Fantaisie Travaux Pratiques n°6

« Générique avancé, Exceptions, Java 8 : interfaces, trait, Lambda »

Ce TP poursuit le TP5 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 fantaisieTP5 (en le sélectionnant puis ctrl+c),
- Coller le projet (ctrl+v), et renommer le fantaisieTP6.

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

I. Généricité avancée

Restreindre le joker utilisé pour typer les monstres dans les classes :

- « GroupeHomme » pour la méthode *choixCampHomme* et sa classe interne « ComparateurArme »,
- Dans la classe « Bataille »,
- Dans la classe « Grotte » pour les méthodes *ajoutSalle* et *afficherPlanGrotte*,
- Dans la classe « Homme » pour la méthode *choisirArme*,
- Dans la classe « TestGestionProtagoniste ».

II. Les exceptions

1. *Exception personnalisée*

Créer dans le package « *protagoniste* » l'exception « ZoneDeCombatNonCompatibleException ».

Prendre en compte dans la classe « Grotte » la précondition suivante : un monstre ne peut être placé que dans une salle de même zone de combat.

Modifier la classe « TestGestionGrotte » en conséquence.

Tester votre exception en modifiant la ligne :

```
grotte.ajouterSalle(ZoneDeCombat.TERRESTRE, guillotimort);
```

par :

```
grotte.ajouterSalle(ZoneDeCombat.AQUATIQUE, guillotimort);
```

RESULTAT

```
protagoniste.ZoneDeCombatNonCompatibleException: La zone de combat de la salle et de type AQUATIQUE, alors que celle du monstre est TERRESTRE
        at bataille.Grotte.ajouterSalle(Grotte.java:30)
        at testsfonctionnels.TestGestionGrotte.main(TestGestionGrotte.java:59)
```

2. *Exception et entrées / sorties*

Travail préliminaire

Sous le package « *livre* » créer l'interface « Livre » proposant le service *écrire* avec en paramètre d'entrée une chaine.

Créer la classe « Ecran » qui implémente le service *ecrire* en affichant la chaine passée en paramètre sur la console.

Télécharger sous Moodle la classe « HistoireFantaisie » dans le package « *livre* ». Assurez-vous de ne pas avoir d'erreur de compilation. Si vous exécutez le main rien ne doit apparaitre dans la console.

Modifier la classe « HistoireFantaisie » en lui ajoutant l'attribut livre de type « Livre ». Cet attribut est initialisé par le constructeur. Dans le main modifier la création de l'instance `histoireFantaisie` en lui ajoutant une instance d'un livre.

Ajouter la méthode *afficherLesMonstres* qui écrit l'ensemble des monstres de la grotte (avec la méthode *ecrire* de l'attribut livre).
Dans le main ajouter un appel à cette méthode.

Résultat attendu :

```
guillotimort, aqualave, givrogolem, cramombre, dragotenebre, requispectre,
soufflemort, vampirien, marinsangant.
```

Ecrire dans un fichier avant Java 7

En vous aidant du cours 6, plus particulièrement de la diapo « Les exceptions & les entrées/sorties » ainsi que du TD, créer la classe « Fichier » qui est une autre implémentation de l'interface « *livre* ».
La méthode *ecrire* écrira le texte passé en paramètre d'entrée dans le fichier histoire.txt.

Ci-dessous le code de base à structurer correctement en respectant les bonnes pratiques.

```
String chemin = "./src/livre/histoire.txt";
File writer = new File(chemin);
FileWriter fichier = new FileWriter(writer, true);
fichier.write(texte);
fichier.close();
```

Aide le chemin : `"./src/livre/histoire.txt"` permet de créer le fichier directement dans le paquetage « *livre* ». Pour le visualiser cliquer à droite sur le package puis Refresh. Vous pouvez lire directement le fichier dans Eclipse. N'oubliez pas de le supprimer entre deux essais.

Ecrire dans un fichier à partir de Java 7

Mettre en commentaire votre code et le réécrire en utilisant le « try-with-resources » disponible depuis Java7.

III. Implémentation par défaut, trait, lambda (Java 8+)

Dans cette partie nous testerons simplement les exemples vus en cours.

1. *Les interfaces*

Reprendre l'interface « Livre » afin de définir, pour la méthode *ecrire*, un comportement par défaut : l'affichage sur la console.

Modifier la classe Ecran et tester.

2. *Les traits*

Implémenter l'interface « *Orderable* » vu en cours dans le package « attaque » :

```
public interface Orderable<T> extends Comparable<T> {

   default boolean isAfter(T other) {
      return compareTo(other) > 0;
   }

   default boolean isBefore(T other) {
      return compareTo(other) == 0;
   }

   default boolean isSameAs(T other) {
      return compareTo(other) == 0;
   }
}
```

Modifier dans la classe Arme `implements Comparable<Arme>` par : `implements Orderable<Arme>`.

Compléter la classe « TestGestionArme » avec le code ci-dessous :

```
System.out.println("\n\n****** TP6 *****");
LancePierre lancePierre = new LancePierre();
Boomerang boomerang = new Boomerang();
Arc arc1 = new Arc(3);
Arc arc2 = new Arc(1);
System.out.println("lance-pierre < boomerang ? "
      + lancePierre.isAfter(boomerang));
System.out.println("arc1 = arc2 ? " + arc1.isSameAs(arc2));
System.out.println("arc2 tire sa seule flèche est n'est plus
      opérationnel");
arc2.utiliser();
System.out.println("arc1 = arc2 ? " + arc1.isSameAs(arc2));
System.out.println("arc1 > arc2 ? " + arc1.isBefore(arc2));
```

Résultat attendu :

```
****** TP6 *****
lance-pierre < boomerang ? true
arc1 = arc2 ? true
arc2 tire sa seule flèche est n'est plus opérationnel
arc1 = arc2 ? false
arc1 > arc2 ? true
```

3. Utilisation de fonctions anonymes

Pour améliorer l'encapsulation dans la classe « GroupeHommes », remplacer la classe « ComparateurHommes » du TP5 par une implantation anonyme. Suivre le message de Sonar pour une implantation fonctionnelle de l'interface « Comparator ».

Implanter sous forme de pipeline la méthode « affichageMonstres » du TP3.