amU Faculté des sciences Aix Marseille Université

Année 2024-25

Domaine Sciences et Technologies
Licence Informatique : math-info
Initiation au Génie logiciel : TP 2
Codes UE : SIN5U34
TP 2 : Bataille navale

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.

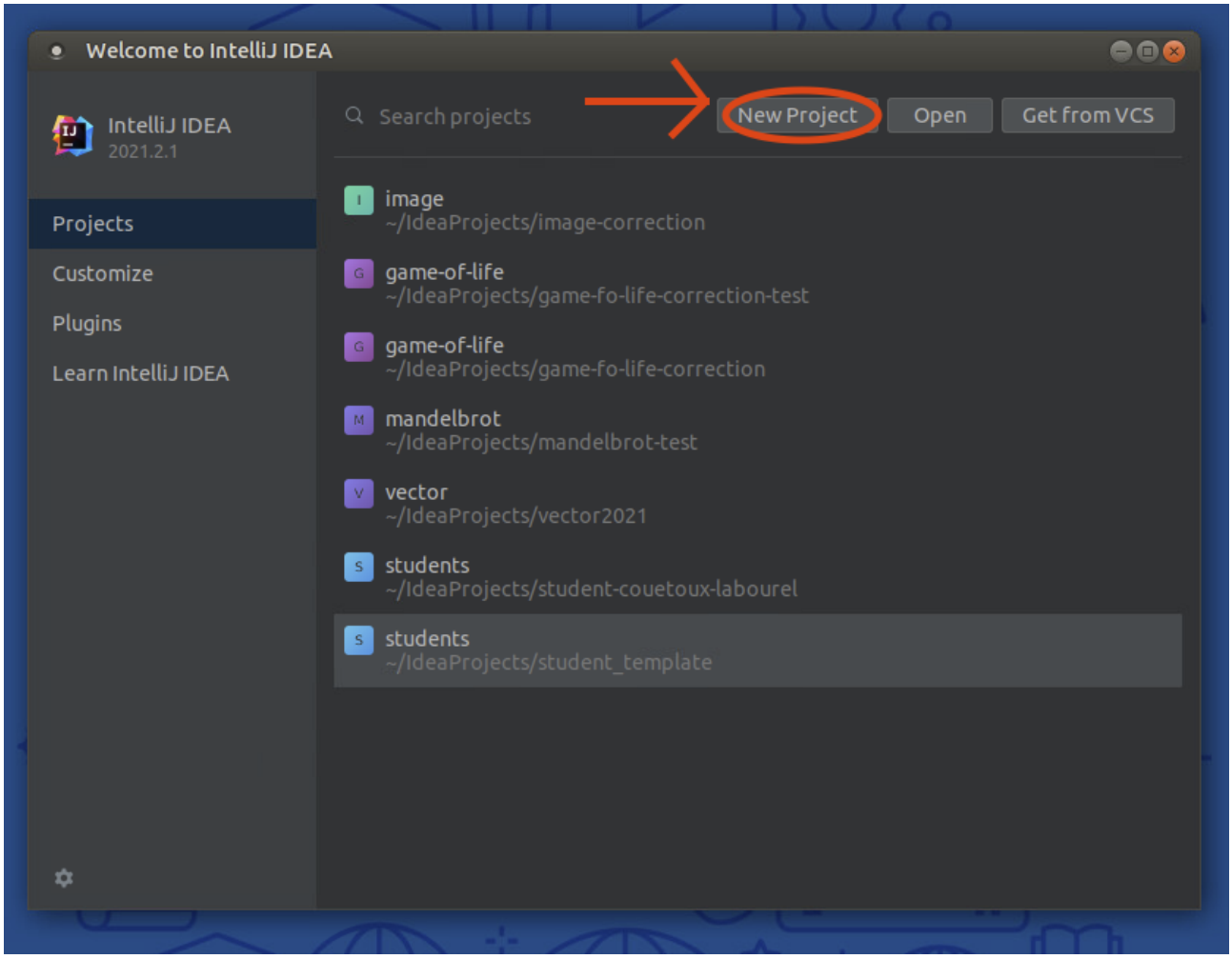
L'objectif de ce TP sera de programmer une version très simplifiée du jeu de bataille navale.

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. Vous pouvez au choix soit créer votre projet à l'aide d'IntelliJ ou bien en ligne de commande en suivant les instructions d'une des deux sections suivantes.

Création de projet versionné à l'aide d'IntelliJ

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

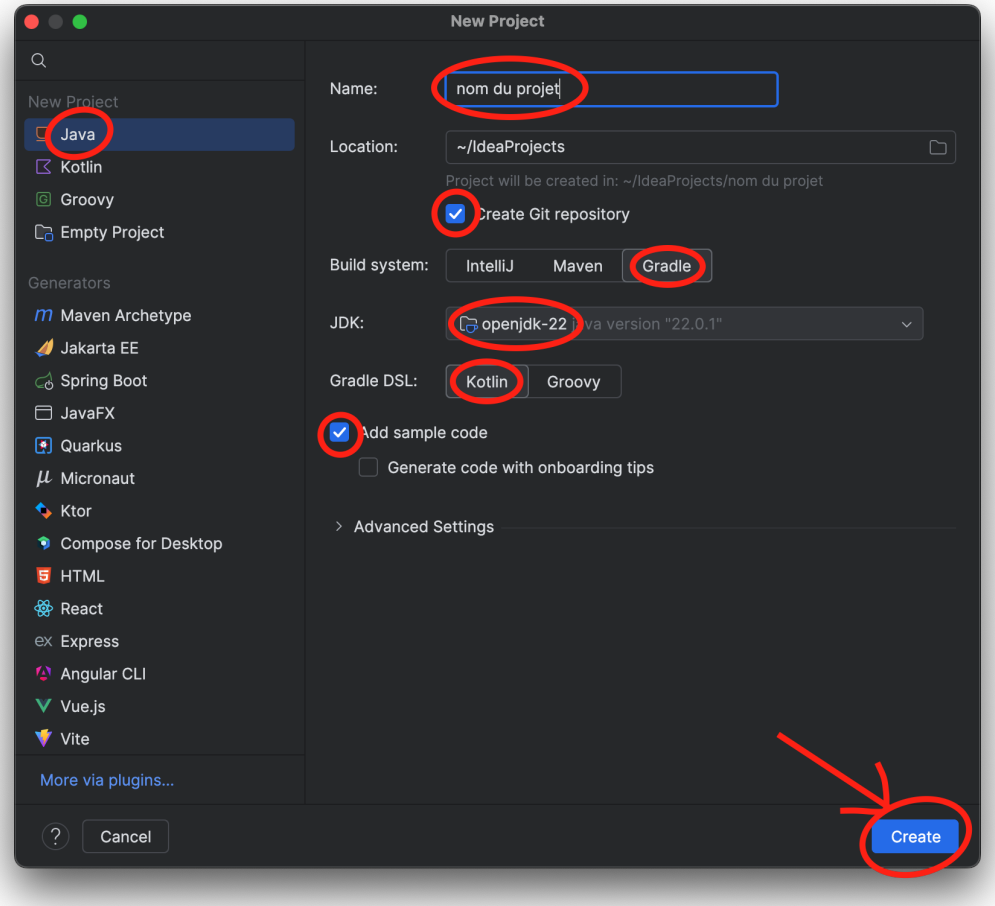


Vous pouvez aussi créer un nouveau projet en allant dans le menu `file` -> `New` -> `Project`.

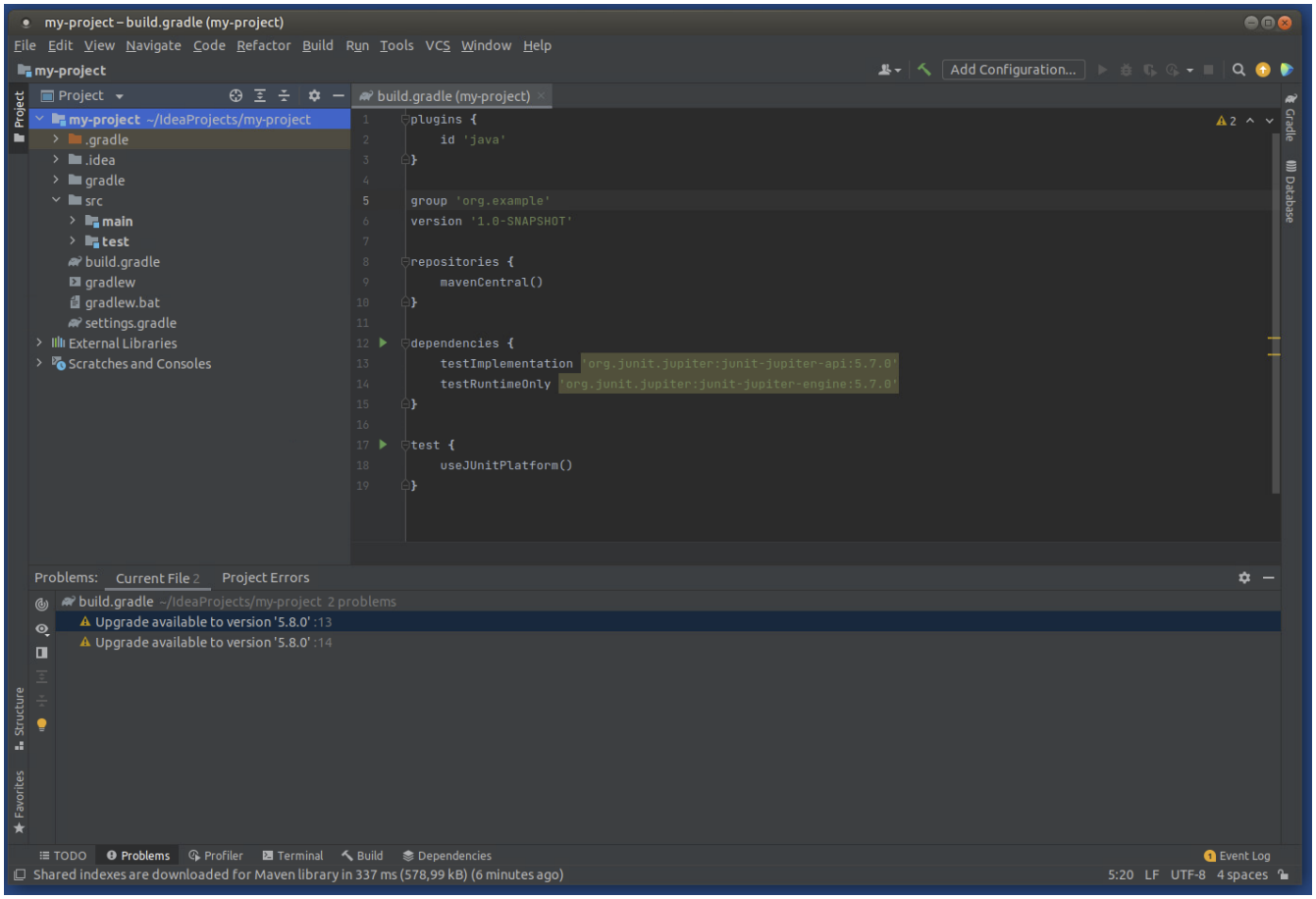
— Créer un projet. Pour ce faire, il vous faut (dans l'ordre du haut vers le bas) :

— choisir un nom (*name*) pour votre projet,
— cocher la case *Create Git repository* pour créer un dépôt git local,
— choisir Gradle comme moteur de production (*Build system*),
— choisir un JDK de version 17 ou plus et
— cliquez sur le bouton *Create*.

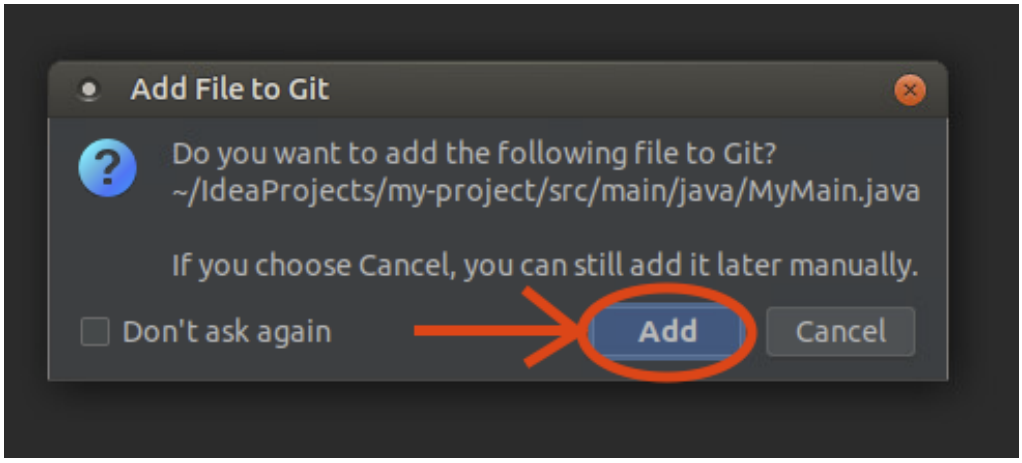
Gradle est ce que l'on appelle un moteur de production. C'est un outil pour compiler et exécuter les projets. Il est particulièrement utile pour gérer les dépendances de bibliothèques (code issu de base de code préexistante). En fait, cet outil télécharge automatiquement les bibliothèques configurées dans le projet dans le fichier `build.gradle`.



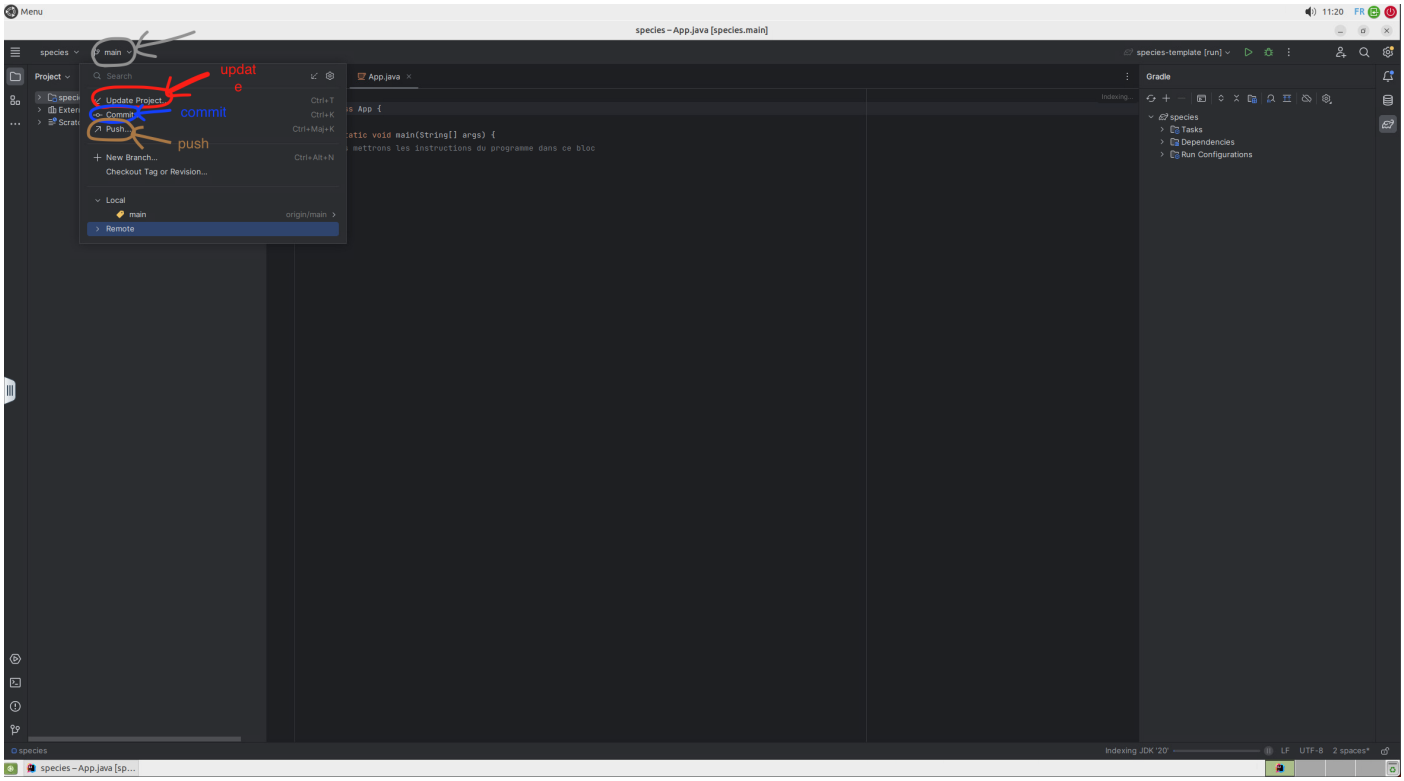
— Votre projet devrait se configurer automatiquement et vous devriez obtenir l'affichage suivant.



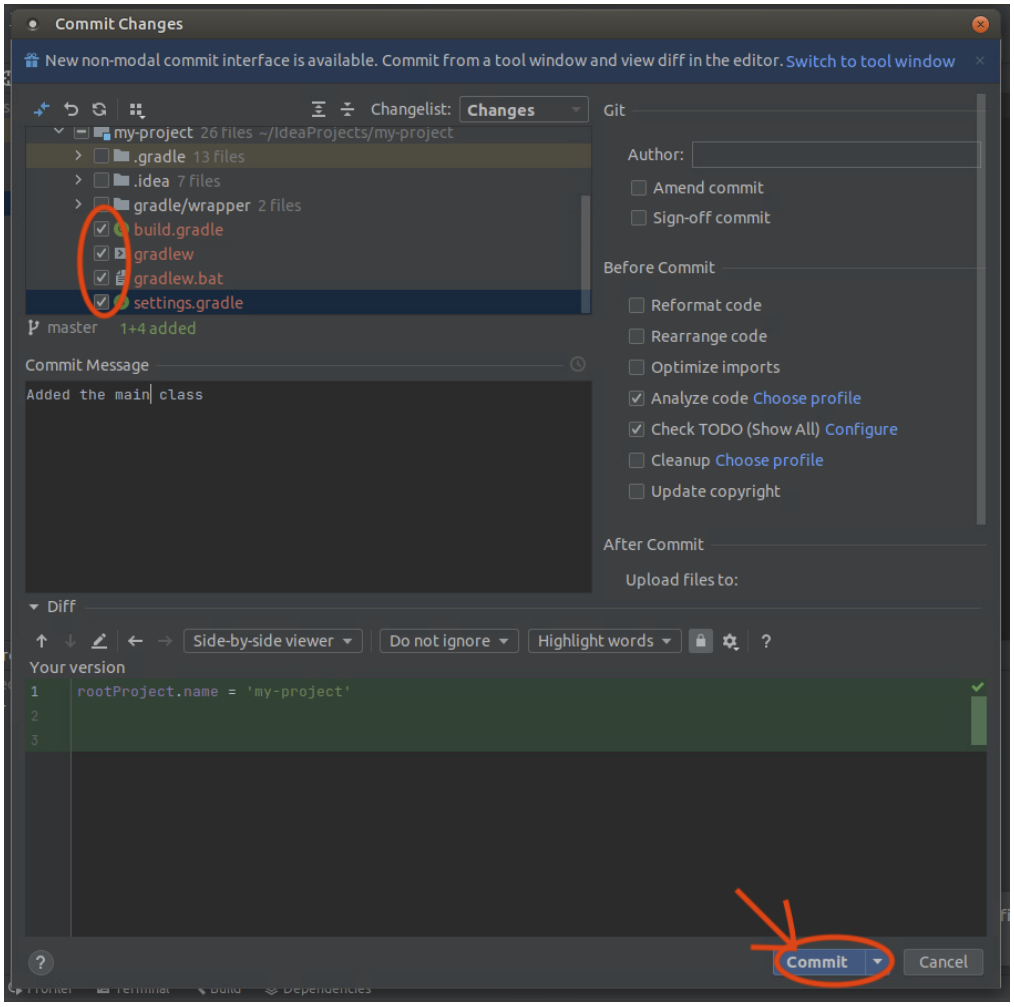
— Créez une classe dans le répertoire `src` -> `main` -> `java`. Une fenêtre va s'ouvrir pour demander si vous voulez rajouter le fichier contenant votre classe dans le dépôt `git`. Cliquez sur le bouton `add` pour valider cet ajout.



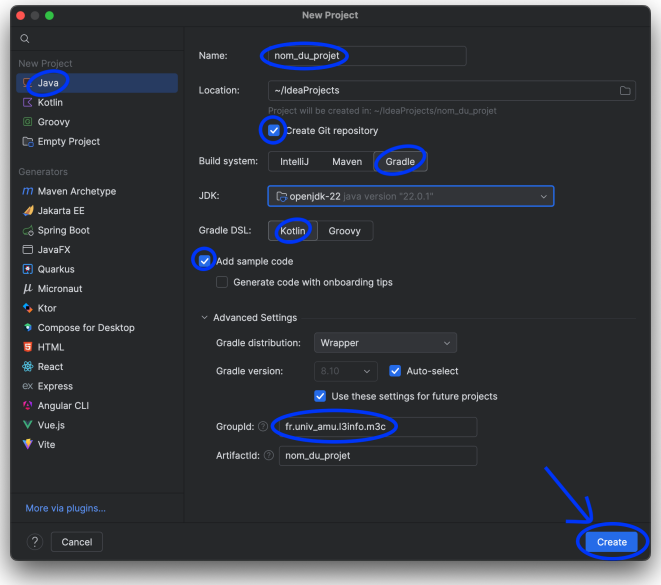
— Vous pouvez effectuer les opérations de base de git (`update` du projet, `commit` et `push`) grace aux raccourcis en haut à droite de la fenêtre.



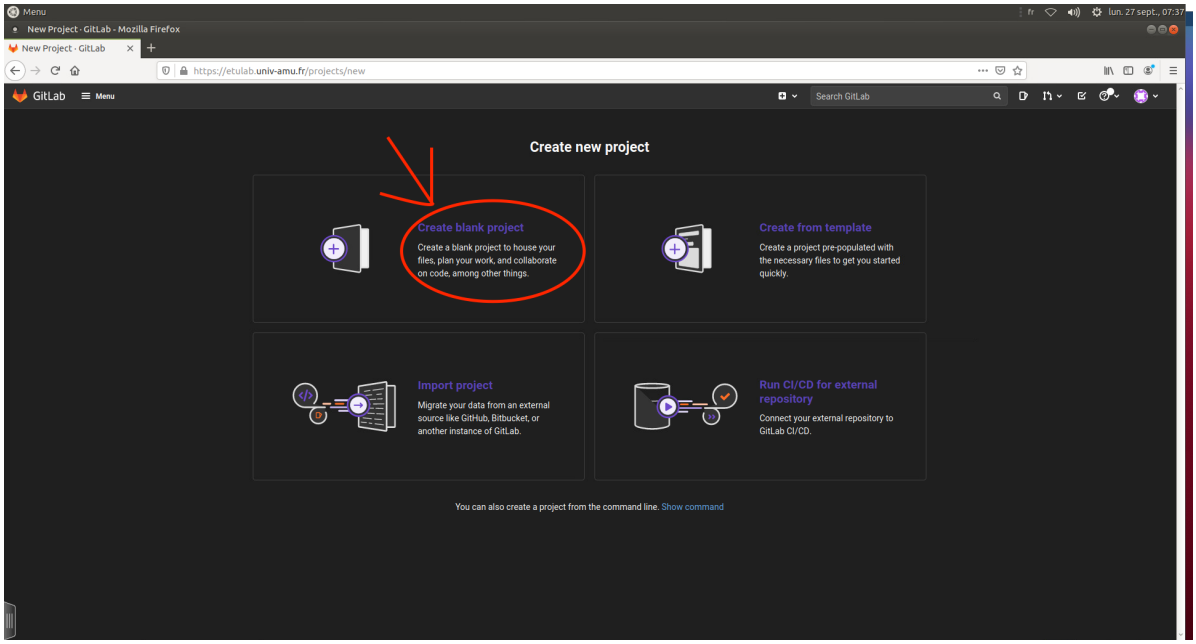
— Faites le premier commit de votre projet en ajoutant en plus de vos fichiers `.java` les fichiers de configuration de gradle : `build.gradle`, `gradlew`, `gradlew.bat` et `settings.gradle`.



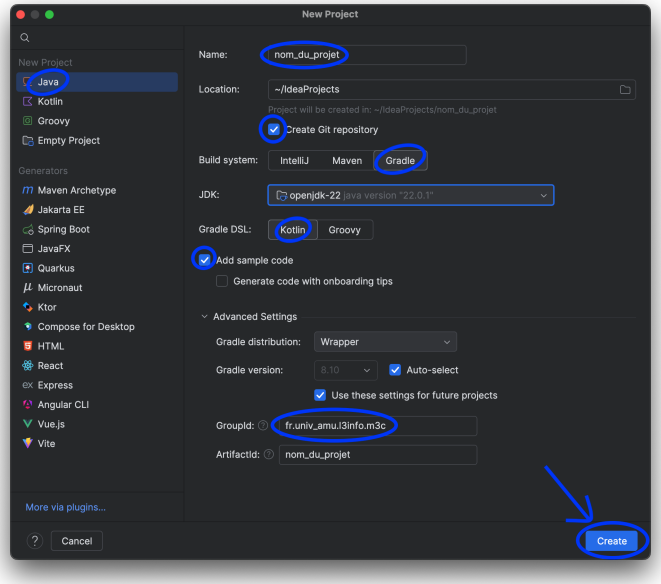
— Connectez vous via à un navigateur à votre compte etulab et créez un nouveau projet en cliquant sur `menu` puis `Create new project`.



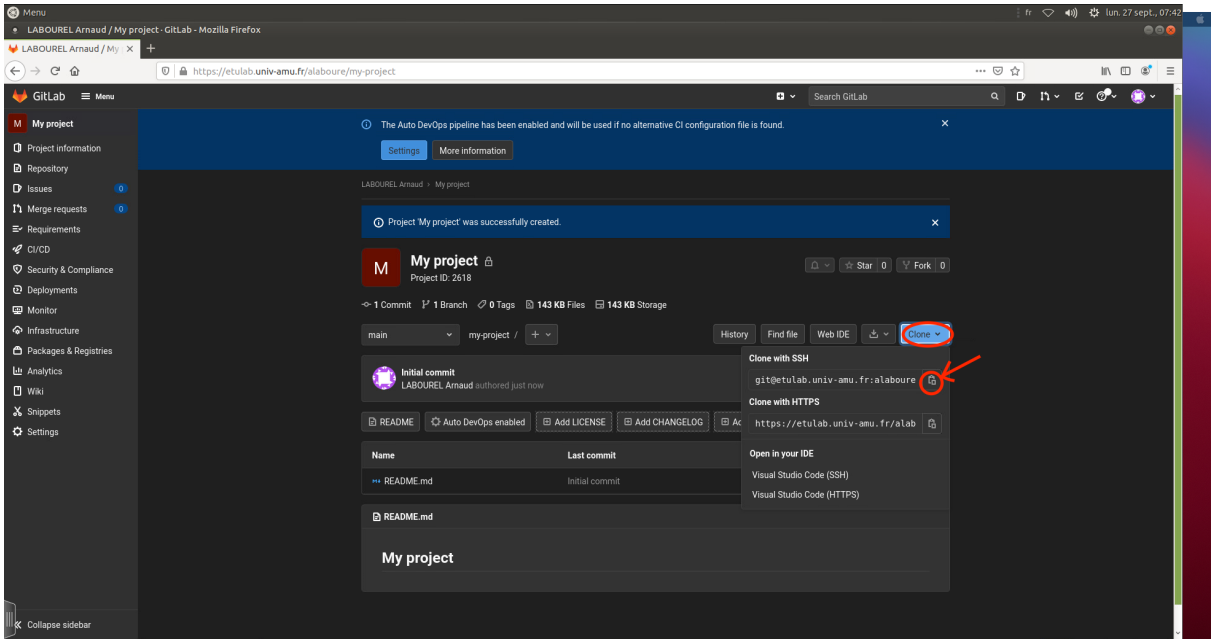
— Choisissez `Create blank project`.



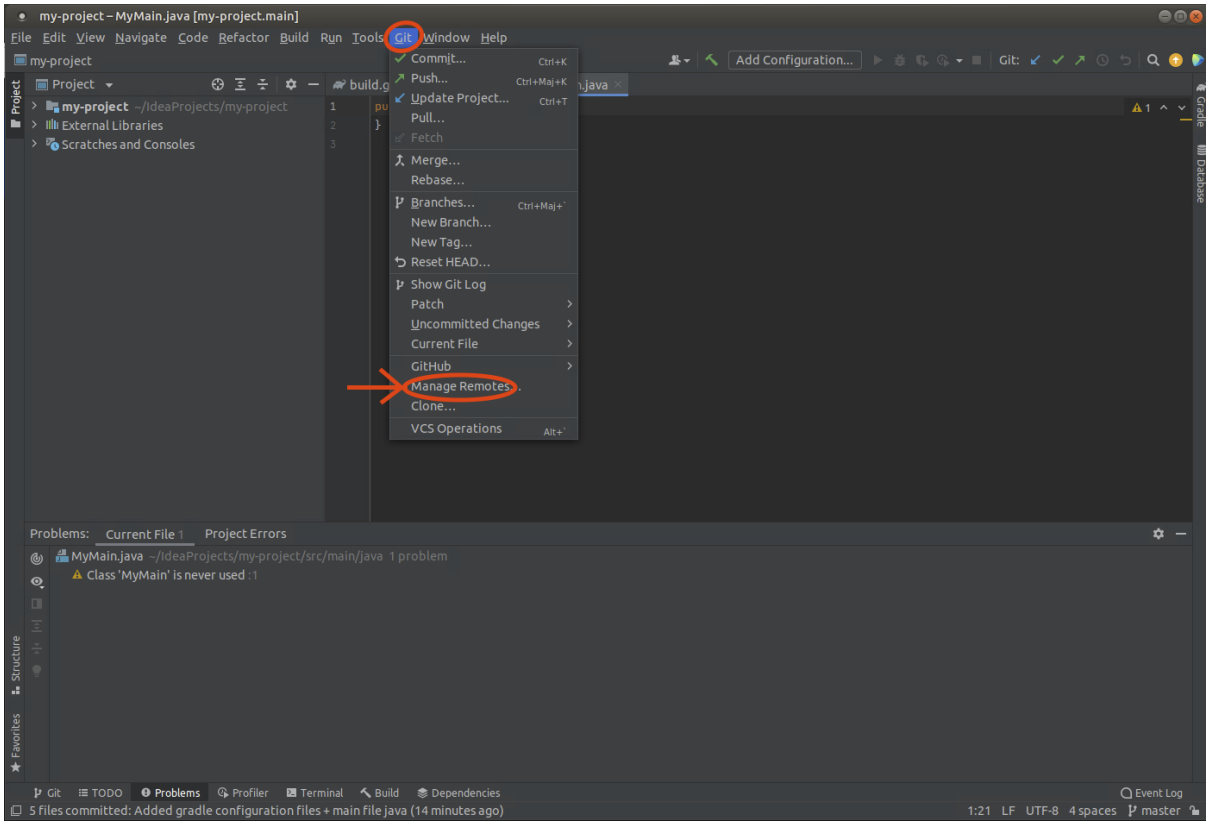
— Choisissez un nom pour votre projet, décochez la création du `README.md` et validez la création en cliquant sur le bouton `Create project`.



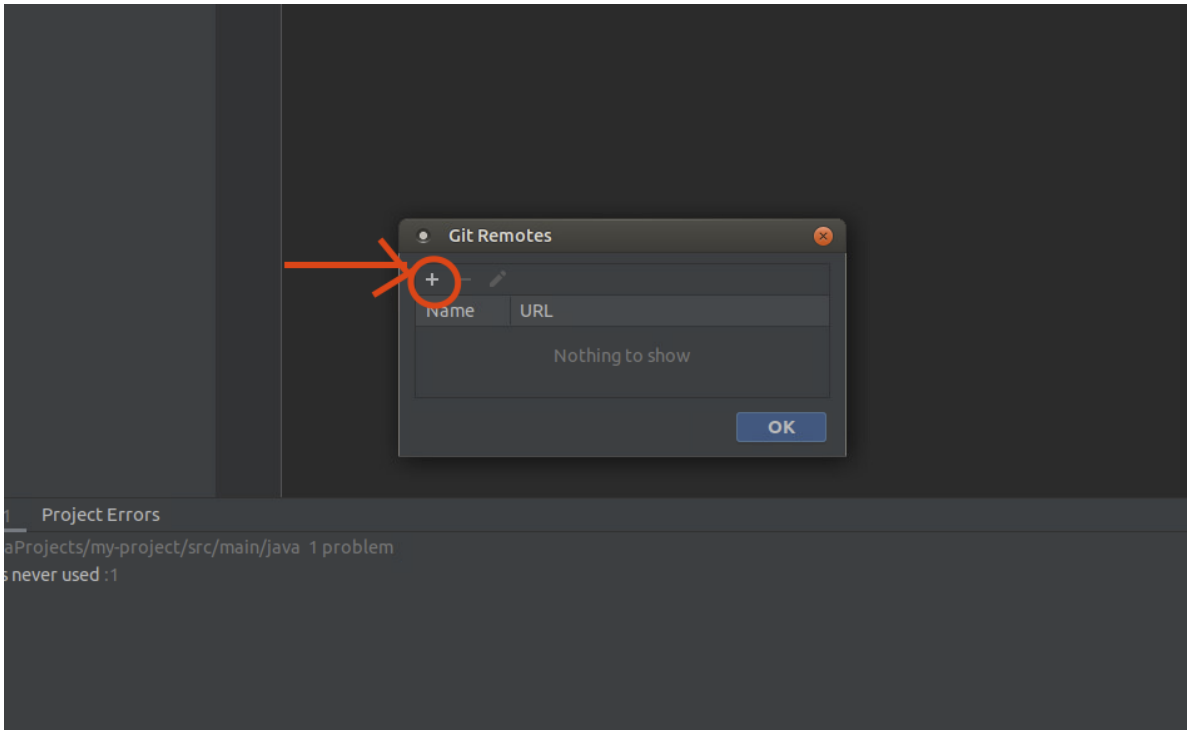
— Copiez l'adresse pour cloner votre projet en cliquant sur le bouton `clone` puis sur l'icône de copie `ssh`.



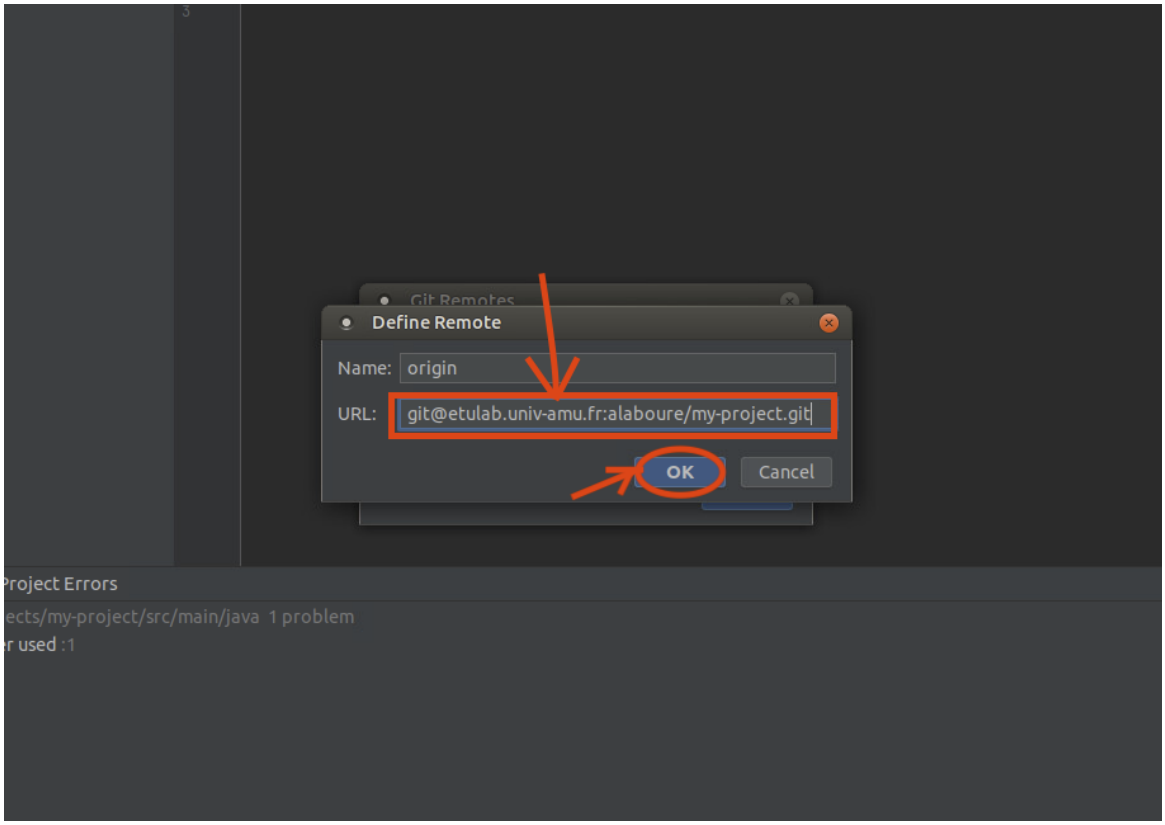
— Retournez sur votre projet sur `IntelliJ` et allez dans le menu : `git` puis `Manage Remotes`.



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

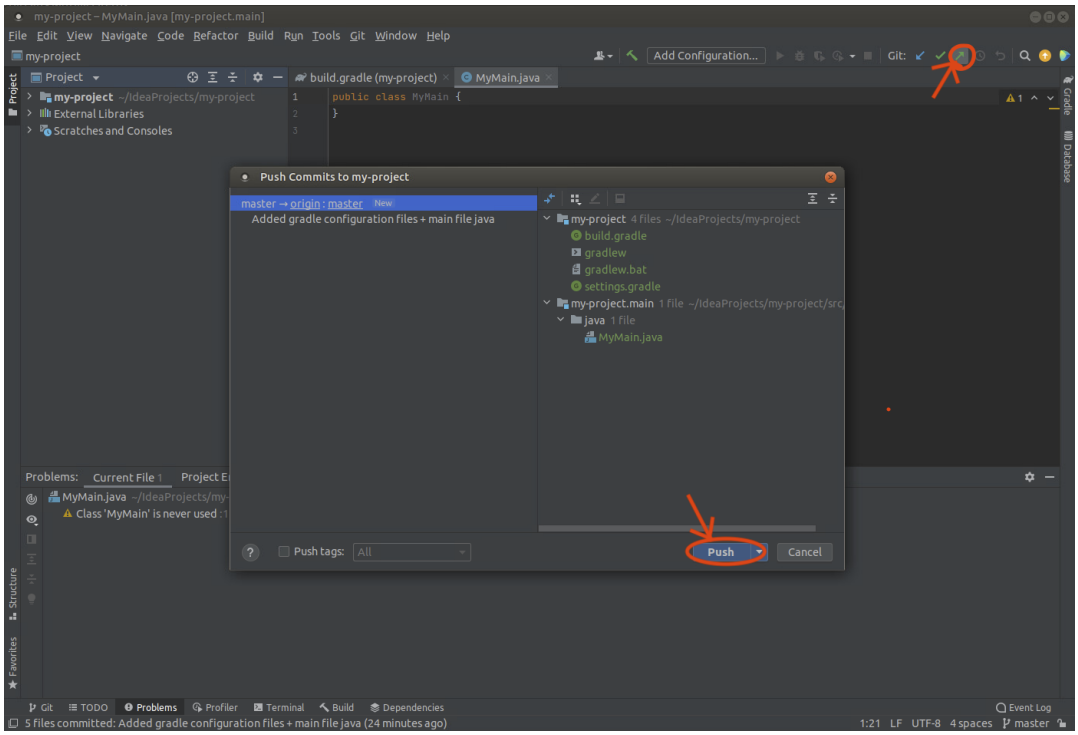


— Copiez l'adresse de votre dépôt etulab dans le champ `URL` et validez l'ajout en cliquant sur le bouton `OK` deux fois (un fois pour la fenêtre `Define remote` et une autre fois pour la fenêtre `Git remotes`).

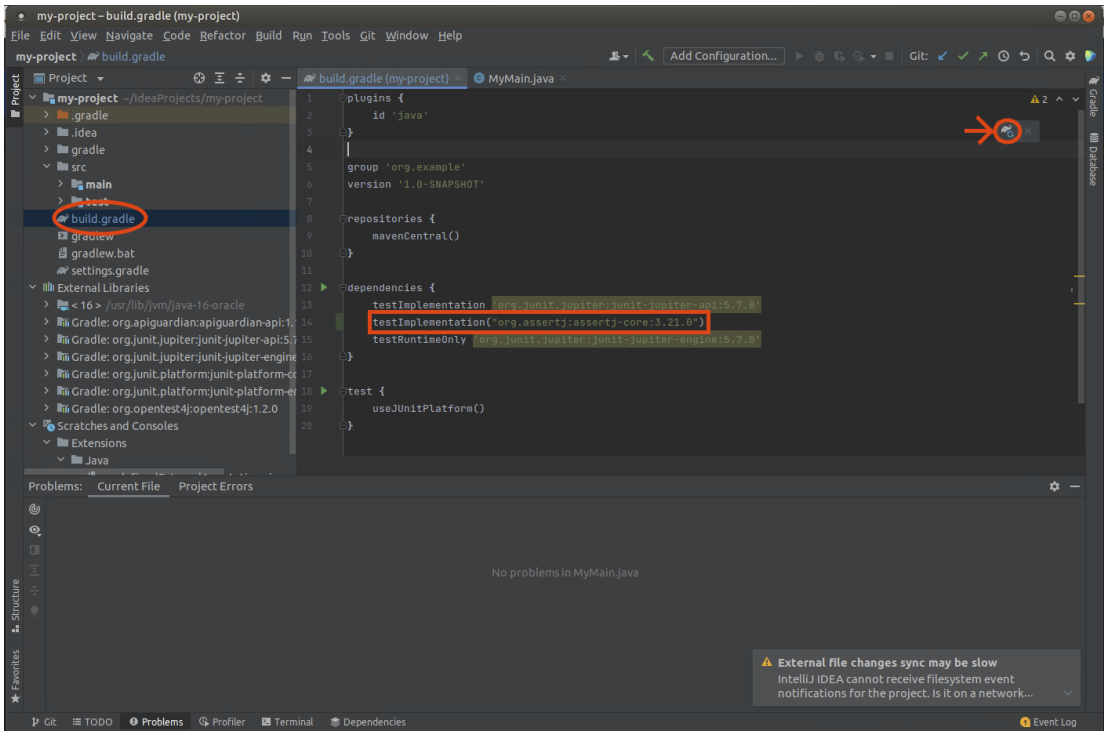


— Vous pouvez maintenant faire le premier `push` de votre projet en cliquant par exemple dans le bouton

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:3.23.1')` 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.



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 utilisant uniquement le terminal shell et donc sans IntelliJ IDEA.

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. Cet outil gère le téléchargement automatique des 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 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 taper la commande `gradle run`.

— Le fichier `AppTest.java` contient des tests qui sont exécutés lorsque vous taper la commande `gradle test`. Si les tests échouent la commande vous donne un lien à ouvrir avec un navigateur.

Création de dépôt `git` en ligne de commande

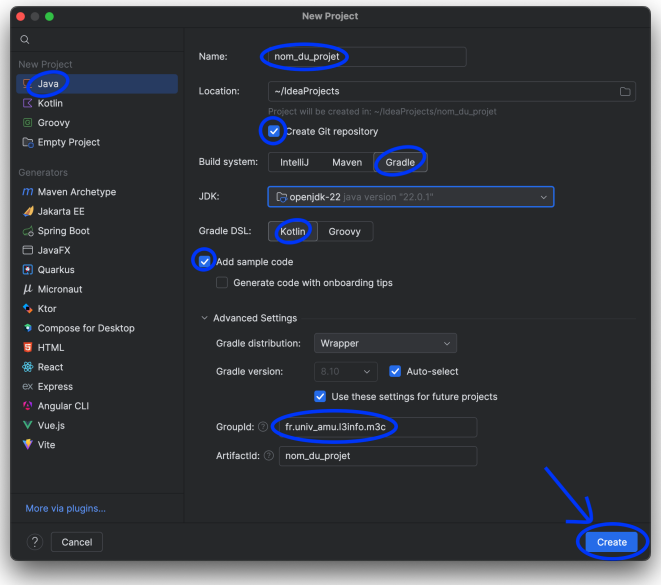
— Dans le répertoire contenant votre projet `gradle`, exécutez la commande `git init --initial-branch=main` pour créer un dépôt `git` local.

— 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`.

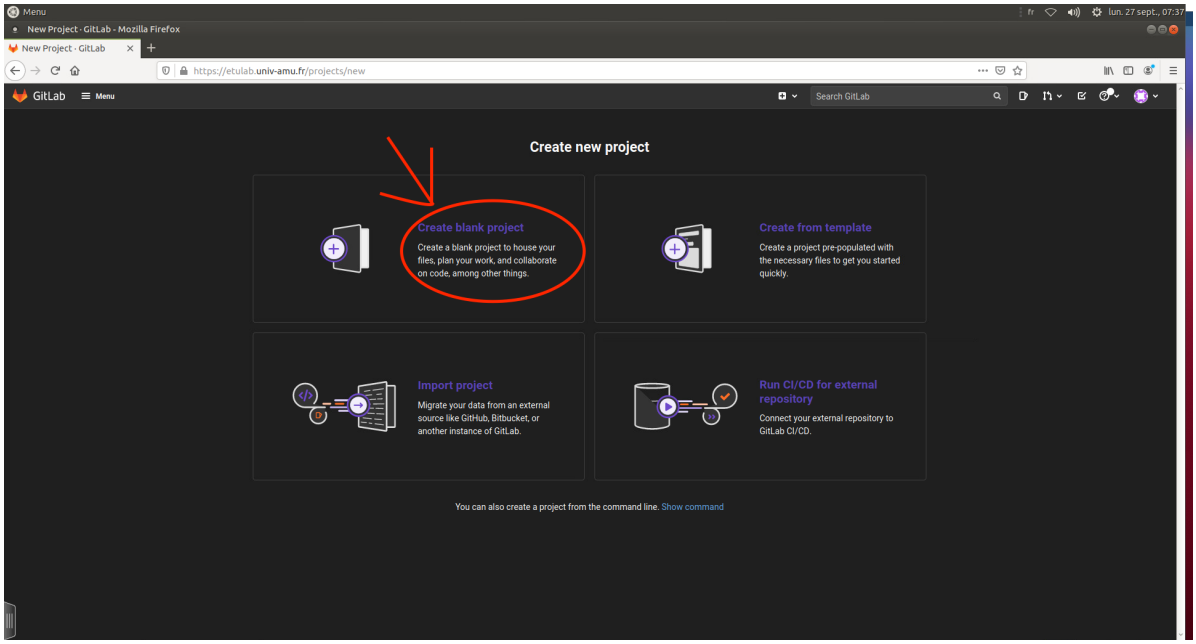
— Il vous faudra ajouter tous les fichiers `Java` ainsi que les fichiers de configuration de gradle : `build.gradle`, `gradlew`, `gradlew.bat` et `settings.gradle`.

— Exécutez la commande `git commit -m"premier message"` pour faire un premier `commit` de votre projet avec votre propre message.

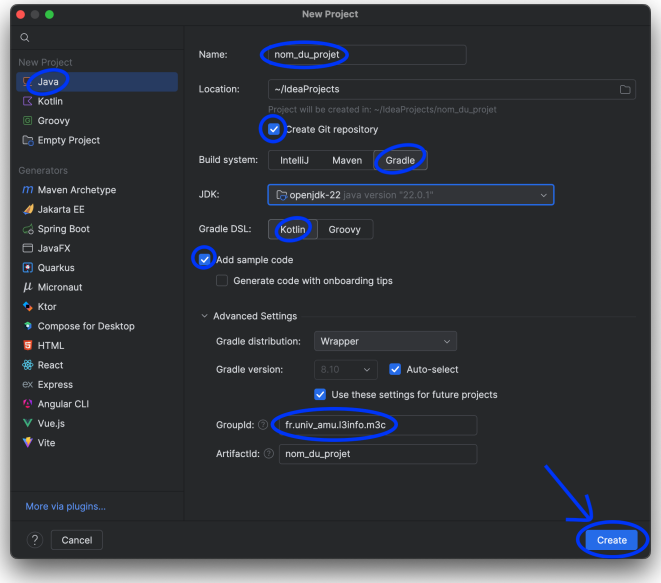
— Connectez vous via à un navigateur à votre compte etulab et créez un nouveau projet en cliquant sur `menu` puis `Create new project`.



— 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
```

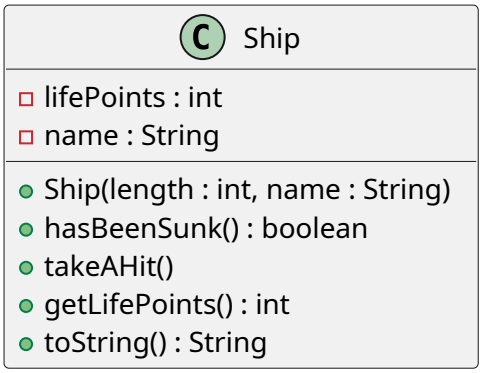
— Faites le premier `push` à l'aide de la commande `git push -u origin master`. Pour les `push` suivants, vous pourrez simplement utiliser la commande `git push` car vous avez configuré les branches.

La classe `Ship`

Spécification de la classe `Ship`

La classe `Ship` permet de représenter les bateaux dans le jeu de bataille navale. 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 si ce nombre était strictement positif. Un bateau est coulé quand son nombre de points de vie arrive à 0 (méthode `hasBeenSunk()`).

Voici le diagramme de cette classe :



Tâches

Tâche 1 : Créez la classe `Ship` dans le répertoire `src/main/java` de votre projet qui répond aux spécifications ci-dessus. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

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

l'enum `ShotResult`

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

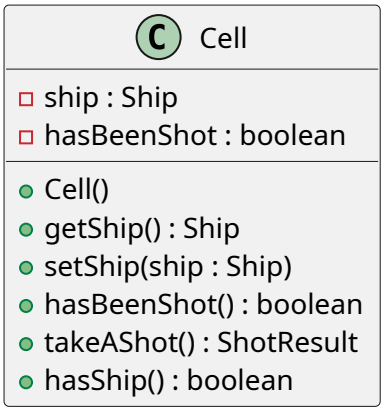
Tâches

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

La classe `Cell`

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) grâce à la méthode `hasBeenShot()`.



Tâches

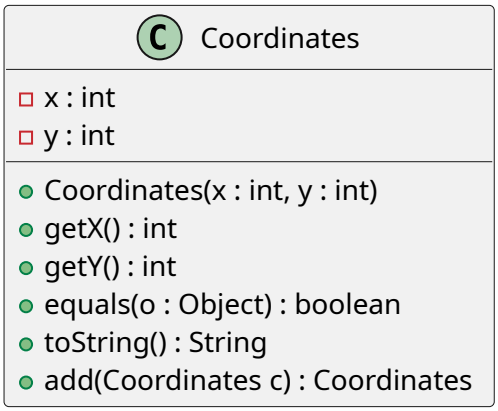
Tâche 4 : Créez la classe `Cell` dans le répertoire `src/main/java` de votre projet qui répond aux spécifications ci-dessus. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

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`.

La classe `Coordinates`

Spécification de la classe `Coordinates`

Pour représenter les coordonnées des cases du plateau de jeu, on définit la classe `Coordinates`. Voici son diagramme de la classe :



Détails du comportement des méthodes :

— La méthode `toString` renvoie une chaîne de caractère au format suivant : `(X, Y)` avec `X` la coordonnée en x et `Y` la coordonnée en y.

— La méthode `equals` teste si l'objet passé en argument correspond à des coordonnées égales aux coordonnées courantes. Deux coordonnées sont considérées égales si elles ont les mêmes coordonnées en x et en y.

— La méthode `add` renvoie de nouvelles coordonnées issues de la somme des coordonnées passées en argument avec les coordonnées courantes.

Tâches

Tâche 6 : Créez la classe `Coordinates` dans le répertoire `src/main/java` de votre projet qui répond aux spécifications ci-dessus. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

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`.

La classe `Position`

Spécification de la classe `Position`

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

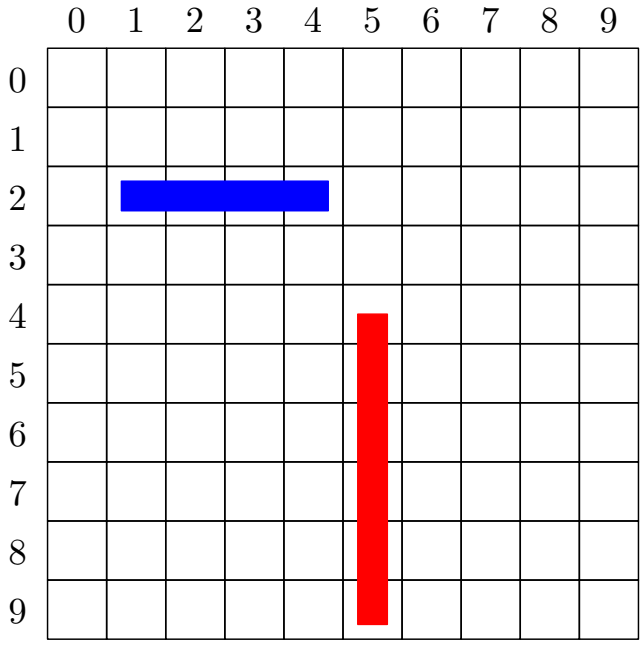
Une position a trois attributs :

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

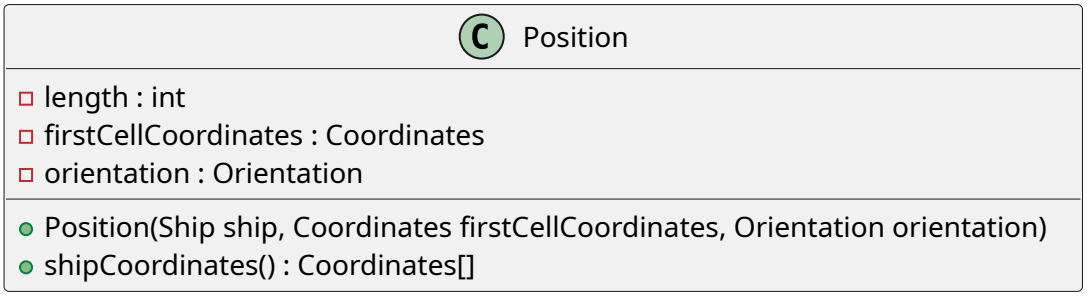
position2
length = 6
orientation = VERTICAL
firstCell = (5, 4)



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



Tâches

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

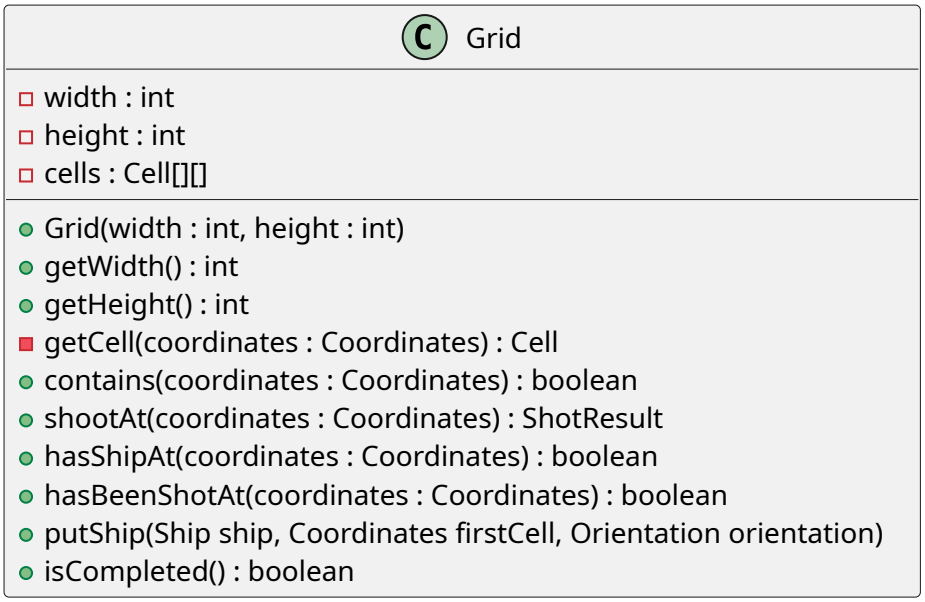
Tâche 9 : Créez la classe `Position` dans le répertoire `src/main/java` de votre projet qui répond aux spécifications ci-dessus. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

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`.

La classe `Grid`

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`.



— Le constructeur de la classe devra construire un tableau avec les tailles données en argument et le remplir avec des cases vides de type `Cell` (et donc sans bateau).
— La méthode `contains` renvoie `true` si les coordonnées passées en arguments sont valides dans la grille (c'est-à-dire ne dépasse pas de la taille de la grille) et `false` sinon.

— 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 aux coordonnées `coordinates` sur le plateau de jeu.
— La méthode `hasShip` renvoie `true` si la case correspondant aux coordonnées spécifiées contient un bateau et `false` sinon.

— La méthode `hasBeenShotAt` renvoie `true` si la case correspondant aux coordonnées spécifiées a déjà subi un tir et `false` sinon.
— La méthode `put` place un bateau sur les cases correspondant à la position spécifiée.
— La méthode `isCompleted` renvoie `true` s'il y a eu un tir sur chacune des cases contenant un bateau, et `false` sinon.

Tâches

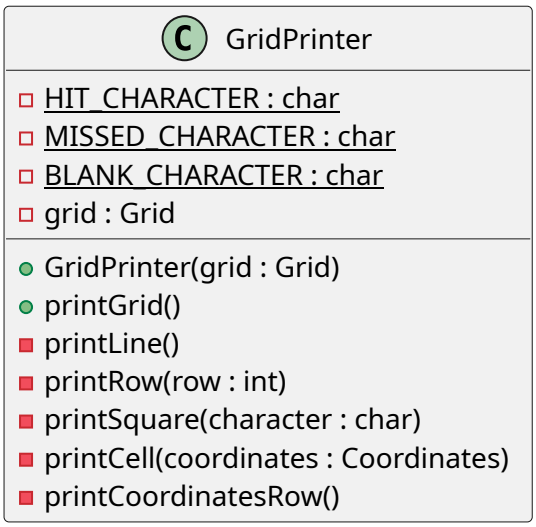
Tâche 11 : Créez la classe `Grid` dans le répertoire `src/main/java` de votre projet. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

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`.

La classe `GridPrinter`

Spécification de la classe `GridPrinter`

Afin de pouvoir jouer à la bataille navale, il est nécessaire de pouvoir afficher la grille de jeu. À cette fin, nous allons définir une classe permettant d'afficher une grille. Il pourrait paraître naturel de coder les fonctionnalités d'affichage directement dans la classe `Grid` mais il est en fait judicieux de séparer la fonctionnalité d'affichage du code de base de `Grid`. En effet, l'affichage peut être modifié (pour passer par exemple d'un affichage via le terminal à un affichage graphique utilisant JavaFX) sans modifier les fonctionnalités de base de `Grid` que l'on appelle généralement le modèle. Il est donc profitable de séparer ce deux fonctionnalités : l'affichage et le modèle afin de rendre chacune plus facile à modifier.



- Les constantes `HIT_CHARACTER`, `MISSED_CHARACTER` et `BLANK_CHARACTER` correspondent aux caractères à afficher dans les cases en fonction de la situation. Elles sont égales respectivement à `'H'` (tir sur un bateau), `'M'` (tir sans bateau) et `' '` (caractère espace tous les autres cas).
- La méthode `printLine` affiche une ligne du tableau. Par exemple, pour une grille de largeur 3, `printLine` devra afficher `+---+---+---+---+`.
- La méthode `printCoordinatesRow` affiche la ligne des coordonnées des colonnes. Par exemple, pour une grille de largeur 3, `printCoordinatesRow` devra afficher `|   | A | B | C |`.
- La méthode `printRow` affiche la rangée correspondante de la grille. Par exemple, pour la grille considéré ci-dessus, un appel à `printRow(0)` devra afficher `| 0 | H |   |   |`.
- La méthode `printSquare` affiche une case du tableau contenant le caractère spécifié. Par exemple, un appel à `printSquare('A')` doit afficher la chaîne de caractère `" A |"`.
- La méthode `printCell` affiche la case de la grille correspondant aux coordonnées spécifiées. Le caractère affiché dépend de la situation de la case :
  - `'H'` s'il y a eu un tir et qu'un bateau est présent ;
  - `'M'` s'il y a eu un tir et qu'il n'y a pas de bateau ;
  - `' '` dans les autres cas. Par exemple, un appel à `printCell(new Coordinates(2,2))` sur la grille considérée ci-dessus doit afficher la chaîne de caractères `" A |"`.

— La méthode `printGrid` affiche la grille spécifiée. Par exemple, une grille de taille 3×3 devra s'afficher de la manière suivante (avec `H` pour indiquer un tir ayant touché et `M` un tir ayant manqué) :

```
+---+---+---+---+
|   | A | B | C |
+---+---+---+---+
| 0 | H |   |   |
+---+---+---+---+
| 1 |   |   |   |
+---+---+---+---+
| 2 |   |   | M |
+---+---+---+---+
```

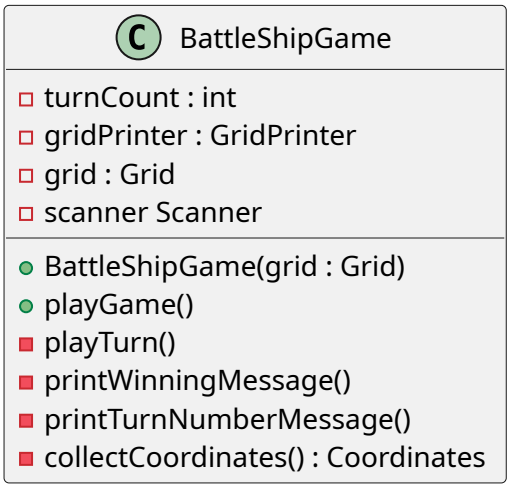
Tâches

Tâche 13 : Créez la classe `GridPrinter` dans le répertoire `src/main/java` de votre projet. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

La classe `BattleShipGame`

Spécification de la classe `BattleShipGame`

La classe représentant l'état du jeu de la bataille navale s'appelle `BattleShipGame`. C'est cette classe qui gérera les tours de jeu et l'interaction avec l'utilisateur. Ici, on considérera une version très simplifiée du jeu dans lequel il n'y a qu'une grille et le but du joueur est de couler tous les bateaux en le minimum de coups.



Description des éléments de la classe (un exemple de partie est disponible à la fin de la planche de TP):

— l'attribut `turnCount` : le nombre de tours de jeu ;
— l'attribut `gridPrinter` : l'afficheur de la grille ;
— l'attribut `grid` : la grille du jeu ;

— l'attribut `scanner` : le lecteur pour l'entrée standard, à initialiser avec `new Scanner(System.in)` (Javadoc de Scanner);
— le constructeur `BattleShipGame(grid : Grid)` qui initialise tous les attributs à partir de la grille spécifiée ;
— la méthode `playGame` : fait jouer une partie, c'est-à-dire fait jouer des tours jusqu'à que la grille soit complétée ;
— la méthode `playTurn` : fait jouer un tour, affiche le nombre de tours, collecte les coordonnées entrées par le joueur, puis affiche un message

— la méthode `printTurnNumberMessage` affiche le nombre de tours ;
— la méthode `printShotResultMessage` affiche le résultat du tir (`You sunk a ship.`, `You hit a ship.` ou `You missed.`) ;
— la méthode `collectCoordinates` affiche un message demandant les coordonnées à l'utilisateur puis récupère celle-ci. On utilisera le format `B3` (une lettre de `A` à `J` suivi d'un chiffre de `0` à `9`) qu'on pourra récupérer avec `scanner.next("\\p{Upper}\\d")`, cela nous donnera une chaîne de 2 caractères à partir de laquelle on déduira des coordonnées ;
— la méthode `printWinningMessage` affiche le message de victoire.

Tâches

Tâche 14 : Créez la classe `BattleShipGame` dans le répertoire `src/main/java` de votre projet. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

La classe `BattleShipApp`

Spécification de la classe `BattleShipApp`

Cette classe sera le point d'entrée de votre programme. Elle contiendra uniquement une méthode `public static void main(String[] args)` qui créera une grille de type `Grid`, placera un bateau de type `Ship` dans celle-ci, créera une instance du jeu de type `BattleShipGame` et lancera une partie.

Tâches

Tâche 15 : Créez une classe `BattleShipApp` dans le répertoire `src/main/java` de votre projet. Votre code devra donner la Javadoc pour toutes les méthodes publiques de la classe.

Tâche 16 : Donnez le diagramme de toutes les classes du projet en indiquant les types de relations entre les classes et leur multiplicité. Il n'est pas nécessaire de redonner les attributs et les méthodes des classes.

Tâche 17 (optionnelle) : Améliorez le jeu de la façon suivante :

— coder un générateur aléatoire de grille qui place au hasard des bateaux dans les grilles
— coder un véritable jeu de bataille navale contre l'ordinateur. Il vous faut rajouter :

— le fait d'avoir deux grilles dans le jeu : la vôtre et celle de l'ordinateur ;
— des affichages différencier entre les deux grilles (bateaux visibles pour votre grille) ;
— la possibilité de placer vous-même les bateaux au départ dans votre grille ;
— un générateur aléatoire de grille pour l'ordinateur ;
— un générateur de coups pour faire jouer l'ordinateur (on pourra imaginer le faire jouer purement au hasard) ;
— une méthode pour décider du gagnant.

Exemple de partie

```
+---+---+---+---+---+---+
|   | A | B | C | D | E |
+---+---+---+---+---+---+
| 0 |   |   |   |   |   |
+---+---+---+---+---+---+
| 1 |   |   |   |   |   |
+---+---+---+---+---+---+
| 2 |   |   |   |   |   |
+---+---+---+---+---+---+
Turn number 0
Enter coordinates
A1
You missed.
+---+---+---+---+---+---+
|   | A | B | C | D | E |
+---+---+---+---+---+---+
| 0 |   |   |   |   |   |
+---+---+---+---+---+---+
| 1 | M |   |   |   |   |
+---+---+---+---+---+---+
| 2 |   |   |   |   |   |
+---+---+---+---+---+---+
Turn number 1
Enter coordinates
A0
You hit a ship.
+---+---+---+---+---+---+
|   | A | B | C | D | E |
+---+---+---+---+---+---+
| 0 | H |   |   |   |   |
+---+---+---+---+---+---+
| 1 | M |   |   |   |   |
```

```
+---+---+---+---+---+---+
| 2 |   |   |   |   |   |
+---+---+---+---+---+---+
Turn number 2
Enter coordinates
B0
You hit a ship.
+---+---+---+---+---+---+
|   | A | B | C | D | E |
+---+---+---+---+---+---+
| 0 | H | H |   |   |   |
+---+---+---+---+---+---+
| 1 | M |   |   |   |   |
+---+---+---+---+---+---+
| 2 |   |   |   |   |   |
+---+---+---+---+---+---+
Turn number 3
Enter coordinates
C0
You sunk a ship.
You have won in 4 turns.
```