

In – B1 – 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 de champs, de méthodes d'objet et de constructeurs.
- Redéfinir des méthodes héritées de la classe `java.lang.Object` comme `toString()` et `equals(Object)`
- Définir des enums composés de champs, 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 champs n'est pas requise pour ce laboratoire.

Importer le projet

Durée estimée : 05 min

Importe le projet `poo.lab02`, disponible sur HELMo-Learn. Tu dois obtenir une structure semblable à l'image suivante.

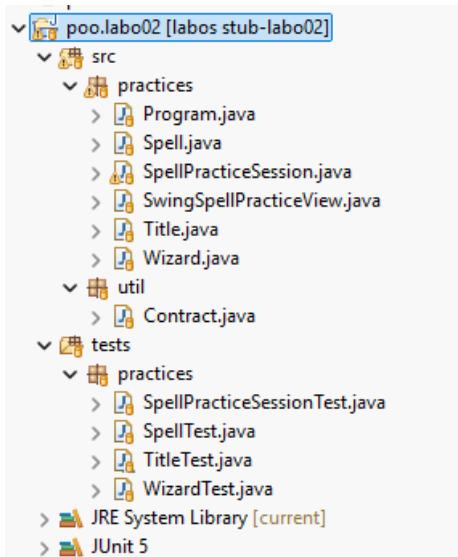


Figure 1 Structure de départ attendue

Exercices

Le laboratoire contient des séries d'exercices de difficultés croissantes. Après chaque série, le responsable propose une solution et répond à tes questions. Les exercices dépendent des exercices précédents. Fais-les 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 temps de pause précis entre chaque mot, pour ne pas en perdre le contrôle.

Exercice 1

⌚ **Objectif :** déclarer une méthode d'objet ⏳ **Durée estimée :** 15 minutes

La classe `practices.Title` définit les titres qu'un sorcier peut atteindre pendant sa carrière. Un titre possède un nom et un temps de pause entre chaque mot. Ce temps de pause définit la vitesse à laquelle un sorcier peut lancer un sortilège : une petite valeur indique une vitesse élevée.

Le tableau suivant liste les titres possibles, du plus prestigieux au plus commun. Chaque titre correspond à un objet constant, déjà déclaré par la classe.

Titre	Temps entre chaque mot
Directeur (headmaster)	1
Professeur (professor)	3
Diplômé (graduated)	5
Étudiant (student)	7

Quand il lance des sortilèges, un sorcier peut passer au titre suivant. Le tableau ci-dessous présente les titres suivants d'un titre donné. Notez que le titre suivant directeur est lui-même.

Titre actuel	Titre suivant
Directeur (headmaster)	Directeur
Professeur (professor)	Directeur
Diplômé (graduated)	Professeur
Étudiant (student)	Diplômé

Déclare une méthode d'objet `Title next()` dans la classe `Title`. Elle retourne le titre suivant de l'objet qui reçoit l'appel. Pour ce faire, élabore un `switch` qui fait correspondre le nom de ce titre aux noms disponibles et qui retourne l'objet correspondant.

⚠ Attention
Utilise les constantes comme résultat. Ne crée pas de nouvel objet.

Écris des tests unitaires dans `TitleTest` qui vérifient que le tableau de correspondance précédent est implémenté.

*** Title n'est pas instanciable**
Tu as probablement constaté que le constructeur de `Title` est privé : sa visibilité est limitée à la classe `Title`. En conséquence, seule cette classe peut créer des objets. Ce choix de conception facilitera la conversion de cette classe en enum dans le dernier exercice.

Exercice 2

Objectif : déclarer une classe d'objet **Durée estimée** : 30 minutes

Déclare une classe `practices.Spell` avec :

- un constructeur `public Spell(String givenIncantation)` qui initialise un champ `incantation` avec `givenIncantation`.
Précondition : `givenIncantation` est différent de null et non blanc.
- Une méthode `public String getIncantation()` qui retourne l'incantation de la formule en lettres majuscules.
- Une méthode `public String cast(int timeBetweenWords)` qui retourne une représentation de l'incantation où :
 - chaque mot est mis en lettres majuscules.
 - Les mots sont séparés par `timeBetweenWords` symboles « . »¹.
 - L'incantation est terminée par « ! ».

Précondition : `timeBetweenWords` est un entier strictement positif.

Astuce

La classe `String` déclare toujours les méthodes d'objet `isBlank()` et `repeat(int)` vus au laboratoire précédent. La classe `java.util.StringJoiner` est utile pour construire des strings à l'aide de séparateurs, d'un préfixe et d'un suffixe... La page 35 du syllabus illustre son fonctionnement.

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

Tests pour `getIncantation()`

Incantation du sortilège	Résultat attendu
"wingardium leviosa"	"WINGARDIUM LEVIOSA"

Tests pour `cast(int)`

Incantation du sortilège	argument de <code>cast(int)</code>	Résultat attendu
"wingardium leviosa"	3	"WINGARDIUM...LEVIOSA !"
"Stupefy"	5	"STUPEFY !"
"Mucus ad nuseum"	2	"MUCUS..AD..NUSEUM !"

Tester la prise en charge d'une précondition

Utilise l'assertion suivante dans un test unitaire pour valider le respect d'une précondition.

```
Assertions.assertThrows(IllegalArgumentException.class, () -> {
    // Code qui enfreint la précondition
});
```

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

Par exemple, si tu veux tester que le constructeur de Spell(String) rejette les strings blancs, écris :

```
Assertions.assertThrows(IllegalArgumentException.class, () -> {
    new Spell(" \t \n");
});
```

Exercice 3

Objectifs :

- Déclarer une classe d'objet
- Faire muter un objet

Durée estimée : 40 minutes

Un sorcier lance des sorts avec un temps entre chaque mot qui dépend de son titre (exercice 1).

Déclare une classe `practices.Wizard` avec :

- un constructeur `public Wizard(String givenName, Title givenTitle)` qui initialise deux champs `name` et `title`. Si `givenName` est un `String null` ou blanc, le constructeur le remplace par la valeur « You-Know-Who ». Si `givenTitle` est null, le constructeur le remplace par la valeur `Title.STUDENT...` À moins que tu sois face à « You-Know-Who » qui a d'office le titre `Title.HEADMASTER`.
- Une méthode `public String getName()` qui retourne le nom de ce sorcier.
- Une méthode `public Title getTitle()` qui retourne le titre de ce sorcier.
- Une méthode d'objet `public String cast(Spell spell)` qui retourne une phrase du type « [nom de ce sorcier] casts [incantation du sortilège] ».

Précondition : `spell` est différent de null.

Tu dois appeler la méthode `cast(int)` de l'objet `spell` en t'aïdant du temps du titre de ce sorcier. Par ailleurs, si la précondition est respectée, `cast(Spell)` augmente un compteur de sortilèges lancés. Quand il atteint le temps associé au titre du sorcier, ce dernier passe au titre 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 au fur et à mesure.



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

Exercice 4

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 camarades aux invocations de sorts.

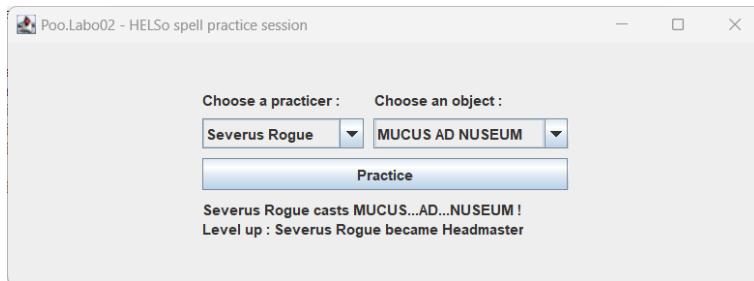
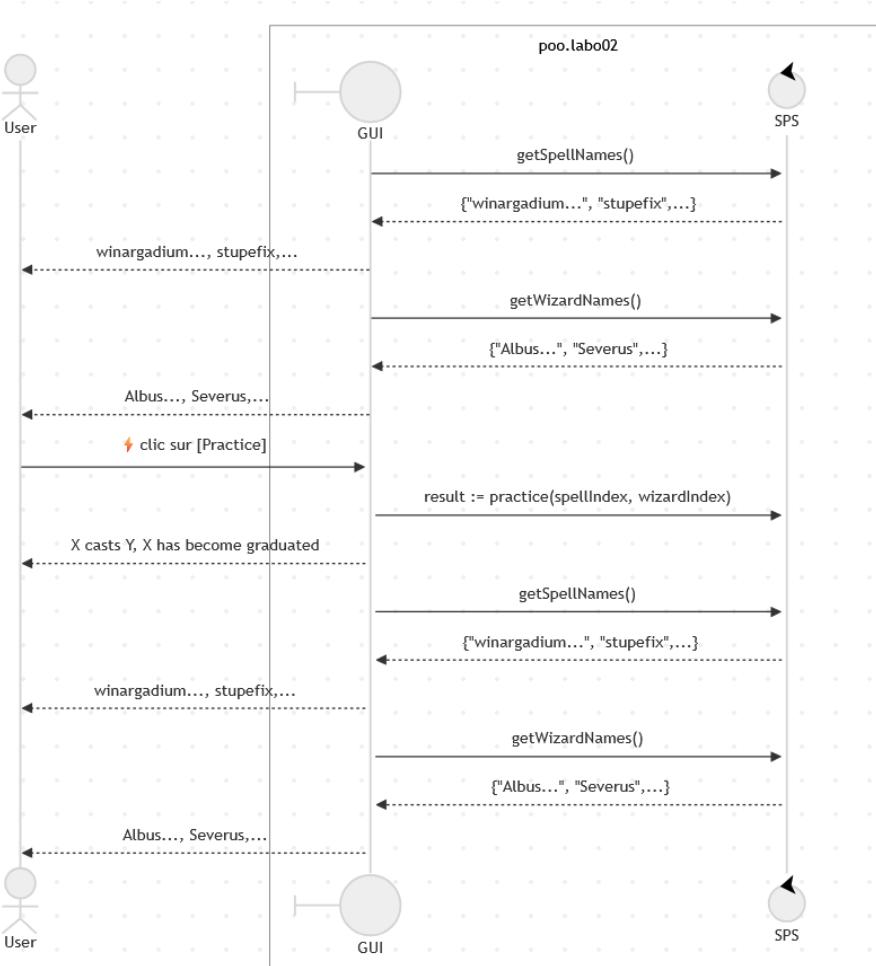


Figure 2 Exemple d'affichage

La figure suivante illustre comment le programme interagit avec l'utilisateur. Tu devras intervenir au niveau du participant SPS qui est un objet de la classe SpellPracticeSession.



Voici tes camarades et les sorts à invoquer.

Tes camarades

Nom	Titre 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 `practices.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 titre supérieur après l'invocation. Le second message aura le format "`Title update : <nom du sorcier> has become <nom du nouveau titre>`".

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 titre suivant.



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



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 »...

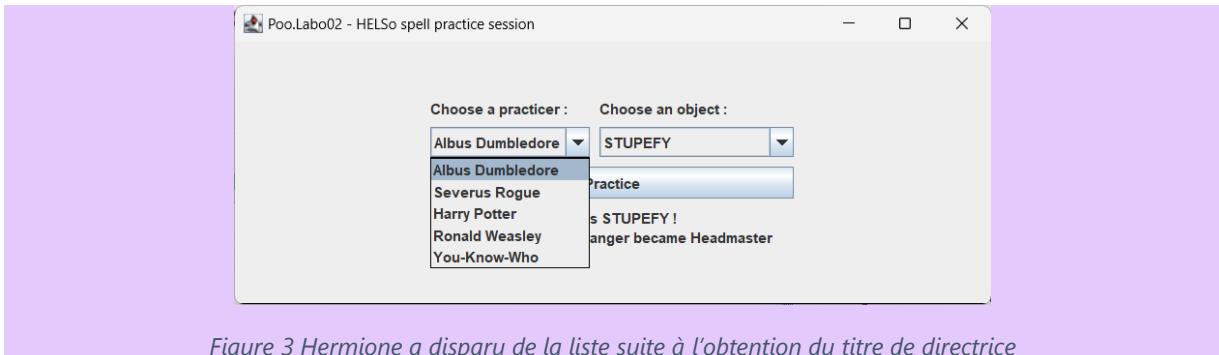


Figure 3 Hermione a disparu de la liste suite à l'obtention du titre de directrice

Exercice 5

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 : ce sont des objets immuables². Leurs classes sont de bons candidats à la redéfinition de méthodes héritées de Object.

Redéfinis les méthodes :

- **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 l'incantation du sortilège.



Redéfinir equals(Object)

La redéfinition de la méthode `equals(Object)` passe par plusieurs étapes expliquées aux pages 59 à 62 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. Tu peux déjà prendre connaissance du concept en lisant l'article [Objets immuables en Java](#)

Exercice 6

Durée estimée : 45 min

Objectif : Définir une enum

Dans les exercices précédents, nous avons défini les titres par une classe `Title` 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 `Title`. 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 : elles énumèrent tous les objets du type. Comme une **enum** déclare tous ses objets, tu ne peux pas en créer de nouveaux avec l'opérateur `new` : tu es limité à ceux déclarés par **l'enum**.

Transforme la classe `Title` en **enum**. Remplace le mot clé `class` par **enum** et adapte le code en conséquence. Supprime 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.