UE Conception Oriente Objet

Devoir Surveillé

durée : 3h – documents écrits autorisés
livres et calculatrice interdits
Dictionnaire de langue (électronique ou non) autorisé pour les étrangers

Les exercices sont indépendants.
La clarté des réponses sera prise en compte dans l'évaluation. Evidemment vos réponses devront être convaincantes et donc précises !

Exercice 1 : Musique maestro

Un `Orchestre` est composé d'`Instruments`. Il est possible d'*ajouter* un instrument à un orchestre ou d'*obtenir la liste* des instruments qui le commposent.
Parmi les instruments on distingue différentes familles : les `Clavier`, dont le `Piano` ou le `Synthetiseur` sont des exemples, les `Corde`, comme le `Violon` ou la `GuitareElectrique`, les `Vent`, les `Percussion`.
Les instruments à vent se décomposent eux mêmes en `Cuivre` (`Trompette`, etc.) et en `Bois` (`Hautbois`, `Saxophone`, etc.).
Ces instruments ont des comportements communs : on peut tous les *jouer* ou les *accorder*. Mais chaque famille ou instrument d'une famille a également ses propres caractéristiques telles que le *nombre de cordes*, le *nombre de touches* du clavier, etc.
Parmi ces instruments certains sont `Electrique`, c'est le cas du `Synthetiseur` et de la `GuitareElectrique`. Enfin, certains sont des `InstrumentPhilharmonique`, c'est-à-dire qu'on les trouve dans un orchestre philharmonique, c'est le cas des `Trompettes`, `Violons` et `Hautbois`.

Q 1 . Chacun des mots apparaissant en `police courrier` dans le texte précédent correspond à une entité : classe, classe abstraite ou interface. Chaque élément apparaissant en *police italique* est une fonctionnalité qui peut se traduire par une méthode et, éventuellement, un attribut.

En faisant les choix de conception qui vous paraissent les mieux adaptés donnez sous la forme d'un diagramme UML faisant apparaître **tous** ces éléments (c'est-à-dire **tous** ceux en `courrier` ou en *italique* dans le texte ci-dessus) ainsi que les dépendances d'héritage ou d'implémentation entre ces entités.

Il est **inutile de détailler plus** les diagrammes et de faire apparaître d'autres éléments (attributs ou méthodes) que ceux demandés.

Exercice 2 : Documents

On s'intéresse dans cette exercice à la représentation de documents structurés.
On considère ici des documents structurés en sections, sous-sections et paragraphes. La structuration est hiérarchique. Ainsi les documents sont structurés en sections, elles-mêmes structurées en sous-sections, elles-mêmes structurées en paragraphes.
Tous les éléments de la structure sont caractérisés par un contenu.
L'élément de base de tels documents est le *paragraphe*. Son contenu est un texte que l'on peut représenter simplement par une chaîne de caractères.
Les autres éléments sont des structures composées : la sous-section, la section et le document lui-même. Chacun de ces éléments possède un titre et son contenu est constitué de plusieurs autres éléments. Comme indiqué précédemment la nature de ces éléments varie selon les cas, il s'agit :

- de sections pour les documents,
- de sous-sections pour les sections,
- de paragraphes pour les sous-sections.

Il est possible d'obtenir un itérateur sur ces éléments, ces objets seront donc du type `Iterable` (voir la documentation en annnexe).
Les documents possèdent en plus un auteur (une chaîne de caractères) et un résumé qui est un paragraphe.
Un exemple de tel document est proposé en annexe.

Q 1 . Utilisez au mieux l'héritage, les interfaces et les types génériques pour proposer un ensemble de types permettant de modéliser de tels documents.

Vous donnerez votre réponse sous la forme d'un diagramme UML détaillé (héritage/implémentation, visibilité et type des attributs, des paramètres de méthodes, des types de retour des méthodes, etc.).

Q 2 . Donnez le code java de la classe permettant de réprésenter les documents ainsi que le code java de tous ses super-types éventuels (`Object` et `Iterable` exceptés).

Exercice 3 : Personnages de jeu vidéo

Pour les besoins d'un jeu vidéo en mode texte et au tour par tour on a besoin de représenter les personnages qui y interviennent. Il s'agit à la fois du joueur[1] et des personnages du jeu. Les premiers sont appelés *personnages joueurs* et les seconds *personnages non joueurs*.
Chaque personnage est caractérisé par :

- son nom,
- un niveau d'énergie,
- la liste de ses capacités, c'est-à-dire des actions qu'il est capable d'effectuer dans le jeu.

Les personnages sont tous de type `Personnage`. Les capacités des personnages sont modélisées par des objets de type `Action` dont le code est fourni en annexe. Les actions possibles sont diverses et variées : *manger*, *ne rien faire*, *dormir*, *ramasser* (un objet), *attaquer* (un autre personnage), etc. On peut considérer que les actions gèrent dans la méthode `execute()` l'affichage correspondant aux conséquences de leur exécution.

On donne le diagramme de la classe jeu suivant :

Jeu
- lesPerso : List<Personnage>
+ joue() + estFini():boolean

Q 1 . Compléter ce schéma UML pour que cette classe soit conforme au design pattern *singleton*.

S'agissant d'un jeu au tour par tour, le principe est qu'à chaque tour de jeu chacun des joueurs exécute une action. Le code de la méthode `joue()` est donc :

```
public void joue() {
  while (! this.estFini()) {
     Iterator<Personnage> it_perso = this.lesPerso.iterator();
     while (it_perso.hasNext()) {
       Personnage perso = it.next();
       // le personnage agit : il choisit une action puis l'exécute
       perso.agit();
       // si le personnage est mort on le retire du jeu
       if (perso.getEnergie() < 0) {
         it_perso.remove() ;
       }
     }
  }
}
```

La méthode `agit()` du type personnage a donc pour rôle de réaliser le choix de l'action parmi celles possibles, puis d'effectuer cette action choisie. Dans le cas des personnages non joueurs ce choix est géré par le jeu. On peut dans une première approche considérer qu'il s'agit d'un choix aléatoire. Dans le cas des personnages joueurs le choix de l'action est évidemment réalisé par le joueur : un menu proposant les actions possibles est affiché et soumis au joueur qui choisit parmi les propositions. On peut par exemple utiliser la classe `ListChoser` étudiée au TP n^{o}2, son code est rappelé en annexe.

Q 2 . Sous forme de diagrammes UML détaillés, faites la proposition complète qui vous semble la plus pertinente pour gérer les personnages, qu'ils soient "joueurs" ou "non joueurs", conformément à ce qui est décrit ci-dessus. Les personnages doivent donc pouvoir agir comme indiqué et votre solution doit faire apparaître clairement comme vous gérez cela.

Si vous le jugez utile, vous pouvez accompagner vos diagrammes d'un texte **court et précis** présentant votre solution.

Q 3 . Donnez le code java correspondant à votre proposition.

(Inutile de développer au delà de ce qui est présenté dans le sujet, en particulier aucune classe de type `Action` n'est demandée).

Q 4 . Afin d'améliorer le comportement des personnages non joueurs on décide de modifier comment ils choisissent leur action (en mettant en place un programme d'intelligence artificielle pour effectuer ce choix par exemple).

Indiquez comment vous prendriez en compte cet ajout. Vous pouvez répondre en complétant le diagramme UML précédent et/ou par un texte (toujours **court et précis**).

Aucun code java n'est demandé.

[1] ou même des joueurs.

Annexe

`java.lang.Iterable<T>`

```
public interface Iterable<T>

Implementing this interface allows an object to be the target of the
"foreach" statement.

Iterator<T> iterator()

    Returns an iterator over a set of elements of type T.
    Returns:
        an Iterator.
```

Exemple de document structuré.

Titre du document

Timoléon

Résumé. *Ceci est le résumé du document écrit par Timoléon qui en est donc l'auteur et dont le titre apparaît ci-dessus. Ci-dessous vous trouverez les sections et leur contenu, c'est-à-dire les sous-sections elles-mêmes composées de paragraphes.*

1. **Titre Première section**
   (a) *Titre Première sous-section*
   - Je suis le premier paragraphe de la première sous-section de la première section.
   - Je suis le second paragraphe de la première sous-section de la première section.

   (b) *Titre Seconde sous-section*
   - Je suis le premier paragraphe de la seconde sous-section de la première section.
   - Je suis le second paragraphe de la seconde sous-section de la première section.

2. **Titre de la seconde section**
   (a) *Titre de sous-section*
   - Premier paragraphe de la première sous-section de la seconde section. Il n'y a qu'un seul paragraphe et une seule sous-section dans ce cas.

Le type `Action`

```
public interface Action {
  /** indique si l'action est possible ou non en fonction de la
  * situation courante du jeu
  * @return true si et seulement si l'action est possible à cet instant
  */
  public boolean estPossible();
  /** fournit une chaîne de caractères représentant le nom pour cette action
  * @return le nom de cette action
  */
  public String toString();
  /** le personnage passé en paramètre exécute cette action
  * @param p le personnage qui exécute l'action
  */
  public void execute(Personnage p);
}
```

La classe `ListChoser` du TP2

En supposant la classe `scanner.TestScanner` du TP2 également fournie.

```
package generics;
import java.util.*;

public class ListChoser {

    public <T> T chose(String msg, List<T> list) {
       if (list.isEmpty()) {
           return null;
       }
       System.out.println(msg);
       System.out.println("      0 - none");
       int index = 1;
       for (Object item : list) {
           System.out.println("      " + (index++) + " - " + item);
       }
       System.out.println("            choice ?");
       int choice = scanner.TestScanner.saisieEntier(list.size()+1);

       if (choice == 0) {
           return null;
       } else {
           return list.get(choice-1);
       }
    }
}
```