

Programmation orientée objet - Java - TP noté

Pour cet examen, votre fond d'écran doit être vert clair. Vous devez absolument mettre tous les fichiers que vous souhaitez rendre dans le répertoire **EXAM**, qui est déjà présent sur votre compte à l'ouverture de la session. Tout ce qui ne se trouve pas dans ce dossier sera perdu lorsque vous vous déconnecterez.

Sous Eclipse, votre workspace doit être dans **EXAM**. Sinon configurez le workspace d'Eclipse (File > Switch WorkSpace) pour qu'il corresponde à un répertoire dans **EXAM**). Attention, les noms des classes, des champs et des méthodes que l'on vous demande doivent impérativement être respectés. De plus, le code doit être indenté correctement.

Commencez par vérifier la configuration d'Eclipse.

- Vérifiez que le JRE est `/usr/local/apps/java23`.
- Vérifiez que le compilateur est java 23 avec `enable preview`.
- Créez ensuite votre Java project qui porte votre nom : **NOM_PRENOM**.

La javadoc 23 et les slides du cours (seuls documents autorisés) sont accessibles ici :
<http://igm.univ-mlv.fr/~juge/javadoc-23/>
<https://igm.univ-mlv.fr/~beal/JavaL3.html>

Toutes les classes seront écrites dans un package **fr.uge.shop** pour la partie 1 et **fr.uge.shop2** pour la partie 2. On écrira une méthode **main** dans une classe **Main** dans chaque package pour faire les tests. Les tests des exemples donnés dans chaque question doivent être faits (les mettre en commentaire s'ils ne marchent pas). Ces tests sont pris en compte dans la notation.

PARTIE 1

Exercice 1.

On cherche à modéliser une gestion de pièces de monnaie anciennes. Les pièces seront de type **String** et peuvent être en or ("**gold**") ou en argent ("**silver**").

1. (a) Écrire un record **Wallet** qui permet de représenter un porte-monnaie contenant des pièces de ce type (type **String**). L'implémentation se fera avec un record qui comporte deux champs entiers (**gold**, **silver**) pour indiquer le nombre de pièces en "**gold**" et en "**silver**". On ne pourra créer que des porte-monnaie qui ont un nombre positif ou nul de pièces pour chaque type de pièces.
(b) Ajouter une méthode **add()** qui renvoie un nouveau porte-monnaie en ajoutant un nombre strictement positif (non nul) de pièces d'un même métal.
(c) Ajouter une deuxième constructeur sans argument qui crée un porte monnaie qui contient 0 pièces de chaque type.
(d) Compléter le record **Wallet** de telle sorte que le code suivant fonctionne et donne l'affichage demandé.

```
IO.println("exo 1");
var wallet1 = new Wallet();
var wallet2 = wallet1.add(2, "gold");
var wallet3 = wallet2.add(3, "silver");
IO.println(wallet1); // 0g 0s
IO.println(wallet2); // 2g 0s
IO.println(wallet3); // 2g 3s
```

2. On souhaite ajouter une méthode **get(String metal)** qui permet de savoir combien de pièces de type **metal** sont présentes dans un porte-monnaie. Ajouter la méthode **get(String metal)** de telle sorte que le code suivant fonctionne.

```
IO.println("exo 2");
var wallet4 = new Wallet().add(3, "gold");
IO.println(wallet4.get("gold")); // 3
IO.println(wallet4.get("silver")); // 0
```

3. On souhaite maintenant gérer la notion de caddie (**Caddy**) qui va contenir des armes (**Weapon**). Les armes peuvent être des épées (**Sword**) ou des boucliers (**Shield**) (un bouclier est une arme défensive). Une épée est définie par un nom (**name**, une chaîne de caractères), ainsi qu'un prix représenté par un certain nombre de pièces d'or ou d'argent (**price**, de type **Wallet**). Un bouclier est défini par un booléen (**small**) qui indique si un bouclier est petit ou normal (true = petit bouclier, false = bouclier normal), et par son prix **price** de type **Wallet**.

Un **Caddy** est vide à la création. On peut y ajouter des armes (épées ou boucliers) en utilisant une méthode **add**. Une même arme peut figurer plusieurs fois dans le caddie. Le caddie contiendra donc une liste d'armes.

L'affichage d'un caddie affiche la liste de ses armes dans l'ordre d'insertion, chaque arme sur une ligne. L'affichage d'une épée affiche son nom et son prix avec le format visible comme ci-dessous. Pour un bouclier, s'il est petit, l'affichage commence par "small shield" tandis que s'il est normal, l'affichage commence par "shield". En plus du type de bouclier, l'affichage d'un bouclier donne aussi son prix avec le même format que pour le prix d'une épée.

- Écrire les records **Sword** et **Shield** implémentant une interface **Weapon**.
- Écrire la classe **Caddy** qui contient une liste de **Weapon**.
- Ajouter la méthode **add** à **Caddy**.
- On veut de plus qu'il ne soit possible d'ajouter que des épées et des boucliers, et pas d'autres armes, à un caddie. Faites les modifications nécessaires pour avoir cette condition.
- Faire en sorte que le code suivant fonctionne.

```
IO.println("exo 3");
var sword1 = new Sword("Skullcrusher", new Wallet().add(5, "gold"));
var caddy1 = new Caddy();
caddy1.add(sword1);
IO.println(caddy1);
// sword Skullcrusher 5g 0s
var sword2 = new Sword("Peacekeeper", new Wallet().add(4, "gold"));
var wallet5 = new Wallet().add(6, "gold");
wallet5 = wallet5.add(5, "silver");
var shield1 = new Shield(false, wallet5);
var caddy2 = new Caddy();
caddy2.add(sword2);
caddy2.add(shield1);
IO.println(caddy2);
// sword Peacekeeper 4g 0s
// shield 6g 5s
```

Note : pour l'affichage, il n'y a pas retour à la ligne à la fin de l'affichage. C'est println qui s'en charge.

4. On veut pouvoir faire la somme des prix des armes d'un caddie, mais avant de faire cela, nous allons ajouter une méthode **sum** dans **Wallet** pour faire la somme de deux **Wallet**. La méthode renverra un nouvel objet **Wallet**.
- Écrire la méthode **sum** dans **Wallet**.

- (b) Ajoutez une méthode `price` à un caddie qui fait la somme des prix des armes présentes dans le caddie. La méthode renverra le porte-monnaie qui contient 0 pièces de chaque type si le caddie est vide.

Faites en sorte que le code suivant fonctionne.

```
IO.println("exo 4");
var wallet6 = new Wallet().add(2, "gold");
var wallet7 = new Wallet().add(3, "gold");
var wallet8 = wallet6.sum(wallet7);
IO.println(wallet8); // 5g 0s
var shield4 = new Shield(true, new Wallet().add(3, "gold"));
var shield5 = new Shield(true, new Wallet().add(3, "gold"));
var sword5 = new Sword("Soulrinker", new Wallet().add(7, "gold"));
var caddy5 = new Caddy();
caddy5.add(shield4);
caddy5.add(shield5);
caddy5.add(sword5);
caddy5.add(sword5);
IO.println(caddy5.price()); // 20g 0s
```

5. (a) Ajoutez une méthode `copiesNumber` qui renvoie une map dont les clés (de type `Weapon`) sont les armes contenues dans le caddie, et la valeur associée à une clé est le nombre d'exemplaires de cette arme dans le caddie.
- (b) Ajouter une méthode `copiesNumber2` qui fait la même chose mais renvoie cette fois une map non modifiable.

```
IO.println("exo 5");
IO.println(caddy5.copiesNumber());
// {small shield 3g 0s=2, sword Soulrinker 7g 0s=2}
```

6. (Question facultative).

- (a) Ajoutez une méthode `public List<Weapon> swords()` qui renvoie la liste non modifiable des épées contenues dans le caddie (seulement les épées).
- (b) Si vous ne l'avez fait comme cela à la question précédente, écrire une méthode `public List<Weapon> swords2()` qui fait la même chose que `swords` en utilisant le pattern matching de types de Java.

Faites en sorte que le code suivant fonctionne.

```
IO.println("exo 6");
IO.println(caddy5.swords());
// [sword Soulrinker 7g 0s, sword Soulrinker 7g 0s]
```

PARTIE 2

Exercice 2.

Faites un copier-coller du package `fr.uge.shop` dans un package `fr.uge.shop2` et travaillez dans `fr.uge.shop2` pour cette partie.

On souhaite fournir une nouvelle implantation de `Caddy` qui permette, pour chaque arme présente dans le caddie, d'accéder en temps constant à son nombre d'exemplaires dans le caddie. Les armes seront donc stockées dans une map de type `HashMap<Weapon, Integer>` qui associe à chaque arme son nombre d'exemplaires dans le caddie.

Dans cette partie, on va

- garder les records `Shield`, `Sword`, `Wallet` et l'interface `Weapon`,
- changer la classe `Caddy` et la méthode `main`.

1. Changer la classe `Caddy` en enlevant la liste et en ajoutant la map comme champ.
2. Modifier les méthodes `add` et `copiesNumber`. Ajouter une méthode `get` qui renvoie le nombre d'exemplaires d'une arme présente dans le caddie. L'opération doit être en temps constant.
3. Modifier la méthode `price`.
4. Écrire une méthode `toString()` qui affiche le caddie comme dans le code indiqué plus bas (une arme par ligne suivie d'un ":" suivi du nombre d'exemplaires de l'arme).
5. (Question facultative). Changer la méthode `List<Weapon> swords` (ou `List<Weapon> swords2`) qui renvoie la liste des épées. La méthode devra tenir compte du nombre d'exemplaires de chaque épée. Par exemple, si une épée figure 3 fois dans le caddie, elle devra apparaître 3 fois dans la liste retournée.

```

////////////////////////////////////
IO.println("exo 7");
var shield4 = new Shield(true, new Wallet().add(3, "gold"));
var shield5 = new Shield(true, new Wallet().add(3, "gold"));
var sword5 = new Sword("Souldrinker", new Wallet().add(7, "gold"));
var caddy5 = new Caddy();
caddy5.add(shield4);
caddy5.add(shield5);
caddy5.add(sword5);
caddy5.add(sword5);
IO.println(caddy5.price()); // 20g 0s
////////////////////////////////////
IO.println("exo 8");
IO.println(caddy5.copiesNumber());
//{small shield 3g 0s=2, sword Souldrinker 7g 0s=2}
////////////////////////////////////
IO.println("exo 9");
IO.println(caddy5.get(new Shield(true, new Wallet().add(3, "gold"))));
// 2
IO.println(caddy5);
// small shield 3g 0s: 2
// sword Souldrinker 7g 0s: 2
////////////////////////////////////
IO.println("exo 10");
IO.println(caddy5.swords());
// [sword Souldrinker 7g 0s, sword Souldrinker 7g 0s]

```