

Langage Orienté Objet / Java

Partie 5 – Interface et généricité

The Learning Souls Game

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

Nous avons créé de nombreux items de différents types qui peuvent être utiles à nos personnages. Il nous faut aussi maintenant fournir des sacs pour qu'ils puissent les transporter afin de les avoir « sous la main » pendant leurs périples.

Un sac servira à ranger tout ce qu'un personnage voudra emmener avec lui, ou pourra ramasser au cours de son aventure. Comme dans la plupart des RPG, il permettra aux personnages de collecter tout ce qui peut l'être, dans la limite de sa capacité (poids max). Il servira de fourre-tout par lequel transitera tout ce qui est collectable (**collectible**) en jeu.

Ces éléments collectables pourront être des consommables (**consumables**), mais il pourra aussi s'agir d'items assez différents comme des armes (**weapons**), des pièces d'armure (**armor**), ou encore des items de **buffs**. D'un point de vue informatique, il est légitime de se demander comment créer cette notion de sac, capable de recevoir et gérer des objets aussi disparates. Nous allons ici voir que l'approche Objet fournit un moyen très efficace pour envisager des objets de types très différents selon des points de vue qui peuvent malgré tout être communs : la notion d'**interface**.

NB : nous allons dans cette partie plutôt nous focaliser sur des tests avec le héros.

Cependant, les sacs concernent aussi les monstres. En effet, il faut prévoir que nos héros auront un jour la possibilité de piller le sac des monstres qu'ils ont tué pour « *looter* » des items collectables à utiliser, équiper, ou vendre. De fait, les mécanismes évoqués ci-dessus seront principalement implémentés au niveau de la classe **Character**.

1. Bag

Un sac est destiné à permettre la gestion d'items pouvant être collectés. Un sac aura une capacité indiquant le poids maximal qu'il peut contenir pour stocker des instances de **Collectible**. Chaque (sous-)type de **Collectible** pourra être plus ou moins lourd. Ainsi, une pièce d'armure pèsera 4 kg, alors qu'une boisson n'en occupera qu'un.

1.1. Définissez l'interface **lsg.bags.Collectible**.

- ✎ Cette interface indique que ses instances devront implémenter la méthode :
`public int getWeight() ;`

Cette méthode retourne un entier correspondant au nombre de kilos du collectable.

1.2. Faites en sorte que les classes suivantes deviennent des **Collectible** :

- ✎ **ArmorItem** : 4 kg
- ✎ **Weapon** : 2 kg
- ✎ **Consumable** : 1 kg
- ✎ **BuffItem** : 1 kg
- ✎ **DragonSlayerLeggings** (cas particulier d'**ArmorItem**) : 3 kg

1.3. Définissez la classe **lsg.bags.Bag** avec les membres suivants :

- ✎ **capacity** : un attribut privé entier correspondant à la capacité du sac (nombre de kilos pouvant être transportés)
- ✎ **weight** : un attribut privé entier correspondant au nombre de kilos « utilisés »
- ✎ **items** : un attribut privé **HashSet** de **Collectible**
- ✎ Un constructeur à 1 paramètre entier permettant de fixer la capacité du sac lors de son instanciation (NB : cette capacité ne sera ensuite jamais modifiée)
- ✎ Les getters publics pour **capacity** et **weight**
- ✎ **public void push(Collectible item)** : permet d'ajouter un item dans le sac. Ne fait rien si l'item est trop gros par rapport à la place restante.
- ✎ **public Collectible pop(Collectible item)** : retire un item du sac. La méthode renvoie l'item retiré s'il était bien présent dans le sac, **null** sinon.
- ✎ **public boolean contains(Collectible item)** : méthode qui indique si l'item passé en paramètre se trouve bien dans le sac.
- ✎ **public Collectible[] getItems()** : retourne un tableau contenant les items du sac. (NB : on ne fournit pas directement le **HashSet** par principe d'encapsulation)
- ✎ **public String toString()** : surcharge de la méthode éponyme pour affiche le contenu d'un sac. Exemple avec sac à 10 slots contenant 3 **Collectible** :

Nom de la classe

```
Bag [ 3 items | 9/10 kg ]  
• Shotgun (min:6 max:20 stam:5 dur:100) [2 kg]  
• Dragon Slayer Leggings(10.2) [3 kg]  
• Ringed Knight Armor(14.99) [4 kg]
```

NB: on affichera « • **empty** » si le sac est vide.

1.4. Créez **lsg.bags.SmallBag** et **lsg.bags.MediumBag**, sous classe de **Bag** :

- ✎ Une instance de **SmallBag** a toujours une capacité de 10 kg.
- ✎ Une instance de **MediumBag** a toujours une capacité de 40 kg.
- ✎ Testez les méthodes **push**, **pop**, etc. dans un **main** avec des **print**. Exemple :

```
SmallBag [ 4 items | 10/10 kg ]
• Black Witch Veil(4.6)[4 kg]
• Uncle Greg's spicy Maroilles burger [40 life point(s)][1 kg]
• Basic Sword (min:5 max:10 stam:20 dur:100)[2 kg]
• Dragon Slayer Leggings(10.2)[3 kg]

Pop sur Dragon Slayer Leggings(10.2)

SmallBag [ 3 items | 7/10 kg ]
• Black Witch Veil(4.6)[4 kg]
• Uncle Greg's spicy Maroilles burger [40 life point(s)][1 kg]
• Basic Sword (min:5 max:10 stam:20 dur:100)[2 kg]
```

1.5. Dans la classe Bag, créez une méthode statique :

public static void transfer(Bag from, Bag into)

Cette méthode transfère le contenu du sac source dans le sac de destination dans la limite de la capacité de ce dernier. Les items qui n'ont pas pu être transféré restent dans le sac source. (NB : on n'essaiera pas d'optimiser.)

Exemple de test :

```
Sac 1 :
Bag [ 3 items | 9/10 kg ]
• Shotgun (min:6 max:20 stam:5 dur:100)[2 kg]
• Dragon Slayer Leggings(10.2)[3 kg]
• Ringed Knight Armor(14.99)[4 kg]

Sac 2 :
Bag [ 0 items | 0/5 kg ]
• (empty)

Sac 2 après transfert :
Bag [ 2 items | 5/5 kg ]
• Shotgun (min:6 max:20 stam:5 dur:100)[2 kg]
• Dragon Slayer Leggings(10.2)[3 kg]

Sac 1 après transfert :
Bag [ 1 items | 4/10 kg ]
• Ringed Knight Armor(14.99)[4 kg]
```

2. Character

Nous allons maintenant équiper nos personnages avec un sac, et leur permettre de s'en servir en ajoutant une série de membres à **Character**.

Vous testerez les méthodes créées au fur et à mesure en faisant des affichages console dans une méthode d'instance de la classe **LearningSoulsGame** nommée **testBag()**.

Dans **Character**, créez les membres suivants :

2.1. **bag** : attribut privé de type **Bag** qui sera instancié en **SmallBag** lors de l'instanciation d'un **Character**.

2.2. **void pickUp(Collectible item)** : méthode publique qui ajoute un item dans le sac du personnage (ne fait rien si le sac est plein).

Pour faciliter les tests, la méthode affiche un message du type :

```
Ynovator picks up Dragon Slayer Leggings(10.2)
```

2.3. **Collectible pullOut(Collectible item)** : méthode publique qui retire un item du sac du personnage, si cet item s'y trouve bien. La méthode retourne l'item retiré, ou **null** s'il n'était pas dans le sac.

Pour faciliter les tests, la méthode affiche un message du type :

```
Ynovator pulls out Uncle Greg's spicy Maroilles burger [0 life point(s)]
```

2.4. **printBag** : méthode publique qui affiche dans la console le contenu du sac du personnage.

Exemple :

```
BAG : SmallBag [ 3 items | 9/10 kg ]
• Shotgun (min:6 max:20 stam:5 dur:100) [2 kg]
• Dragon Slayer Leggings(10.2) [3 kg]
• Ringed Knight Armor(14.99) [4 kg]
```

2.5. **getBagCapacity** : méthode publique qui retourne la taille du sac du personnage.

2.6. **getBagWeight** : méthode publique qui retourne le nombre de slots encore disponibles dans le sac du personnage.

2.7. **getBagItems** : méthode publique qui retourne un tableau contenant les items contenus dans le sac du personnage.

2.8. **Bag setBag(Bag bag)** : méthode publique qui remplace le sac du personnage par le nouveau sac passé en paramètre. Les items de l'ancien sac sont transférés dans le nouveau dans la limite de sa capacité. L'ancien sac est retourné : il contient les items qui n'ont éventuellement pas pu être transférés si le nouveau sac était trop petit. Elle affiche un message du type :

```
Ynovator changes SmallBag for MediumBag
```

2.9. **equip(Weapon weapon)** : Recherche l'arme passée en paramètre dans le sac, et l'équipe (donc la retire du sac). Ne fait rien si l'arme n'est pas dans le sac.

equip(Consumable consumable) : même fonctionnement.

Ces méthodes affichent un message du type :

```
Ynovator pulls out Shotgun (min :6 max :20 stam :5 dur :100) and equips it !
```

3. Hero

- 3.1. Dans la classe **Hero**, créez les méthodes **equip(ArmorItem item, int slot)** et **equip(Ring ring, int slot)** sur le modèle mis en œuvre pour **equip(Weapon weapon)** dans **Character**.

Ces méthodes affichent un message du type :

Ynovator pulls out Dragon Slayer Leggings (10.2) and equips it !

- 3.2. Essayez/testez ces méthodes dans **testBag()** (de **LearningSoulsGame**)

4. Character

Habituellement, les personnages utilisent un consommables en l'équipant (**setConsumable**), puis en le consommant (**consume**). Toutefois, au milieu du combat, il se peut que le consommable équipé (ex. un kit de réparation) ne corresponde pas au besoin (ex. de la stamina, donc une boisson), et que le personnage n'ait pas le temps d'effectuer la suite d'actions nécessaires pour remonter en urgence la stat voulue.

Pour pallier à ce « problème », nous allons fournir aux personnages des méthodes qui lui permettent de récupérer dans le sac et de consommer à la hâte (sans l'équiper) le 1^{er} consommable d'un type donné. Le but est d'ajouter les méthodes **fastDrink()**, **fastEat()** et **fastRepair()** pour lesquelles le personnage plonge la main dans le sac, et consomme instantanément et sans le choisir le 1^{er} consommable du bon type trouvé.

- 4.1. Créez tout d'abord la méthode privée **fastUseFirst** à un paramètre nommé **type** qui correspond à une **classe** qui doit hériter de **Consumable**. (cf. cours sur les méthodes génériques).

La méthode « fouille » le sac à la recherche du premier consommable instance de la classe **type**, et le consomme.


Si, une fois utilisé, le consommable est vide (capacité à 0), il est jeté (retiré du sac). Dans le cas contraire, on le laisse dans le sac.

Dans tous les cas, la méthode retourne le consommable qui a été utilisé, ou **null** si le sac ne contenait pas de consommable du bon type.

- 4.2. Créez les méthodes d'urgence :

 **public Drink fastDrink()** : consomme la 1^{ere} boisson trouvée dans le sac.

 **public Food fastEat()** : consomme la 1^{ere} nourriture trouvée dans le sac.

 **public RepairKit fastRepair()** : consomme 1 charge du 1^{er} kit trouvé dans le sac.

Ces méthodes affichent un message du type :

Ynovator eats FAST :

Ynovator eats Uncle Greg's spicy Maroilles burger [40 life point(s)]

Ynovator pulls out Uncle Greg's spicy Maroilles burger [0 life point(s)]

4.3. Essayez/testez ces méthodes dans **testBag()** (de **LearningSoulsGame**).

 Vous pouvez ajouter la méthode **printWeapon()** dans **Character** pour faciliter les affichages.

Exemple global :

```
#####
#  THE LEARNING SOULS GAME  #
#####

Create exhausted hero :
[ Hero ]          Ynovator          LIFE :    1          STAMINA :    0          PROTECTION :    0.00          BUFF:    0.00          (ALIVE)

Ynovator picks up Dragon Slayer Leggings(10.2)
Ynovator picks up Ringed Knight Armor(14.99)
Ynovator picks up Shotgun (min:6 max:20 stam:5 dur:100)

BAG : SmallBag [ 3 items | 9/10 kg ]
* Shotgun (min:6 max:20 stam:5 dur:100) [2 kg]
* Dragon Slayer Leggings(10.2) [3 kg]
* Ringed Knight Armor(14.99) [4 kg]

Ynovator changes SmallBag for MediumBag

BAG : MediumBag [ 3 items | 9/40 kg ]
* Shotgun (min:6 max:20 stam:5 dur:100) [2 kg]
* Dragon Slayer Leggings(10.2) [3 kg]
* Ringed Knight Armor(14.99) [4 kg]

Ynovator picks up Hot Grandmother Coffe [10 stamina point(s)]
Ynovator picks up Uncle Greg's spicy Maroilles burger [40 life point(s)]
Ynovator picks up 12 years old Oban [150 stamina poin(s)]
Ynovator picks up Repair Kit [10 durability poin(s)]
Ynovator picks up Repair Kit [10 durability poin(s)]

BAG : MediumBag [ 8 items | 14/40 kg ]
* 12 years old Oban [150 stamina poin(s)] [1 kg]
* Shotgun (min:6 max:20 stam:5 dur:100) [2 kg]
* Repair Kit [10 durability poin(s)] [1 kg]
* Uncle Greg's spicy Maroilles burger [40 life point(s)] [1 kg]
* Dragon Slayer Leggings(10.2) [3 kg]
* Hot Grandmother Coffe [10 stamina point(s)] [1 kg]
* Ringed Knight Armor(14.99) [4 kg]
* Repair Kit [10 durability poin(s)] [1 kg]

--- AVANT ---
[ Hero ]          Ynovator          LIFE :    1          STAMINA :    0          PROTECTION :    0.00          BUFF:    0.00          (ALIVE)
ARMOR            1 : empty          2:empty          3:empty          TOTAL:0.0
WEAPON : Grosse Arme (min:0 max:0 stam:1000 dur:99)
BAG : MediumBag [ 8 items | 14/40 kg ]
* 12 years old Oban [150 stamina poin(s)] [1 kg]
* Shotgun (min:6 max:20 stam:5 dur:100) [2 kg]
* Repair Kit [10 durability poin(s)] [1 kg]
* Uncle Greg's spicy Maroilles burger [40 life point(s)] [1 kg]
* Dragon Slayer Leggings(10.2) [3 kg]
* Hot Grandmother Coffe [10 stamina point(s)] [1 kg]
* Ringed Knight Armor(14.99) [4 kg]
* Repair Kit [10 durability poin(s)] [1 kg]

--- ACTIONS ---
Ynovator drinks FAST :
Ynovator drinks 12 years old Oban [150 stamina point(s)]
Ynovator pulls 12 years old Oban [0 stamina point(s)]

Ynovator eats FAST :
Ynovator eats Uncle Greg's spicy Maroilles burger [40 life point(s)]
Ynovator pulls out Uncle Greg's spicy Maroilles burger [0 life point(s)]

Ynovator pulls out Shotgun (min:6 max:20 stam: 5 dur:100) and equips it !

Ynovator pulls out Dragon Slayer Leggings(10.2) and equips it !

Ynovator repairs FAST :
Ynovator repairs Shotgun (min:6 max:20 stam: 5 dur:100) with Repair Kit [10 durability point(s)]

--- APRES ---
```

[Hero]	Ynovator	LIFE :	41	STAMINA :	50	PROTECTION :	10.20	BUFF:	0.00	(ALIVE)
ARMOR	1 : Dragon Slayer Leggings(10.2)			2:empty		3:empty		TOTAL:10.2		
WEAPON : Shotgun (min:6 max:20 stam:5 dur:101)										
BAG : MediumBag [4 items 7/40 kg]										
* Repair Kit [9 durability poin(s)] [1 kg]										
* Hot Grandmother Coffe [10 stamina point(s)] [1 kg]										
* Ringed Knight Armor(14.99) [4 kg]										
* Repair Kit [10 durability poin(s)] [1 kg]										