

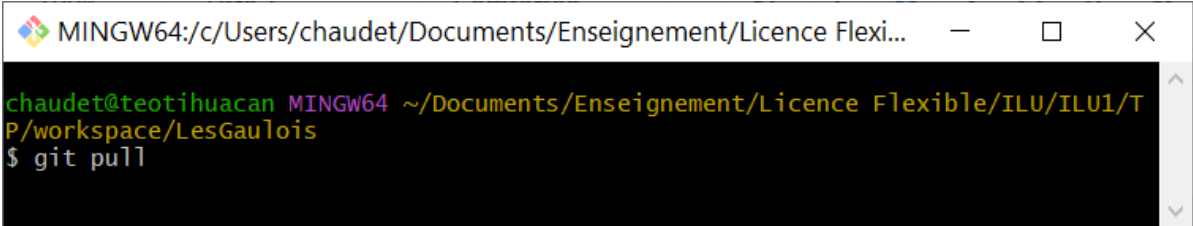
## Un peu plus de codage en Java

### 1 Environnement GitHub (Rappel)

Mettre à jour votre dépôt local avec le contenu du dépôt distant.

Pour cela, il existe deux solutions : en ligne de commande ou avec l'outil graphique.

- En ligne de commande :
  - Avec un **explorateur de fichier** se placer dans le répertoire du projet 'LesGaulois' qui se trouve sous votre workspace (vous devez y trouver le fichier `.gitignore`), puis cliquer droit (sans rien sélectionner) et cliquer sur 'Git Bash Here'.
  - Dans Git Bash (ouvert directement dans le répertoire du projet 'LesGaulois'), tapez la commande `git pull`.



```
MINGW64:/c:/Users/chaudet/Documents/Enseignement/Licence Flexi...  
chaudet@teotihuacan MINGW64 ~/Documents/Enseignement/Licence Flexible/ILU/ILU1/TP/workspace/LesGaulois  
$ git pull
```

- Avec l'outil graphique, le bouton 'Fetch origin' permet de mettre à jour la fenêtre principale pour connaître les modifications à mettre à jour localement. Il suffit ensuite de cliquer sur le bouton 'Pull origin'.

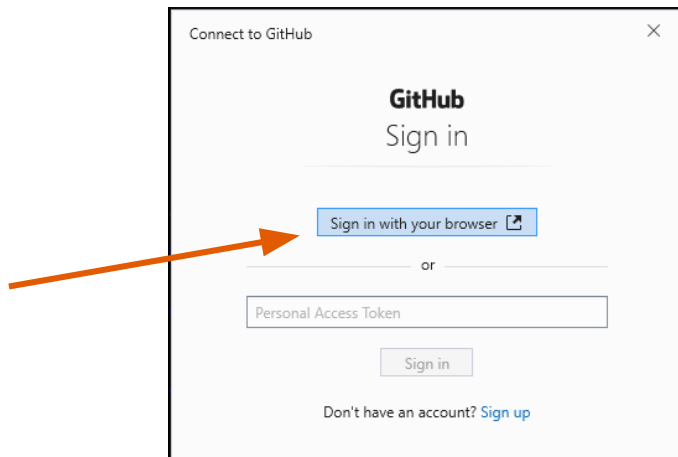
## Historisation périodique de votre projet

Lorsque vous avez fini une question du sujet de TP (ce qui vous a amené à créer ou modifier plusieurs classes), vous devez procéder à l'historisation de votre dépôt local et la mise à jour du dépôt distant. Là encore, deux solutions sont possibles : en ligne de commande ou avec l'outil graphique.

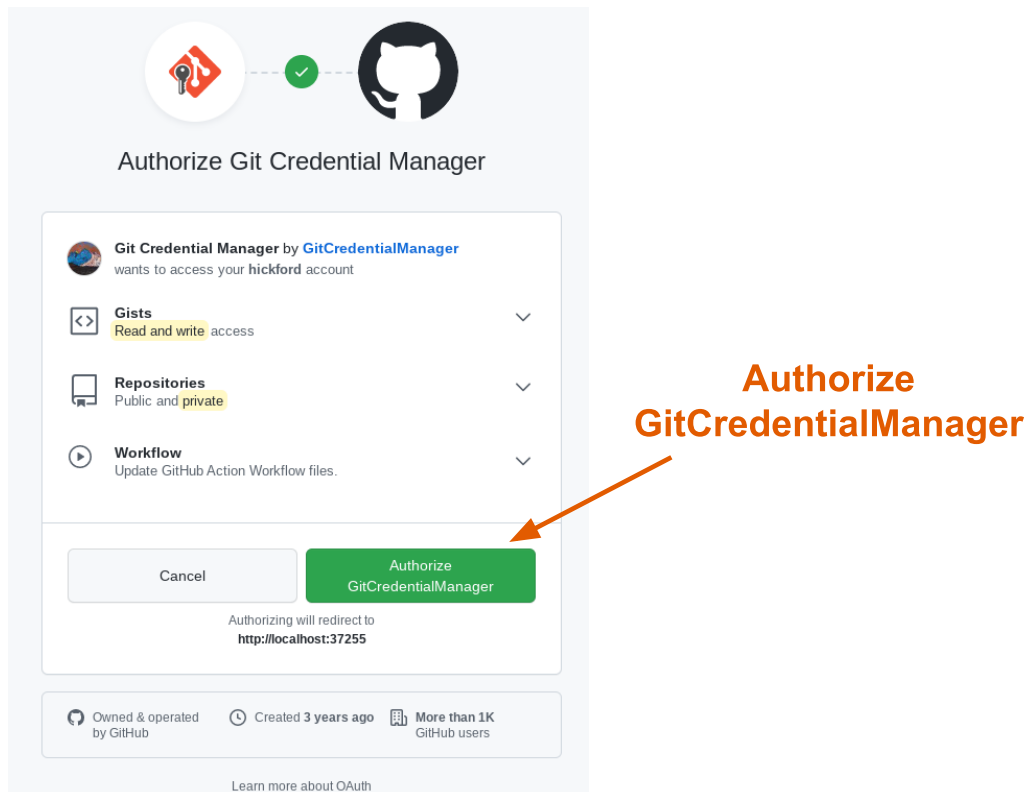
### *En ligne de commande*

- Avec un **explorateur de fichier** retrouver sous votre projet LesGaulois (vous devez y trouver le fichier `.gitignore`) puis cliquer droit (sans rien sélectionner) et sélectionner 'Git Bash Here'.

- Sous Git Bash : utiliser les commandes git :
  - `git add .`
  - `git commit -m <Intitulé des modifications>`  
**exemple :** `git commit -m "TP1 methode toString"`
  - `git push`
- Une fenêtre peut s'ouvrir en vous demandant de vous identifier, Sélectionner "Sign in with your browser",



- Une autre fenêtre s'ouvre (extrait ci-dessous), sélectionner sur "Authorize GitCredentialManager"



*Avec l'outil graphique*

La procédure est simplifiée à deux actions :

- Dans l'onglet 'Changes', la liste des fichiers créés ou modifiés apparaît avec une présélection de tous les fichiers. Il ne reste plus qu'à écrire l'intitulé du commit (et la description éventuelle en-dessous) et cliquer sur le bouton 'Commit to main' en bas de fenêtre.
- La fenêtre principale se met à jour et présélectionne le bouton 'Push origin'. Il n'y a plus qu'à cliquer dessus.

## 2 Programmation java Les Gaulois

### 1. Le village des Gaulois

*Les tableaux*

Ci-dessous la classe `Village` écrite dans le TP précédent.

```
public class Village {
    private String nom;
    private Chef chef;

    public Village(String nom) {
        this.nom = nom;
    }

    public void setChef(Chef chef) {
        this.chef = chef;
    }

    public String getNom() {
        return nom;
    }
}
```

- Un village rassemble certes un chef, mais aussi des villageois ! Ajouter deux attributs : un tableau `villageois` de gaulois, un entier `nbVillageois` correspondant au nombre de villageois habitant dans ce village qui sera initialisé à 0 (lorsque l'on crée le village il n'y a pas encore d'habitant).
- Ajouter au constructeur de la classe `Village` un paramètre d'entrée `nbVillageoisMaximum`. Grâce à ce paramètre, vous pouvez créer le tableau `villageois` (déclaré précédemment) avec en nombre de cases le nombre maximum de villageois.

- c. Maintenant, il va falloir ajouter des gaulois dans le village. Compléter la méthode `ajouterHabitant` qui prend en paramètre d'entrée un gaulois et le place dans le tableau `villageois` (n'oubliez pas de mettre à jour le nombre de villageois !)
- d. Ecrire la méthode `trouverHabitant` qui prend en paramètre d'entrée le numéro du villageois dont on veut récupérer la référence.
- e. Un main dans la classe `Village` :
- écrire un main dans lequel vous créez un `Village village` dont le nom est "Village des Irréductibles" et qui possède au maximum 30 habitants.
  - Quel message (qu'on appelle message de levée d'exception) apparaît si vous écrivez l'instruction :  

```
Gaulois gaulois = village.trouverHabitant(30);
```
  - Placer l'instruction ci-dessus en commentaire et en-dessous (toujours en commentaire) écrire pourquoi on obtient l'exception du point précédent.
  - Créer le chef du village Abraracourcix qui a une force de 6 et l'ajouter au village.
  - Créer le gaulois Astérix qui a une force de 8, et l'ajouter au village.
  - Que se passe-t-il si vous écrivez les instructions :  

```
Gaulois gaulois = village.trouverHabitant(1);  
System.out.println(gaulois);
```
  - Placer les instructions ci-dessus en commentaire et en-dessous (toujours en commentaire) expliquer la réponse obtenue
- f. Toujours dans la classe `Village` :
- Écrire la méthode `afficherVillageois` qui affiche le chef puis les noms de tous les villageois en évitant les messages d'erreur !  
Exemple de sortie attendues :
- ```
Dans village du chef Abraracourcix vivent les légendaires gaulois :  
- Asterix  
- Obelix
```
- g. A nouveau dans le main de la classe `Village` :
- Créer le gaulois Obélix qui a une force de 25, et l'ajouter au village.
  - Appeler la méthode `afficherVillageois` pour contrôler qu'Obélix fait bien partie des villageois affichés.
- h. Sauvegarder votre travail sous GitHub (cf p1 - Historisation périodique de votre projet - exemple de message pour le commit "le village gaulois")

## *Les pré, post conditions et les invariants*

Ci-dessous l'extrait de la classe `Romain` écrite dans le TP précédent.

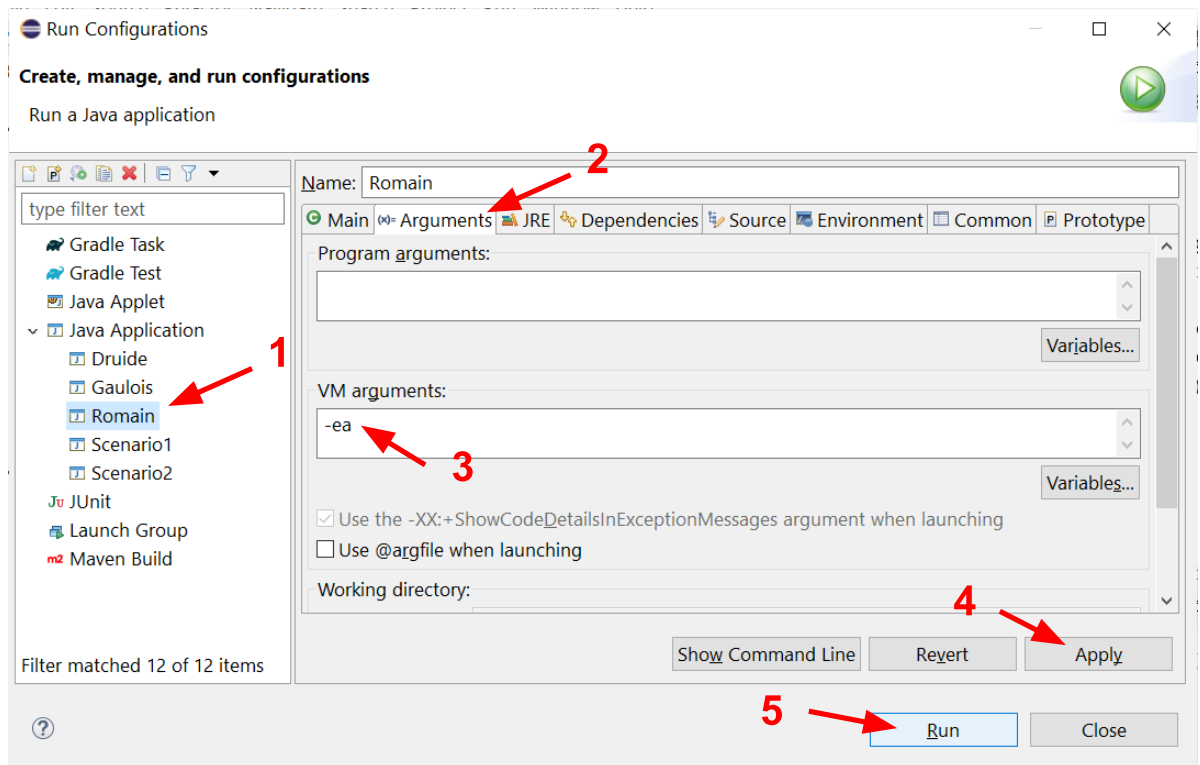
```
public class Romain {
    private String nom;
    private int force;

    public Romain (String nom, int force) {
        this.nom = nom;
        this.force = force;
    }

    public void recevoirCoup(int forceCoup) {
        force -= forceCoup;
        if (force > 0) {
            parler("Aïe !");
        } else {
            parler("J'abandonne...");
        }
    }
}
```

### a. Travail sur l'invariant :

- Vérifier l'invariant "la force d'un Romain est toujours positive" à la création d'un objet.
- Modifier le main de la classe `Romain` en modifiant la force de Minus pour -6.
- Tester. Rien ne se passe ? Rassurez-vous c'est normal : il faut activer la vérification des assertions. Cliquer sur l'onglet Run puis sélectionner Run Configurations... (voir la figure ci-dessous)
  1. sous Java Application sélectionner la classe où se trouve votre main (dans notre exemple la classe `Romain`).
  2. appuyer sur l'onglet "Arguments"
  3. Dans la zone de texte "VM arguments" écrire -ea (pour enable assertions)
  4. Appuyer sur le bouton "Apply"
  5. Vous pouvez lancer l'exécution en appuyant sur le bouton "Run"



- Vous devez obtenir une `java.lang.AssertionError`
  - Remettez la valeur de la force de Minus à 6.
- b. Travail sur les pré et post conditions
- Vérifier la précondition "la force d'un Romain est positive" au début de la méthode `recevoirCoup`
  - Vérifier la postcondition "la force d'un Romain a diminué" à la fin de la méthode `recevoirCoup`. N'hésitez pas à ajouter des variables locales au besoin.
- c. Sauvegarder votre travail sous GitHub (cf p1 - Historisation périodique de votre projet - exemple de message pour le commit "invariant pre post conditions")

## 2. Des Romains bien protégés

### Les énumérés

- a. Dans le package « personnages » créer l'enum `Equipement` possédant deux énumérés : `CASQUE` et `BOUCLIER`.

Tester votre énumération dans le main de la classe `Romain` en faisant afficher `CASQUE` et `BOUCLIER`.

- b. Modifier l'énumération `Equipement` en lui ajoutant :
- un attribut `nom`,
  - un constructeur prenant en paramètre d'entrée une chaîne qui permettra d'initialiser la valeur de l'attribut `nom`.

- Par conséquent, vous devrez ajouter cette chaîne pour chacun des énumérés par exemple `CASQUE("casque")` et `BOUCLIER("bouclier")`.
  - Ajouter la méthode `toString` qui affichera le nom de l'énuméré.
- c. Dans la classe `Romain` créer deux attributs :
- un tableau `equipements` contenant des objets de type `Equipement` de deux cases,
  - un entier `nbEquipement` initialisé à 0.
- d. Veuillez lire l'ensemble de la question avant de commencer à la coder.

Dans la classe `Romain` créer la méthode `sEquiper` qui prend en paramètre un équipement.

Si le romain a déjà deux équipements la méthode affichera : « Le soldat », son nom et « est déjà bien protégé ! ». Par exemple :

```
Le soldat Minus est déjà bien protégé !
```

Si le soldat possède déjà un équipement alors il faut regarder dans la première case du tableau s'il s'agit du même équipement que celui donné en paramètre d'entrée. La méthode affichera : « Le soldat », son nom et « possède déjà », le nom de l'équipement et un point d'exclamation. Par exemple :

```
Le soldat Minus possède déjà un casque !
```

Dans les autres cas, l'équipement est ajouté au tableau et le nombre d'équipements est incrémenté. La méthode affichera : « Le soldat », son nom, « s'équipe avec un », le nom de l'équipement et un point. Par exemple :

```
Le soldat Minus s'équipe avec un bouclier.
```

Pour cette méthode :

- vous devrez OBLIGATOIREMENT utiliser une structure `switch` sur le nombre d'équipement.
- la méthode ne doit pas avoir de portion de code dupliqué, au besoin créer une méthode privée. Si besoin, sélectionner la portion de code dupliquée, et choisir la commande 'Extract Method...' du menu Refactor.

Tester la méthode dans le `main` de la classe `Romain`, en ajoutant à Minus deux casques puis un bouclier et enfin un autre casque. Vous devez obtenir l'affichage suivant :

```
Le soldat Minus s'équipe avec un casque.  
Le soldat Minus possède déjà un casque.  
Le soldat Minus s'équipe avec un bouclier.  
Le soldat Minus est déjà bien protégé !
```

- e. Sauvegarder votre travail sous GitHub (cf p1 - Historisation périodique de votre projet - exemple de message pour le commit "les énumérés")