# TP : Un projet / Un workspace
# Mise en place du socle technique de la SAE

*Dans le premier TP, il vous a été dit : « Dans un **workspace**, vous pourrez créer plusieurs projets java, mais rien ne vous empêche de créer autant de **workspaces** que vous le souhaitez.*
*Une bonne pratique serait d'ailleurs que vous créiez un **workspace** dédié à la **SAE** le moment venu 😉 »*

***Une bonne pratique consiste, sous Eclipse, à utiliser, un workspace par projet (application)***
*(la notion de workspace est propre à Eclipse,*
*sous IntelliJ, vous n'aurez pas le choix, vous ne verrez d'office qu'un seul projet)*

*Pour l'instant, nous avons créé tous nos projets dans le même workspace **`javabut1`***
*Ce workspace nous a permis de regrouper un ensemble de petits projets*
*qui illustrent les notions de bases de la COO et POO au travers de petits exemples.*
*Le temps est venu de préparer le socle technique de la SAE,*
*mais avant nous allons nous entraîner*
*en révisant les notions vues précédemment un autre petit projet `warcardgame`…*

## Partie 1
## Mise en place d'un projet `warcardgame` dans un nouveau workspace `cardgame`

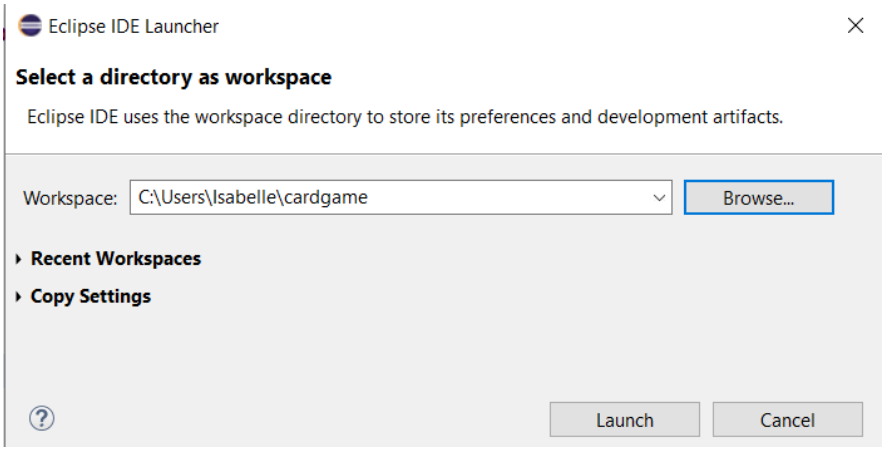
### 1. Mise en place d'un nouveau workspace `cardgame` :

Lancez l'IDE Eclipse comme à votre habitude. Nous allons créer un nouveau workspace **`cardgame`.**

❑ Pour créer un nouveau workspace, sélectionnez **`File → Switch Workspace → Other …`**

❑ Cliquez sur le bouton **Browse…** , et créez un nouveau dossier **`cardgame`** , par exemple, au même niveau que le précédent workspace javabut1 : vous devez voir javabut1 dans la liste des répertoires, pas être dans **`javabut1`** 😉.
***Remarque*** : Pour un bon fonctionnement de certains plug-ins (comme Infinitest par exemple), le chemin que vous choisirez pour votre nouveau dossier **`cardgame`** ne doit pas contenir d'espace.

❑ Faites en sorte que la fenêtre *Eclipse Launcher* contienne désormais comme valeur de *Workspace* le chemin vers `cardgame` et cliquez sur **`Launch`**.
Eclipse va se fermer et se relancer. Fermez l'onglet Welcome….
Vous êtes dans le nouveau workspace **cardgame.**
Si tout s'est bien passé, votre vue Package Explorer doit être actuellement vide 😉




Retenez que lorsque vous voudrez (re)changer de workspace,
vous devrez (re)passer par **`File → Switch Workspace`**

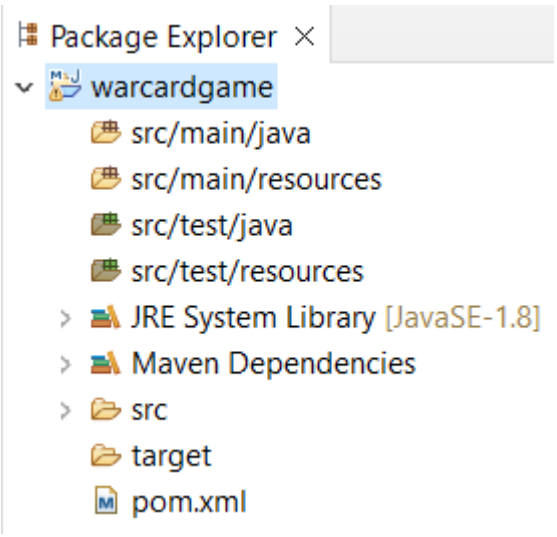
## 2. Mise en place d'un projet maven `warcardgame` dans le workspace `cardgame`

*… sur le même principe la partie 2 : **Prise en main de maven***
*du TP de la semaine dernière …*

Comme vous l'avez fait au TP précédent, créez maintenant un projet **`warcardgame`** dans votre *workspace* **`cardgame`**.

❑ Comme précédemment, dans votre **`pom.xml`** :
- vous paramétrerez **l'encodage UTF 8**.
- vous choisirez la **version de java** que vous souhaitez du moment qu'elle soit **supérieure ou égale au JDK 8** : il faut savoir que ¾ des projets en production sont encore en JDK 8.
- Vous ajouterez une **dépendance vers JUnit 5**.

❑ Soyez malin pour modifier votre **`pom.xml`** et n'oubliez pas **d'*updater* votre projet maven une fois le pom modifié** pour bien voir apparaître dans la structure du projet la version java que vous avez choisie sur la ligne JRE System Library (ici du **1.8**).




*… sur le même principe que le TP de la semaine dernière,*
*nous allons mettre en place la gestion de version avec git sous Eclipse sur le projet warcardgame*
*et faire en sorte que votre dépôt local soit connecté*
*avec un dépôt distant (remote) du même nom sur votre compte github…*

## 3. En local : Configuration de la gestion de version sur `warcardgame`

*… sur le même principe que la partie 3 : **Configurer la gestion de version d'un projet Maven***
*d'un précédent TP …*

❑ Mettez en place la **gestion de version avec git** sur votre projet **`warcardgame`** en faisant en sorte que votre **dépôt local .git soit directement dans votre projet** (comme au précédent TP).

❑ Faites votre premier commit avec le fichier **`pom.xml`** et le message **initial commit**
❑ Profitez de ce premier commit pour configurer votre **.gitignore**

## 4. En local : Let's code & commit : Pas de jeu sans carte !

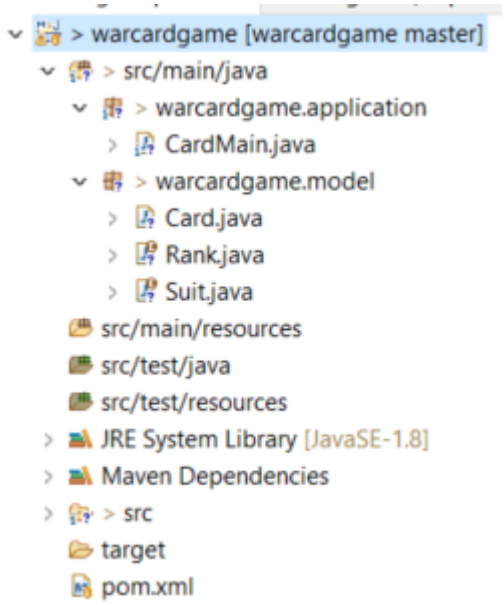
### ❑ Code de production (`src/main/java`) : Implémentez !

⇨ Vous avez sans doute compris que le projet **`warcardgame`** ne sera autre que la suite du projet **`cardgame`** que vous aviez déjà commencé à implémenter dans un TP précédent.

- Si vous aviez fini cette implémentation, vous pouvez directement récupérer votre code et l'importer dans le projet **`warcardgame`**
- sinon vous trouverez le code des classes **`Card`**, **`CardMain`** et des énumérations **`Rank`** et **`Suit`** dans le gist suivant : https://unil.im/cardgame (https://gist.github.com/iblasquez/624e208369db425528cc7755ce44f685)

⇨ Sauvegardez bien les fichiers, puis pour chaque fichier :

- corrigez les erreurs de compilation liées aux `import` en ajoutant **tous les import d'un même fichier d'un seul coup** en utilisant le menu de l'IDE : **`Source→ Organize Import`** (retenir le raccourci clavier **`CTRL+Shift+O`** de cette action qui pourra vous être très utile dans tous vos futurs développements java sous Eclipse 😉)
- **(re)formattez vos fichiers de code** en vous aidant du menu suivant de **l'IDE : `Source→Format`**
- sauvegardez votre fichier si ce dernier a été modifié suite aux actions précédentes 😉




⇨ Faites en sorte de que votre projet **`warcardgame`** soit structuré de la même manière que l'était le projet **`cardgame`** c.-à-d. que sa structure soit similaire à la copie d'écran ci-contre

⇨ Exécutez la classe **`CardMain`** pour vérifier que le comportement attendu est le bon !

### ❑ Code de test (src/test/java) : Vérifiez et Validez !

⇨ Pour l'instant, nous n'écrirons pas de tests automatisés.
Le code de la classe `CardMain` nous permet de vérifier et valider le bon comportement du code implémenté.

### ❑ Commitez !

⇨ Commitez les quatre fichiers java précédents avec, par exemple, comme message de commit :

```
add Card, Rank, Suit and CardMain
```

## 5. Pousser le code local vers un `remote`

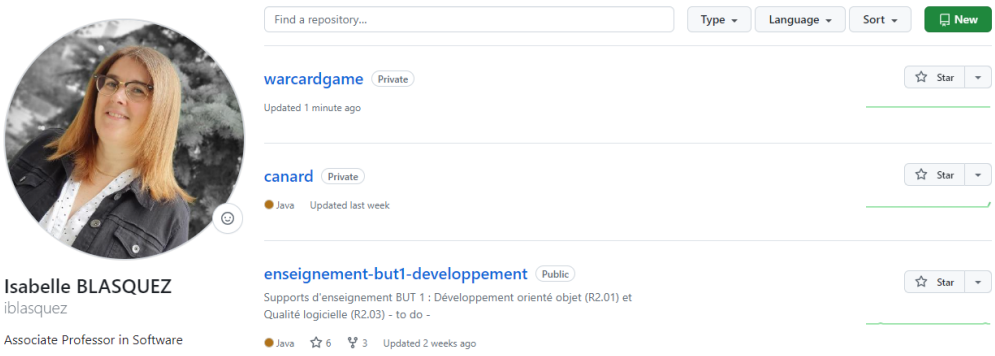
*… sur le même principe que l'étape 4 :* ***Pousser son projet sur un remote la première fois***
*de la partie 4 :* ***Apprendre à versionner régulièrement son projet durant la phase d'implémentation***
*d'un précédent TP …*

### ❑ Pousser vers le remote (votre compte `Github`)

Poussez directement le projet local **`warcardgame`** (que vous venez de créer et versionner) comme dépôt privé sur votre compte **Github**.

***Remarque :*** Gitkraken peut vous faciliter la tâche, mais si vous êtes un habitué de *git*, rien ne vous empêche d'effectuer cette tache en ligne de commande 😉

### ❑ Vérifier que le code local a bien été poussé sur le remote 😉




⇨ Le projet **`warcardgame`** doit maintenant apparaître comme dépôt privé sur votre compte Github.

⇨ Cliquez sur le dépôt **`warcardgame`** pour consulter le détail de ce dépôt :
- → vérifiez si votre nombre de commits est bien de 2.
- → vérifiez si tous les fichiers ont bien été poussés.

# Partie 2
# Mise en place du projet (collaboratif) de la SAE dans un nouveau workspace `saebut1`

Pour cette partie, vous allez travailler par **équipe de SAE** (trinôme ou quator).

Pour commencer, regroupez votre équipe pour commencer.
⇨ Décidez qui dans votre équipe va héberger le projet **`scrabble`** sur son compte **github** c.-à-d. jouer le rôle du ***propriétaire*** du serveur distant (remote).
Notez le compte github qui vous servira de remote : @_____
Ne vous inquiétez pas, tous les autres membres de l'équipe auront accès à ce remote en tant que *collaborateurs* : c'est le principe d'un travail collaboratif 😉

## 1. Pour tous les membres de l'équipe
## Création en local d'un `workspace` uniquement dédiée à la SAE : `saebut1`

❑ Créez un nouveau **workspace** **`saebut1`** à l'aide de **`File → Switch Workspace → Other …`**
Laissez ce **workspace vide**, n'ajoutez aucun projet pour le moment 😉

❑ Tous les membres de l'équipe, hormis le futur propriétaire du dépôt, peuvent fermer leur eclipse pour le moment et se regrouper autour du futur propriétaire 😉

## 2. Uniquement par *le propriétaire*
## Mise en place en local d'un projet *maven* `scrabble` dans le `workspace saebut1`

❑ Créez un projet **`scrabble`** dans votre *workspace* **`saebut1`**

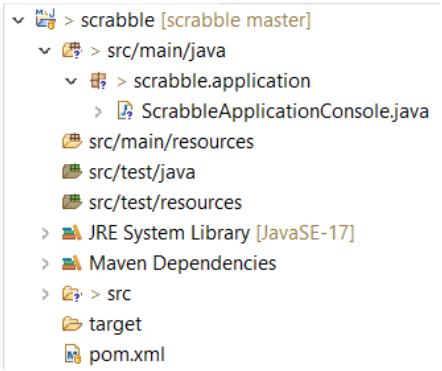
❑ Comme précédemment, dans votre **`pom.xml`** :
- vous paramétrerez **l'encodage UTF 8**.
- vous choisirez la **version de java** que vous souhaitez du moment qu''elle soit **supérieure ou égale au JDK 8** : il faut savoir que ¾ des projets en production sont encore en JDK 8.
- Vous ajouterez une **dépendance vers JUnit 5**.

❑ Soyez malin pour modifier votre **`pom.xml`** et n'oubliez pas **d'*updater* votre projet maven une fois le pom modifié** pour bien voir apparaître dans la structure du projet la version java que vous avez choisie sur la ligne JRE System Library.

❑ Mettez en place la **gestion de version avec git** sur votre projet **`scrabble`** en faisant bien en sorte que votre **dépôt local .git soit directement dans votre projet** (comme au précédent TP).

❑ Faites votre premier commit avec le fichier **`pom.xml`** et le message **initial commit**
❑ Profitez de ce premier commit pour configurer votre **.gitignore**

❑ Créez dans le code de production (**src/main/java**) une première classe **ScrabbleApplicationConsole** dans un paquetage ***scrabble.application*** qui contiendra une méthode **main** qui lorsqu'elle sera exécutée affichera sur la console un message du genre :




```
------------------------------------------------------------
--  Bienvenue dans notre magnifique jeu de scrabble !  --
------------------------------------------------------------
```

❑ Exécutez la classe **ScrabbleApplicationConsole** pour vérifier que le comportement attendu est le bon !

❑ Commitez ce nouveau fichier avec un message de commit similaire au suivant :

**ajout ScrabbleApplicationConsole**

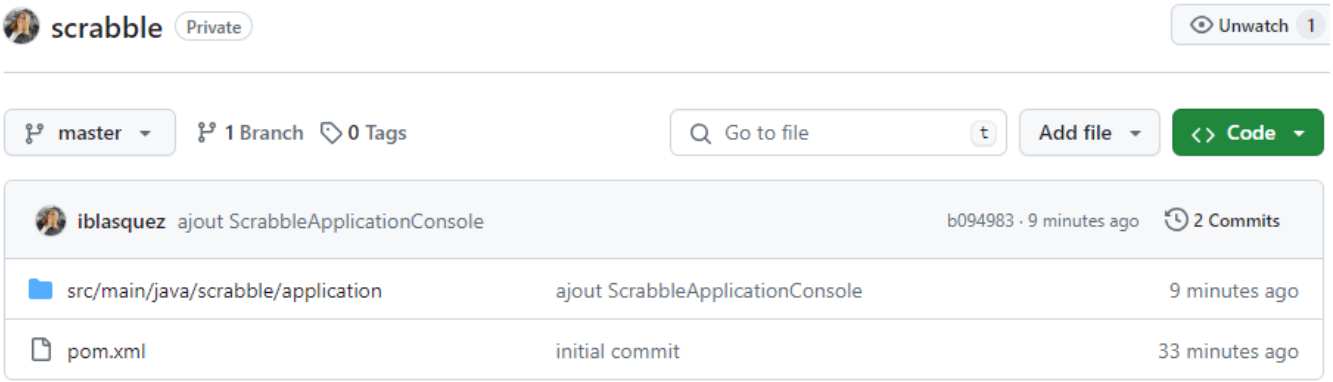
***Remarque*** : **ajout** ou **add** ?.... à vous de décider (en équipe) si vous voulez écrire votre message en français ou en anglais en fonction de la décision (collaborative) de coder le projet scrabble en français ou en anglais 😉

## 3. Uniquement par *le propriétaire*
## Pousser le code local du projet scrabble vers un dépôt distant (remote)

❑ Il ne reste plus qu'à pousser directement le projet local **scrabble** (que vous venez de créer et versionner) comme dépôt privé sur votre compte **Github**.

❑ Vérifiez que le projet scrabble local a bien été poussé comme dépôt privé sur le github et qu'il contient bien :

→ **2 commits**
→ le fichier **pom.xml**
→ dans **src/main/java/scrabble/application** le fichier **ScrabbleApplicationConsole.java**




❑ Notez tous l'adresse pour accéder à ce dépôt depuis un navigateur :

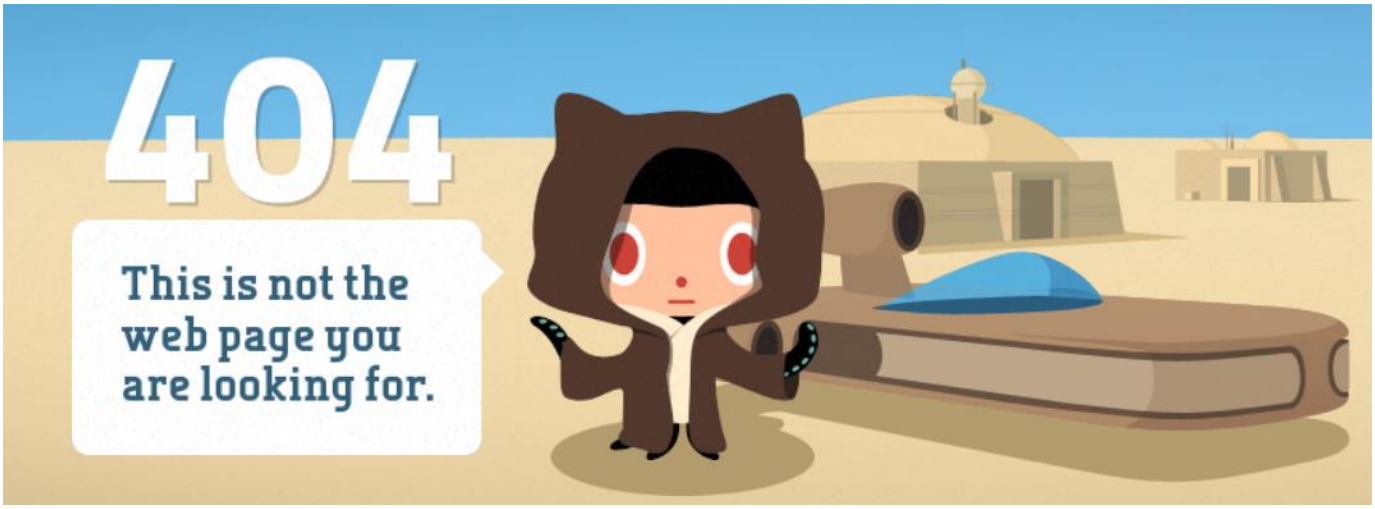
**https://github.com/_____/scrabble**

**Compte github du *propriétaire***

**Le nom du dépôt est le nom du projet en local**

## 4. Pour tous les membres de l'équipe
## Préparation au travail collaboratif

❑ A l'aide d'un navigateur, rendez-vous à l'adresse web précédente. Que se passe-t-il ?




Pour l'instant, seul le ***propriétaire*** peut voir et interagir avec son dépôt scrabble qui est privé.
Le propriétaire va donc devoir modifier les droits d'accès à son dépôt afin que tous les autres membres de l'équipe (que l'on appellera des ***collaborateurs***) puisse également voir ce dépôt et interagir avec (en poussant/tirant du nouveau code).

❑ Avant de continuer, notez ci-dessous les identifiants des comptes github de tous les membres de l'équipe :

- ⇨ Github du ***propriétaire*** : _____
- ⇨ Github du ***collaborateur n°1*** : _____
- ⇨ Github du ***collaborateur n°2*** : _____
- ⇨ Eventuellement Github du ***collaborateur n°3*** : _____

❑ Pour la suite, vous aurez aussi besoin des comptes github de vos enseignants de SAE qui sont les suivants :

**iblasquez**
**Laurent-Dubreuil**
**CristinaOnete**
**alaporte13**
**Beber46**
**Synetta**

## 5. Uniquement par *le propriétaire* dépôt distant (`remote`)
## Ajout des collaborateur(s) sur le dépôt distant (`remote`) du `scrabble`

❑ Pour ajouter des collaborateurs sur un dépôt Github, il faut se rendre sur le dépôt en question, puis cliquez sur **`Settings`** (dernière option du bandeau Vertical).

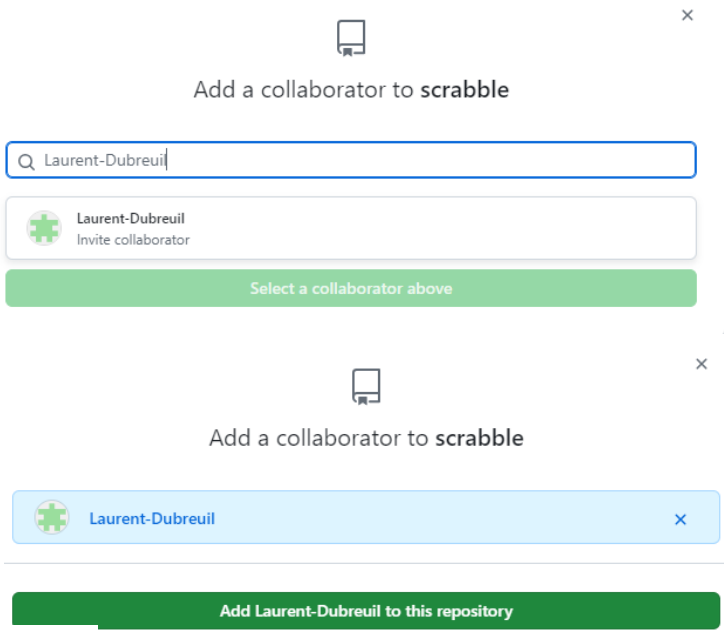
❑ Dans la rubrique **`Access`**, choisir ensuite **`Collaborators`**
Github va surement vous demander de re-saisir votre mot de passe 😉

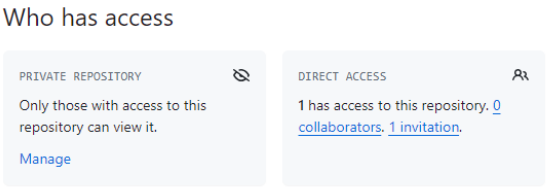
❑ Cliquez ensuite sur le bouton vert
**« `Add People` »**

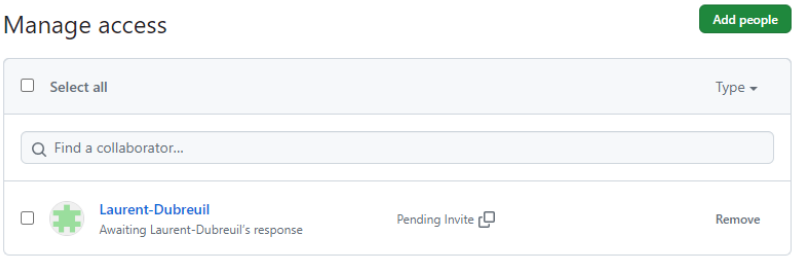
❑ Entrez dans la barre de recherche, l'identifiant du compte cherché (sans le @ !!!)

❑ **Sélectionnez le compte** que vous souhaitez *inviter en tant que collaborateur* (parfois il peut apparaître plusieurs comptes)

❑ puis **cliquez sur le bouton** vert
**« `Add….to this repository` »**
afin qu'une invitation à rejoindre soit envoyée au futur collaborateur : il est alors indiqué ***Pending Invite*** pour spécifier que l'invitation a été envoyée.




❑ ***En tant que propriétaire du dépôt***, ajoutez comme collaborateurs :
- → les comptes github de tous vos enseignants de SAE (donnés à la page 7)
- → les comptes github de tous les autres membres de votre équipe (notés page 7)

⇒ Au final, le nombre de personnes ayant accès au dépôt devra être de :
- 9 personnes pour les équipes de 3 membres (8 collaborateurs + le propriétaire)
- 10 personnes pour les équipes de 4 membres (9 collaborateurs + le propriétaire)

***En tant que collaborateur***, vous devez absolument valider l'invitation qui a été envoyée sur votre compte Github (c'est l'étape suivante que nous allons décrire)
Tant que vous n'aurez pas validé cette invitation (vous restez ***Pending Invite***)
et vous ne pourrez rien faire sur ce dépôt.

## 6. Pour tous les *collaborateurs* (hormis le *propriétaire*)
## Valider l'invitation pour rejoindre le dépôt distant (`remote`) du `scrabble`

❑ Pour validez votre invitation, en tant que collaborateur, vous pouvez :

⇨ soit vous connectez à votre compte github et vous verrez qu'une notification est apparue sur l'icône de votre boite aux lettres à côté de votre avatar :


Cliquez sur cette icône qui vous permettra de rejoindre le dépôt **`scrabble`** en cliquant sur le symbole de validation ✓ sur la ligne relative à l'invitation.

⇨ soit vous pouvez consulter votre boîte mails classique et
cliquez sur le bouton **View Invitation** que propose le mail qui provient de Github.

❑ Une fois votre invitation validée, si vous vous rendez à l'adresse du dépôt (celle que vous avez notée à la page 6), vous devriez maintenant voir le contenu de ce dépôt 😉

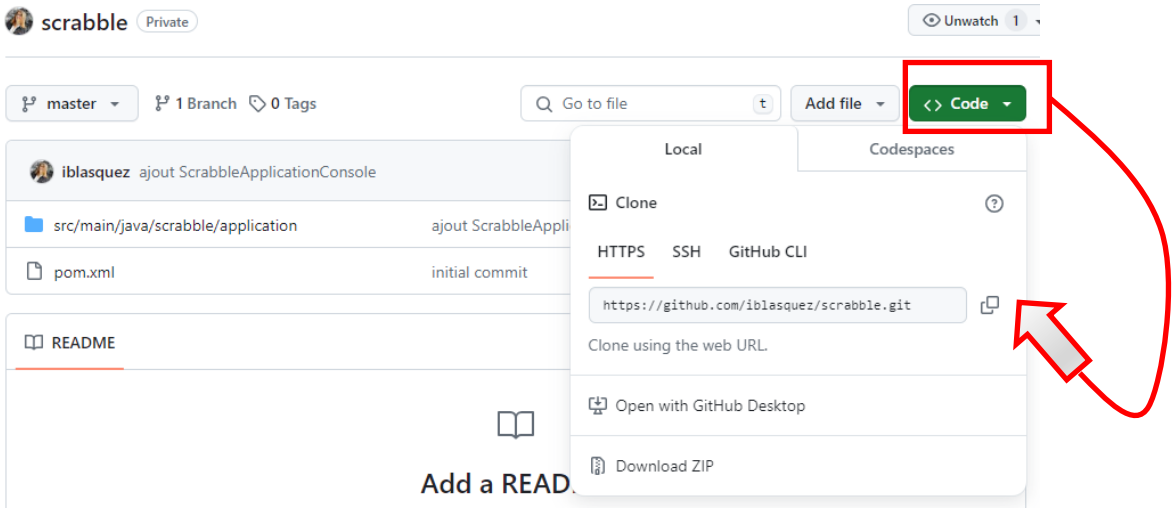
❑ *En tant que propriétaire*, une fois l'invitation validée par un collaborateur, le libellé *Pending Invite* disparaît et le mot **`Collaborator`** apparaît sous le nom du compte en question qui peut désormais interagir avec ce dépôt.

## 7. Pour tous les collaborateurs (hormis le propriétaire)

## Cloner le `remote` en local (récupérer & connecter le projet `scrabble` en local)

*Dans la suite, nous allons connecter le remote au local par GitKraken,*
*mais rien ne vous empêche d'utiliser à la place* ***la ligne de commande `git clone`*** 😉

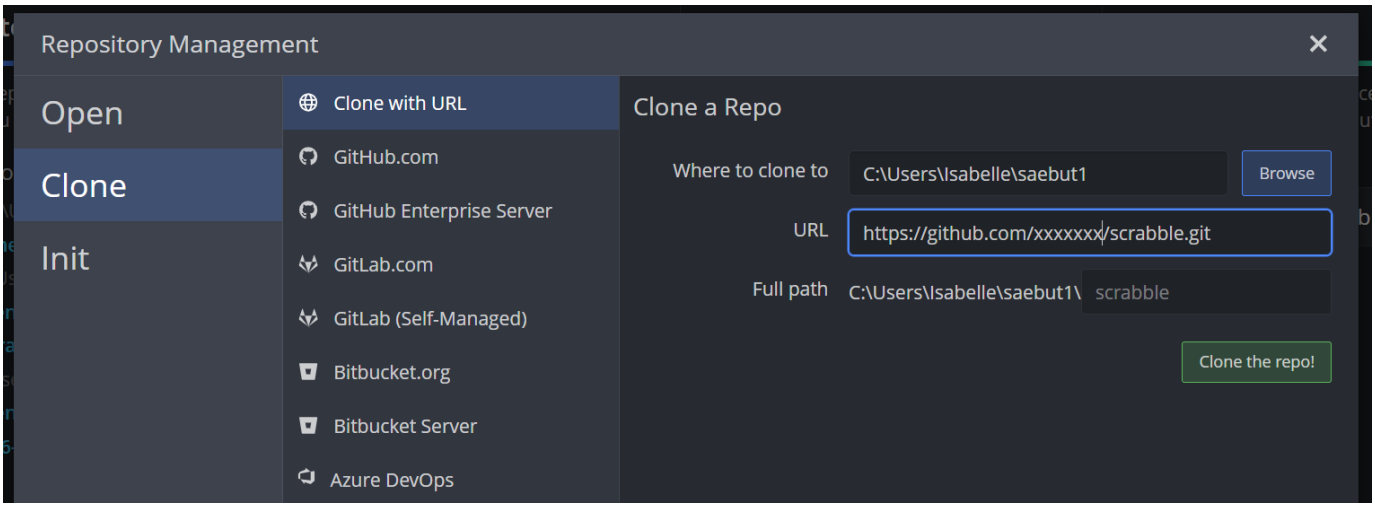
❑ Avant tout, rendez-vous **dans un navigateur web** sur le **dépôt** `scrabble` que **vous voulez cloner.**



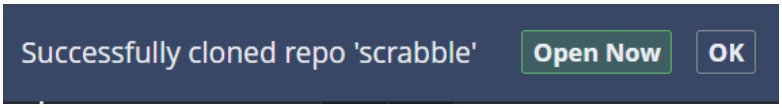

❑ Cliquez sur le bouton vert **Code** qui vous ouvrira un onglet **Local**, dans lequel vous pouvez **copier l'adresse https (en cliquant sur les deux carrés)**, cette adresse sera celle à utiliser lors de l'opération de clonage qui va suivre.

❑ **<u>En local,</u>** ouvrez **le Repository Management** de GitKraken :

- ⇨ Sélectionnez le menu **Clone** puis **Clone with URL**
- ⇨ Indiquez dans **where to clone** le chemin vers votre **workspace `saebut1`**
- ⇨ Collez dans **`URL Path`** l'adresse **https** (qui finit par **`.git`** et où **xxxxxx** est le compte du propriétaire) que vous venez de copier
- ⇨ Terminez en cliquant sur le bouton **Clone the repo !** .




Si le clonage s'est bien passé, vous devriez voir le message suivant 😉

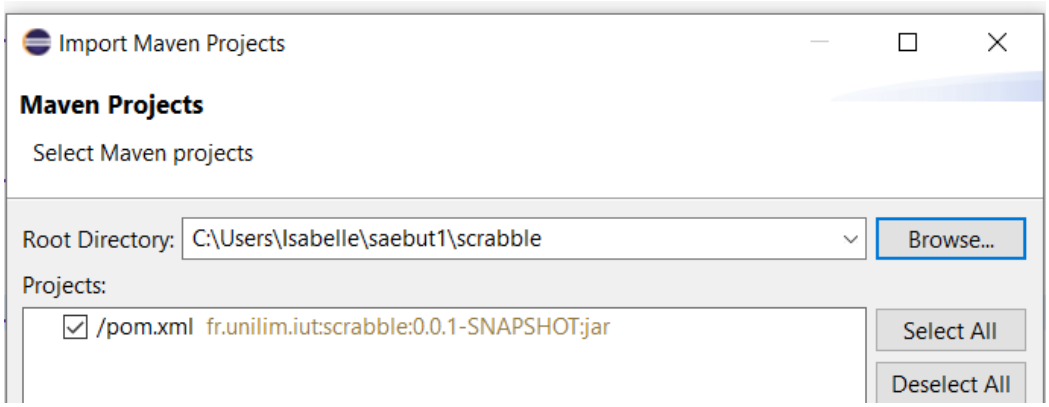


❑ **<u>En local,</u>** ouvrez **votre explorateur de fichiers** pour vérifier que votre **répertoire `saebut1`** contient bien un répertoire **`scrabble`**.

❑ **<u>En local,</u>** ouvrez Eclipse en sélectionnant **`le workspace saebut1`**.
Pour l'instant, ce workspace est vide car il est nécessaire d'importer le projet scrabble dedans.
La commande clone de git a juste rapatrié le code en local. A vous ensuite de mettre ce code dans l'IDE de votre choix et pour la SAE ce sera Eclipse 😉

❑ **Pour importer un projet maven java existant dans le workspace en cours**, rien de plus simple :

- ⇨ A l'aide d'un **clic droit**, sélectionnez **`Import`**
- ⇨ puis recherchez **`Maven`** et **`Existing Maven Projects`** et cliquez sur **`Next`**
- ⇨ Dans **Root Directory**, faites apparaître grâce au bouton **Browse le chemin en local jusqu'au projet scrabble** (que vous venez de cloner) .
- ⇨ Dans la rubrique **Projects**, vérifiez que la case devant **`/pom.xml`** est bien cochée.
- ⇨ Terminez en cliquant sur **`Finish`.**




❑ Si tout s'est bien passé, le projet **`scrabble`** doit apparaître dans la **vue `Package Explorer`** en tant que projet *maven* soumis au gestionnaire de version 😉

❑ En effet, si vous faites un clic droit suivi de **`Team` → `Show inHistory`** vous retrouvez l'historique git de ce projet.
Rappel : lors de votre premier commit en local, vous devrez configurer votre **.gitignore** en local
(on ne commite pas le `.gitnore` car il dépend de l'IDE sur lequel on travaille).

**…. Ne touchez pas au code pour le moment !**
**Attendez que chaque membre de votre équipe ait récupéré le projet en local avant de continuer et aidez-vous les uns les autres au besoin… 😉**

Vous devriez maintenant être tous en mesure de travailler en local et de synchroniser votre code avec le **remote** 😉

## 8. Pour tous les membres de l'équipe
## Premiers `push` & `pull`

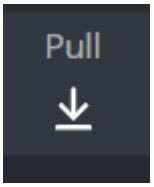
❑ Lorsque vous arrivez en début de séance, vous devez **tous synchroniser** votre code local avec le `remote` c-a-d **tirer(`pull`)** le code du dépôt distant vers le dépôt local.

❑ En fin de séance, vous devez **tous envoyer votre travail** sur le remote c-a-d **synchroniser** votre code local avec le remote en **poussant (`push`)** le code du dépôt local vers le dépôt distant.

Comme vous faites vos premiers pas avec *git*, je vous conseille de réaliser vos premiers **`Push`** et **`Pull`** à l'aide de GitKraken. Mais, je n'ai aucun doute que quand vous vous sentirez plus à l'aise, vous préférez la ligne de commande 😉

*Au travers de l'exercice suivant très simple,*
*nous allons simuler comment vous allez devoir collaborer au cours de cette SAE.*

❑ **Tous les membres de l'équipe réalisent un `Pull`** afin que tous les codes en local soient synchronisés avec le `remote`. Pour tirer le code du `remote` et resynchroniser votre historique *git* local, il vous suffira avec GitKraken d'utiliser le bouton `Pull`




❑ **Seul le collaborateur n°1 va pour l'instant toucher au code du projet.**

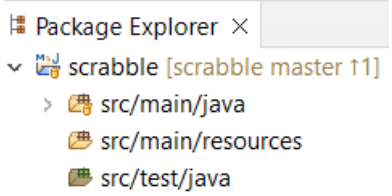
→ Depuis son IDE, le collaborateur n°1 ajoute une nouvelle instruction java dans la méthode **`main`** de la classe `ScrabbleApplicationConsole` afin que le message affiché soit désormais similaire au suivant (où Laurent est bien sûr remplacé par le prénom ou le pseudo du premier collaborateur 😉).

```
-----------------------------------------------------
--  Bienvenue dans notre magnifique jeu de scrabble !  --
--  développé par Laurent                              --
-----------------------------------------------------
```

→ Une fois le code fonctionnel et testé (visuellement), le collaborateur n°1 va **effectuer un commit** pour enregistrer la modification précédente **avec un message explicite** bien sûr 😉

→ Une fois le commit effectué, **consultez la vue Package Explorer** dans l'IDE du collaborateur n°1. Le prompt qui suit le nom du projet `scrabble` apparaît comme la copie ci-contre ce qui signifie :

- On est sur la branche **`master`**
- On est un commit « en avance » sur le remote : **`↑1`**
  (« en avance » est représenté par la flèche vers le haut ↑)
  ( le chiffre **1** signifie qu'on a **1** commit d'avance)




→ le collaborateur n°1 va ensuite **pousser les modifications qu'il a apportées au code existant vers le remote** pour resynchroniser le `remote` avec le nouveau code fonctionnel 😉 … avec par exemple, le bouton **`Push`** de **`GitKraken.`**

❑ **Tous les membres de l'équipe**

→ Si depuis un navigateur web, vous vous rendez maintenant sur la page github du dépôt, vous verrez que ce dépôt est désormais composé de **3 commits**.

→ Chaque membre de l'équipe va donc tirer le code du remote à l'aide d'un **`Pull`** pour que son historique local soit synchronisé avec le `remote`**.**

→ Chaque membre va ensuite ouvrir la classe `ScrabbleApplicationConsole` dans son IDE pour constater que la modification de code apportée par le collaborateur n°1 est bien redescendue dans le code de son projet **`scrabble`** local 😉

❑ **Seul le collaborateur n°2 va pour l'instant toucher au code du projet.**

→ Depuis son IDE, le collaborateur n°2 ajoute une nouvelle instruction java dans la méthode **`main`** de la classe `ScrabbleApplicationConsole` afin que le message affiché soit désormais similaire au suivant (où Anais est bien sûr remplacé par le prénom ou le pseudo du deuxième collaborateur)

```
------------------------------------------------------
--  Bienvenue dans notre magnifique jeu de scrabble !  --
--  développé par Laurent                              --
--  et par Anais                                       --
------------------------------------------------------
```

→ Une fois le code fonctionnel et testé (visuellement), le collaborateur n°2 va **effectuer un commit** pour enregistrer la modification précédente **avec un message explicite**

→ le collaborateur n°2 va ensuite **pousser son code vers le remote** (**`Push`**).

❑ **Tous les membres de l'équipe**

→ Consultez la page github de votre projet scrabble pour constater qu'il contient désormais un commit de plus.

→ Tirez le code du remote ( **`Pull)`** pour resynchroniser votre code en local.

→ Ouvrez ensuite la classe `ScrabbleApplicationConsole` pour vérifier que vous disposer bien du nouveau code en local.

Procédez de la même manière afin que tous les membres de l'équipe aient ajouté leur nom dans le message de bienvenue 😉

Remarque : Le propriétaire est également considéré maintenant comme un collaborateur du projet 😉

Quelques remarques par rapport au **travail collaboratif** :

- **Normalement, on commite lorsqu'on apporte un petit bout de code fonctionnel qui marche** afin de toujours disposer sur la branche d'une application opérationnelle.
  Bien sûr, dans la *vraie vie*, on ne commite pas à chaque instruction écrite, l'exemple précédent était juste pour vous familiariser de la manière la plus simple avec le principe du travail collaboratif.

- **On essaye de commiter fréquemment « *petit pas par petit pas* »** c.-à-d**. petit** bout de code fonctionnel par **petit** bout de code fonctionnel.

- Normalement, on ne pousse pas à chaque fois qu'on commite 😉

- Il faut toutefois essayer de **synchroniser le plus souvent possible le code avec votre `remote` pour éviter les conflits** : vous en aurez inévitablement …. Très souvent lorsqu'on implémente de manière collaborative, les conflits apparaissent. Afin d'éviter certains conflits, nous vous conseillons de fréquemment vous resynchroniser avec le dépôt.

- De bonnes pratiques, vous ont été énoncées dans un TP précédent, pensez à les revoir et les garder en mémoire. Parmi elles :
  - **Pensez à toujours synchronisez (`pull`)** votre code en arrivant (en début de séance)
  - **Pensez à toujours pousser votre code (`push`)** au minimum en partant (en fin de séance)
  - Il est recommandé de faire également **quelques `push` & `pull` (durant la séance)** pour éviter (ou résoudre le plus rapidement possible) des (éventuels) conflits :
    - Si vous travaillez sur des classes, des fonctionnalités ou des branches bien distinctes des autres membres de votre équipe, vous aurez peu de chance de voir apparaître des conflits
    - Mais **si vous travaillez sur le même code que d'autres membres de votre équipe, méfiance !!!** Des conflits vont surement apparaître plus fréquemment. Ainsi, pour limiter ces désagréments, il est recommandé de **`push` & `pull` plus souvent pendant vos séances de travail** 😉
  - **Toujours commiter un code qui fonctionne** (sauf éventuellement en fin de journée en mentionnant un **WIP**)

Chez mon client du jour 😀 #BlagueDeDev ?


**Surtout garder à l'esprit de faire des `push` et des `pull` très régulièrement pour rester synchroniser avec le `remote` et essayer ainsi d'éviter de trop gros/nombreux conflits** 😉

Image extraite de : https://twitter.com/agilex/status/836123942683308032