Aix*Marseille université
Socialement engagée

Année 2021-22

Domaine Sciences et Technologies
Portail René Descartes

Programmation 1 : TP 8
Code UE : SPO2U07L

TP 8 : Bataille navale

1 Description du jeu de bataille navale

À la bataille navale (*battleship game*), le plateau de jeu est représenté par une grille rectangulaire de cases sur lesquelles on peut poser des bateaux. Les bateaux sont larges d'une case et longs d'un nombre variable de cases. Ils peuvent être posés verticalement ou horizontalement sur le plateau.

Le plateau est masqué au joueur qui attaque et celui-ci doit couler tous les bateaux du joueur défenseur (*defender*). Pour cela, il propose une position du plateau qui désigne la case sur laquelle il tire.

Plusieurs cas sont alors possibles :

— si cette case n'est pas occupée par un bateau, le défenseur annonce dans l'eau (*missed*) ;
— dans le cas contraire, le défenseur annonce touché (*hit*) si toutes les cases occupées par le bateau touché n'ont pas déjà été visées par l'attaquant,
— le défenseur annonce coulé (*sunk*) si toutes les autres cases du bateau ont déjà été touchées.
— lorsqu'une case avait déjà été visée précédemment, la réponse est toujours dans l'eau.

L'objectif de ce TP sera de programmer une version très simplifiée du jeu de bataille navale.

2 Création de projet

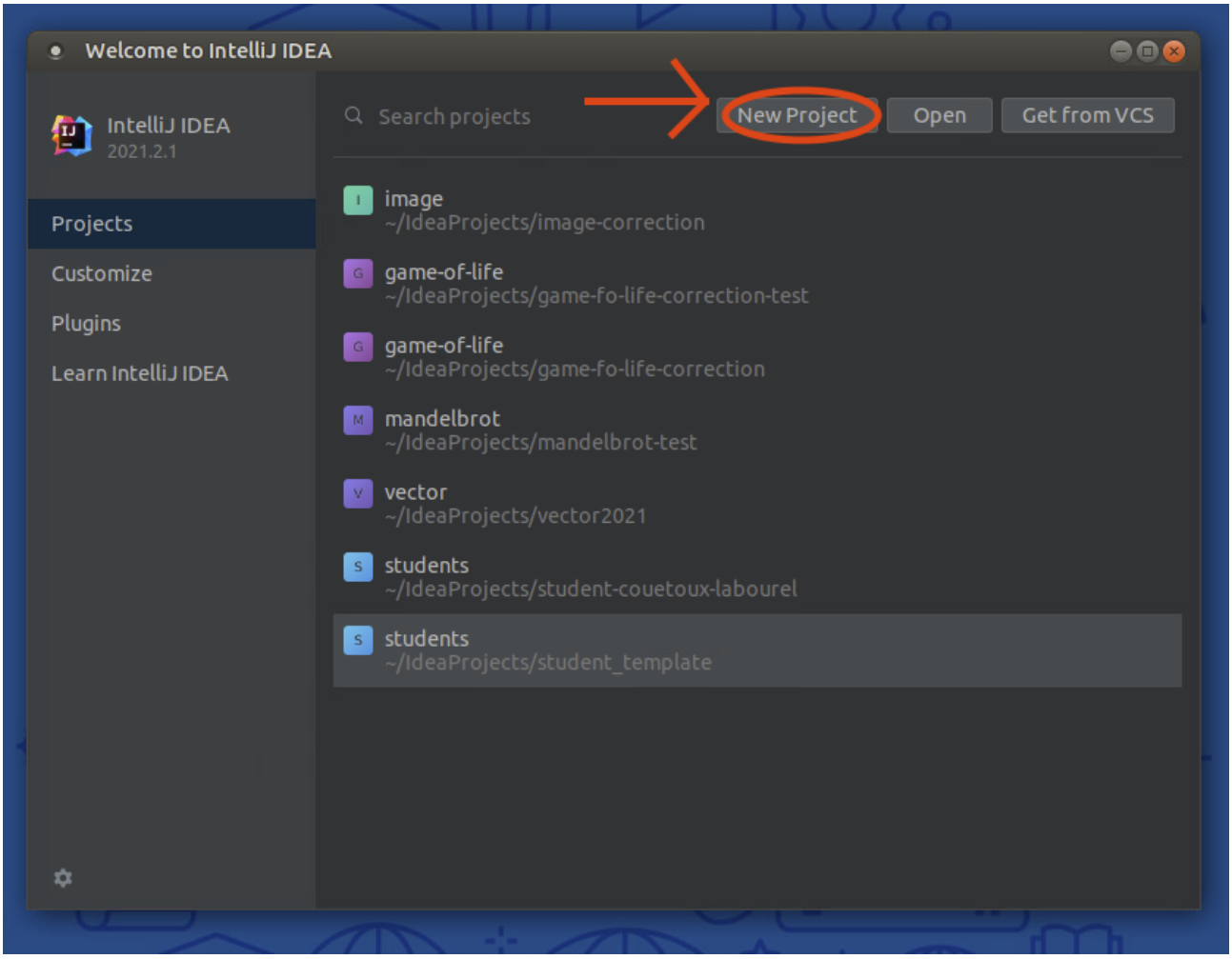
Le but de la première partie de ce TP est d'apprendre à configurer un projet en utilisant IntelliJ IDEA ou bien entièrement en ligne de commande.

Si vous effectuer ce TP depuis votre machine personnelle, il vous faut installer les outils suivants :

— **Terminal shell** si vous êtes sous windows, nous vous conseillons d'installer ubuntu pour avoir accès à un terminal shell. Sous MacOS et linux, un terminal shell est disponible de base.
— **JDK version 17 ou plus** la méthode d'installation dépend de votre système d'exploitation :
  — **Windows** : Télécharger et exécuter l'Installeur windows
  — **MacOS** : Télécharger et exécuter l'Installeur MacOS
  — **Linux** : exécuter la commande suivante dans le terminal `sudo apt install openjdk-17-jdk`. Un mot de passe administrateur vous sera demandé.
— **Gradle** : Suivez les instructions au lien suivant : install gradle
— **git** : Télécharger et installer git au lien suivant : download git
— **IntelliJ IDEA** : vous pouvez télécharger IntelliJ IDEA ou bien JetBrains Toolbox qui est un outils pour gérer les IDE proposé par l'entreprise JetBrains. En tant qu'étudiant, vous avez d'ailleurs accès à des licences gratuites pour leurs produits et notamment la version ultimate d'IntelliJ IDEA. Vous pouvez aussi utiliser Eclipse ou Visual studio code qui sont des outils similaires.

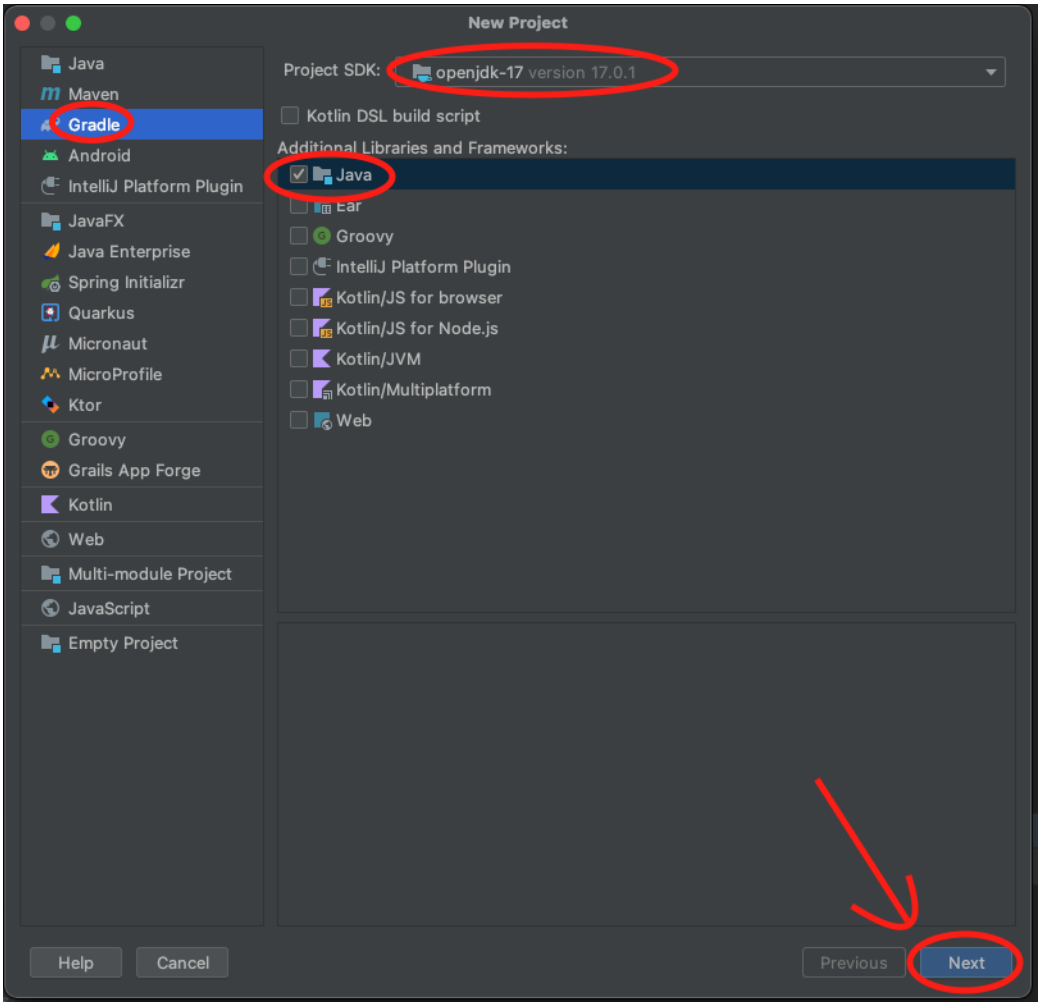
2.1 Création de projet versionné à l'aide d'IntelliJ

— Lancez IntelliJ IDEA
— Commencez la création d'un nouveau projet soit en fermant tout vos projets en cours via le menu `file -> Close project` et en cliquant sur `New Project`.

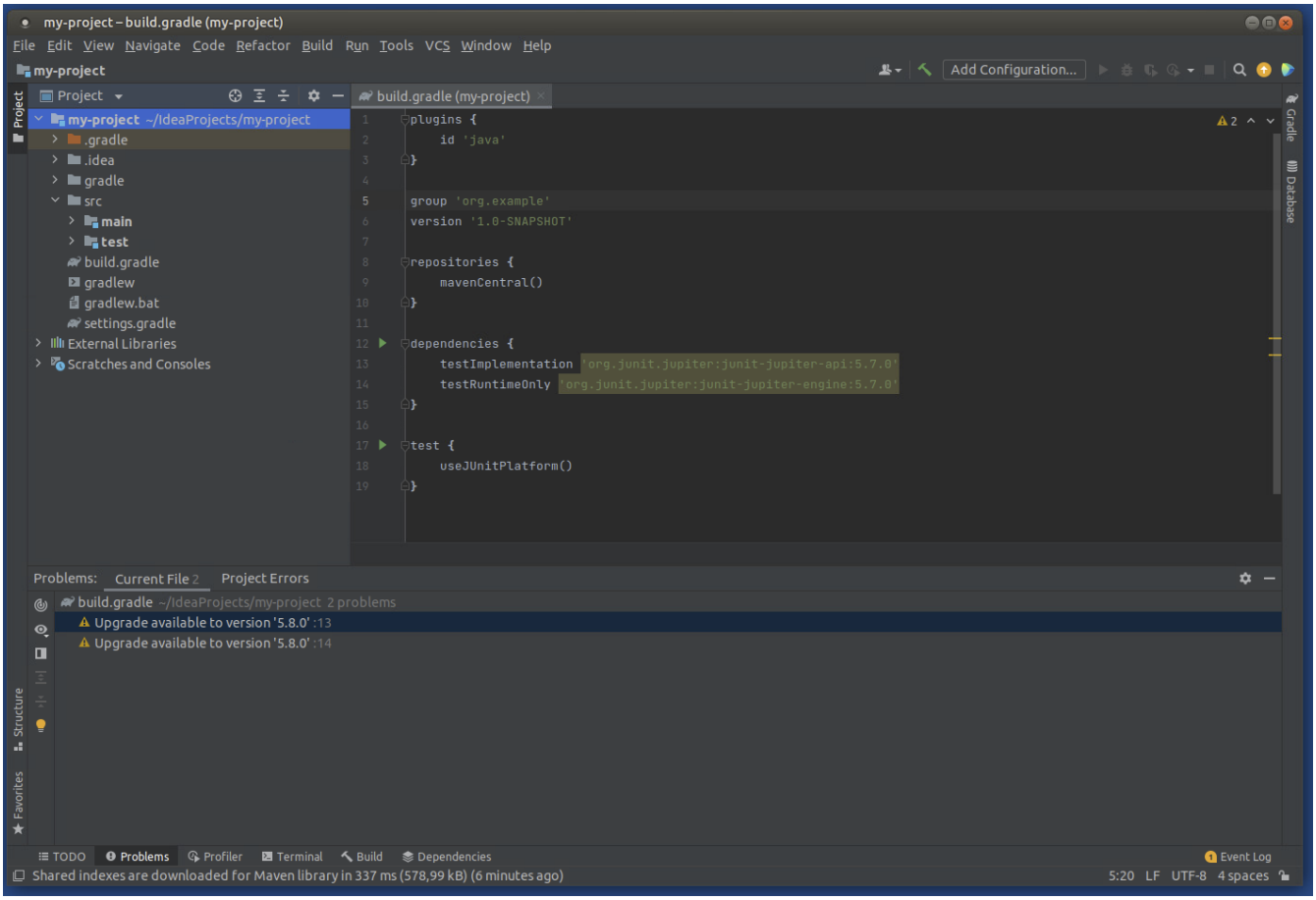


Vous pouvez aussi créer un nouveau projet en allant dans le menu `file` -> `New` -> `Project`.

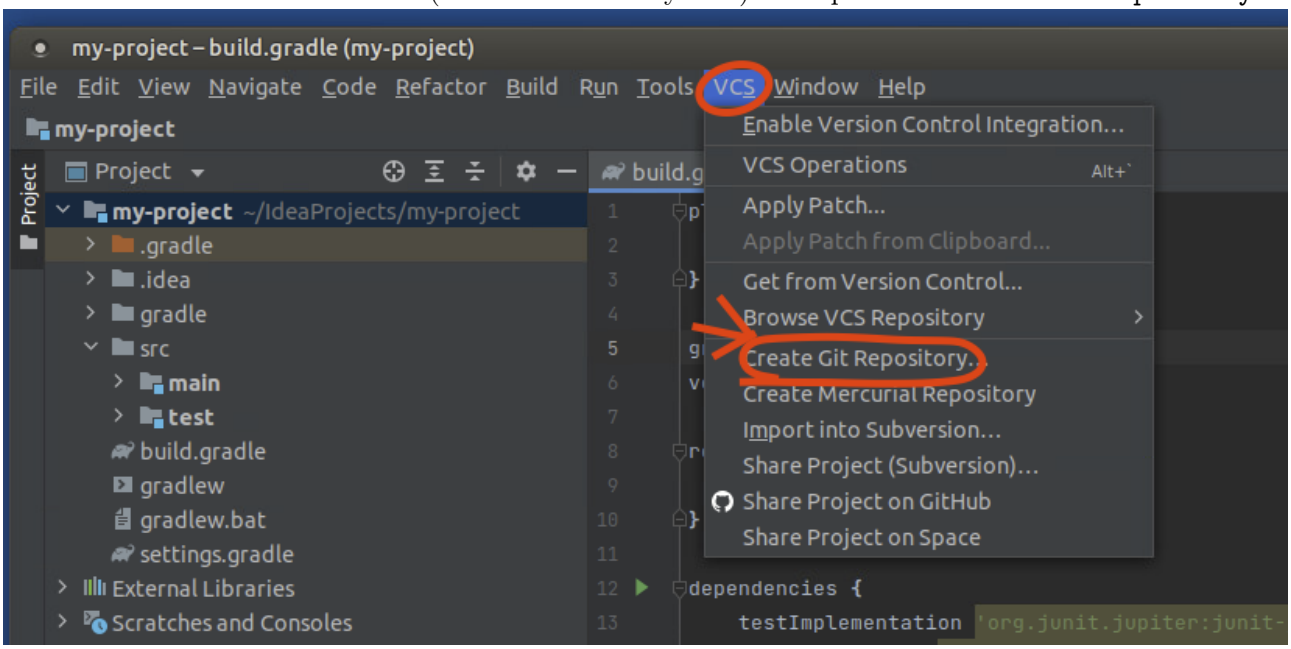
— Créer un projet gradle en choisissant `gradle` dans la liste à gauche. Gradle est ce que l'on appelle un moteur de production. C'est un outil pour compiler et exécuter les projets. Il est particulièrement utile pour gérer les dépendance de bibliothèques. En fait, cet outil télécharge automatiquement les bibliothèques configurées dans le projet dans le fichier `build.gradle`. Cochez `java` (si ce n'est pas déjà fait) en tant qu'`Additionnal Libraries and Frameworks` et vérifier que le `Project SDK` a un numéro de version supérieure ou égal à 17. Cliquer sur le bouton `Next`.



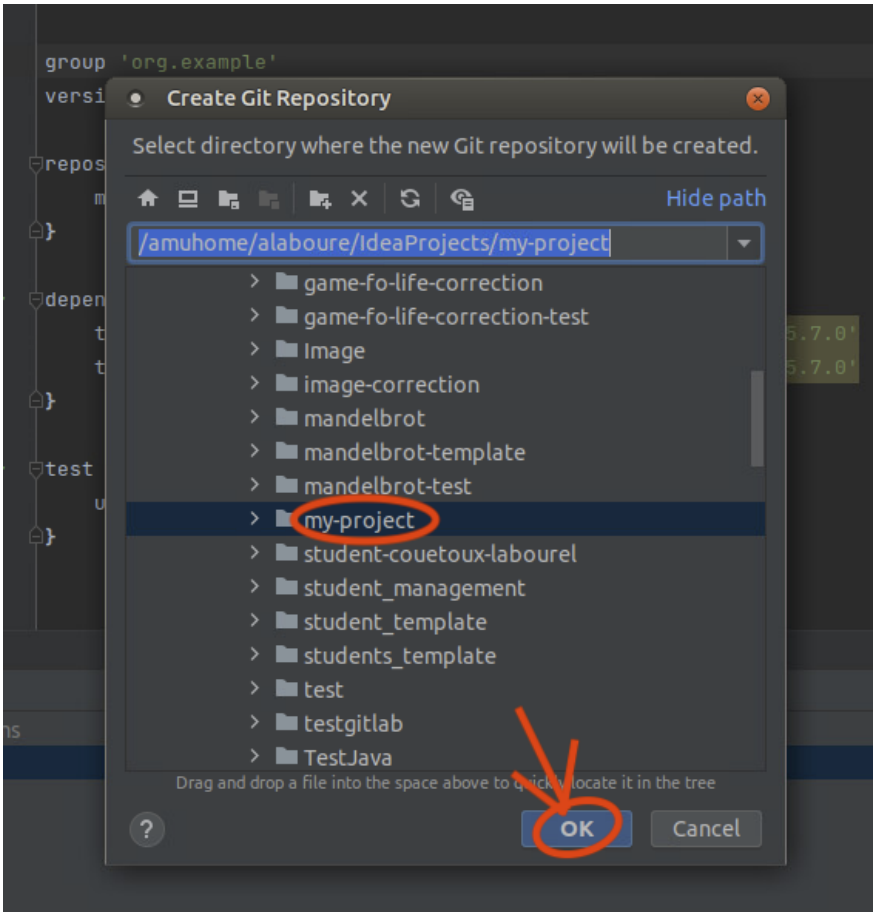
— Entrer le nom de votre projet (à priori `battleship` pour ce qui nous concerne) et validez sa création en cliquant sur sur le bouton `finish`.
— Votre projet devrait se configurer automatiquement et vous devriez obtenir l'affichage suivant.



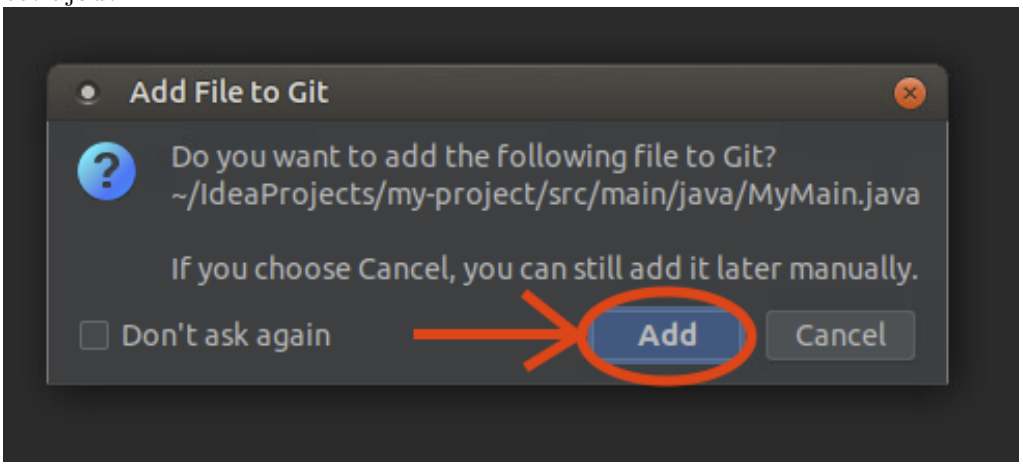
— vous allez maintenant créer un dépôt local `git` qui va permettre de faire de la gestion de version en local. Pour cela allez dans le menu `VCS` (Version Control System) et cliquez sur `Create Git Repository`.



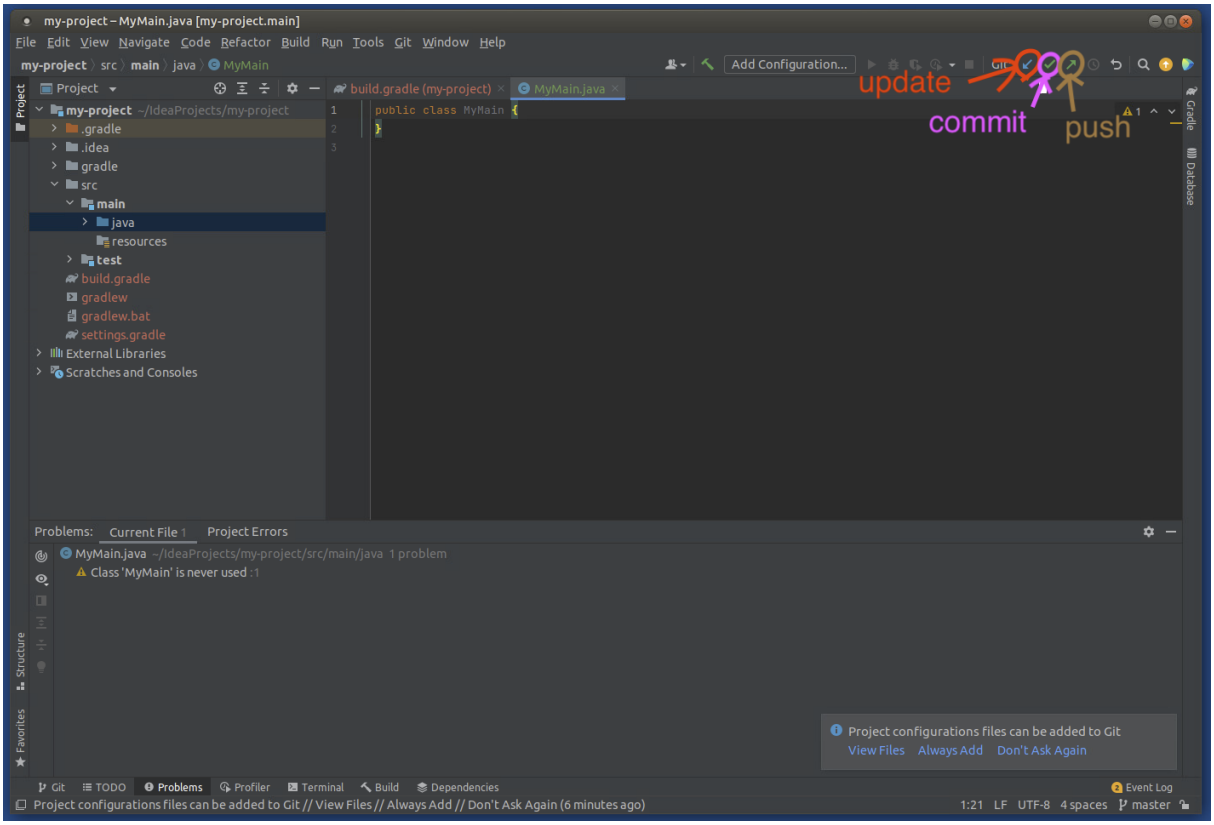
— Choisissez le répertoire pour votre dépôt (normalement le choix de base est le répertoire contenant votre projet) et cliquez sur le bouton `ok` pour valider.



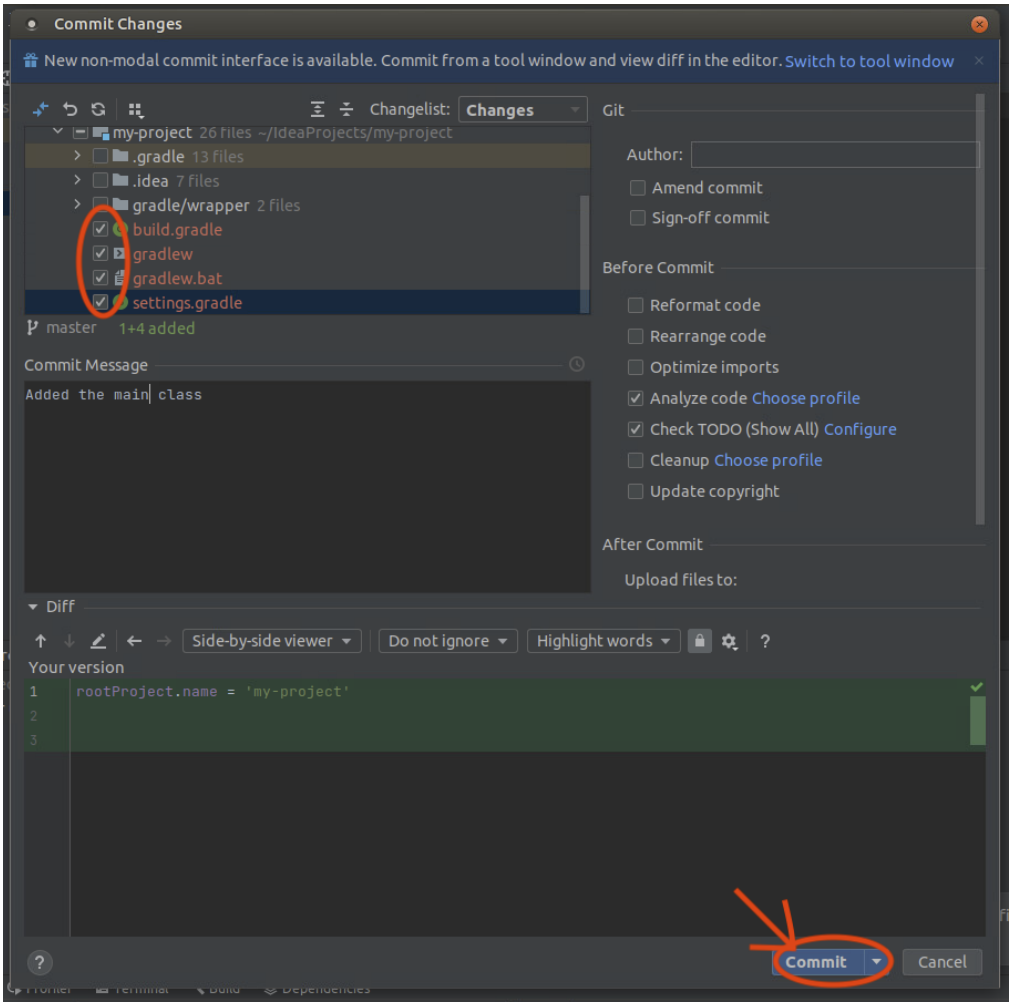
— Créez une classe dans le répertoire `src` -> `main` -> `java`. Une fenêtre va s'ouvrir pour demander si vous voulez rajouter le fichier contenant votre classe dans le dépôt `git`. Cliquez sur le bouton `add` pour validez cet ajout.



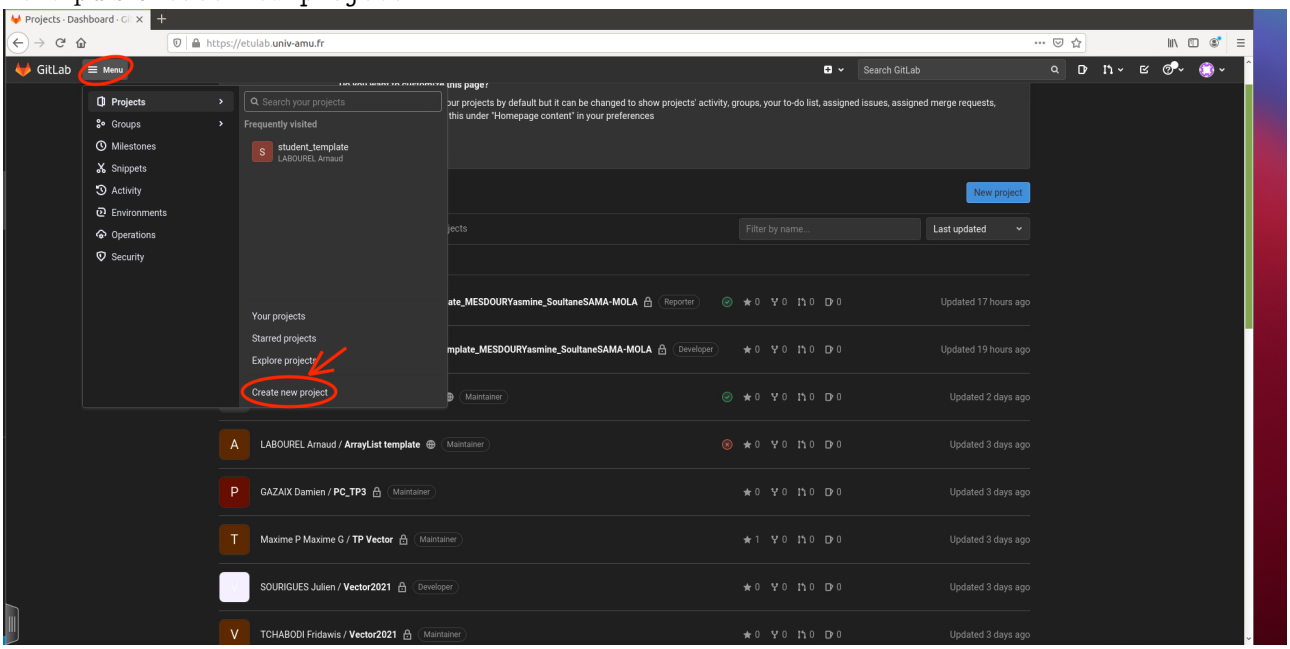
— Vous pouvez effectuer les opérations de base de git (`update` du projet, `commit` et `push`) grace aux raccourcis en haut à droite de la fenêtre.



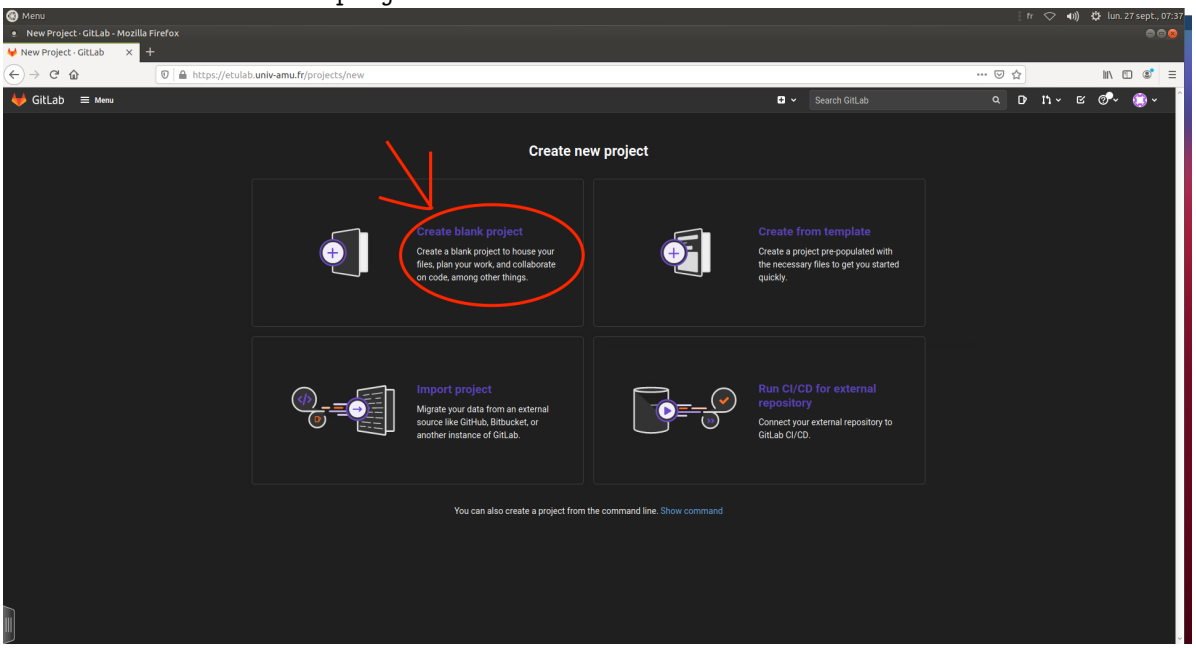
— Faites le premier commit de votre projet en ajoutant en plus de vos fichiers `.java` les fichiers de configuration de gradle : `build.gradle`, `gradlew`, `gradlew.bat` et `settings.gradle`.



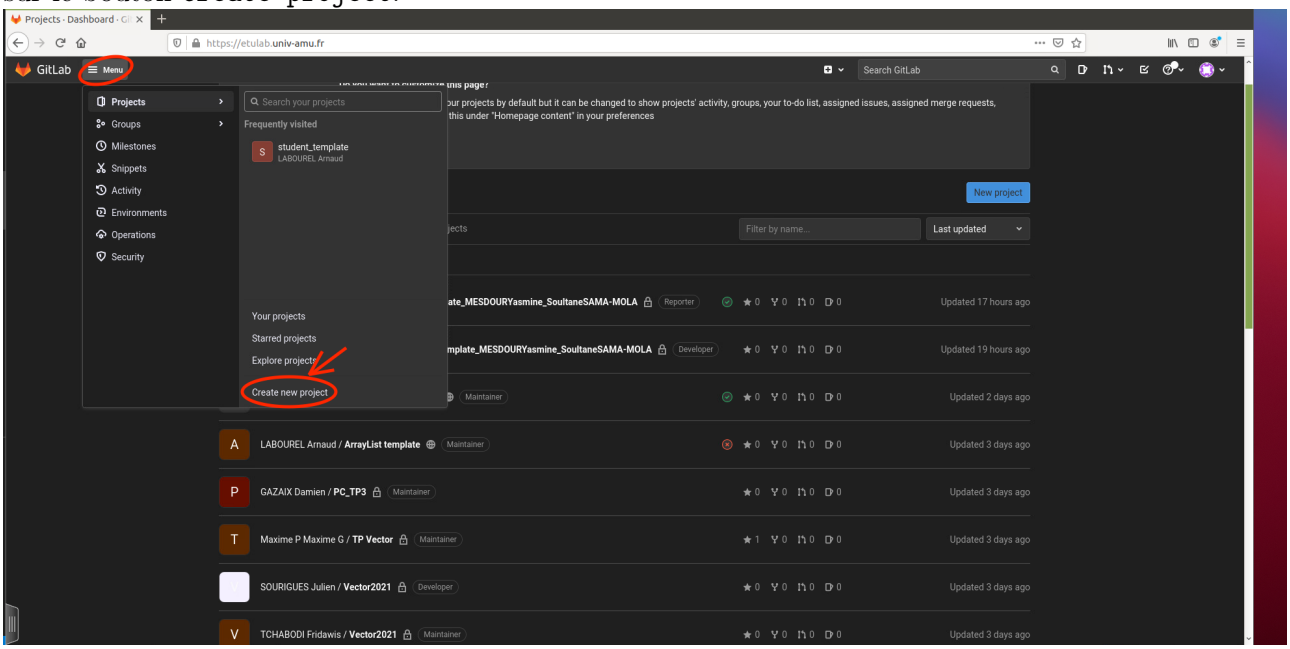
— Connectez vous via à un navigateur à votre compte etulab et créez un nouveau projet en cliquant sur `menu` puis `Create new project`.



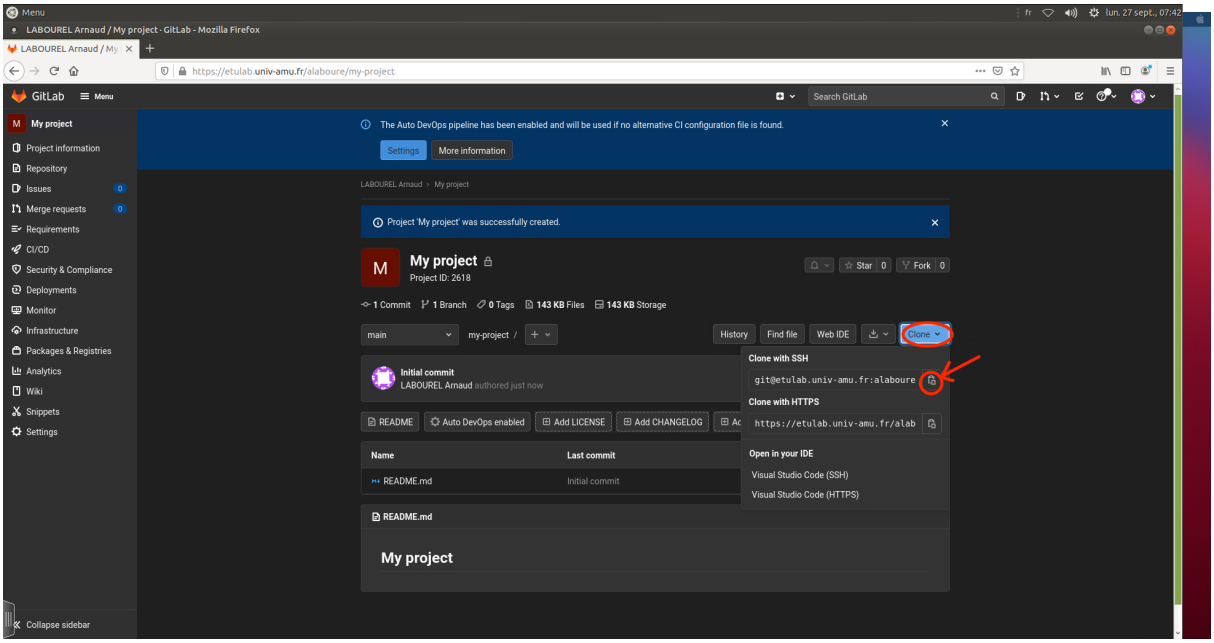
— Choisissez `Create blank project`.



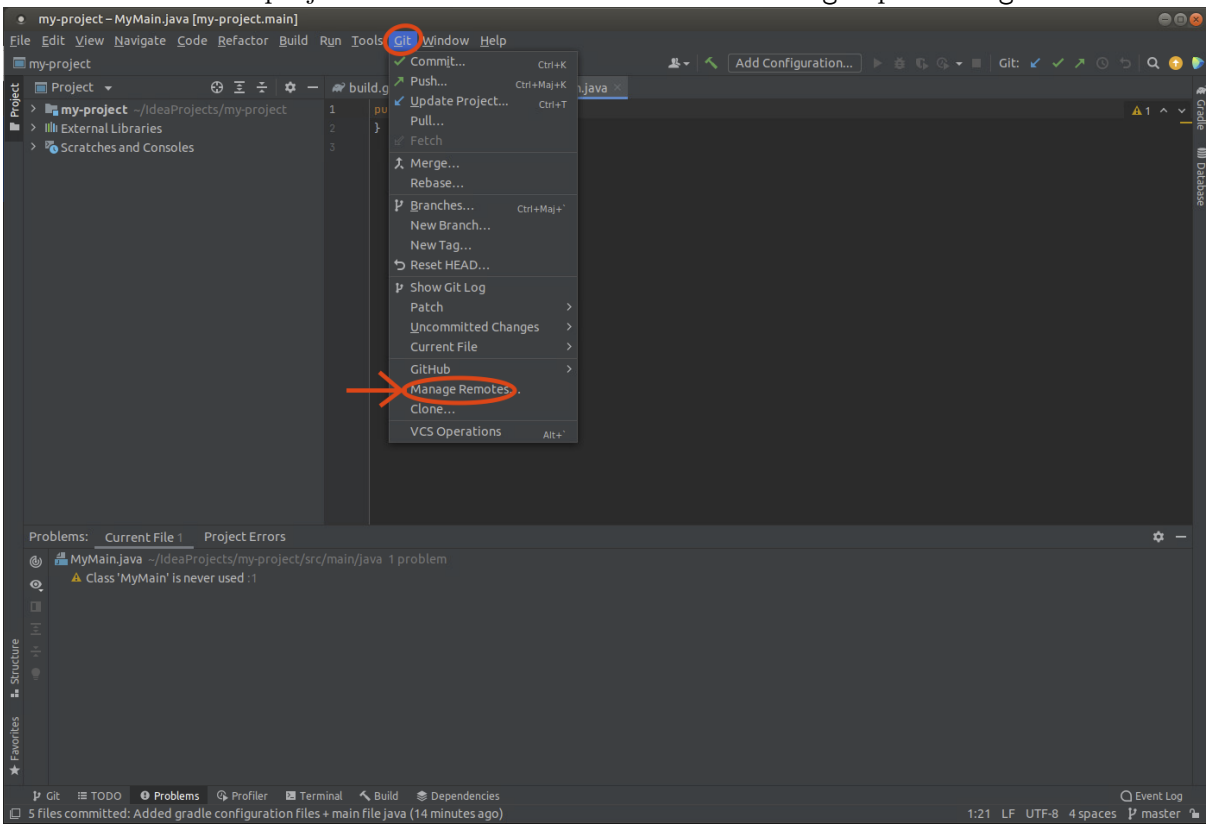
— Choisissez un nom pour votre projet, décochez la création du `README.md` et validez la création en cliquant sur le bouton `Create project`.



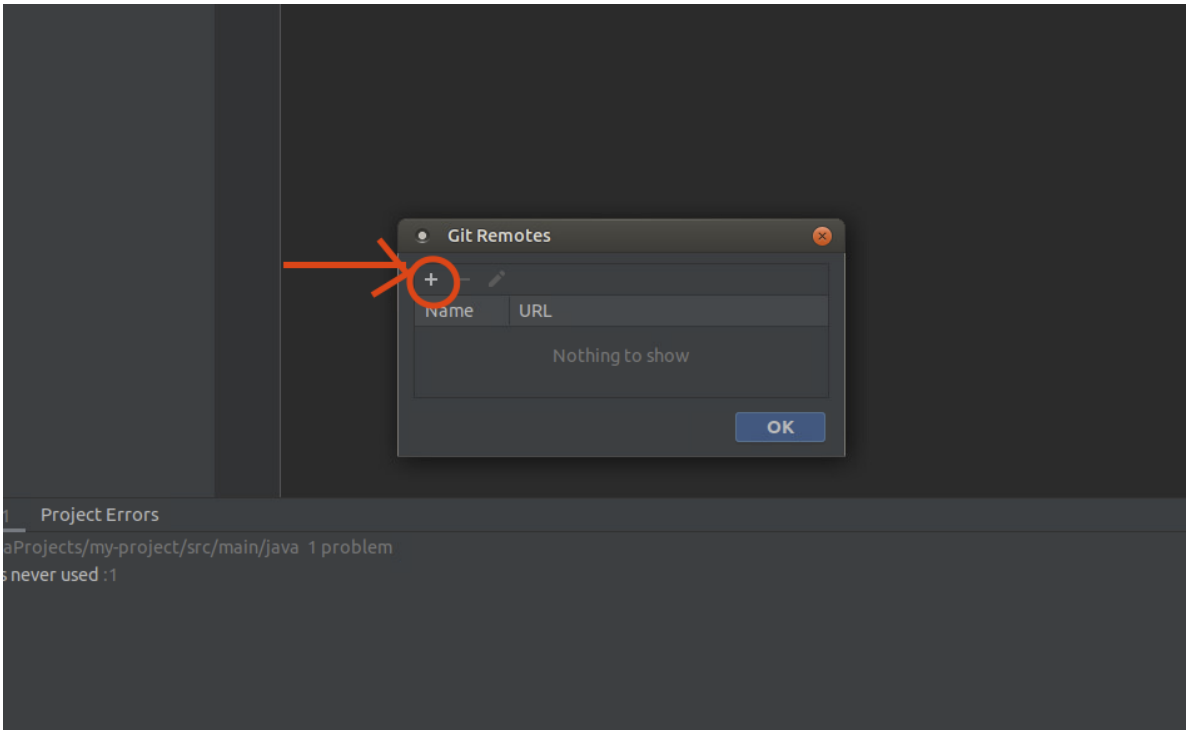
— Copiez l'adresse pour cloner votre projet en cliquant sur le bouton `clone` puis sur l'icône de copie `ssh`.



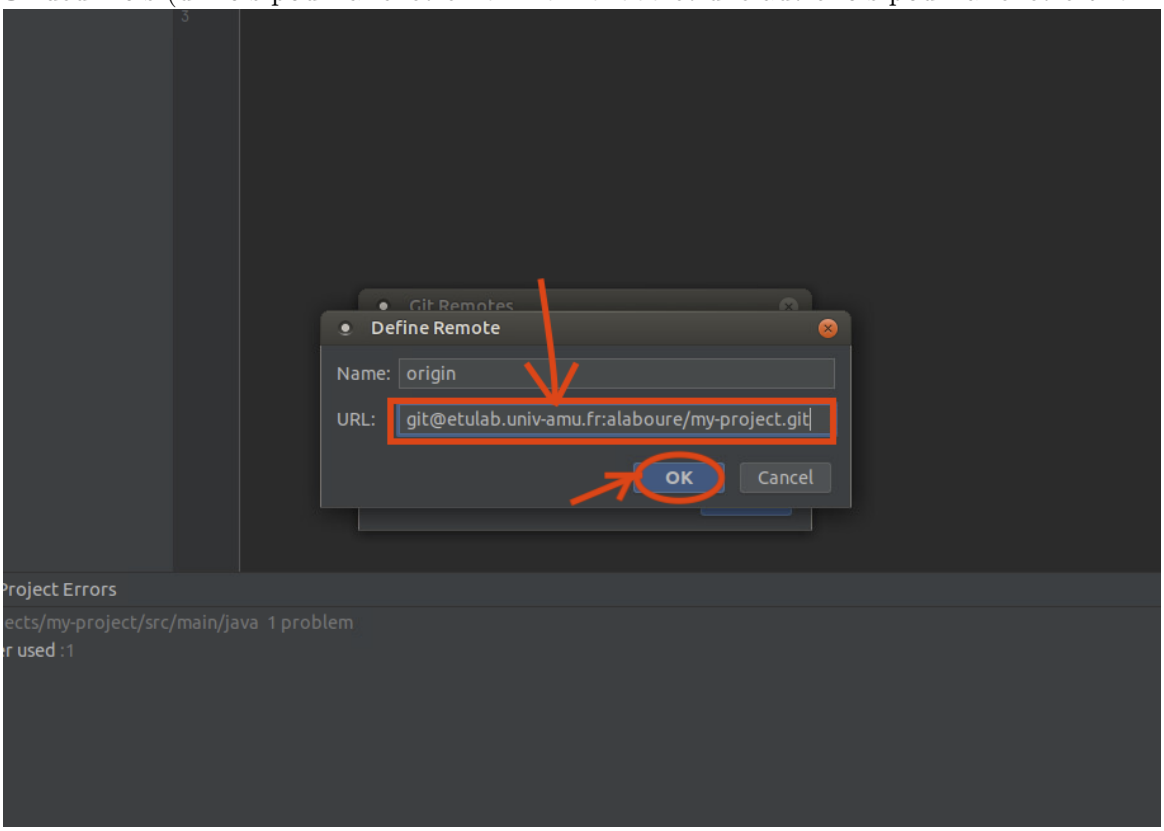
— Retournez sur votre projet sur `IntelliJ` et allez dans le menu : `git` puis `Manage Remotes`.



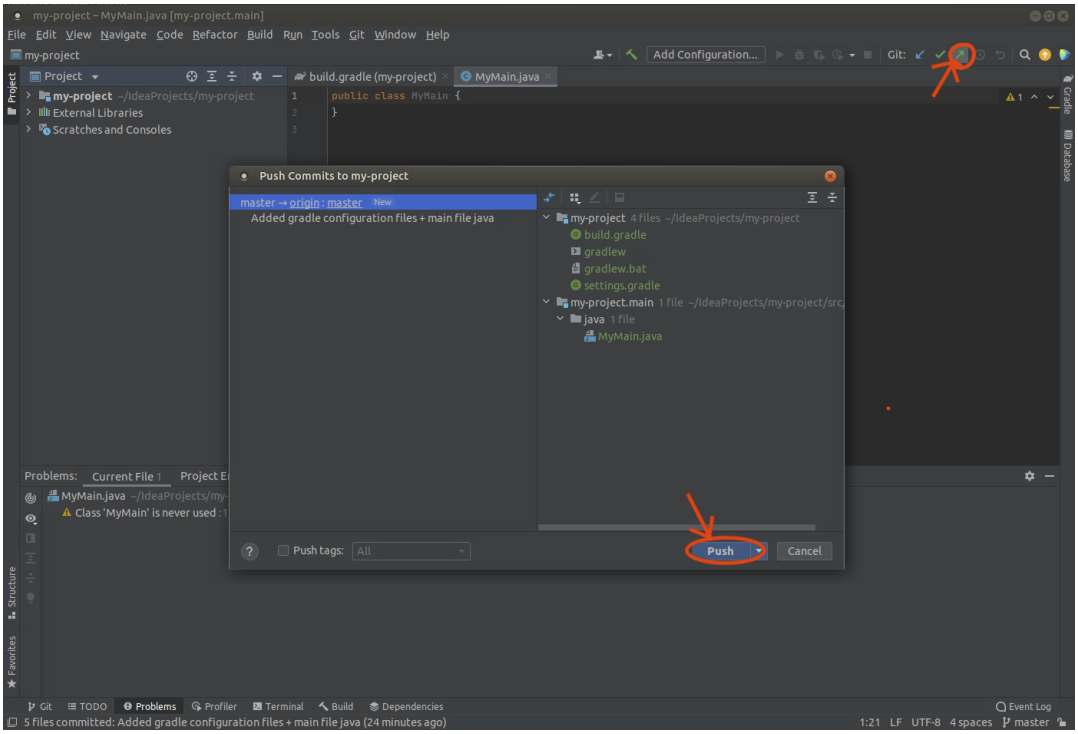
— Cliquez sur le bouton + pour ajouter un dépôt distant à votre dépôt local.



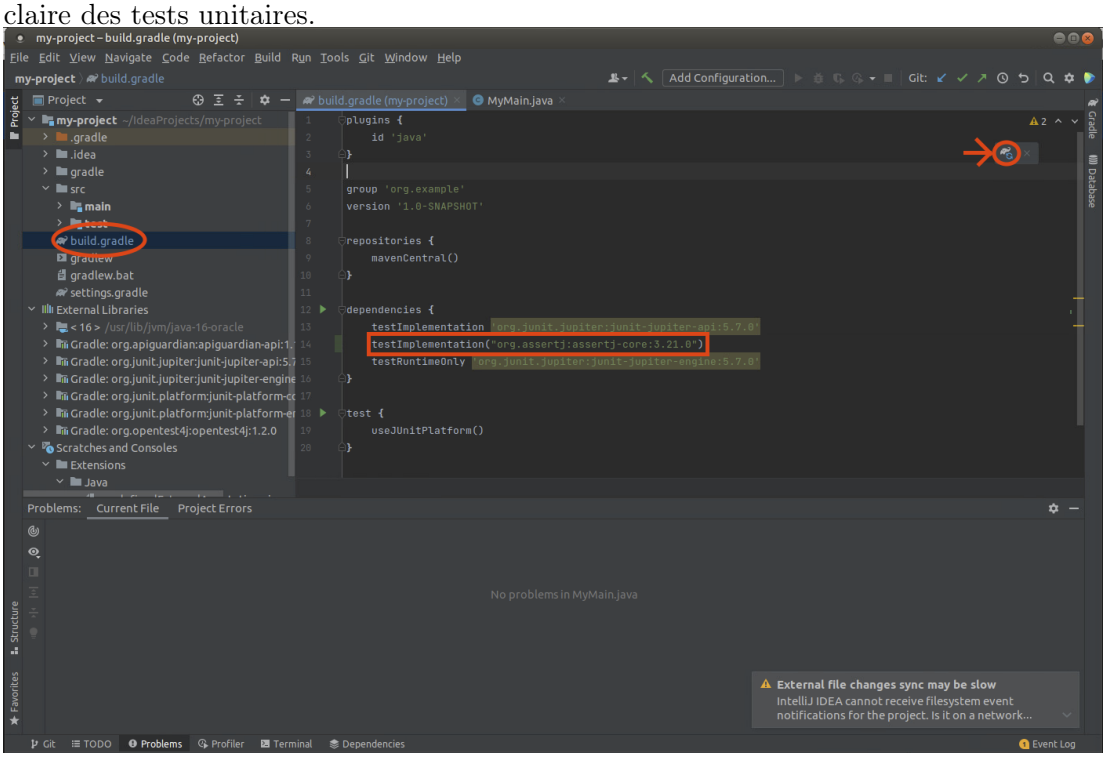
— Copiez l'adresse de votre dépôt etulab dans le champs `URL` et validez l'ajout en cliquant sur le bouton `OK` deux fois (un fois pour la fenêtre `Define remote` et une autre fois pour la fenêtre `Git remotes`).



— Vous pouvez maintenant faire le premier `push` de votre projet en cliquant par exemple dans le bouton dédié en haut à droite puis en validant en cliquant sur le bouton `push`.



— Vous pouvez ajouter des dépendances (bibliothèques utilisées dans votre projet) en modifiant le fichier `build.gradle`. Vous pouvez par exemple ajouter la ligne `testImplementation("org.assertj:assertj-core:3.22.0")` dans les `dependencies` puis cliquez sur l'icône de mise à jour afin de rajouter la bibliothèque AssertJ qui permet la rédaction plus claire des tests unitaires.



2.2 Création de projet versionné en ligne de commande

Si vous avez créé votre projet avec IntelliJ IDEA, vous pouvez ignorer cette partie qui décrit la création de projet sans IntelliJ IDEA.

2.2.1 Création de projet gradle en ligne de commande

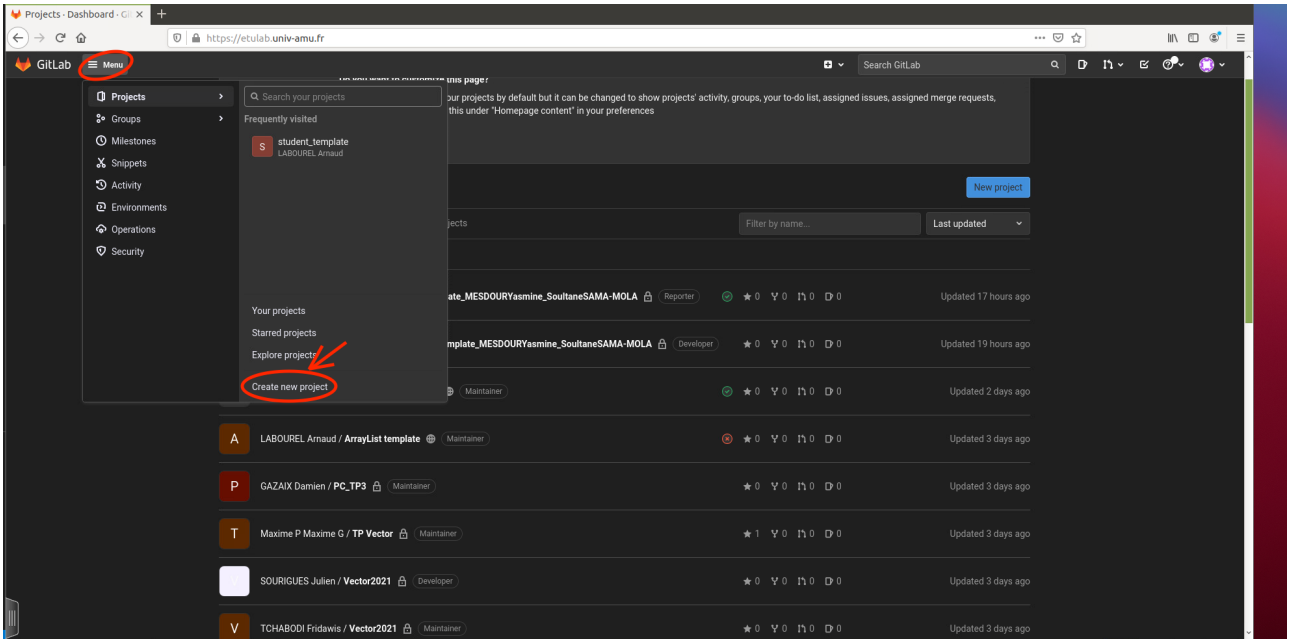
— La première étape de la création du projet est de créer un projet gradle. Gradle est ce que l'on appelle un moteur de production. C'est un outil pour compiler et exécuter les projets. Il est particulièrement utile pour gérer les dépendance de bibliothèques. En fait, cet outil télécharge automatiquement les bibliothèques configurées dans le fichier `build.gradle`.
— Créez le répertoire qui va contenir votre projet à l'aide de la commande `mkdir my-project` et placez-vous dedans à l'aide de la commande `cd my-project` (vous pouvez remplacer `my-project` par n'importe quel autre nom).
— Pour commencer la création de votre projet lancez la commande `gradle-7.3 init`.
  — On vous demande tout d'abord quel type de projet vous souhaitez créer. Tapez `2` puis `entrée` pour choisir application.
  — On vous demande le langage de projet. Tapez `2` puis `entrée` pour choisir Java.
  — On vous demande si votre projet va contenir des fonctionnalités partagées entre plusieurs sous-projets. Tapez `1` puis `entrée` pour choisir `No`.
  — On vous demande si vous préférez utiliser `Groovy` ou `Kotlin`. Tapez `2` puis `entrée` pour choisir `Groovy`.
  — On vous demande quel framework de tests vous souhaitez utiliser. Tapez `4` puis `entrée` pour choisir `Junit Jupiter`.
  — On vous demande le nom de votre projet. Vous pouvez laisser le nom par défaut ou bien définir votre propre nom de projet.
  — On vous demande le `package` par défaut de votre projet. Vous pouvez laisser le `package` par défaut ou bien définir votre propre `package`. Pour le reste de ce tutoriel on supposera que le nom du package est `demo`.
— L'arborescence de fichier de votre projet devrait être la suivante :

— Le fichier `App.java` contient la méthode `main` qui est exécuté lorsque vous tapper la commande `gradle run`.
— Le fichier `AppTest.java` contient des tests qui sont exécutés lorsque vous tapper la commande `gradle test`. Si les tests échouent la commande vous donne un lien à ouvrir avec un navigateur.

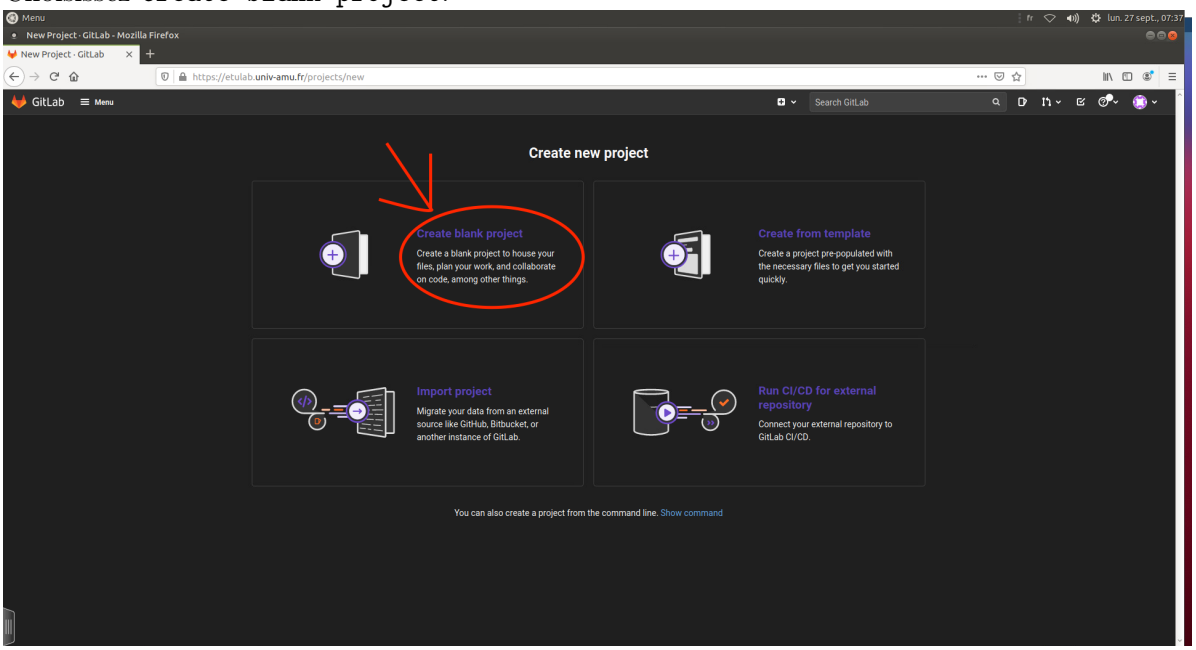
2.2.2 Création de dépôt `git` en ligne de commande

— Dans le répertoire contenant votre projet `gradle`, exécutez la commande `git init --initial-branch=main` pour créer un dépôt `git` local.
— Dans le répertoire contenant votre projet `gradle`, exécutez la commande `git add non_de_fichier` pour ajouter les fichiers de votre projet à votre prochain `commit`.
— Il vous faudra ajouter tous les fichiers `Java` ainsi que les fichiers de configuration de gradle : `build.gradle`, `gradlew`, `gradlew.bat` et `settings.gradle`.

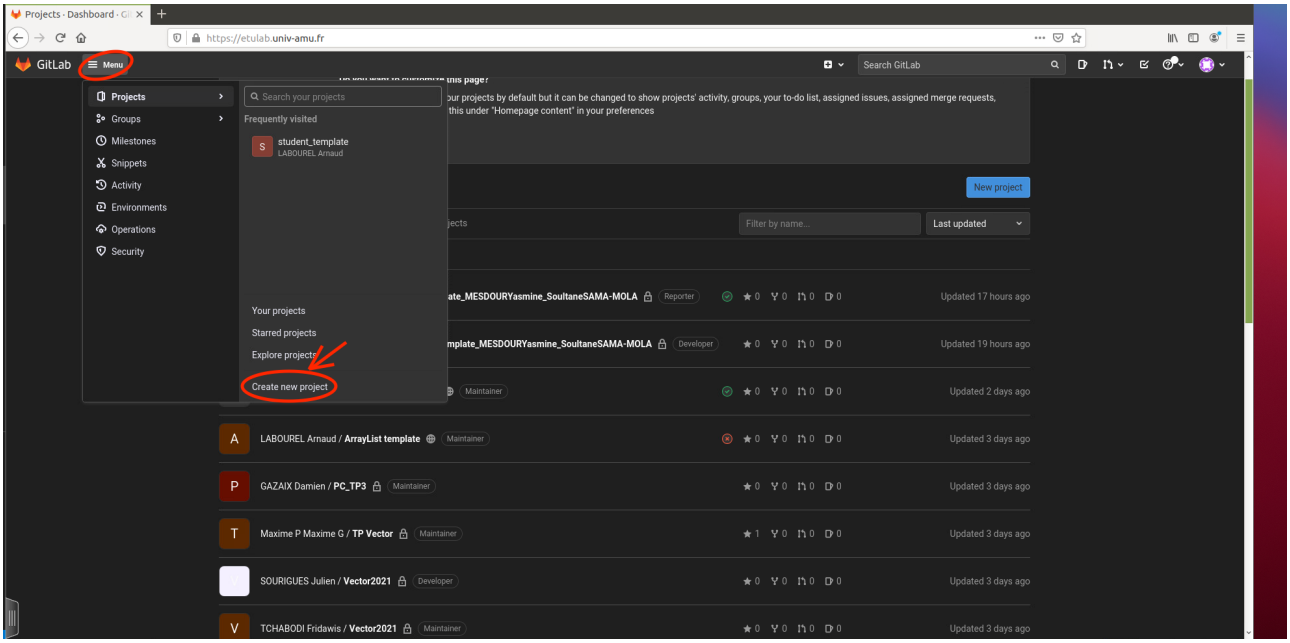
— Exécutez la commande `git commit -m"premier message"` pour faire un premier `commit` de votre projet avec votre propre message.
— Connectez vous via à un navigateur à votre compte etulab et créez un nouveau projet en cliquant sur `menu` puis `Create new project`.



— Choisissez `Create blank project`.



— Choisissez un nom pour votre projet, décochez la création du `README.md` et validez la création en cliquant sur le bouton `Create project`.



— Copiez l'adresse pour cloner votre projet en cliquant sur le bouton `clone` puis sur l'icône de copie `ssh`.
— Utiliser la commande suivante pour rajouter l'adresse du dépôt distant à votre dépôt local. Il vous faudra bien évidemment remplacer le dernier argument de la commande par l'adresse de votre projet que vous avez préalablement copiée.

```
git remote add origin git@git@etulab.univ-amu.fr:votre_login/le-nom-de-votre_projet.git
```

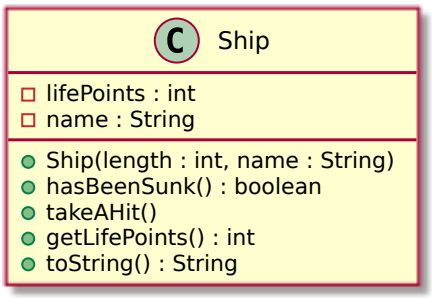
— Faites le premier `push` à l'aide de la commande `git push -u origin master`. Pour les `push` suivants, vous pourrez simplement utiliser la commande `git push` car vous avez configuré les branches.

3 La classe `Ship`

3.1 Spécification de la classe `Ship`

La classe `Ship` permet de représenter les bateaux. Une instance de `Ship` est construite avec une longueur et un nom. La longueur d'un bateau détermine son nombre de points de vie initial (*life points*). Lorsqu'il est touché (méthode `takeAHit()`), ce nombre est décrémenté d'un. Un bateau est coulé quand son nombre de points de vie arrive à 0 (méthode `hasBeenSunk()`).

Voici le diagramme de cette classe :



3.2 Tâches

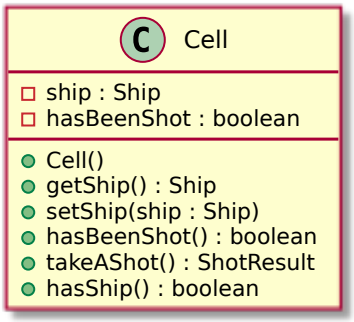
Tâche 1 : Créez la classe `Ship` dans le répertoire `src/main/java` de votre projet qui répond aux spécifications ci-dessus.

Tâche 2 : Créez une classe de test nommée `ShipTest` dans le répertoire `src/test/java` de votre projet qui devra vérifier via des tests la spécification du comportement de la classe `Ship`.

4 La classe `Cell`

4.1 Spécification de la classe `Cell`

La classe `Cell` permet de représenter les cases du plateau de jeu. Son diagramme de classe est donné ci-dessous. Une case est initialement vide. Mais elle peut être occupée par un seul bateau (méthode `getShip()`). Il faut donc pouvoir poser un bateau sur une case (méthode `setShip()`). L'attaquant peut tirer sur une case (méthode `takeAShot()`). Si la case est occupée, lors du premier de ces tirs, cela a pour conséquence que le bateau correspondant est touché. Seul le premier tir sur une case compte, un bateau ne peut pas être touché deux fois par un tir sur la même case. Il est donc nécessaire de pouvoir savoir si une case a déjà été visée ou non par un tir de l'attaquant (peu importe qu'elle comporte initialement un bateau ou non) grace à la méthode `hasBeenShot()`.



On appelle `ShotResult` le type permettant de représenter les 3 réponses possibles après un tir d'un attaquant : `MISSED`, `HIT` et `SUNK`.

4.2 Tâches

Tâche 3 : Créez le type `ShotResult` dans le répertoire `src/main/java` de votre projet.

Tâche 4 : Créez la classe `Cell` dans le répertoire `src/main/java` de votre projet qui répond aux spécifications ci-dessus.

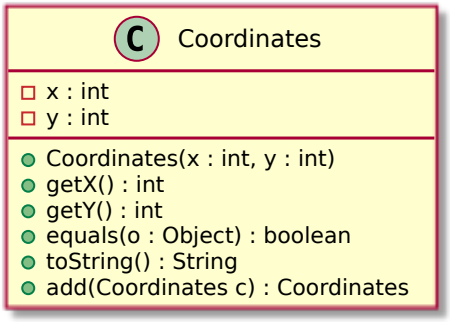
Tâche 5 : Créez une classe de test nommée `CellTest` dans le répertoire `src/test/java` de votre projet qui devra vérifier via des tests la spécification du comportement de la classe `Cell`.

5 La classe `Coordinates`

5.1 Spécification de la classe `Coordinates`

Pour représenter les coordonnées des cases du plateau de jeu, on définit la classe `Coordinates`. La méthode `toString` renvoie un chaîne de caractère au format suivant : `(X, Y)` avec `X` la coordonnée en x et `Y` la coordonnée en y. Deux positions sont considérées égales si elles ont les mêmes coordonnées en x et en y.

Voici son diagramme de la classe :



5.2 Tâches

Tâche 6 : Créez la classe `Coordinates` dans le répertoire `src/main/java` de votre projet qui répond aux spécifications ci-dessus.

Tâche 7 : Créez une classe de test nommée `CoordinatesTest` dans le répertoire `src/test/java` de votre projet qui devra vérifier via des tests la spécification du comportement de la classe `Coordinates`.

6 La classe `Position`

6.1 Spécification de la classe `Position`

Pour représenter le position d'un bateau dans une grille, on définit la classe `Position`.

Une position a trois attributs :

— `length` : le nombre de cases occupées par le bateau

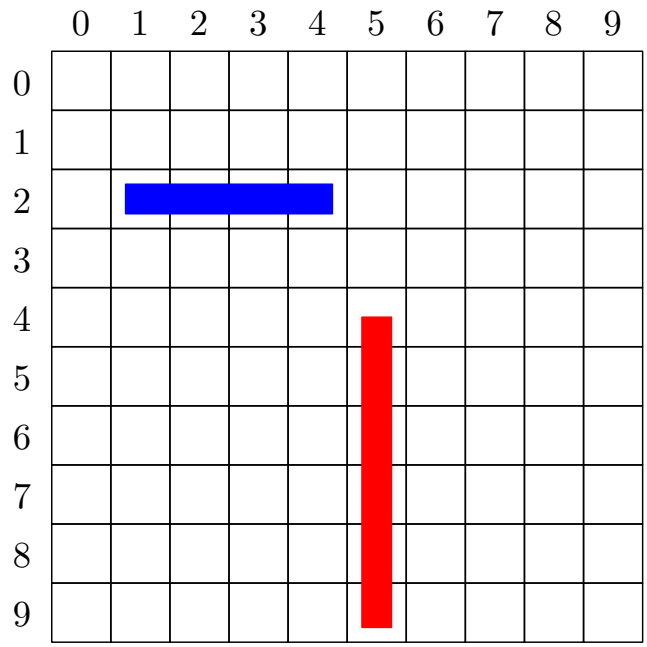
— `firstCell` : les coordonnées de la case ayant les plus petites coordonnées en x et en y parmi les cases occupées par le bateau.

— `orientation` : l'orientation du bateau (`HORIZONTAL` ou `VERTICAL`)

La figure ci-dessous illustre la valeur des attributs d'instance de `Position` pour deux bateaux (`position1` correspond au bateau en bleu alors que `position2` correspond au bateau en rouge).

position1
length = 4
orientation = HORIZONTAL
firstCell = (1, 2)

position2
length = 6
orientation = VERTICAL
firstCell = (5, 4)

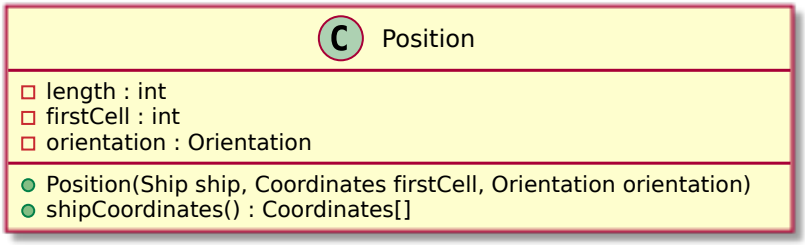


La classe possédera une seule méthode `shipCoordinates` qui renverra un tableau des cases occupés par le bateau. Pour les deux positions de la figure, les retours de la méthode devra être les suivants :

— `position1.shipCoordinates()` devra renvoyer le tableau de coordonnées $[(1,2),(2,2),(3,2),(4,2)]$

— `position2.shipCoordinates()` devra renvoyer le tableau de coordonnées $[(5,4),(5,5),(5,6),(5,7),(5,8),(5,9)]$

Le diagramme de la classe est le suivant :



6.2 Tâches

Tâche 8 : Créez le type `Orientation` dans le répertoire `src/main/java` de votre projet.

Tâche 9 : Créez la classe `Position` dans le répertoire `src/main/java` de votre projet qui répond aux spécifications ci-dessus.

Tâche 10 : Créez une classe de test nommée `PositionTest` dans le répertoire `src/test/java` de votre projet qui devra vérifier via des tests la spécification du comportement de la classe `Position`.

7 La classe `Grid`

7.1 Spécification de la classe `Grid`

La classe représentant le plateau de jeu (la grille) s'appelle `Grid`. On décide de représenter son état par un tableau à deux dimensions de cases qui sont des objets de type `Cell`.

La méthode `shootAt` de la classe `Grid` est utilisée lorsque le joueur attaquant vise une case. Son résultat est la réponse lorsque l'attaquant vise la case située à la position `p` sur le plateau de jeu. La position `p` est passée en paramètre de la méthode.

7.2 Questions

Tâche 11 : Créez la classe `Grid` dans le répertoire `src/main/java` de votre projet. C'est à vous de définir les signatures des méthodes de cette classe ainsi que leur comportement.

Tâche 12 : Créez une classe de test nommée `GridTest` dans le répertoire `src/test/java` de votre projet qui devra vérifier via des tests la spécification du comportement de la classe `Grid`.

8 La classe `App` (bonus)

Tâche 13 : Créez une classe `App` contenant une méthode `main` qui permet de jouer à la bataille navale. On pourra imaginer au début une version très simplifiée dans laquelle il n'y a qu'une grille et un unique joueur peut décider à chaque tour de coordonnées de tir, le but étant de couler tous les bateaux en un minimum de tours de jeu.

9 Rappels sur les tests

9.1 Méthodologie de test

Une classe de test par classe à tester. Une méthode de test par méthode ou cas à tester.

Le code d'une classe de test testant une classe `NameTestedClass` a le format suivant :

```
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.*;

public class NameTestedClassTest {
  @Test
  void testTestMethod(){
    // code containing assertions
  }
}
```

9.2 Assertions (liste non-exhaustive)

Pour tester, on utilise des assertions qui doivent être vraies. Vous trouverez ci-dessous une listes des assertions les plus utiles.

- `assertThat(object).isNotNull()` : vérifie que la référence **n'**est **pas** `null`
- `assertThat(actual).isSameAs(expected)` : vérifie que les deux objets sont les mêmes (même référence : utilisation de `==`).
- `assertThat(condition).isTrue()` : vérifie que `condition` est vraie.
- `assertThat(condition).isFalse()` : vérifie que `condition` est faux.
- `assertThat(actual).isEqualTo(expected)` : vérifie que `expected` est égal à `actual` (en appelant `equals` sur `actual`).
- `assertThat(actual).isNotEqualTo(expected)` : vérifie que `expected` **n'**est égal **pas** à `actual` (en appelant `equals` sur `actual`).
- `assertThat(iterable).contains(e1, e2, ..., ek)` : vérifie que `iterable` (qui peut être une `List`, un tableau, ...) contient `e1, e2, ..., ek`.
- `assertThat(iterable).containsOnly(e1, e2, ..., ek)` : vérifie que `iterable` (qui peut être une `List`, un tableau, ...) contient exactement `e1, e2, ..., ek` dans cet ordre.
- `assertThat(actual).isCloseTo(expected, within(delta))` : vérifie que $|expected - actual| \leq delta$ (comparaison de double).