

Langage Orienté Objet / Java

Partie 3 – Spécialisation et classe abstraite

The Learning Souls Game

Ce TP doit être réalisé à partir du code créé dans la partie précédente.

Comme dans tout RPG qui se respecte, nous allons fournir à nos personnages un « habillage » qui leur permettra d'atténuer les dégâts qu'ils reçoivent.

Dans notre jeu, un **Hero** sera protégé par des pièces d'armure alors qu'un **Monster** sera protégé par l'épaisseur de sa peau. Nous allons dans un premier temps spécialiser nos 2 classes en conséquence, et créer les classes qui manquent (les armures).

A partir de ces bases, nous verrons comment l'approche **Objet** permet de mettre en place (de manière efficace et élégante) un mécanisme d'absorption des dégâts qui est à la fois générique (commun à tous les personnages), mais qui repose en même temps intégralement sur les spécificités de chacun de nos types de personnages (l'armure des héros, et la peau des monstres).

1. Monster

Une fois n'est pas coutume, nous allons tout d'abord nous occuper des monstres dont la protection correspond à une épaisseur de peau.

1.1. Créez l'attribut **skinThickness** :

 Privé, de type **float** et avec une valeur par défaut de **20**.

1.2. Créez les accesseurs en réfléchissant bien à leur visibilité (qui pourra éventuellement être modifiée plus tard...)

2. Armures

La protection de la classe Hero est un peu plus complexe puisqu'elle est constituée d'un ensemble de pièces d'armure. Le jeu proposera diverses pièces d'armures, chacune avec ses spécificités, mais toutes ces pièces auront une structure et des mécanismes communs. Comme vous en avez maintenant l'habitude, ces éléments communs vont être synthétisés dans une super classe que nous nommerons **ArmorItem**, puis que nous spécialiserons en diverses sous-classes pour forger des pièces particulières.

2.1. Créez la classe **lsg.armor.ArmorItem** avec les membres suivants :

- ✎ **name** : **String** contenant le nom de la pièce (fixé lors de l'instanciation), pour lequel on créera seulement un accesseur **public** en lecture.
- ✎ **armorValue** : **float** contenant la valeur d'armure de l'item, pour lequel on créera seulement un accesseur **public** en lecture.
- ✎ Un constructeur à 2 paramètres permettant de nommer et fixer la valeur d'armure d'une pièce lors de son instanciation.
- ✎ Une surcharge de la méthode **toString** qui renvoie une chaîne de la forme :

nom de la pièce(valeur d'armure)

2.2. Grâce à **ArmorItem**, créez les pièces d'armure **BlackWitchVeil**, **DragonSlayerLeggings**, et **RingedKnightArmor** dans le package **lsg.armor** et pour lesquelles le **toString()** hérité d'**ArmorItem** renvoie :

- ✎ Black Witch Veil(4.6)
- ✎ Dragon Smayer Leggings(10.2)
- ✎ Ringed Knight Armor(14.99)


3. Hero en armure

Maintenant que des pièces d'armure ont été forgées, nous allons équiper notre héros.

Un héros possède 3 emplacements de pièce d'armure (slots numérotés de 1 à 3). Les slots peuvent être vides. Nous allons implémenter tout cela sous la forme de tableau (attention aux indices).


3.1. Modifiez **Hero** avec les membres suivants :

- ✎ **armor** : un tableau destiné à recevoir des pièces d'armure et dont la taille sera fixée à **3** lors de l'instanciation d'un **Hero** grâce à la constante de classe entière **Hero.MAX_ARMOR_PIECES**.
- ✎ **setArmorItem** : une méthode à **2 paramètres** (pièce d'armure, slot concerné).
 - Si le numéro du slot est inférieur à **0**, ou supérieur à **MAX_ARMOR_PIECES**, la méthode ne fait rien !
 - ATTENTION: les slots sont numérotés à partir de 1, alors que les indices d'un tableau le sont à partir de 0 ! (et on ne veut pas de bidouille du genre : créer un tableau à 4 cases pour seulement 3 emplacements à utiliser...)
- ✎ **getTotalArmor** : retourne un **float** correspondant à la somme des valeurs d'armure des pièces.

 **armorToString** : renvoie une chaîne contenant une description formatée de l'armure totale portée par le héros (y compris les slots vides marqués « empty »).

Exemple : (NB : dans la console : tout est sur une seule ligne)

ARMOR	1:Black Witch Veil(4.6)	2:empty	3:Ringed Knight Armor(14.99)
TOTAL:	19.59		

 **getArmorItems** : renvoie un tableau d'**ArmorItem** contenant uniquement les pièces d'armure portées.

- la taille du tableau résultat correspond donc au nombre de cases différentes de **null** dans **armor**.
- le tableau résultat peut avoir une taille de 0, mais ne peut contenir des cases **null**.

3.2. Créez un **main** dans **Hero** (pour tester) : y instancier un héros, l'équiper, et afficher les statistiques de son armure (cf. exemple ci-avant).

4. Absorption de dégâts

Il est important de noter que si le mécanisme d'absorption de dégâts est bien commun à tous les personnages (donc partagé par tous les **Character**), la manière d'implémenter cette protection va différer d'un type de personnage à un autre. En effet, un héros sera protégé par les pièces d'armure qu'il va porter. Par contre, un monstre ne portera jamais d'armure, mais est naturellement protégé par sa peau plus ou moins épaisse.

Nous avons donc a priori un problème : le mécanisme d'absorption de dégâts lors de la réception d'une attaque est commun et doit donc être implémenté au niveau de **Character** (par transformation de **getHitWith**), mais le mécanisme permettant de calculer le taux d'absorption est lié aux sous-classes, et ne peut donc être implémenté qu'au niveau de **Hero** et **Monster**.

Bien heureusement, dans l'approche objet, ce problème peut être facilement résolu grâce à la notion de **classe abstraite** (cf. votre cours). Pour rappel, une classe abstraite est une classe qui déclare (et utilise généralement dans son code) une (ou des) méthode(s) qui n'est (sont) pas implémentée(s) à son niveau, mais dont l'implémentation est déléguée aux sous-classes.

Dans notre programme, l'idée sera donc d'étendre la méthode **getHitWith** de **Character** en utilisant une méthode **computeProtection** dont le but est de renvoyer une valeur permettant de calculer l'amoindrissement des dégâts subis. Étant utilisée dans **getHitWith**, la méthode **computeProtection** doit être déclarée au niveau de **Character**. Cependant, **computeProtection** ne peut être implémentée à ce niveau : au niveau de **Character**, on ne sait pas si le personnage est un **Hero** ou un **Monster**, et donc on ne sait pas s'il faut utiliser des pièces d'armure ou une épaisseur de peau pour effectuer le calcul...

La méthode **computeProtection** doit donc être déclarée en tant que méthode **abstraite**, indiquant ainsi que son implémentation sera réalisée dans les sous-classes de **Character** : elles seules auront assez d'information pour rendre effectif le calcul de protection, et donc le mécanisme d'absorption (hérité de **getHitWith**).

4.1. **Character** : déclarez la méthode abstraite **computeProtection** qui renvoie un **float** représentant la protection du personnage.

4.2. QUESTION : du fait de l'introduction de la méthode **computeProtection** au niveau de **Character**, les classes **Hero** et **Monster** présentent une erreur. Laquelle ? Pourquoi ?

4.3. Corrigez **Monster** en faisant en sorte que la valeur de protection de ses instances corresponde à la valeur d'épaisseur de leur peau.

4.4. Corrigez **Hero** en faisant en sorte que la protection de ses instances corresponde à leur armure totale (la somme des valeurs d'armure des pièces portées).

4.5. Dans **Character**, modifiez la méthode **toString()** pour qu'elle affiche la protection.

NB : Utilisez **Locale.US** pour avoir un '.' au lieu d'une ',' dans l'affichage du **float**.

Exemple : un héro portant une « **Ringed Knight Armor** » :

[Hero]	Ynovator	LIFE:	100	STAMINA:	50	PROTECTION:	14.99	(ALIVE)
----------	----------	-------	-----	----------	----	-------------	-------	---------

4.6. **Character** : modifiez **getHitWith** en y utilisant **computeProtection** :

- lorsque la protection dépasse 100, le personnage ne subit aucun dégâts (et donc il ne perd pas de vie)
- lorsque la protection est inférieure à 100, les dégâts sont amoindris du pourcentage correspondant (arrondir grâce à **Math.round**).

Exemple. Un coup avec une valeur de 10pts de dégâts sur un personnage avec une protection de 50pts ne retirera finalement que 5PV, c'est à dire 50% des dégâts reçus.

4.7. **LearningSoulsGame** : lancez le jeu (sans modifier **play_v1**) et constatez l'effet d'absorption des dégâts de la peau du monstre (que nous avons par défaut fixée à **20pts** dans la classe

Monster). Exemple :

[Hero]	Ynovator	LIFE:	100	STAMINA:	50	PROTECTION:	0.00	(ALIVE)
[Monster]	Monster_1	LIFE:	10	STAMINA:	10	PROTECTION:	20.00	(ALIVE)
Hit enter key for next move >								
Ynovator attacks Monster_1 with Basic Sword (ATTACK:8 DMG : 6)								
[Hero]	Ynovator	LIFE:	100	STAMINA:	30	PROTECTION:	0.00	(ALIVE)
[Monster]	Monster_1	LIFE:	4	STAMINA:	10	PROTECTION:	20.00	(ALIVE)

4.8. LearningSoulsGame :

- ✎ Ecrivez une méthode **play_v2** qui équipe un héros avec une armure, qui lance le **fight1v1**, puis lancez là à la place de **play_v1** pour constater l'effet de l'armure du héros sur l'absorption des dégâts.

5. Lycanthrope

- 5.1. Créez la classe **lsg.characters.Lycanthrope**, une sous-classe de **Monster** dont les instances possèdent par défaut :

- ✎ Le nom « **Lycanthrope** »
- ✎ Une arme de type **Claw**
- ✎ Une épaisseur de peau de **30pts**

- 5.2. Créez une méthode **play_v3** similaire à **play_v2**, mais qui a pour monstre une instance de **Lycanthrope**.

Exemple :

[Hero]	Ynovator	LIFE:	100	STAMINA:	50	PROTECTION:	14.99	(ALIVE)
[Lycanthrope]	Lycanthrope	LIFE:	10	STAMINA:	10	PROTECTION:	20.00	(ALIVE)
Hit enter key for next move >								
Ynovator attacks Lycanthrope with Basic Sword (ATTACK:5 DMG : 3)								
[Hero]	Ynovator	LIFE:	100	STAMINA:	30	PROTECTION:	14.99	(ALIVE)
[Lycanthrope]	Lycanthrope	LIFE:	7	STAMINA:	10	PROTECTION:	30.00	(ALIVE)
Hit enter key for next move >								
Lycanthrope attacks Ynovator with Bloody Claw (ATTACK:69 DMG : 59)								
[Hero]	Ynovator	LIFE:	41	STAMINA:	30	PROTECTION:	14.99	(ALIVE)
[Lycanthrope]	Lycanthrope	LIFE:	7	STAMINA:	5	PROTECTION:	30.00	(ALIVE)




6. Buffs

Comme dans de nombreux RPG, nos personnages vont pouvoir porter des items (bagues, talismans, ...) qui peuvent parfois leur donner des **buffs**. Dans notre jeu, un buff correspond à un pourcentage qui amplifie une attaque (calculée dans **attackWith**).

Exemple : Un personnage lance une attaque. Cette attaque a été calculée à 10pts de dégâts (selon l'ancien algorithme). Toutefois, si le personnage, au moment de cette attaque, porte des bagues qui lui octroient un buff global de 50, son attaque sera finalement augmentée de 50%. Ainsi, la méthode **attackWith** de **Character** ne retournera finalement pas 10pts (le calcul sans buff), mais 15pts de dégâts (grâce aux buffs). Bien entendu, si le personnage n'avait pas de buff (buff global à 0), la valeur retournée par **attackWith** est la même que dans l'ancienne version, c'est à dire 10pts dans notre exemple.

Les instances de **Héro** peuvent avoir des buffs en portant des bagues, c'est à dire des sous-classes de **Isq.buffs.Ring** dans les classes qui vous ont été fournies. Chaque héro peut porter au plus 2 bagues, et le buff global qu'il reçoit alors est la somme des buffs des bagues portées au moment de l'attaque. La manière d'équiper/retirer des bagues est similaire à celle que nous avons mise en œuvre pour les pièces d'armure.

6.1. Etudiez et ajoutez des commentaires (**Javadoc**) dans les classes qui vous ont été fournies pour documenter leur fonctionnement (et expliquer leur effet), en particulier pour détailler les spécificités des classes instanciables comme **RingOfDeath**, **DragonSlayerRing**, **MoonStone**, etc.

-  Port d'au maximum 2 anneaux (Ring) pour les héros.
-  Port d'au maximum 1 talisman pour les monstres.
-  Intégration et prise en compte du calcul de(s) buff(s) résultant du ou des items portés par un personnage lors de l'appel à **attackWith**.

INDICE : **Character** est une classe abstraite...

Notez l'effet « Ring of Death » (l'une des bagues portées par le héros).

```

[ Hero ]      Ynovator      LIFE :    100      STAMINA :    50      PROTECTION :    10.20      BUFF:    14.00      (ALIVE)
[ Lycanthrope ]  Lycanthrope  LIFE :     10      STAMINA :     10      PROTECTION :    30.00      BUFF:     0.00      (ALIVE)

Hit enter key for next move >

Ynovator attacks Lycanthrope with Basic Sword (ATTACK:10 | DMG: 7)
[ Hero ]      Ynovator      LIFE :    100      STAMINA :    50      PROTECTION :    10.20      BUFF:    14.00      (ALIVE)
[ Lycanthrope ]  Lycanthrope  LIFE :     3       STAMINA :     10      PROTECTION :    30.00      BUFF:     0.00      (ALIVE)

Hit enter key for next move >

Ynovator attacks Lycanthrope with Bloody Claw (ATTACK:107 | DMG: 96)
[ Hero ]      Ynovator      LIFE :     4       STAMINA :    30      PROTECTION :    10.20      BUFF:   10014.00      (ALIVE)
[ Lycanthrope ]  Lycanthrope  LIFE :     3       STAMINA :     5       PROTECTION :    30.00      BUFF:     0.00      (ALIVE)

Hit enter key for next move >

Ynovator attacks Lycanthrope with Basic Sword (ATTACK:910 | DMG: 3)
[ Hero ]      Ynovator      LIFE :     4       STAMINA :    10      PROTECTION :    10.20      BUFF:   10014.00      (ALIVE)
[ Lycanthrope ]  Lycanthrope  LIFE :     0       STAMINA :     5       PROTECTION :    30.00      BUFF:     0.00      (DEAD)

--- Ynovator WINS !!! ---

```

