# TP révisions – Héritage – Tests unitaires

# Objectifs

Application de quelques révisions autour des principes POO de Java.

# Outils

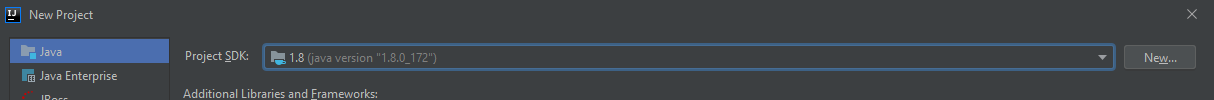
Au moins 3 possibilités contexte ENC :

1. Eclipse
2. IntelliJ (JetBrains) : disponible, licence idem que PHPStorm à votre nom sur le réseau
3. Android Studio : installation d’un contexte Java à l’intérieur d’Android Studio

Aujourd’hui j’utilise la 2ème possibilité, avec IntelliJ (il ne nous manquera plus que PyCharm pour le python … !).

Vous pouvez également le faire sous Eclipse si vous êtes habitués à Eclipse ! (cf annexe)

Après avoir installé IntelliJ, créez un nouveau projet, par exemple Pirates, de type Java.



# Installation

Copiez les fichiers suivant dans le répertoire src de Pirates: AppelPirates, Capitaine, Marin, Navire et l’interface Piraterie.

Créez une nouvelle classe Pirate.

**Exercice 1 :** bases et héritage

1. Votre classe Pirate doit ***étendre*** la classe Marin et ***implémenter*** l’interface Piraterie.

Elle a pour attribut un entier, le ***degré de sauvagerie***.

Définissez également une constante, le seuil de sauvagerie, qui vaut 50.

Créez le constructeur, getter et setter et la méthode toString (redéfinition).

1. Ajouter et retirer des Marins à l’équipage

Dans la classe Navire, ajoutez 2 méthodes qui permettent d’ajouter et de retirer un Marin à l’équipage du navire.

1. Surcharge méthode equals

On considère que 2 pirates sont identiques s’ils ont la même fonction et le même degré de sauvagerie, indépendamment de leur nom.

Implémentez (plus exactement redéfinissez) la méthode correspondante pour répondre à ce besoin.

* Quelle technique de POO utilise le générateur de code par défaut ?
* Commenter le code généré

1. Implémenter la méthode pillage

Concernant la méthode pillage :

* si le degré de sauvagerie du pirate est supérieur au seuil, tout l’équipage est détruit.
* sinon, si le degré de sauvagerie est inférieur au seuil, seul le Capitaine est tué

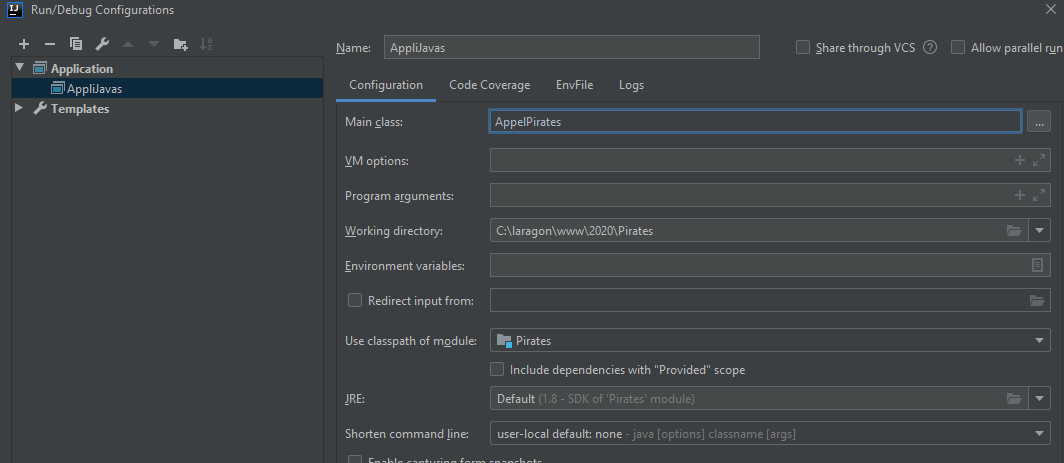
Attention : pour cette dernière sous partie, vous devrez utiliser un Itérateur car votre collection ne pourra pas être modifiée en même temps que vous la parcourez via une boucle.

Vérifiez que cela fonctionne en exécutant la classe AppelPirates.java.

Pour ce faire : clic sur Add Configuration (à droite du marteau vert) puis.

Cliquez sur + puis choisir Application/

Remplir comme ci-dessous :



Exécutez le programme.

Maintenant ajoutez un Pirate nommé gargamel et avec un degré de sauvagerie de 35.

Réaffichez le navire et vérifiez que le Capitaine a disparu de l’équipage.

**Exercice 2 :** validation via tests unitaires

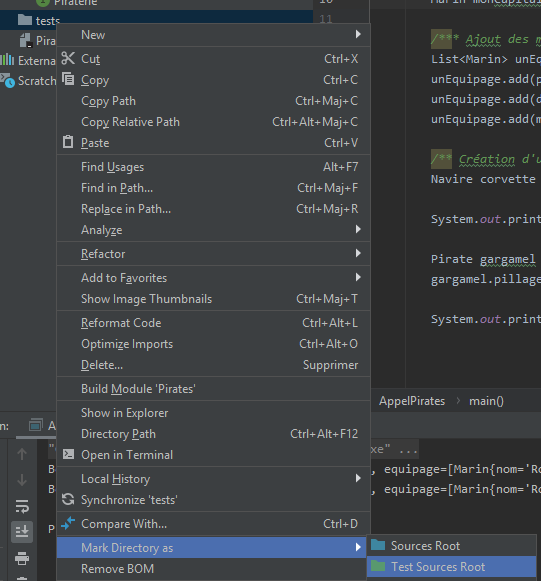
L’objectif est de valider nos méthodes sans passez par des impressions dans la log (System.out …). On veut également tester facilement les différents cas (selon degré de sauvagerie …).

1. Installation d’un répertoire de test

Tout d’abord créez un nouveau répertoire nommé tests :

Sur le projet Pirates, clic droit et New 🡪 Directory. Appelez le tests.

Ensuite marquez ce nouveau répertoire comme Test Sources Root : clic droit sur **tests** et Mark directory as Test sources Root (cf ci-dessous).

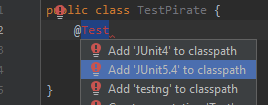


1. Installation de Junit 5. Ecriture de tests unitaires

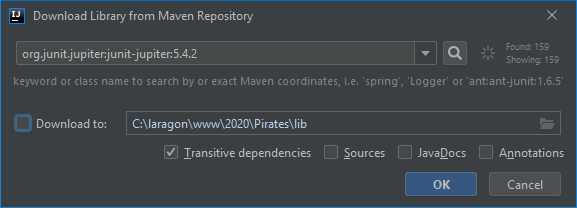
Créez dans le folder tests une nouvelle classe appelée TestPirate.

Dans la classe, créez une annotation @Test

Elle va apparaître en rouge. Cliquez Alt – Entrée dessus et choisissez Add Junit5.4 …



Puis



Ok

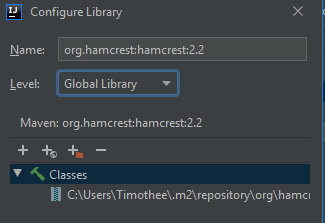
1. Installation de Hamcrest

Hamcrest est une librairie complémentaire de tests unitaires pour Java, autrefois intégrée à Junit4, dorénavant séparée.

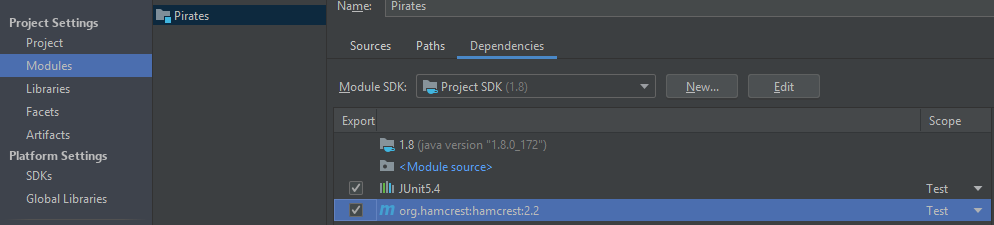
Nous devons l’importer dans notre projet via le menu modules.

Files 🡪 Project Structure 🡪 Modules 🡪 + 🡪 2. Library 🡪 From Maven

Recherchez org.hamcrest:hamcrest:2.2



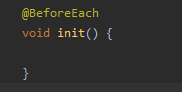
Une fois revenu sur les Modules, vérifiez bien que hamcrest est coché avec le scope Test (Compile Ok aussi).



1. Ajoutez une méthode testEgalitePirates

En dessous de l’annotation @Test, ajoutez une méthode de test testEgalitePirates, sans paramètre et qui ne retourne rien

1. Ajoutez une méthode d’initialisation de données utile



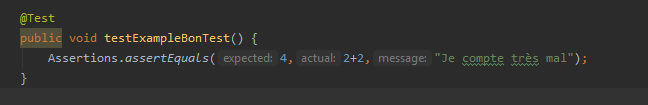
Ajoutez comme attributs de la classe de test 2 Marins, 1 capitaine,1 équipage (liste de marins), 1 pirate et 1 navire.

Dans le init(), instanciez les Marins, equipage et Navire en vous inspirant de la classe AppelPirates

Ecriture de tests Junit : Syntaxe

Chaque méthode de test doit renvoyer void et avoir un nom qui commence par ***test***. Elle doit être précédée d’une annotation JUNIT5 @Test.

**Exemple** :



1. Implémentez la méthode de test d’égalité de pirate

Par exemple : création d’un 2ème pirate, avec même fonction et degré de sauvagerie que le 1er mais un nom différent 🡪 le test d’égalité doit être Ok

En revanche, si le degré de sauvagerie ou la fonction est différente, le test doit être faux.

Vous vous reporterez à la documentation JUNIT5 : <https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html>

Une assertion d’égalité sera de la forme : Assertions.assertEquals(expected, actual, message)

Une assertion de non égalité : Assertions.assertNotEquals(expected, actual, message)

1. Implémentez la méthode de test d’ajout de Marin à l’équipage

On peut créer un Marin, l’ajouter à l’équipage de notre navire, puis utiliser la librairie hamcrest de test de Collections. Exemples : <https://www.baeldung.com/hamcrest-collections-arrays>

Par exemple :

assertThat(message,uneCollection,hasItem(unElement)) permet de tester si la collection uneCollection contient l’élément unElement

On remarquera la syntaxe différente de celle de JUNIT5, et la position du message en cas d’erreur qui se situe au début plutôt qu’à la fin.

1. Implémentez la méthode de test de retrait de Marin à l’équipage

Par exemple :

assertThat(message,uneCollection,not(hasItem(unElement))) permet de tester si la collection uneCollection ***ne*** contient ***pas*** l’élément unElement

1. Ajoutez une méthode testPillageTresSauvage

Ajoutez une méthode de test testPillageTresSauvage, sans paramètre et qui ne retourne rien.

Validez la méthode pillage lorsque un Pirate avec un degré de sauvagerie supérieur au seuil pille

Par exemple :

assertThat(message,uneCollection,not(hasItem(unElement))) permet de tester si la collection uneCollection ***ne*** contient ***pas*** l’élément unElement

1. Ajoutez une méthode testPillagePeuSauvage

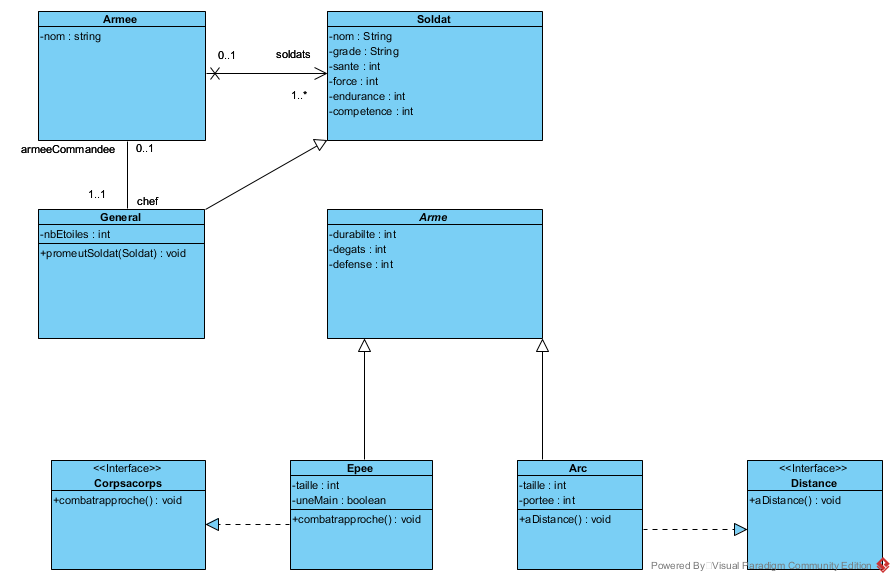
Validez la méthode pillage lorsque un Pirate peu sauvage pille

# Soldat – Armée – Armada

Votre Product Owner souhaite étoffer le projet. Les utilisateurs veulent également des soldats, des armes, une armée, pour pouvoir diversifier les environnements.

**Exercice 3** : dérivation diagramme classe UML conceptuel vers Java

a. L’architecte génie logiciel a effectué le diagramme suivant qu’on vous demande d’implémenter dans le même projet.



Soyez attentifs aux **roles** qui vont ici donner des attributs si l’association est navigable.

L’entité Arme est en *italique* : cherchez à quoi cela correspond pour un diagramme de classe UML.

La flèche *pointillée* indique qu’une classe ***réalise*** (implémente) une ***interface***.

Vous ajouterez également tous les getters et setters, constructeurs et méthode d’affichage.

b. Liaison des associations 1 - 1

Lorsque nous avons étudié la dérivation des diagrammes de classe UML vers Java, nous avons laissé de côté ce problème.

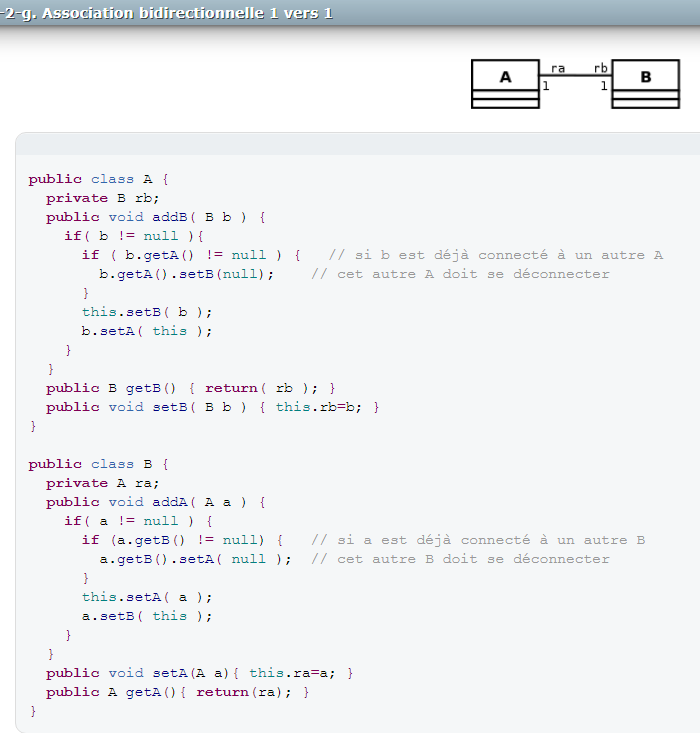
Prenons un exemple :

1 General a un attribut Armee parmi ses attributs, puisque son association est navigable de General vers Armee avec une multiplicité 1

1 Armee a un attribut General nommé chef parmi ses attributs, puisque l’association est navigable de Armee vers General avec une multiplicité 1

Il faut maintenant s’assurer que la liaison est **bijective** ! c’est-à-dire que les instances sont bien reliées dans les 2 sens !

Cf pour exemple (source Audibert) :

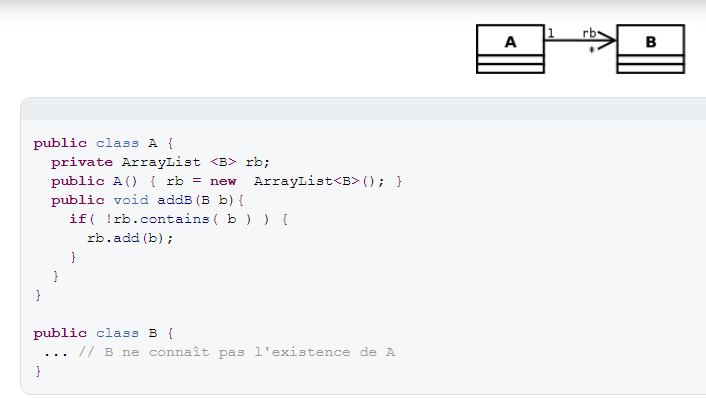


A faire : ajouter une méthode avec la signature suivante ****dans la classe **Armee** en adaptant l’exemple ci-dessus.

Idem pour la classe **General** : ajouter une méthode avec la signature suivante 

1. Liaison des associations 1 - \*

Lorsque nous avons étudié la dérivation des diagrammes de classe UML vers Java, nous avons laissé de côté ce problème.



Ajoutez ce qu’il faut dans la classe General en vous inspirant de l’exemple ci-dessus d’Audibert.

1. Tests unitaires

Créez une nouvelle classe de tests unitaires. Déclarez 3 soldats, 1 Armee et un General. Dans le setup (annoté par un @BeforeEach) initialisez les variables.

Créez 3 tests unitaires, un pour l’ajout d’un chef à l’armée, un pour l’ajout d’une armée à un chef, enfin un autre pour l’ajout de soldats à l’armée.

Dans chacune de ces méthodes de tests , effectuez plusieurs assertions avec junit5 et hamcrest pour vérifier l’initialisation et le bon fonctionnement des méthodes.

1. L’Armada !

Avec tout ce beau monde, il ne vous reste plus qu’à créer une classe Conquete.

Cette classe possède un attribut, nommé par exemple armada, définit comme une Map de <Navire,Armee> , c’est-à-dire qu’on fait monter une Armee dans un navire.

Ajoutez une méthode d’affichage à cette classe qui affiche tous les éléments de l’armada : pour chacun de ces éléments, on veut voir le nom du navire, les noms des membres de l’équipage et le contenu de l’armée embarquée correspondante, en s’appuyant évidemment sur les méthodes d’affichage de toutes les classes manipulées.

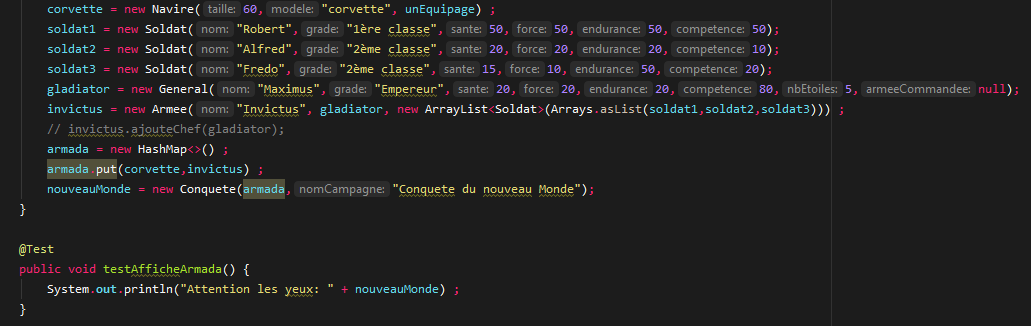
**Hints** :

On a vu dans la fiche sur les Collections qu’on pouvait afficher le contenu d’un tableau avec la méthode statique toString de la classe Arrays : Arrays.toString(Tableau[]) où Tableau est un tableau d’objets.

De la même façon on peut imprimer une liste (List) d’objets avec la syntaxe : Arrays.toString(*List<Classe>*.toArray()) où List<Classe> est notre collection d’objets.

Par exemple :  où soldats est une List<Soldat>

Pour l’affichage, on peut le faire simplement dans une classe de test dont voici un extrait de code ci-dessous ;



**Remarque** : ici on fait exprès d’instancier le General gladiator avec une Armee à null , en revanche d’instancier l’Armee invictus avec notre General gladiator pour « casser » la relation 1—1 qu’on a rendue bijective avec notre méthode ajouteChef à l’étape b.

En effet, l’affichage toString va faire une boucle infinie si la relation est bijective … (General imprime son instance Armee associée qui imprime son instance General associée etc etc jusqu’à plantage mémoire).

# Introduction aux Design Pattern

**Exercice 4 :** AVANCE : fonction de rappel

Basé sur la reprise des concepts du « DOC4-Fiche Java - CallBack & Async - AVANCE.docx »

Votre chef de projet s’emballe. Il vous demande d’implémenter une classe Mutin qui hérite de Pirate et implémente une interface OnEquipageEventeListener qui définit une méthode siAjouteEquipage (qui reçoit en paramètre le Navire observé pour lequel on ajoute l’équipage …).

L’idée est que un Mutin est en quelque sorte un espion, qui a un certain degré de compétence, qui va observer ce qui se passe sur le Navire, notamment quand un Marin est ajouté.

Si au moment de l’ajout d’un Marin à l’équipage d’un Navire, le degré de compétence du Mutin espion est suffisant (par exemple supérieur à la taille du Navire divisé par le nombre de Marins dans l’équipage), alors il s’ajoute à l’équipage.

On n’a pas besoin de la version asynchrone pour se faire.

Vous vérifierez que le test unitaire de la méthode d’ajout d’un Marin n’est pas cassé.

Et vous ajouterez un nouveau test unitaire validant que le Mutin s’ajoute, ou pas, à l’équipage, selon son degré de compétence.

# Annexes

# Installation du projet avec Eclipse

File 🡪 New 🡪 Java Project

Appelez le Pirates

Dans le répertoire src, placez les fichiers :

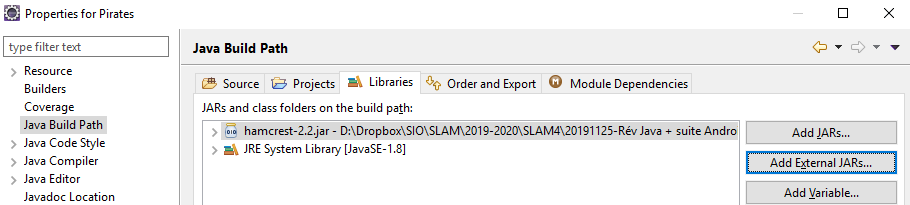
AppelPirates, Capitaine, Marin, Navire et l’interface Piraterie.

Créez un nouveau Source Folder appelé tests : File 🡪 New 🡪 Source folder

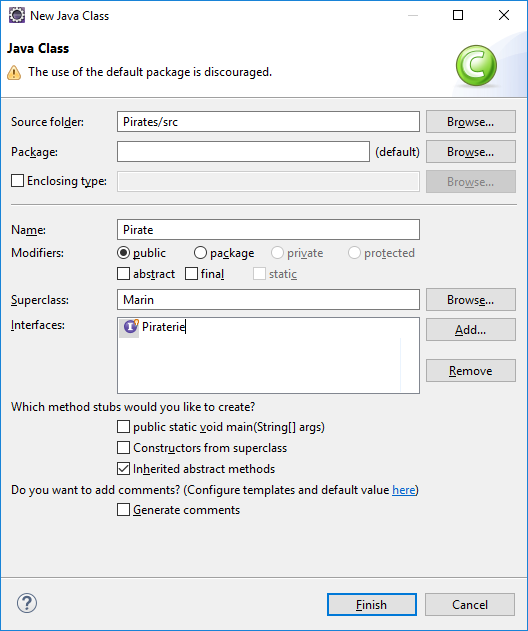
Importez la librairie hamcrest.

Clic droit sur Pirates 🡪 Properties 🡪 Java Build Path 🡪 Libraries 🡪 Add External Jars

Ajoutez hamcrest.jar

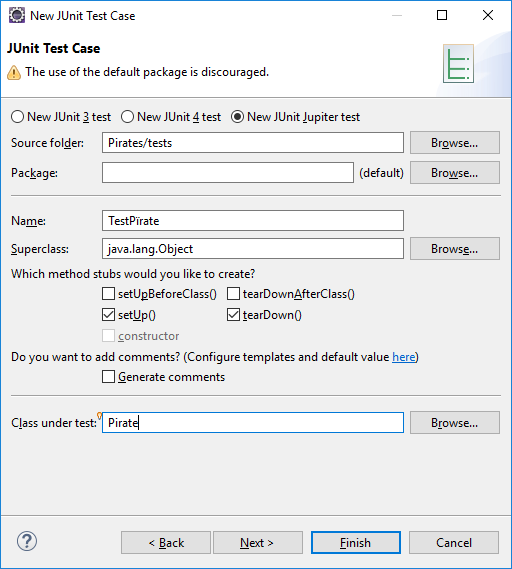


Dans src, créez une nouvelle classe Pirate.



Dans tests, créez un nouveau test unitaire TestPirate

File 🡪 New 🡪 Other 🡪 Junit Test case



Acceptez la proposition d’ajouter Junit5 library to the build path.