

Conception d'applications – L3 informatique – UBO

Mise en place de l'environnement de développement

L'UE « conception d'applications » consiste à développer un projet Java en utilisant des outils et méthodes de gestion de code :

- Eclipse comme IDE (ou un autre logiciel de votre choix mais avec moins d'assistance des enseignants dans ce cas)
- Git pour le partage de code
- JUnit pour le test du code
- Javadoc pour la documentation du code
- Checkstyle pour la qualité du code
- Intégration d'une interface graphique avec Scenebuilder
- Déploiement d'un logiciel exécutable

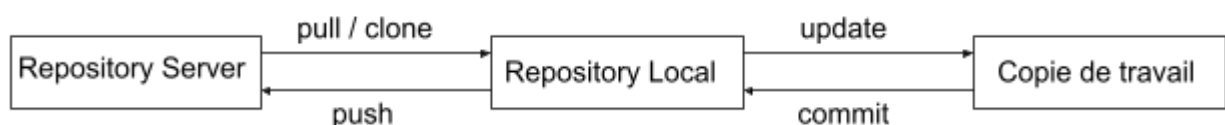
Ce document contient des informations techniques permettant d'utiliser ces différents outils.

1. Git

Git est un outil de gestion de version de code. Il offre principalement les fonctionnalités suivantes :

- Partage de code source entre plusieurs développeurs
- Conservation de l'historique des mises à jour du code source
- Développement parallèle de plusieurs branches dans le même projet que l'on pourra ensuite fusionner
- Gestion des conflits quand plusieurs développeurs modifient les mêmes parties d'un fichier

Par rapport aux autres outils similaires, Git est décentralisé. Son principe est qu'un serveur central gère un dépôt (repository) de projets et qu'un développeur clone un projet dans un dépôt local. Ensuite, il travaille sur la copie locale du projet avant de valider les modifications sur le dépôt local puis sur le dépôt central. Ce développement est intéressant notamment pour les logiciels libres où n'importe qui peut cloner du code existant sans nécessiter un droit d'accès au serveur principal. L'inconvénient est que la mise à jour sur le serveur se fait en deux étapes, d'abord un "commit" local puis un "push" sur le serveur :



Il existe de nombreux serveurs Git librement utilisables sur Internet mais l'université propose également son Gitlab : <https://gitlab-depinfo.univ-brest.fr/>

Sur le Gitlab de l'université, dans le cadre LDAP, vous vous identifiez en utilisant votre login et mot de passe de votre compte à l'université (ENT). A chaque fois que Git vous demandera une authentification, ce sont ces identifiants qu'il faut utiliser.

1.1 Clé SSH (optionnel)

Dans le cadre du projet, vous n'avez pas besoin d'associer une clé SSH à votre compte Git mais dans un autre contexte cela pourrait être nécessaire et voici comment le faire.

A votre première connexion, vous verrez apparaître un message vous disant que vous devez ajouter une clé SSH pour gérer des projets. Cliquez sur ce message pour la rajouter. Si le message n'apparaît plus, cliquez en haut à droite sur l'icône représentant une personne, sélectionnez *Edit profile* puis *SSH Keys* dans le menu qui apparaît à gauche.

Une clé SSH se compose d'une partie publique et d'une partie privée. C'est la partie publique qu'il faut associer à votre compte sous Gitlab. Il faut de préférence générer une clé au format EdDSA :

- Exécuter en ligne de commande (Windows ou Linux) : `$ ssh-keygen -t ed25519`
- Lancer l'outil PuttyGen sous Windows

A la création de votre clé, vous pouvez rajouter une phrase ou un mot de passe (c'est optionnel mais fortement recommandé pour sécuriser la clé privée). Sauvegardez les clés publiques et privées ou notez l'endroit où les fichiers ont été générés, et copiez/collez la clé publique dans la zone de texte requise sous Gitlab. Si vous avez un message sur fond rouge, c'est qu'il y a eu un problème...

1.2 Création d'un projet Git

La création du projet Git se fait en plusieurs étapes :

- Installation d'une interface graphique Git, le logiciel Fork
- Créer son compte Gitlab UBO
- Initialiser le répertoire local Gitlab qui contiendra le projet Java

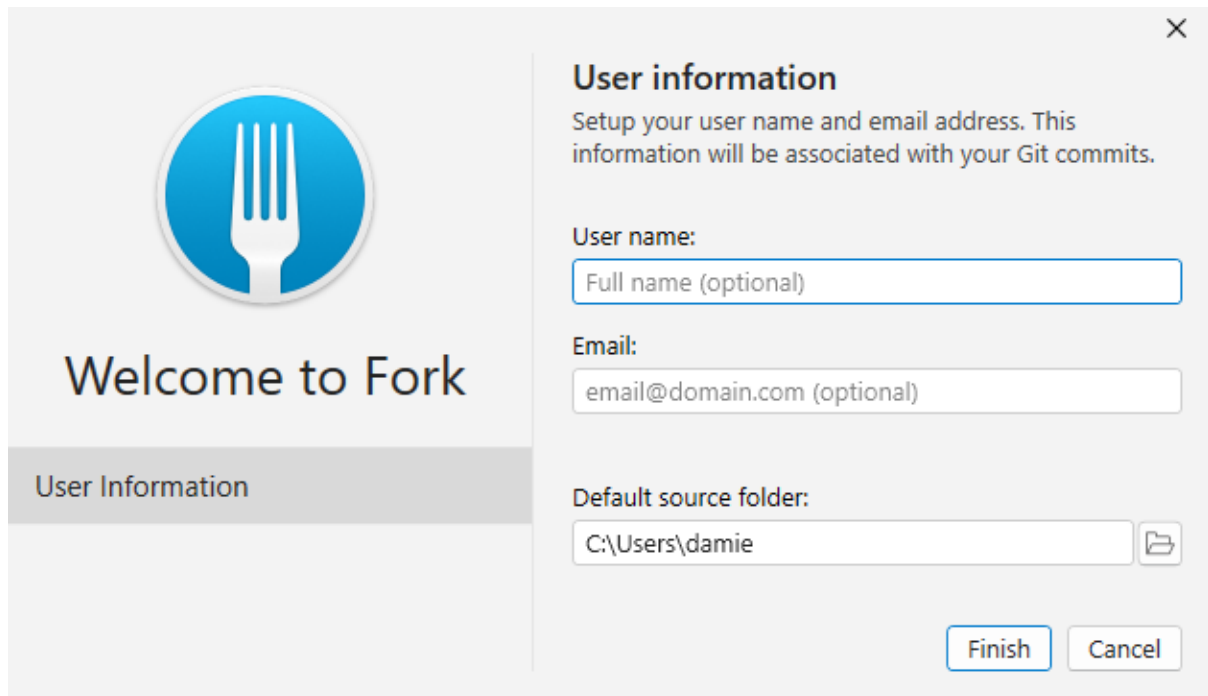
Installation d'une interface graphique Git, le logiciel Fork

Il est possible d'utiliser Git directement dans Eclipse mais les manipulations sont assez complexes¹. Pour rendre l'utilisation de git plus simple, nous vous proposons d'utiliser le logiciel Fork qui est une interface graphique Git.

Lien du logiciel à installer : <https://git-fork.com/>

Attention : Le logiciel s'installe automatiquement sur le disque C et non sur le disque H. Vous aurez donc à le réinstaller si vous changez d'ordinateur durant le projet. Néanmoins cela ne veut pas dire que vous perdrez votre travail si vous changez de PC.

¹ Pour information, si vous voulez utiliser Git sous Eclipse : passez en perspective Git (*Window -> Perspective -> Open perspective -> Other -> Git*). Ensuite dans les menus contextuels, vous avez également le sous-menu *Teams* qui permet de faire les *commit*, *update*, *push* et *pull* dans un projet où Git est utilisé.



Entrez ici votre nom-prénom et votre email et le répertoire que fork doit prendre par défaut pour chercher des répertoires clonés (ce n'est pas très important).

Créer son compte Gitlab UBO

Vous devez vous rendre sur le Gitlab de l'université (<https://gitlab-depinfo.univ-brest.fr/>) et vous connecter avec vos identifiants ENT.

Initialisation du projet sur Gitlab **(uniquement le chef de projet)**

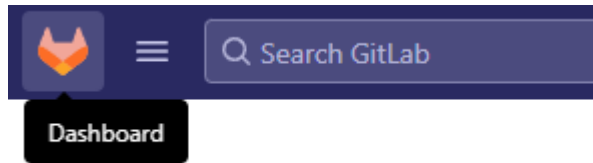
Il est conseillé que vous fassiez la partie initialisation du projet Gitlab en groupe afin que vous appreniez tous à le faire. Néanmoins cette partie devra être assurée sur un seul poste et un seul compte Gitlab.

Cette partie concerne plusieurs étapes :

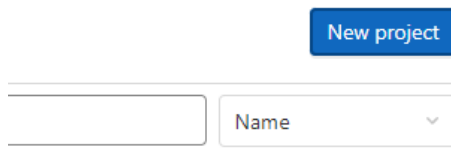
- Création du projet Git qui contiendra votre projet Java
- Ajout des autres membres du groupe au projet Git distant avec les droits suffisants
- Clonage (commande *clone*) du projet git distant avec Fork sur votre PC
- Ouverture du projet Git local (le projet Git distant que vous venez de cloner sur votre PC) en tant que workspace sur Eclipse
- Création du projet Java depuis Eclipse
- Indiquer à Fork de ne jamais envoyer certains fichiers générés par Eclipse au répertoire distant
- Sauvegarde des modifications faites sur le répertoire local (commande *commit*)
- Envoie du commit tout juste réalisé au répertoire distant (commande *push*)

Création du répertoire distant qui contiendra le projet java :

Allez sur votre liste de projet en cliquant sur l'icône en haut à gauche.




Créez un nouveau projet


The image shows the "New project" form in GitLab. At the top is a blue button labeled "New project". Below it is a form with a text input field and a dropdown menu labeled "Name".

Sélectionnez l'option *Create a project* et *Blank project*


Create new project




Create blank project
Create a blank project to store your files, plan your work, and collaborate on code, among other things.



Create from template
Create a project pre-populated with the necessary files to get you started quickly.



Import project
Migrate your data from an external source like GitHub, Bitbucket, or another instance of GitLab.



Run CI/CD for external repository
Connect your external repository to GitLab CI/CD.

You can also create a project from the command line. [Show command](#)

Indiquez le nom du projet, laissez les autres informations telles-queelles et créez le projet.

New project > Create blank project

Project name

monSuperProjet

Invitez les membres de l'équipe en cliquant sur *Invite members*.

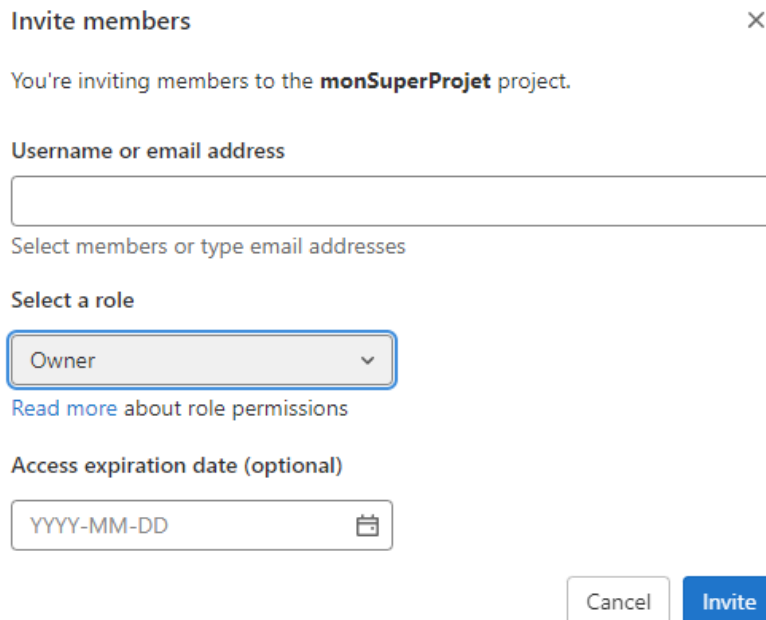
The screenshot shows the 'Create blank project' page in GitLab. At the top, the project name 'monSuperProjet' is entered in a text box. Below this, there's a section 'Invite your team' with a subtext 'Add members to this project and start collaborating with your team.' and a blue button labeled 'Invite members'. Further down, it states 'The repository for this project is empty' and provides options to 'Clone', 'Upload File', 'New file', 'Add README', 'Add LICENSE', 'Add CHANGELOG', 'Add CONTRIBUTING', 'Set up CI/CD', and 'Configure Integrations'.

Dans le cas où vous n'auriez pas cet affichage, vous pouvez aller dans *Project information*, *Members* puis sur le bouton bleu *Invite members*.

The screenshot shows the GitLab sidebar for the project 'monSuperProjet'. The sidebar includes a search bar at the top. Below it, there's a list of navigation items: 'Project information', 'Issues' (with 0 issues), 'Merge requests' (with 0 merge requests), and 'CI/CD'. A dropdown menu is open, showing 'Activity', 'Labels', and 'Members'.

Vous pouvez ici rechercher l'ensemble des utilisateurs du Gitlab, ajoutez les membres de l'équipe et assurez vous de leur assigner le rôle *Owner*. Dans un projet classique, on donnerait plutôt le rôle de *Developer* aux membres de l'équipe de développement mais cela rajoute la nécessité d'utiliser différentes branches dans le projet, qui est une complexité inutile pour le moment.

Les comptes Gitlab UBO n'étant créés qu'à la première connexion, vous devez attendre que les membres se soient connectés à la plateforme pour pouvoir les inviter sur le projet.



Invite members

You're inviting members to the **monSuperProjet** project.

Username or email address

Select members or type email addresses

Select a role

Owner

[Read more](#) about role permissions

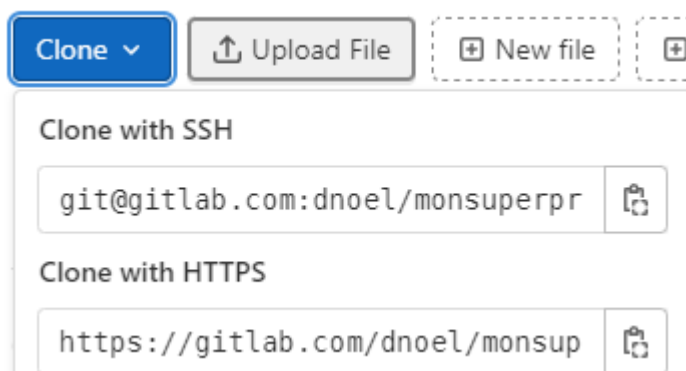
Access expiration date (optional)

YYYY-MM-DD

Cancel Invite

1.3 Clonage (commande clone) du projet git distant avec Fork sur votre PC

Sur le site officiel de Gitlab, on peut cliquer sur le bouton bleu *Clone* pour avoir la possibilité de récupérer l'URL du projet.



Clone

Upload File

New file

Clone with SSH

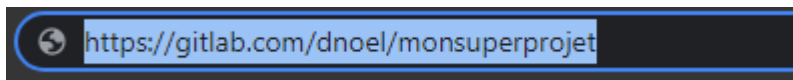
git@gitlab.com:dnoel/monsuperpr

Clone with HTTPS

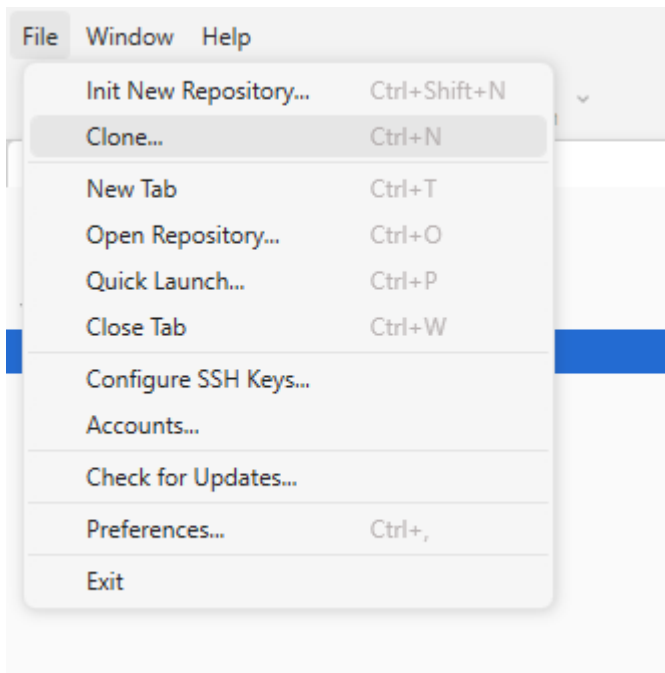
https://gitlab.com/dnoel/monsup

Néanmoins sur le Gitlab de l'UBO cette URL n'est pas bonne

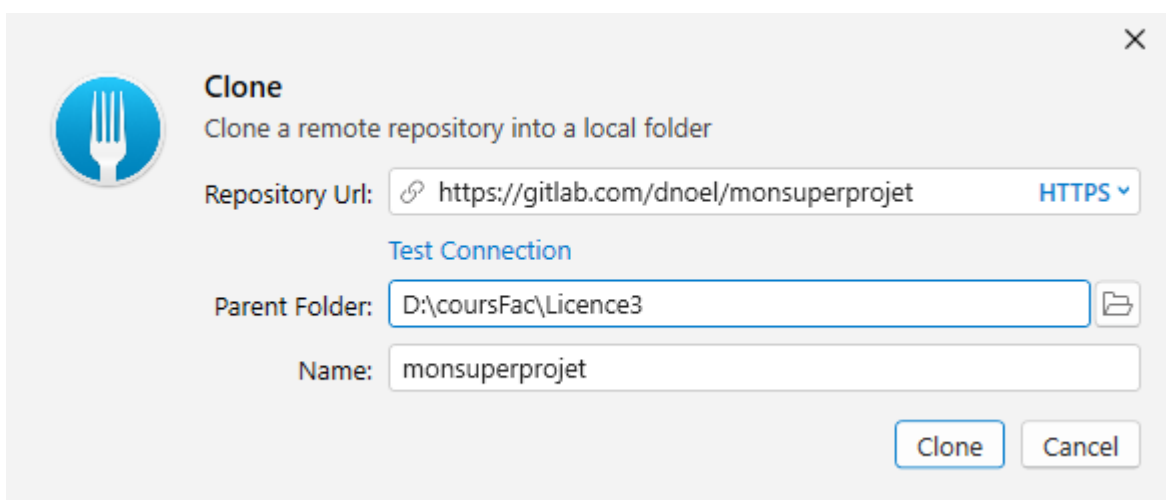
Vous devez donc copier directement l'URL dans le navigateur lorsque vous êtes sur la page d'accueil de votre projet.



Allez ensuite dans Fork, puis *File* en haut à droite et *Clone*



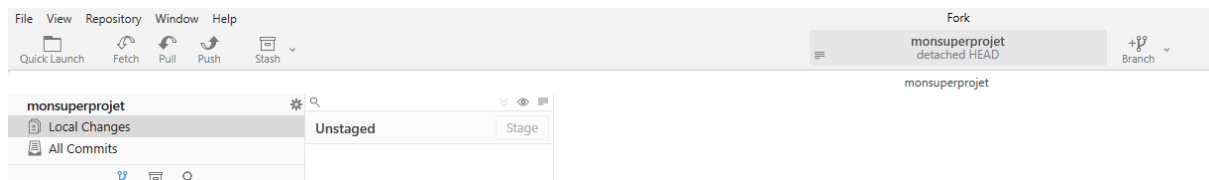
Entrez l'URL dans le formulaire, indiquez le répertoire où vous voulez cloner votre projet.



Vous pouvez avoir à entrer vos identifiants de connexion au Gitlab UBO (identifiants ENT) si une configuration Git n'existe pas déjà.

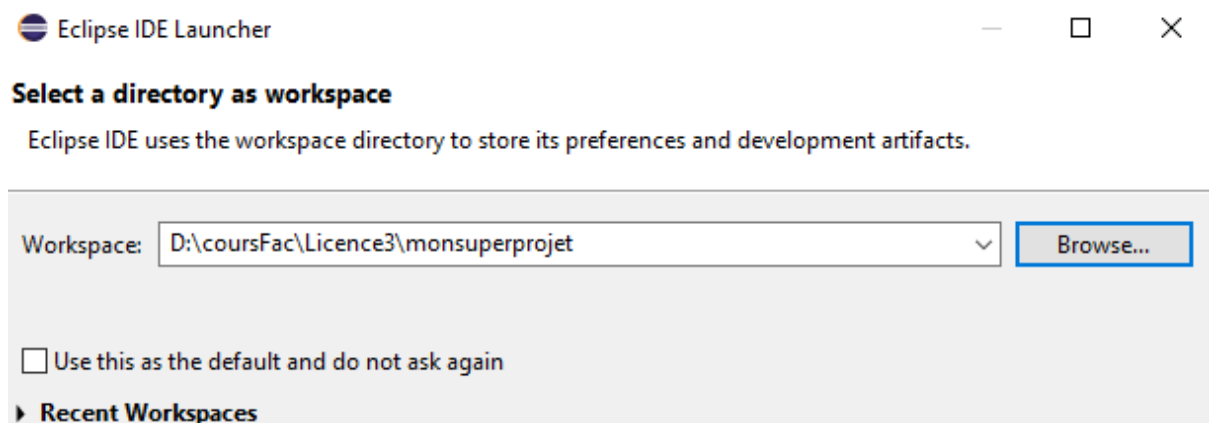
Après avoir cloné le projet, assurez-vous que Fork soit bien ouvert dessus.

Vous aurez alors deux onglets horizontaux en haut à gauche, un nommé *Local Changes* qui permet de voir vos modifications faites sur le projet local et d'actualiser le projet distant avec des modifications. Le deuxième onglet est l'onglet *All Commits* qui permet de visualiser toutes les modifications réalisées par les différents membres de l'équipe.



1.4 Ouverture du projet Git local (le projet Git distant que vous venez de cloner sur votre pc) en tant que workspace sur Eclipse

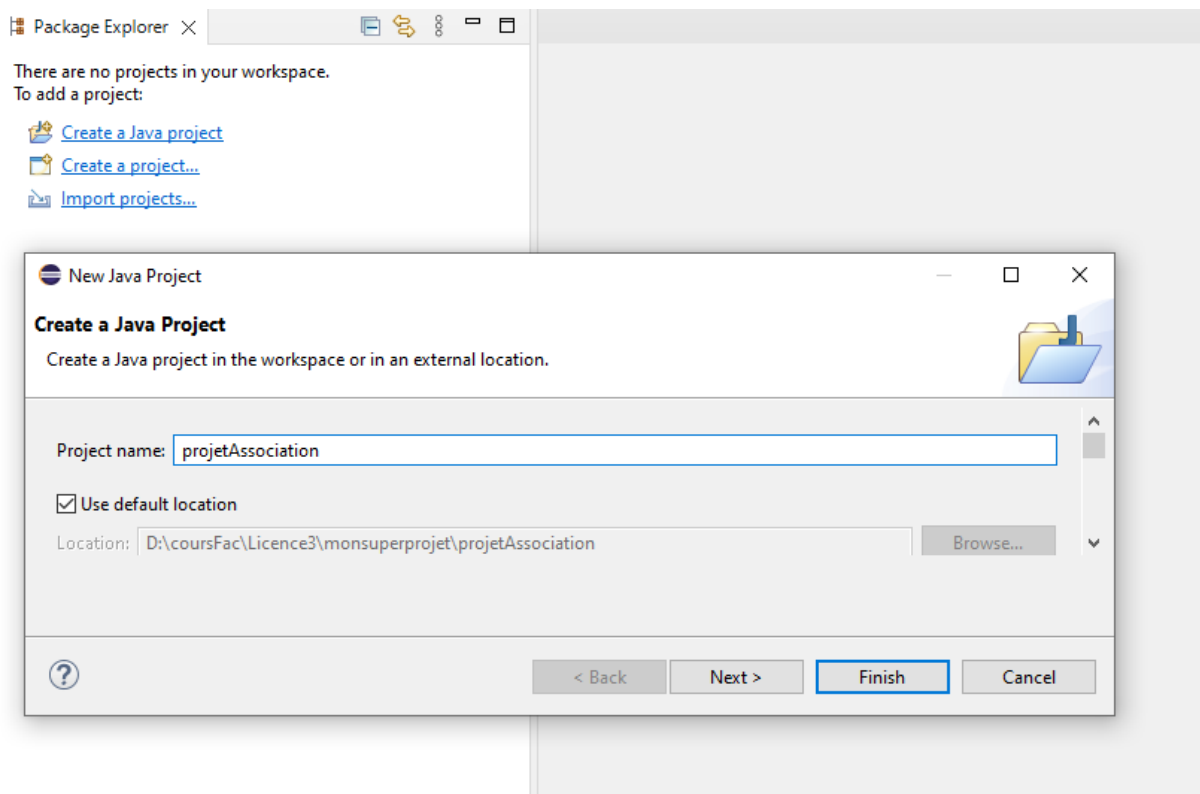
Lancez Eclipse et indiquer le projet Git local que vous venez de créer comme le workspace que eclipse doit utiliser puis lancer eclipse



1.5 Création du projet java depuis Eclipse (**uniquement le chef de projet**)

Allez dans le package explorer et cliquez sur *Create a Java project*.

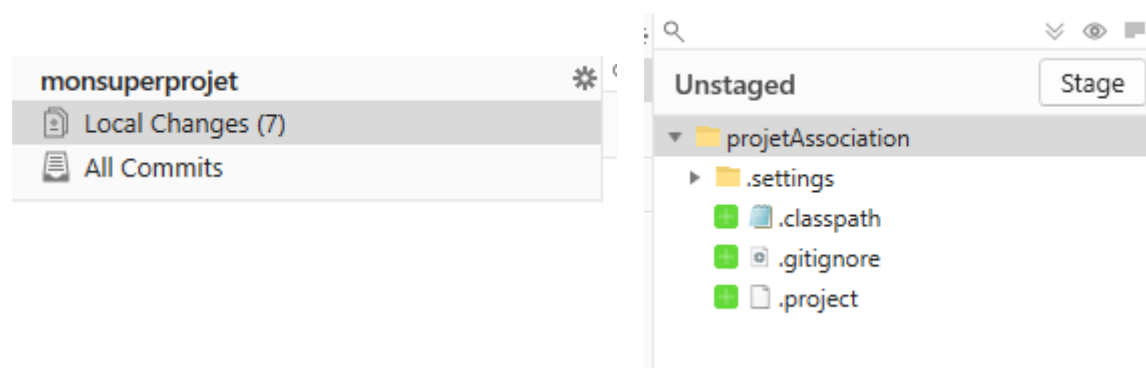
Indiquez le nom du projet java, le JRE que vous souhaitez utiliser et cliquez sur *finish*.



1.6 Indiquer à Fork de ne jamais envoyer certains fichiers générés par eclipse au répertoire distant.

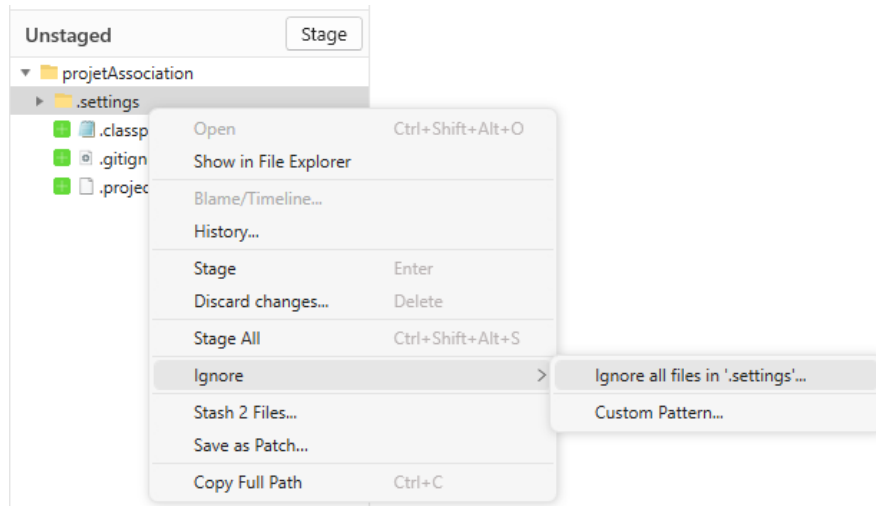
Lorsque l'on veut ajouter du contenu à un projet Git pour la première fois, il arrive souvent que l'on ne veuille jamais envoyer certains fichiers au projet distant. C'est le cas avec certains fichiers générés par Eclipse.

Retournez sur Fork et regardez vos modifications dans l'onglet Local Changes

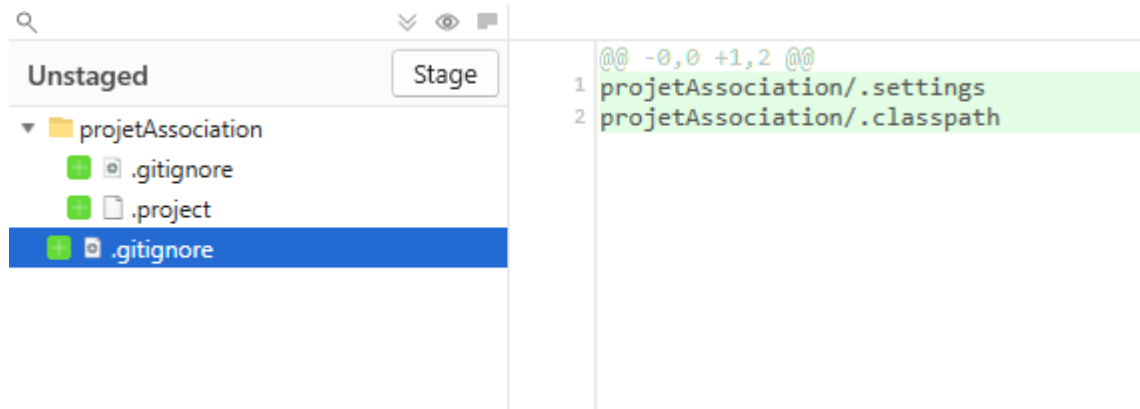


Le but ici est d'indiquer à Fork d'ignorer les changements faits sur le dossier *.settings* ni sur le fichier *.classpath* qui sont des fichiers propres à la configuration Eclipse de chaque membre du projet. Pusher ces fichiers pourrait entraîner des problèmes pour les collègues lorsqu'ils récupéreront vos changements.

Cela se fait par le fait d'ajouter le chemin des fichiers non-voulus au fichier `.gitignore`. Vous pouvez pour ça faire un clic droit sur le fichier à ignorer et sélectionner *Ignore* puis *Ignore all files*.



Vous devriez après ça ne plus voir dans Fork `.settings` ni `.classpath` dans les changements apportés au projet Git local. Vous devriez de plus avoir dans un `.gitignore` le chemin de `.settings` et `.classpath`.



1.7 Sauvegarde des modifications faites sur le répertoire local (commande commit).

Vous devez maintenant sauvegarder vos changements sur Fork pour pouvoir les envoyer au projet Git distant.

Pour cela vous devez indiquer à Fork quels changements vous voulez valider en les basculant de la case *Unstaged* à la case *Staged*. Pour cela vous pouvez sélectionner le dossier `projetAssociation` et cliquer sur le bouton *Stage*. Vous pouvez aussi simplement double-cliquer sur le dossier pour le stage.

Indiquez un message de commit avant de cliquer sur le bouton en bas à gauche pour commit les changements dans *Staged*.

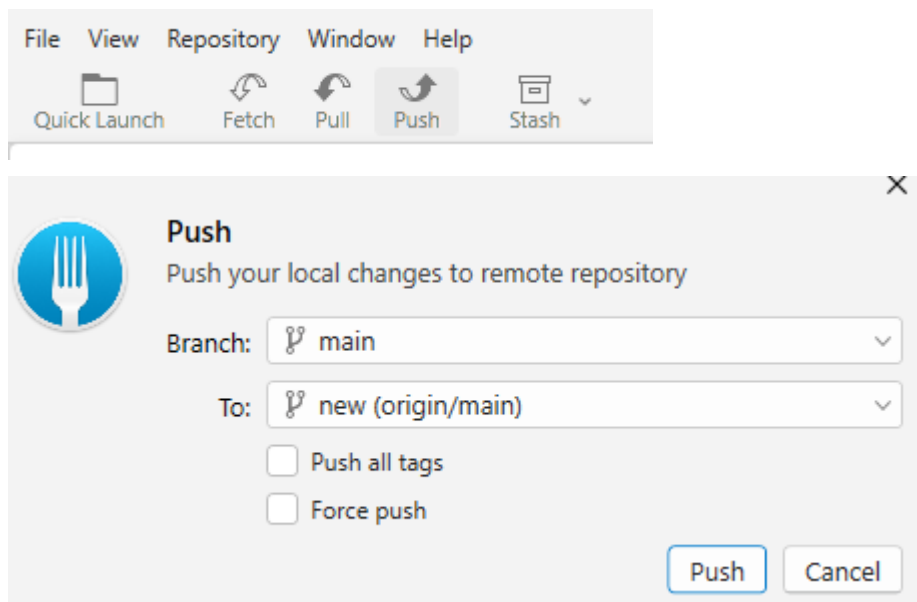


1.8 Envoi du commit tout juste réalisé au répertoire distant (commande push)

Rendez-vous dans la liste des commit en cliquant sur l'onglet *All Commits*. Vérifiez que vous voyez bien votre commit.



Lorsque votre commit n'a pas un logo de renard à sa gauche, cela veut dire que vous n'avez pas encore push le commit, vous pouvez le push cliquant sur le bouton *Push* en haut à gauche.

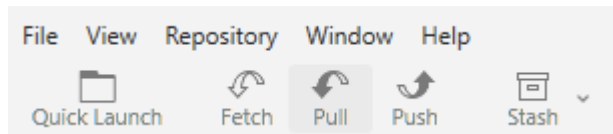


Vos informations de connexion au Gitlab peuvent vous êtres demandées.



Vous devriez désormais avoir un logo de renard à gauche du logo validé, cela indique que vos modifications sont bien présentes sur le serveur.

Pour récupérer les commits des autres membres de l'équipe, vous utiliser la commande *Pull*.



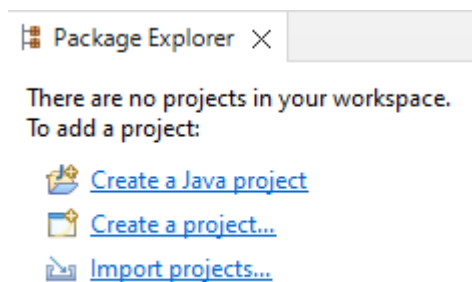
1.9 opérations à mener pour les développeurs (**autre que chef de projet**)

Vous avez déjà dû faire les opérations 1.2 correspondants à l'installation de Fork et la création d'un compte GitLab.

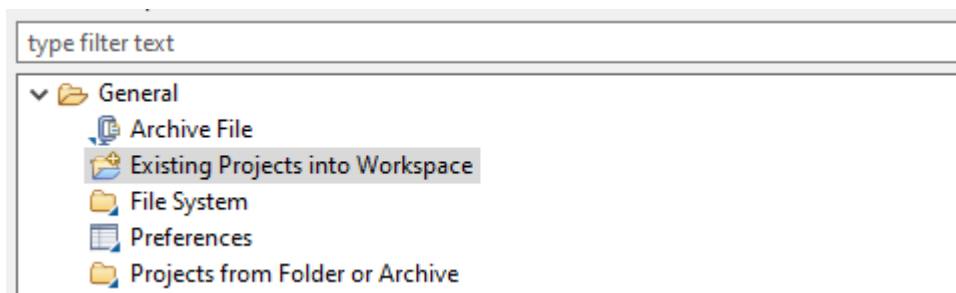
Vous devez maintenant faire les opérations 1.3 et 1.4

Vous devez ensuite importer le projet contenu dans le répertoire que vous venez de cloner.

Pour cela allez dans le package explorer et cliquez sur *Import project*



Vous cliquez ensuite sur *General -> Existing Projects into Workspace*



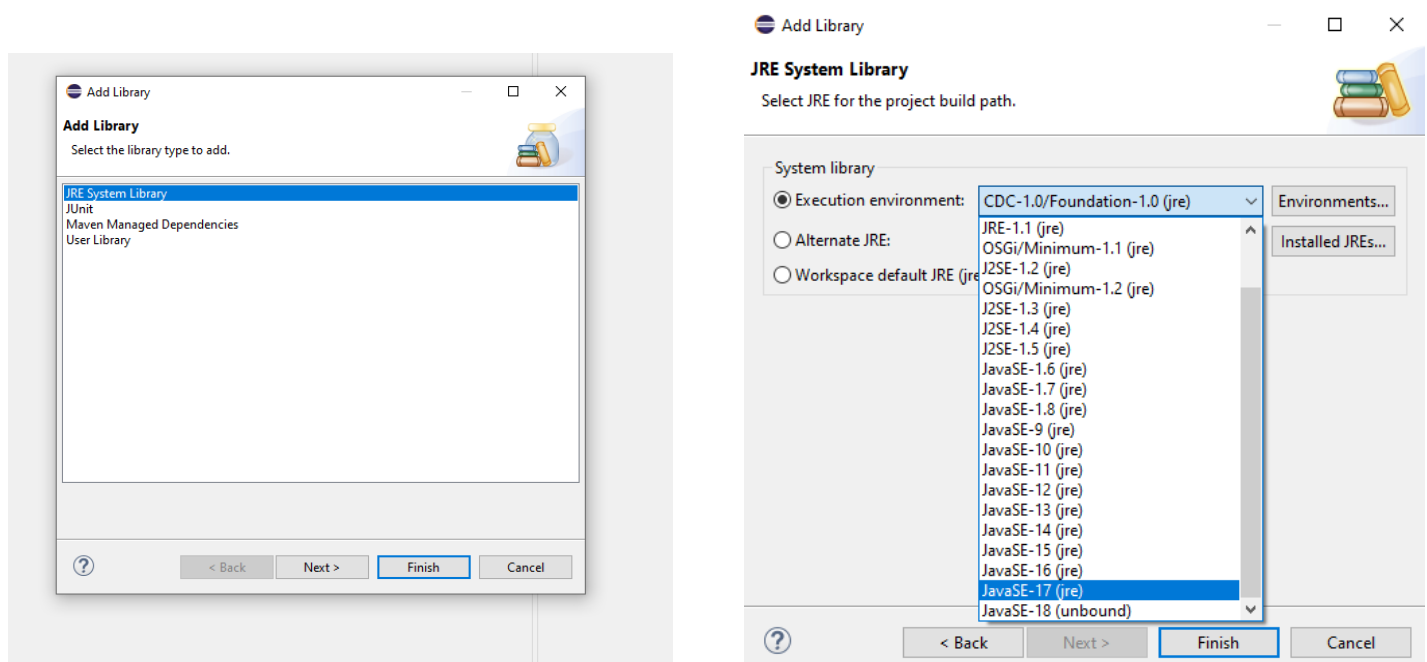
Vous sélectionnez ensuite le dossier *projetAssociation* à l'intérieur du répertoire git que vous avez cloné précédemment. Vous pouvez cliquer sur *Finish*.

.git	14/11/2022 18:47	Dossier de fichiers
.metadata	14/11/2022 18:47	Dossier de fichiers
projetAssociation	14/11/2022 18:47	Dossier de fichiers

Il ne vous reste plus qu'à configurer votre projet sous Eclipse.

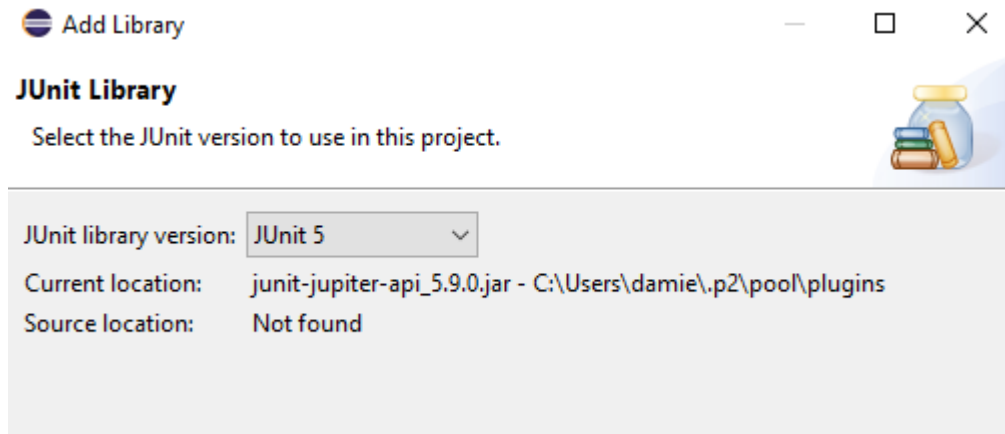
Vous devez pour cela ajouter au projet les librairies Java et Junit disponibles sur votre système.

Pour cela faites clic-droit sur votre projet java dans l'*explorateur de package* -> *Build Path* -> *Add Library* -> *JRE System Library* -> sélectionnez la partie *Execution environment* et sélectionnez la version de JRE la plus grande.

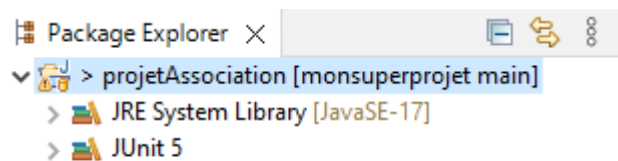


Ajoutez ensuite la librairie JUnit en faisant :

clic-droit sur votre projet java dans l'*explorateur de package* -> *Build Path* -> *Add Library*
-> *JUnit* et sélectionnez la version 5



Vous devriez donc avoir deux librairies ajoutées à votre projet dans l'explorateur de package.



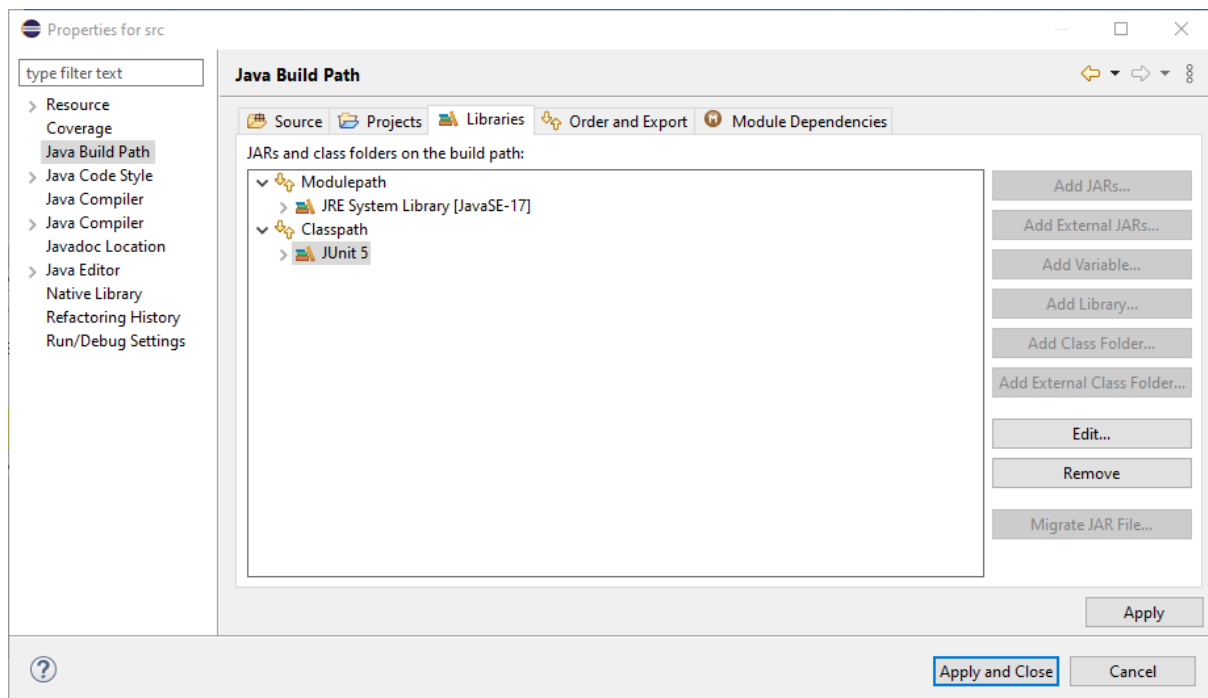
Note: cette méthode ne marche pas toujours

Si vous avez des sources qui apparaissent bizarrement comme src qui est considéré comme un package, il faut faire les configurations via : clic-droit -> *Properties* -> *Java Build Path*.

Sélectionnez *Module Path* puis cliquer sur *Add Library* pour rajouter Java 17.

Sélectionnez *Classpath* puis cliquer sur *Add Library* pour ajouter JUnit 5.

La fenêtre de configuration doit être la suivante :



1.9bis Ajout des sources du projet au projet Git (un seul étudiant)

Arrivé à ce point, chaque étudiant doit avoir un projet Git utilisable sous Eclipse. Il faut maintenant rajouter les sources fournies (src.zip) au projet Git. **Pour éviter des conflits, un seul étudiant (le chef de projet) devra faire la manipulation :**

- Décompresser l'archive src.zip et placer tout le contenu du répertoire src de l'archive dans le répertoire src du projet Eclipse
- Si vous avez une erreur sur la classe TestInformationPersonnelle, c'est normal, c'est parce que JUnit n'est pas encore configuré (cf section suivante)
- Passer sous Fork, mettre tout le contenu du répertoire src en *Stage* puis exécuter *Commit* et *Push*

Un fois que cela est fait par le chef de projet, les autres étudiants peuvent faire un *Pull* sous Fork pour récupérer le code rajouté au projet Git.

Si vous souhaitez récupérer les modifications uploadées sur le projet distant, il faut utiliser la commande *fetch* qui permet de récupérer et appliquer directement les modifications à votre projet local.



Attention, assurez-vous d'avoir fait un commit de vos modifications locales importantes sous peine de perdre votre travail en utilisant la commande fetch.

Si vous souhaitez pouvoir visualiser les modifications faites par vos collègues sans modifier votre travail en local, vous pouvez utiliser la commande pull qui n'écrase pas vos modifications.

1.10 Ajout des modifications au projet Git

Référez-vous à la partie 1.7 pour l'ajout de vos modifications au répertoire Git distant.

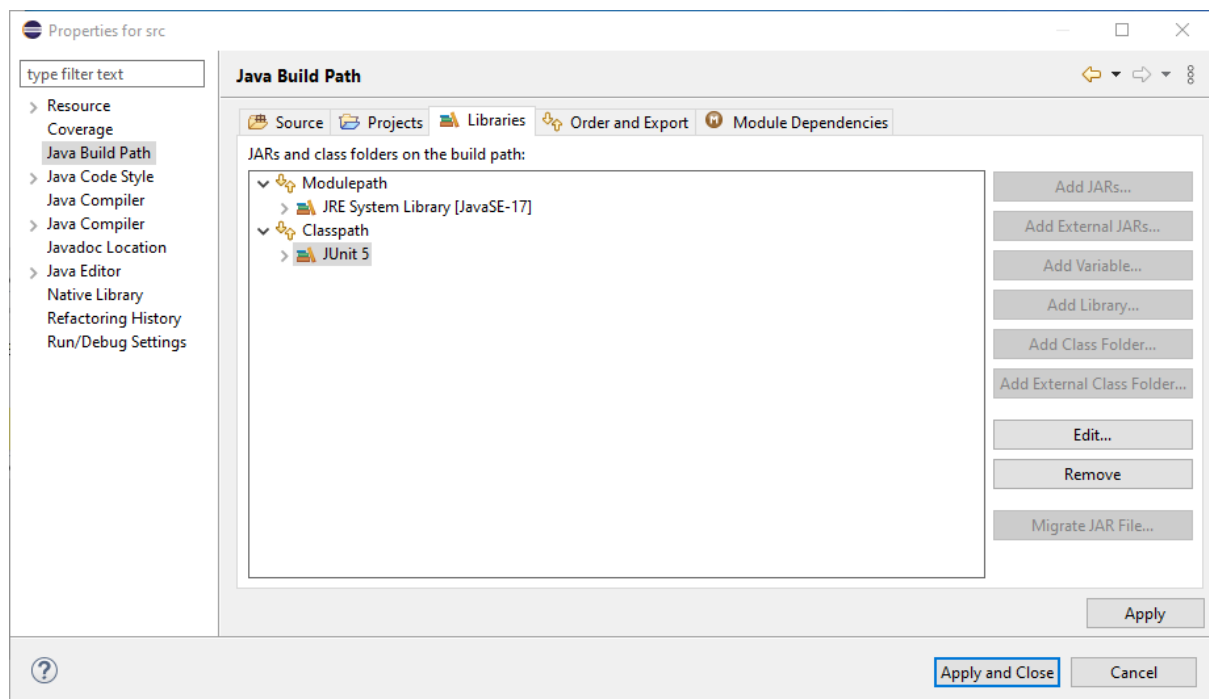
Il se peut que vous ne puissiez pas push vos modifications si vous n'avez pas récupéré les dernières modifications faites par vos collègues sur le répertoire distant. Dans ce cas vous n'avez qu'à faire un fetch avant de faire push de nouveau.

2. JUnit

JUnit permet de faire des tests unitaires (et d'autres types de tests) sur du code Java. Le principe général est de créer une classe qui contient des méthodes de tests qui effectuent des actions sur des objets et vérifient ensuite qu'une condition est valide (l'oracle du test). Ces tests ont pour objectif d'assurer que le code implémenté remplit correctement les fonctionnalités attendues.

Configuration de JUnit dans le projet Java sous Eclipse

Sélectionnez le nom du projet dans l'explorateur à gauche, cliquez sur *Properties*. Dans la fenêtre, sélectionnez *Java Build Path* puis l'onglet *Libraries*. Sélectionnez *Classpath* puis à droite cliquez sur *Add Library* : sélectionnez JUnit en version 5. Le résultat doit être le suivant :



Création et exécution de tests JUnit

Par convention, les tests sont placés dans un répertoire nommé tests se trouvant dans src. Ce répertoire existe dans les sources que vous avez récupérées et contient déjà une classe de tests : *TestInformationPersonnelle.java*

Une classe de test est nécessairement publique. Dans une classe de test (depuis JUnit 4), on peut ajouter une balise devant une méthode (nécessairement déclarée comme *public*), dont les plus importantes sont (cf. fichier *TestInformationPersonnelle.java*) :

- *@BeforeEach* (*@Before* pour JUnit 4) : cette méthode est exécutée avant chaque test
- *@AfterEach* (*@After* pour JUnit 4) : cette méthode est exécutée après chaque test

- *@Test* : cette méthode correspond à un test JUnit

A l'intérieur d'une méthode de test, il faut ajouter une condition de test pour déterminer si le test est OK ou pas. Les conditions les plus importantes sont :

- *assertEquals(uneVariable, uneValeur);* : le test est ok si *uneVariable* est égal à *uneValeur*
- *assertTrue(uneVariableBooléenne);* : le test est ok si *uneVariableBooléenne* est égal à *true*, *false* sinon

On peut ajouter plusieurs conditions de test dans une méthode de test. Dans ce cas, le test est considéré OK si toutes les conditions sont OK.

Exécuter un test JUnit

Clic droit sur une classe de test et *Run As / JUnit Test*

Le résultat est donné en nombre : nombre de tests effectués, nombre d'erreurs.

La liste des tests effectués est affichée avec le résultat pour chaque test :

- S'il y a une erreur, double clic sur un test permet d'accéder au test en échec
- Si pas d'erreur, tout est vert

Pour plus d'informations

<https://junit.org/junit4> et <https://junit.org/junit5>
<https://www.vogella.com/tutorials/JUnit4/article.html>

3. Javadoc

Javadoc est un utilitaire Java qui permet de générer une documentation au format HTML du code Java qui est développé. L'intégralité de la documentation de l'API du langage Java a été réalisée de cette façon : <https://docs.oracle.com/en/java/javase/17/docs/api/>

Concrètement, il s'agit de commenter le code Java en utilisant des commentaires formatés d'une façon particulière. On peut commenter tout le contenu d'une classe : décrire la classe elle-même, décrire chacun de ses attributs et chacune de ses méthodes. L'utilité principale de cette documentation est qu'un développeur utilisant une de vos classes doit comprendre ce qu'elle fait et comment l'utiliser sans avoir besoin de regarder son code. Par exemple, ici vous avez la documentation de l'interface `java.util.Set` qui permet de manipuler des collections de type ensemble en Java :

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Set.html>

Ecriture des commentaires

Un commentaire Javadoc commence par `/**`, ensuite chaque ligne du commentaire commence par `*` et la dernière ligne ferme le commentaire par `*/`

Le commentaire se place juste avant la définition de la classe, d'un attribut ou d'une méthode. Pour décrire une méthode, on définit des tags : `@param` pour décrire un paramètre, `@return` pour décrire la valeur de retour et `@exception` pour décrire une exception qui peut être levée.

Sous Eclipse, placez le curseur la ligne au-dessus de la définition d'une méthode, tapez `/**` et quand vous appuyez sur Entrée, Eclipse vous génère automatiquement les balises correspondant à la méthode. Il vous reste alors à écrire les commentaires.

Dans le code qui vous est donné, toutes les interfaces et la classe `InformationPersonnelle` sont commentées en respectant le format Javadoc.

La description d'un package se fait d'une façon un peu particulière puisqu'un package Java est simplement un répertoire. Il faut créer un fichier nommé `package-info.java` dans le répertoire du package pour y placer le commentaire sur le package. Regardez le contenu de ce fichier dans les packages association et tests du code qui vous est donné.

Génération de la documentation

Pour générer le HTML correspondant à votre code Java : menu *Project* puis *Generate Javadoc*. Sélectionnez le projet et les packages de code source dans l'explorateur à gauche. Vous pouvez changer l'endroit où la documentation est générée. On vous propose aussi de

choisir la visibilité qui correspond aux visibilités que vous connaissez en Java : *private*, *protected*, *public* ...

Si vous sélectionnez *private*, la documentation de l'intégralité de votre code sera générée. Si vous sélectionnez *public*, seule la documentation sur les parties publiques (classes, attributs et méthodes) de votre code sera générée. C'est généralement ce mode là que l'on utilise. En effet, le but de la Javadoc est de comprendre comment utiliser votre classe si par exemple vous définissez une librairie. Dans ce cadre, le programmeur se contente d'appeler les méthodes publiques de vos classes mais sans toucher à leur code. Il n'a donc pas accès aux parties *private* ou *protected* de votre code et seule la documentation de la partie public lui est utile.

Cela dit, dans le contexte du projet de l'UE, vous pouvez générer toute la documentation donc sélectionnez la visibilité *private*.

Cliquez sur *Finish* pour générer le code HTML et ouvrez le dans un navigateur Web pour voir le résultat (fichier *index.html* à la racine).

Pour plus d'informations, notamment la liste des balises Javadoc :

<https://www.jmdoudoux.fr/java/dej/chap-javadoc.htm>

4. Checkstyle

Checkstyle (<https://checkstyle.sourceforge.io/>), à l'instar d'autres outils comme SonarQube, sert à définir et gérer la qualité de code. Avoir un code de qualité correspond à respecter des critères de présentation du code (formatage, conventions de nommage...) et de complexité du code (nombre d'attributs dans une classe, nombre de lignes de code d'une méthode...). L'objectif est d'avoir un code uniformément présenté, plus simple à lire et à comprendre afin de pouvoir le maintenir plus facilement.

Installation

Si vous utilisez les PC des salles de TP, Checkstyle est déjà installé sous Eclipse. Sinon, il faudra le rajouter à votre Eclipse :

- *Help* -> *Install New Software*
- Dans *Work with*, entrez : <https://checkstyle.org/eclipse-cs-update-site>
- Vous verrez apparaître Checkstyle avec une case à cocher devant : cochez là et cliquez sur *Next* pour démarrer l'installation

Vérification de l'installation/configuration de Checkstyle

Sélectionnez votre projet, menu *Project* d'Eclipse / *Properties* / *Checkstyle* -> *Checkstyle active for this project* et *Google Checks – (Global)* sélectionné dans la liste.

Configuration formatage code Java

Pour faciliter la mise en forme automatique de votre code, on va importer un format de code Java pour Eclipse. Sélectionnez votre projet / *Properties* / *Java Code Style* / *Formatter*

-> *Enable project specific settings* et *Import* avec le fichier *Formatter-Eclipse.xml*

Ensuite activez Checkstyle pour votre projet :

Sélectionnez votre projet / clic droit / *Checkstyle* -> *Activate Checkstyle*

Vérification du code du projet

[Ctrl]+[Shift]+[F] : mise en forme automatique selon le guide style importé

[Ctrl]+[s] : la sauvegarde (re)lance la vérification du code par Checkstyle

Le résultat est visible dans le code ou dans la vue Checkstyle violations


Pour plus d'informations

<http://objis.com/tutoriel-integration-continue-checkstyle-dans-eclipse/#partie1>

5. Divers sur Eclipse

Eclipse vous offre des fonctionnalités pour implémenter rapidement des parties de votre code. Par exemple pour une classe, il peut générer directement les accesseurs des attributs, des constructeurs et les méthodes `equals()`, `hashCode()` ou `toString()`. Vous pourrez ensuite les modifier au besoin.

Pour cela, sélectionnez une classe dans l'explorateur de fichiers d'Eclipse, clic droit puis *Source* et vous trouverez plusieurs choix pour générer du code dans les classes. Vous pouvez également préciser qu'une classe spécialise une autre classe ou implémente une interface (via ce menu ou à la création de la classe) et Eclipse définira le squelette des méthodes à implémenter.

Eclipse affiche également une petite ampoule  devant certaines lignes de code : en cliquant dessus, des suggestions de modification de code vous sont proposées, typiquement pour gérer des exceptions ou des imports.

6. Réalisation d'un exécutable

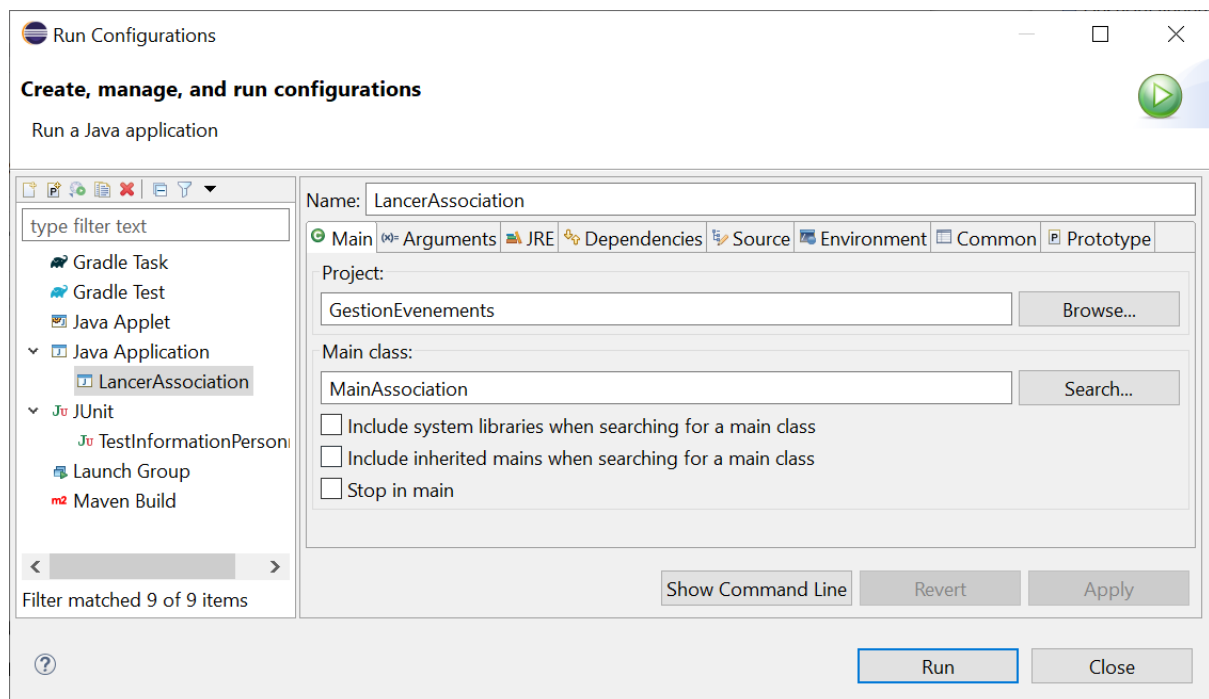
Le but final de votre projet est d'obtenir un exécutable que l'on peut lancer en dehors d'Eclipse. En Java, il faut fabriquer un fichier au format JAR (pour Java Archive) qui exécutera la méthode *main()* d'une des classes du projet.

Création d'une configuration d'exécution

Pour tester la création d'une configuration d'exécution, vous allez en associer une à la classe *MainAssociation* à la racine de *src* du code qui vous a été fourni et qui contient une méthode *main()* exécutable.

Dans l'explorateur de fichiers d'Eclipse, sélectionner ce fichier puis clic droit -> *Run As -> Run configurations ...*

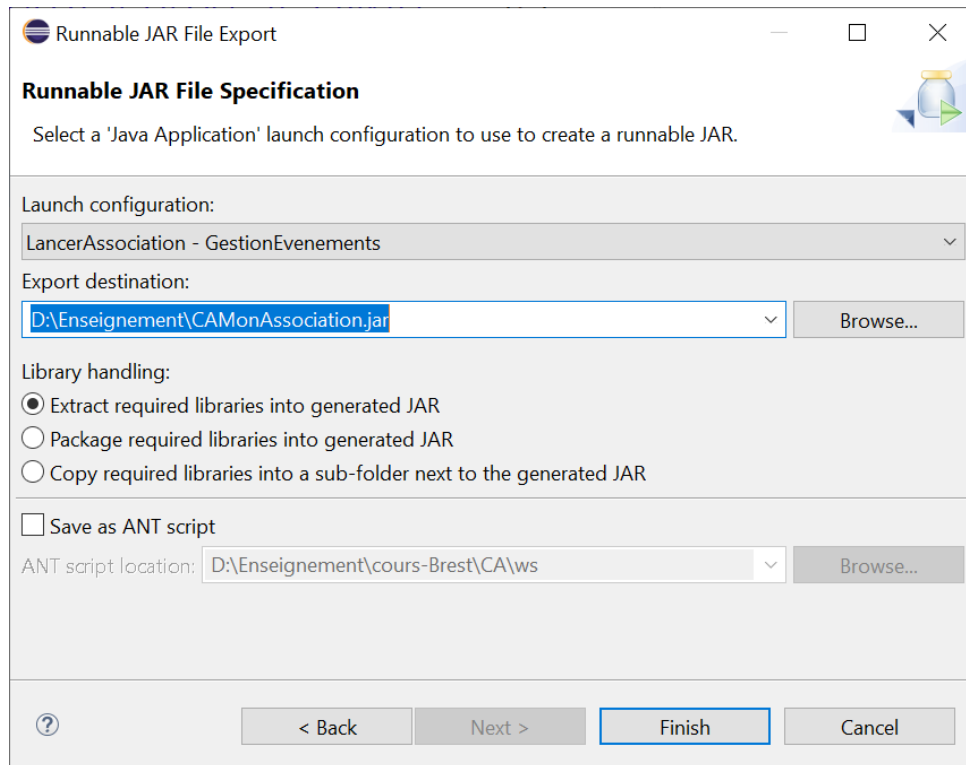
Dans la liste à gauche, double-cliquez sur *Java Application* pour créer une configuration Java. Changez le nom en haut à gauche comme vous le voulez en vérifiant que la *Main class* est bien celle qui contient la fonction *main()* à lancer, puis cliquez sur *Apply*. Ici, j'ai mis *LancerAssociation* comme nom :



Création du JAR

Une fois votre configuration réalisée, ouvrez le menu File d'Eclipse : *Export -> Java -> Runnable JAR file* puis *Next*.

Dans la fenêtre, choisissez la configuration que vous venez de créer. Préciser l'emplacement et le nom d'un fichier JAR :



Lancement du JAR

Le fichier JAR généré peut se lancer directement en cliquant dessus dans l'explorateur de fichier de votre système d'exploitation (sous Linux, lui rajouter préalablement les droits d'exécution). On peut aussi le lancer en ligne de commande dans un terminal :

```
$ java -jar CAMonAssociation.jar
```

7. Intégration d'une interface graphique

TODO