Fantaisie Travaux Pratiques n°3

« Comparaison d'objet, refactoring, les vues, TreeSet»

Ce TP poursuit le TP2 sur les monstres. Afin d'avoir toujours une version stable par sujet de TP :

- Ouvrir Eclipse, sélectionner le même workspace que pour le premier TP (par exemple : Z:\Licence3\Semestre1\POO2\workspace).
- Copier le projet fantaisieTP2 (en le sélectionnant puis ctrl+c),
- Coller le projet (ctrl+v), et le renommer fantaisieTP3.

Dans ce TP vous travaillerez donc uniquement dans ce nouveau projet.

I. Comparaison des Monstres – *ordre naturel, ordre imposé, refactoring*

1. Comparaison d'objets, ordre naturel

Les êtres vivants sont rangés naturellement dans l'ordre alphabétique de leur nom (dans le TP précédant nous avons considéré que 2 êtres vivants étaient égaux s'ils avaient le même nom). Implémenter l'ordre naturel dans la classe « EtreVivant ».

Dans la classe « AideEcrivain » implémenter la méthode *ordreNaturelMonstre* qui récupère l'itérateur sur les monstres du camp des monstres et retourne la concaténation des noms de ces monstres triés dans l'ordre alphabétique et séparés par une virgule. Tester dans la classe « TestGestionProtagonistes ».

```
System.out.println("\n**** TP3 ****");
System.out.println("\nordre naturel :\n"
    + aideEcrivain.ordreNaturelMonstre());
```

Résultat attendu :

```
**** TP3 ****

ordre naturel :
dragotenebre, guillotimort, marinsangant, vampirien
```

2. Comparaison d'objets, ordre imposé, refactoring

Toujours dans la classe « AideEcrivain » créer l'attribut monstresDomaineSet qui contiendra un ensemble de monstres triés selon leur domaine (feu, glace ou tranchant) puis selon leur ordre naturel.

Créer la méthode *updateMonstresDomaine* qui ajoute un à un les monstres du camp des monstres à l'ensemble monstresDomaineSet.

Implémenter la méthode *ordreMonstreDomaine* qui retourne une chaîne contenant les noms des monstres structurés comme suit: le nom du

domaine, un saut de ligne, les noms des monstres par ordre alphabétique séparés par une virgule puis le domaine suivant …

Rappel :

- l'ordre naturel des énumérés est l'ordre dans lequel les éléments sont donnés.
- Si votre code dépasse 20 lignes faites un refactoring.

Tester.

```
System.out.println("\nordre par domaine :\n"
    + aideEcrivain.ordreMonstreDomaine());
```

Résultat attendu :

```
ordre par domaine :
FEU :
dragotenebre,
GLACE :
marinsangant,
TRANCHANT :
guillotimort, vampirien
```

3. Comparaison d'objets, ordre imposé, refactoring

Toujours dans la classe « AideEcrivain » créer l'attribut monstresZoneSet qui contiendra un ensemble ordonné de monstres triés selon leur zone de combat (aérien, aquatique ou terrestre) puis selon leur force et enfin selon l'ordre naturel.

Créer la méthode *updateMonstresZone* qui ajoute un à un les monstres du camp des monstres à l'ensemble monstresZoneSet.

Implémenter la méthode *ordreMonstreZone* qui retourne une chaîne contenant les noms des monstres et structurée comme suit : le nom de la zone de combat, un saut de ligne, les noms des monstres, le caractère ":", leurs forces séparées par une virgule puis la zone de combat suivante …

Rappel :

- l'ordre naturel des énumérés est l'ordre dans lequel les éléments sont donnés.
- Si votre code dépasse 20 lignes faite un refactoring.

Tester.

```
System.out.println("\nordre par zone de combat :\n"
    + aideEcrivain.ordreMonstreZone());
```

Résultat attendu :

```
ordre par zone de combat :
AERIEN :
dragotenebre : 200, vampirien : 10,
AQUATIQUE :
marinsangant : 150,
TERRESTRE :
guillotimort : 80
```

II. Vues sur les ensembles de Monstres – *vue, objet dégénéré*

1. *Travail préliminaire*

Toujours dans la classe « AideEcrivain », créer les attributs :
- monstresDeFeu,
- monstresDeGlace,
- monstesTranchants,

comme étant tous des attributs de type « NavigableSet ».

Créer des getters sur chacun d'entre eux.

Il s'agit de créer trois vues sur la collection monstresDomaineSet, ne donnant que les monstres d'un domaine particulier. Pour cela nous allons coder deux solutions (cf. paragraphe 2 et 3).

2. *Première solution – vue sur un ensemble*

L'idée est de repérer le premier monstre de glace, et d'initialiser la vue monstresDeFeu comme le sous-ensemble de la collection monstresDomaineSet allant de la tête de l'ensemble au premier monstre de glace.

Ecrire la méthode *firstMonstreDomaine* qui retourne le premier monstre de l'ensemble monstresDomaineSet dont le domaine est celui donné en paramètre d'entrée de la fonction.

Ecrire la méthode *initMonstresDeFeu* qui initialise la vue monstresDeFeu.

Tester en initialisant la vue, puis en l'affichant. La méthode static *affichageMonstre* permet d'afficher le nom et le domaine du monstre.

```
aideEcrivain.initMonstresDeFeu();
NavigableSet<Monstre<?>> monstres = aideEcrivain.getMonstresDeFeu();
String affichage = affichageMonstres(monstres);
System.out.println("\nmonstres de feu :\n" + affichage);
```

Résultat attendu :

```
monstres de feu :
dragotenebre monstre de FEU
```

Ajouter les monstres `soufflemort` et `craombre`, puis afficher de nouveau la vue.

```
Monstre<Glace> soufflemort = new Monstre<>("soufflemort", 120,
      ZoneDeCombat.AERIEN, Domaine.GLACE, new Tornade(8));
Monstre<Feu> cramombre = new Monstre<>("cramombre", 80,
      ZoneDeCombat.TERRESTRE, Domaine.FEU, new BouleDeFeu(2), new Lave(1),
      new Eclair(1));
soufflemort.rejointBataille(bataille);
cramombre.rejointBataille(bataille);
monstres = aideEcrivain.getMonstresDeFeu();
affichage = affichageMonstres(monstres);
System.out.println("\nmonstres de feu :\n" + affichage);
```

Résultat attendu :

```
monstres de feu :
cramombre monstre de FEU, dragotenebre monstre de FEU
```


A priori si vous ne vous êtes pas trompé, cela fonctionne. Mais que se passe-t-il si vous ajoutez le monstre `givrogolem` puis que vous affichez de nouveau la vue ?

```
Monstre<Glace> givrogolem = new Monstre<>("givrogolem", 200,
      ZoneDeCombat.TERRESTRE, Domaine.GLACE, new PicsDeGlace(10),
      new Tornade(1));
givrogolem.rejointBataille(bataille);

monstres = aideEcrivain.getMonstresDeFeu();
affichage = affichageMonstres(monstres);
System.out.println("\nmonstres de feu :\n" + affichage);
```

Résultat obtenu (pas forcément attendu !) :

```
monstres de feu :
cramombre monstre de FEU, dragotenebre monstre de FEU, givrogolem monstre
de GLACE
```

Pour rester indépendant du premier monstre du domaine suivant nous allons passer à la deuxième solution en utilisant un objet dégénéré.

3. *Deuxième solution – vue sur un ensemble, objet dégénéré*

Mettre en commentaire les méthodes *firstMonstreDomaine* et *initMonstresDeFeu*. Pour cela, avec Eclipse, sélectionner le code de la méthode puis appuyer sur les touches : ctrl + Maj (↑) + M. Vous pouvez utiliser la même manipulation pour décommenter le bloc.

Réécrire la méthode *initMonstresDeFeu* en utilisant cette fois un objet dégénéré.

Relancer le test, vous obtiendrez pour le dernier affichage :

```
monstres de feu :
cramombre monstre de FEU, dragotenebre monstre de FEU
```

Ecrire les méthodes *initMonstresDeGlace* et *initMonstresTranchant*.

Tester

```
aideEcrivain.initMonstresDeGlace();
aideEcrivain.initMonstresTranchant();

Monstre<Feu> aqualave = new Monstre<>("aqualave", 30,
      ZoneDeCombat.AQUATIQUE, Domaine.FEU, new Lave(5));
Monstre<Tranchant> requispectre = new Monstre<>("requispectre", 200,
      ZoneDeCombat.AQUATIQUE, Domaine.TRANCHANT, new Griffe());

aqualave.rejointBataille(bataille);
requispectre.rejointBataille(bataille);

monstres = aideEcrivain.getMonstresDeFeu();
affichage = affichageMonstres(monstres);
System.out.println("\nmonstres de feu :\n" + affichage);
```

Idem pour les monstres de glace et les monstres tranchants

Résultat attendu :

```
monstres de feu :
aqualave monstre de FEU, cramombre monstre de FEU, dragotenebre monstre de FEU

monstres de glace :
givrogolem monstre de GLACE, marinsangant monstre de GLACE, soufflemort monstre de GLACE

monstres tranchant :
guillotimort monstre de TRANCHANT, requispectre monstre de TRANCHANT, vampirien monstre
de TRANCHANT
```