Info – Bloc 1 – UE09 : Programmation Orientée Objet

Laboratoire 2

Durée prévue : 6 heures

Objectifs visés

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

- Définir des classes d'objets composées d'attributs, de méthodes d'objet et de constructeurs.
- Redéfinir des méthodes héritées de la classe `java.lang.Object` comme `toString()` et `equals(Object)`
- Définir des énumérations d'objets composées d'attributs, de méthodes d'objet et de constructeurs.

Plus généralement, ils seront capables de créer des types d'objets et de les manipuler en Java. Bien qu'abordée aux cours théoriques, l'encapsulation des attributs n'est pas requise pour ce laboratoire.

Corrections du labo 01

Durée estimée : 20 minutes

Ton responsable du laboratoire présente une correction pour l'exercice 7 du labo 01. Il revient ensuite sur certains concepts liés aux objets :

- Pour utiliser un objet, il faut l'avoir créé
- Valider les arguments, en particulier les arguments `null`, est important. Des tests unitaires doivent montrer que vous validez vos arguments.
- Distinguer la comparaison de références (`== et !=`) et la comparaison d'objets (`equals(Object)`).
- Garnir un tableau à l'aide de la boucle `for(elem:itérable)` est compliqué. La boucle `for` classique convient mieux. Plus généralement, il faut choisir la structure de contrôle adaptée.

Créer le projet poo.labo02

Durée estimée : 05 min

Note : cette activité peut se faire avec le responsable.

Crée un projet `poo.labo02` dans ton espace de travail.

Ajoute un second répertoire de sources appelé `tests` pour tes tests unitaires. Ajoute également JUnit aux bibliothèques. Ajoutes-y le fichier WizardTest.java, disponible en annexe.

Ajoute les fichiers `Level.java`, `Program.java` et `SpellPracticeSession.java` au répertoire src://poo/labo02/. Enfin, crée le package `poo.labo02.views` et ajoutes-y le fichier `SwingSpellPracticeView.java`. Ces fichiers sont disponibles en annexe.

Ton projet est configuré. Tu dois obtenir une structure similaire à celle de l'image ci-dessous.

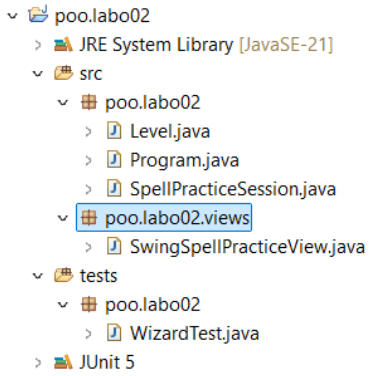

Figure 1 Structure de départ attendue

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.

Tu es un jeune sorcier étudiant à la Haute École Libre des Sorciers (HELSo). Pendant tes cours, tu étudies des sortilèges (*spell*) dont tu dois prononcer l'incantation avec un débit de mots précis, pour ne pas en perdre le contrôle.

Exercice 1

Objectifs :

- Déclarer une classe d'objet
- Déclarer des attributs d'objet
- Déclarer un constructeur initialisant les attributs d'un objet
- Déclarer des méthodes d'objet

Durée estimée : 45 minutes

Déclare une classe `poo.labo02.Spell`. Cette classe possède un constructeur `public Spell(String givenIncantation)` qui initialise un champ `incantation` à l'aide de `givenIncantation`. Si `givenIncantation` est un string `null` ou blanc, le constructeur le remplace par la valeur `"NO INCANTATION".`

`Spell` déclare deux méthodes d'objet :

- `public String getIncantation()` qui retourne l'incantation de la formule en lettre majuscule.
- `public String cast(int elapsedTime)` qui retourne une représentation de l'incantation où chaque mot est mis en lettres majuscules. Les mots seront séparés par `elapsedTime` symboles « . »[1]. Le résultat sera terminé par « ! ». Si `elapsedTime` est une valeur négative ou égale à zéro, la méthode la remplace par 1.

- Pour déterminer si un `String` est blanc, appelle la méthode d'objet isBlank().
- La classe `String` prévoit une méthode d'objet `repeat(int)` qui retourne un `String` composé de l'objet qui reçoit l'appel répété `count` fois.
- Les objets de la classe `java.util.StringJoiner` sont utiles pour construire des strings à l'aide de séparateurs, d'un préfixe et d'un suffixe…

Valide tes méthodes avec des tests unitaires. **Attention, ces derniers ne devraient pas accéder aux champs des objets.** Implémente au minimum les tests suivants.

[1] Chaque point représente une pause d'un dixième de seconde.

Méthode getIncantation()

Incantation du sortilège	Résultat attendu
`"wingardium leviosa"`	`"WINGARDIUM LEVIOSA"`
`null`	`"NO INCANTATION"`
`" \n \t"`	`"NO INCANTATION"`

Méthode cast(int)

Incantation du sortilège	argument de cast(int)	Résultat attendu
`"wingardium leviosa"`	3	`"WINGARDIUM...LEVIOSA !"`
`"wingardium leviosa"`	0	`"WINGARDIUM.LEVIOSA !"`
`"wingardium leviosa"`	-5	`"WINGARDIUM.LEVIOSA !"`
`null`	5	`"NO.....INCANTATION !"`
`"Stupefy"`	5	`"STUPEFY !"`
`"Stupefy"`	-1	`"STUPEFY !"`
`"Mucus ad nuseum"`	2	`"MUCUS..AD..NUSEUM !"`

Déclare un test par couple (sortilège, méthode). Déclare par exemple un test unitaire `castsWingardiumLeviosa`, un second test `castsStupefy`, etc.

Exercice 2

Objectifs :

- Déclarer une classe d'objet
- Déclarer une méthode d'objet faisant muter l'objet qui reçoit l'appel

Durée estimée : 50 minutes

Un sorcier lance des sorts avec un temps qui dépend de son niveau de compétences. Un directeur (*headmaster*) utilisera 1 unité de temps, un professeur (*professor*) utilisera 3 unités de temps, un diplômé (*graduated*) utilisera 5 unités et un élève (*student*) utilisera 7 unités. Nous t'avons fourni une classe `poo.labo02.Level` pour représenter le concept de niveau.

Déclare une classe `poo.labo02.Wizard`. Cette classe possède un constructeur `public Wizard(String givenName, Level givenLevel)` qui initialise deux champs `name` et `level`. Si `givenName` est un `String` `null` ou blanc, le constructeur le remplace par la valeur « You-Know-Who ».

Si givenLevel est null, le constructeur le remplace par la valeur Level.STUDENT… À moins que vous ayez affaire à « You-Know-Who » qui aura d'office le niveau Level.HEADMASTER.

`Wizard` propose deux méthodes `public String getName()` et `public Level getLevel()` retournant respectivement le nom et le niveau du sorcier recevant l'appel. `Wizard` déclare une troisième méthode d'objet `public String cast(Spell s)` qui retourne :

- La phrase « [nom du sorcier] casts nothing. », si `s` vaut `null` ;
- Sinon, une phrase du type « [nom de ce sorcier] casts [incantation du sortilège] ».

Tu dois appeler la méthode `cast(int)` de l'objet `s` en t'aidant du temps du niveau de ce sorcier. Par ailleurs, pour chaque appel où `s` ne vaut pas **`null`**, `cast(Spell)` augmente un compteur de sortilèges lancés. Quand le compteur atteint le temps du niveau du sorcier, le sorcier passe au niveau suivant et remet son compteur à zéro. Un sorcier passe donc d'étudiant à diplômé après 7 invocations effectives, de diplômé à professeur après 4 invocations supplémentaires et de professeur à directeur après 3 nouvelles invocations.

Valide tes méthodes avec les tests unitaires proposés dans la classe `WizardTest`. Décommente les tests au fur et à mesure.

Si l'horaire le permet, ce travail est à terminer pour la séance de 2 heures suivante.

Correction avec le responsable

Durée estimée : 20 minutes.

Avant de poursuivre, ton responsable présente les corrections pour ces exercices. Les points d'attention suivants seront notamment passés en revue :

- Les champs d'objet sont des variables locales à l'objet. Les méthodes d'objet accèdent directement aux champs de l'objet qui reçoit l'appel.
- Les constructeurs initialisent un nouvel objet en affectant des valeurs à ses champs. Ces valeurs peuvent dépendre des paramètres du constructeur.
- Les constructeurs ont des en-têtes particuliers. Ils n'ont pas de type de retour et portent le même nom que la classe

Exercice 3

Objectifs :

- Déclarer des tableaux d'objets ;
- Manipuler des tableaux d'objets.

Durée estimée : 100 minutes

Tu dois terminer un programme qui permet d'entrainer tes camardes aux invocations de sorts.

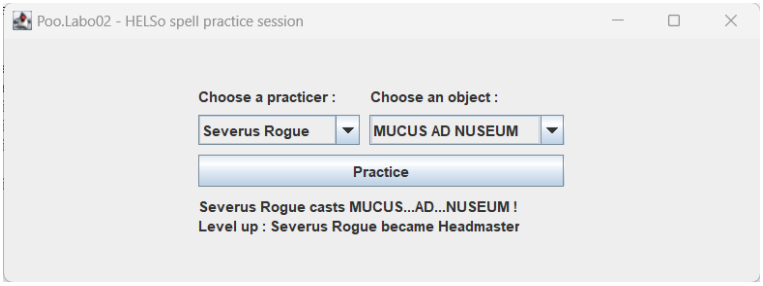

Figure 2 Exemple d'affichage

Voici tes camarades et les sorts à invoquer.

Tes camarades

Nom	Niveau de départ
Albus Dumbledore	Directeur
Severus Rogue	Professeur
Harry Potter	Diplômé
Hermione Granger	Diplômée
Ronald Weasley	Étudiant
« sans nom »	Directeur

Les sorts

Nom
Stupefy
Expecto patronum
Wingardium leviosa
Mucus ad nuseum
Silencio

Complète la classe `poo.lecon02.pratice.SpellPracticeSession`. La classe doit déclarer le tableau de sorciers et celui de sortilèges comme champs d'objet. Tu devras modifier trois méthodes :

- `public String[] getWizardNames()` qui retournera un nouveau tableau composé des noms des sorciers.
- `public String[] getSpellNames()` qui retournera un nouveau tableau composé des noms des sorts.
- `public String[] practice(int wizardIndex, int spellIndex)` qui retourne un tableau composé d'un ou de deux messages. Le premier message correspond au résultat de l'invocation du sort d'indice `spellIndex` par le sorcier d'indice `wizardIndex`. Le second message est ajouté quand le sorcier passe au niveau supérieur après l'invocation. Le second message aura le format `"Level up : <nom du sorcier> became <nom du nouveau niveau>"`.

Écris quelques tests unitaires pour valider tes méthodes. Vérifie notamment que les résultats retournés par `getWizardNames()` et `getSpellNames()` correspondent aux tableaux donnés au début de l'exercice. Vérifie également le cas d'un passage au niveau suivant.

Si l'horaire le permet, ce travail est à terminer pour la séance suivante.

BONUS : en cas de passage au niveau suivant, tout sorcier **devenant** directeur est supprimé de la liste des sorciers. En conséquence, getWizardNames() retourne un tableau ne contenant plus le nom du sorcier devenu Directeur. À la fin, il ne restera plus que Dumbledore et « Tu-Sais-Qui »…

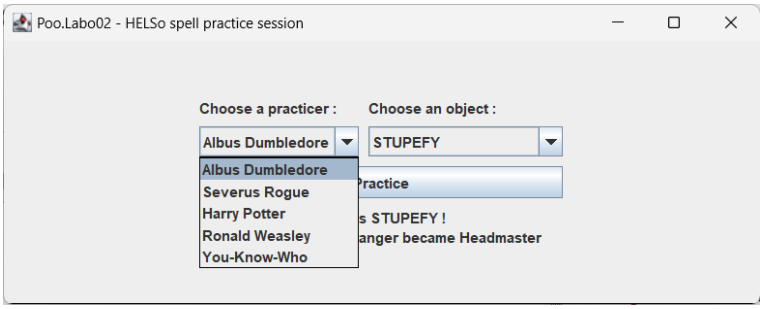

Figure 3 Hermion a disparu de la liste suite à son passage au niveau directrice

Correction avec le responsable

Durée estimée : 30 minutes.

Avant de poursuivre, votre responsable présente les corrections pour l'exercice 3. Le responsable insiste sur le fait qu'un tableau de références a ses éléments initialisés à null, sauf mention du contraire.

Exercice 4

Objectifs : redéfinir des méthodes héritées de la classe Object.

Durée estimée : 30 minutes

Les objets de la classe `Spell` ne changent plus d'état une fois créés : on peut les qualifier d'objets immuables[2]. De tels classes sont de bons candidats à la redéfinition de méthodes héritées de Object.

Redéfinis les méthodes suivantes :

- `public boolean equals(Object o)` qui retourne `true` si o est un objet de la classe `Spell` de même incantation.
- `public String toString()` qui retourne une représentation du sortilège respectant le format `"Spell(incantation: <incantation de cet objet>)"`.

La redéfinition de la méthode equals(Object) passe par plusieurs étapes expliquées par les pages 44 et 45 du syllabus. Consulte-le.

Valide tes méthodes par des tests unitaires. Au minimum, écris les tests suivants :

- Soit un sortilège `s`, on s'attend à ce que `s.equals(s)` retourne `true`.
- Soit un sortilège `s`, on s'attend à ce que `s.equals(null)` retourne `false`.
- Soient un sortilège s et un string `str` quelconque, on s'attend à ce que `s.equals(str)` retourne `false`.
- Soient deux sortilèges `s1` et `s2` de même incantation, on s'attend à ce que `s1.equals(s2)` et `s2.equals(s1)` retournent `true`.
- Soient deux sortilèges `s1` et `s2` d'incantations différentes, on s'attend à ce que `s1.equals(s2)` et `s2.equals(s1)` retournent `false`.

Pour tes tests, veille à créer de nouveaux objets plutôt que d'affecter le contenu d'une variable à une autre variable.

---

[2] Nous reviendrons plusieurs fois sur cette notion pendant le cours. Vous pouvez déjà prendre connaissance du concept en lisant l'article Objets immuables en Java

Exercice 5

Durée estimée : 45 min

Objectif : Définir une énumération d'objets

Dans les exercices précédents, nous avons défini les niveaux à l'aide d'une classe `Level` déclarant des objets « constants ». Tu l'as peut-être remarqué, mais tu ne peux pas instancier cette classe : tu ne peux utiliser que les objets déclarés par `Level`. Pour ce cas de figure, Java propose une seconde construction pour définir des types d'objets : les **`enum`**.

Pour rappel, contrairement aux classes qui définissent les objets par intention, les **`enum`** sont des définitions par extension, c'est-à-dire qu'elles énumèrent **tous** les objets du type. Comme une **`enum`** déclare tous ses objets, vous ne pouvez pas en créer de nouveaux avec l'opérateur **`new`** : vous devez utiliser ceux déclarés par **`l'enum`**.

Transforme la classe `Level` en **`enum`**. Remplace le mot clé **`class`** par **`enum`** et adapte le code en conséquence. Veille à supprimer le code devenu inutile : les méthodes `equals(Object)` et `toString()` ont-elles encore leur raison d'être ? Justifie à partir des propriétés des **`enum`**.

Si tu as bien fait les choses, le reste de ton code s'adaptera à tes modifications.

Correction avec le responsable

Durée estimée : 15 minutes.

Ton responsable présente une correction. Les points d'attention suivants seront notamment passés en revue :

- Java redéfinit la méthode `toString()` d'une énumération pour retourner l'étiquette de l'objet qui reçoit l'appel. Il n'est pas nécessaire de redéfinir `toString()`.
- L'association d'égalité par défaut suffit, car les énumérations ne sont pas utilisables avec l'opérateur new. Il n'est pas nécessaire de redéfinir `equals(Object)`.
- Contrairement aux objets d'une classe, les objets d'une énumération sont utilisables dans une structure **`switch`**.