

## TD4 – Doublures de test (Jean-Marie Mottu)

Nous finissons de travailler sur le cas d'étude d'une classe Chien et reprendrons ensuite une méthode de calcul. Nous

### Partie 1 – Transition TD3 – TD4

#### Exercice 4.1. Injection de dépendance de classe (20 min)

**Question 4.1.1.** Reprenez les questions 3.7.1, 3.7.2 de l'exercice 3.7 pour mettre en place un mécanisme d'injection de dépendance de classe.

**Question 4.1.2.** Reprenez les questions 3.7.3 et 3.7.4 pour créer un premier stub et l'injecter depuis les tests à la place de la dépendance non maîtrisable.

### Partie 2 – Doublure de test

Remarquons qu'à ce stade, on a deux méthodes `ageMoisDateConsole` et `ageMois` dont le fonctionnement est le même mais qui récupèrent la date du calcul différemment (en console par le passage en paramètre du constructeur, ou par le passage de paramètre de la méthode). Poursuivons le test de `ageMoisDateConsole`.

#### Exercice 4.2. Mock (30 min)

**Question 4.2.1.** Implémentez ces 3 cas de test :

```
CT_age2 : 10 mois entre la naissance le (2021, 2, 28) et le (2022, 1, 1)
CT_age3 : 0 mois entre la naissance le (2021, 12, 31) et le (2022, 1, 1)
CT_age4 : 1 mois entre la naissance le (2021, 12, 1) et le (2022, 1, 1)
```

Pour ces 3 cas de test, il faut un stub qui renvoie systématiquement (2022, 1, 1) comme date. La date du jour de la demande est bien maîtrisée, mais on en est déjà à devoir créer un deuxième stub (en plus de celui de la question 3.7.3).

Utiliser un deuxième stub peut nécessiter :

- soit de créer un nouveau chien pour lui passer ce stub en paramètre du constructeur,
- soit de s'assurer que l'attribut de type `DateProvider` soit une var (et pas val), ce qui doit être de nouveau une modification faite définitivement par le développeur.

Les 3 cas de test suivants ont besoin d'une date du jour différente à chaque fois : cela serait laborieux avec trois autres stubs.

```
CT_age5 : 12 mois entre la naissance le (2021, 2, 15) et le (2022, 2, 28)
CT_age6 : 12 mois entre la naissance le (2021, 2, 15) et le (2022, 2, 15)
CT_age7 : 11 mois entre la naissance le (2021, 2, 15) et le (2022, 2, 1)
```

**Question 4.2.2.** Implémentez ces trois précédents cas de test avec des mocks en utilisant « mockk ».

#### Exercice 4.3. Espion (20 min)

En plus de vérifier le bon fonctionnement, nous pouvons aussi faire des vérifications concernant les interactions dans le programme. Cela peut être utile pour du diagnostic (se rendre compte qu'une méthode n'est pas appelée alors qu'elle devrait, ou qu'une autre est appelée plus que prévue) mais aussi pour renforcer les tests.

**Question 4.3.1.** Implémentez ce test toujours pour `ageMoisDateConsole` :

```
@Test
fun testAgeMois_CT_age7bis() {
    ch3.setDateNaissance(2022, 1, 1)
    assertEquals(1, ch3.ageMois(LocalDate.of(2022, 2, 1)),
        "1 mois entre le (2022, 1, 1) et le (2022, 2, 1)")
}
```

**Question 4.3.2.** Le test passe-t-il ?

**Question 4.3.3.** Renforcez le cas test en vérifiant que `getDate()` est bien appelé une fois sur le mock. L'est-il ? Pourquoi ?

**Question 4.3.4.** Changez l'appel à la méthode, est-ce bon cette fois ?

**Question 4.3.5.** Ajoutez ce qu'il manque.

## Partie 3 – Cas d'étude : Opérations Dépendantes

On reprend notre cas d'étude des opérations. On considère à nouveau la division et la factorielle, mais cette fois elles vont chercher leurs opérandes dans la console : l'utilisateur doit les taper à chaque fois. On répète les étapes vues au TD3 et au début du TD4 mais cette fois en travaillant en binôme en exploitant gitlab.

### Exercice 4.4. Exploitation d'un gestionnaire de version

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. Cela vous a malheureusement conduit à ne pas versionner, pas sauvegarder, pas partager votre travail.

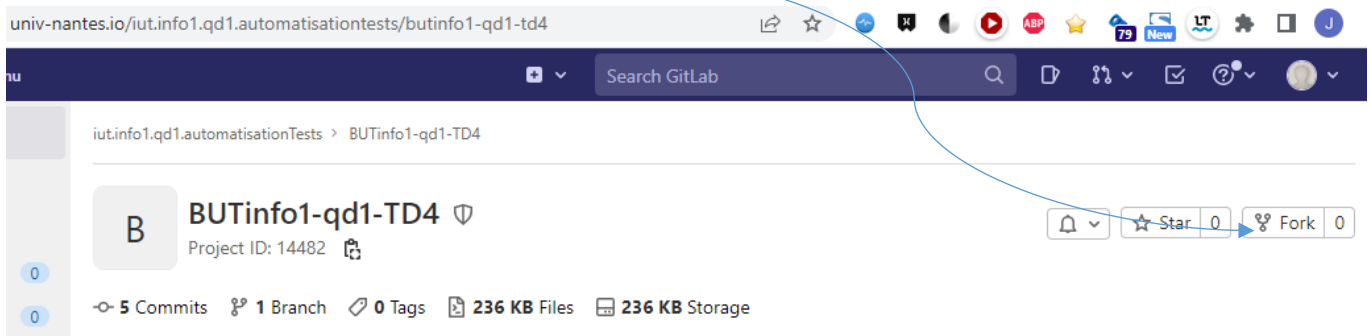
Reprenons ce que nous avons vu aux CM5 et TD7 du module R2.10 - Gestion de projet et des organisations. Travaillez en binôme (et seul trinôme par groupe de TD).

**Question 4.4.1.** Fork

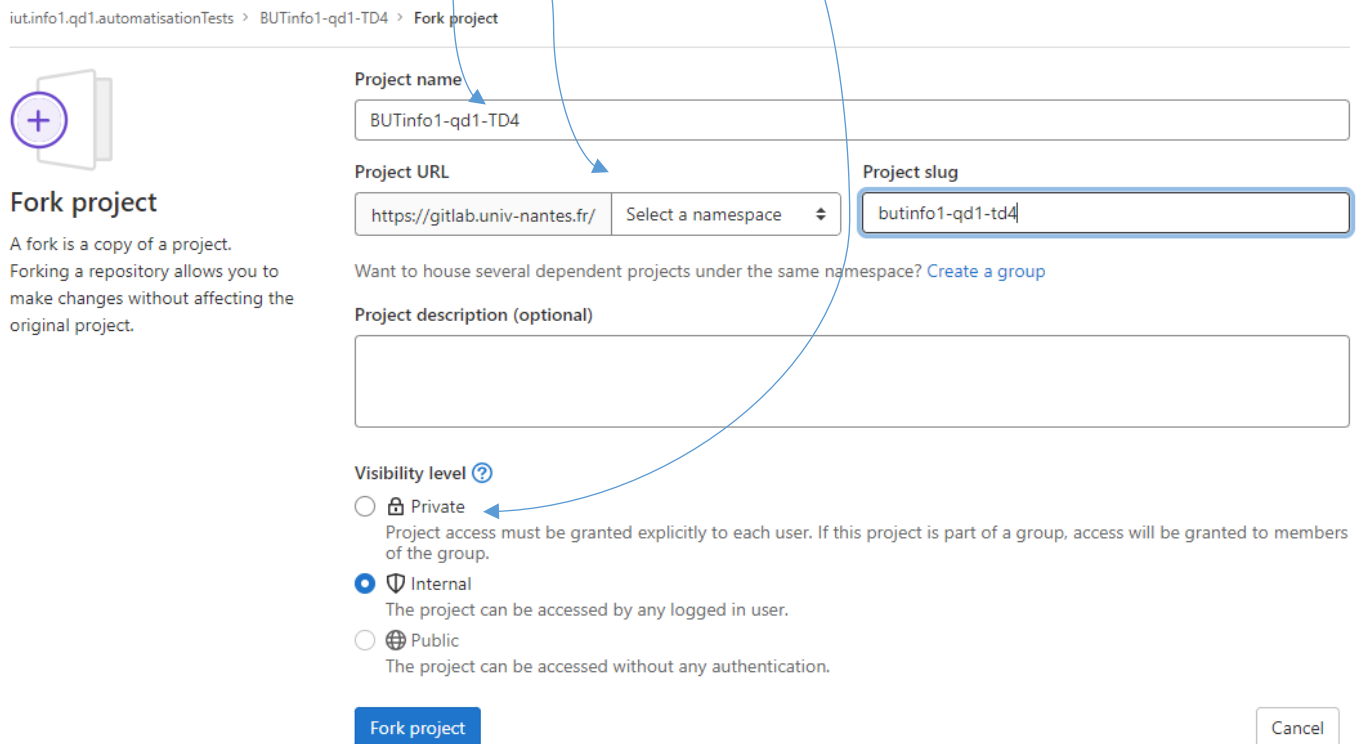
Rendez-vous ici :

<https://univ-nantes.io/iut.info1.qd1.automatisationtests/butinfo1-qd1-td4>

Un étudiant par binôme 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-TD4, mettez le projet « private » (c'est votre travail personnel, à votre binôme) :



The screenshot shows the 'Fork project' form in GitLab. At the top, the breadcrumb is 'iutinfo1.qd1.automatisationTests > BUTinfo1-qd1-TD4 > Fork project'. On the left, there's a 'Fork project' section with a plus icon and text explaining forking. The main form has several fields: 'Project name' (containing 'BUTinfo1-qd1-TD4'), 'Project URL' (containing 'https://gitlab.univ-nantes.fr/' and a 'Select a namespace' dropdown), 'Project slug' (containing 'butinfo1-qd1-td4'), 'Project description (optional)' (empty), and 'Visibility level' (with options: Private, Internal (selected), and Public). Below the visibility level are two buttons: 'Fork project' and 'Cancel'. Blue arrows point from the text above to the 'Project name', 'Project slug', and 'Private' radio button.

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

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

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

#### Question 4.4.2. Clone

Chacun crée un projet IntelliJ en clonant son projet.

#### Question 4.4.3. Recréez la classe de test :

Le binôme 1 s'occupe des tests de la fonction calculant la `division`, l'autre des tests de la fonction calculant la `factorielle`. Faites-le dans la même classe de test (pour un peu plus de challenge).

#### Question 4.4.4. Versionnez, partagez votre travail avec votre binôme

### Exercice 4.5. Dépendances

#### Question 4.5.1. Les tests passent-ils ? Pourquoi ?

#### Question 4.5.2. Prenez un rôle de développeur : L'un crée la classe implémentant l'interface `intProvider`. L'autre ajoute un mécanisme d'injection de dépendance de classe.

#### Question 4.5.3. Versionnez, partagez votre travail avec votre binôme

#### Question 4.5.4. Reprenez le rôle de testeur pour changer vos tests pour utiliser des mocks qui renverront les valeurs nécessaires aux tests plutôt que de les lire en console.

#### Question 4.5.5. Versionnez, partagez votre travail avec votre binôme