**TP 4A - Génie Logiciel**

**EXERCICE 3 Développement**

En tant que développeur de notre application, nous souhaitons mettre en place un système de versionning de code. Nous allons au long de ce TP explorer différentes façons de versionner notre projet.

**Git Bash : Méthode brut pour du versionning en local**

L’ensemble des commandes effectuées précédemment sont faisables avec git bash

Nous nous plaçons dans le dossier du projet (ou se trouve notre .git), nous tapons les commandes suivantes pour :

Initialiser un dépôt git : git init

Indexer les fichiers : git add fichier.txt

Commit les changements : git commit -m « Commentaire »

Ajouter des versions : git tag -a « vx » -m « x-ème version du projet »



Créer une nouvelle branche : git branch « Nouvelle branche »

Revenir dans le passé : git check out v2

Switcher de branche : git switch Branche1

**Eclipse : surcouche graphique à git pour faciliter les commandes**

Cependant, Eclipse peut être utiliser pour comme interface graphique. Il prend en charge toutes les lignes de commandes qui peuvent s’avérer fastidieuses.

Sous Eclipse, en cliquant droit sur le projet et faisant **Team** > **Share Project** > **Git** > **Use or Create a repository in parent folder of project** > **Create repository** > **Finish**, nous relions notre projet en cours de développement à notre répertoire de travail.

Une image contenant texte, capture d’écran, moniteur, noir

Description générée automatiquement

Au fur et à mesure, nous indexons les fichiers en cliquant droit comme précédemment et **Team** > **Commit.**

Une image contenant texte, capture d’écran, moniteur, écran

Description générée automatiquement

Vert : Si on veut choisir de Staged tous les documents modifiés

Rose : Si on veut choisir de Staged les documents sélectionnés uniquement

Jaune : Commit les documents Staged (ne pas oublier de mettre un commentaire pertinant sur la modification effectuée

NB : La zone de texte du message peut éventuellement être cachée, ne pas hésiter a agrandir la page pour la faire apparaitre. (cela ne m’est pas arrivé, c’est faux xD)

\*\*\*

Sur Eclipse, nous pouvons créer des nouvelles branches pour suivre le développement collaboratif. Pour cela, en cliquant droit sur le projet et **Team** > **Switch Branch** > **New Branch**

Une image contenant texte

Description générée automatiquement

Jaune : nous allons nous déplacer sur la nouvelle branche. C’est l’équivalent à un Check out en ligne de commande.

**GitHub : Autre méthode pour du versionning en ligne**

Dorénavant, nous souhaitons push notre répertoire git sur un dépôt GitHub. Il s’agit d’un cloud pour déposer nos projets. Il est encore plus simple de partager les modifications entre nous.

* Faire un tuto de la création d’un compte GitHub ?

Nous créons un nouveau dépôt sur le GitHub de Noémie.

Une image contenant texte, capture d’écran, moniteur, argent

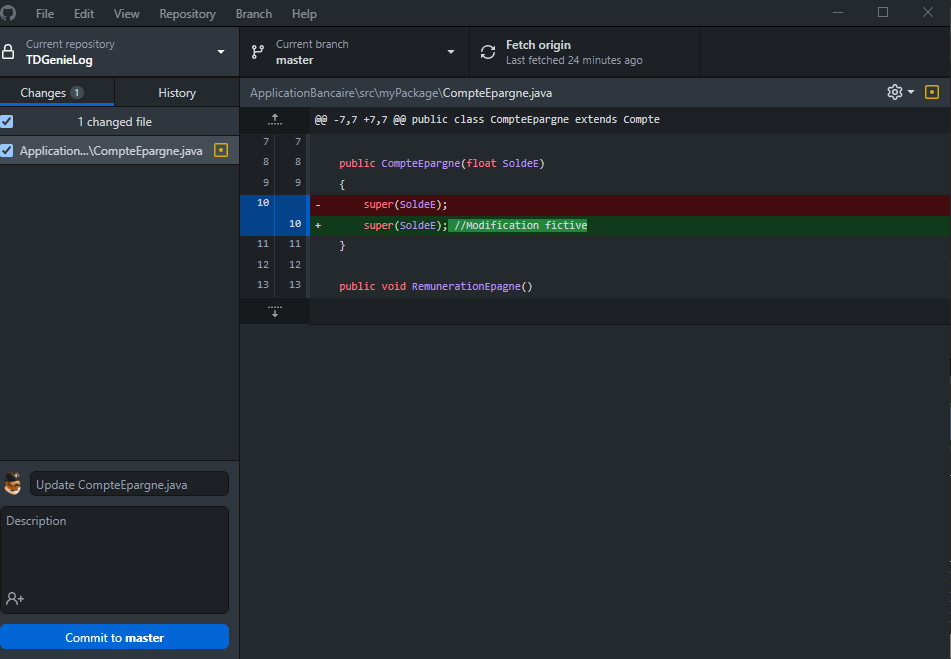
Description générée automatiquement

Le dépôt apparait donc dans sa liste de dépôt de travail.

Une image contenant texte

Description générée automatiquement

On installe GitHub desktop pour faire la liaison entre notre répertoire de travail local et notre dépôt github. Il s’agit d’un cliquodrome très pratique.



Grace à cette interface, nous pouvons réaliser les mêmes actions que dans Eclipse ou git Bash.

L’avantage de travailler avec github est que nous pouvons échanger plus facilement notre projet. Il suffit de récupérer le lien du dépôt et de faire un git clone.

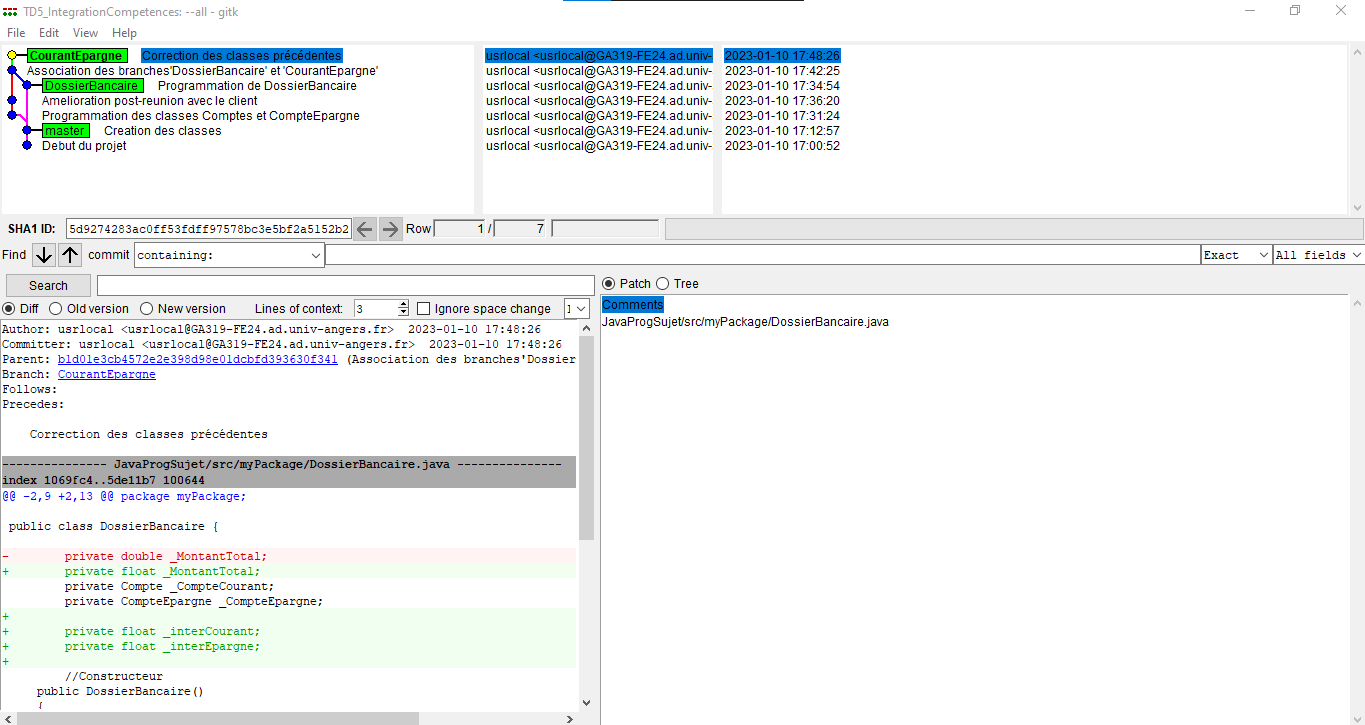
Pour ajouter les modifications ultérieures : **File** > **Add a local repository**

Créer une nouvelle branche : **Branch** > **New Branch** > **Create**

Fusionner deux branches :  **Branch** > **Merge into current branch > Merge**

\*\*\*

Visualisation de l’histoire de notre projet via git gui *: (on mettra l’historique de fin quand on aura fini, c’était pour penser à le mettre)*



**Exercice 2 : LIGNES DE COMMANDES**

Compilation du projet en ligne de commande :

On ouvre une console dans le /src de notre projet et exécutons les lignes

Lecture des READ ME :

###############################

Program compilation and execution from command line

From src/ directory containing myPackage/ subdirectory

###############################

#Compilation

javac myPackage/Main.java

#Execution

java myPackage/Main

Problème, il ne trouve javac :

On va dans **Panneau de configuration** > **Système** > **Paramètres avancés du système** > **Variable d’environnement** > Sélectionner PATH (variable d’utilisateur pour…) > **Modifier** > Ajouter le chemin d’accès de javac sur le PC (il varie d’un PC à l’autre, selon l’installation)

On fait de même pour les tests

###############################

Test suite compilation and execution from command line

From src/ directory containing tests/ subdirectory

###############################

#Compilation ( -cp used to indicate path to required third party libraries for unittesting )

#Note: you may require to adapt path and jar names

C:\Users\noemi\Downloads\TD5\JavaProgSujet\src>javac -cp "C:\Users\noemi\.p2\pool\plugins\org.junit\_4.13.2.v20211018-1956.jar";"C:\Users\noemi\.p2\pool\plugins\org.hamcrest.core\_1.3.0.v20180420-1519.jar" tests\MyTest1.java tests\MyTest2.java

#Execution

#Note: you may require to adapt path and jar names

C:\Users\noemi\Downloads\TD5\JavaProgSujet\src>java -cp "C:\Users\noemi\.p2\pool\plugins\org.junit\_4.13.2.v20211018-1956.jar";"C:\Users\noemi\.p2\pool\plugins\org.hamcrest.core\_1.3.0.v20180420-1519.jar"; tests/MyTestSuite1Runner

On anticipe les problèmes :

On va dans **Panneau de configuration** > **Système** > **Paramètres avancés du système** > **Variable d’environnement** > Sélectionner PATH (variable d’utilisateur pour…) > **Modifier** > Ajouter le chemin d’accès de hamcrest/ junit sur le PC (il varie d’un PC à l’autre, selon l’installation)