Domaine Sciences et Technologies
Master Bio-informatique : DLAD

Aix*Marseille université
Socialement engagée

Année 2023-24

Programmation avancée : TP 2
Code UE : SBICU05

TP 2 : première classe en Java

Mise en place du TP

Introduction

L'objectif de ce TP est de découvrir la programmation orientée objet en Java. Nous verrons comment les classes et les objets nous permettent d'organiser un programme. Java est un langage de programmation, dit orienté objet car il est basé sur la notion d'objet.

Vous allez aussi vous initier (si ce n'est pas déjà fait) à l'utilisation d'un environnement de développement (IDE), c'est-à-dire un logiciel permettant d'éditer le programme en apportant de nombreuses fonctionnalités pratiques. L'IDE qu'on vous conseille d'utiliser est JetBrains est Java IntelliJ, propose des licences gratuites pour les étudiants, il suffit de vous créer un compte étudiant sur leur site, à l'adresse https://www.jetbrains.com/student/. Vous trouverez un document d'aide au lien suivant : Document prise en main d'IDE IntelliJ

Ce TP est l'occasion d'apprendre à utiliser des outils de développement professionnel. En plus de l'IDE IntelliJ, vous aurez à utiliser un outil de gestion de versions, en l'occurrence `git`. Un tel outil permet d'enregistrer à distance votre projet et de permettre à plusieurs personnes de travailler simultanément, sans devoir échanger les fichiers par courriel, et sans se marcher sur les pieds (par exemple modification sans le savoir du même fichier).

Si vous effectuez 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.

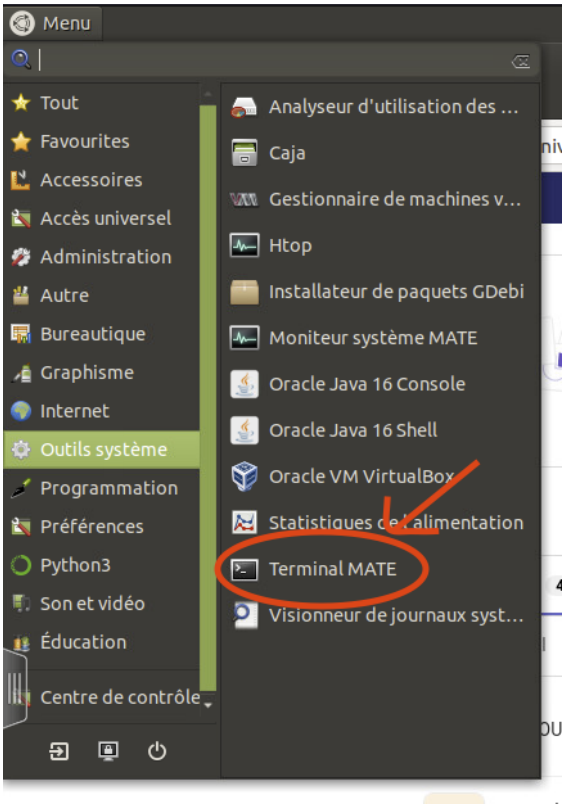
Configuration clé ssh pour le gitlab AMU

La première étape pour utiliser la gestion de version est de vous connecter au gitlab de l'université qui est accessible à l'adresse suivante : https://etulab.univ-amu.fr/. L'identifiant et le mot de passe sont ceux de votre

compte étudiant AMU.

Afin de pouvoir accéder à distance à vos dépôts git, vous allez configurer une clé SSH dans votre profil. Pour cela, il vous faut :

1. Ouvrir un terminal MATE qui est accessible dans le menu au sous-menu *Outils systèmes*.



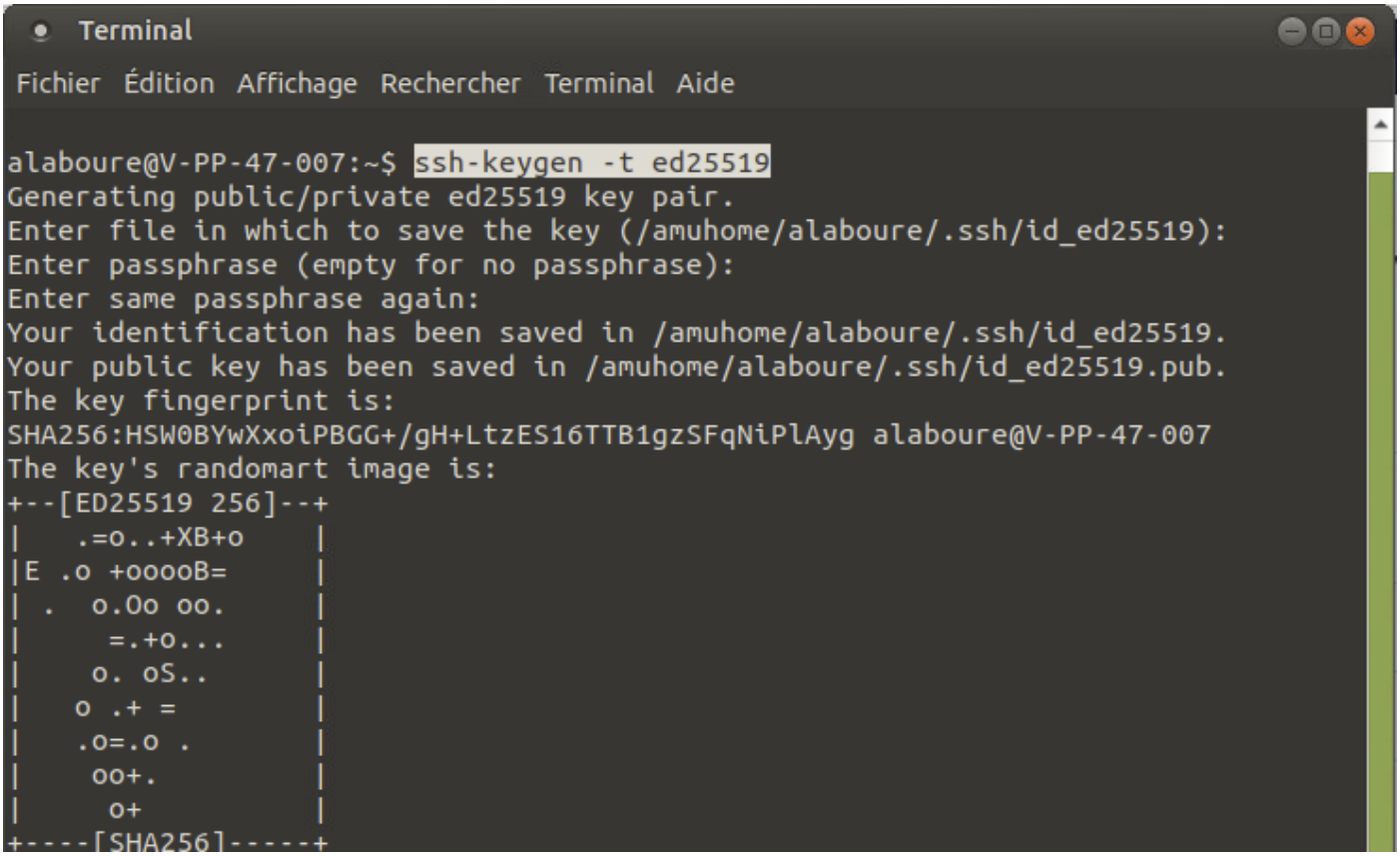
2. Générer une paire de clés privé/publique pour votre compte. Pour cela, il vous faut entrer la commande suivante dans le terminal :

```
ssh-keygen -t ed25519
```

Appuyer une fois sur Entrée pour confirmer que vous voulez sauvegarder vos clés dans le répertoire par défaut.

Ensuite, il vous est demandé de rentrer deux fois une “passphrase”. Vous pouvez entrer une phrase (plusieurs mots donc) qui servira de mot de passe pour l’accès. Puisque la sécurité de vos dépôts n’est pas critique, vous pouvez vous contenter de ne rien rentrer (pas de *passphrase*) et donc d’appuyer de nouveau sur Entrée deux fois.

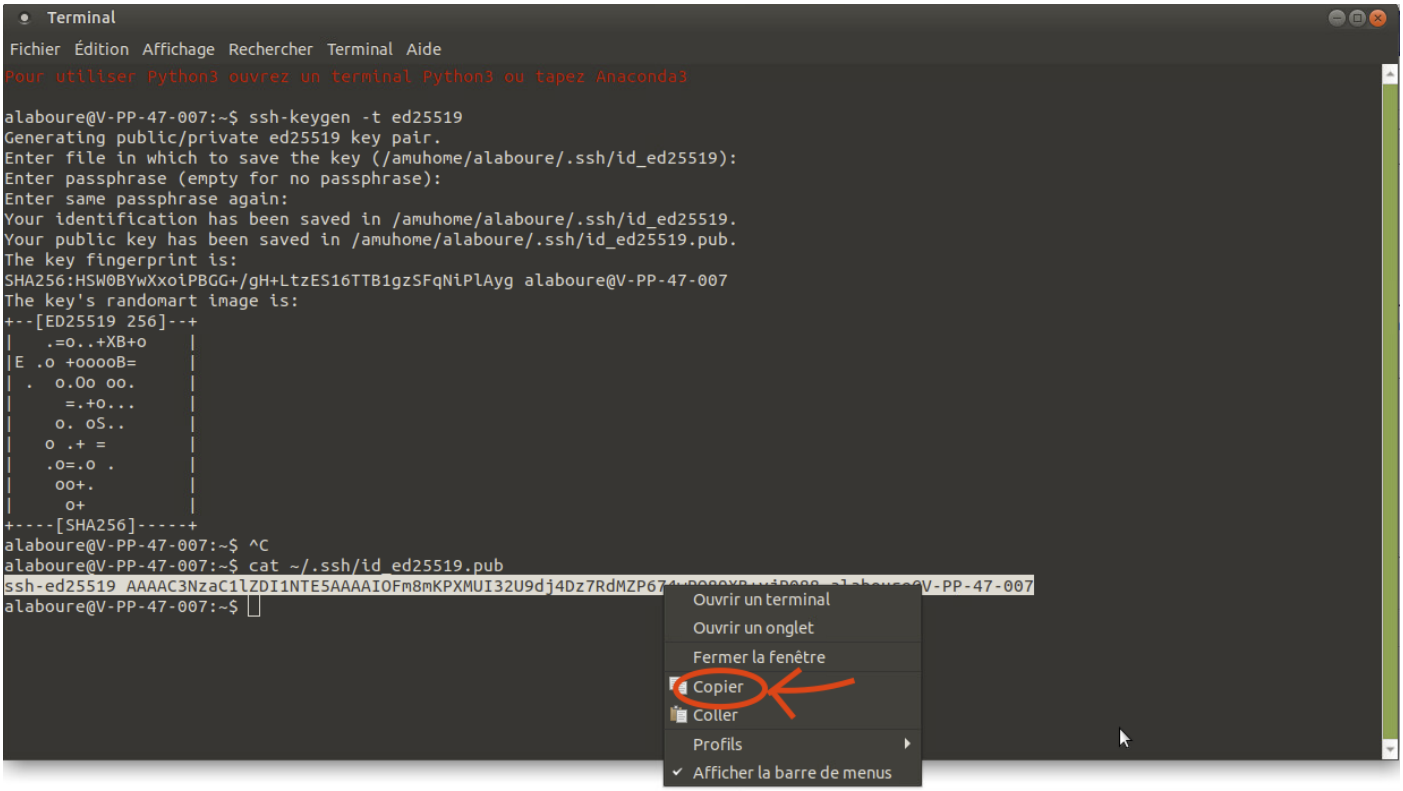
Si tout s’est bien passé, votre couple de clés a été généré et vous devriez voir un affichage similaire à celui ci-dessous :



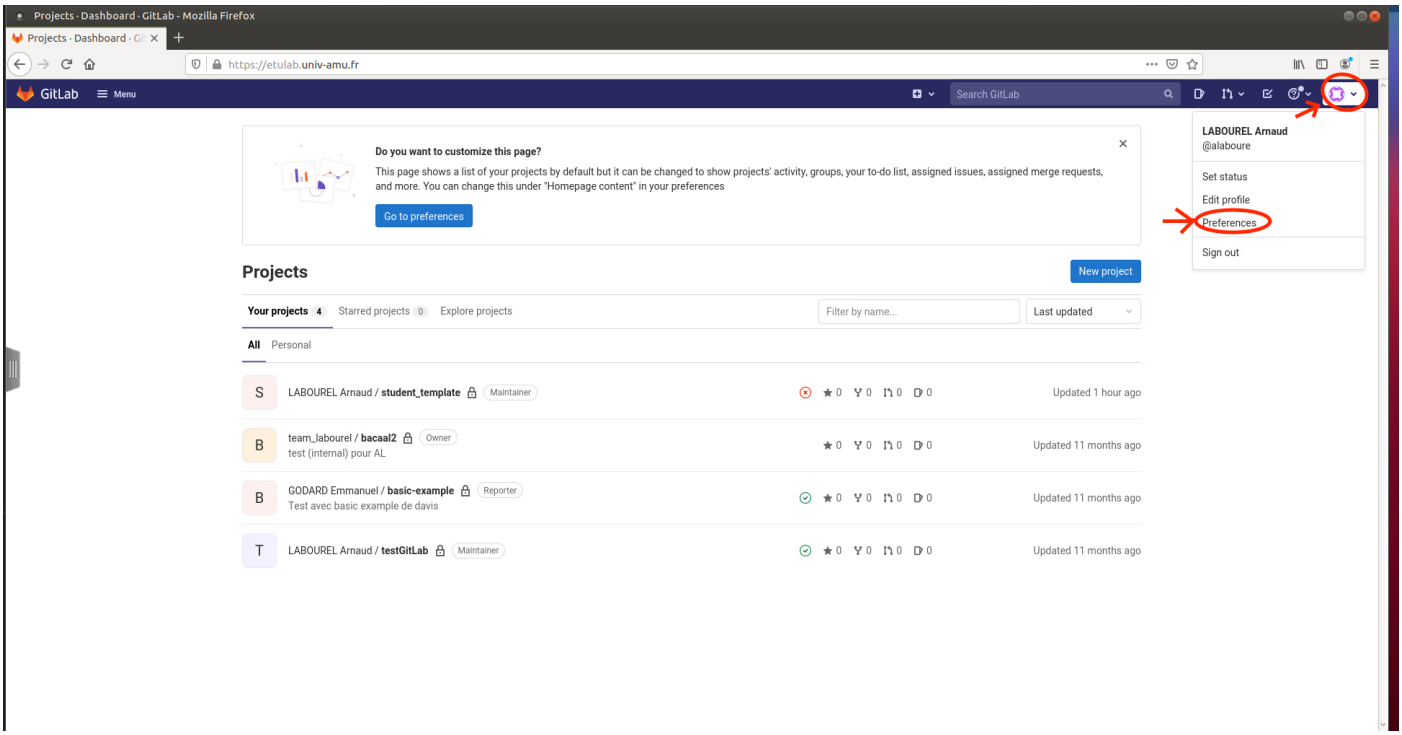
Afficher votre clé dans le terminal en entrant la commande suivante :

```
1  cat ~/.ssh/id_ed25519.pub
```

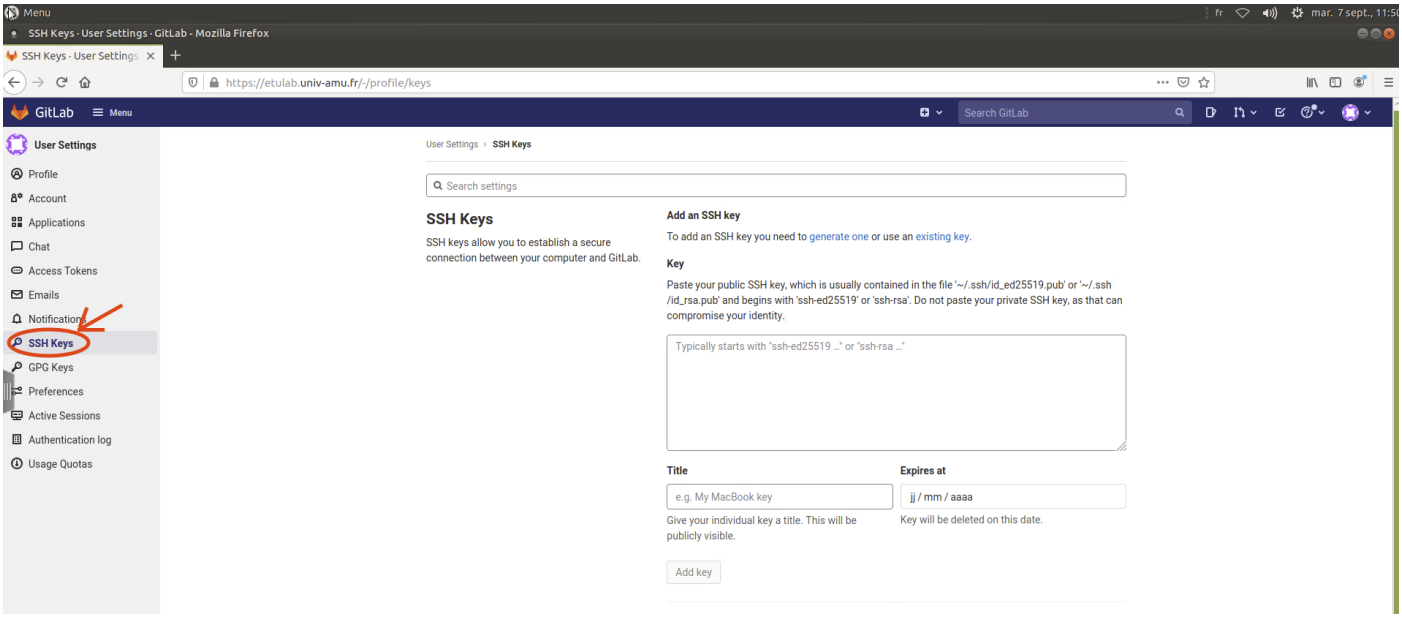
Sélectionner la ligne affichée par le terminal et copier là dans le presse-papier (sélection puis clic droit et copier)



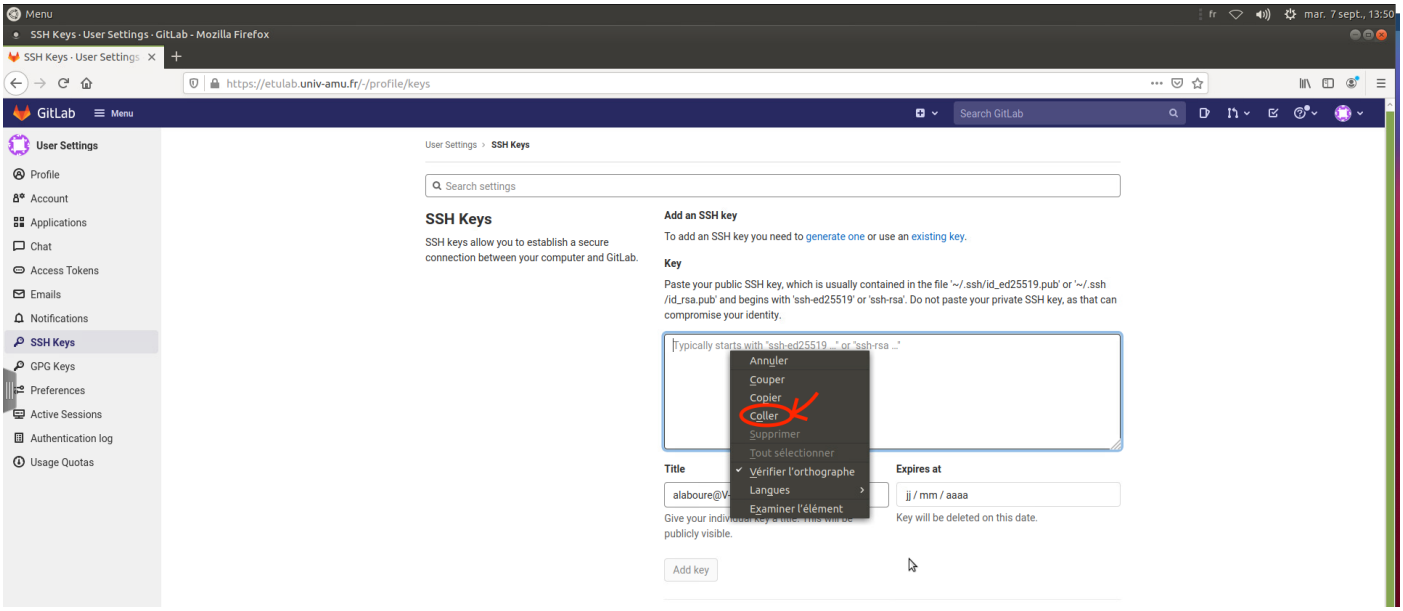
3. Configurer la clé dans votre compte *gitlab*. Pour cela, vous allez accéder aux préférences de votre compte *gitlab*. Il vous faut vous connecter à https://etulab.univ-amu.fr/ puis cliquez sur votre avatar en haut à gauche de l'écran puis sur préférences dans le menu qui apparaît.



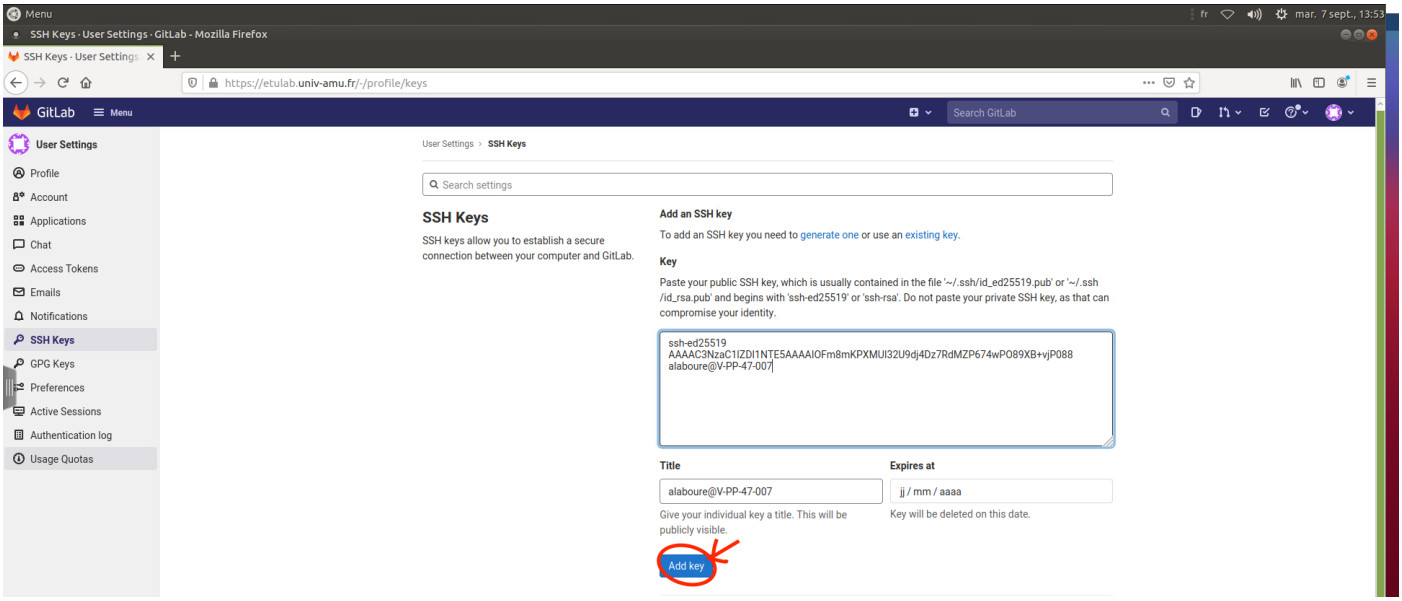
Allez au menu des clés SSH en cliquant sur le lien correspondant dans le menu de gauche.



4. Revenez sur la page de configuration des clés `ssh` sur votre navigateur. Coller votre clé (clic droit puis coller) dans l'espace prévu pour cela.



5. Ajouter la clé en cliquant sur le bouton *add*.



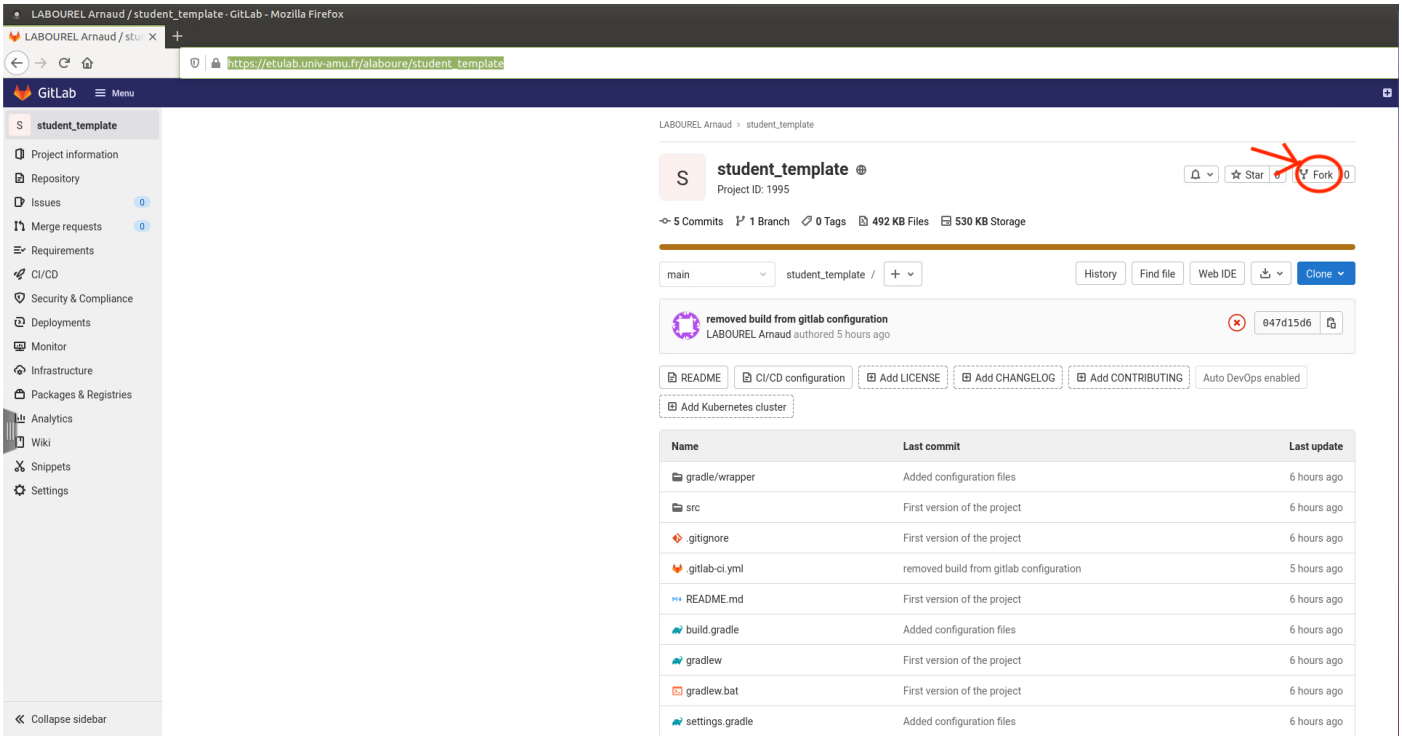
Vous devriez recevoir un mail sur votre mail étudiant confirmant que vous avez rajouté une clé. Vous pouvez rajouter autant de clés que vous voulez. Vous pouvez donc faire de même pour rajouter par exemple des clés supplémentaires si vous souhaitez accéder à votre dépôt depuis chez vous.

Fork d'un projet

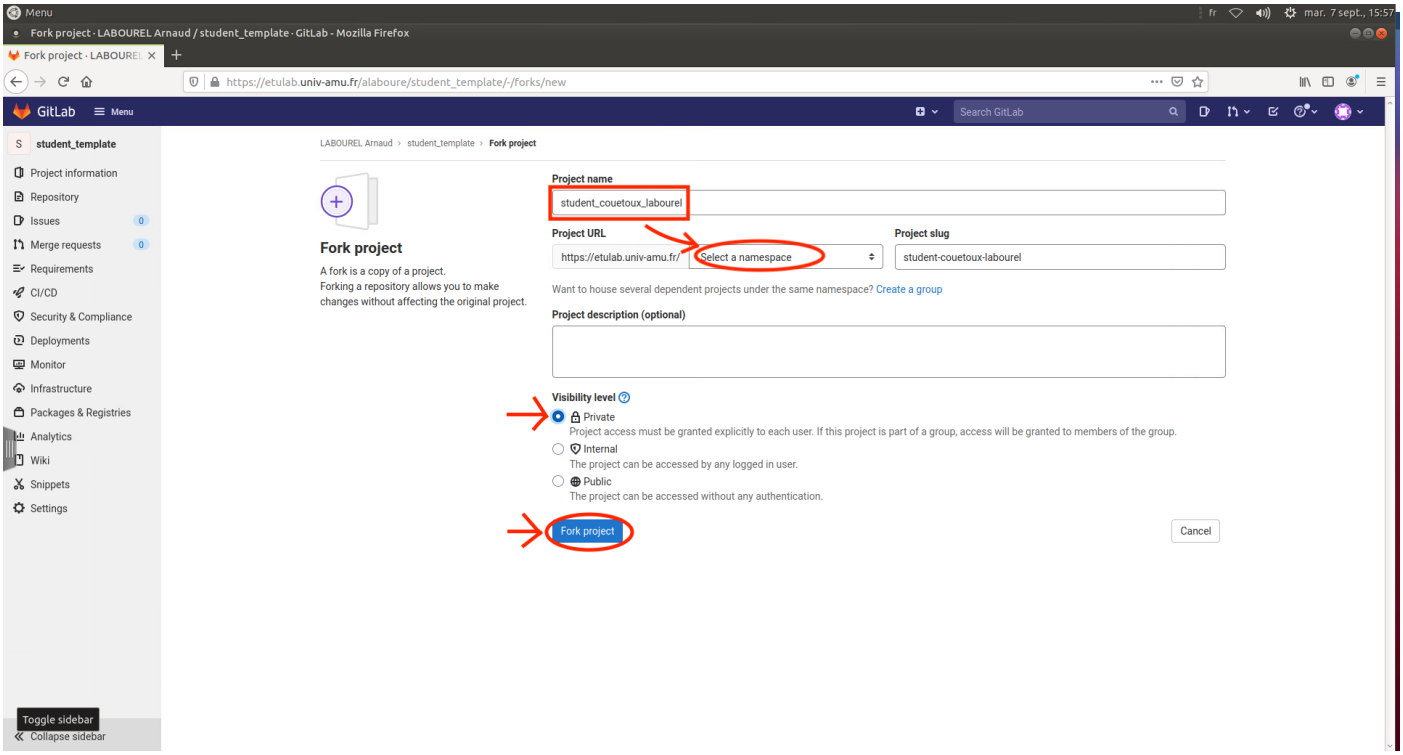
Vous allez maintenant créer votre projet en utilisant un projet déjà existant. Pour cela, il faut :

1. Aller sur le projet `species-template` qui servira de base pour ce premier TP en accédant à l'adresse suivante : https://etulab.univ-amu.fr/alaboure/species-template

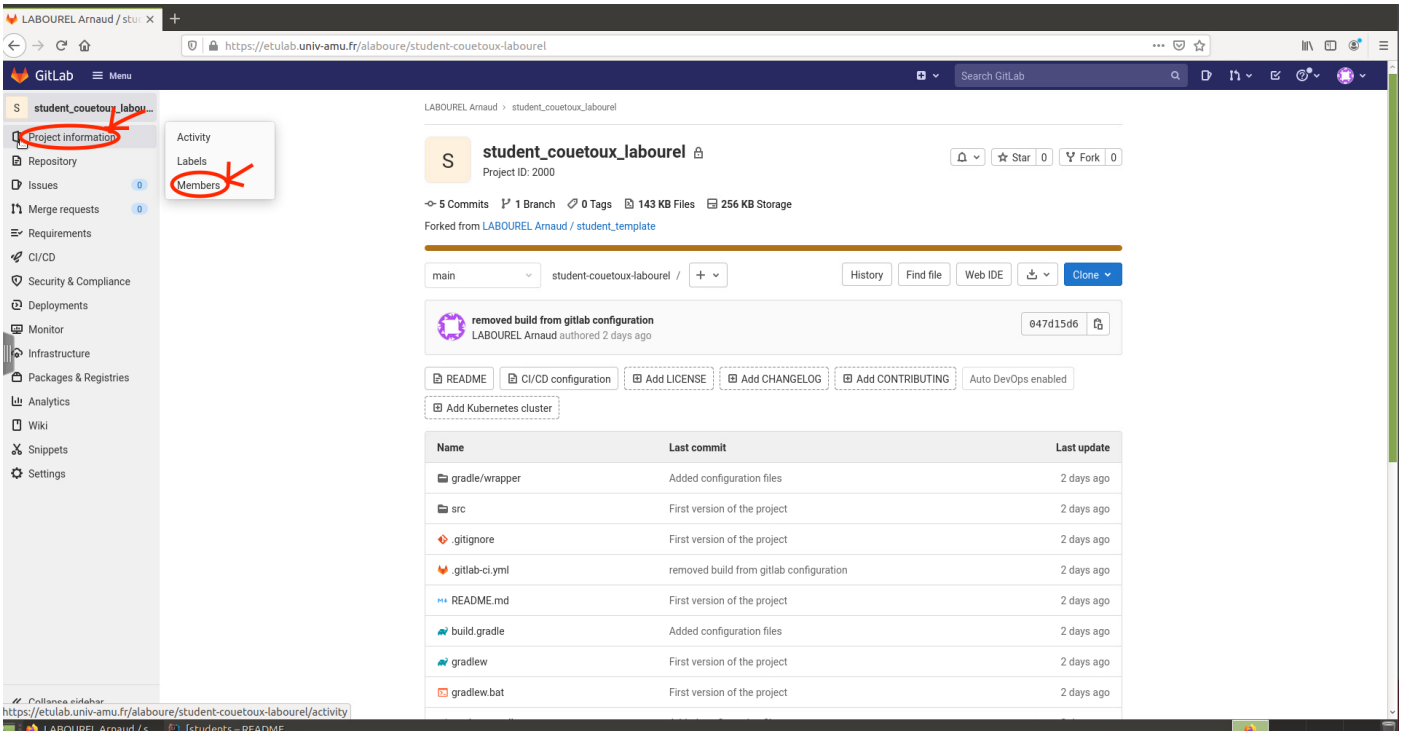
2. Cliquer sur le bouton *fork*.



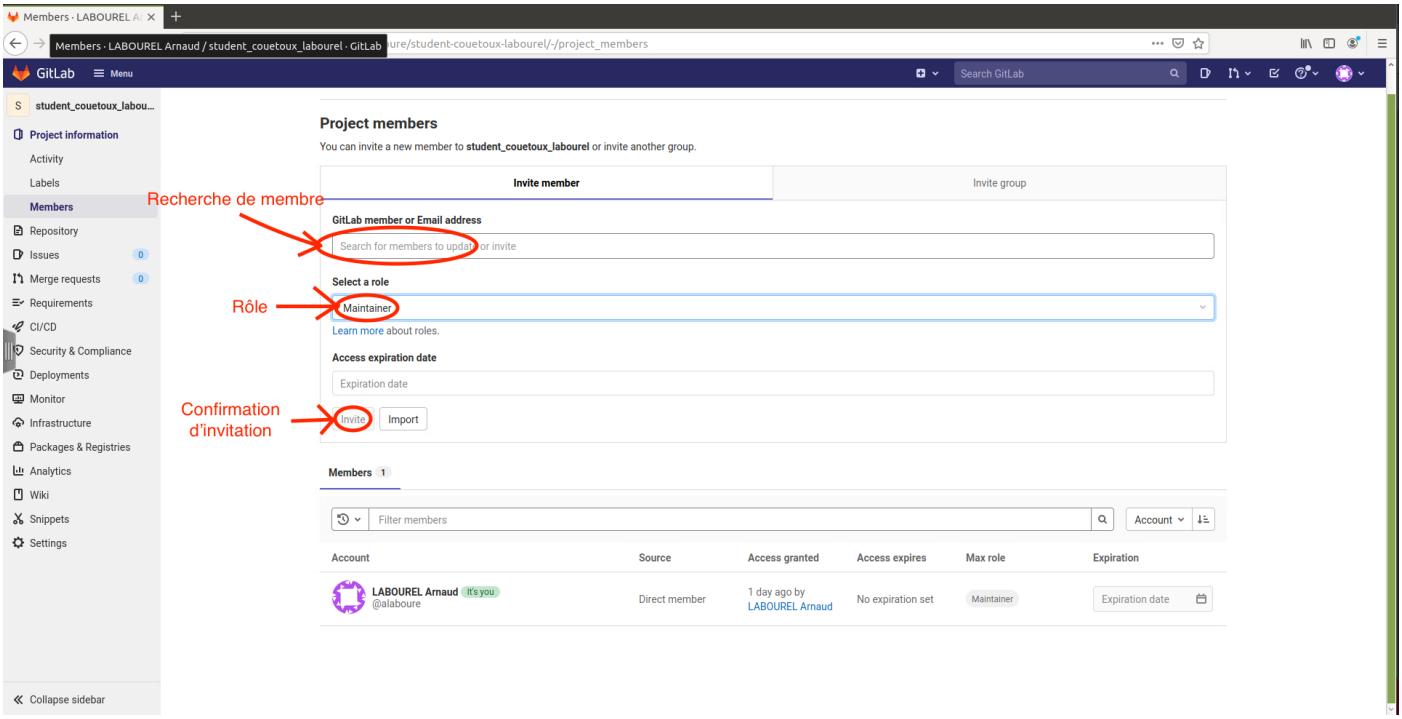
3. Changer le nom du projet en enlevant `template`. Sélectionner comme espace de nom (*space-name*) votre propre compte. Mettez la visibilité du projet en *private* afin que vos camarades en dehors du projet n'y aient pas accès et validez le fork en cliquant sur le bouton `fork project`.



4. Une fois le projet créé, vous pouvez rajouter des membres (par exemple la personne en charge de votre groupe de TP) en cliquant sur `project information` dans le menu de gauche puis `members`.



5. Ensuite vous pouvez rechercher une personne dans la barre dédiée. Une fois celle-ci trouvé vous pouvez lui donner un rôle (au moins `reporter` pour donner l'accès en lecture du code et `maintainer` pour l'accès en lecture et écriture), puis confirmer son invitation en cliquant sur le bouton `invite`.



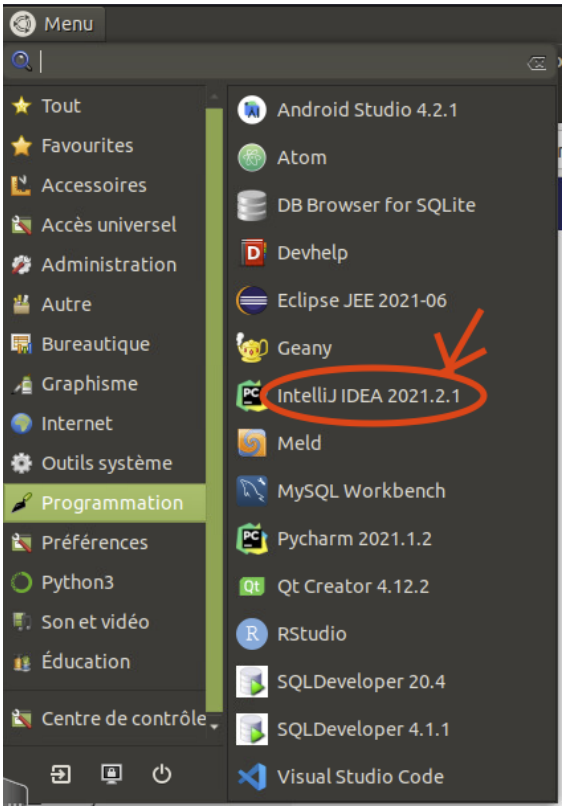
IDE IntelliJ

Environnement de développement (IDE)

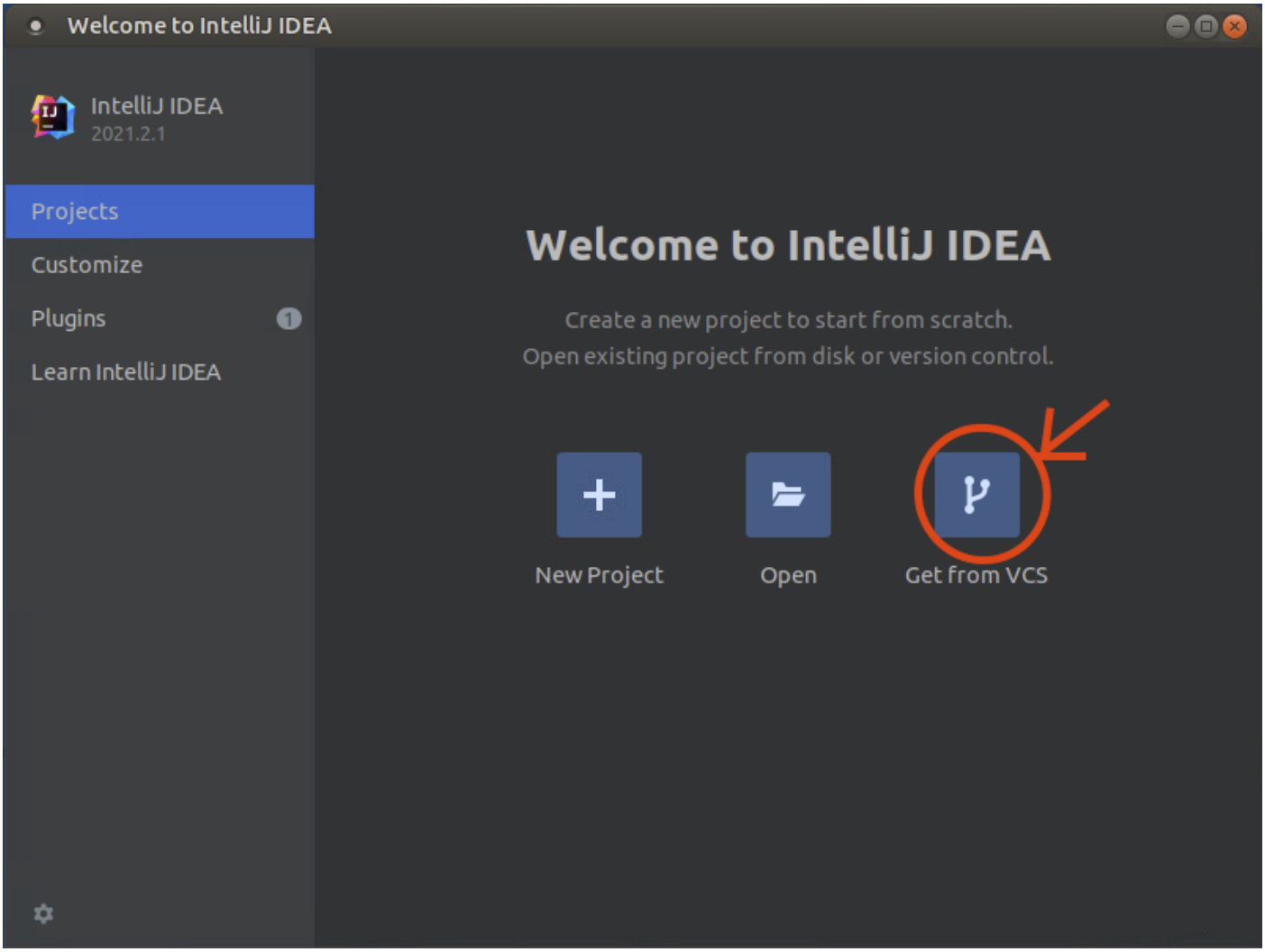
Afin de programmer dans ce cours, nous allons utiliser un environnement de développement (*Integrated Development Environment : IDE*). Il existe de nombreux *IDE* pour Java. Dans ce cours, nous vous conseillons d'utiliser IntelliJ IDEA de **JetBrains** mais vous pouvez aussi utiliser Eclipse *netbeans* ou un autre *IDE* si vous êtes familier avec ceux-ci.

Lancement de l'IDE

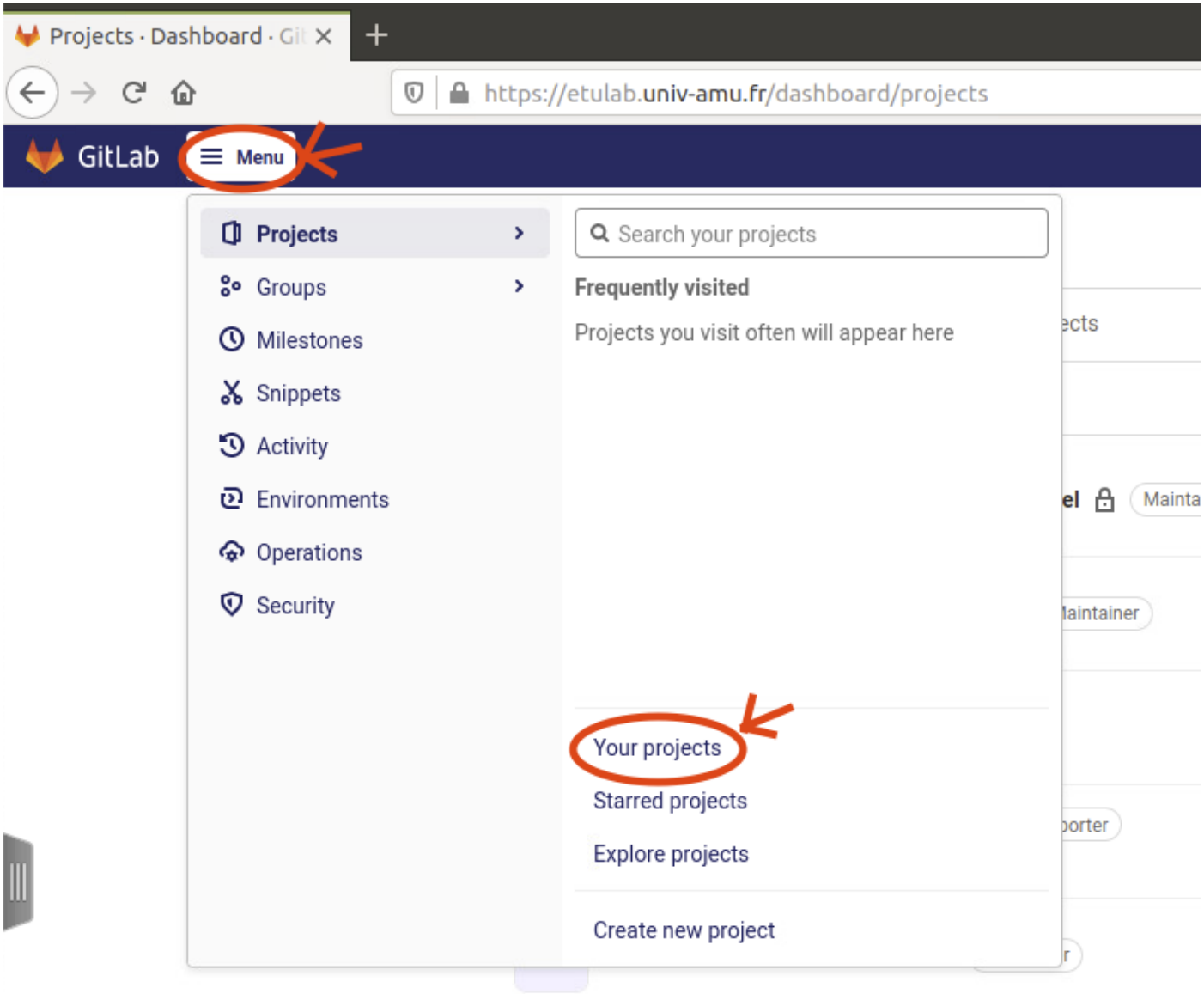
Pour lancer le logiciel sur les machines de l'université, il vous faut aller dans le Menu (en haut à gauche de l'écran) et cliquer sur **IntelliJ IDEA** dans la partie Programmation du menu.



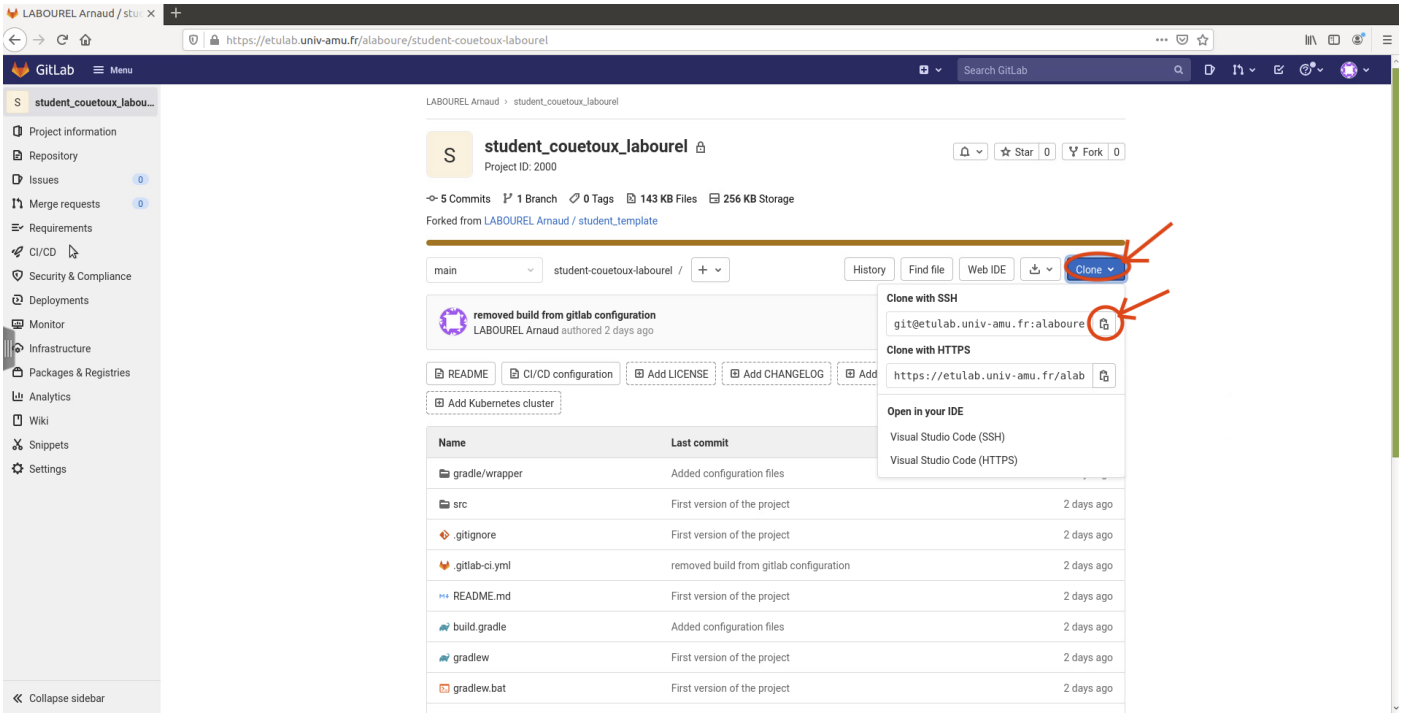
1. Vous devriez accéder à l'écran d'accueil d'IntelliJ IDEA. Vous allez récupérer le contenu de votre dépôt. Pour cela, cliquez sur `Get from VCS` puis sur `git` dans le menu qui apparaît.



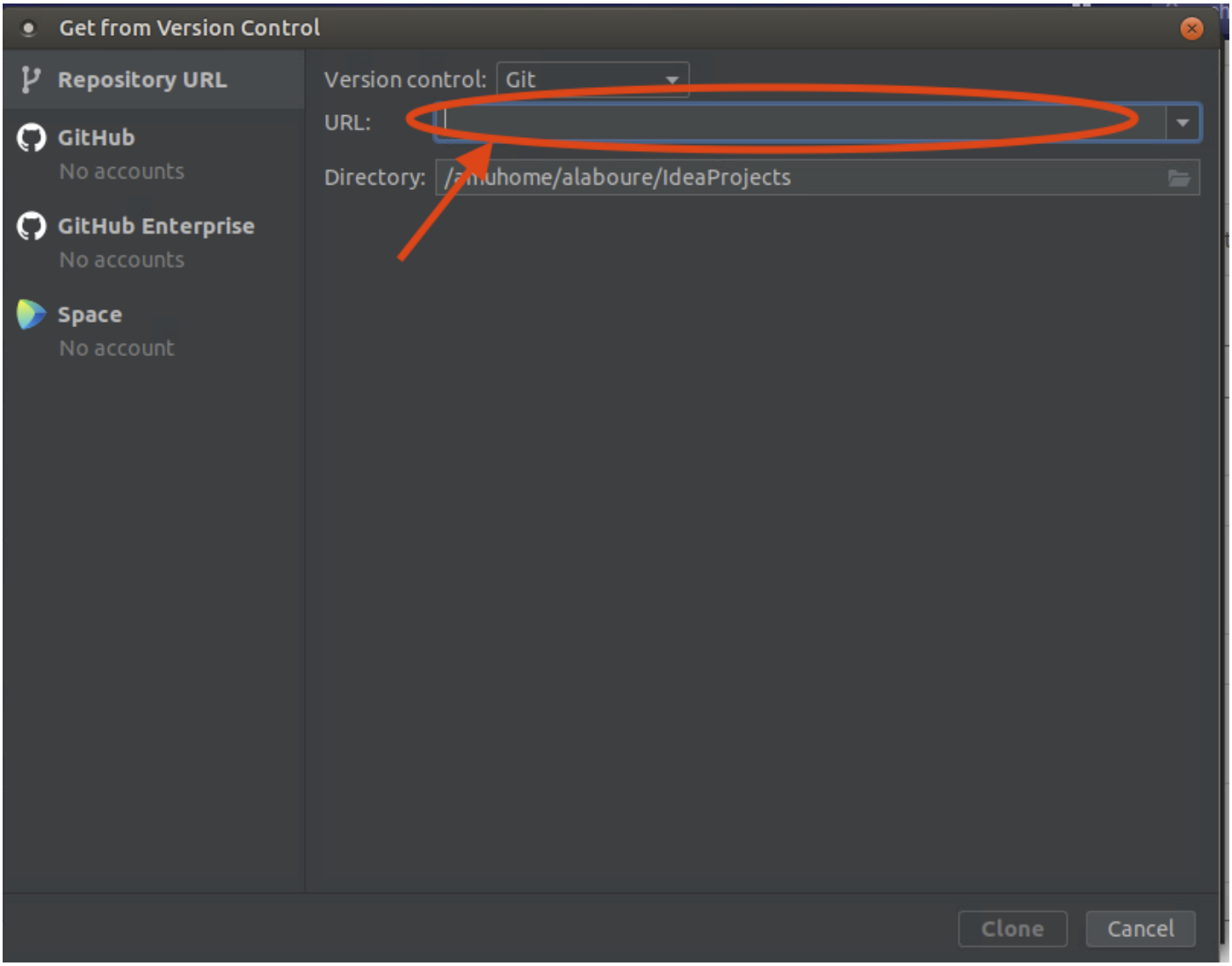
2. Vous allez maintenant récupérer votre dépôt (en fait le cloner, c'est-à-dire créer une copie du dépôt sur votre machine). Pour cela, connectez-vous à votre compte *etulab* dans un navigateur et accéder à votre projet. Vous pouvez accéder à la liste de vos projets.



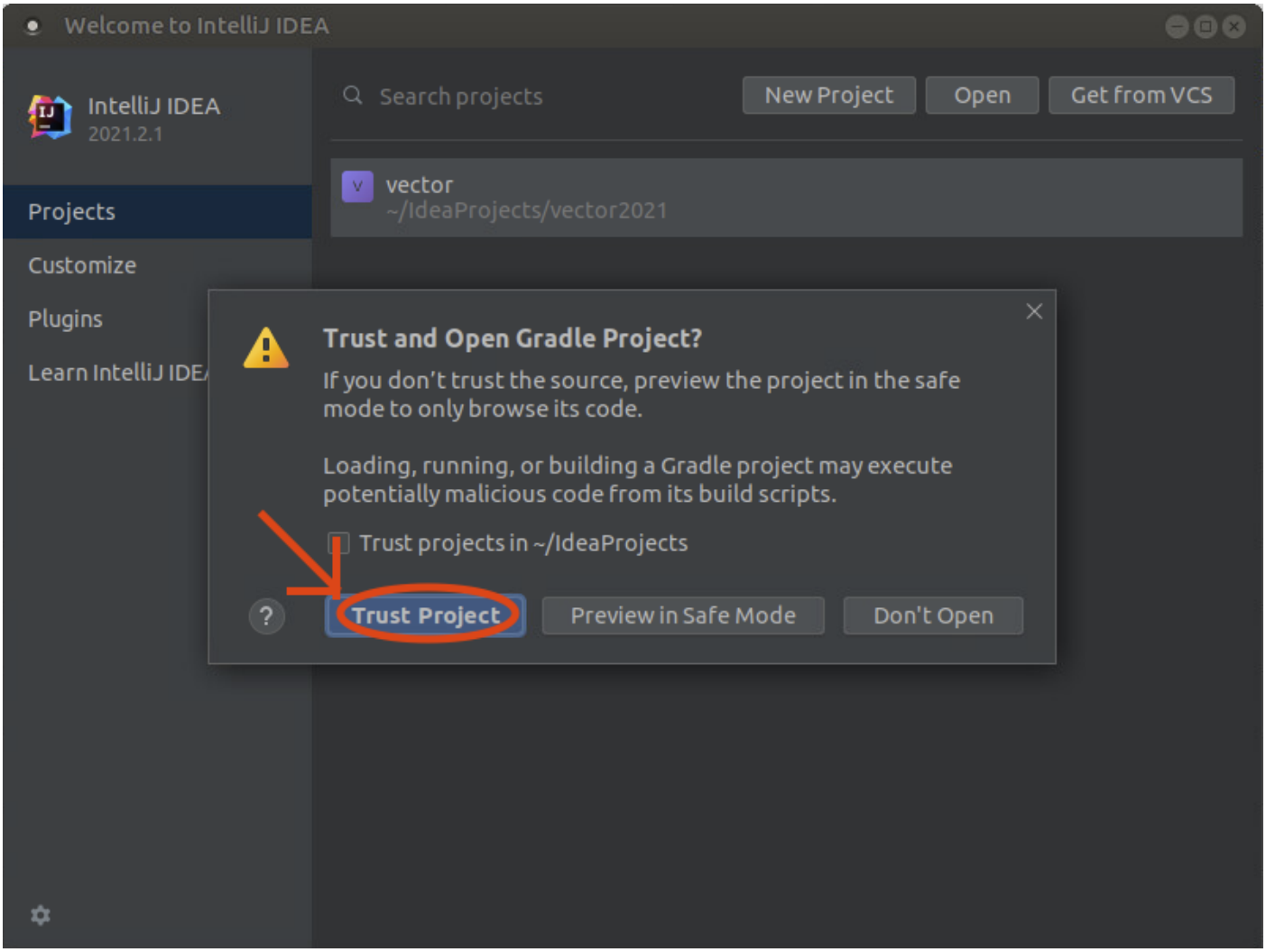
3. Copier l'adresse de votre dépôt sur la page web de votre dépôt en cliquant sur le bouton `clone` puis sur l'icône presse-papier.



4. Coller dans le champ URL d'IntelliJ, l'adresse de votre dépôt puis cliquez sur le bouton `clone`.



5. Après quelques instants de chargement, on vous demandera si vous faites confiance au projet. Cliquer `Trust Project` pour que le projet se lance.



Votre projet devrait maintenant être utilisable sur votre machine.

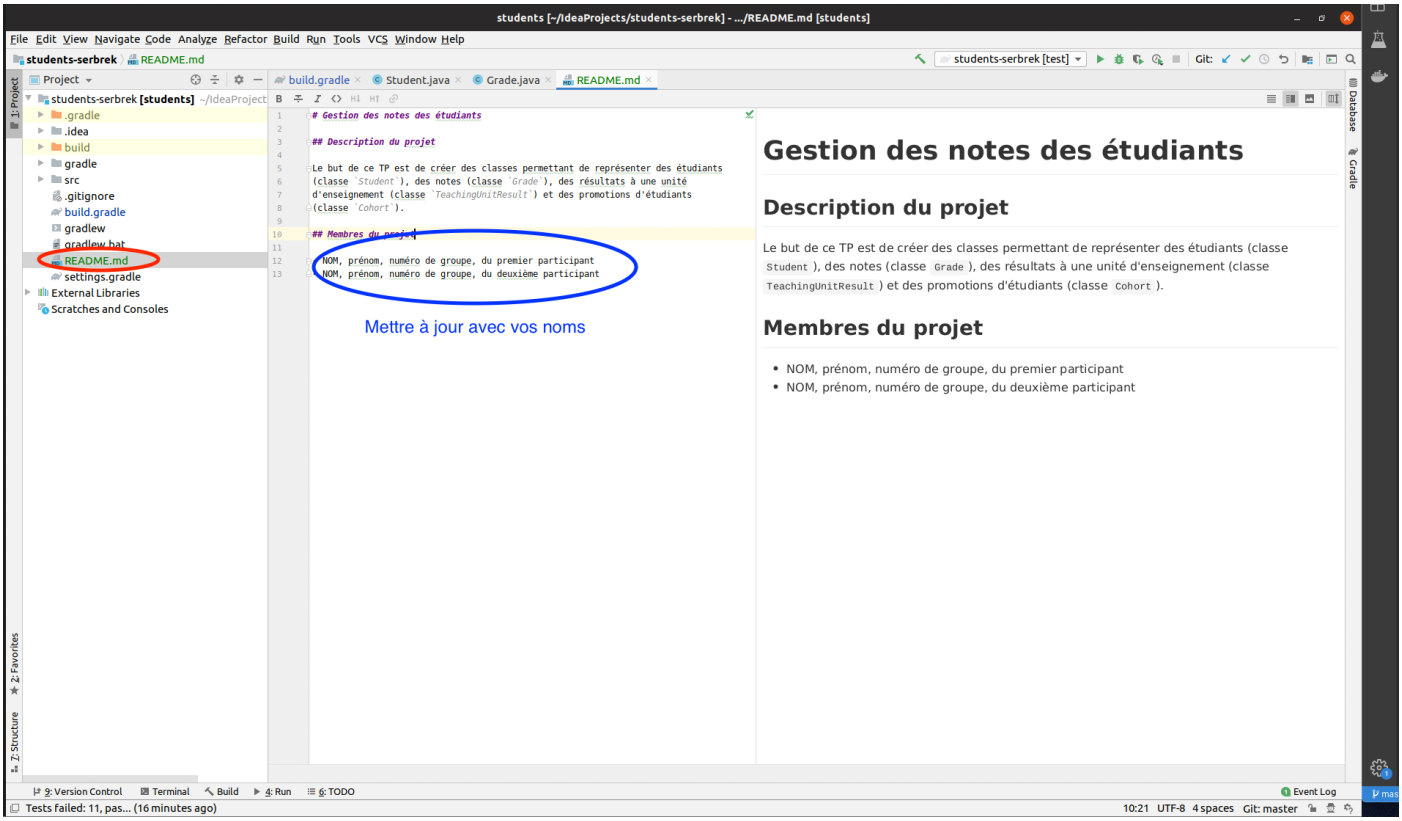
Git

Principe de git

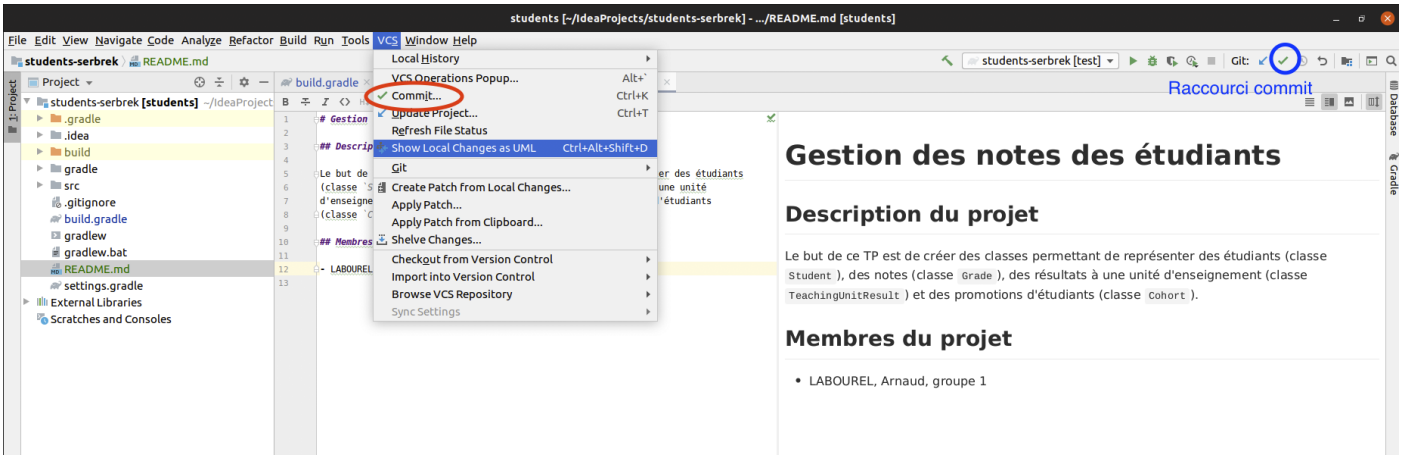
Le principe de `git` est d'avoir un *dépôt* distant : une version de votre projet stockée sur un serveur accessible par Internet (en l'occurrence hébergée par `github`). Vous disposez en plus de dépôts locaux sur les ordinateurs sur lesquels vous travaillez. Vous faites vos modifications sur votre ordinateur, et lorsque vous avez accompli une amélioration qui fonctionne bien, vous pouvez la faire enregistrer (`commit`) par `git`. Ces enregistrements sont locaux à votre ordinateur, et vous pouvez en faire autant que vous le souhaitez. Si vous voulez partager votre travail avec d'autres personnes (ou avec nous-même sur un autre ordinateur), il vous faut l'envoyer vers le dépôt distant (`push`). À l'inverse, si vous souhaitez récupérer le travail fait par vos coéquipiers ou vous-même fait sur une autre machine, il faut ramener ces modifications depuis le dépôt distant (`pull`). IntelliJ est capable de gérer `git` ; vous trouverez dans le menu `VCS` l'option `Commit`, et l'option `Git` qui contient `push` et `pull`.

Première modification de votre dépôt

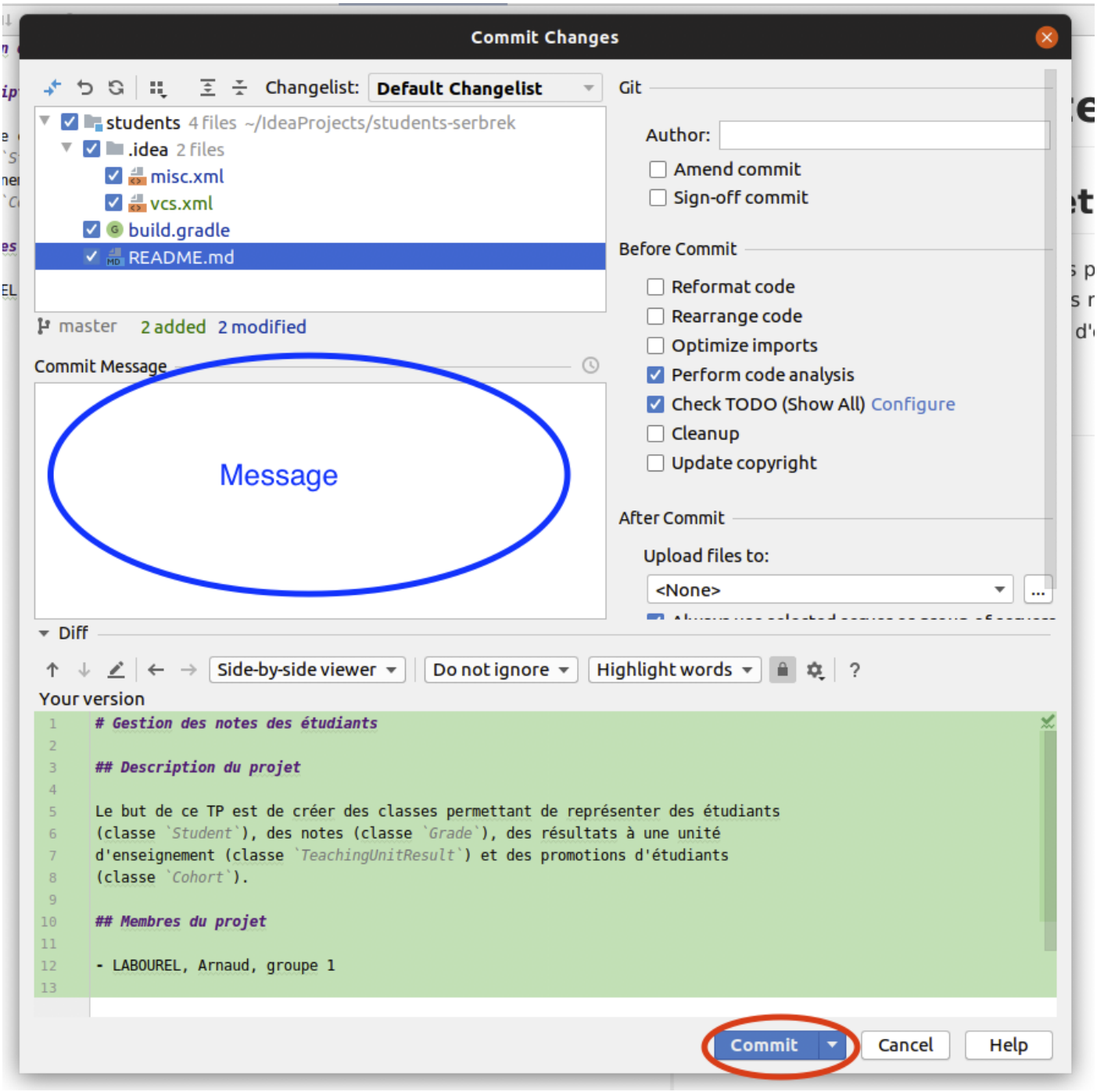
1. Modifiez le fichier `README.md`. Mettez votre nom.



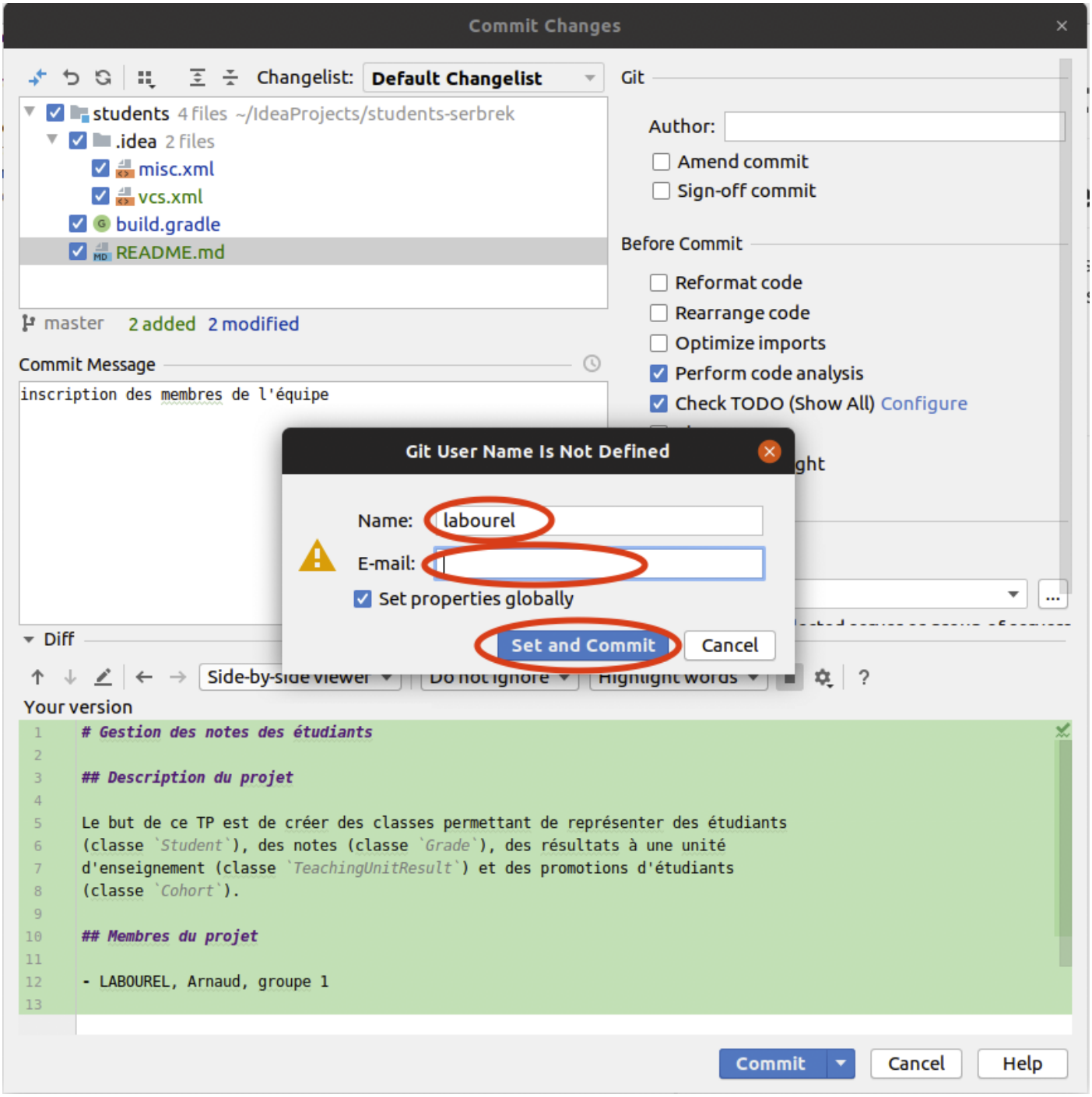
2. Vous allez maintenant mettre à jour votre dépôt local (celui sur votre machine) en effectuant un `commit`. Pour cela vous pouvez aller dans le menu `Git` -> `Commit` ou bien cliquer sur le raccourci en haut à droite de votre fenêtre.



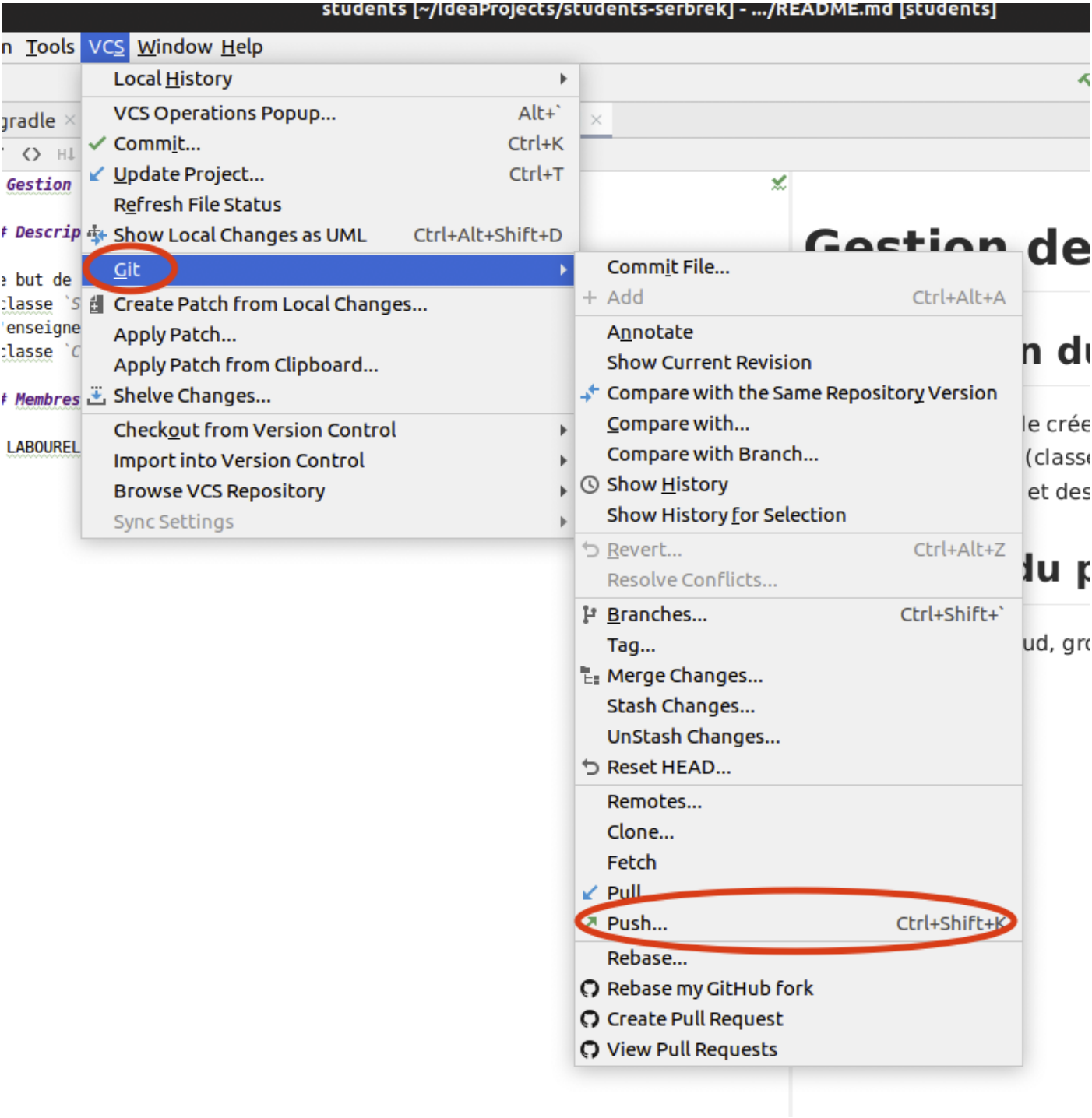
3. Faites un *commit* avec pour message `"ajout nom membre du projet"` en cliquant sur commit après avoir rempli le champ message.



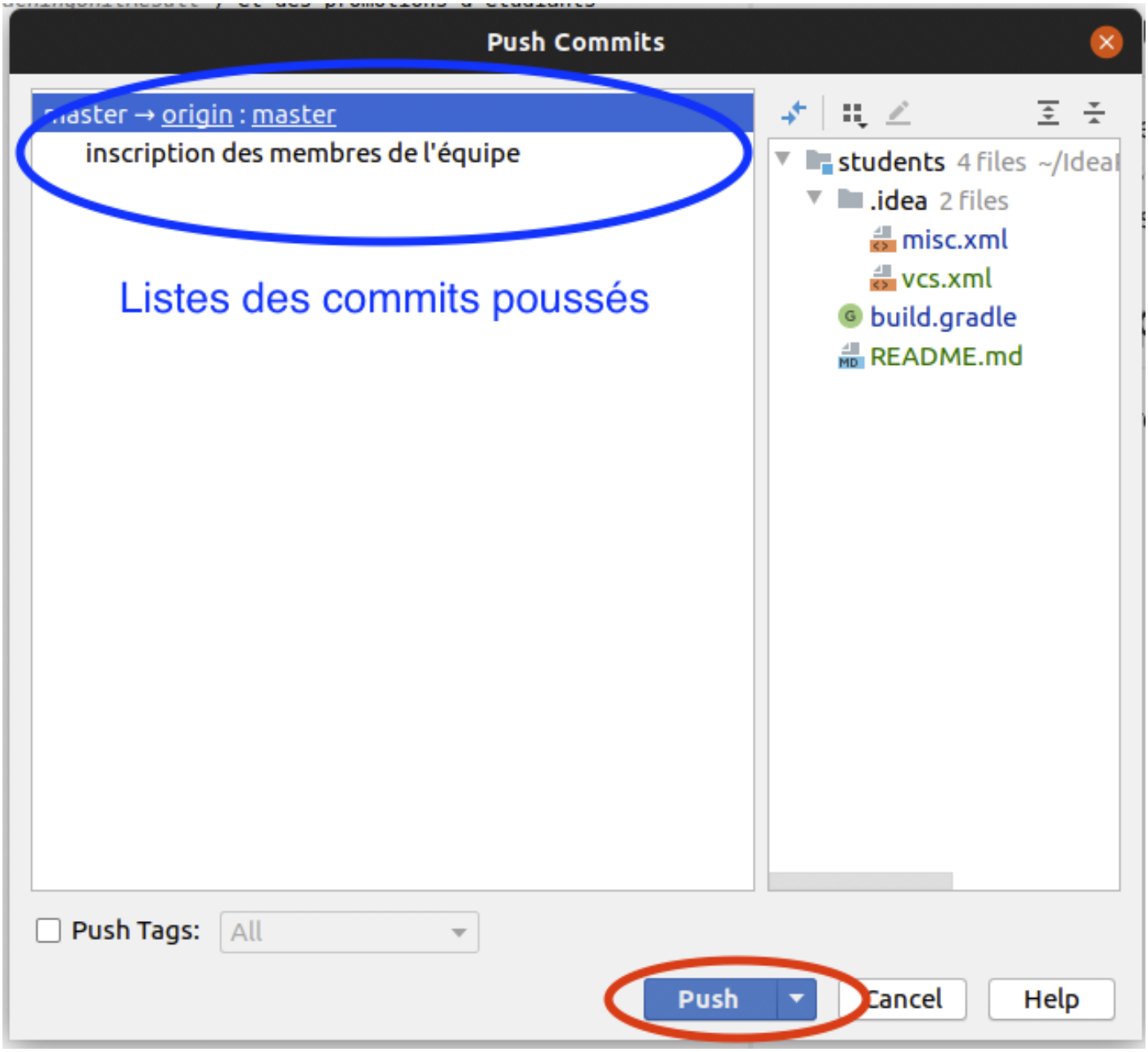
4. Préciser votre nom et email et cliquez sur le bouton `Set and Commit`.



5. Vous avez mis à jour votre dépôt local (sur votre machine) mais pas le dépôt distant (celui sur les serveurs de github). Pour cela, il faut faire un `push`. Allez dans le menu `VCS` -> `git` -> `push`.

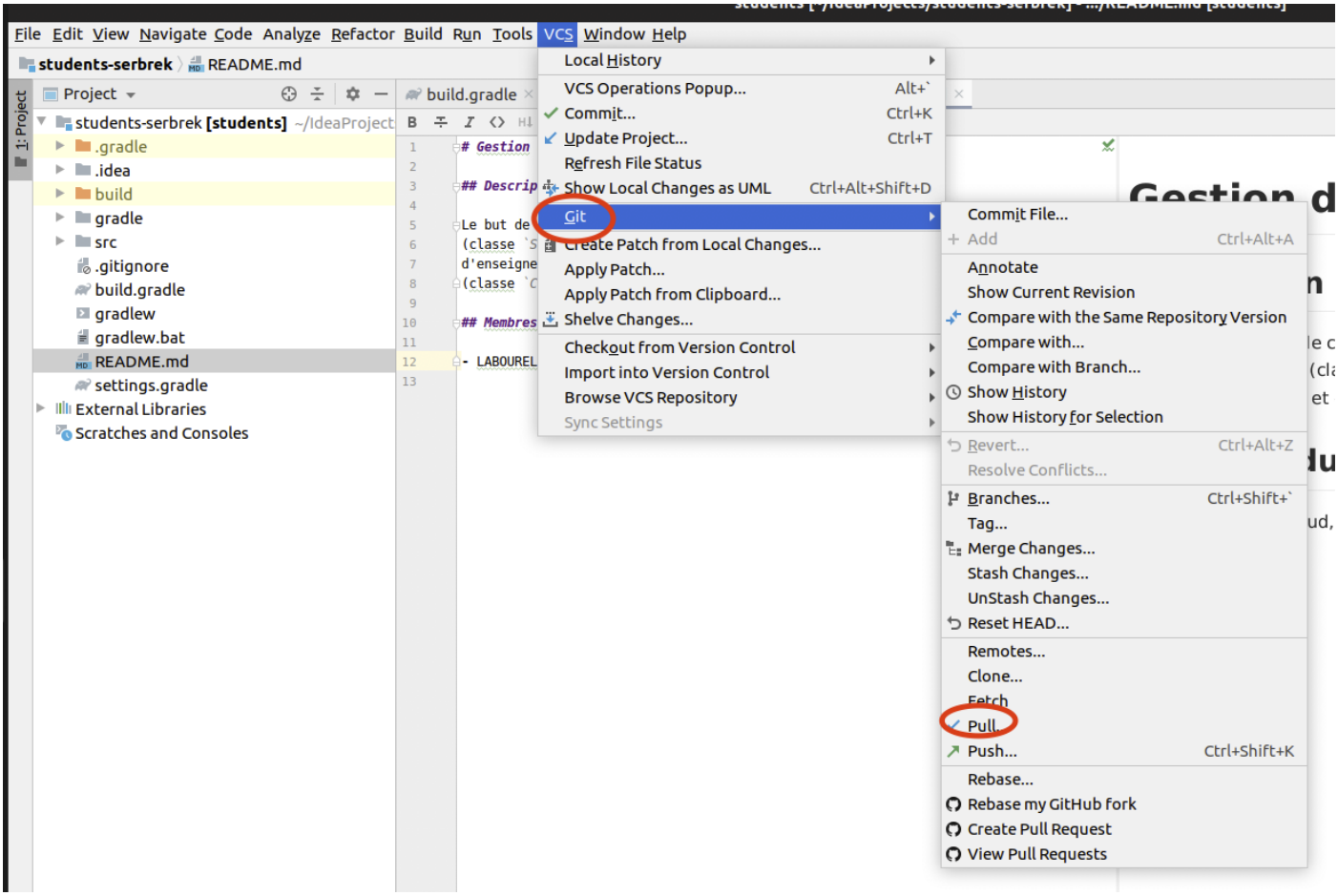


6. Cliquez sur le bouton *push* de la fenêtre qui vient d'apparaître. Vous pouvez voir la liste des commits que vous vous apprêtez à pousser.



Si tout se passe bien un popup `push successful` devrait apparaître en bas à droite de votre fenêtre.

7. Si jamais vous avez besoin de récupérer le projet sur le serveur (par exemple après un *push* depuis un autre ordinateur), il vous suffit de faire un pull. Allez dans le menu `Git` -> `pull`.



Méthodologie pour le TP

Vous allez maintenant pouvoir attaquer la correction du programme java du dépôt. Pour cela vous allez devoir respecter les consignes suivantes :

- À chaque modification de programme, faites un `commit` en sélectionnant les fichiers modifiés. Chaque `commit` doit contenir un message précisant la nature des modifications effectuées.
- À chaque tâche terminée, faites un `push` de votre travail.
- Ceci est le minimum. Vous pouvez faire plus de `commit` et plus de `push`, ainsi que des `pull` si vous avez travaillé avec un autre ordinateur.
- Si vous avez un problème et souhaitez l'aide de votre instructeur en dehors des séances, un `push` lui permet de voir votre programme.

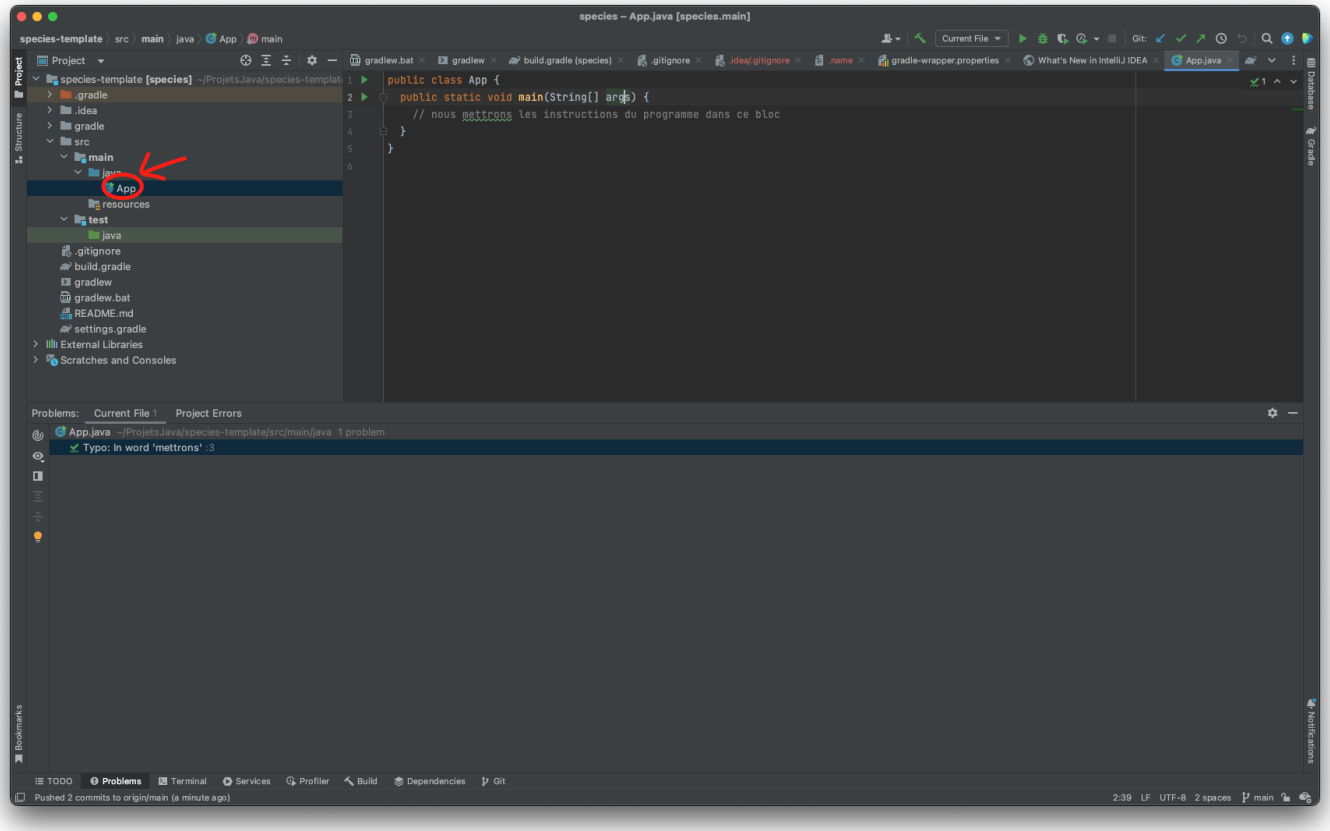
Code Java

Point d'entrée du programme : méthode `main`

Un programme Java est organisé en classes, chaque classe étant définie dans un fichier. Pour l'instant, vous disposez d'une unique classe `App`.

Le fichier `App.java` est dans le répertoire `src/main/java` du projet. Il est accessible via le menu à gauche

d'IDEA.



Pour avoir un programme exécutable, le projet doit avoir un *point d'entrée*, qui contient les instructions qui seront évaluées lors de l'exécution. En Java, les points d'entrée sont toujours déclarés de la manière suivante :

```
1    public static void main(String[] args) {
2      // nous mettrons les instructions du programme dans ce bloc
3    }
```

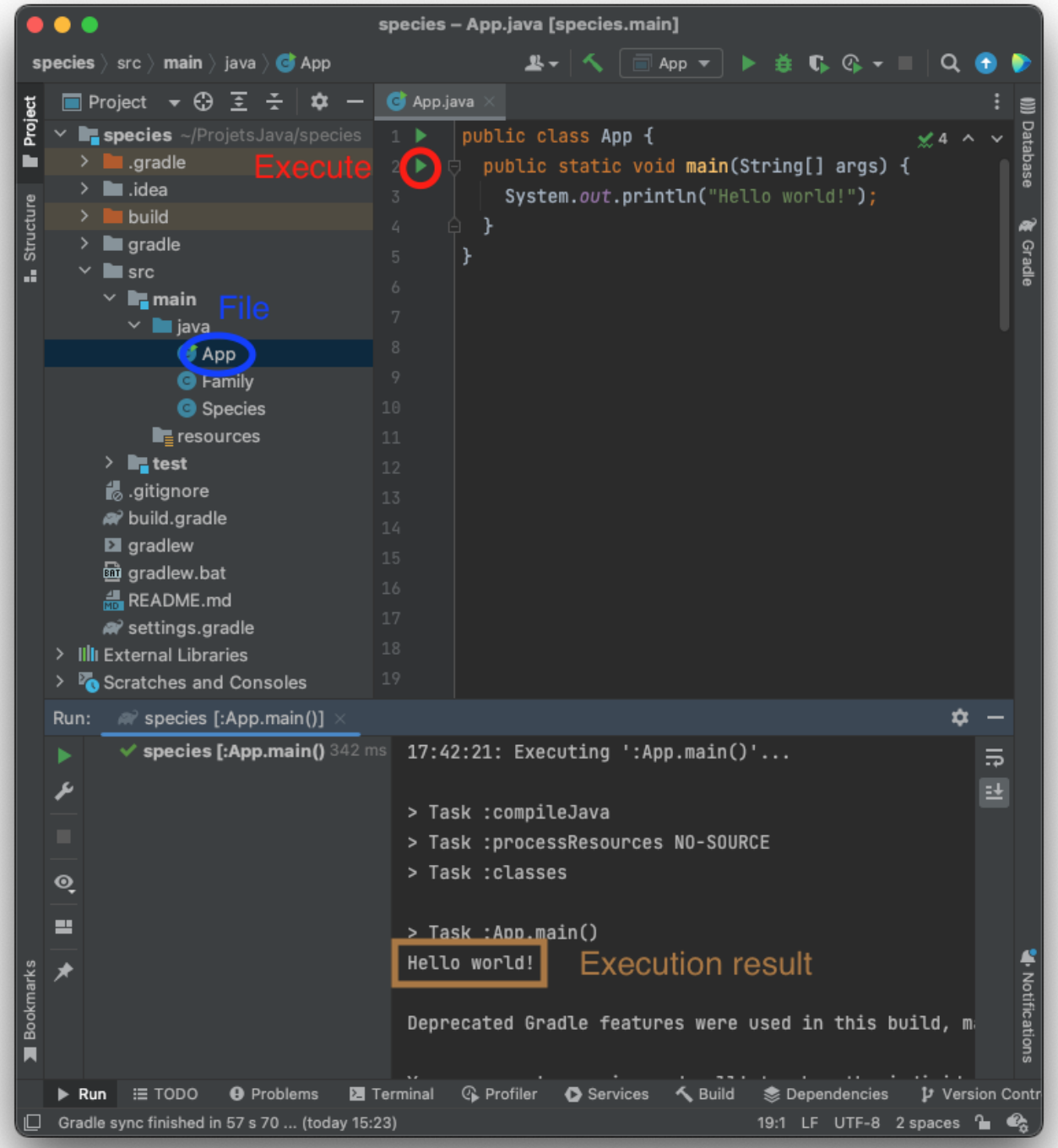
Nous étudierons plus tard ce que veut dire cette formulation, en résumé, elle signifie que nous déclarons une fonction (`static` en Java, on emploie le terme *méthode statique*), sans retour (`void`), sans restriction (`public`), et qui s'appelle `main`. La partie entre parenthèses définit les paramètres de la fonction (ici un tableau de textes nommés `args` et donc de type `String[]` que nous n'utiliserons pas). Ensuite, une paire d'accolades délimite les instructions de la méthode statique `main` (pour le moment vide).

1. Rajouter la ligne d'instruction suivante à l'intérieur du bloc de la fonction `main` :

```
1    System.out.println("Hello world!");
```

qui doit faire afficher à la console la chaîne de caractère "Hello world!".

2. Exécuter le programme en cliquant sur un des boutons triangles à gauche du code.



Si tout s'est bien passé, vous devez voir s'afficher des lignes suivies d'une ligne `Hello world !`.

La première ligne correspond à la commande ayant permis la *compilation* du programme et la deuxième à la

commande ayant permis son *exécution*. Le résultat étant produit sur la dernière ligne.

Compiler le programme consiste à transformer votre programme tel que vous l'avez écrit, c'est-à-dire compréhensible par un être humain, vers un langage propre à l'ordinateur, en l'occurrence du *bytecode java*. Cette transformation est effectuée par la commande *javac*, le *compilateur*. Ainsi vous pourriez aussi compiler votre programme sur votre ordinateur, sans utiliser `repl.it`, du moment que vous avez installé un compilateur Java.

Exécuter le programme consiste à évaluer les instructions prévues par le programmeur, de façon à produire l'effet désiré (calculer les décimales de π, envoyer un courriel, lancer une partie de jeu vidéo, ...). Ce n'est pas votre programme qui est directement exécuté, mais la version compilée obtenue avec `javac`. L'exécution des instructions en *bytecode Java* est supervisée par le programme `java`.

Afin de bien comprendre le fonctionnement de la compilation, on va utiliser le terminal, accessible en bas d'*IntelliJ*, ou bien dans le menu (terminal *MATE* qui est accessible dans le menu au sous-menu *Outils systèmes*) et exécuter le programme sans utiliser les boutons d'IntelliJ.

3. Dans le terminal, tapez la commande `cd src/main/java` (pour aller dans le dossier contenant les fichiers `.java`), puis la commande `ls` affichant le contenu du dossier.

Vous devriez constater que le dossier contient uniquement le fichier `App.java`.

4. Toujours dans le terminal, lancez la commande de compilation `javac App.java`.

Cette commande a créé un fichier compilé `App.class`, vous pouvez le vérifiez avec la commande `ls`.

5. Lancer dans le terminal la commande d'exécution `java App`.

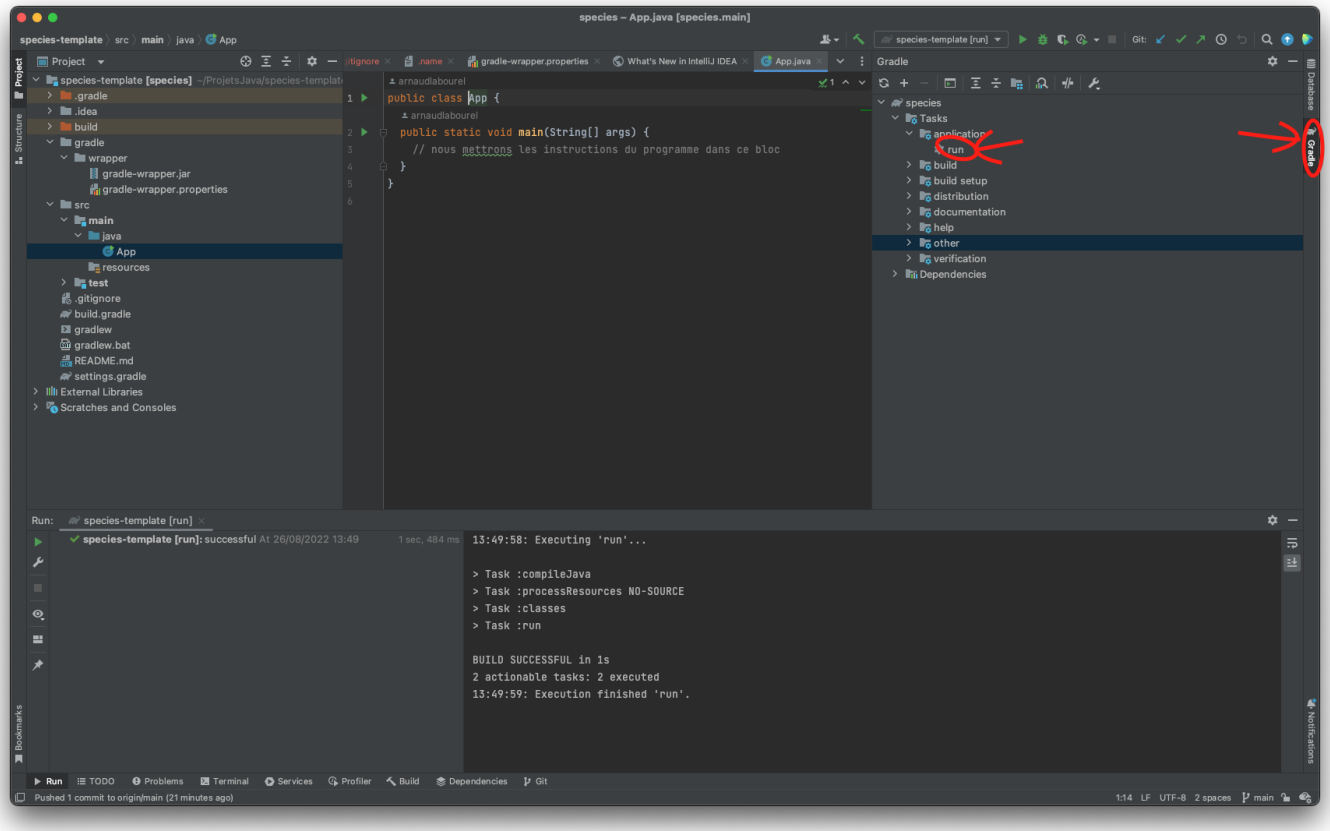
Le programme s'est exécuté et le répertoire est inchangé (pas de nouveau fichier).

Finalement, votre terminal doit ressembler à ceci :

```
> species cd src/main/java
> java ls
App.java
> java javac App.java
> java ls
App.class    App.java
> java App
Hello world!
```

Vous pouvez faire exactement la même manipulation sur votre ordinateur personnel (sous Linux ou Mac OS ; pour Windows, les commandes doivent être adaptées), du moment que `java` a été correctement installé.

Il existe une autre façon d'exécuter le code qui utilise l'outil *gradle* qui sera à privilégier pour les projets plus conséquents. Pour cela il faut ouvrir le menu *gradle* à droite et double-cliquer sur *run* dans application.



6. Exécutez le programme avec *gradle*.

Il est aussi possible d'exécuter *gradle* en ligne de commande en lançant la commande `./gradlew run` à la racine de votre projet.

Gestion d'espèces et de familles

Dans ce TP, vous allez modéliser des espèces (*species* en anglais) d'animaux. Nous souhaitons modéliser les espèces d'animaux ci-dessous (les noms sont en anglais) :

Nom commun	Nom scientifique	Gestation en jours	Âge à maturité en jours
Cat	Felis catus	65	296
Lion	Panthera leo	108	1048
Leopard	Panthera pardus	97	1147
Tiger	Panthera tigris	105	1217
Cougar	Puma concolor	92	623
Serval	Leptailurus serval	74	852

1. Supprimer les instructions du `main`.

Nous allons représenter ces données dans le programme.

2. Créer plusieurs variables contenant les noms communs, noms scientifiques, durée de la gestation, et âge à maturité des trois premières espèces : *Cat*, *Lion* et *Leopard*.

La syntaxe des instructions est disponible au lien suivant : Éléments de syntaxe Java

3. Ajouter une instruction pour afficher en console la durée de gestation moyenne sous le format `The average gestation period is 90.0 days`.

Astuce 1 : pour taper `System.out.println` sous IntelliJ, il suffit de taper `sout` puis entrée.

Astuce 2 : `System.out.println` peut afficher plusieurs valeurs les unes à la suite des autres, il suffit de les séparer par des symboles `+`. Ces valeurs peuvent être des chaînes de caractères, mais aussi des nombres (qui seront converties automatiquement en chaîne de caractère). Par exemple on peut écrire :

```
1   System.out.println("sin(pi/3) = " + Math.sin(Math.PI / 3));
2   // ce qui affiche : sin(pi/3) = 0.8660254037844386
```

4. Ajouter une instruction pour afficher en console l'age maximal à maturité sous le format `The maximal age at maturity is 1147 days`. Vous utiliserez pour cela, la méthode `Integer.max` qui prend en arguments deux entiers et retourne la plus grande des deux valeurs.

5. Rajouter des instructions pour afficher l'ensemble des trois espèces, sous la forme :

```
1 Cat (Felis catus), age at maturity: 296 days, gestation period: 65 days
2 Lion (Panthera leo), age at maturity: 1048 days, gestation period: 108 days
3 Leopard (Panthera pardus), age at maturity: 1147 days, gestation period: 97
     days
4 The average gestation period is 90.0 days
5 The maximal age at maturity is 1147 days
```

Vous avez sûrement remarqué que vous avez écrit plusieurs fois les mêmes instructions, avec seulement de petites variations. Lorsqu'on programme, on essaie d'éviter de se répéter, c'est le principe DRY (*Don't Repeat Yourself*). Nous allons réorganiser le programme de sorte qu'il ne soit pas nécessaire de se répéter.

La première remarque est que les espèces sont tous constitués de quatre informations :

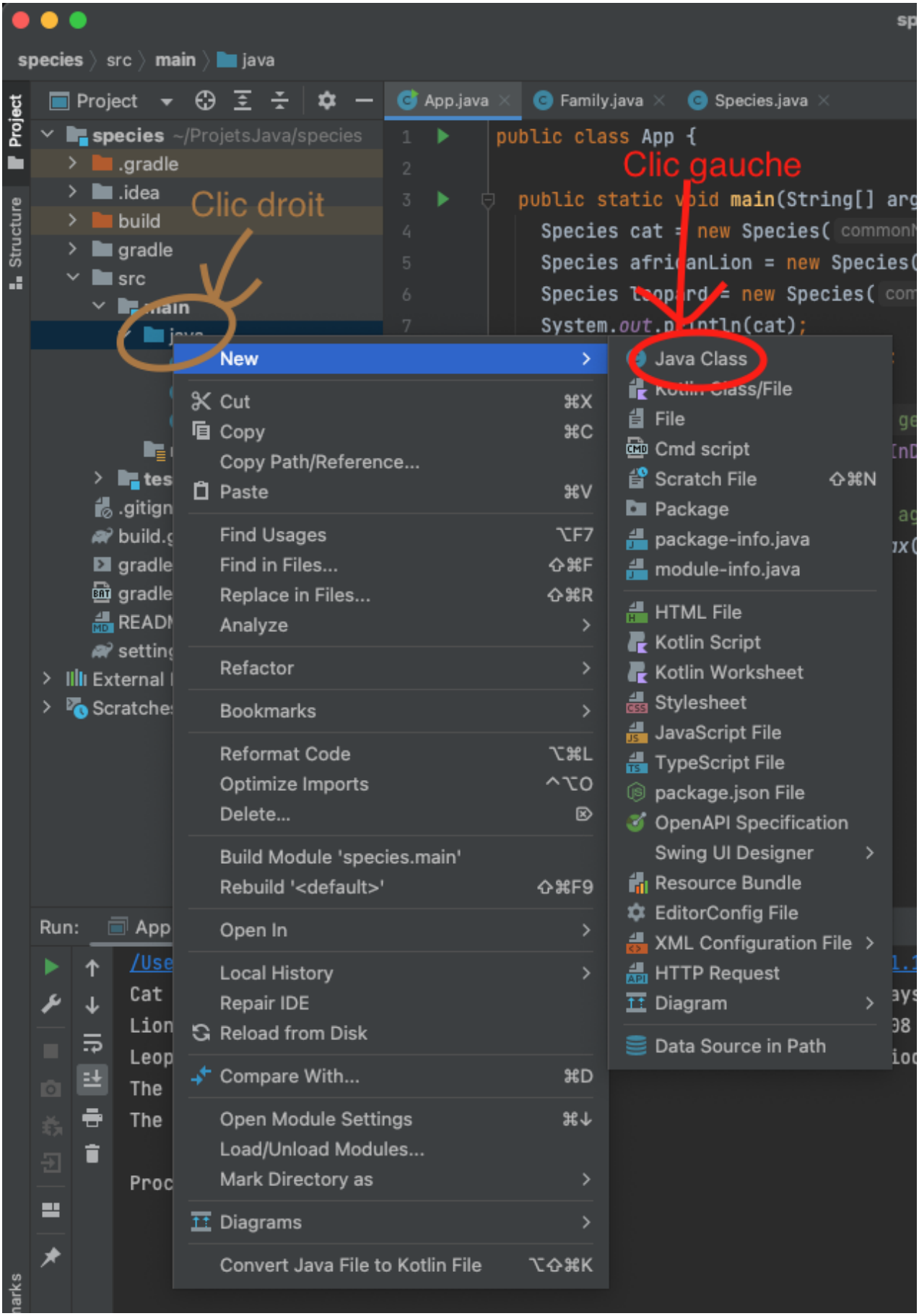
— le nom commun de l'espèce (*common name*) ;
— le nom scientifique de l'espèce (*scientific name*) ;

— l'age à maturité en jours (*age at maturity in days*) et
— la période de gestation en jours (*gestation period in days*).

Plutôt que de manipuler les quatre informations séparément, nous allons les grouper pour les mettre dans une seule variable. Un tel groupement va constituer un *objet*. Nous aurons donc trois objets, un par espèce.

Pour créer un objet, il faut un plan, le plan des objets espèces. Les plans sont appelés *classes*, nous devons donc créer une classe des espèces.

6. Créer une classe `Species` (espèce en anglais) en cliquant droit sur la ligne à gauche correspondant au dossier `java` dans `src/main/` puis en choisissant `create class` en cliquant gauche, lui donner le nom `Species` (bien respecter la majuscule !) et validez en appuyant sur la touche entrée.


species src main java
Project
App.java
Family.java
Species.java
Clic droit
Clic gauche
New
Cut ⌘X
Copy ⌘C
Copy Path/Reference...
Paste ⌘V
Find Usages ⌥F7
Find in Files... ⇧⌘F
Replace in Files... ⇧⌘R
Analyze
Refactor
Bookmarks
Reformat Code ⌥⌘L
Optimize Imports ^⌥O
Delete...
Build Module 'species.main'
Rebuild '<default>' ⇧⌘F9
Open In
Local History
Repair IDE
Reload from Disk
Compare With... ⌘D
Open Module Settings ⌘↓
Load/Unload Modules...
Mark Directory as
Diagrams
Convert Java File to Kotlin File ⌥⇧⌘K
Java Class
File
Cmd script
Scratch File ⇧⌘N
Package
package-info.java
module-info.java
HTML File
Kotlin Script
Kotlin Worksheet
Stylesheet
JavaScript File
TypeScript File
package.json File
OpenAPI Specification
Swing UI Designer
Resource Bundle
EditorConfig File
XML Configuration File
HTTP Request
Diagram
Data Source in Path
Run:

Le fichier créé contient uniquement la déclaration de classe `public class Species` suivie d'une accolade ouvrante et d'une accolade fermante. C'est entre ces accolades que nous allons définir le contenu de la classe, par des déclarations.

Les déclarations sont de trois sortes : d'*attribut*, de *méthode* et de *constructeur*. Un attribut est une information que possède chaque objet créé selon le plan de cette classe. Dans notre cas, les informations sont le nom commun de l'espèce, le nom scientifique de l'espèce, l'âge à maturité en jours et la période de gestation en jours. Les objets espèces pourront ne pas avoir les mêmes valeurs pour ces informations, mais ils doivent avoir ces quatre informations.

Pour déclarer un attribut, dans le bloc de la classe, on ajoute une déclaration avec la syntaxe *identifiant du type de l'attribut* suivi de l'*identifiant de l'attribut*, terminé par un point-virgule. Par exemple, pour déclarer le nom commun on écrit :

```
1    String commonName;
```

L'identifiant de l'attribut ne doit pas contenir d'espaces. On utilise donc le *camel case* (de l'anglais, littéralement « casse de chameau ») qui est une notation consistant à écrire un ensemble de mots en les liant sans espace ni ponctuation, et en mettant en capitale la première lettre de chaque mot. Par convention, en Java le premier mot doit commencer par une lettre en minuscule.

7. Ajouter la déclaration du nom commun de l'espèce, du nom scientifique de l'espèce, de l'âge à maturité en jours et de la période de gestation en jours pour la classe `Species`. Choisissez le type et le nom les plus appropriés dans chaque cas !

8. Replacez-vous dans la classe `App`. Au début de la méthode `main`, déclarer trois variables de type `Species`, une par espèce (nommées `cat`, `lion` et `leopard`). En effet, le fait de créer une classe a aussi créé le type des objets construits selon cette classe.

Pour l'instant, ces trois variables sont vides. Pour créer un objet selon la classe `Species`, on utilise le mot réservé de Java `new`, suivi du nom de la classe, suivi d'une paire de parenthèse. Par exemple :

```
1  Species cat = new Species();
```

9. Initialiser les trois variables de type `Species`, avec trois objets différents de la classe `Species`.

Les attributs d'un objet s'utilisent comme des variables : ce sont des emplacements de mémoires dans lesquels on peut stocker des valeurs (entiers, nombres flottants, références d'objet). Pour accéder à un attribut, on utilise la syntaxe *identifiant de l'objet* suivi d'un point suivi de l'*identifiant de l'attribut*. Par exemple dans la méthode `main`, `cat.commonName` est l'attribut `commonName` de l'objet référencé par la variable `cat`.

10. Essayer d'afficher le nom commun de `cat` en utilisant l'attribut `commonName`. Que constatez-vous ?

11. Utiliser des affectations pour attribuer à chaque espèce son nom commun. Modifier les instructions d'affichage pour qu'elles utilisent l'attribut `commonName` de chaque espèce, plutôt que les variables de type `String` définies dans le `main` auparavant.

12. De la même façon, initialiser les attributs du nom scientifique de l'espèce, de l'age à maturité en jours et de la période de gestation en jours de chaque objet.

13. Remplacer aussi le calcul de l'âge à maturité maximal et de la durée de gestation moyenne en utilisant les attributs des espèces directement.

Nous souhaitons maintenant changer l'affichage pour afficher par espèce la durée de la période allant de la conception à la maturité et donc obtenir l'affichage suivant :

```
1 Cat (Felis catus), conception to maturity: 12 months
2 Lion (Panthera leo), conception to maturity: 39 months
3 Leopard (Panthera pardus), conception  to maturity: 41 months
4 The average gestation period is 90.0 days
5 The maximal age at maturity is 1147 days
```

La durée de cette période se calcule en sommant la durée de gestation et l'age à maturité et en divisant par 30 pour obtenir un nombre de mois. Afin d'obtenir un nombre de mois entier, on utilisera la méthode `Math.round` qui prend une valeur flottante et retourne la valeur entière (de type `long`) la plus proche. Afin d'obtenir une valeur de type `int`, il faudra donc forcer la conversion en mettant `(int)` devant l'expression du calcul de la valeur.

Plutôt que faire ce calcul dans la classe `App`, nous allons l'ajouter directement dans la classe `Species`.

La deuxième sorte de déclaration pouvant être faite dans une classe est la déclaration de méthode. Une méthode d'un objet est un point d'accès permettant aux autres objets de faire des requêtes à cet objet. Par exemple, nous allons écrire une méthode `getConceptionToMaturityInMonths` permettant de demander à un objet de type `Species` quel est la durée de la période allant de la conception à la maturité de cette espèce.

Pour dire que les objets créés depuis la classe `Species` possèdent une méthode `getConceptionToMaturityInMonths`, il faut donc déclarer la méthode dans la classe `Species`. Cette méthode calcule une durée en mois, et donc lorsque la requête `getConceptionToMaturityInMonths` est adressée à un objet de type `Species`, celui-ci doit répondre cette durée. On dit que `getConceptionToMaturityInMonths` *retourne* une durée en mois, qui est de type entier (`int`).

La syntaxe d'une déclaration de méthode est *identifiant du type de retour* suivi de l'*identifiant de la méthode* suivi d'une paire de parenthèses (qui pourra contenir des paramètres), puis d'une paire d'accolades. La paire d'accolade définit un bloc qui doit contenir les instructions de la méthode.

```
int getConceptionToMaturityInMonths(){
  // bloc d'instructions de la méthode
}
```

Les instructions de la méthode peuvent être arbitraires, mais la dernière instruction exécutée doit être une instruction `return`. Cette instruction permet de terminer le traitement de la méthode et de spécifier la valeur retournée. Sa syntaxe est spécifiée dans le document d'Éléments de syntaxe Java.

14. Écrivez la méthode `getConceptionToMaturityInMonths` dans la classe `Species` de telle sorte qu'elle renvoie la durée de la période en nombre de mois allant de la conception à la maturité.

On se replace dans la classe `Main`, au niveau de l'affichage des espèces. On peut maintenant remplacer chaque affichage en utilisant la méthode que nous venons d'écrire. Pour obtenir la durée de la période à afficher, on appelle la méthode `getConceptionToMaturityInMonths` de cette espèce. La syntaxe pour un appel de méthode est spécifiée dans le document d'Éléments de syntaxe Java. Puisque la méthode va retourner la durée calculée, l'appel de méthode est une *expression* qui a comme valeur la valeur retournée.

15. Réécrire l'affichage des espèces pour afficher la durée de la période allant de la conception à la maturité.

On souhaite calculer la durée minimale de la période allant de la conception à la maturité et l'afficher de la manière suivante :

```
Cat (Felis catus), conception to maturity: 12 months
Lion (Panthera leo), conception to maturity: 39 months
Leopard (Panthera pardus), conception to maturity: 41 months
The average gestation period is 90.0 days
The maximal age at maturity is 1147 days
The minimal period from conception to maturity is 12 months
```

16. Changez le code de la méthode `main` de la classe `App` pour obtenir l'affichage ci-dessus.

La plus grosse source de répétition de notre programme pour l'instant est l'affichage de chaque espèce. La solution est de créer une méthode `display` dans la classe `Species`, qui se charge d'écrire la description de l'espèce dans la console. Cette méthode effectue un affichage, mais n'a pas de valeur à répondre. On dit qu'elle ne retourne rien, et à la place de son type de retour, on utilise le mot réservé `void`. Aussi, on n'est pas obligé d'utiliser l'instruction `return` dans une méthode qui ne retourne rien.

17. Écrivez la déclaration de la méthode `display` (laisser le bloc d'instructions vide).

18. Pour les instructions, commencer par récupérer une des instructions d'affichage actuellement dans `main`, et la couper-coller vers `display`.

Tout ce qu'il reste à faire est de remplacer l'identifiant de l'objet de type `Species`, par l'identifiant de l'objet que nous voulons afficher.

Mais quel objet voulons-nous afficher ? La méthode `display` est une méthode déclarée dans la classe `Species`. Cela signifie que tout objet créé à partir de la classe `Species` (souvenez-vous qu'une classe est un plan de construction) possédera sa propre méthode `display`, comme il possède aussi ses propres attributs comme `commonName`.

Pour que la méthode `display` d'un objet fonctionne correctement, il faut qu'elle utilise les attributs `commonName`, `scientificName` et la méthode `getConceptionToMaturityInMonths` (si vous les avez nommées ainsi) du même objet. Comme nous écrivons seulement un plan de construction, nous ne connaissons pas forcément tous les objets qui seront créés à partir de ce plan. Mais nous pouvons néanmoins parler de l'objet qui sera construit, en utilisant le mot réservé `this`. Ainsi `this.commonName`, utilisé dans la méthode `display`, fait référence à l'attribut `commonName` de l'objet dont la méthode `display` est évalué.

19. Corriger les instructions dans `display` pour les rendre correctes.

20. Dans la classe `Main`, remplacer les instructions d'affichage des espèces par des appels à la méthode `display`.

21. Vérifier que votre programme fonctionne toujours correctement.

Nous continuons de chasser les redondances. La façon dont nous initialisons les attributs des espèces n'est pas satisfaisantes, puisque nous faisons trois fois les mêmes quatre instructions. Par ailleurs, lorsqu'on crée une espèce, il se peut qu'on oublie d'initialiser un des attributs de cette espèce.

Pour améliorer cela, nous allons déporter les instructions d'initialisation directement dans la classe `Species`, en ajoutant un *constructeur*. Le rôle d'un constructeur est d'initialiser les attributs d'un objet (attention le terme est trompeur, le constructeur ne construit rien du tout).

Les constructeurs sont la troisième sorte de déclarations possibles dans une classe, après les attributs et les méthodes. Pour déclarer un constructeur, on utilise la syntaxe *identifiant de la classe* suivi d'une paire de parenthèses, suivi d'une paire d'accolades.

La paire de parenthèses peut contenir des paramètres séparés par des virgules, chaque paramètre se composant du type et de l'identifiant du paramètre. Les paramètres contiendront typiquement des valeurs à utiliser pour initialiser l'objet, par exemple les valeurs initiales des attributs de l'objet.

La paire d'accolades peut contenir des instructions, ce sont ces instructions qui serviront à initialiser l'objet.

22. Créer une déclaration d'un constructeur dans la classe `Species`. Ce constructeur a 4 paramètres : `initCommonName` de type `String`, `initScientificName` de type `String`, `initGestationPeriodInDays` de type `int` et `initAgeAtMaturityInDays` de type `int`. Pour l'instant, laisser le bloc d'instructions vide.

Regarder la classe `Main`. Des erreurs sont apparues. En effet, lorsqu'une classe définit au moins un constructeur, lors de la création d'un objet avec `new`, les arguments fournis dans l'instruction `new` doivent correspondre aux paramètres attendus par un des constructeurs.

Les arguments se placent entre les parenthèses qui suivent le nom de la classe, séparés par des virgules. Les arguments doivent respecter l'ordre et le type des paramètres du constructeur.

23. Corrigez les trois instructions `new` de la méthode `main`.

On retourne maintenant dans la classe `Species`, pour terminer le constructeur en écrivant ses instructions. Souvenez-vous que `this` dénote l'objet courant, et vous donne donc accès à ses méthodes et ses attributs.

24. Écrivez quatre instructions dans le constructeur de `Species`, chacune initialisant un attribut avec la valeur d'un des paramètres du constructeur. Vérifier le bon fonctionnement de votre programme.

Gestion d'une famille d'espèces

Les espèces qu'on a considérées pour le moment font toute partie de la même famille et il semblerait donc raisonnable de les stocker dans un objet de type `Family`.

1. Créer une classe `Family`.

Quels sont les attributs d'une famille ? Une famille peut être caractérisée par deux caractéristiques (et donc deux attributs) :

— le nom de la famille (`Felidae` dans notre cas) ;
— les espèces de la famille, c'est-à-dire une liste d'espèces.

2. Ajouter dans la classe `Family` la déclaration d'un attribut correspondant au nom de la famille.

Java propose de nombreuses classes pré-écrites et prêtes à être utilisée. Ces classes constituent la *librairie standard*. Parmi ces classes, certaines sont des classes représentant des listes. Pour créer une liste d'espèces, on écrit :

```
List<Species> listOfSpecies = new ArrayList<>();
```

On reconnaît la syntaxe de la déclaration et de l'initialisation d'une variable d'identifiant `listOfSpecies`. Son type, un peu particulier, est `List<Species>`, ce qui signifie *liste d'espèces*. L'initialisation est faite avec un nouvel objet (utilisation de `new`) construit avec selon le plan défini par la classe `ArrayList` (la présence des symboles `<>` sera expliquée plus tard dans le cours).

3. Recopier cette déclaration dans la classe `Family`, pour en faire une déclaration d'attribut de cette classe.

Si vous exécutez le code, vous obtenez deux erreurs dont la première est

```
java: cannot find symbol
  symbol:   class List
  location: class Family
```

Le compilateur ne trouve pas le terme `List` et il en est de même pour le terme `ArrayList` dans l'erreur qui suit. En effet, pour utiliser une classe de la librairie standard, il faut le déclarer par une déclaration d'import. Les classes à importer dans ce cas sont `java.util.List` et `java.util.ArrayList`.

4. Sur la première ligne de la classe `Family` (donc avant la déclaration de la classe), ajouter `import java.util.List;`. Sur la ligne d'après, procéder de même pour `import java.util.ArrayList;`.

5. Définissez un constructeur dans `Family` permettant de créer une famille sans espèces avec un nom donné en argument.

6. Ajouter une méthode `addSpecies` dans la classe `Family`.

Cette méthode ne retourne rien. Elle prend en paramètre un objet de type `Species`. Pour mettre un paramètre à une méthode, faites comme pour les constructeurs : ajoutez-le entre les parenthèses de la déclaration de méthode, sous le format *identifiant du type* suivi de l'*identifiant du paramètre*.

Comme instruction de la méthode, on veut ajouter l'espèce à la liste de la famille. La liste `listOfSpecies` est elle-même un objet, qui possède des méthodes. Ces méthodes permettent de faire des manipulations sur la liste.

7. Dans le bloc de la méthode `addSpecies`, commencer une instruction en tapant l'identifiant de la liste (`listOfSpecies`) suivi d'un point. Un menu déroulant s'affiche (parfois cela prend un peu de temps), listant toutes les méthodes de l'objet `listOfSpecies`. L'une des méthodes s'appellent `add`, et comme son nom l'indique permet d'*ajouter* un élément à la liste. Les noms de méthodes sont choisis pour expliciter le rôle de la méthode. Vous pouvez choisir la méthode

avec les flèches et valider avec entrée ou tabulation. Terminer l'instruction d'ajout de l'espèce dans la liste, en fournissant à la méthode `add` son argument qui est l'espèce à ajouter.

Maintenant, nous souhaitons ajouter une méthode `display`, affichant toutes les espèces de la famille.

Pour cela, il faut considérer toutes les espèces de la liste, un par un, et les afficher. Pour effectuer des instructions pour chaque élément d'une liste, on utilise une boucle `for`. La syntaxe de la boucle `for` en Java est :

```
for (Species oneSpecies : listOfSpecies) {
  // instructions à faire pour chaque espéce
}
```

Dans les parenthèses, on trouve d'abord l'*identifiant de type* des éléments de la liste, puis l'*identifiant de variable* qui sera utilisé pour chaque espèce dans les instructions du bloc de la boucle (entre les accolades), puis le deux-points puis l'*identifiant de la liste*. Entre les accolades, se trouvent les instructions à effectuer sur chaque objet de la liste. Ces instructions peuvent utiliser l'identifiant `oneSpecies` pour faire référence à l'objet provenant de la liste.

8. Ajoutez une méthode `display`, affichant toutes les espèces de la famille. Pour cela, recopiez la boucle et mettre une instruction provoquant l'affichage de l'espèce. Ajouter une instruction avant la boucle pour afficher le nom de la famille.

9. Dans la classe `Main`, créez une nouvelle variable `felidae` de type `Family`, initialisée avec un nouvel objet qui aura pour nom `Felidae`.

10. Utilisez la méthode `addSpecies` pour ajouter les trois espèces dans la famille.

11. Remplacez l'affichage de chaque espèce par un seul appel de la méthode `display` à l'objet `felidae`.

```
Felidae
Cat (Felis catus), conception to maturity: 12 months
Lion (Panthera leo), conception to maturity: 39 months
Leopard (Panthera pardus), conception to maturity: 41 months
The average gestation period is 90.0 days
The maximal age at maturity is 1147 days
The minimal period from conception to maturity is 12 months
```

12. Ajouter une méthode `maximalAgeAtMaturityInDays` à la classe `Family`, permettant de calculer l'age maximal à maturité des espèces de la famille.

Pour calculer cet âge, il faut partir d'un âge égal à 0, considérer tous les éléments de la liste un par un, et mettre à jour l'age si l'age de l'élément est supérieur. On utilise une variable `ageMax`, initialement égale à 0, qui contiendra l'age maximal.

13. Ajouter une méthode `minimalConceptionFromMaturityInMonths` à la classe `Family`, permettant de calculer la durée minimale de la période entre la conception et la maturité des espèces de la famille.

Pour calculer cette durée, il faut partir d'une durée égale à `Integer.MAX_VALUE`, considérer tous les éléments de la liste un par un, et mettre à jour la durée si la durée de l'élément est inférieure. On utilise une variable `minDuration`, initialement égale à `Integer.MAX_VALUE`, qui contiendra la durée minimale.

14. Ajouter une méthode `averageGestationPeriodInDays` à la classe `Family`, permettant de calculer la durée moyenne de la période de gestation des espèces de la famille.

Pour calculer cette durée, il faut partir d'une somme nulle, considérer tous les éléments de la liste un par un, et ajouter la durée de gestation de l'élément. On utilise une variable `sum`, initialement égale à `0`, qui contiendra la somme des durées. Une fois cette somme calculée, il suffira de la diviser par le nombre d'éléments de la liste qu'on pourra obtenir avec la méthode `size()` de `List`.

15. Modifiez la méthode `display` de la classe `Family`, pour qu'elle affiche en plus de ce qu'elle affiche déjà :
    - la durée moyenne de la période de gestation,
    - l'age maximal à maturité,
    - la durée minimale de la période allant de la conception à la maturité.

16. Modifiez aussi la méthode `main` de la classe `Main`, pour que l'affichage reste le même qu'avant.

17. Rajoutez dans la fonction *main*, les trois variables correspondant aux espèces Tigre, Cougar et Serval et rajoutez-les à la famille pour obtenir l'affichage ci-dessous.

```
Felidae
Cat (Felis catus), conception to maturity: 12 months
Lion (Panthera leo), conception to maturity: 39 months
Leopard (Panthera pardus), conception to maturity: 41 months
Tiger (Panthera tigris), conception to maturity: 44 months
Cougar (Puma concolor), conception to maturity: 24 months
Serval (Leptailurus serval), conception to maturity: 31 months
The average gestation period is 90.16666666666667 days
The maximal age at maturity is 1217 days
The minimal period from conception to maturity is 12 months
```