TD n°1 – Atelier de développement java

Objectif

L'objectif de de ce TD est de prendre en main l'atelier de développement IntelliJ de JetBrain dans sa version communautaire en mettant en place sur un code simple, les tests unitaires et leur couverture, l'analyse de code et la construction automatique de l'application. Nous nous concentrons ainsi, pour l'instant, sur les outils de production du développeur en dehors d'un groupe projet.

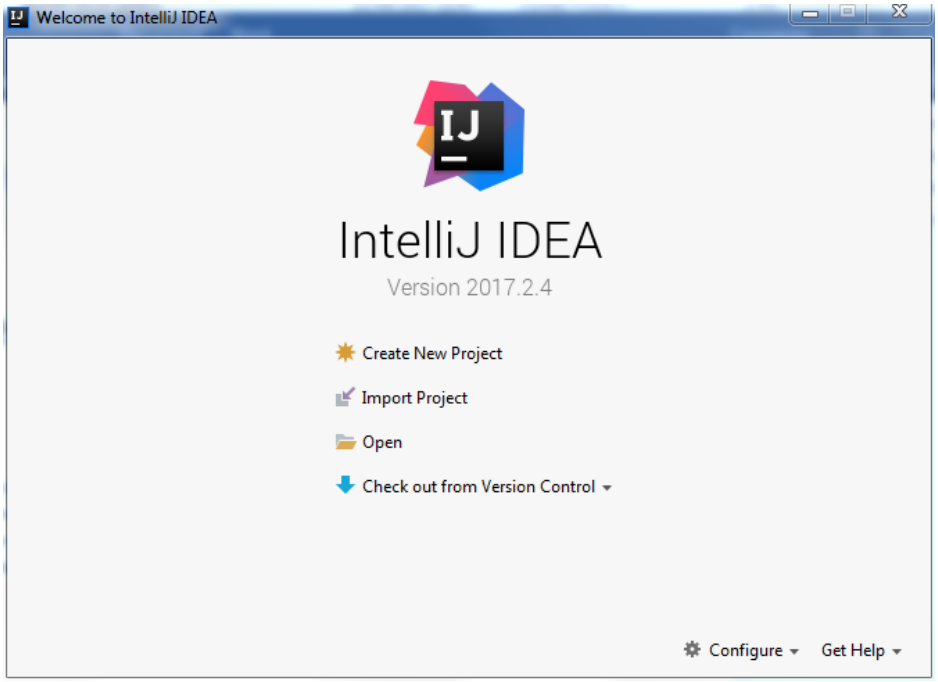
Installation du jdk 1.8.0_131

L'installation peut se faire sur votre portable ou sur un disque D : des machines de l'IUT. Il est conseillé de faire un **répertoire plateforme-production** dans lequel vous installerez tous les outils.

1) Décompresser le fichier jdk1.8.0_131 disponible sur commun dans votre répertoire **plateforme-production**

Installation d'intelliJ

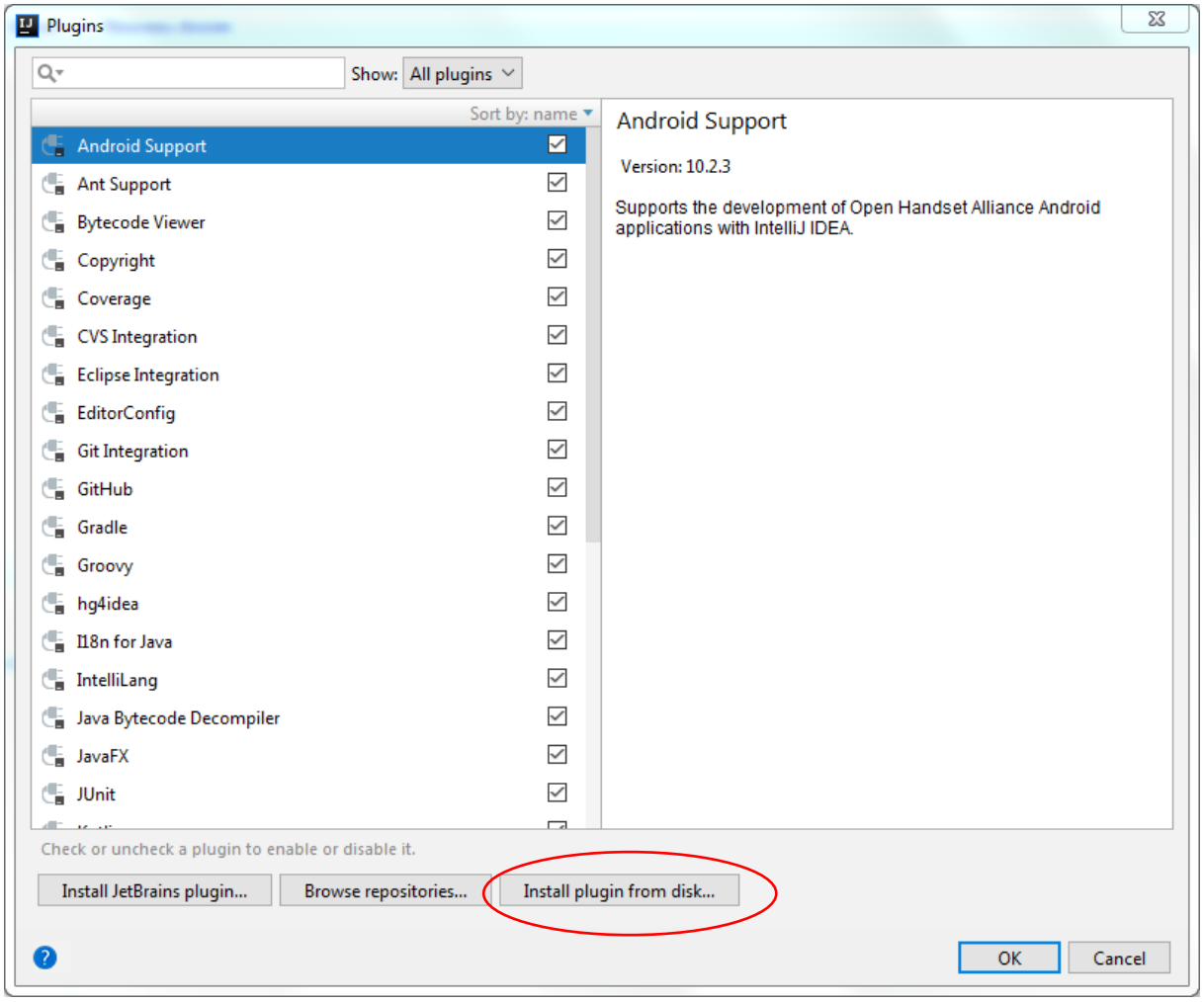
1) Décompresser le fichier **ideaIC-2017.2.4.win.zip** disponible sur commun dans le répertoire **intellij** du répertoire **plateforme-production**
2) Lancer intelliJ avec l'exécutable **idea64.exe situé dans le répertoire bin**
    a. **Ne rien importer**
    b. **Choisir les paramètres par défaut (ne pas créer de projet)**



Installation du plugin checkstyle

Ce plugin permet de faire de la vérification de code à la volée.

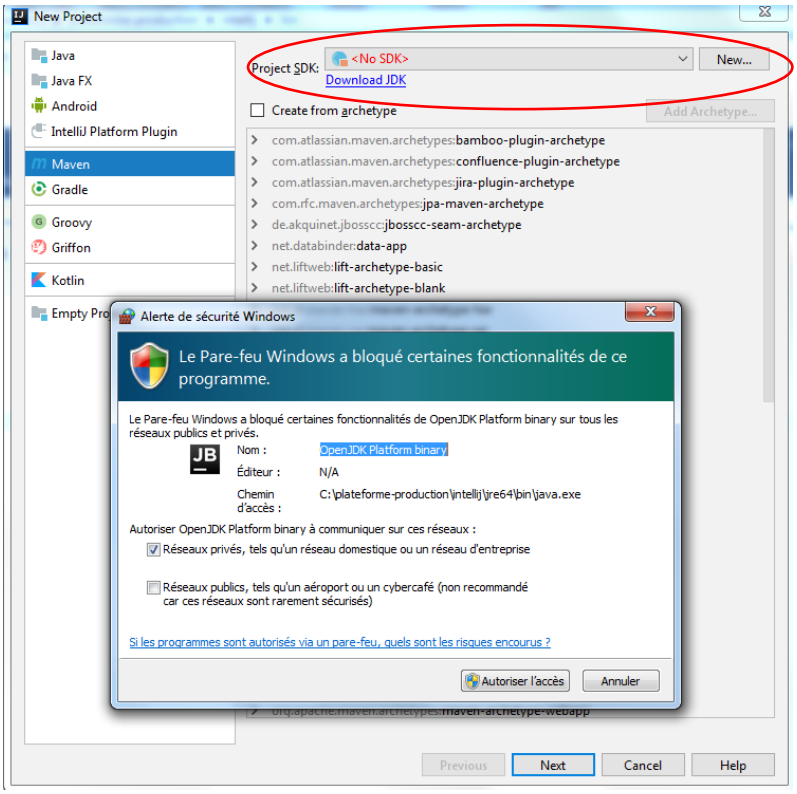
1) Cliquer sur Configure/plugin et le dialogue suivant apparaît :

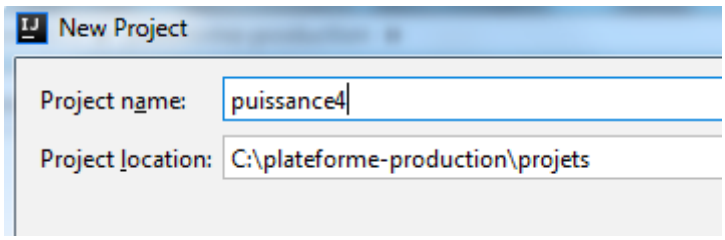


2) Installer le plugin avec le fichier CheckStyle-IDEA-5.10.0.zip disponible sur commun et relancer IntelliJ comme proposé.

Création d'un premier projet

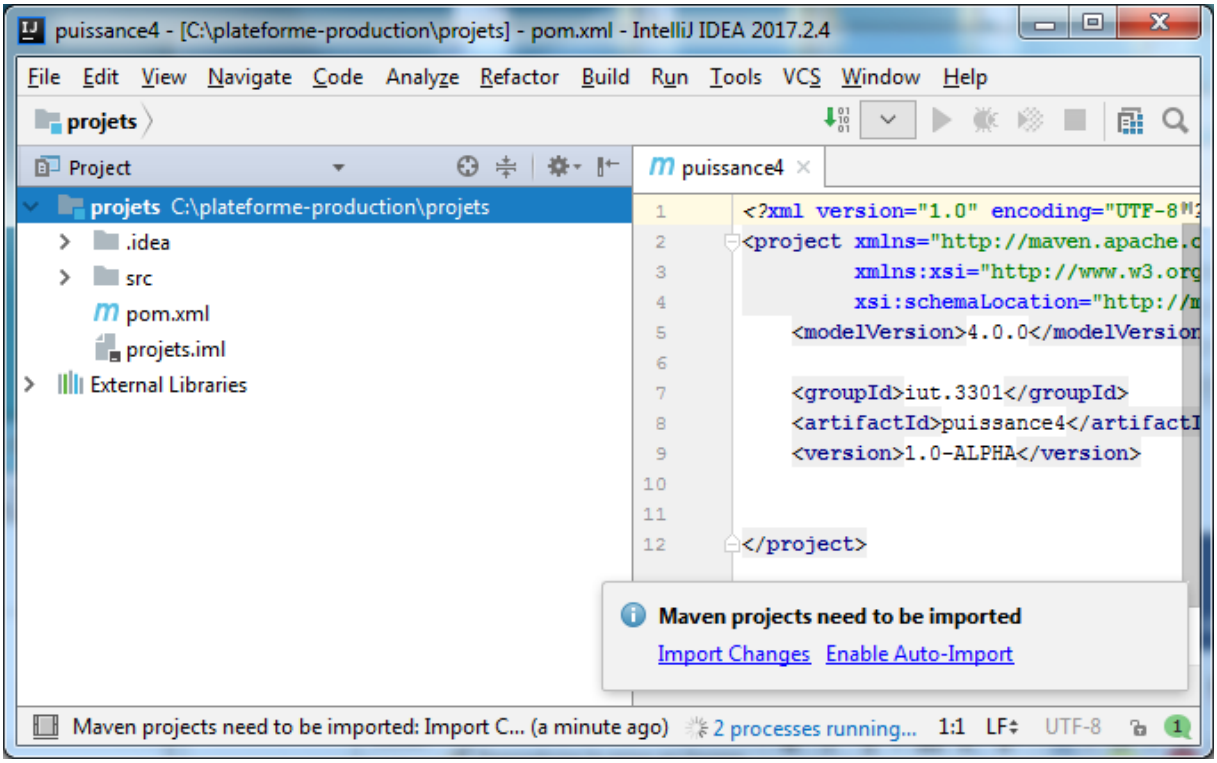
1) Créer un projet et accepter les connexions entrantes
2) Sélectionnez un projet Maven
3) Paramétrer le Project SDK vers le jdk 1.8.0_131 que nous avons installé précédemment
4) Créer le projet maven
    a. GroupId : iut.m3301
    b. ArtifactId : puissance4
    c. Version 1.0-ALPHA
5) Sauver votre projet à l'intérieur de la plateforme de production dans le répertoire **plateforme-production\projets** et validez





Le projet puissance4 en maven

1) Cliquer sur l'onglet projet et la structure du projet apparaît :



1) Explorer la structure du répertoire src :
    a. main : les sources du projet
    b. test : les sources des tests

En Maven cette structure est obligatoire car Maven cherche toujours à lancer des tests lorsqu'on lui demande de construire l'application.

Qu'est-ce que Maven ?

Vous pouvez consulter « maven in 5 minutes » sur internet.

Maven est un constructeur d'application et **un gestionnaire de dépendances** qui est capable d'automatiser toutes les tâches de construction et de déploiement d'une application sur un serveur tout en intégrant les tâches de contrôles qualités et de tests ainsi que la génération de rapports (tests , qualité , build)

Gestionnaire de dépendances ? : il suffit de déclarer la version de la bibliothèque que nous voulons et maven l'installe et télécharge toutes ses dépendances pour nous.

Installation de la dépendance JUnit 4.12 avec maven

La déclaration des dépendances se fait dans le fichier pom.xml en utilisant les dépendances

1) Ajouter les lignes suivantes dans le fichier pom.xml

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>
</dependencies>
```

2) Intellij fait de l'auto import de dépendance mais on peut le forcer par
    a. Cliquer droit sur pom.xml puis **maven** puis **reimport**

Développement d'un modèle simplifié pour jouer au puissance4

1) Mettre la compatibilité en version 1.8 : menu File/Settings/Build ../Java Compiler et ByteCode Target vezrsion
2) Créer un package **puissance4.modele** dans les 2 répertoires **src/main/java et src/test/java**
3) Spécifications du modèle
    a. Structure de 6 x 8 cases (6 lignes et 8 colonnes). Tableau de 8 piles ?
    b. La structure stocke des entiers constants CASE_VIDE=0, PION_JAUNE=1, PION_ROUGE=2
    c. Le modèle n'a pas la responsabilité de savoir si des pions sont alignés ou de gérer des joueurs, il s'agit uniquement d'une structure de stockage
    d. Le modèle doit disposer des méthodes publiques suivantes
        i. void lacherPionDansColonne(int pion , int col)
        ii. void vider()
        iii. int pionEnPosition(int lig , int col) lig=1 : ligne du bas col=1 : colonne gauche
        iv. boolean colonnePleine(int col)
        v. int nbPionsJoues()
    e. Gestion des exceptions
        i. ExceptionPionInconnu : lorqu'on tente de jouer un pion ni jaune ni rouge
        ii. ExceptionColonnePleine lorsqu'on tente de jouer sur une colonne pleine
        iii. ExceptionColonneHorsBornes lorsqu'on passe un mauvais numéro de colonne
        iv. ExceptionCaseHorsBornes lorsqu'on demande une position de case hors bornes
4) Tests unitaires à faire
    a. Tester que les pions s'empilent bien dans une colonne avec les fonctions lacherPionDansColonne et pionEnPosition
    b. Tester la fonction colonne pleine
    c. Tester la fonction nbPionsJoues
    d. Tester les 4 exceptions

5) On pourra avoir une approche TDD et écrire les tests en premier comme si le modèle était déjà implémenté

Compilation et lancement des tests

C'est très simple !

- Cliquer droit sur le projet et Run all Tests (ou par menu run)
- Cliquer droit sur recompile puis run sur le fichier de tests

Analyse de couverture du code par les tests

C'est aussi simple !

- Cliquer droit sur le projet et Run all Tests with Coverage (ou par menu run)
- Cliquer droit sur Run "FichierTest" with Coverage

Explorer la couverture des lignes de code par vos tests. Quel % de couverture ?

Analyse de votre code et paramétrage des règles checkstyle

Il suffit d'activer les règles Sun ou Google par **Setting / Checkstyle**

Ensuite l'analyse se fait par **cliquer droit / Analyze / Inspect Code** sur votre projet

Peut-on désactiver certains contrôles ? Peut-on faire son propre profil de règles checkstyle ? Chercher sur internet.