28/04/2020

Simona Arizanova - Mickael PEEREN

SuperHero

La création de classe fait par les grands

Table des matières

[Tutoriel Classe Objet (Java / BlueJ) 1](#_Toc38974825)

[**1.** **Créer une classe** 1](#_Toc38974826)

[**2.** **Tester les méthodes d’une classe** 4](#_Toc38974827)

[**3.** **Liens entre deux classes** 9](#_Toc38974828)

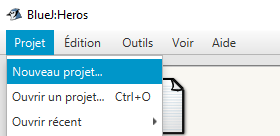
# Tutoriel Classe Objet (Java / BlueJ)

Bienvenue à vous super héros, je suis Java-man, et je vais vous apprendre le Java et les concepts objets avec l’aide de BlueJ.

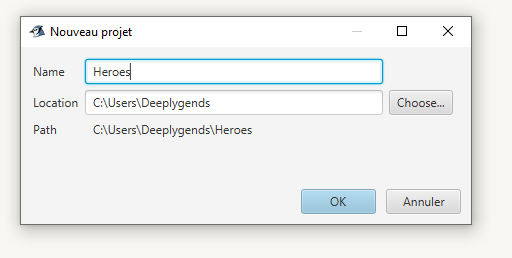
Vous allez m’aider à créer un super héro afin de combattre les méchants.

## **Créer une classe**

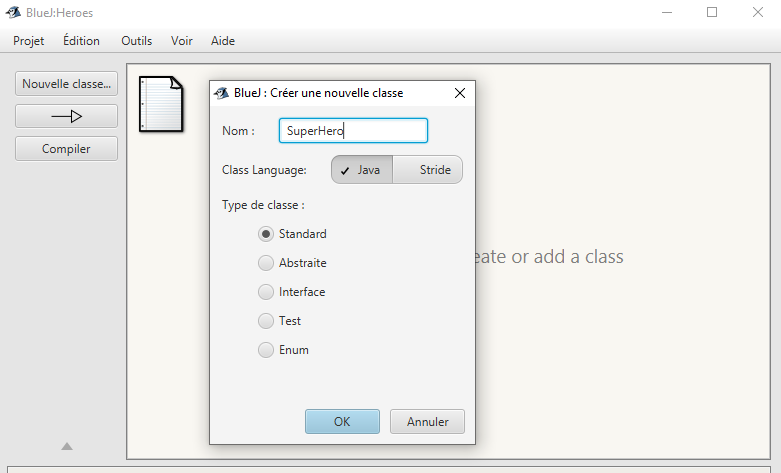
Créer un nouveau projet dans **BlueJ** via **Projet** > **Nouveau Projet …**



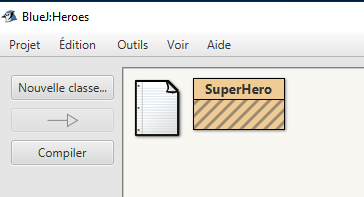
Entre le nom de projet que vous voulez (Avengers, Heroes, Rick et Morty, etc …)



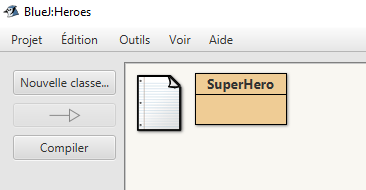
Ensuite, afin de créer notre équipe (constituée de super héros je vous le rappelle) nous devons créer la classe SuperHero.



Félicitations, jeune padawan, la classe SuperHero vient d’être créée mais …… elle est toute barrée.



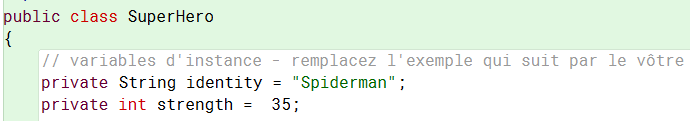
Pas de panique, Compile-man est ici pour vous aider. Cliquez sur compiler et Tadam….



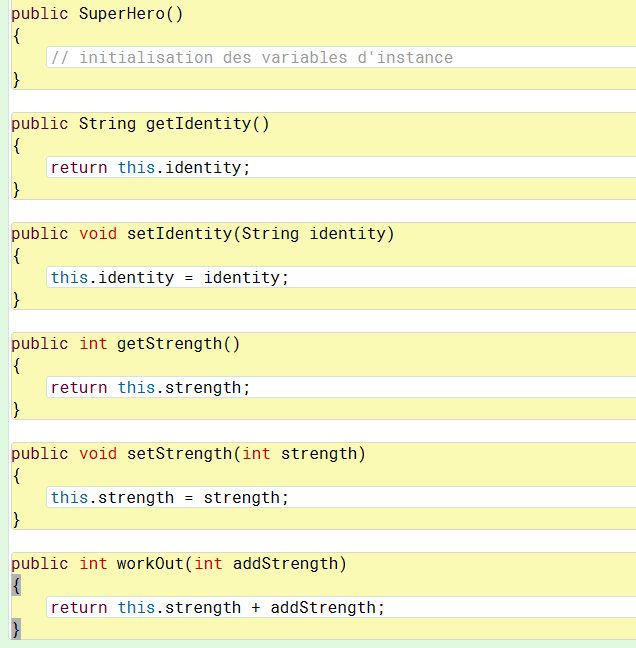
Et si nous allons voir ce qu’il y a dans notre classe :



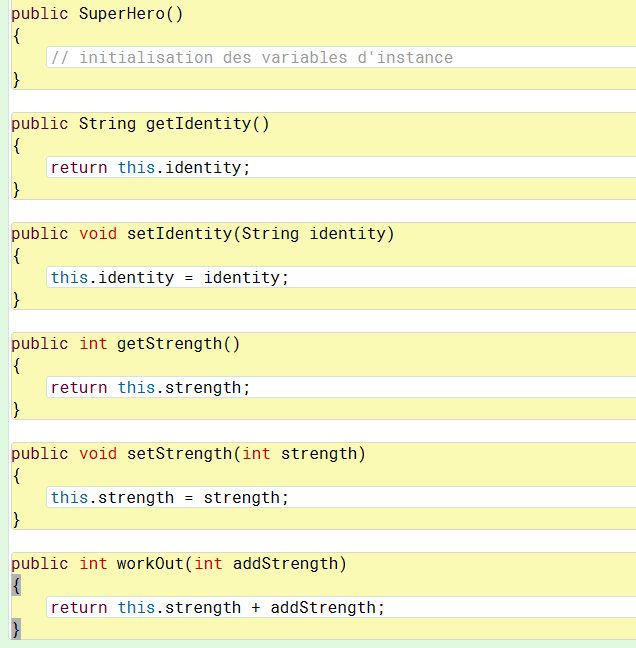
Cela ne représente pas vraiment un super-héros, apportons quelques modifications. Un Super-héros possède une identité et une certaine force. Ajoutons un champ identity de type string (chaine de caractère) et un champ strength de type int (entier). Ces deux champs seront privés.



Ensuite nous allons ajouter les getter (méthodes d’accès pour récupérer une donnée) et les setter ( pour initialiser une donnée privée) de nos deux champs.



Cela commence à ressembler à un super-héros. Mais pour rester un super-héros, il faut s’entrainer dur. Ajouter une méthode workOut qui prend en paramètre un entier (la force gagnée pendant l’entrainement) et qui va nous renvoyer la force totale du super-héros après son entrainement.

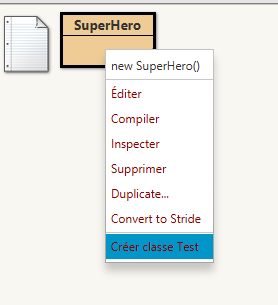


Très bien, notre super-héros est fort et entrainé. Mais on devrait vérifier s’il s’est bien entrainé non ? Même les super-héros trichent parfois …

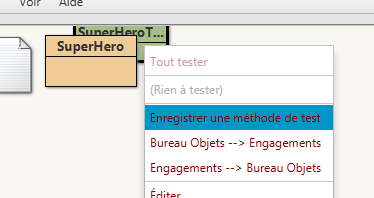
## **Tester les méthodes d’une classe**

Créons une classe de test pour tester notre méthode workOut :

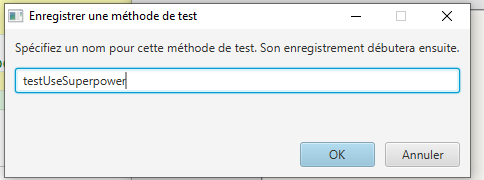
Clique droit > Créer classe Test



La classe de test apparait en vert, encore une fois : Clique droit dessus > Enregistrer une méthode de test.



Bien nommer notre méthode de test afin de bien savoir ce qu’elle teste.

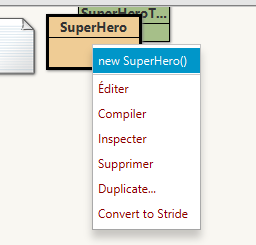


Tiens, tiens. L’enregistrement est lancé, à partir de maintenant tout ce que tu feras sera enregistré (Big Brother est là !).

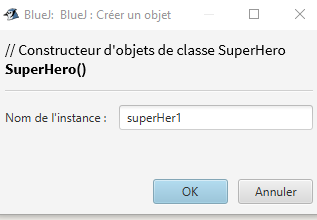


Pour tester l’entrainement d’un super-héros il faut créer une instance de super-héros :

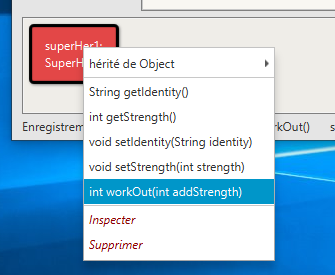
Clique droit > new SuperHero()



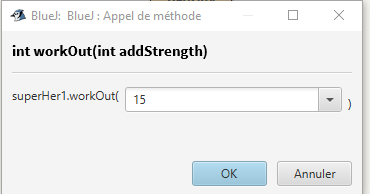
On nomme la variable comme on veut (ici superHer1)



Et enfin on lance la methode workOut de superHer1, notre premier super-héros

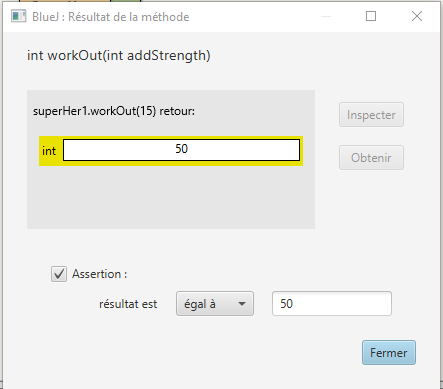


Après son entrainement, on imagine qu’il va gagner 15 de force :



De base, un nouveau super-héros a 35 de force, donc on devrait obtenir 50. On écrit 50 à coté de « **égal à ».**

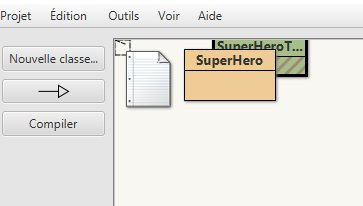
Encadré en jaune est le résultat de notre opération (ouf c’est le bon d’ailleurs)



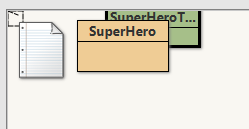
Notre test est terminé donc ………… On clique sur Terminer



On oublie pas de compiler



Ce qui nous enlève les barres de notre joli carré vert.



Et on clique sur Exécuter les tests



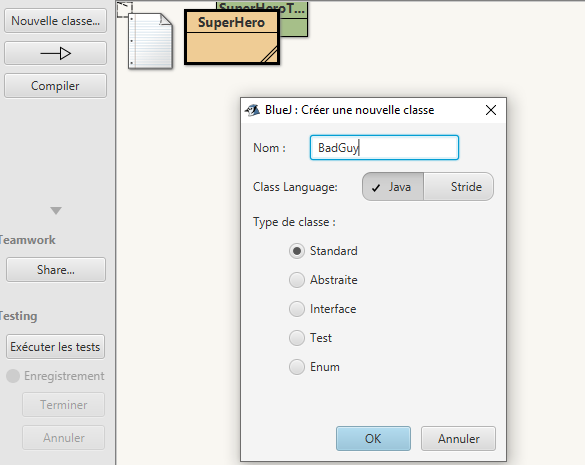
Et on obtient le résultat de notre test



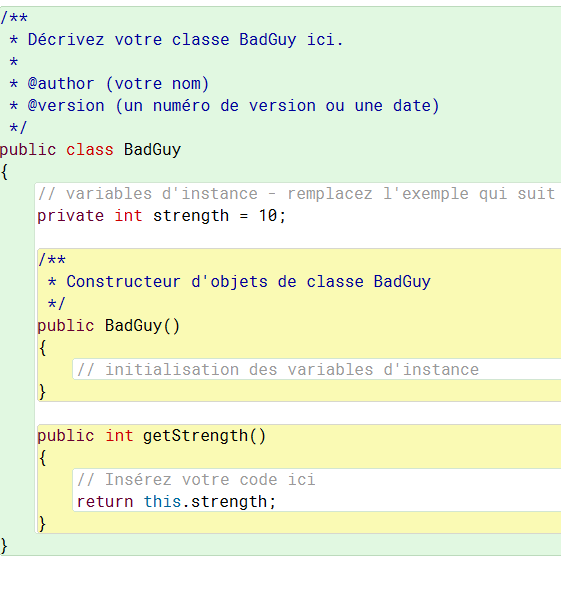
Yyyeeaaaaahhhh !! Première victoire de notre super-héros (C’est en grande partie grâce à toi)

## **Liens entre deux classes**

Passons aux choses sérieuses : Le méchant arrive pour mettre des bâtons dans les roues de notre super-héros (Mouhahahaa). Créons une classe BadGuy.



Le méchant, il n’est jamais très malin (donc on ne va pas lui mettre trop de champs). En général, c’est une brute donc il n’aura qu’un champ strength (avec son accesseur getStrength)

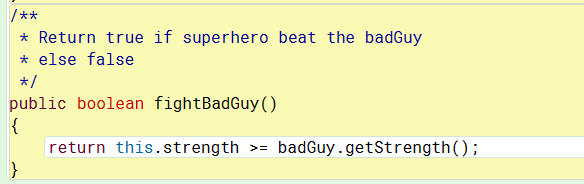


La ville n’est plus sûre depuis que notre méchant est arrivé. Notre super-héros va devoir lui apprendre les bonnes manières. Dans notre classe SuperHero, nous allons ajouter un champ BadGuy. En effet, il vient sans vergogne dans la ville de notre héro.

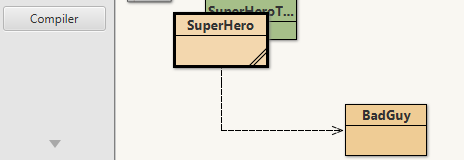


Nous venons d’associer un méchant a notre héros. Maintenant, place à la bagarre. Notre super-héros va essayer de se débarrasser de notre super-méchant. Nous allons créer une méthode fightBadGuy dans notre classe SuperHero pour qu’il combatte son méchant.

La méthode compare la force des deux opposants, et renvoie « True » si notre super-héros a une force supérieure à son méchant (et il lui botte allégrement les fesses). Sinon, on renvoie false.



Compile-man vous demande de compiler. Exécution !

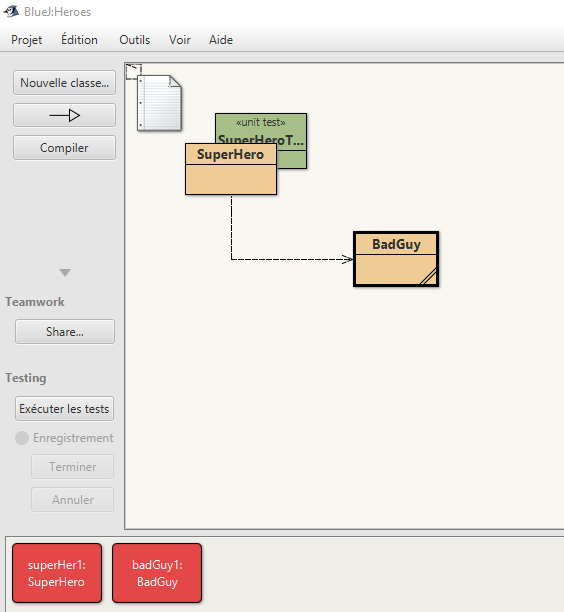


12

On avance bien. Essayons de sauvegarder une instance de Superhero avec une instance de BadGuy dans notre fichier de test.

Tout d’abord, sur la classe SuperHero, Clique Droit > new SuperHero()

Ensuite, sur la classe BadGuy, Clique Droit > new BadGuy()

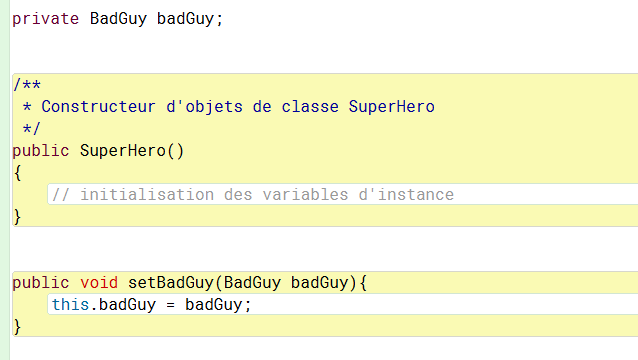


Comment faire pour dire à notre super-héros : Tiens, ça c’est ton méchant, il est là pour t’embêter ?

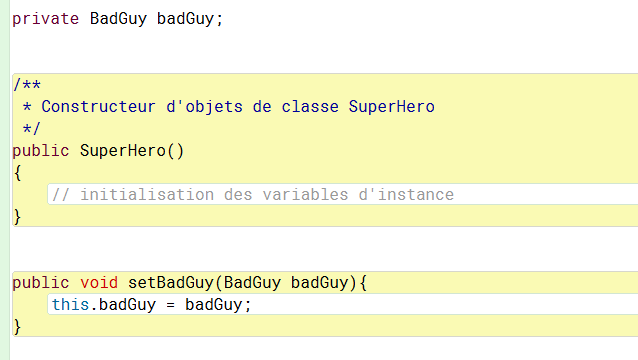
Nous sommes obligés de créer un setter dans notre classe SuperHero pour lui dire qui est son méchant. On supprime l’initialisation ci-dessous :



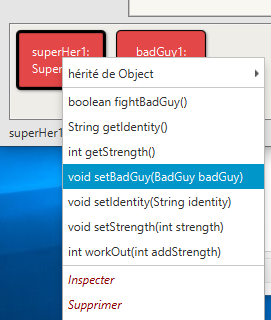
Par



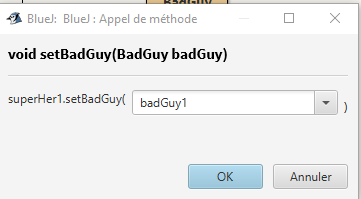
Et on ajouter le setter setBadGuy pour la classe SuperHero



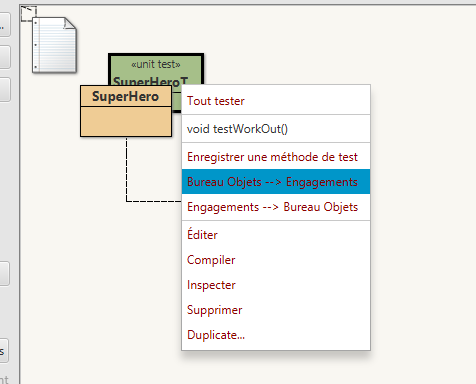
Maintenant, nous pouvons dire a notre SuperHero superHer1 que badGuy1 est son méchant via là méthode setBadGuy



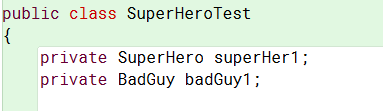
On précise en paramètre le nom de la variable du méchant (ici badGuy1)



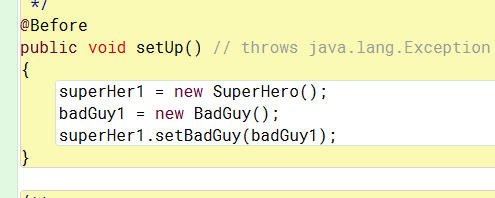
Et, pour insérer ces deux instances dans le setUp de notre fichier test, nous faisons Clique Droit > « Bureau Objets 🡪 Engagements »



Tadam, Test-Man a encore frappé. Allons voir le code de SuperHeroTest en double cliquant dessus .



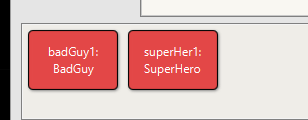
Deux champs sont apparus. Notre super-héros et son méchant. Et ci-dessous on a leur initialisation dans la methode setUp



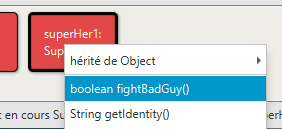
13

Enfin, la bataille finale !!! Pour tout combat, il y a des règles. Vérifions que les nôtres soient bien respectées. Nous allons tester la méthode fightBadGuy.

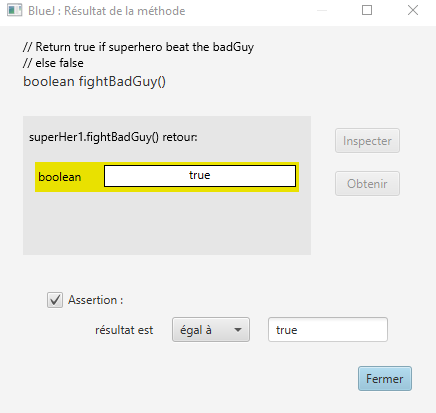
Pour cela, sur notre classe SuperHeroTest, Clique droit > « Enregistrer une methode de test ».



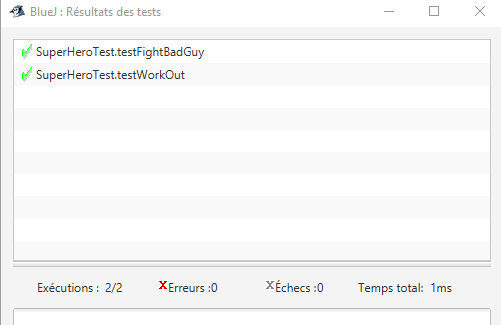
Comme par magie, nos deux objets sont réapparus. Testons la méthode fightBadGuy



Notre super-héros a une force de 35 et le méchant une force de 10 donc on s’attend à ce que le résultat soit true.



On fait ce que Compile-man sait faire le mieux : On Compile. Et On fait ce que Test-man fait le mieux : On Teste en exécutant les tests (tout simplement)

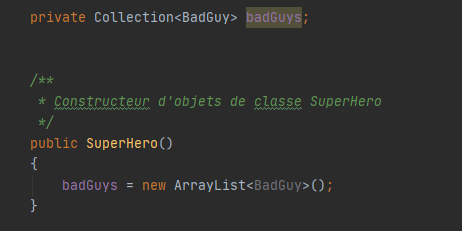


Bravo ! Grâce à toi la ville est enfin débarrassée du grand méchant ! Et notre super-héros peut continuer à veiller sur nos vies et nos espoirs.

## Implémentation d’un attribut de relation 0..1 - \* - La bi-directionnalité

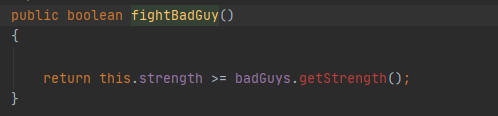
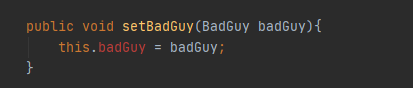
Okay, on a un superhéros contre un seul méchant ? Cela ne tient pas trop la route. Notre super-héros est très fort et il peut se permettre de dérouiller plusieurs méchants.

Nous allons donc transformer l’attribut de type BadGuy dans SuperHero en Collection<BadGuy>

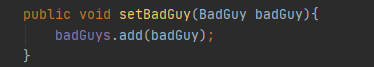


Sans oublier d’initialiser la collection dans le constructeur de SuperHero ( Sinon NullReference-Man se moquera de toi)

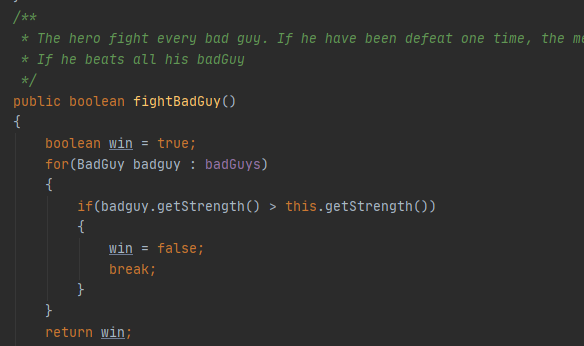
Voilà. Désormais on peut enfin combattre le crime comme il faut. Mais … KESAKO ? Il y a des erreurs de partout.



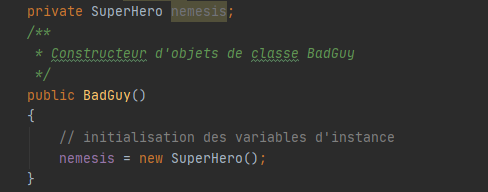
Il va falloir corriger tout cela pour que notre super-héros puisse se battre correctement. Nous devons modifier la méthode setBadGuy pour qu’elle ajoute le méchant.



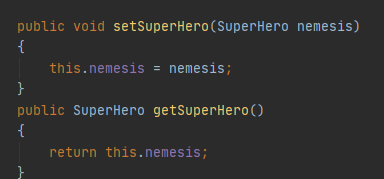
Et nous allons modifier notre méthode fightBadGuy pour que notre héro combatte toute sa liste de méchant



Ah ouf il n’y a plus d’erreurs. Mais nous sommes confrontés à un problème … tu ne vois pas de quoi je parle ? Aller tu me fais marcher ! Nous n’avons pas de relation bidirectionnelle. Le héro connait ses méchants mais le méchant ne connait pas le héro contre qui il se bat. Rectifions cela !



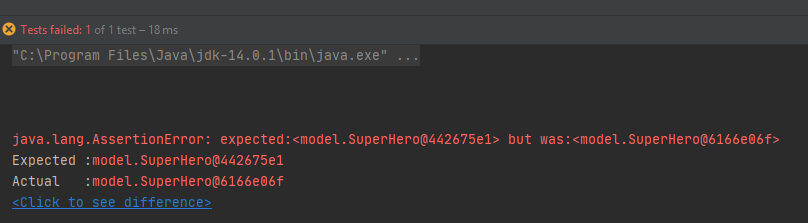
Il faut pouvoir affecter le bon super-héros au méchant. Créons les méthodes d’accès dans la classe badGuy.



Et maintenant on écrit le test.



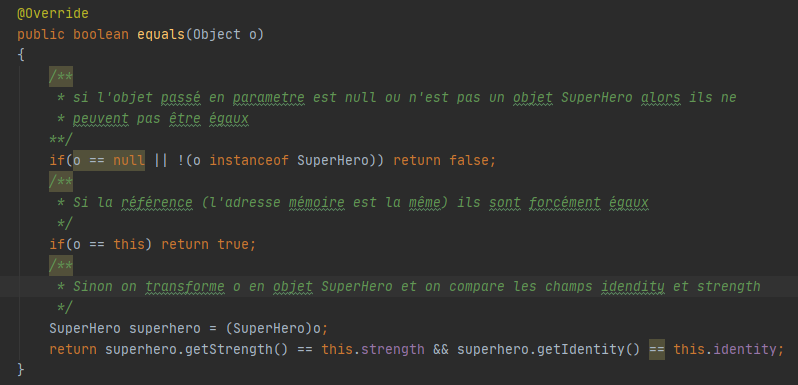
Et on lance les tests et …. Mais « Qué ça dire que ceci » ?



testGetSuperHero a échoué : pourtant quand on crée un nouveau super héros il s’appelle « Spider-man » et possède 35 de force alors pourquoi ?

C’est parce qu’un objet créé, même s’il possède les mêmes valeurs dans ses attributs qu’un autre objet ne possède pas forcément la même adresse en mémoire. Donc si deux objets possèdent les mêmes caractéristiques mais pas la même adresse mémoire, est ce que pour autant on doit les considérer comme différents ?

Nous allons donc utiliser et définir la méthode equals afin de rendre deux objets SuperHero, possédant les mêmes données, égaux.



On peut relancer notre test et …. C’est un succès !!!

Je vous laisse ajouter la méthode getSuperHero et son test 😉

## Refactoring

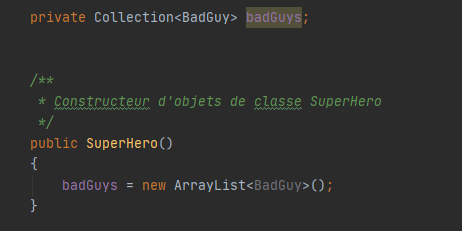
Le refactoring est utilisé lorsque nous modifions la structure d’un objet, du programme et qui va impliquer de devoir modifier la logique visuelle et de compréhension du code. Nous allons chercher à adapter le code que nous avons écrit aux modifications qui ont été faites.

### Rename Method

Cette méthode consiste à renommer les variables, attributs ou méthodes pour les rendre cohérentes avec leur fonction. Par exemple, avant d’implémenter la Collection de BadGuy nous avions ceci :



Et, subtilement, nous l’avons modifié de la manière suivante :

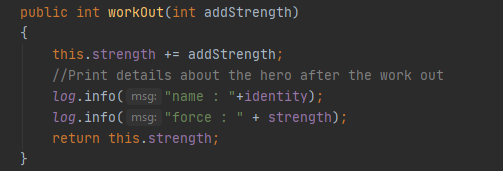


En effet badGuy au singulier est devenu badGuys. Cette exemple est subtile mais nous allons faire une modification qui n’a pas été encore faite : la méthode fightBadGuy. En effet, son nom implique que notre héro va se battre contre seulement un seul méchant. Or notre méthode le fait combattre tous les méchants. On va devoir la renommer :



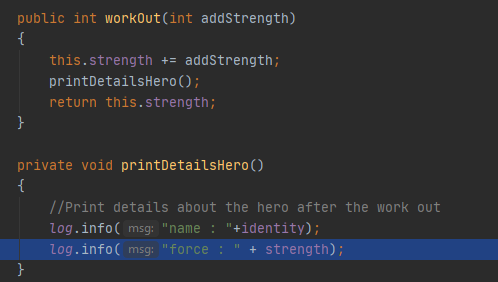
### Extract Method

La méthode d’extraction permet d’atomiser le code suivant la responsabilité. Imaginons que nous ajoutons se code dans la méthode workOut de SuperHero :



L’affichage des détails est effectuer grâce à un champ de type Logger nommé log dans la classe SuperHero.

L’affichage des détails n’est pas vraiment lié à la méthode workOut on va dont l’extraire et en faire une méthode :



Voilà nous avons séparer la responsabilité et cela permet de réutiliser l’affichage des détails d’un héros dans le journal de log.