TD5 – Oracle – Outillage (Jean-Marie Mottu)

Première partie – reprise TD4 – Outillage

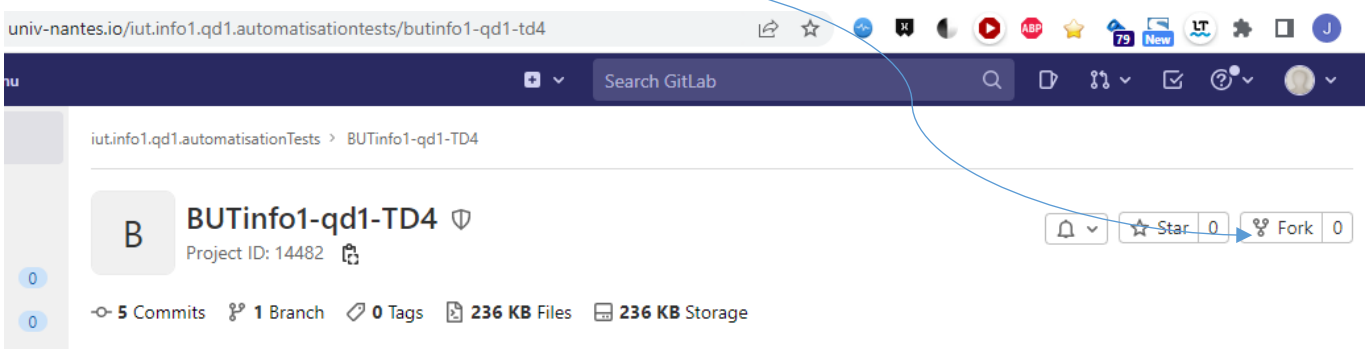
Exercice 1 – Exploitation d'un gestionnaire de version

Repartons du cas d'étude du TD4, la liste de notes.
Jusqu'à présent, nous n'avons pas fait pleinement usage de gitlab en tant que gestionnaire de version. Le prof s'en sert effectivement pour cela, mais votre usage était juste de récupérer un projet en clonant son dépôt. Ainsi, vous n'avez pas versionné, pas sauvegardé, pas partagé votre travail.

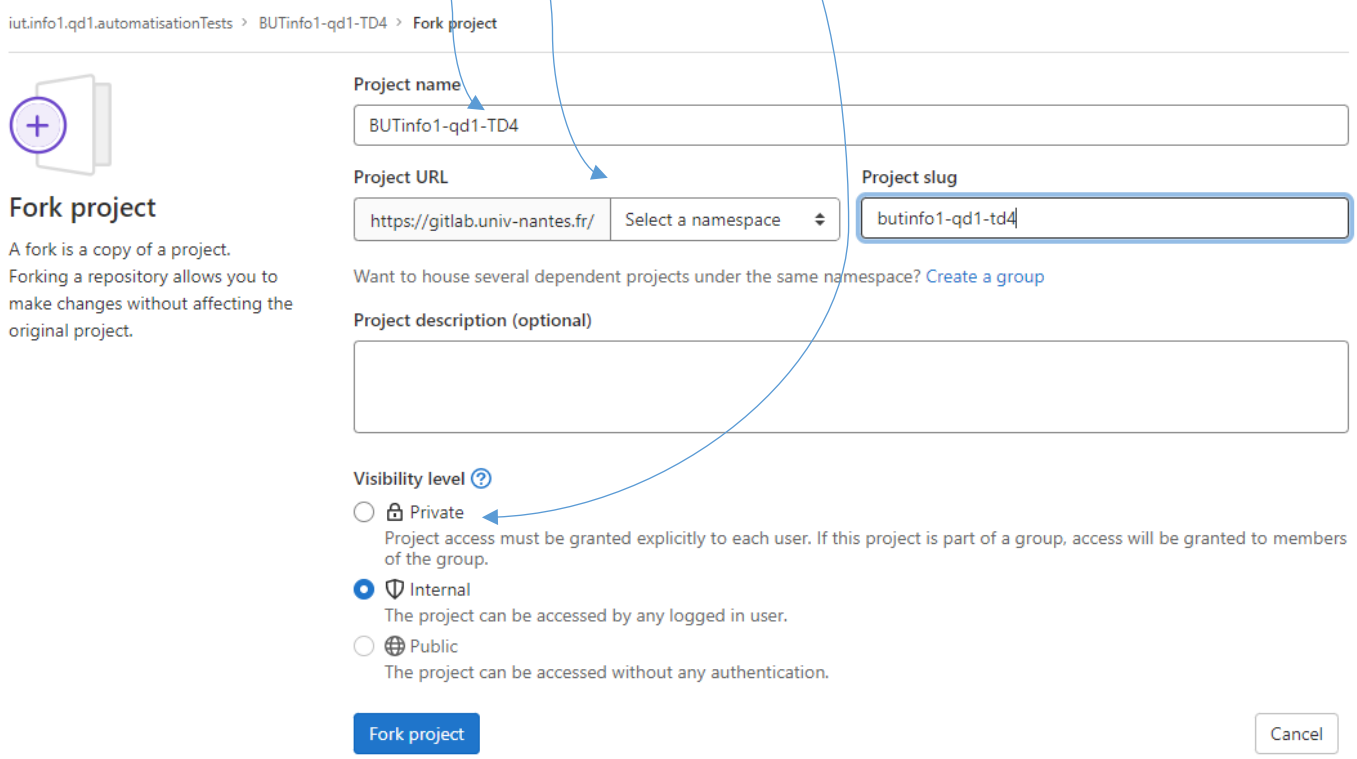
Reprenons ce que nous avons vu aux CM5 et TD7 du module R2.10 - Gestion de projet et des organisations. Travaillez en binôme (et seul trinôme par groupe de TD).

Question 5.1 Fork
Rendez-vous ici :
https://univ-nantes.io/iut.info1.qd1.automatisationtests/butinfo1-qd1-td4

Un étudiant par trinôme crée un fork du projet :



A l'écran suivant choisissez votre namespace (le nom de celui qui fait le fork), changez le TD4 en nomBinome1-nomBinome2-TD5, mettez le projet « private » (c'est votre travail personnel, à votre binôme) :



Finalement, invitez votre binôme (pour qu'il bosse aussi) et votre chargé de TD (pour qu'il note votre travail).

Members > Invite Member, cherchez leurs noms et donnez-leur les droits de « Maintainer ».

Tout le monde peut voir le projet dans sa liste de projet https://univ-nantes.io

Question 5.2 Clone

Chacun crée un projet IntelliJ en clonant son projet.

Question 5.3 Recréez la classe de test :

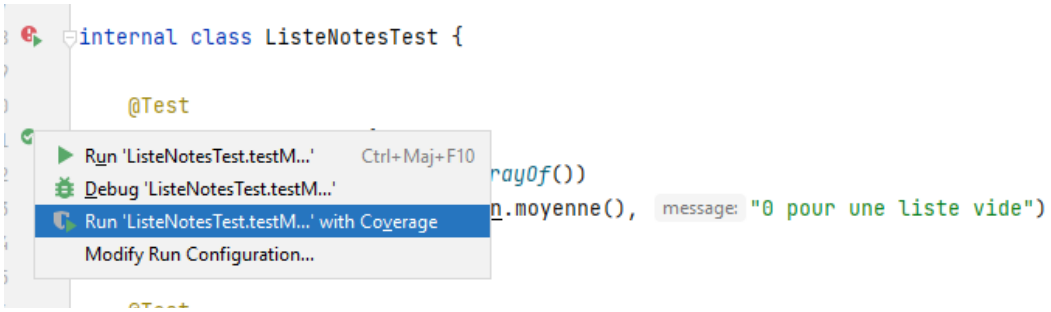
Le binôme 1 s'occupe des tests de la fonction calculant la moyenne, l'autre des tests de la fonction calculant le nombre d'occurrence. Faites-le dans la même classe (pour un peu plus de challenge)

Question 5.4 : Versionnez, partagez votre travail avec votre binôme

Exercice 2 – Diagnostic outillé

Etant reparti d'un fork du projet intial, le code sous tests est bien faux et certains tests échouent. Reprenons les phases de diagnostic à l'aide d'IntelliJ.

Question 5.5 Exécutez les cas de test un à un en affichant la couverture de code
Pour cela, cliquez sur le « play » dans la marge au niveau de la signature de la méthode de test et faites « run … with Coverage »



Revenez dans la classe sous test et observez dans la marge de la méthode sous test le vert et le rouge indiquant quelles instructions ont été ou n'ont pas été exécutées.

Question 5.6 Vérifiez dans vos matrices de diagnostic si les couvertures correspondent à ce que vous aviez fait manuellement.

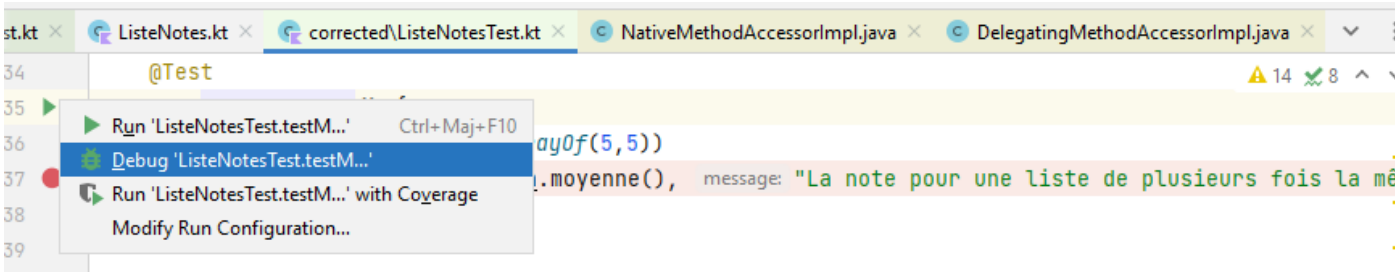
Exercice 3 – Correction outillée

Une fois que l'algorithme de diagnostic a classé les instructions dans l'ordre de leur potentialité d'être fausses ou faussement exécutées à cause d'un faux prédicat, il faut comprendre et corriger la faute.

Pour cela, le mode debug de l'IDE est indispensable.
On évitera de décorer le code : le testeur n'a pas le droit de faire du temporaire au risque qu'il soit définitif (ou masque des fautes, ce qui est improbable avec des println, mais ils perturbent quand même les performances) :
https://www.francetvinfo.fr/decouverte/bizarre/un-radar-disait-fuck-you-aux-automobilistes-qui-roulaient-trop-vite_84555.html

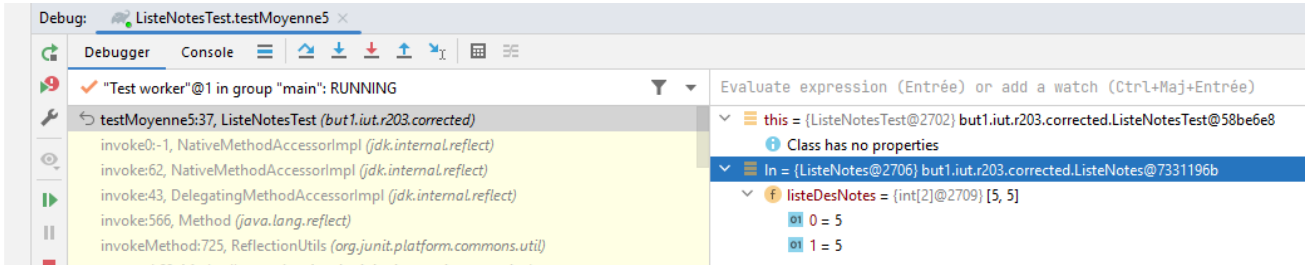
Il s'agit d'une forme de traçabilité dynamique, pas à pas.

Question 5.6 Commencez par ajouter un point d'arrêt dans le cas de test 3 de la fonction moyenne, à la ligne de l'appel à la méthode sous test :
Ce point d'arrêt est nécessaire pour dire où commencer le diagnostic.
Un clic dans la marge ajoute ce point rouge, puis lancez le test en « debug » :



Dans un onglet « debugger », vous obtenez la pile d'appel : tous les appels de méthodes qui sont imbriqués depuis le début de l'exécution du programme (un main qu'on a même pas lancé nous même mais qui l'a été par le framework de test) jusqu'à la méthode de test où l'on a mis le point d'arrêt.

Dans l'onglet « evaluate expression », vous obtenez des informations sur l'état du système :



La classe n'a pas d'attribut (properties) mais la liste est bien créée et remplie.

Question 5.7 Avancez pas à pas.
On peut avancer par instruction « step over », ou par appel « step into ».
Progressez jusqu'à trouver l'instruction fausse et constatez le problème au deuxième tour de boucle.
La lecture est difficile, un IDE comme IntelliJ montre même ses limites : une seule variable somme est listé alors que le problème est d'en affecter une nouvelle à chaque tour de boucle au lieu de mettre à jour celle de la fonction.

Question 5.8 Placez des points d'arrêt selon le résultat de l'algorithme de SBFL pour la fonction qui compte le nombre d'occurrence :
Il faut en placer plusieurs :

- Un à l'instruction désignée première, et potentiellement fausse
- D'autres aux prédicats y menant et qui potentiellement sont faux en menant à cette instruction.

Question 5.9 Avancez pas à pas.

Deuxième partie –Oracle

Exercice 4 – Oracle heuristique

Pour le moment vous avez conçu vos cas de test avec des oracles discrets que vous avez implémentés en contrôlant les valeurs exactes dans des assertions.

Question 5.10 : Considérons les fonctions `moyenne, nombreOccurence`, pour chacune d'entre elle concevez un oracle heuristique.

Par exemple pour la fonction `factorielle`, un oracle heuristique serait factorielle(x) >=0 en exprimant une propriété sur n'importe quelle sortie. Ainsi pour n'importe quelle donnée de test, on peut utiliser cet oracle.
On peut aussi faire des oracles heuristiques qui lient entrée et sortie : une propriété sur une entrée implique une propriété sur la sortie. Dans le CM, nous avions : pour x entre 0 et PI, alors 0<=`sin`(x)<=1.

Question 5.11 : Considérons `factorielle,` identifiez une relation métamorphique

Troisième partie – Développement et test collaboratif

Exercice 5 – Développement

Question 5.12 : Chaque binôme implémente sur son ordi, sans concertation une de ses deux méthodes :

- Un binôme : Quelle est la note maximale et combien d'élèves l'ont
- L'autre binôme : Quelle est la note minimale et combien d'élèves l'ont

Exercice 6 – Tests fonctionnels de Min/Max

Les binômes développeurs prennent le rôle de testeur pour tester le code de l'autre.

Question 5.13 : Chacun conçoit des tests fonctionnels pour la fonction qu'il teste :

Il faut un minimum de spécification pour tester ces méthodes :
Les notes sont des entiers. La note maximale étant 20 et la minimale 0.
La liste de notes n'est pas ordonnées, elle contient une liste de notes qu'on imagine entrées par le professeur au fur et à mesure des copies corrigées. Chaque note est donc la note d'un élève.

Question 5.14 : Implémentez ces cas de tests.

Question 5.15 : Exécutez ces cas de tests.

Si vous avez tous vos tests qui passent du premier coup, bravo.

Exercice 7 – Diagnostic de Min et Max

Question 5.16 : Effectuez le diagnostic et la correction de vos méthodes grâce à l'algorithme de Jones avec l'aide de l'IDE pour la couverture et le mode débug.