Info – Bloc 1 – UE09 : Programmation Orientée Objet

Laboratoire 3

Durée prévue : 6 heures avec du travail à domicile

Objectifs visés

À la fin du laboratoire, les étudiants seront capables de :

- Surcharger des constructeurs et appeler un constructeur dans un autre constructeur
- Définir des relations de compositions où un objet possède, parmi ses champs, des références à d'autres objets

Plus généralement, ils seront capables d'initialiser et de composer des objets à partir d'autres objets en Java.

Créer le projet poo.labo03

Crée un projet `poo.labo03`. Dans le répertoire `src` du projet, crée le paquetage `poo.labo03` et ajoutez-y les fichiers `Wizard.java, HouseCup.java et Program.java` proposés en annexe. Crée un package `poo.labo03.views` et ajoutes-y le fichier `SwingHouseCupView.java`, également proposé en annexe.

Ajoute un second répertoire de source appelé `tests` pour tes tests unitaires et crée le paquetage `poo.labo03`. Ajoutes-y les fichiers `WizardTestData.java, WizardTest1.java, WizardTest2.java, WizardTest3.java, WizardTest4.java et HousesCupTest.java` proposés en annexe. Ajoute JUnit aux bibliothèques du projet.

Tu dois obtenir une structure semblable à celle présentée par la Figure 1.

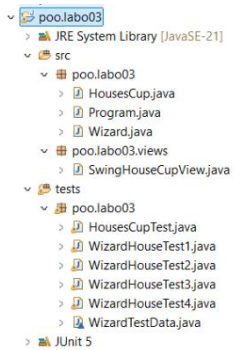

Figure 1 Structure de départ du projet

Exercices

Le laboratoire contient des séries d'exercices de difficultés croissantes. Après chaque série, le responsable propose une solution et répond à tes questions. Certains exercices dépendent des exercices précédents. Il vaut mieux les faire dans l'ordre.

À l'HELSo, les sorciers sont regroupés en quatre maisons (*WizardHouse*) qui, chaque année, se disputent la coupe des quatre maisons. Nous voulons un programme qui classe les maisons selon les points gagnés par leurs sorciers.

Attention

Sauf mention du contraire, les champs d'objet seront privés tandis que les méthodes seront publiques.

Exercice 1

Objectifs :

- Définir une relation de composition avec un seul composant
- Redéfinir des méthodes héritées de la classe `Object`

Durée estimée : 20 min

Une maison possède un sorcier fondateur dont découle le nom de la maison. Le tableau ci-dessous liste les quatre maisons de l'HELSo.

Nom complet du fondateur	Nom de la maison
Godric Gryffindor	Gryffindor
Rowena Ravenclaw	Ravenclaw
Helga Hufflepuff	Hufflepuff
Salazar Slytherin	Slytherin

Déclare la classe `poo.labo03.WizardHouse`. Cette classe déclare un champ d'objet de type `Wizard` correspondant à son fondateur. Ce champ sera initialisé par un constructeur prenant en paramètre une référence à un `Wizard`. Si le paramètre vaut `null`, le constructeur initialise le fondateur avec la constante `Wizard.YOU_KNOW_WHO`, déjà définie.

Astuce

Appelle la méthode de classe Objects.requireNonNullElse(T, T) pour valider le paramètre.

`WizardHouse` déclare un accesseur `String getName()` qui retourne le nom de cette maison. Si le fondateur est « tu-sais-qui », le nom de la maison sera « Death Eaters ». Sinon, elle retourne le nom de famille du fondateur.

Astuce

Fais collaborer tes objets ! Ajoute l'accesseur `getLastname()` à la classe `Wizard` et appelle-le depuis `WizardHouse.getName()`.

`WizardHouse` redéfinit également `toString()` pour retourner le nom de la maison et celui de son fondateur. Ta classe devra réussir les tests de la classe `WizardHouseTest1`.

Exercice 2

Objectifs :

- Définir une relation de composition où plusieurs composants jouent le même rôle
- Redéfinir les méthodes héritées de `Object`

Durée estimée : 40 min

Au cours des années, les maisons voient arriver de nouveaux sorciers. Le fondateur est un cas particulier : il est membre permanent de sa maison.

Dans `WizardHouse`, déclare la méthode d'objet `void add(Wizard w)` qui ajoute le sorcier `w` à la maison recevant l'appel. Si `w` est une référence nulle ou une référence à un sorcier déjà membre de la maison, la méthode ne fait rien. Sinon, elle affecte `w` à la première position disponible d'un tableau des membres. Si toutes les positions du tableau sont occupées, `add(Wizard)` crée un tableau de capacité supérieure, y copie les membres existants et ajoute `w` à la fin.

Définis l'accesseur `int getWizardsCount()` qui retourne le nombre de membres actuels**.** Attention, ne compte pas les cases disponibles du tableau des membres.

Ta classe devra réussir les tests de la classe `WizardHouseTest2`.

Approche recommandée

Pour implémenter le comportement souhaité, une approche adaptée consiste à mémoriser le nombre de membres dans un champ d'objet, appelons-le `count`. La Figure 2 présente l'état attendu d'une maison après l'appel au constructeur. Après la création d'une maison, `count` vaut 1, car il y a le membre fondateur. Si le tableau de départ contient un seul élément, `count` est égal à la taille du tableau.

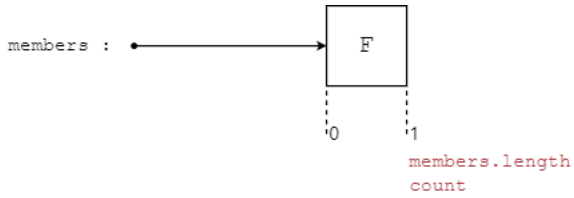

Figure 2 État après appel au constructeur

Le fait que `count` soit égal à la taille du tableau indique que ce tableau est rempli : aucune case n'est disponible pour un nouveau membre. Nous devons créer une copie du tableau avec plus de cases. Une stratégie souvent suivie consiste à doubler la taille de la copie.

La Figure 3 présente cette étape. Dans cet exemple, nous créons une copie de taille 2 et l'affectons à `members`. Note qu'aucune référence ne désigne l'ancien tableau qui devient candidat au ramasse-miettes[1].

[1] Si ce dernier est appelé.

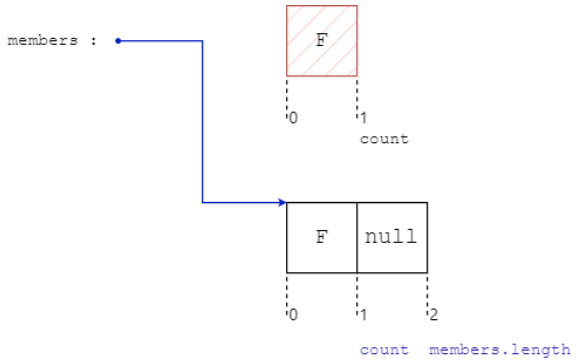

Figure 3 Création d'une copie et affectation à le champ members

Une fois la copie créée, il reste à affecter le nouveau membre à la case disponible **et** à actualiser `count` pour que sa valeur corresponde au nombre de membres (Figure 4). Dans cet exemple, `count` est de nouveau égal à la taille du tableau : le prochain ajout demandera de créer une copie de taille 4. Cette copie pourra accueillir deux nouveaux membres avant d'être remplies.

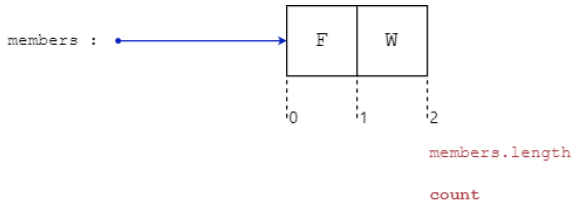

Figure 4 Affectation du nouveau membre et mise à jour de count

Retour avec le responsable

Durée estimée : 30 min

Le responsable présente une correction. Les points d'attention suivants seront abordés :

- Quand un objet A possède une référence à un objet B parmi ses champs, il existe une relation de composition entre A et B où A joue le rôle de composite et B celui de composant.
- Il y a également composition quand un objet référence **plusieurs composants**.
- Un composant est typiquement initialisé par le constructeur à l'aide d'un paramètre.
- Tu peux appeler les méthodes de l'objet référencé.
- Dans un tableau d'objets, chaque case est une référence susceptible d'avoir la valeur `null`.

Exercice 3

Objectif : définir une relation de composition où plusieurs composants jouent le même rôle

Durée estimée : 30 min

Des sorciers peuvent quitter une maison, sauf le fondateur.

Déclare une méthode d'objet `void remove(Wizard w)` dans la classe `WizardHouse` qui tente de supprimer le sorcier `w` des membres. Si w est une référence nulle ou une référence au fondateur, la méthode ne fait rien. Sinon, elle cherche le sorcier `w` dans le tableau. Si `w` appartient au tableau, `remove(Wizard)` marque la case correspondante comme libre et réorganise éventuellement le tableau.

Ta classe devra réussir les tests de la classe `WizardHouseTest3`.

Cet exercice est à terminer pour la séance de laboratoire suivante.

Approche recommandée

L'idée de la suppression est de garder le tableau des membres dense : à la fin de la suppression, tous les membres restants doivent se situer au début du tableau et il ne peut pas y avoir de cases libres entre deux membres.

La Figure 5 présente une situation où nous voulons supprimer le membre `M1`. La variable `memberIndex` correspond à la position qu'occupe ce membre.

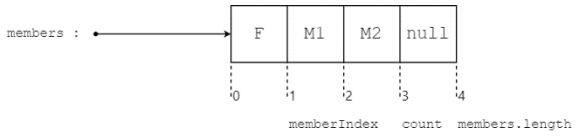

Figure 5 Situation de départ

La première étape consiste à remplacer ce membre par `null`. Avec ce remplacement, nous avons créé un « trou » entre le fondateur et `M2` que nous devons combler.

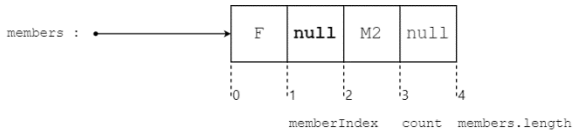

Figure 6 Affectation de null à la case contenant M1

Si le tableau était dense avant l'étape précédente, nous pouvons affirmer qu'il existe un membre que nous pouvons utiliser pour combler le trou. Ce membre est le dernier du tableau et sa position correspond au nombre de membres avant la suppression – 1. Nous pouvons dès lors permuter la case `memberIndex` avec celle correspondant au dernier membre. La dernière étape consistera à retirer 1 aux nombres de membres pour qu'il corresponde au nombre de membres après suppression.

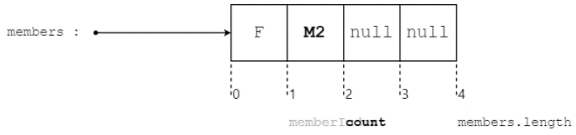

Figure 7 Permutation avec le dernier membre et mise à jour du compteur de membres

Grâce à la permutation, nous retrouvons un tableau dense. Notez que ce tableau compte deux cases libres : nous pourrons faire deux ajouts avant d'avoir à créer un nouveau tableau.

Retour avec le responsable

Durée estimée : 20 min

Le responsable présente une correction. Les points d'attention suivants seront abordés :

- Dans un tableau d'objets, chaque élément est une référence susceptible d'avoir la valeur `null`.
- Il est important de garder la condition `première position disponible == dernière position occupée + 1`. Pour ce faire, affecte le dernier sorcier à la position du sorcier supprimé et diminue le compteur de sorciers.
- Dans les figures 5 à 7, `memberIndex` est une variable locale à la méthode `remove(Wizard)`. Elle ne devrait pas prendre la forme d'un champ, car son utilité est limitée à l'appel de méthode.

Avant d'attaquer la série suivante

Nous t'invitons à écrire les tests unitaires suivants dans la classe `WizardHouseTest4`. Pour te faire gagner du temps, tu trouveras des sorciers prédéfinis dans la classe `WizardTestData`.

Exercice 4

Objectif : Construire un résultat à partir des composants

Durée estimée : 20 min

Une maison peut fournir les noms de ses membres. Le premier nom correspondra à celui du fondateur.

Déclare l'accesseur `String[] getWizardsNames()` dans la classe `WizardHouse`. Cette méthode retourne le tableau des noms des membres. Attention, le résultat dépend des membres de la maison au moment de l'appel et ne doit pas compter d'éléments `null`.

Valide ta méthode à l'aide de tests unitaires. Montre notamment que le résultat change selon les membres que tu ajoutes et retires. Tu peux t'inspirer de l'exemple ci-dessous.

```
Étant donnée la maison Gryffindor dont les membres sont GODRIC_GRYFFINDOR
(fondateur), HARRY_POTTER et HERMIONE_GRANGER.

Quand je demande à cette maison le nom de ses membres.

Alors je devrais obtenir le tableau [ « Godric Gryffindor », « Harry
Potter », « Hermione Granger » ].
```

Exercice 5

Objectifs :

- Définir des surcharges de constructeurs ;
- Appeler un constructeur dans un second constructeur à l'aide de `this(args)`.
- Encapsuler les champs pour éviter les modifications externes à l'objet

Durée estimée : 20 min

Nous souhaitons créer des maisons avec des membres de départ autres que le fondateur.

Déclare une surcharge de constructeur `WizardHouse(Wizard founder, Wizard[] members)`. Nous supposerons que le fondateur n'appartient pas au second paramètre. Cependant, le second paramètre peut contenir des références nulles. Le constructeur créera le tableau des membres en respectant le principe des copies défensives pour les tableaux.

Supprime les répétitions de code en appelant ton nouveau constructeur dans le constructeur à un seul paramètre.

Valide ton constructeur par au moins test unitaire. Vérifie notamment que l'ajout ou la suppression d'un membre de la maison est sans effets de bord sur le tableau fourni en argument. Tu peux t'inspirer de l'exemple ci-dessous.

```
Étant donnés le sorcier GODRIC_GRYFFINDOR, le tableau de sorciers
[HARRY_POTTER, HERMIONE_GRANGER] et la maison gryffindor créé avec les
variables précédentes.

Quand je remplace HERMIONE_GRANGER par DRAGO_MALEFOY dans le tableau de
départ Et que je demande à la maison gryffindor les noms de ces membres.

Alors je devrais obtenir le tableau [ « Godric Gryffindor », « Harry
Potter », « Hermione Granger » ].
```

Exercice 6

Durée estimée : 30 min

Objectifs :

- Déduire un champ d'un énoncé
- Définir une méthode de comparaison.

Nous voulons maintenir un classement des maisons sur base des points reçus par leurs membres. Par exemple, si Harry Potter gagne 10 points, ces 10 points iront à la maison Gryffondor. Si Hermione Granger gagne après 5 points, la maison Gryffondor aura 15 points, etc. **Pour créer ce classement, nous comparerons les maisons selon les points qu'elles auront récoltés. Par ailleurs, les maisons avec le même nombre de points seront triées selon leurs noms.**

Déclare une méthode d'objet `void grantPoints(String wizardName, int newPoints)` dans la classe `WizardHouse`. Si la maison qui reçoit l'appel a pour membre le sorcier `wizardName`, elle ajoute `newPoints` à ses points. Sinon, elle ne fait rien, en particulier si wizardName est `null`.

Déclare un accesseur `int getPoints()` qui retourne les points de cette maison.

Déclare une méthode d'objet `int compareTo(WizardHouse other)` qui retourne un entier :

- Négatif si le nombre de points de cette maison est plus petit que le nombre de points de l'autre maison ou si les 2 maisons ont le même nombre de points et que le nom de cette maison est plus petit que celui de l'autre maison.
- Positif si le nombre de points de cette maison est plus grand que le nombre de points de l'autre maison ou si les 2 maisons ont le même nombre de points et que le nom de cette maison est plus grand que celui de l'autre maison.
- Égal à 0 si cette maison et l'autre maison ont le même nombre de points et le même nom.

Valide ta méthode à l'aide de tests unitaires. Montre notamment que le résultat de la comparaison est inversé quand l'objet qui reçoit l'appel et celui donné en argument sont permutés. Inspire-toi de l'exemple ci-dessous.

Étant données les maisons gryffindor (10 points), slytherin(10 points) et hufflepuff(0 points).

Les assertions suivantes sont vérifiées :
1. gryffindor comparé à lui-même retourne 0
2. Poufsoufle comparé à gryffindor retourne un entier négatif (P < G sur base du score)
3. gryffindor comparé à Poufsoufle retourne un entier positif (G > P sur base du score)
4. slytherin comparé à null lance un NullPointerException
5. gryffindor comparé à slytherin retourne un entier négatif (score identique, mais « g » vient avant « s » dans l'ordre lexicographique)
6. slytherin comparé à gryffindor retourne un entier positif (score identique, mais « s » vient après « g » dans l'ordre lexicographique)

Astuce

Pour vérifier l'assertion 4, écris l'instruction suivante :

```
assertThrows(NullPointerException.class, () -> {
      slytherinHouse.compareTo(null);
});
```

Retour avec le responsable

Durée estimée : 20 min

Le responsable présente une correction. Les points d'attention suivants seront abordés :

- Quand une classe possède plusieurs constructeurs, on essaie de réduire le code redondant en appelant un constructeur dans un autre. Typiquement, le dernier constructeur appelé est celui qui a la plus longue liste de paramètres.
- La méthode `compareTo(WizardHouse)` a un comportement analogue à la méthode `compareTo(String)` définie dans la classe `String`.

Exercice 7

Durée estimée : 90 min

Objectifs :

- Déduire la relation de composition d'un texte

Nous avons défini comment comparer deux maisons, il nous reste à définir un classement des maisons exploitant notre méthode de comparaison.

Déclare une classe `WizardHouseRanking` dont les objets seront initialisés à l'aide d'un tableau de `WizardHouse` reçu en argument. Si le tableau est null, le classement sera considéré comme vide.

Déclare un accesseur `String[] getWizardsNames()` qui retourne un tableau des noms des membres de toutes les maisons. Ce tableau sera trié par ordre croissant.

Déclare deux autres accesseurs `WizardHouse getHouseByRank(int rank)` et `int getHousesCount()` qui retournent la maison occupant un rang reçu en paramètre (0 correspondant à la maison avec le plus de points, 1 correspondant à la maison en seconde position, etc.) et le nombre de maisons que compte ce classement. Ta méthode lèvera une `IndexOutOfBoundsException` en cas de rang invalide.

Astuce

Appelle la méthode Objects.checkIndex(int index, int length) pour valider le rang.

Déclare une méthode d'objet `void grantPoints(String wizardName, int newPoints)` qui appelle la méthode `grantPoints(String wizardName, int newPoints)` des maisons. **Une fois les points actualisés, la méthode devra trier le tableau des maisons.** Nous te proposons d'adapter l'algorithme du tri par insertion découvert dans les labos d'algo[2]. Attention, il faut trier par ordre décroissant des scores.

Même si cela est vivement recommandé, nous ne te demandons pas d'écrire de tests unitaires pour cette classe. Les tests fournis pour l'exercice 8 devraient suffire…

Exercice 8

Durée estimée : 30 min

Achève un programme pour classer les maisons.

[2] Ce lien, déjà donné en algo, peut t'aider à mieux comprendre l'algorithme : Algorithme de tri par insertion (free.fr)

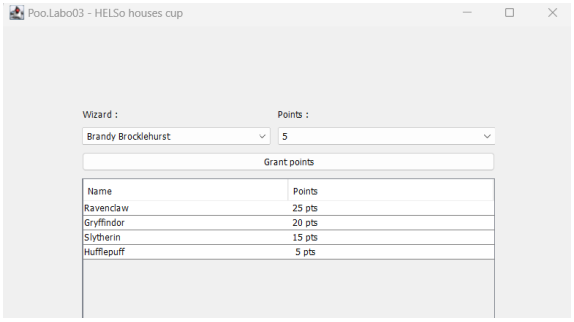

Figure 8 Exemple d'exécution les maisons apparaissent triées par ordre décroissant des points

Complète la classe `HousesCup` afin qu'elle crée un classement pour les maisons suivantes.

Gryffindor
Godric Gryffindor
Harry Potter
Hermion Granger

Hufflepuff
Helga Hufflepuff
Justin Finch-Fletchey
Susan Bones
Hannah Abbot

Ravenclaw
Rowena Ravenclaw
Terry Boot
Brandy Brocklehurst

Slytherin
Salazar Slytherin
Millicent Bubstrode
Drago Malefoy

Complète l'accesseur `getMembersNames()` afin qu'il demande au classement les noms des sorciers.

Complète l'accesseur `getRankingTable()` pour retourner un tableau 2D de strings comptant autant de lignes que de maisons. Chaque ligne possède deux colonnes : la première correspond au nom de la maison et la seconde met en forme le nombre de points. Le tableau sera construit à l'aide des accesseurs `getHouseCount()` et `getHouseByRank(int)` du classement.

Complète enfin la méthode `grantPoints(String, int)` pour appeler la méthode de même nom du classement.

Valide tes adaptations avec les tests unitaires de la classe `HousesCupTest`. Ceux-ci devraient suffire à valider ta classe `WizardHouseRanking`.

Une correction sera proposée au début du labo suivant.

Améliorations

Utilisation de l'ellipse

Remplace le second paramètre du constructeur `WizardHouse(Wizard, Wizard[])` par une ellipse. Propose un test qui illustre l'utilisation de l'ellipse. Existe-t-il un risque d'effet de bord lié aux tableaux avec cette seconde version ? Fais de même pour le constructeur de `WizardHouseRanking(Wizard, Wizard[])`.

Tri par ordre décroissant des points et par ordre croissant des noms

L'algorithme de comparaison des maisons fait que les maisons avec le même nombre de points sont triées par ordre décroissant de leurs noms. Adapte ta méthode pour que les maisons avec le même nombre de points soient triées par ordre croissant de leurs noms.

À la découverte des listes

Tu peux simplifier l'implémentation de la classe `WizardHouse` en remplaçant le tableau des membres par un objet de la classe `ArrayList<Wizard>`. Le site w3schools propose une page dédiée à l'utilisation de cette classe : Java ArrayList (w3schools.com).