 MASTER DÉVELOPPEMENT LOGICIEL	M1 Info
	Développement collaboratif, Qualité

TP 2 – Prise en main d'Apache Maven, IntelliJ¹ et JUnit

Ce TP et les suivants s'effectuent dans l'environnement de développement IntelliJ IDEA Ultimate, avec Java 8 (JDK). Référez-vous au message posté sur Moodle si vous avez votre propre laptop.

Si vous utilisez les machines de TP de l'UPS :

1. Démarrez sous GNU/Linux Fedora.
2. Lancez IntelliJ en tapant « `idea.sh` » dans un terminal.
3. Choisissez l'option « Licence Server » et spécifiez l'adresse du serveur de licence (en faisant attention à ne pas introduire d'espaces avec le copier-coller) :
`http://fsi-ens-vjetons8.univ-tlse3.fr:8080`

Remarque concernant Maven :

IntelliJ embarque l'outil Maven et le dossier `~/.m2` correspond au repository Maven local ⇒ toutes les dépendances de vos projets seront rapatriées dans ce dossier. (Ce dossier n'est donc **pas spécifique** à l'UE DeQo et pourra être utilisé dans les autres UEs utilisant Java.)

Exercice 1 – Création du projet my-simple-stack

Dans cette exercice on verra comment Maven gère les plugins ajoutés à un projet.

1. Dans IntelliJ, créez un nouveau projet de type Maven en **cochant l'option « Create from archetype »** puis sélectionnez l'archetype « **maven-archetype-quickstart** » version **RELEASE** (Il existe plusieurs archetypes du type « quickstart » dans la liste. Faites attention à bien choisir celui de MAVEN !). Si la question vous est posée, **spécifiez le JDK** : créez un nouveau JDK pointant sur le JDK « `jdk1.8.0_172` ».
Configurez votre projet en choisissant le **groupid** « `deqo.votre_quadrigramme` » et l'**artifactId** « `my-simple-stack` ». Remplacez `<votre_quadrigramme>` dans groupId par votre quadrigramme. Note : le **quadrigramme** de François Dupond est « `fdup` ».
2. Choisissez d'utiliser la version de Maven 3 embarquée avec IntelliJ. Ajoutez une propriété « **package** » avec la valeur « `deqo.votre_quadrigramme.mysimplestack` » (mysimplestack sans trait d'union, puisque un nom de packaging Java ne peut pas contenir de traits d'union). Une fois le projet créé, vous développerez vos classes dans le **package Java** « `deqo.votre_quadrigramme.mysimplestack` ».
3. Spécifiez le nom du projet « `my-simple-stack` » et la localisation du projet dans un dossier ad-hoc de votre espace de travail.
4. Rendez visible la fenêtre « Maven projects » (menu View → Tool Windows → Maven Projects) puis cliquez sur le bouton « Reimport all Maven projects » pour resynchroniser la configuration Maven avec votre IDE.
5. Notez la présence d'un élément `<build><pluginManagement>... </pluginManagement></build>` généré automatiquement dans le POM. Celui-ci est important pour *fixer la version de certains plugins Maven* dont la version par défaut a récemment causé des incompatibilités²

¹ IntelliJ IDEA Java édité par la société JetBrains

² cf. <https://stackoverflow.com/q/51091539> et <https://stackoverflow.com/q/23260057>



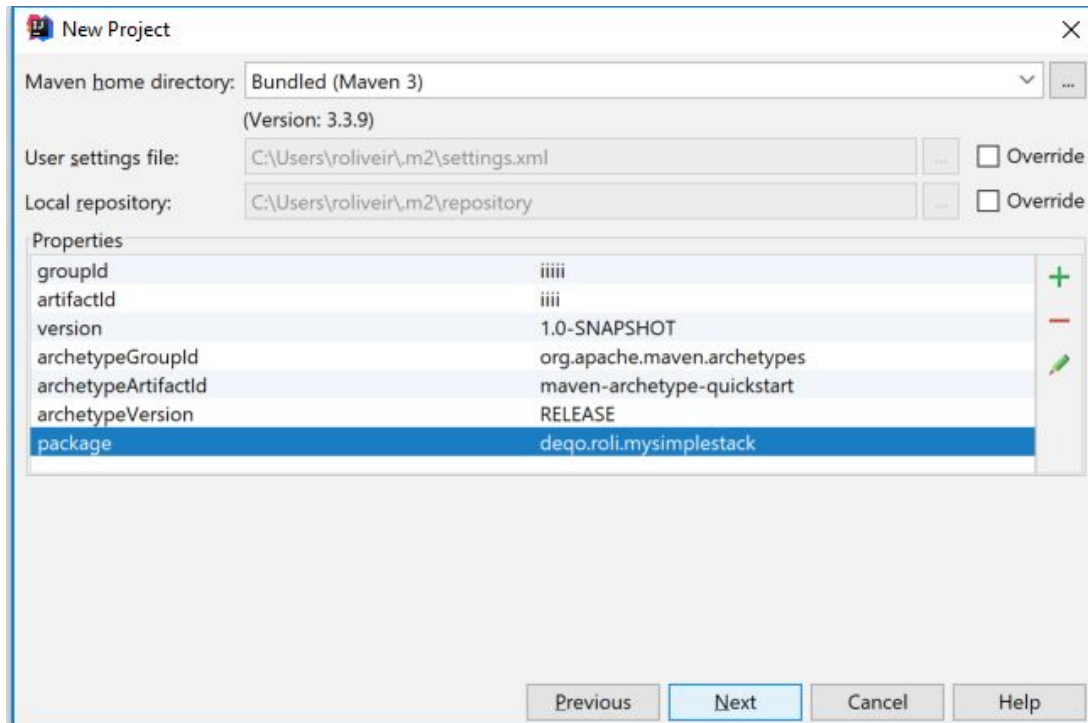


Figure 1. Configuration de Maven et choix du « Local repository »

6. Modifiez le POM de votre projet de sorte que celui-ci utilise Java 1.8 et la dernière version de JUnit 4 (= 4.12) à la place de la version 3.8.1. Les informations de ce type (sur les dépendances) sont à rechercher sur le site <http://search.maven.org/>. Cliquez sur le bouton « Import Changes » ou « Reimport all Maven projects » pour synchroniser la configuration Maven avec l'IDE.
7. Nous allons ajouter les plugins ci-dessous. Après avoir ajouté le code XML correspondant (p.ex. grâce aux diapositives 36/52 et 37/52 du [cours Maven](#)), prenez le temps de suivre les liens ci-dessous pour réviser à quoi sert chacun.
 1. JXR - <https://maven.apache.org/jxr/index.html>
 2. Checkstyle - <https://maven.apache.org/plugins/maven-checkstyle-plugin/index.html>
 3. FindBugs - <http://www.mojohaus.org/findbugs-maven-plugin/index.html>
 4. JaCoCo - <https://www.jacoco.org/jacoco/index.html>

Note: la version de FindBugs est bien 3.0.3, pas 3.0.4-SNAPSHOT !
8. Synchronisez la configuration Maven de votre projet pour qu'il utilise ces plugins. Rappel : ces plugins sont dans la catégorie « Reporting » et seront ainsi utilisés lors de la génération du site du projet par Apache Maven (cf. support de cours).

NOTE : il est possible qu'IntelliJ colore en rouge le nom des plugins dans le POM ; ce n'est pas un problème et c'est typique lors de la première utilisation des plugins, et cela disparaît en principe quelques instants après le premier lancement réussi des phases « clean test site » (cf. question 9).





9. Spécifiez deux configurations de lancement Maven (menu Run→Edit configurations, puis clic sur « + » (en haut à gauche) et sur « Maven ») pour :
 1. Le lancement des phases clean et package. Nommez-la « clean package », ensuite tapez « clean package » dans le champ « Command line »;
 2. Le lancement des phases clean, test et site. Nommez-la « clean test site », ensuite tapez « clean test site » dans le champ « Command line ».
10. Testez vos 2 configurations en les lançant l'une après l'autre. Explorez le dossier « target » de votre projet pour constater le résultat des 2 commandes Maven. À l'aide d'un navigateur, explorez en détail le site généré par Maven (y compris les pages générées par les plugins), en cliquant avec le bouton droit sur « **target/site/index.html** », puis “Open in browser->...”. Sur le site web, les rapports générés par les plugins se trouvent dans le menu “Project Reports” à gauche.

Exercice 2 – Gestion du projet my-simple-stack avec Git et IntelliJ

Vous allez maintenant voir comment intégrer le système de contrôle de version Git à votre projet dans IntelliJ.

1. Dans le menu VCS cliquez sur « Enable Version Control Integration ». Sélectionnez Git comme *version control system*. Contrôlez avec le terminal intégré d'IntelliJ (menu View→Tool Windows→Terminal ou raccourci Alt+F12) que le dossier .git a bien été créé à la racine de votre projet (commande `ls -la`).

Conseil si vous êtes sous Windows : pour utiliser Git Bash comme terminal dans IntelliJ, cliquez sur File→Settings→Tools→Terminal, puis spécifiez le chemin de **vos** Git Bash dans le champ « Shell path ». Exemples : « "C:\Program Files\Git\bin\sh.exe" --login -i » ou « "C:\Git\bin\sh.exe" --login -i ». Après cette étape, (re)lancer le terminal d'IntelliJ puis faites « `ls -la` » pour vous assurer de la création effective du dossier .git à la racine du projet.

2. Dans IntelliJ, à la racine de votre projet supervisé à présent par Git, créez le fichier « .gitignore » permettant d'ignorer :
 - les fichiers ayant pour extension « .class »,
 - le dossier « target/ »,
 - le dossier « .idea/ »
 - les fichiers ayant pour extension « .iml ».


Remarque : la commande `cat > fileName` permet de créer et éditer le fichier “fileName”. Une fois l'édition terminée, Ctrl+D pour sortir du mode édition de fichier. Ensuite, la commande `cat fileName` permet de visualiser le contenu du fichier.

3. Rendez visible la fenêtre « Version Control » (menu View→Tool Windows→Version Control) puis ajoutez l'ensemble des fichiers à l'index (cliquez droit sur l'item « Unversioned files » puis sur « add to VCS »). Effectuez le « commit » de vos fichiers après avoir renseigné le « commit message » (clic droit sur l'item « Default » puis sur « Commit »).
4. Créez sur votre compte GitHub un projet (repository) nommé « my-simple-stack ».
5. Via le terminal (menu View→Tool Windows→Terminal), ajoutez à votre projet local le remote « origin » référençant le projet my-simple-stack (cf. cours de Git sur moodle).

Rappel : pour un **accès HTTPS** l'URL du dépôt GitHub commencera par « https://... ».

6. Effectuez la synchronisation du remote « origin » avec votre projet local (menu VCS→Git→Push). Vérifier sur votre compte GitHub que le projet « my-simple-stack »



 MASTER DÉVELOPPEMENT LOGICIEL	M1 Info
	Développement collaboratif, Qualité

correspond bien à celui de votre dépôt local.

Exercice 3 – Création des cas de test avec JUnit

1. Écrivez dans le projet « my-simple-stack » un programme permettant d'implémenter l'interface SimpleStack ci-dessous. Écrivez les interfaces et classes nécessaires.

```

public interface SimpleStack {
    /**
     * Tests if this stack is empty
     */
    public boolean isEmpty();

    /**
     * Returns the number of items in this stack.
     */
    public int getSize();

    /**
     * Pushes an item onto the top of this stack.
     * null item is allowed.
     */
    public void push(Item item);

    /**
     * Looks at the object at the top of this stack without removing it from the stack.
     */
    public Item peek() throws EmptyStackException;

    /**
     * Removes the object at the top of this stack and returns that object as the value of this function.
     * @throws EmptyStackException if this stack is empty.
     */
    public Item pop() throws EmptyStackException;
}

```

2. À partir du code d'une classe (par exemple Item) IntelliJ peut vous aider à générer un squelette de classe de test (ItemTest). Il suffit de cliquer sur le nom de la classe dans le fichier Item.java, taper Alt+Entrée, choisir Create Test, puis Entrée. Dans la fenêtre, choisissez JUnit 4, et cochez les méthodes à tester.
3. En vous inspirant des supports de cours de JUnit, écrivez des tests pour la classe Item. Utilisez le template *Given/When/Then*. Pour les exécuter, cliquez avec le bouton droit sur le nom de la classe ItemTest, puis "Run ItemTest". Faites plusieurs essais pour voir le feedback de JUnit. Exemple : des tests qui passent, des tests qui ne passent pas à cause de problèmes dans la classe, et à cause de problèmes dans le cas de test.
4. Introduisez les tests nécessaires de telle sorte que la **couverture du code par les tests** soit de 100%.
Rappel: Pour voir la couverture de tests de votre projet, utilisez le rapport généré par le plugin "Cobertura", ajouté dans le projet dans l'exercice 1.7.4. Pour voir le rapport, il faut suivre les étapes de l'exercice 1.10.
5. À l'aide d'IntelliJ, effectuez le « commit » et le « push » de vos fichiers sur GitHub.



Cette œuvre est mise à disposition selon les termes de la [Licence Creative Commons Paternité - Pas d'Utilisation Commerciale 3.0 France](#).