Université de Yaoundé I
Département d'Informatique

Master 1

INF4067 : UML et Design Patterns

2023-2024

Fiche de TD1

Valéry Monthe

Exercice 1 :

Les bibliothèques Java proposent une classe **java : :util : :Random** qui permet de produire des nombres pseudo-aléatoires. Les principales méthodes de la classe Random sont les suivantes :

- Random() : constructeur par défaut
- Random(long seed) : constructeur avec spécification de graine
- Foat nextFloat() : retourne le prochain nombre de la séquence pseudo-aléatoire, avec une distribution uniforme entre 0.0 et 1.0.

Les nombres générés ne sont jamais réellement aléatoires mais sont produits par une fonction complexe produisant des nombres difficilement prévisibles. La graine (seed) joue ici un rôle particulier : elle permet de donner une valeur initialisant la fonction en question. Plusieurs instances de **java : :util : :Random** donneront les mêmes suites de nombres aléatoires pourvu que la graine soit la même. Spécifier une graine permet ainsi de reproduire le comportement d'un système même lorsqu'il utilise le « hasard ». Pour plus de commodité dans la reproduction du comportement d'un système, on cherche souvent à garantir de n'avoir qu'une seule instance générant des nombres aléatoires, cette instance pouvant être initialisée avec une graine.

1. Quel patron de conception peut-il être utilisé pour gérer cette contrainte ?
2. Donnez sa structure générique
3. Utilisez le patron de conception de la question 1 pour proposer le code d'une classe garantissant l'unicité d'une telle instance. Ce code doit contenir une partie montrant l'utilisation de notre classe.

Exercice 2 :

Un éditeur de jeux développe un jeu permettant aux enfants de connaître les animaux. Les enfants peuvent, en particulier, apprendre la forme et le cri des animaux parmi lesquels le chat et la vache. Le chat est modélisé par la classe LeChat possédant au moins les deux méthodes formeChat() et criChat() et la vache est modélisée par la classe LaVache possédant les deux méthodes criVache() et formeVache(). Comme le montrent les noms des méthodes, la première spécification de ce jeu est propre aux animaux modélisés. L'éditeur souhaite améliorer ce jeu en créant une interface commune à tous les animaux qui lui permette d'en ajouter de nouveaux, sans modifier l'interface avec le code client, et d'utiliser le polymorphisme dans la gestion des animaux (manipuler des troupeaux hétéroclites...).

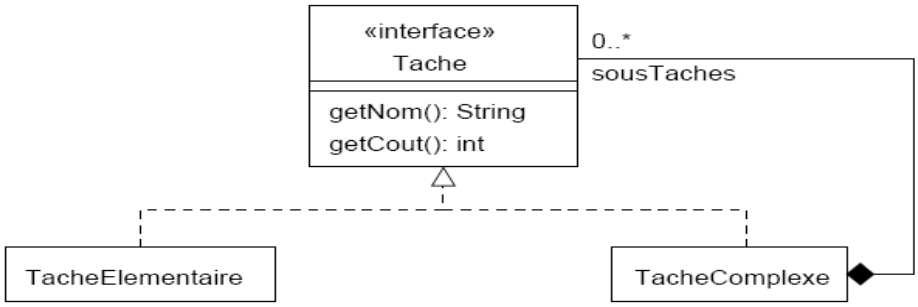
1. Proposez une modélisation des classes pour cette nouvelle version du jeu en faisant apparaître le client. Les classes seront Chat, Vache et non plus LeChat, LaVache.

On souhaite réutiliser tout le code développé dans la version précédente (avec LeChat, LaVache...) dans le nouveau logiciel utilisant les nouvelles classes (Chat, Vache...). Pour cela, on aimerait incorporer les anciennes méthodes pour éviter de les réécrire.

2. Quel patron de conception permet de résoudre ce problème ?
3. Donnez sa structure générique
4. Proposez une modélisation.

Exercice 3:

L'architecture des tâches est donnée à la figure 1 où le détail des classes ***TacheElementaire*** et ***TacheComplexe*** n'est pas donné. Une tâche est caractérisée par un nom et un coût. Une tâche est soit une tâche élémentaire, soit une tâche complexe qui est alors composée de sous-tâches. Il est ainsi possible d'ajouter une sous-tâche à une tâche complexe, ajouter(Tache) ou de supprimer une sous-tâche, supprimer(Tache). Le coût d'une tâche complexe est la somme des coûts des tâches qui la composent.



1. Indiquer le ou les patrons de conception utilisés dans cette architecture.
2. Le listing ci dessous donne le code de l'interface ***Tache***. Écrire en Java la classe ***TacheElementaire*** qui est une réalisation de l'interface Tache.

```
public interface Tache {
/** Obtenir le nom de la tâche. */
String getNom();
 /** Obtenir le coût de la tâche. */
 int getCout();
}
```

3. Indiquer quel est le coût de la tâche tA (résultat affiché à l'écran) construite comme indiqué dans le listing ci dessous.

```
 public class TestTache1
{
  public static void main(String[] args)
{
        TacheComplexe tA = new TacheComplexe("A");
        tA.ajouter(new TacheElementaire("A1", 10));
        tA.ajouter(new TacheElementaire("A2", 20));

        System.out.println("Cout de tA = " + tA.getCout());
 }
 }
```

Exercice 4 :

Une figure simple peut être un point, une ligne ou un cercle. Une figure peut être composée d'autres figures, simples ou elles-mêmes composées d'autres figures. Toutes les figures peuvent être dessinées ou translatées.

1. Quel patron de conception peut-il être utilisé pour modéliser cette situation ?
2. Donnez sa structure générique
3. Utilisez le patron de conception de la question 1 pour proposer une solution à ce problème.