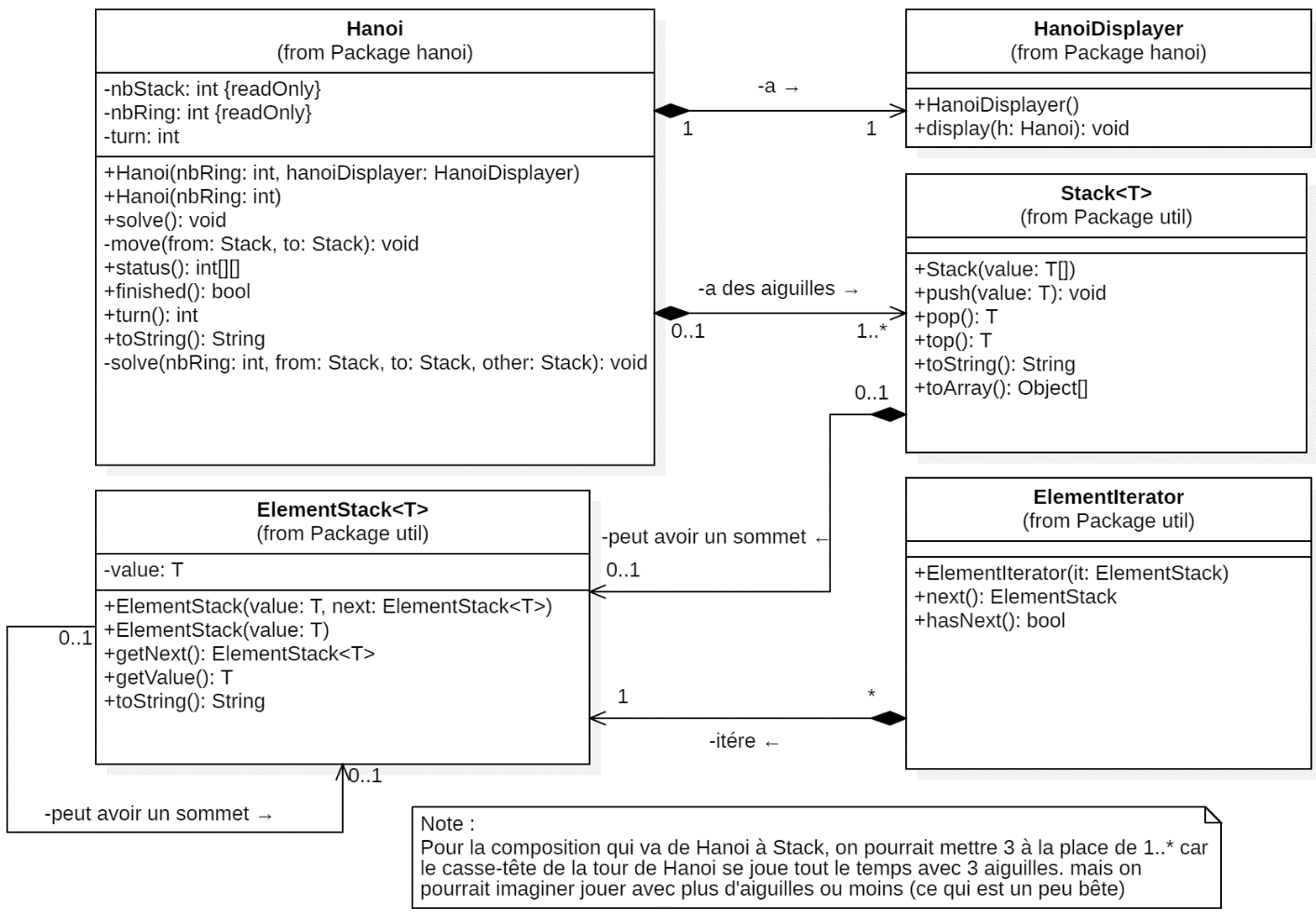
1. UML

2) Description des classes

2.1) Hanoi

La classe Hanoi est la classe qui permet de résoudre le casse-tête des tours de Hanoï. Cette classe possède deux constructeurs. Le premier permet de créer un objet de type Hanoi en spécifiant uniquement le nombre de disques. Le deuxième constructeur permet de créer un objet en spécifiant le nombre de disque ainsi que le HanoiDisplayer qui va se charger d’afficher la résolution du casse-tête à l’aide des outils fournit avec la consigne du laboratoire.

Concernant l’algorithme utilisé, nous nous somme inspiré de l’algorithme récursif que nous avions vu durant nos cours d’ASD (pour plus d’information, voir le chapitre « 3) Description de l’algorithme »).

2.2) HanoiDisplayer

Cette classe est responsable de l’affichage de la résolution du casse-tête. Elle possède uniquement une fonction display.

2.3) Stack

Afin de représenter les aiguilles ainsi que les disques, nous avions du implémenter une Stack. En réalité, cette structure de données ressemble plus à une liste chaînée qu’à une pile. Toutefois, le comportement de la classe Stack respecte le comportement d’une pile standard. En outre, cette classe permet d’instantier une pile.

Une fois la pile créer on peut insérer un élément au sommet (empiler), enlever l’élément se situant au sommet (dépiler), accéder à la valeur de l’élément se situant au sommet. De plus, on peut également générer un tableau d’objet à l’aide d’une Pile. Ce tableau reflèlte l’état d’une aiguilles du casse-tête.

Cette classe est générique

2.4) ElementStack

Cette classe permet de modéliser les données qui sont dans la pile. Cette classe est générique.

2.5) ElementIterator

La classe ElementIterator permet de modéliser des itérateur permettant d’itérer sur les ElementStack de notre Stack. Elle implémente la méthode next et hasNext.

La méthode next permet de retourner l’élément suivant l’itérateur et de faire pointer l’itérateur sur ce dernier. Concernant la méthode hasNext, elle contrôle si l’ElementStack sur lequel l’itérateur pointe est suivi d’un autre ElementStack.

2.6) TestHanoi

Cette classe est une classe de test. En effet, dans cette dernière nous avons réalisé diverse opération afin de valider le fonctionnement de notre classe Hanoi. Nous vérifions que les objets se construisent correctement. Nous contrôlons également si notre algorithme effectue un nombre minimal de déplacement de disque et si le flag « finished » se met correctement à jour.

Pour rappel, le nombre minimal de déplacement se calcul de la façon suivante :

|  |  |  |
| --- | --- | --- |
| Test | Résultat attendu | Résultat obtenu |
| Construction d’objet hanoi avec 3 disque | Aucune erreur |  |
| Construction d’objet hanoi avec 3 disque et displayer | Aucune erreur |  |
| Construction d’objet hanoi avec 0 disque | Erreur |  |
| Construction d’objet hanoi avec -3 disque | Erreur |  |
| Construction d’objet hanoi avec 3 disque et un displayer null | Erreur |  |
| Test de résolution des tours d’hanoi avec 3 disque | Nb de tours requis = 7  Finished = true |  |

2.7) TestStack

Cette classe nous permet de tester le bon fonctionnement de notre Stack. Nous nous assurons également du bon fonctionnement de la classe ElementIterator. Concernant la Stack, nous nous assurons que les méthodes pop, push, top fonctionnent bien. De plus, nous contrôlons également si la stack se construit correctement. La génération du tableau d’objet à partir d’une stack est également controlée.

Concernant les tests effectués pour la classe ElementIterator, nous avions tout simplement fait pointer un itérateur sur le sommet d’une pile. Ensuite nous retournons la valeur de l’élément pointé. Une fois cela fait, nous itérons sur la pile tant que l’élément pointé possède un ElementStack qui le suit.

|  |  |  |
| --- | --- | --- |
| Test | Résultat attendu | Résultat obtenu |
| Construction d’un stack et affichage de celle-ci | Affichage d’un stack vide |  |
| Pop sur une stack vide | Erreur |  |
| Top sur une stack vide | Erreur |  |
| Création d’un iterateur sur une stack vide | Erreur |  |
| On ajoute dans une stack vide les valeurs suivantes : 7, 14, 95, 33 et 444, puis affichage | [ <444> <33> <95> <14> <7> ] |  |
| Dépiler 2 fois sur la stack précédente | [ <95> <14> <7> ] |  |
| On ajoute 23 à la stack précédente | [ <23> <95> <14> <7> ] |  |
| Dépiler 2 fois sur la stack précédente, puis affichage du sommet | 14 |  |
| Affichage de la stack précédente | [ <14> <7> ] |  |
| Construction d'une autre stack contenant (10, 20, 30, 40, 50) | [ <50> <40> <30> <20> <10> ] |  |
| Transformation de la stack en tableau d’objet, puis affichage du tableau | [ <50> <40> <30> <20> <10> ] |  |
| Création d’un itérateur qui pointe sur le sommet de la stack précédente, puis affichage de son contenu | 50 |  |
| Boucle avec l’itérateur, hasNext et .next() | Affichage de toute la pile dans le bon ordre |  |

3) Description de l’algorithme

L’algorithme utilisé dans notre programme pour résoudre les tours d’Hanoi est l’algorithme que nous avons étudié durant le cours d’Algorithme et structure de données. C’est un algorithme récursif qui a comme cas général de déplacer les N-1 disques (N étant le nombre de disques) sur l’aiguille du milieu et comme cas trivial de déplacer le dernier disque sur l’aiguille à droite.

Pseudo-code :

Solve(int n, aigulle to, aigulle from, Stack other)

Si n == 1 Alors

On déplace le disque de from à to

Return

Solve(n-1, from, other, to)

On déplace le disque de from à to

Solve(n-1, other, to, from)

4) Justification des choix

4.1) Hanoi

4.2) Stack

- La fonction getHead() ne sert à rien. Elle a été implémenté uniquement afin de pouvoir effectuer un test complet de la classe stack.

4.3) ElementStack

Nous avons décidé de mettre la visibilité de cette classe en package car elle doit être utilisé uniquement par la pile et l’itérateur.

4.1) ElementIterator

- La fonction toString() ne sert à rien. Elle a été implémenté uniquement afin de pouvoir effectuer un test complet de la classe Stack.

- Le constructeur de cette classe devrait être en mode package car un ElementIterator doit pouvoir itérer uniquement sur les ElementStack qui sont les éléments qui peuvent être stocké dans la pile (classe Stack). Cependant, nous avons décidé de laisser la visibilité du constructeur en public afin de pouvoir instancier un ElementIterator dans la classe de TestStack afin de pouvoir vérifier son bon fonctionnement.

5) Réponse à la question

On sait que les moines effectuent un nombre minimal de mouvement et qu’il y a 64 disques à déplacer. Ils doivent donc effectuer 264 – 1 mouvements.

Sachant que le monde a actuellement 13.8 milliards d’années, le temps restant à l’univers est :