 (soit la valeur par défaut).

Question 2 (3 pts)

Dans le projet Library, définissez une classe Node. Cette classe devra représenter un arbre de données, et aura donc au minimum les propriétés suivantes :

- Value (string)
- Id (Guid, autogénéré)
- Parent (Node)
- Children (liste de Node)
- Depth (lecture seule). La profondeur correspond au nombre d'ancêtre du node. Le node racine a donc une profondeur de 0.

Extrayez l'ensemble de ces propriétés dans une interface INode (et Node devra implémenter cette interface).

Implémentez l'égalité de 2 nodes (interface IEquatable<Node>), la définition de l'égalité étant que les id et value soient égaux. Vous pouvez utiliser le code généré par Resharper ou Rider.

Question 3 (3 pts)

Définissez et implémentez les méthodes suivantes (dans l'interface et dans la classe)

- void AddChildNode(Node node) – ajoute un enfant à un Node. Cette méthode devra au passage mettre à jour le champ Parent du Node ajouté.
- Void AddNodes(IEnumerable<Node> nodeList) – ajoute une liste d'enfants à un node.
- void RemoveChildNode(Guid id) - retire de la liste des enfants le node ayant l'id donné en paramètre.

- void RemoveChildNode(Node node) – retire un enfant à un Node. L'identification du Node à retirer sera fait au moyen de la méthode Equals (qui a été surchargée à la question 2).

Question 4 (4 pts)

Définissez et implémentez la méthode « Node FindTraversing(Guid id) ». Cette méthode devra rechercher un Node, selon son id en traversant l'arbre, c'est-à-dire en recherchant dans les enfants, puis les enfants des enfants, etc…, en descendant à partir de l'item sur lequel elle est appelée. Elle renvoie le node trouvé, ou une valeur nulle.

Cette méthode sera également présente dans l'interface.

Implémentez également la méthode Node FindTraversing(Node node), qui fonctionne de façon similaire, mais en utilisant Node.Equals (toujours celui surchargé en question 2).

Question 5(3 pts)

Surchargez la méthode ToString() afin d'afficher un élément et tous ses enfants de la façon suivante :

```
Item1 {00000000-0000-0000-0000-000000000000}
|-Item11 {00000000-0000-0000-0000-000000000000}
|-|-Item111 {00000000-0000-0000-0000-000000000000}
|-|-|-Item1111 {00000000-0000-0000-0000-000000000000}
|-|-Item112 {00000000-0000-0000-0000-000000000000}
|-|-|-Item1121 {00000000-0000-0000-0000-000000000000}
|-|-|-Item1122 {00000000-0000-0000-0000-000000000000}
|-|-Item113 {00000000-0000-0000-0000-000000000000}
|-Item12 {00000000-0000-0000-0000-000000000000}
|-Item13 {00000000-0000-0000-0000-000000000000}
```

L'indentation, correspondant au niveau du Node, est représentée par le nombre de séquences « pipe tiret » affichées. Egalement, l'ordre d'affichage doit se faire branche après branche, de façon à afficher clairement la vue en arbre. Enfin, après la value doit apparaitre l'id, au format indiqué dans le screenshot ci-dessus.

Cette méthode étant appelée sur l'élément racine (qui est ici « item1 »), il ne peut donc y avoir qu'une seule racine (peu importe si cette racine a des parents).

Question 6 (3 pts)

Refactoring : transformez Node et INode afin que le paramètre devienne générique. La classe devient donc Node<T> et l'interface, INode<T>.

Question 7 (3 pts)

Implémentez la sérialisation et désérialisation en JSON d'un Node (et ses enfants) en utilisant la librairie Newtonsoft.Json.