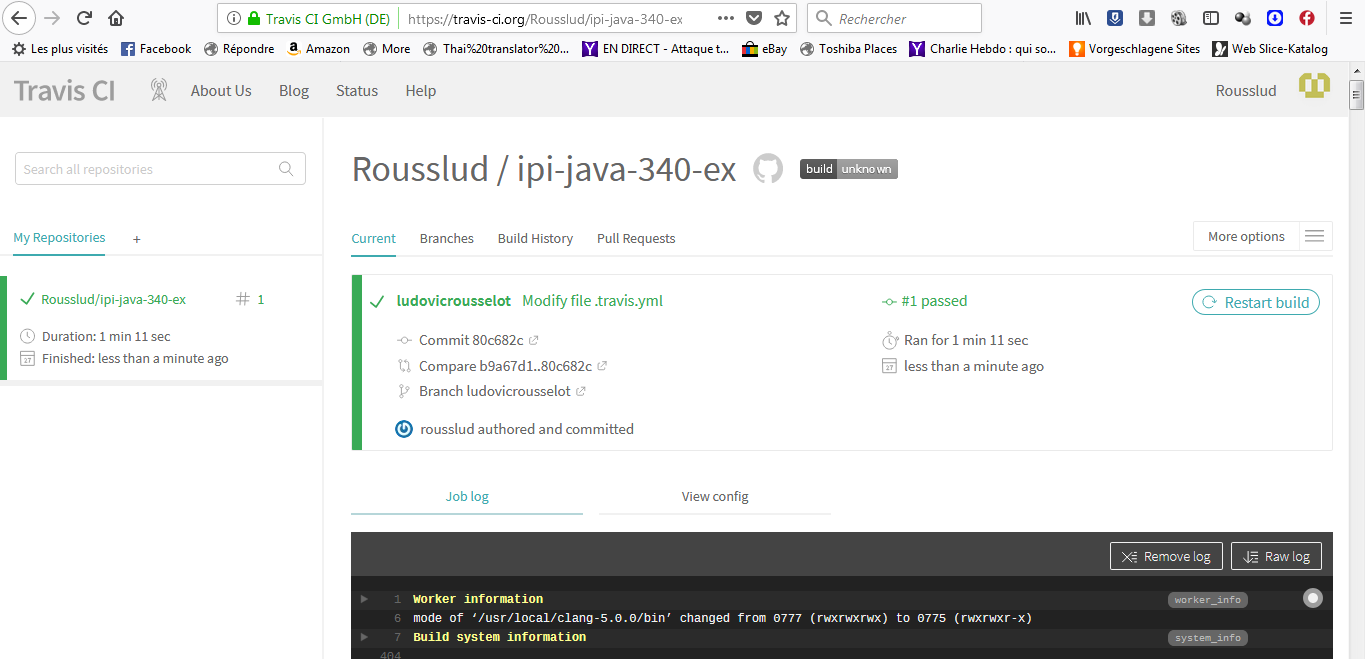
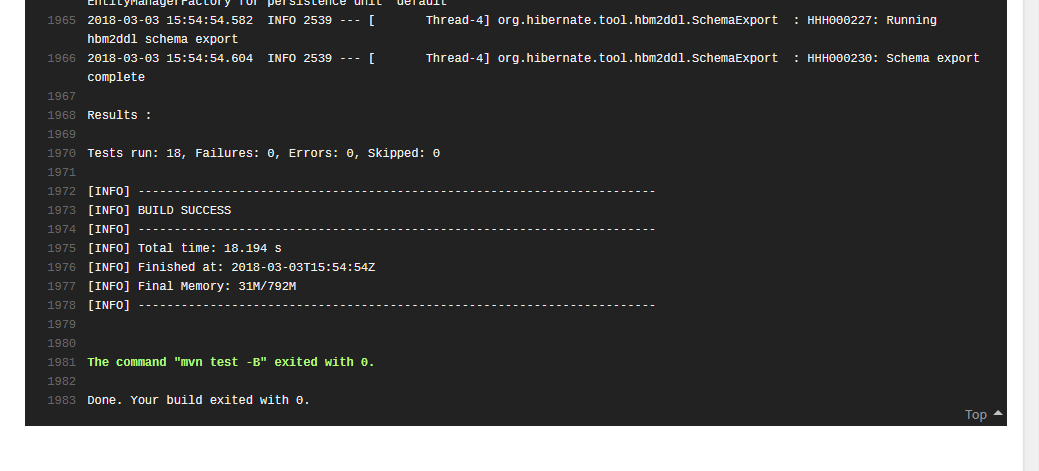
* Git commit : [Preparation: add file 'how to do'](https://github.com/Rousslud/ipi-java-340-ex/commit/b9a67d1165cce3c6c20e263eadc1720299f350c2)
* On a fait un peu de cours.

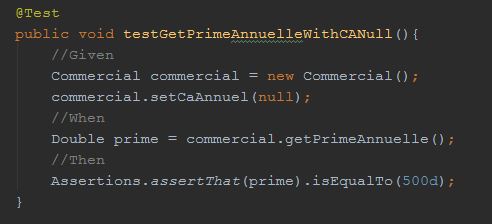
Attention, on est dans le module java 340, mais le cours sur github s’appelle java-350. Il s’est trompé dans le nommage. On a fait cours jusqu’à la page <https://pjvilloud.github.io/ipi-java-350-cours/#/tools-continuous>

* Configuration du projet :
  + Attention, prendre le java-340-ex. Il faut obligatoirement le forker. Ensuite, le cloner pour travailler en local.
  + Prendre le readme du java-350-ex, et l’ajouter à ton repo local. Une fois fais, commiter et pusher.
* Utiliser eclipse ou intellij pour importer le projet
* NOTA BENE : Eclipse peut demander à installer l’extension STS
* Te connecter sur Travis : <https://travis-ci.org> et te connecter avec ton compte github
  + Chercher ton repo distant java-340-ex et cliquer sur Activate repo
* Dans eclipse, créer un fichier .travis.yml à la racine du projet (au même niveau que .gitignore).
  + Voir sur <https://docs.travis-ci.com/user/languages/java/> ce que tu dois mettre dans le fichier .travis.yml (y a juste une seule ligne ;))
  + Commiter le fichier puis pusher.
  + Git commit : [Modify file .travis.yml](https://github.com/Rousslud/ipi-java-340-ex/commit/80c682cef350249944bdbe9de54c6ca530d0b53f)
  + Normalement, en attendant un peu, tu auras sur le site travis, un fichier log qui apparaîtra pour dire que c’est OK

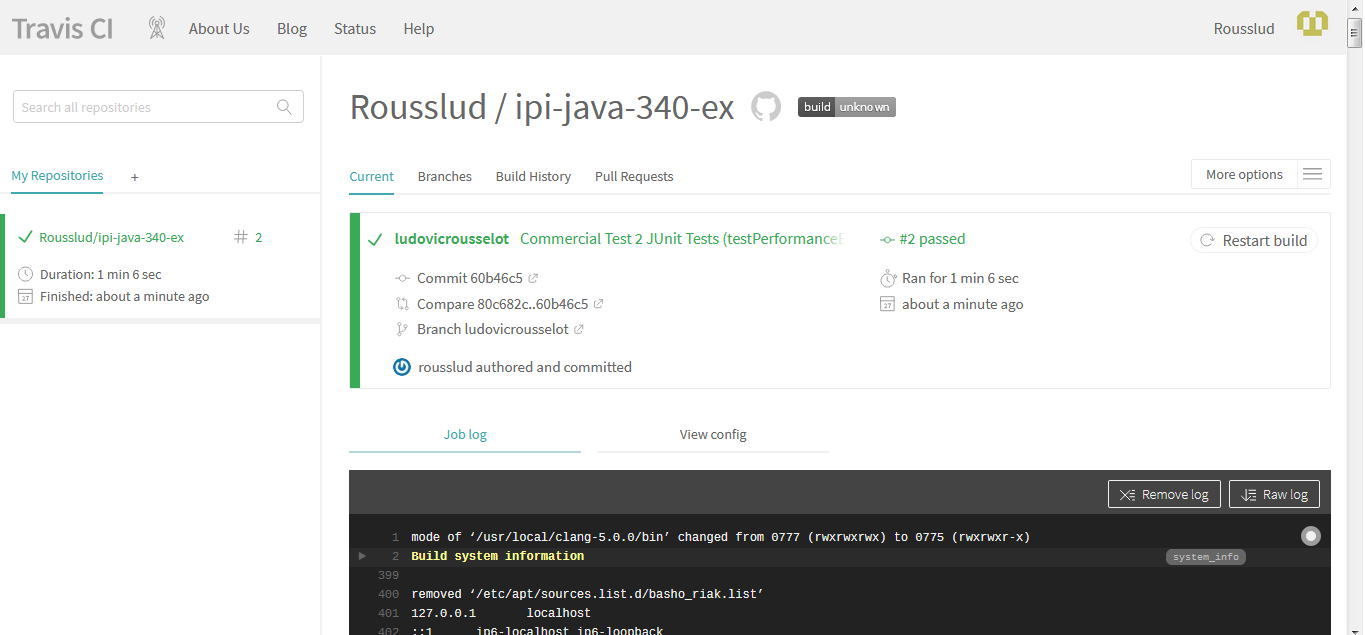


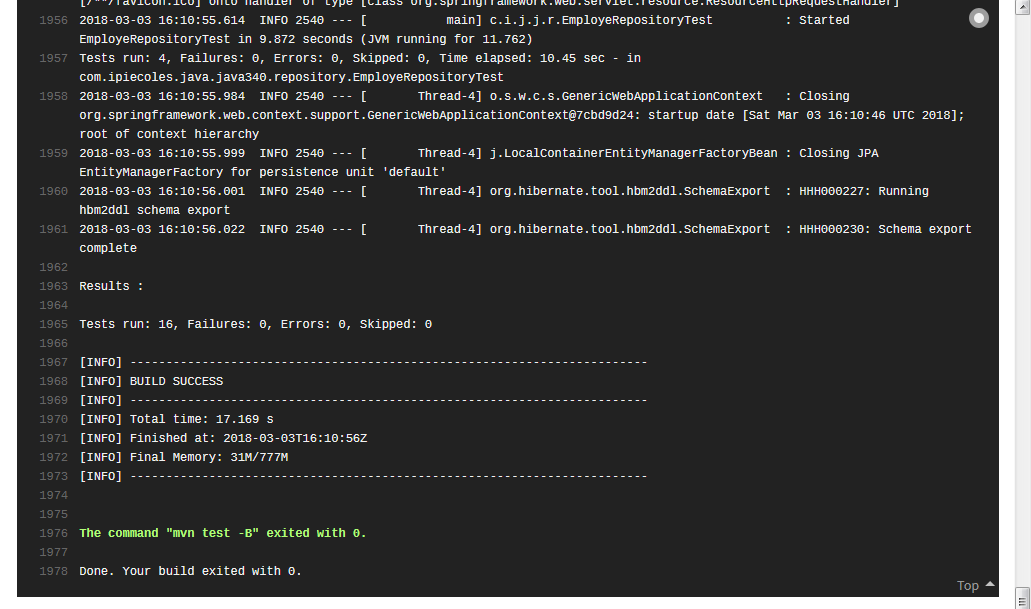


* Cours : de <https://pjvilloud.github.io/ipi-java-350-cours/#/fiabilite> à <https://pjvilloud.github.io/ipi-java-350-cours/#/fiabilite-unit-junit-classes>
* Exo : voir le readme, ce sont les deux premières questions de la partie « Tests unitaires »
* Correction :
  + Le fichier CommercialTest.java doit être créé dans l’arborescence suivante : src/test/java/com.ipiecoles.java.java340.model
  + Voici le test que le prof a fait :



* Ensuite, sous Eclipse, faire un « Run As JUnit test » pour voir que tes tests fonctionnent.
* Si ça fonctionne, commiter / pusher.
* Git commit : [Commercial Test 2 JUnit Tests (testPerformanceEquality + testGetPrimeAnnuelleWithCANull](https://github.com/Rousslud/ipi-java-340-ex/commit/60b46c5a9b32c5a0d9b68022ff0bc4964e27bf68))
* Attendre un peu, et voir si sur travis, le log a changé :



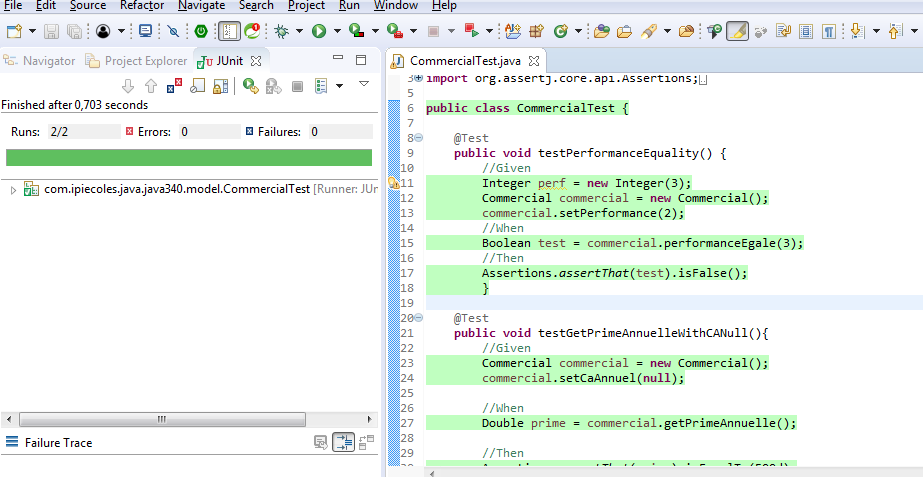


* Faire d’autres test à toi. ~~Attention, ce sont ces tests qui seront notés, donc faut que ça passe ;)~~ je me suis trompé. Ce qui sera noté, ce sera des tests sur les autres classes (Manager, Technicien…) à faire soi-même

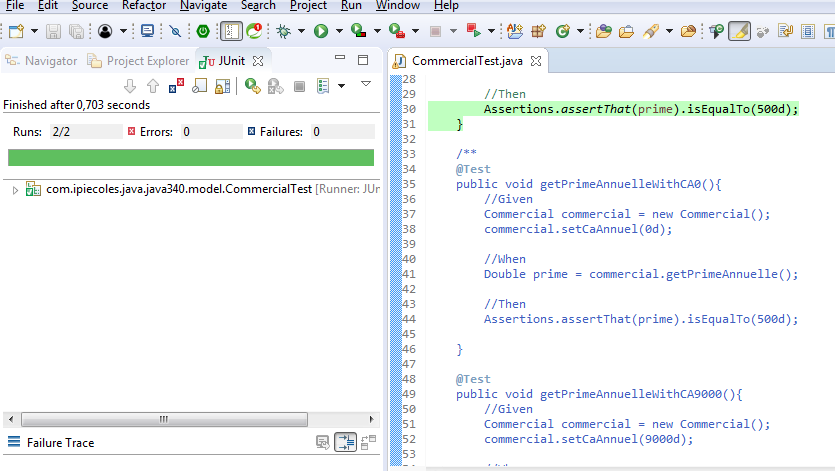
**13/02**

* Il nous a appris une astuce : clic droit sur ta classe de test (CommercialTest) et faire « Coverage as JUnit Test». Cela permet de voir la couverture de code qui est testé par la classe de test

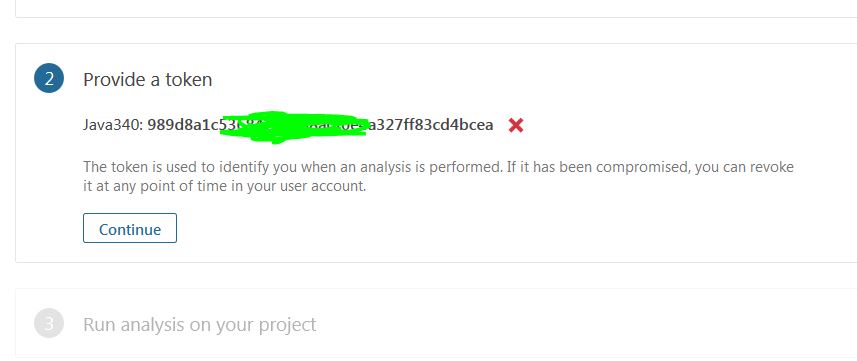
Couvert :



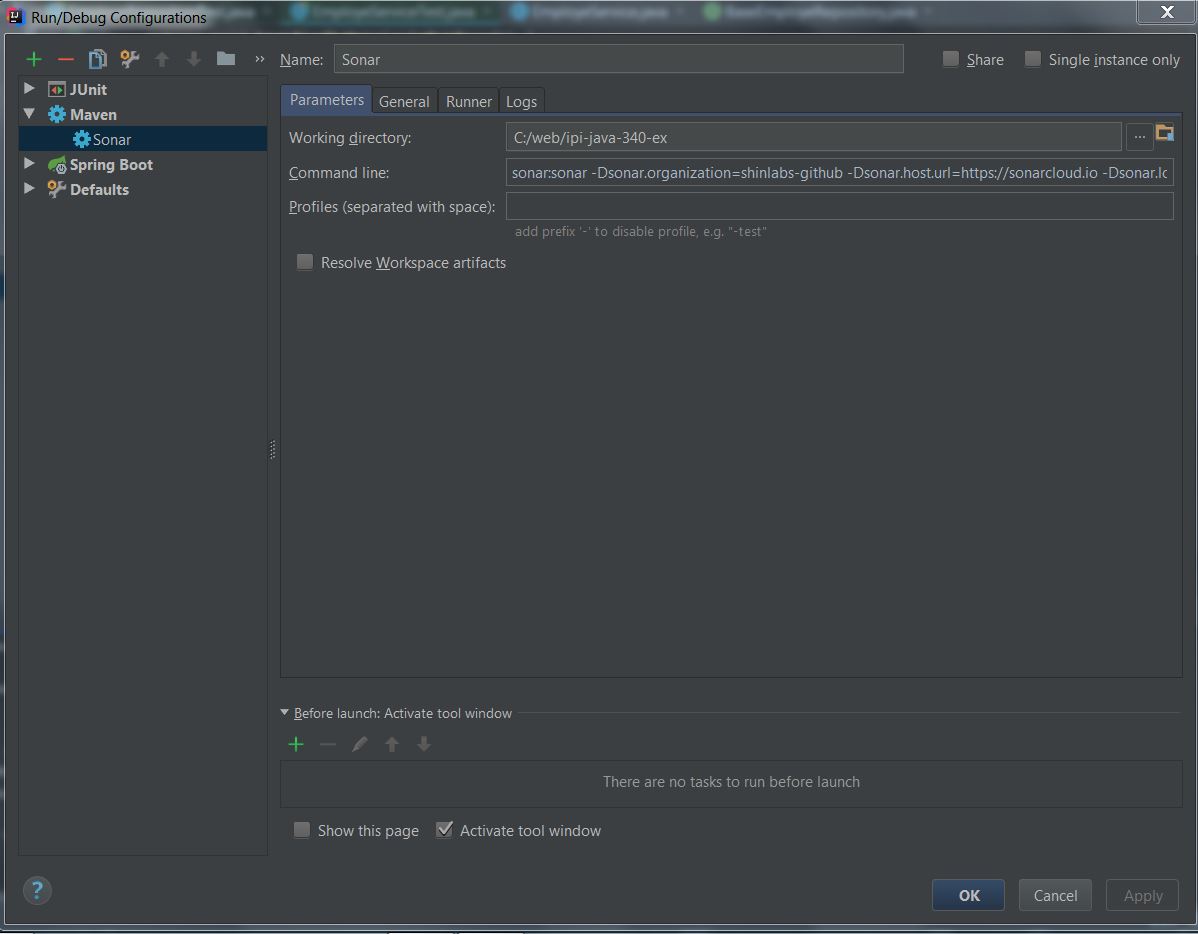
Non couvert :



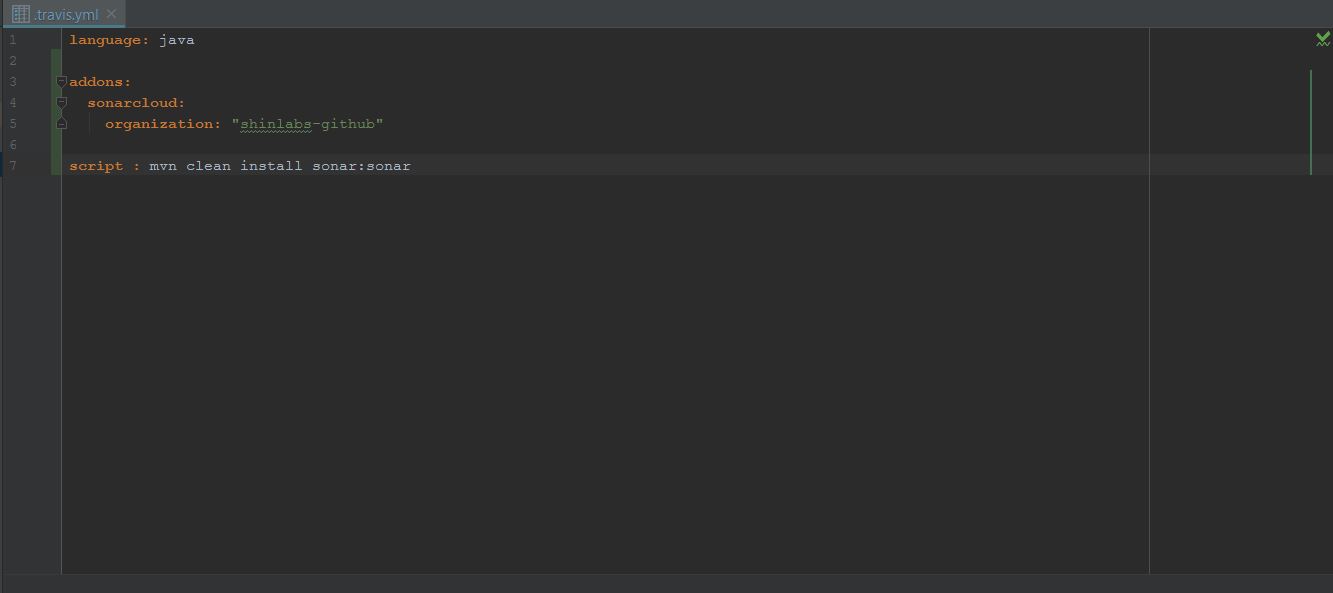
* Autres tests de CommercialTest :
  + testPerformanceEquality() devient testPerformanceNotEquals() (le nom n’était pas cohérent)
  + testGetPrimeAnnuelleWithBigCA()
  + getPrimeAnnuelleWithCA0()
  + getPrimeAnnuelleWithCA9000()
  + testgetPrimeAnnuelleWithCA100000()
* Git commit : [Commercial Test JUnit Tests : more tests](https://github.com/Rousslud/ipi-java-340-ex/commit/f9a79db47b459515b298021aa03750fcd26c271d)
* On a vu du cours : <https://pjvilloud.github.io/ipi-java-350-cours/#/fiabilite-unit-junit-parametres>
* Faire des tests avec parameters pour la methode equivalenceNote de la classe Commercial
  + (non fait) : Une autre façon de faire des parameters, un peu plus simple, mais qui nécessite une nouvelle dépendance à rajouter au pom.xml. (maj : pas obligatoire, le prof ne l’a pas fait) Voir ici : <https://github.com/Pragmatists/JUnitParams>
  + Git commit : [Parameterized Tests: EquivalenceNote](https://github.com/Rousslud/ipi-java-340-ex/commit/1648a97f32c4fc9a377d7c4f28908cc6b51b7a4f)
* On a vu du cours : <https://pjvilloud.github.io/ipi-java-350-cours/#/fiabilite-unit-junit-ressources>
* Créer un fichier application.properties dans : src/test/resources. Y copier/coller ce qu’il y a dans le cours
  + Pour l’instant, on ne peut pas faire de test sur les méthodes des repositories, car il faut que Spring soit lancé en même temps que les tests. D’où le point suivant :
  + Git commit : [Modify application.properties to be able to use a H2 memory database](https://github.com/Rousslud/ipi-java-340-ex/commit/a421a76feeb90c5bdf0d8f98ceb2d07b1d18d92a)
* On a vu du cours : <https://pjvilloud.github.io/ipi-java-350-cours/#/fiabilite-unit-junit-spring> et <https://pjvilloud.github.io/ipi-java-350-cours/#/fiabilite-unit-junit-beforeafter>
* Exo partie TU : 5 et 6 -> voir le commit « EmployeRepositoryTest OK » (comme toujours, commiter / pusher et voir sur travis si ça s’est bien passé)
* Git commit : [EmployeRepositoryTest with @DataJpaTest](https://github.com/Rousslud/ipi-java-340-ex/commit/d6443bd759928d2f5885fd62810b2c5106468751)
* Rajouter un plugin : EMF builder generator dans Eclipse (aller dans marketplace) et Builder Generator – Code tools dans intellij (aller dans File>Settings>Plugins>Browse Repositories)
* Cours sur mock : <https://pjvilloud.github.io/ipi-java-350-cours/#/fiabilite-unit-junit-mock> à <https://pjvilloud.github.io/ipi-java-350-cours/#/fiabilite-unit-junit-mock2>
* Faire du mock sur EmployeService avec EmployeServiceTest
* On va tester notre code avec sonar.
  + Se rendre sur <https://about.sonarcloud.io/> et se connecter avec le compte github
  + Cliquer sur analyse new project
  + Cliquer sur continue
  + Mettre « Java340 » et cliquer sur generate
  + Copier le token quelque part



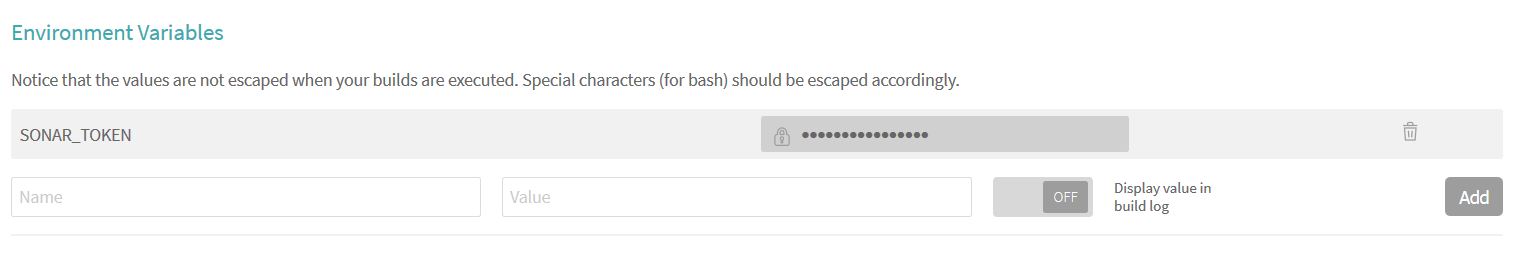
* + Cliquer sur continue puis java, puis maven
  + Copier le morceau de command line qui apparaît
  + Cliquer sur finish
  + Aller dans Intellij : Run>Edit configurations
  + Appuyer sur + puis choisir Maven
  + Configurer comme la photo suivante (attention, on a enlevé mvn et les slashs dans la partie « command line »)



* + Appuyer sur OK
  + Puis Run (build)
  + Et là, on se rend compte que ça marche pas haha
* Donc on essaye de repasser par Travis pour tester notre code
  + Ouvrir le fichier .travis.yml
  + Rajouter son organisation. En général « nom\_du\_compte-github ». Attention à l’indentation !



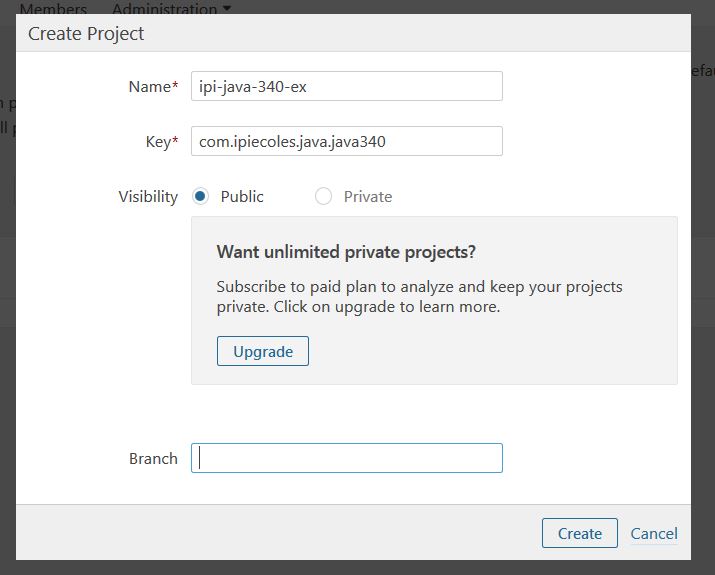
* + Commiter / pusher
  + Aller sur travis > more options > settings
  + Rajouter une variable environnement avec le nom comme sur l’image suivante, et la value c’est le numéro token qu’on a copié plus tôt. Appuyer ensuite sur Add et obtenir l’image suivante



* + Bon ça marche toujours pas. La suite au prochain épisode (il regardera ça pour mercredi). En gros sonar permet d’avoir un retour sur la présence de bugs, la présence de vulnérabilité, le pourcentage de code testé…
* Cours : <https://pjvilloud.github.io/ipi-java-350-cours/#/fiabilite-metriques>

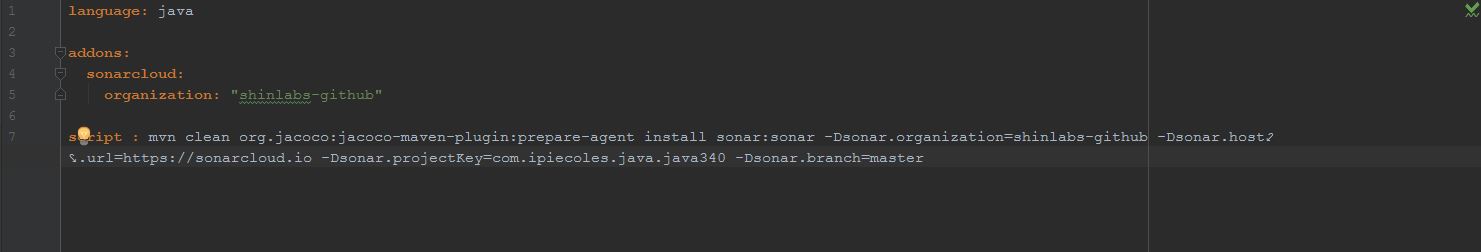
**14/02**

* Pour enfin faire marcher sonarcloud : en fait il faut créer un projet avec une nouvelle clé
  + Aller voir sur https://sonarcloud.io/organizations/«nomgithub»-github/projects qu’il n’y a aucun projet.
  + Aller dans Administration > project management, puis cliquer sur « nouveau projet »
  + Remplir la fenêtre comme suit :



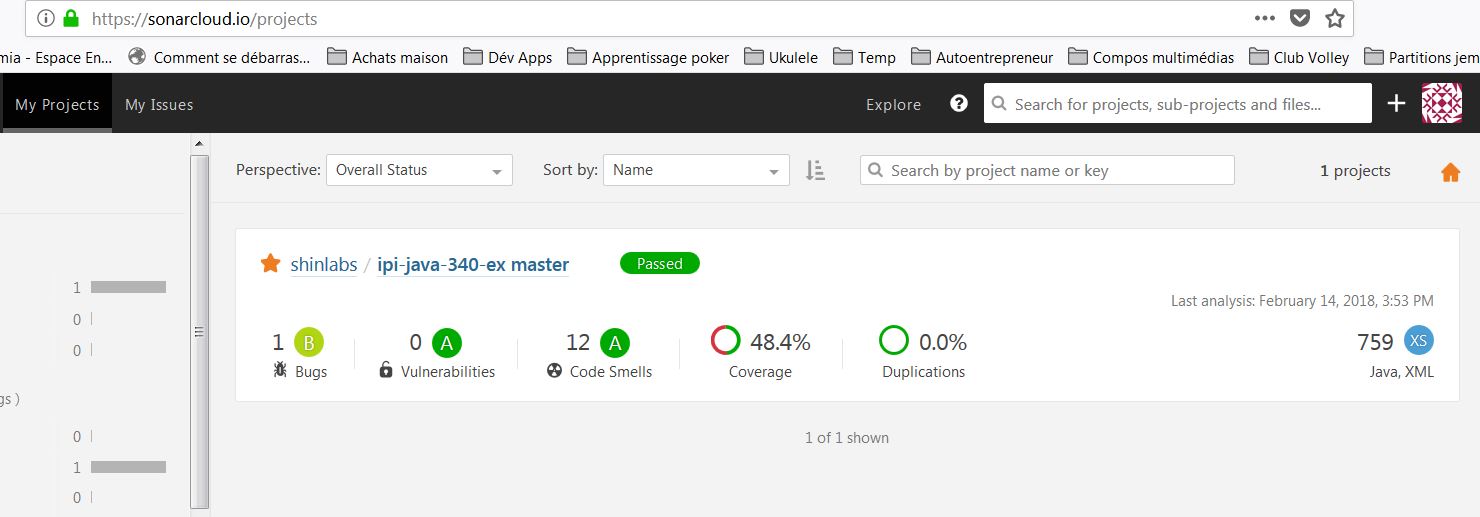
laclequetuveux

* + Appuyer sur create. S’il y a une erreur (clé déjà existante), c’est pas grave, passer à l’étape suivante
  + Aller sur .travis.yml (eclipse / intellij) et le remplir comme suit (ne pas oublier d’adapter le shinlabs-github



laclequetuveux

* + Commiter / pusher, et vérifier que tous les tests du projet passent
  + Voir sur travis si le build est bien passé
  + Voir sur sonarcloud si le projet a bien été testé :



Travail à rendre

Refaire un test classique avec Manager et l’une des trois méthodes suivantes : setSalaire, getPrimeAnnuelle, augmenterSalaire.

Faire aussi tests paramétrées sur l’une des trois méthodes restantes précédentes.

Faire quelque chose comme avec findByNomOuPrenomAllIgnoreCase mais avec findEmployesPlusRiche (cas nominaux et un ou deux cas limites)

Faire du Mock avec ManagerService avec la méthode addTechnicien