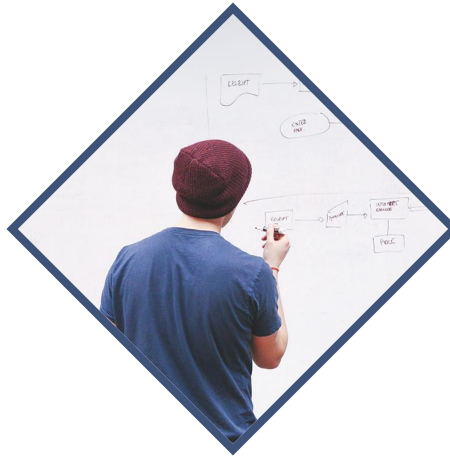


Formation Gradle

IUT Informatique



BONJOUR !

Je suis Laurent Michenaud

Je suis architecte solutions mobiles.

18 ans d'expériences.

lmichenaud@gmail.com



PLAN

1. Premiers pas avec gradle.
2. Les tâches.
3. Les dépendances et les dépôts.
4. Les sous projets.
5. Exercices complémentaires

1

INTRODUCTION

Premier pas avec Gradle.



- Open Source
- Apache Software Licence
- DSL (Domain specific language): basé sur Groovy.
- Construction d'un projet basé sur un graph de tâches que l'on peut personnaliser.
- Support des builds avec des sous projets.
- Supporte les dépôts Maven 2.

Industrialisation des projets Java, Android, Groovy, ...

Permet de gérer et d'automatiser la
production des projets de différentes natures.





A partir des sources...

- Compiler le projet.
- Exécuter les tests unitaires.
- Contrôler la qualité du code source.
- Produire les livrables (jar, apk, ..).
- Déployer les livrables sur des dépôts de packages.
- Générer la documentation du projet.
- ...

Installation 1/3

- Télécharger Gradle sur le site :
 - ▷ <https://gradle.org/install/>
- Dézipper l'archive

Installation 2/3

- Variables d'environnement
 - ▷ GRADLE_HOME -> chemin où est installé gradle
 - ▷ export GRADLE_HOME=/path/to/maven/dir
 - ▷ PATH -> ajouter le répertoire bin de gradle au PATH.
 - ▷ export PATH=\$GRADLE_HOME/bin:\$PATH
 - ▷ Spécifique IUT Informatique de Nantes
 - ▷ unset GRADLE_USER_HOME
- Tester avec la commande
 - ▷ gradle -v

Installation 3/3

- Pour configurer le proxy, éditer le fichier `~/.gradle/gradle.properties` et ajouter les lignes suivantes :
 - ▷ `systemProp.http.proxyHost=proxy-etu`
 - ▷ `systemProp.http.proxyPort=3128`
 - ▷ `systemProp.https.proxyHost=proxy-etu`
 - ▷ `systemProp.https.proxyPort=3128`
- Le fichier `~/.gradle/gradle.properties` est le fichier de configuration globale pour l'utilisateur en cours. Cette configuration sera appliquée à l'ensemble des projets de l'utilisateur.

Création d'un projet Java avec Gradle

- Pour initialiser un projet avec gradle :
 - ▷ Créer un répertoire pour les fichiers du projet.
 - ▷ Exécuter à l'intérieur la commande :
 - ▷ `gradle init`
 - ▷ Ou pour un projet Java :
 - ▷ `gradle init --type java-library`

Création d'un projet Java avec Gradle

- Les fichiers créés sont les suivants :
 - ▷ build.gradle (décrit les tâches du projet)
 - ▷ settings.gradle (configuration du build)
 - ▷ gradle/wrapper
 - ▷ gradle-wrapper.jar
 - ▷ gradle-wrapper.properties
 - ▷ gradlew
 - ▷ gradlew.bat

Gradle Wrapper

- Le système des gradle wrappers permet d'exécuter gradle sur des machines qui ne possèdent pas gradle ou qui n'ont pas la version requise.
- Les gradle wrappers sont des scripts à la racine du projet et doivent être utilisés pour invoquer gradle sur le projet :
 - ▷ gradlew pour OSX & Linux.
 - ▷ gradlew.bat pour Windows.

Gradle Wrapper

- Si gradle n'est pas installé sur la machine ou pas dans la bonne version, alors le script gradlew rapatrie la version indiquée dans le fichier *gradle/wrapper/gradle-wrapper.properties*
- Les différentes versions de gradle sont installés dans le répertoire :
 - ▷ **`$USER_HOME/.gradle/wrapper/dists`**
- Les fichiers gradlew, gradlew.bat, gradle/* doivent être commités.

Gradle Wrapper

- Spécifique IUT Informatique de Nantes
- Modifier le numéro de version indiqué dans le fichier *gradle/wrapper/gradle-wrapper.properties*
 - ▷ 4.8.1
- Lancer la commande :
 - ▷ ./gradlew tasks
- La version 4.8.1 est téléchargée et utilisée par le projet

Le fichier build.gradle

- Le fichier build.gradle est le script de construction.
- Il est écrit dans le langage Groovy.
- Il décrit le projet ainsi que les tâches qui peuvent être invoqué à partir de la commande gradlew.
- Pour connaître les tâches qui peuvent être invoquées sur le projet :
 - ▷ ./gradlew tasks

Le fichier build.gradle

- Le build.gradle créé précédemment contient en commentaire une configuration minimaliste pour un projet Java.
- Ouvrir le fichier build.gradle et décommenter cette configuration.
- Exécuter à nouveau la commande
 - ▷ ./gradlew tasks

Les tâches pour les projets Java

- De nouvelles tâches apparaissent dont :
 - ▷ assemble : compilation du projet.
 - ▷ build : compilation et exécution des tests unitaires.
 - ▷ clean: supprime le répertoire de build.
 - ▷ test: exécute les tests unitaires.
 - ▷ javadoc: création de la javadoc

Structure du projet Java

- Par défaut, la structure attendue est la même que celle d'un projet Maven :
- src
 - ▷ main
 - ▷ java (*classes Java*)
 - ▷ resources (*autres fichiers comme les fichiers .properties*)
 - ▷ test
 - ▷ java (*classes de tests*)
 - ▷ resources (*ressources de tests*)

Les tâches pour les projets Java

Exécuter la commande :

- `./gradlew build`

Les fichiers `.class` sont générés dans le sous-répertoire :

- `build/classes/`

Le jar est généré dans le répertoire :

- `build/libs/`

Analyse du fichier build.gradle

- Le plugin Java a été activé :
 - ▷ `apply plugin: 'java'`
- Le repository Maven a été déclaré :
 - ▷ `repositories {`
 - ▷ `jcenter()`
 - ▷ `}`

Analyse du fichier build.gradle

Les dépendances du projets ont été déclarées :

- dependencies {
- compile 'org.slf4j:slf4j-api:1.7.25'
- testCompile 'junit:junit:4.12'
- }

Import des projets dans les IDE

Eclipse

- ▶ Plugin Gradle intégré par défaut dans les versions Eclipse Java Developer et JEE Developer
- ▶ Via la fonction : “Import...” puis “gradle”

■ IntelliJ

- ▶ Intégré de base dans la version community.
- ▶ Spécifique IUT Info de Nantes :
 - ▶ File -> Settings -> chercher “gradle” et modifier la variable “Service directory path”. Elle doit pointer vers le répertoire /home/<votrelogin>/.gradle
 - ▶ File -> Settings -> chercher “proxy” et configurer le proxy

Exercice 1

- Notre projet a besoin d'un groupId, artifactId, version pour être compatible avec les dépôts Maven.
 - ▷ L'artifactId est déjà défini dans le fichier settings.properties : correspond au rootProject.name
- Spécifier le groupId du projet dans le build.gradle
 - ▷ group='org.iut.nantes'
- Spécifier la version du projet dans le fichier build.gradle
 - ▷ version='0.1.0'

Exercice 2

- Spécifier la version de Java avec laquelle les sources sont compatibles dans le fichier build.gradle :
 - ▷ `sourceCompatibility='1.8'`
- Spécifier l'encoding des sources du projet :
 - ▷ `compileJava.options.encoding='UTF-8'`
- Importer le projet dans Eclipse.
 - ▷ Effectuer les opérations de compilation.

Exercice 3

- Créer une classe Calculette dans le répertoire `src/main/java/org/iut/nantes` avec une méthode réalisant une addition de 2 nombres passés en paramètre.
- Créer une classe App.java qui :
 - ▷ Instancier votre classe calculette.
 - ▷ Réalise une addition quelconque en utilisant l'instance de calculette.

Exercice 4 : Ecriture d'un test unitaire

- Créer la classe de test `CalculetteTest.java` dans le répertoire `src/test/java/org/iut/nantes`
- Ajouter une méthode `testAddition()` pour tester la méthode `addition()` de la classe `Calculette`.
- Exécuter les tests :
 - ▷ En ligne de commande :
 - ▷ `./gradlew -q test`

Exercice 5 : Génération de la javadoc

- Générer la javadoc du projet avec la commande :
 - ▷ `./gradlew javadoc`
- Ouvrir le site de la javadoc générée dans le répertoire
 - ▷ `build/docs/javadoc`

2

Les tâches

Caractéristiques d'une tâche

- Sont décrites dans un fichier build.gradle
- Une tâche a un nom.
- Une tâche représente un travail à effectuer.
- Une tâche est affectée à un projet.
- Une tâche est composée d'une séquence d'actions (closure).

Les closures d'une tâche

- Une tâche possède une liste d'actions qui sont exécutées lors de la phase d'exécution.
- Il est possible d'indiquer si l'action/closure s'exécute :
 - avant les autres (doFirst)
 - après les autres (doLast)

Les closures d'une tâche

- Ajout d'une action doFirst dans la déclaration de la tâche
 - task hello {
 - doFirst {
 - println 'Hello Venus'
 - }
 - }

Les closures d'une tâche

- Ajout d'une action doFirst à une tâche existante :
 - `hello.doFirst {`
 - `println 'Hello Venus'`
 - `}`

Les closures d'une tâche

- Ajout d'une action doLast dans la déclaration de la tâche
 - task hello {
 - doLast {
 - println 'Hello Venus'
 - }
 - }

Les closures d'une tâche

- Ajout d'une action doLast à une tâche existante :
 - hello.doLast {
 - println 'Hello Venus'
 - }

Exécution d'une tâche

- build.gradle :
 - ▷ task hello {
 - doLast {
 - ▷ println 'tutorialspoint'
 - }
 - ▷ }
- Exécution :
 - ▷ ./gradlew hello

Dépendances entre tâches

- Une tâche peut dépendre d'une autre tâche :

```
task taskX {  
    println 'taskX'  
}  
task taskY(dependsOn: 'taskX') {  
    doFirst {  
        println "taskY"  
    }  
}
```

Dépendances entre tâches

- Il est possible d'exclure une tâche au moment de l'exécution
 - ▷ `./gradlew -q tasky -x taskx`
- Par défaut, gradle arrête le build si une erreur se produit. Il est possible de le forcer à continuer avec l'option `continuous` :
 - ▷ `./gradlew -q tasky --continuous`

Description d'une tâche

- La commande *gradle tasks* permet d'afficher les tâches disponibles et leurs descriptions.
- Pour définir la description d'une tâche :

```
task hello {  
    description 'Print Hello World'  
    println("Hello World")  
}
```

Cycle de vie de gradle

1. **Initialisation** : gradle détermine quels projets font partis du build et crée une instance de Project pour chacune (settings.gradle)
2. **Configuration** : les projets sont configurés.
3. **Exécution** : gradle détermine l'ensemble des tâches à exécuter par rapport aux paramètres indiqués avec la commande gradle, puis les exécute.

Cycle de vie de gradle

settings.gradle

```
println 'This is executed during the initialization phase.'
```

build.gradle

```
println 'This is executed during the configuration phase.'
```

```
task myTask {  
    println 'This is also executed during the configuration phase.'  
}
```

```
myTask.doLast {  
    println 'This is executed during the execution phase.'  
}
```

Exercice 1

- Créer 3 tâches A, B, C
- Ajouter une dépendance pour que A s'exécute avant B.
- Ajouter une dépendance pour que B s'exécute avant C.
- Exécuter C.
- Exécuter C sans B.

Exercice 2

- Ajouter une closure à la tâche A qui affiche “Ca va ?”
- Ajouter une deuxième closure à la tâche A qui affiche “Bonjour”. Bonjour doit s’afficher avant “Ca va ?”
- Ajouter une troisième closure (en position de fin) à la tâche A qui affiche “au revoir”
- Afficher un message “Configuration de B” lors de la phase de configuration de la tâche B.
- Afficher un message “Tous les tests sont OK” à la fin de la tâche existante de tests.

3

Les dépendances et les repositories

Déclaration des dépendances

- Gradle permet de définir les dépendances d'un projet.
 - ▷ apply plugin: 'java'
 - ▷ repositories {
 - ▷ mavenCentral()
 - ▷ }
 - ▷ dependencies {
 - ▷ **compile** group: 'org.hibernate', name: 'hibernate-core', version: '3.6.7.Final'
 - ▷ **testCompile** group: 'junit', name: 'junit', version: '4.+'
 - ▷ }

Déclaration des dépendances

- Pour utiliser une dépendance d'un dépôt :
 - ▷ `compile group: 'org.hibernate', name: 'hibernate-core', version: '3.6.7.Final'`
 - ▷ `compile 'org.hibernate:hibernate-core:3.6.7.Final'`
- Pour déclarer une dépendance avec un sous projet
 - ▷ `compile project(':shared')`
- Pour utiliser un jar stocké dans le projet
 - ▷ `compile files('libs/a.jar', 'libs/b.jar')`

Version des dépendances

- Il est préconisé de toujours utiliser des versions fixes pour les dépendances mais :
 - ▷ compile "junit:junit:4.12"
- Pour utiliser toujours la dernière version de JUnit :
 - ▷ compile "junit:junit:+"
- Pour utiliser la dernière version 4.x :
 - ▷ compile "junit:junit:4.+"

Gestion de la transitivité

- Les dépendances sont transitives. Pour désactiver la transitivité sur une dépendance :
 - ▷ **compile** ('org.hibernate:hibernate-core:3.6.7.Final') {
 - ▷ transitive = false
 - ▷ }
- Pour exclure une dépendance transitive :
 - ▷ **compile**('org.hibernate:hibernate-core:3.6.7.Final') {
 - ▷ exclude group: 'com.android.support', module: 'support-annotations'
 - ▷ })

Gestion des conflits

- Si deux dépendances n'utilisent pas la même version d'une autre librairie, vous pouvez forcer la version à utiliser :
 - ▷ `configurations.all {`
 - ▷ `resolutionStrategy {`
 - ▷ `force "com.google.code.findbugs:jsr305:3.0.1"`
 - ▷ `}`
 - ▷ `}`

Déclaration des dépendances

- Pour voir les dépendances d'un projet :
 - ▷ `./gradlew -q dependencies`

Déclaration des repositories

- Des fonctions permettent rapidement d'ajouter les dépôts de package couramment utilisés :
 - ▷ `repositories {`
 - ▷ `mavenCentral()`
 - ▷ `jcenter()`
 - ▷ `google()`
 - ▷ `}`

Déclaration des repositories

- Une configuration de dépôts plus détaillée :

```
▷ repositories {  
    maven {  
  
        credentials {  
  
            username 'admin'  
  
            password 'admin123'  
        }  
  
        url "http://repo.mycompany.com/maven2"  
    }  
}
```

Publication d'une dépendance sur un dépôt

- **apply plugin: 'maven-publish'**
- publishing {
 - ▷ publications {
 - ▷ mavenJava(MavenPublication) {
 - ▷ from components.java
 - ▷ }
 - ▷ }
- repositories {
 - ▷ maven { url "\$buildDir/repo" }
 - ▷ }
- }

Exercice 1

- Importer la librairie :
 - ▷ groupId: org.springframework.ws
 - ▷ artifactId: spring-ws-core
 - ▷ version: 2.4.0.RELEASE
- En ligne de commande, exécuter :
 - ▷ ./gradlew dependencies
- Comprendre pourquoi la librairie aopalliance est incluse dans le projet.
- Exclure cette dépendance transitive.

Exercice 2

- Configurer le projet Gradle pour qu'il déploie ses versions de développement dans un dépôt local.

4

Projet avec sous-projets

Projets avec sous projets

- Il est possible de découper un projet en plusieurs sous projets avec des dépendances entre sous projets.
- Intérêts :
 - ▷ Facilite la réutilisabilité du code.
 - ▷ Améliore la maintenabilité du code.
 - ▷ Permet d'éviter des dépendances illégales entre classes.

Structure d'un projet avec sous-projets

- Le répertoire racine contient :
 - ▷ Le fichier settings.gradle
 - ▷ Le fichier build.gradle principal
 - ▷ Un répertoire par sous projet avec
 - ▷ Le fichier build.gradle du sous-projet (optionnel)

Le fichier settings.gradle

- Les sous projets sont déclarés dans le fichier settings.gradle :
 - ▷ include 'calcul', 'app'

Projets avec sous projets

- Pour lister les sous projets disponibles :
 - ▷ `./gradlew -q projects`
- Pour voir les tâches uniquement d'un projet :
 - ▷ `./gradlew -q :project1:tasks`

Projets avec sous projets

- La configuration générale peut être définie dans le build.gradle racine
 - ▷

```
allprojects {  
    group = 'com.example.gradle'  
    version = '0.1.0'
```
 - ▷

```
}
```

Projets avec sous projets

- A partir du build.gradle racine, il est aussi possible d'appliquer une configuration aux sous projets mais pas au projet racine :
 - ▷ subprojects {
 apply plugin: 'java'
 - ▷ }
- Autre solution: définir la configuration dans le build.gradle du sous projet.

Exercice 1

- Découper votre projet Gradle en sous-projets avec :
 - ▷ 1 sous projet calcul contenant la classe Calculette.
 - ▷ 1 sous projet app contenant la classe App.

Exercices complémentaires

Exercice 1 : qualité du code source

- Contrôler la qualité du code source en ajoutant le plugin checkstyle :
 - ▷ apply plugin: 'checkstyle'
- Trouver la tâche à exécuter pour lancer le contrôle du code source.
- Trouver le rapport html qui a été généré.
- Réaliser quelques corrections des anomalies remontées.

Exercice 2 : qualité du code source

- Idem avec le plugin Findbugs

Exercice 3

- Commiter le projet sous GitHub
- Mettre en place le plugin release sur le projet.

Exercice 4

- Mettre en place la configuration nécessaire pour créer un zip avec les sources du projet.