# Info – Bloc 1 – UE09 : Programmation Orientée Objet

**Laboratoire 4**

**Durée prévue :** 4 heures

## Objectifs visés

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

- Définir des relations d'héritage entre les classes
- Redéfinir des méthodes héritées
- Manipuler des objets à l'aide de références dont le type correspond à leur classe mère.

Plus généralement, les étudiants seront capables d'exploiter les mécanismes liés à l'héritage de classes.

# Correction du labo 03

Le responsable propose une correction pour le dernier exercice du labo 03.

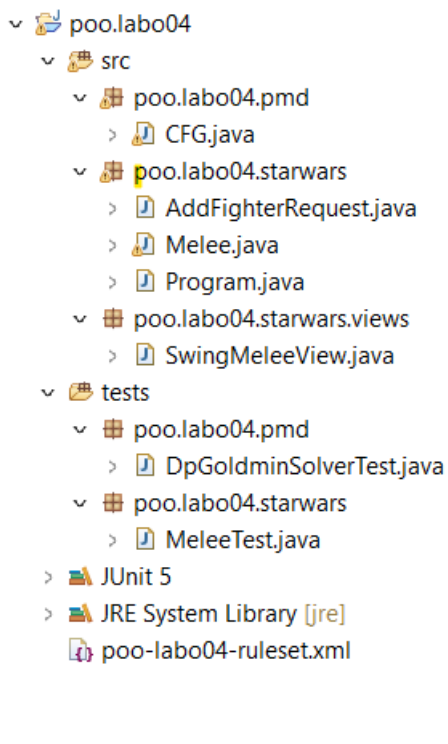
# Créer le projet poo.labo04

Crée un projet `poo.labo04`. Télécharge le fichier poo-labo04-ruleset.xml proposé en annexe et place-le à la racine de ton projet.

Dans le répertoire `src`, crée les paquetages `poo.labo04`, `poo.labo04.pmd`, `poo.labo04.starwars` et `poo.labo04.starwars.views`. Ajoute le fichier *CFG.java* au paquetage `poo.labo04.pmd`. Ajoute les fichiers *Program.java, Melee.java et AddFighterRequest.java* au paquetage poo.labo04.starwars. Ajoute le fichier *SwingMeleeView.java* au paquetage `poo.labo04.starwars.views`.

Ajoute un second répertoire de source appelé `tests` pour tes tests unitaires et crée les paquetages `poo.labo04`, `poo.labo04.pmd` et `poo.labo04.starwars`. Ajoute le fichier *DpGoldmineSolverTest.java* au paquetage poo.labo04.pmd et le fichier *MeleeTest.java* au paquetage `poo.labo04.starwars`.

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



# Découvrir PMD

**Durée estimée :** 15 min

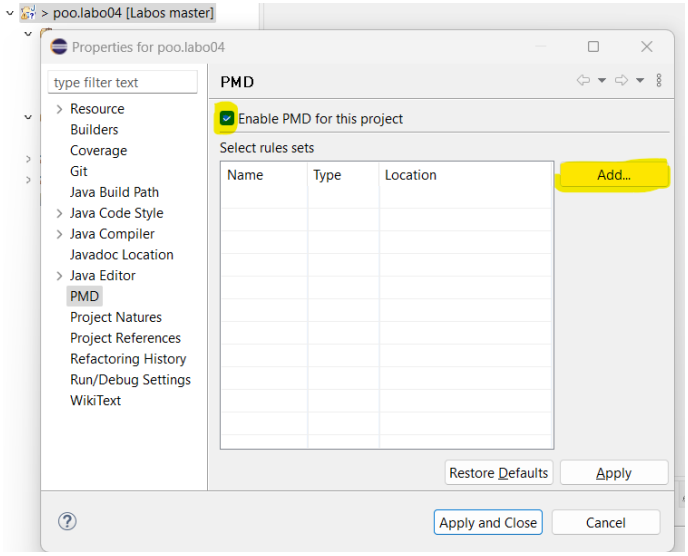
**Objectif :** analyser les alertes PMD pour remanier son code.

À partir de ce laboratoire, nous utiliserons PMD pour évaluer certains aspects de ton code. Les métriques présentées ici interviennent notamment dans l'évaluation de l'activité intégrative.

## Installer le plugin PMD pour Eclipse

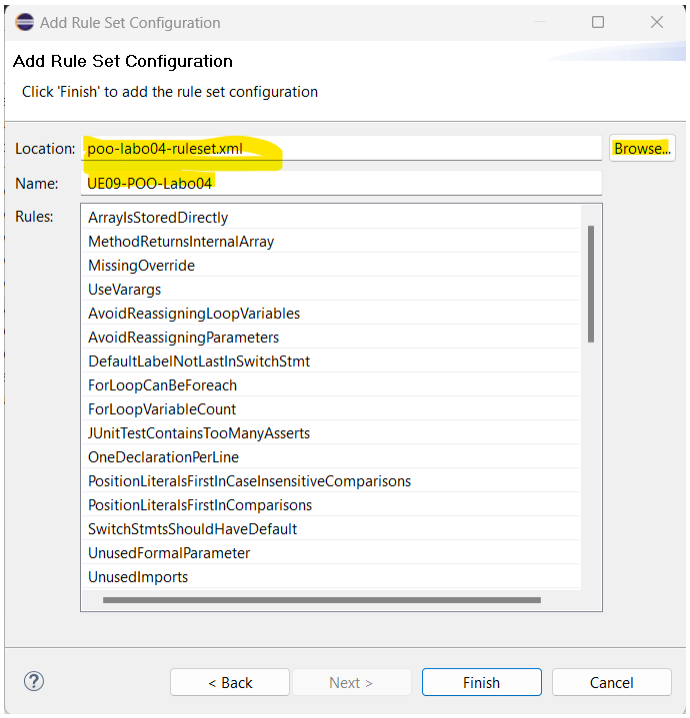
Tu peux installer et utiliser PMD avec Eclipse à l'aide d'un plugin. Pour l'installer, glisse et dépose le lien **INSTALL ECLIPSE-PMD**, disponible sur la page d'installation officielle, sur ton espace de travail Eclipse ouvert[1].

Une fois le plugin installé, va dans les propriétés de ton projet, clique sur l'onglet *PMD*, active PMD pour ce projet et ajoute l'ensemble de règles en cliquant sur le bouton *Add*…




Une boîte de dialogue apparait où tu peux choisir l'item *Project*. Sélectionne ensuite le fichier *poo-labo04-ruleset.xml* et clique sur *Finish*. Applique enfin les changements : ton projet est configuré pour PMD.

[1] Tu devras accepter la licence d'utilisation et plusieurs certificats de sécurité.




## Remanier du code avec PMD

Ouvre le fichier CFG.java. La méthode déclarée résout le problème de la mine d'or à l'aide d'une approche par Programmation Dynamique[2]. Si tu ouvres le panneau des problèmes (*Menu Window>Show View>Problems*), tu constateras de nombreuses alertes. Ces alertes viennent de PMD qui t'indique que plusieurs règles ne sont pas respectées.

Ton objectif est de remanier le code de la classe `CFG` pour supprimer toutes les alertes. Nous te fournissons des tests unitaires dans la classe *DpGoldmineSolverTest.java* pour t'assurer que tu n'introduis aucune régression. Nous te présentons quelques règles appliquées par PMD.

## Règles appliquées

Parmi les règles appliquées, quelques-unes interviennent directement dans l'évaluation de l'activité intégrative. Sois attentif à toute alerte les concernant et corriger rapidement les problèmes constatés. D'autres règles appliquent plusieurs bonnes pratiques à suivre en Java. Nous te recommandons de les suivre.

### Le Nombre de Lignes de Code Non-Commentées (Number of Non-Commented Source Statement, NCSS Count)

*Le NCSS Count* compte le nombre de lignes de code d'une méthode. Cette métrique ne compte pas les lignes de commentaires et les sauts de lignes. Un NCSS élevé signale une méthode qui mérite d'être décomposée en sous-méthodes.

**Seuil d'alerte pour une méthode :** 15.

**Que faire en cas d'alerte ?** Tu dois décomposer la méthode en infraction. Idéalement, ta méthode doit compter au plus 10 NCSS.

---

[2] Nous ferons notre possible pour étudier cette approche pendant les leçons d'algorithme. Consulte cette page pour plus d'information sur ce problème : Gold Mine Problem - TutorialCup

### La complexité cyclomatique (Cyclomatic Complexity ou CC)

*La CC* compte les chemins d'exécution possibles d'une méthode. Une méthode sans structure de contrôle possède une CC de 1, etc.

Une CC élevée indique une méthode pouvant prendre de nombreux chemins d'exécution, ce qui la rend difficile à comprendre. Par ailleurs, la CC est un bon indicateur du nombre de cas de tests à imaginer si vous souhaitez valider tous les chemins d'exécution d'une méthode.

**Attention**

Ne confonds pas nombre de chemins d'exécution d'une méthode et cas de tests à imaginer. Le nombre de chemins d'exécution dépend d'un algorithme, les cas de tests à imaginer dépend du problème à resoudre.

**Seuil d'alerte pour une méthode :** 10.

**Que faire en cas d'alerte ?** Tu dois décomposer la méthode en infraction en identifiant des sous-tâches, idéalement indépendantes. Dans un monde parfait, toute méthode devrait avoir une CC de 5 ou moins.

### Les conventions d'écriture

Nous sommes sensibles aux conventions d'écriture qui facilitent la lecture du code par un autre programmeur (ou par toi quelques mois plus tard). Nous sommes particulièrement attentifs aux formats des noms :

- De paquetages (pas de majuscules)
- De classes (CamelCase)
- De variables (camelCase)
- De constantes (SNAKE_CASE)

### Les commentaires requis

Tu dois documenter ton code, notamment pour les futurs utilisateurs de tes objets. Tu dois documenter à cette fin, ta classe et ses méthodes publiques.

Une bonne documentation de méthode doit mentionner :

- Une courte description (1 phrase) de ce réalise votre méthode ;
- Une courte description des paramètres ;
- Une courte description du résultat.

La documentation d'une méthode devrait également indiquer la précondition, c'est-à-dire la règle que les données doivent respecter, et la postcondition, c'est-à-dire la règle qui lie les données d'entrées aux résultats.

**Note :** Tu n'es pas obligé de documenter les accesseurs et les mutateurs, sauf si ces derniers ont des effets de bords inattendus (mutation d'attributs non-liés à la propriété) ou si les paramètres doivent respecter une précondition particulière.

Documente également ta classe en mentionnant ses responsabilités, c'est-à-dire les informations ou les tâches qui lui sont affectées. Indique également les invariants d'un objet de ta classe, c'est-à-dire les règles qui sont respectées par l'objet pendant sa vie.

## Correction avec le responsable

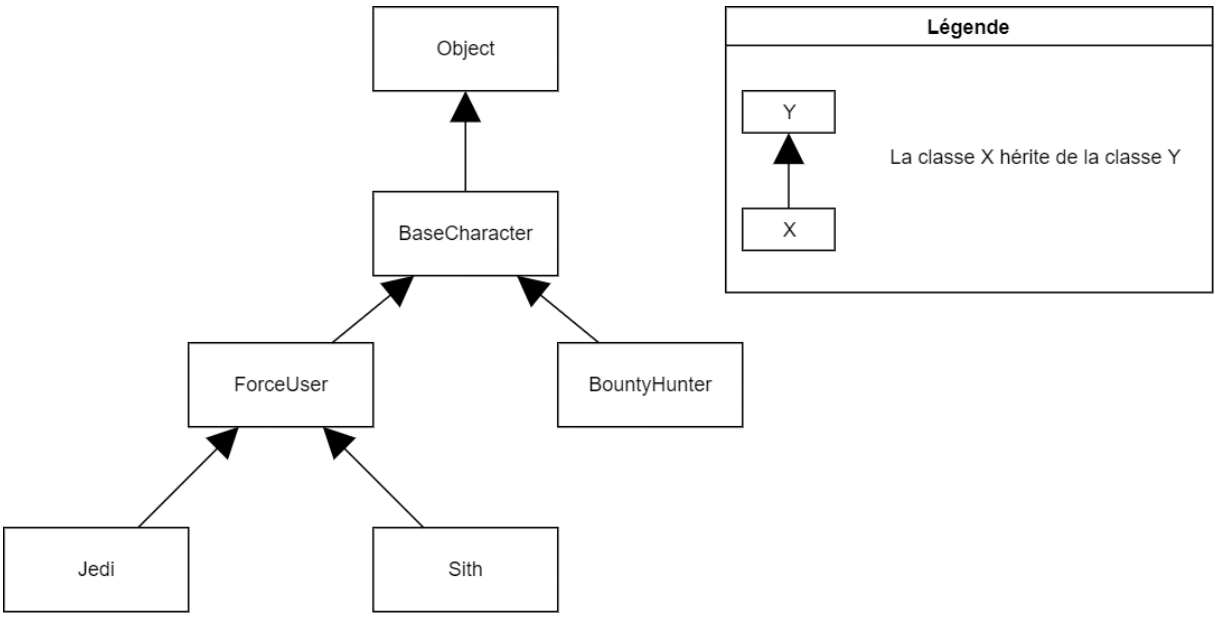
**Durée estimée :** 15 minutes

Le responsable présente quelques remaniements qui font disparaître les alertes. Il montre notamment comment la décomposition d'une méthode en sous-méthodes aident à réduire les métriques CC et NCSS.

# Exercices

Le laboratoire propose des séries d'exercices de difficultés croissantes. Après chaque série, le responsable propose une solution et répond à tes questions. Le premier exercice fait office d'échauffement. Les exercices suivants débouchent sur la réalisation d'un programme exécutable. Il vaut mieux les faire dans l'ordre.

Dans ce laboratoire, nous souhaitons simuler une bataille entre des utilisateurs de la Force. Nous allons à cette fin implémenter la hiérarchie de classe suivante.




## Exercice 1 : les personnages de base

**Durée estimée** : 15 minutes

**Objectif** : déclarer une classe

Déclare la classe `BaseCharacter` dans le paquetage `poo.labo04.starwars`.

Tous les personnages de l'univers `Star Wars` appartiennent à la classe `BaseCharacter`. Chaque personnage possède un nom (`Name`) et des points de vie (`Hit Points`, `HP`) à initialiser à l'aide d'un constructeur (pense à valider les paramètres). Le nom et les points de vie sont des données accessibles en lecture : déclare les accesseurs correspondants.

La classe `BaseCharacter` définit également deux méthodes d'objet :

- `public void loseHP(int points)` qui retire `points` points aux points de vie. Si le nombre de points de vie restant est négatif, l'accesseur correspondant retournera 0.
- `public boolean isAlive()` qui retourne `true` si les points de vie sont `> 0`.

Enfin, elle redéfinit également `toString()` pour y inclure le nom du personnage et son nombre de points de vie.

Valide ta classe en implémentant les tests unitaires suivants dans une classe BaseCharacterTest.

```
Un personnage de base
Indique son nom et ses points de vie
Étant donné le personnage « Senateur Organa » possédant 10 points de vie.
Alors son nom est « Senateur Organa »
Et ses points de vie valent 10.

Remplace un nom null ou blanc par le nom « Inconnu »
Étant donné le personnage null possédant 20 points de vie.
Alors son nom est « ? »
Et ses points de vie valent 20

Remplace les points de vie négatifs par leurs valeurs absolues
Étant donné le personnage « Senateur Organa » possédant -15 points de vie.
Alors son nom est « Senateur Organa »
Et ses points de vie valent 15.

Perd des points de vie
Étant donné le personnage « Senateur Organa » possédant 10 points de vie.
Quand il perd 9 points de vie.
Alors il est vivant
Et ses points de vie valent 1

Meurt quand il perd trop de points de vie
Étant donné le personnage « Senateur Organa » possédant 5 points de vie.
Quand il perd 6 points de vie.
Alors il n'est pas vivant
Et ses points de vie valent 0

Fournit une représentation en String
Étant donné le personnage « Senateur Organa » possédant 15 points de vie.
Alors sa représentation en string vaut « BaseCharacter(name: Senateur Organa,
HP: 15) »
```

## Exercice 2 : les utilisateurs de la force

**Durée estimée** : 30 minutes

**Objectifs** :

- Déclarer une classe héritant d'une classe autre que `Object`.

- Ajouter une méthode.

Parmi les personnages de Star Wars, nous distinguons les utilisateurs de la Force. Un utilisateur de la Force peut l'utiliser sur un autre personnage et lui causer des dégâts. Ces dégâts correspondent aux points de dégâts (Damage Points ou DP) de l'utilisateur qui projette. Cette caractéristique doit être accessible en lecture à l'aide d'un accesseur.

Déclare la classe `ForceUser` dans le paquetage `poo.labo04.starwars`. Cette classe hérite de `BaseCharacter` et définit une méthode `String useForceOn(BaseCharacter target)` qui fait perdre `damagePoints` à `target` **et** retourne un string respectant le format `"{name} utilise la Force sur {target.name}. Dégâts causés : {damagePoints}."`.

Définissez la classe de sorte que les tests suivants réussissent. Implémente-les dans une classe ForceUserTest.

```
Un utilisateur de la force
Est une instance des personnages de base
Étant donné l'utilisateur de la force « Leia Organa » possédant 30 HP et 5 DP.
Alors cet utilisateur est une instance de la classe BaseCharacter.

Indique son nom, ses HP et ses DP
Étant donné l'utilisateur de la force « Leia Organa » possédant 30 HP et 5 DP.
Alors son nom est « Leia Organa ».
Et ses points de vie valent 30.
Et ses points de dégâts valent 5.

N'utilise pas la Force sur un inconnu
Étant donné l'utilisateur de la force « Leia Organa » possédant 30 HP et 5 DP.
Quand il utilise la force sur null
Alors le résultat retourné vaut « Leia Organa n'utilise pas la Force. »

Utilise la Force sur d'autres personnages
Étant donné l'utilisateur de la force « Leia Organa » possédant 30 HP et 5 DP
Et le personnage de base « Storm Trooper 1 » possédent 15 HP.
Quand Leia utilise la force sur le stormtrooper
Alors le résultat retourné vaut « Leia Organa utilise la Force sur
Storm Trooper 1. Dégâts causés : 5 »
Alors les points de vie du stormtrooper valent 10.
```

**Points d'attention**

- La classe `ForceUser` initialise-t-elle la partie héritée à l'aide de `super` ?
- L'héritage aide à réduire les répétitions de code. Veille notamment à ce que les attributs de la classe `BaseCharacter` ne soient pas déclarés une seconde fois dans `ForceUser`.
- Plutôt que d'accéder aux attributs définis par `BaseCharacter`, appelle la méthode `loseHP(int)`. Par ailleurs, appelle les accesseurs définis dans la classe mère.

## Exercice 3 : les chevaliers Jedi

**Durée estimée** : 30 minutes

**Objectif** : redéfinir une méthode en appelant la version héritée.

Les Jedi forment une catégorie spéciale d'utilisateurs de la Force : ils adoptent une attitude passive, pleine de recul, vis-à-vis d'Elle. Ils sont cependant capables d'utiliser la Force sur des ennemis et, lorsqu'ils sont acculés, ils peuvent utiliser la rage de la Force (*force fury*).

Déclare la classe des `poo.labo04.starwars.Jedi`. Cette classe hérite de la classe `ForceUser` et redéfinit la méthode `String useForceOn(BaseCharacter target)` de sorte que, si les points de vie d'un Jedi sont <= 2, ses points de dommage soient temporairement multipliés par dix. Dans ce cas, la méthode retourne un string respectant le format `"{this.name} utilise la rage de la force sur {target.name}. Dégâts causés : {this.damagePoints*10}."`. **Attention,** le multiplicateur disparaît définitivement après la première utilisation.

Valide ta classe en implémentant les tests unitaires suivants dans une classe JediTest.

```
Un Jedi
Est une instance des utilisateurs de la Force
Étant donné le Jedi « Mace Windu » possédant 300 HP et 25 DP.
Alors ce Jedi est une instance de la classe ForceUser.

Utilise la force
Étant donné le Jedi « Mace Windu » possédant 300 HP et 25 DP.
Et le personnage de base « Storm Trooper 2 » possédent 15 HP.
Quand Mace Windu utilise la Force sur Storm Trooper 2.
Alors le résultat retourné vaut « Mace Windu utilise la Force sur
Storm Trooper 2. Dégâts causés : 25. »
Et le Storm Trooper est mort.

N'utilise pas la Force sur un inconnu
Étant donné le Jedi « Mon Li-esi » possédant 18 HP et 25 DP.
Quand Mon Li-esi  utilise la Force sur null.
Alors le résultat retourné vaut « Mon Li-esi n'utilise pas la Force. »

N'utilise pas la rage de la Force sur un inconnu
Étant donné le Jedi « Mon Li-esi » possédant 18 HP et 25 DP.
Et Mon Li-esi perd 16 points de vie.
Quand Mon Li-esi utilise la Force sur null.
Alors le résultat retourné vaut « Mon Li-esi n'utilise pas la Force. »

Utilise la rage de la Force quand il est sur le point de mourir
Étant donné le Jedi « Mace Windu » possédant 300 HP et 25 DP.
Et le personnage de base « AT-AT Walker » possédant 1000 HP.
Et Mace Windu perd 298 points de vie.
Quand Mace Windu utilise la Force sur AT-AT Walker.
Alors le résultat retourné vaut « Mace Windu utilise la rage de la Force sur
```

```
AT-AT Walker. Dégâts causés : 250. »
Et AT-AT Walker possède 750 points de vie.

Utilise la rage de la Force une seule fois
Étant donné le Jedi « Mace Windu » possédant 300 HP et 25 DP.
Et le personnage de base « AT-AT Walker » possédant 1000 HP.
Et Mace Windu perd 298 points de vie.
Quand Mace Windu utilise la Force sur AT-AT Walker deux fois.
Alors le résultat de la seconde utilisation vaut « Mace
Windu utilise la Force sur AT-AT Walker. Dégâts causés : 25. »
Et AT-AT Walker possède 725 points de vie.
```

**Points d'attention**

- La classe `Jedi` initialise-t-elle la partie héritée à l'aide de `super` ?
- L'héritage aide à réduire les répétitions de code. Ta solution doit éviter les répétitions. Plutôt que de copier le comportement défini par `ForceUser.useForceOn(BaseCharacter)`, appelle cette version de la méthode à l'aide de `super`.

**Si l'horaire le permet, cette séquence est à terminer pour la séance suivante.**

## Correction avec le responsable

**Durée estimée :** 30 minutes

Après ces exercices, le responsable présente une solution pour les exercices 1,2 et 3. Les points d'attention suivants sont passés en revue :

- Redéfinir une méthode exige de reprendre l'en-tête d'une méthode héritée, accessible à la classe descendante et non-finale. L'annotation `@Override` évite de confondre redéfinition et surcharge.
- Initialiser la partie héritée se faire en appelant super(param). Les règles sont identiques à l'appel d'un constructeur dans un autre constructeur.
- Dans la redéfinition d'une méthode `m(param)`, appeler `super.m(param)` exécute la version héritée de `M`.
- L'héritage est un mécanisme de réutilisation de code. Les membres déclarés dans la super-classe, en particulier les attributs, ne devraient pas être redéclarés dans la sous-classe.

## Exercice 4 : les Sith

**Objectif visé** : redéfinir une méthode en appelant la version héritée.

**Durée estimée** : 30 minutes

Contrairement à l'ordre Jedi, les `Sith` forment un groupe d'utilisateurs de la Force qui adoptent une attitude impulsive vis-à-vis de la Force. Ils peuvent utiliser la Force de trois façons :

- Ils la projettent comme tous les utilisateurs de la force ;
- Ils peuvent étrangler (*Force Choke*) ce qui multiplie les dégâts par 2 ;
- Ils peuvent lancer des éclairs (*Force Lightning*) ce qui multiplie les dégâts par 5.

Déclare la classe `Sith` dans le paquetage `poo.labo04.starwars`. Cette classe redéfinit la méthode `String useForceOn(BaseCharacter target)` de sorte que :

- Tous les 3 appels, elle étrangle sa cible ;
- Tous les 5 appels, elle lance des éclairs sur sa cible.

En cas de conflits (exemple : lors du 15ème appel), l'attaque la plus forte a la priorité. En cas d'étranglement, la méthode retourne un string respectant le format `"{this.name} étrangle {target.name}. Dégâts causés : {this.damagePoints*2}."`. En cas d'éclairs, la méthode retourne un string respectant le format `"{this.name} lance des éclairs sur {target.name}. Dégâts causés : {this.damagePoints*5}."`.

Définissez la classe de sorte que les tests suivants réussissent. Implémente-les dans la classe SithTest.

```
Un Sith
Est une instance des utilisateurs de la Force
Étant donné le Sith « Darth Sidious » possédant 305 HP et 21 DP.
```

**Alors** ce Sith est une instance de la classe ForceUser.

***Utilise la Force***
**Étant donné** le Sith « Darth Sidious » possédant 305 HP et 21 DP.
**Et** le Jedi « Mace Windu » possédant 300 HP et 25 DP.
**Quand** Darth Sidious utilise la Force sur Mace Windu.
**Alors** le résultat retourné vaut « Darth Sidious utilise la Force sur Mace Windu. Dégâts causés : 21. »
**Et** Mace Windu possède 279 points de vie.

***Utilise l'étranglement toutes les 3 utilisations***
**Étant donné** le Sith « Darth Sidious » possédant 305 HP et 21 DP.
**Et** le Jedi « Mace Windu » possédant 300 HP et 25 DP.
**Quand** Darth Sidious utilise la Force sur Mace Windu *3 fois*.
**Alors** le dernier résultat retourné vaut « Darth Sidious étrangle Mace Windu. Dégâts causés : 42. »
**Et** Mace Windu possède 216 points de vie.

***Utilise les éclairs toutes les 5 utilisations***
**Étant donné** le Sith « Darth Sidious » possédant 305 HP et 21 DP.
**Et** le Jedi « Mace Windu » possédant 300 HP et 25 DP.
**Quand** Darth Sidious utilise la Force sur Mace Windu *5 fois*.
**Alors** le dernier résultat retourné vaut « Darth Sidious lance des éclairs sur Mace Windu. Dégâts causés : 105. »
**Et** Mace Windu possède 90 points de vie.

***Favorise les éclairs à l'étranglement***
**Étant donné** le Sith « Darth Sidious » possédant 305 HP et 21 DP.
**Et** le Jedi « Mace Windu » possédant 300 HP et 25 DP.
**Quand** Darth Sidious utilise la Force sur Mace Windu *15 fois*.
**Alors** le dernier résultat retourné vaut « Darth Sidious lance des éclairs sur Mace Windu. Dégâts causés : 105. »

***N'utilise pas la Force sur un inconnu***
**Étant donné** le Sith « Darth Vader » possédant 400 HP et 50 DP.
**Et** le personnage de base « Z-6PO » possédant 150 HP.
**Quand** Darth Vader utilise la Force sur null.
**Alors** le résultat vaut « Darth Vader n'utilise pas la Force. »

***N'utilise pas l'étranglement sur un inconnu***
**Étant donné** le Sith « Darth Vader » possédant 400 HP et 50 DP.
**Et** le personnage de base « Z-6PO » possédant 150 HP.
**Et** Darth Vader utilise la Force sur Z-6PO *2 fois*.
**Quand** Darth Vader utilise la Force sur null.
**Alors** le résultat vaut « Darth Vader n'utilise pas la Force. »

***N'utilise pas les éclairs sur un inconnu***
**Étant donné** le Sith « Jar Jar Binks » possédant 305 HP et 21 DP.
**Et** le personnage de base « R2D2 » possédant 1500 HP.

```
Et Jar Jar Binks  utilise la Force sur R2D2 14 fois.
Quand Jar Jar Binks  utilise la Force sur null.
Alors le résultat vaut « Jar Jar Binks n'utilise pas la Force. »
```

**Points d'attention**

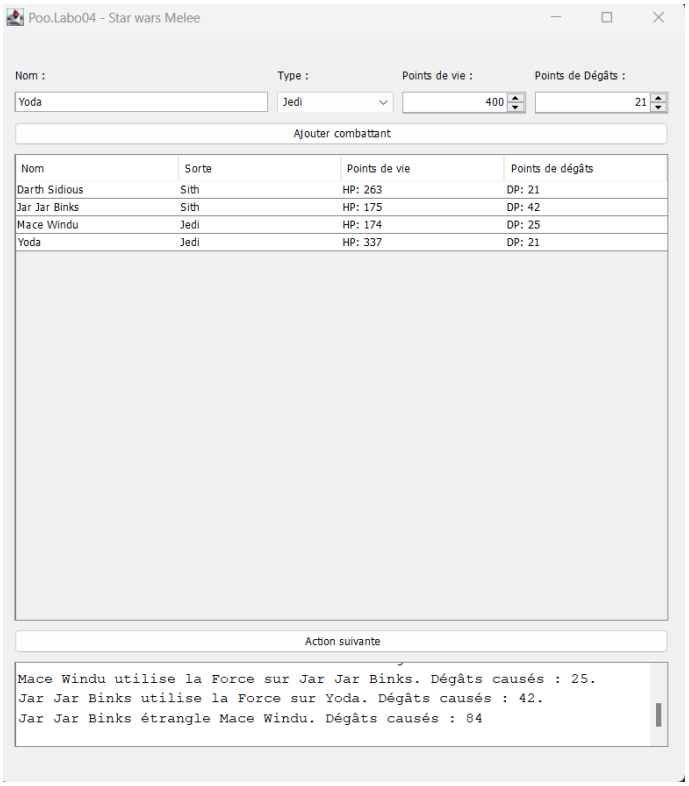
- La classe `Sith` initialise-t-elle la partie héritée à l'aide de `super` ?
- L'héritage aide à réduire les répétitions de code. Ta solution doit éviter les répétitions. Plutôt que de copier le comportement défini par `ForceUser.useForceOn(BaseCharacter)`, appelle cette version de la méthode à l'aide de `super`.

## Exercice 5 : la mêlée

**Objectifs visés** : manipuler des objets de classes différentes, mais exposant les mêmes méthodes.

**Durée estimée** : 60 minutes

Il est temps que les Jedis et les Sith s'affrontent !




Complète la classe `poo.labo04.Melee` afin qu'elle gère de nouveaux combattants et les fasse combattre :

- L'accesseur `String[] getFightersKind()` retourne les types de combattants qu'on peut créer. A priori, on peut créer des Jedi et des Sith. Cette méthode est appelée une seule fois.
- L'accesseur `String[][] getFightersArray()` retourne les combattants vivants sous forme de tableaux. Pour chaque combattant, la méthode retourne son nom, sa classe

concrète, ses points de vie et ses points de dégâts. Cette méthode est appelée après chaque ajout et chaque demande d'action.

- L'accesseur `int getAlivesCount()` retourne le nombre de combattant vivants. Cette méthode est appelée après chaque ajout et chaque demande d'action.
- L'accesseur `String getLastMessage()` retourne les informations relatives à la dernière action valable exécutée. Cette méthode est appelée après chaque demande d'action.
- La méthode `void addFighter(AddFighterRequest args)` crée un utilisateur de la force correspondant à la requête. Le paramètre args contient les données nécessaires à la création : le type de combattant souhaité, son nom, ses points de vie et de dégâts. Cette méthode est appelée pour gérer chaque demande d'ajout.
- La méthode `void makeNextAction()` choisit un attaquant et une cible aléatoirement (veille à ce que l'attaquant et la cible soient deux objets différents) puis demande à l'attaquant d'utiliser la Force sur la cible. La méthode met à jour le dernier message :
    - Le dernier message commence par le résultat de l'utilisation de la Force ;
    - Si la cible décède, la méthode ajoute le message « … est mort ! » ;
    - Si il reste un seul combattant vivant, la méthode ajoute le message « Le vainqueur est … ! ».

**Attention,** `makeNextAction()` ne devrait rien faire s'il reste moins de deux combattants vivants.

Nous te fournissons des tests unitaires pour valider ta classe. Une correction sera proposée au début de la séance suivante.

## Pour les plus rapides

**Objectifs visés :**

- Ajouter une méthode au bon niveau d'abstractions
- Définir une association d'héritage entre deux classes
- Redéfinir une méthode

**Durée estimée :** 45 minutes

Les chasseurs de primes (*Bounty Hunter*) sont des personnages de l'univers Star Wars qui n'utilisent pas la Force, mais qui peuvent, en revanche, abattre d'un seul coup toutes les entités qu'ils chassent. En outre, un chasseur de prime porte une armure qui réduit les dégâts de moitié.

Déclare la classe `BountyHunter`. Cette classe hérite de la classe `BaseCharacter`. Outre le nom et les points de vie, un chasseur de prime possède une résistance qui est une valeur entière comprise entre 2 et 4.

`BountyHunter` redéfinit la méthode `loseHP(int points)` pour diviser les points par la résistance. Elle définit également une méthode `String hunt(BaseCharacter target)` qui retire tous les points de vie de l'entité reçue en paramètre. Cette méthode retourne un message : `"{this.name} a capturé {target.name}."`.

Valide le comportement des chasseurs de prime à l'aide de tests unitaires.

Ajoute le chasseur de prime `"Le Mandalorien"` aux combattants… qui devra probablement changer de type. Quand le programme sélectionne un attaquant, il devra déterminer la classe concrète de l'attaquant :

- Le Mandalorien chassera sa cible ;
- Les utilisateurs de la Force se comporteront comme avant.

**Un peu plus loin :** tu peux constater que, le nom mis à part, les méthodes `hunt(BaseCharacter)` et `useForceOn(BaseCharacter)` ont le même en-tête. Il y a probablement matière à généraliser…