TP n°3 : UML, patrons de conception et exceptions

Un petit jeu

Aujourd'hui le but est de construire un jeu où des monstres s'affrontent sans aucune interaction avec l'utilisateur.

Partie 1 : Conception

Notre jeu s'appuiera sur les quatre patrons :

- *Singleton* : l'arène ;
- *Factory* : pour créer les monstres ;
- *Observer* : l'interface utilisateur ;
- *Decorator* : l'équipement des monstres.

Lisez le reste du sujet et décrivez à l'aide d'un diagramme UML l'architecture de votre code. Ce diagramme peut être fait à la main et dans ce cas une photo de dessin devra être dans le rendu.

Dans le sujet il y a des idées de structures mais elles ne sont pas imposées. Elles ne sont pas optimales mais plus facile à gérer.

Partie 2 : La base

On a besoin d'une arène (`Arene`). Elle doit contenir une méthode `void combat()` qui est la boucle principale. Un combat ne peut avoir lieu que s'il y a exactement deux monstres dans l'arène. On doit pouvoir

- n'avoir qu'une seule instance d'`Arene` ;
- y ajouter un monstre ;
- ajouter un observateur à tous les monstres dans l'arène.

Partie 3 : Les monstres

Création des classes

Il y aura (au moins) deux types de monstre :

- un dragon ;
- un professeur.

Chacun doit pouvoir indiquer le nombre de PV restants ainsi les dégâts qu'il peut infliger.
Ils doivent également pouvoir subir des dégâts.
Les personnages doivent être créés à l'aide d'une `MonstreFactory`.

Voici une idée de structure :

- une interface `Monstre` ;
- une classe abstraite `MonstreBase` qui l'implémente ;
- une classe `Dragon` et une classe `Professeur` qui l'étendent.

Tests

À ce moment vous pouvez créer une `Arene` et y ajouter des monstres.

Partie 4 : Interaction avec l'interface utilisateur

Création des classes

On veut que l'interface soit mise à jour lorsqu'un personnage subit des dégâts.
Pour cela nos personnages devront connaître une liste d'observateurs.
Créez une interface `Observateur` qui contiendra toutes les méthodes qui vous semblent utiles.

Créez une classe `Terminal` qui sera un observateur du combat et affichera les informations dans le terminal.

Tests
À ce moment vous pouvez faire subir des dégâts aux monstres dans l'arène et ils doivent alors notifier cela au `Terminal`.

Partie 5 : Les attaques

On veut maintenant que nos personnages puissent attaquer.
Cependant un personnage mort ne peut pas attaquer (sauf les zombies si vous voulez les ajouter).
Pour gérer cela nous allons indiquer que la méthode permettant d'attaquer doit renvoyer une `MortException` (que vous devez créer) si un personnage mort essaye d'attaquer.

Partie 6 : Améliorations

Création des classes

Nos personnages doivent pouvoir avoir des équipements. N'importe quel type de personnage doit pouvoir avoir n'importe quel type d'équipement.

Appuyez vous sur le *decorator pattern* pour pouvoir donner des améliorations aux personnages. Par exemple un personnage avec une épée pourra frapper deux fois plus fort.

Voici une idée de structure :
- une classe abstraite `MonstreDecorateur` qui implémente `Monstre` ;
- une classe `MonstreAvecEpee` qui étend `MonstreDecorateur`.

Tests
On doit maintenant pouvoir
- créer une arène ;
- y ajouter des monstres (potentiellement améliorés);
- y « connecter » un terminal.

Partie 7 : Le combat

Ajoutez à la classe `Arene` le combat entre les monstres.

Partie 8 : La fin

Voici un exemple de code.

```
public static void main(String[] args) {
    Arene arene = Arene.getInstance();
    MonstreFactory f = new MonstreFactory();
    Observateur terminal = new Terminal();

    Monstre m1 = f.creerMonstre("Dragon avec une épée", 100, 10);
    Monstre m2 = f.creerMonstre("Professeur avec un casque", 200, 20);
    arene.ajouteMonstre(m1);
    arene.ajouteMonstre(m2);

    arene.ajouteObservateurs(terminal);
    arene.combat();
  }
```

et un résultat associé (il y a des fonctionnalités supplémentaires) :

```
Le Dragon (id 1) attaque
Le Professeur (id 2) subit
Le Professeur (id 2) attaque
Le Dragon (id 1) subit
Le Professeur (id 2) gagne
```

Ce qui est affiché précisément n'est pas imposé mais il doit y avoir un affichage des événements.