Construire un projet java avec Gradle...

...quand on est un pauvre développeur C++ avec CMake

D. Arrivault

Laboratoire d'Excellence Archimède
Aix Marseille Université

09 Juin 2016 / LIFTech' 2d session

Outline

Les outils de construction de projets.

- Pour le développeur :
  - Gestion de tout le cycle "édition, compilation/interprétation, tests, installation" avec un même outil.
  - Recompilation/réinterprétation uniquement des sources modifiées.
- Pour l'équipe de développement :
  - Génération simple de paquets
  - Exécution simple des tests
  - Génération simple de la documentation
- Pour l'utilisateur :
  - Installation du logicielle simple.
  - Gestion des dépendances.
  - Adaptation de l'installation à l'environnement système.

Mon Projet.

```
git clone
git@gitlab.lif.univ-mrs.fr:denis.arrivault/ImmutableComplex.git
```

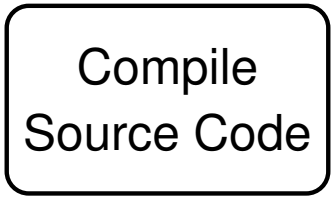
```
ImmutableComplex
├── lib
│   ├── hamcrest-all-1.3.jar
│   └── junit-4.11.jar
└── src
    ├── main
    │   └── java
    │       ├── Complex.java
    │       └── Main.java
    └── test
        └── java
            └── ComplexTest.java
```

Une classe `Complex`, une classe `Main` avec seulement une fonction main et une classe de test unitaire utilisant les décorateurs JUnit.

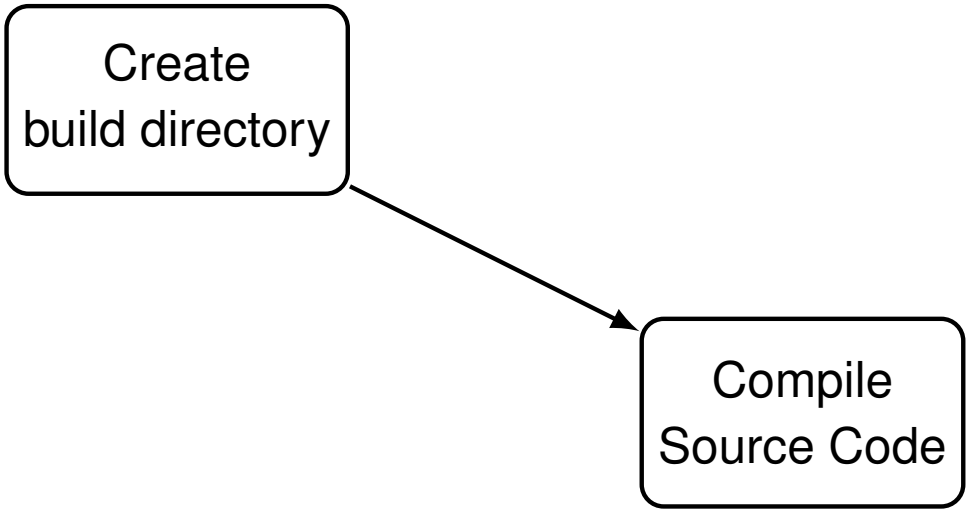
Mon Projet.

```
package main.java;

public class Main {

        public static void main(String[] args) {
                Complex c1 = new Complex(2,4);
                Complex c2 = new Complex(1,2);
                System.out.println(c1.divide(c2));
                System.out.println(c1.equals(c2));
                System.out.println(c1.equals(c2.add(c2)));
        }

}
```

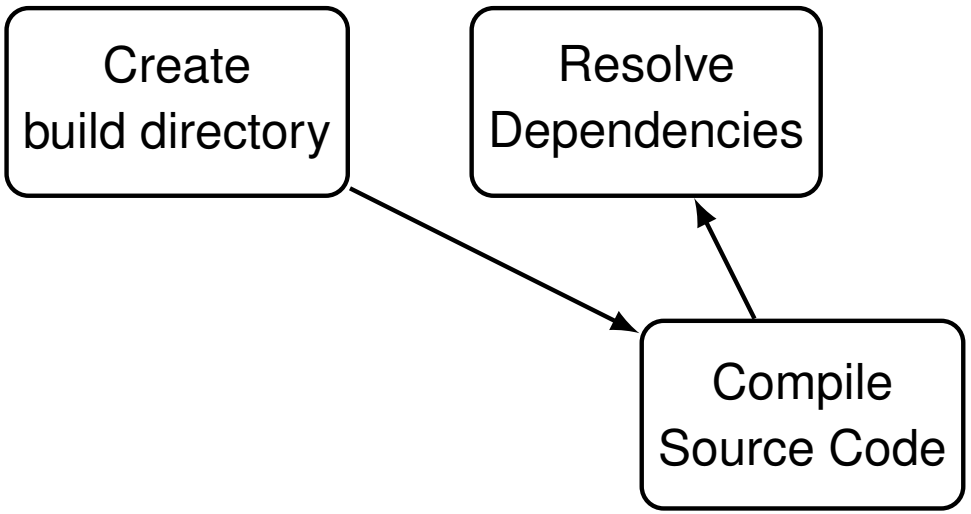
Les étapes de la construction.



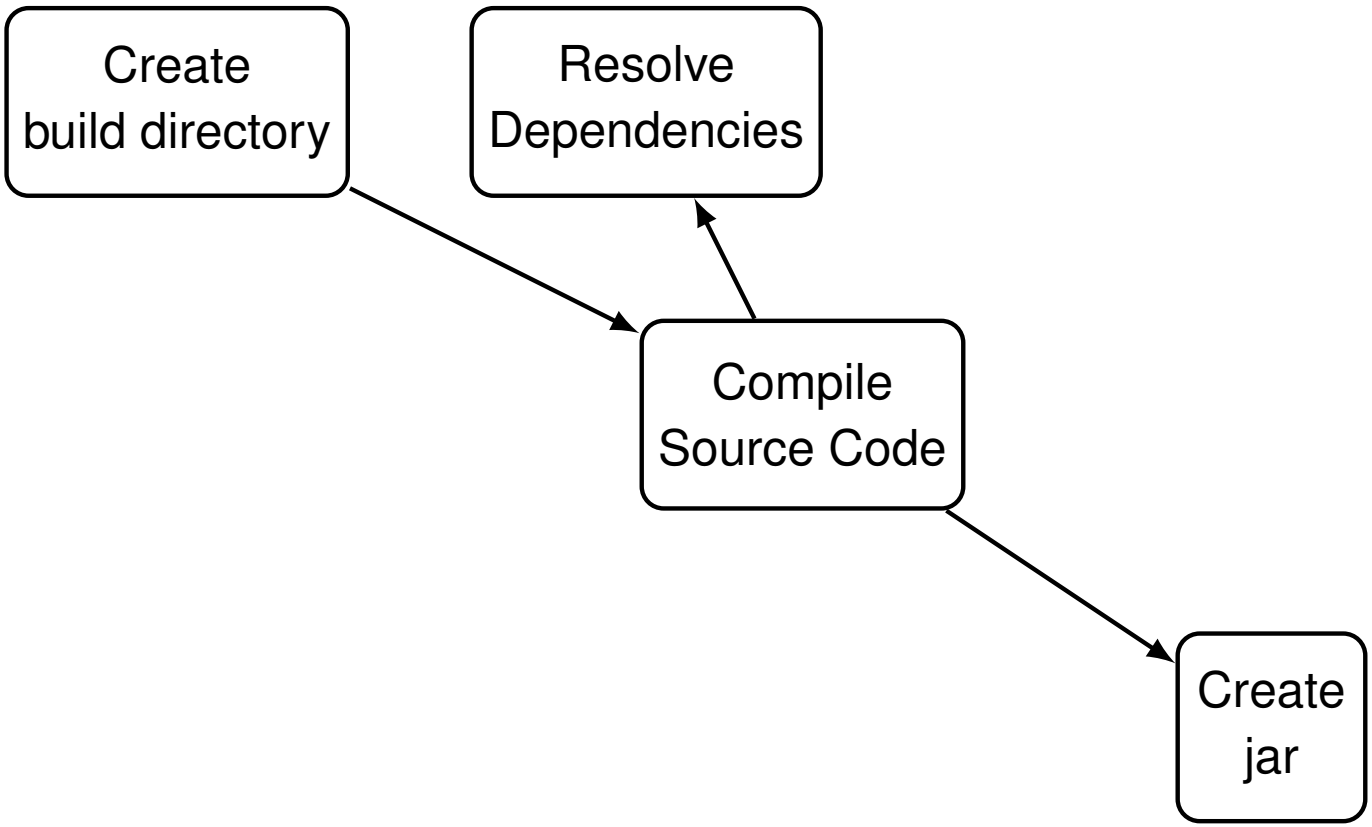
Les étapes de la construction.



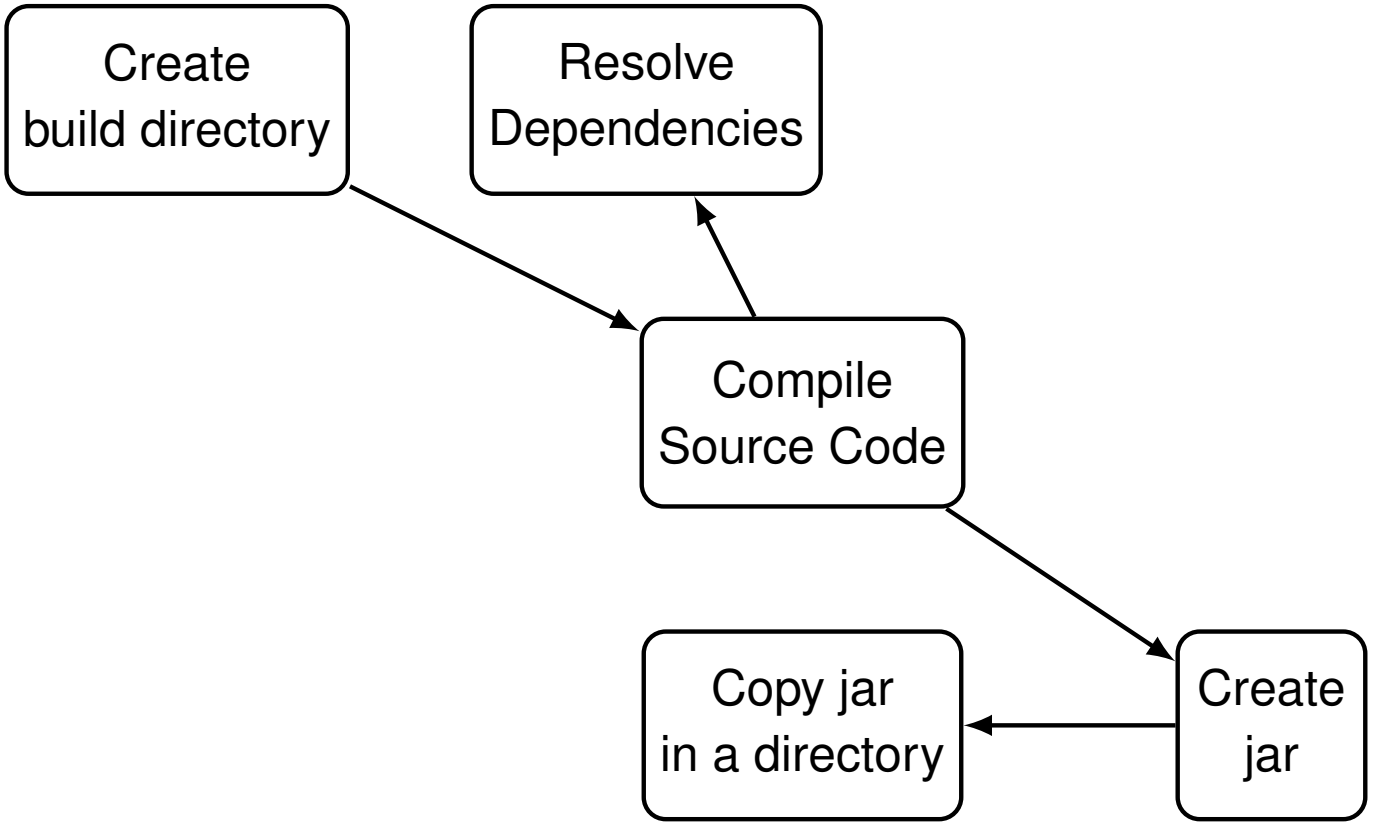
Les étapes de la construction.



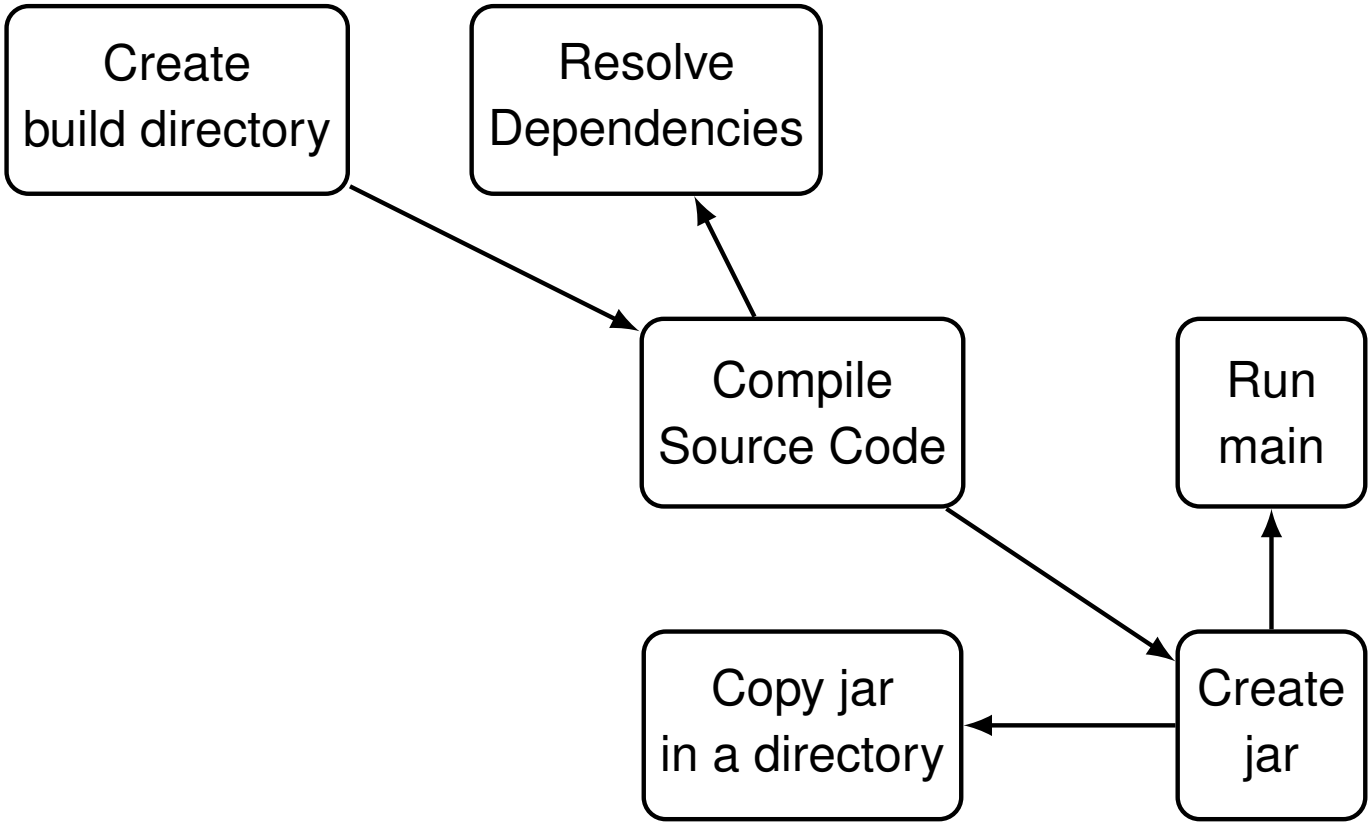
Les étapes de la construction.



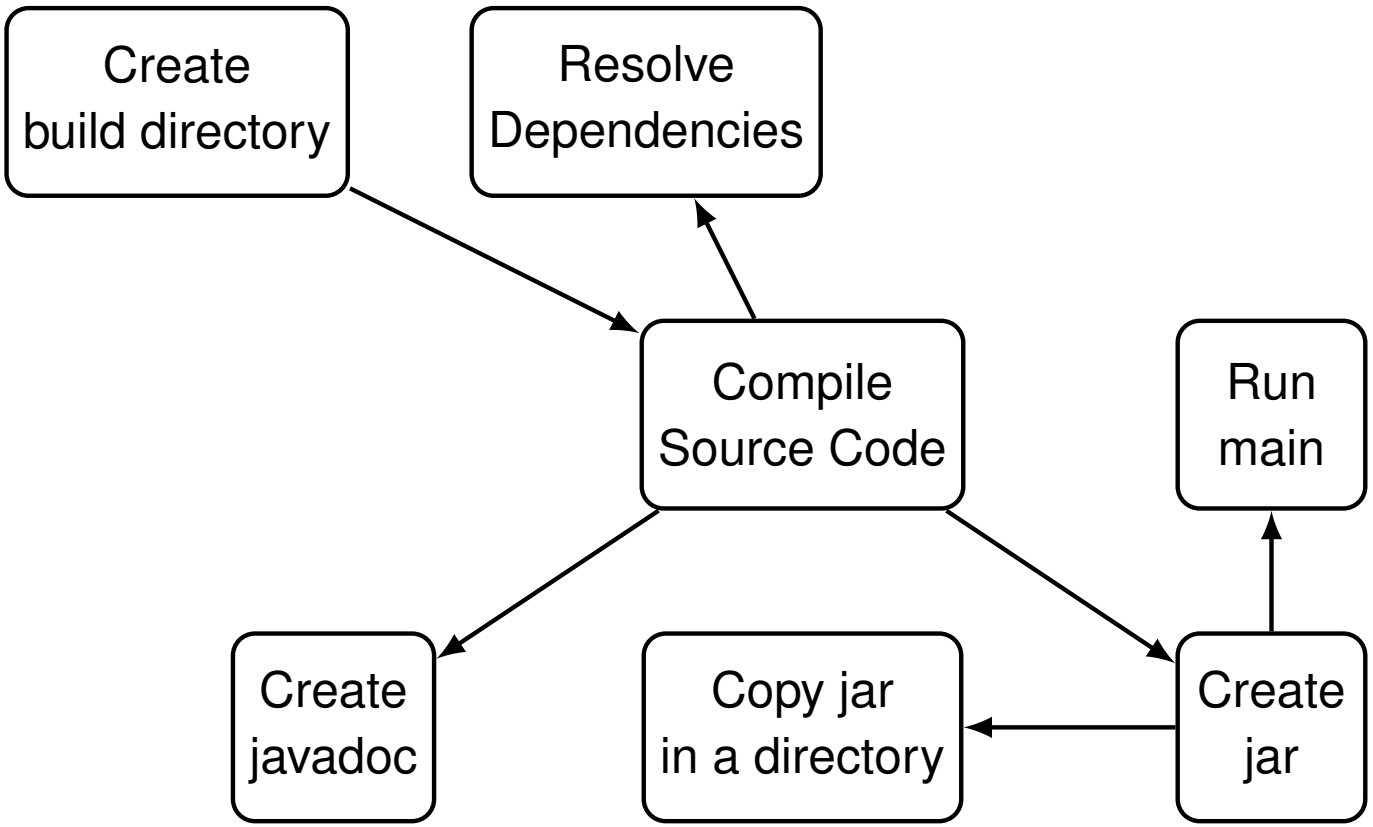
Les étapes de la construction.



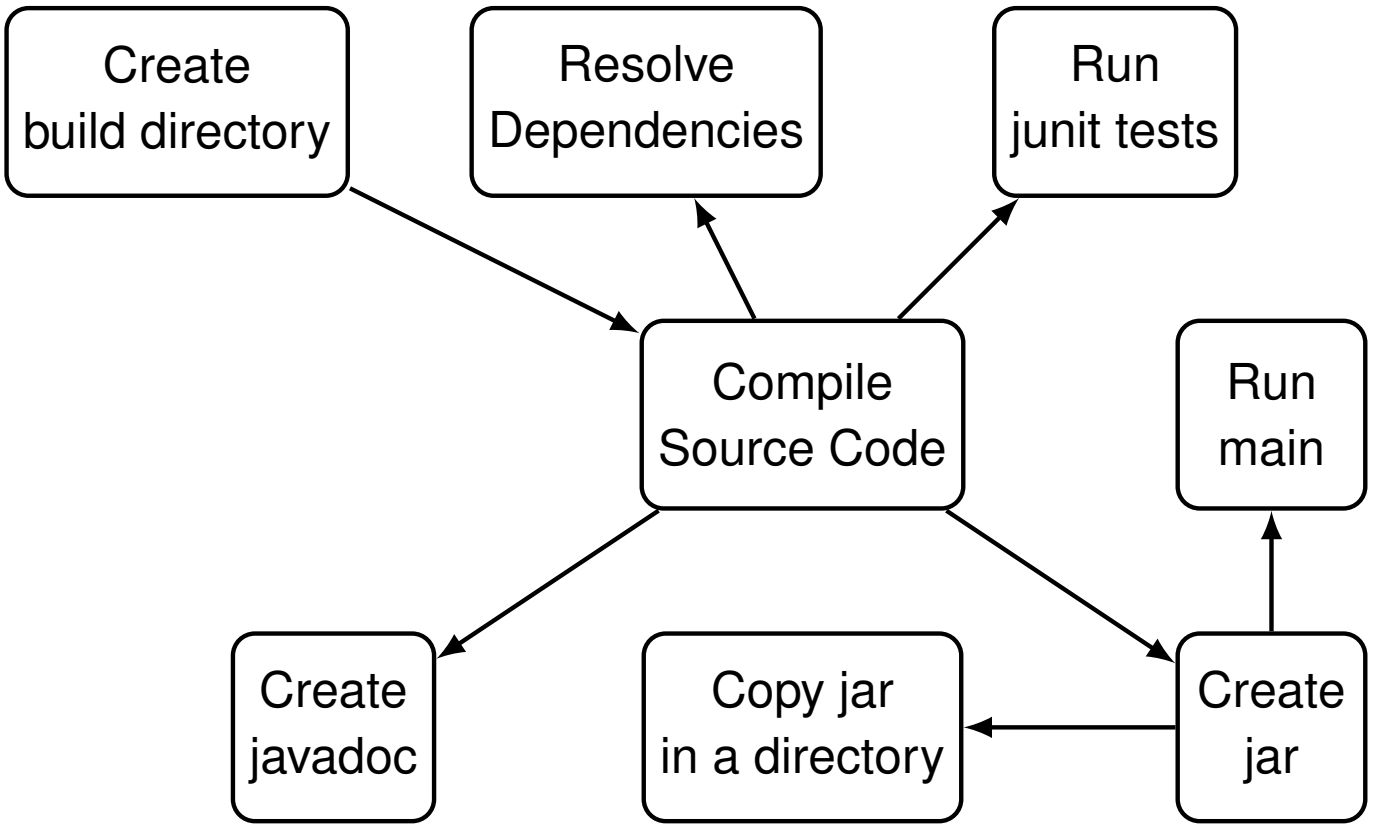
Les étapes de la construction.



Les étapes de la construction.



Les étapes de la construction.



Construction à la main.

Compiler/exécuter les sources

```
$ mkdir bin
$ javac -d bin src/main/java/*.java
$ java -cp bin main.java.Main
(2.0 + 0.0i)
false
true
```

Compiler/exécuter le test

```
$ javac -cp lib/junit-4.11.jar:bin -d bin/ src/test/java/ComplexTest.java
$ java -cp lib/junit-4.11.jar:lib/hamcrest-all-1.3.jar:bin org.junit.
    runner.JUnitCore test.java.ComplexTest
JUnit version 4.11
.........
Time: 0,007

OK (9 tests)
```

Construction à la main.

Générer un jar exécutable

```
$ cd bin
$ echo Main-Class: main.java.Main >manifest.txt
$ jar cvfm ImmutableComplex.jar manifest.txt main/java/*.class
manifeste ajouté
ajout : main/java/Complex.class(entrée = 1628) (sortie = 865)(compression
      : 46 %)
ajout : main/java/Main.class(entrée = 674) (sortie = 418)(compression :
     37 %)
$ java -jar ImmutableComplex.jar
(2.0 + 0.0i)
false
true
```

Générer la javadoc

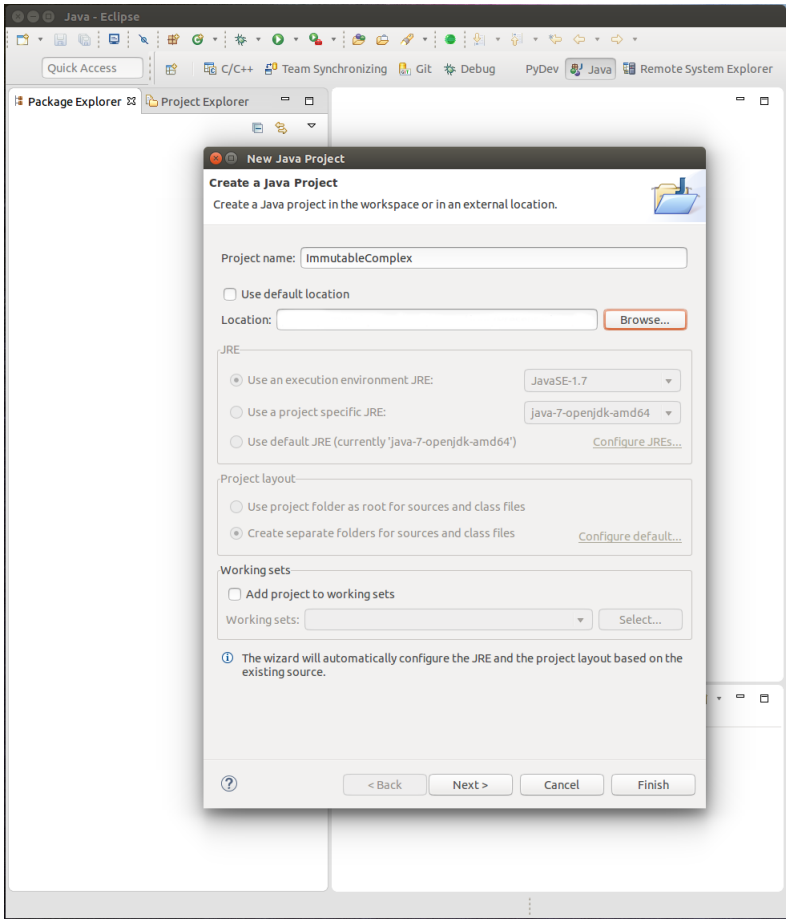
```
$ cd ../src
$ javadoc -d ../doc main.java
```

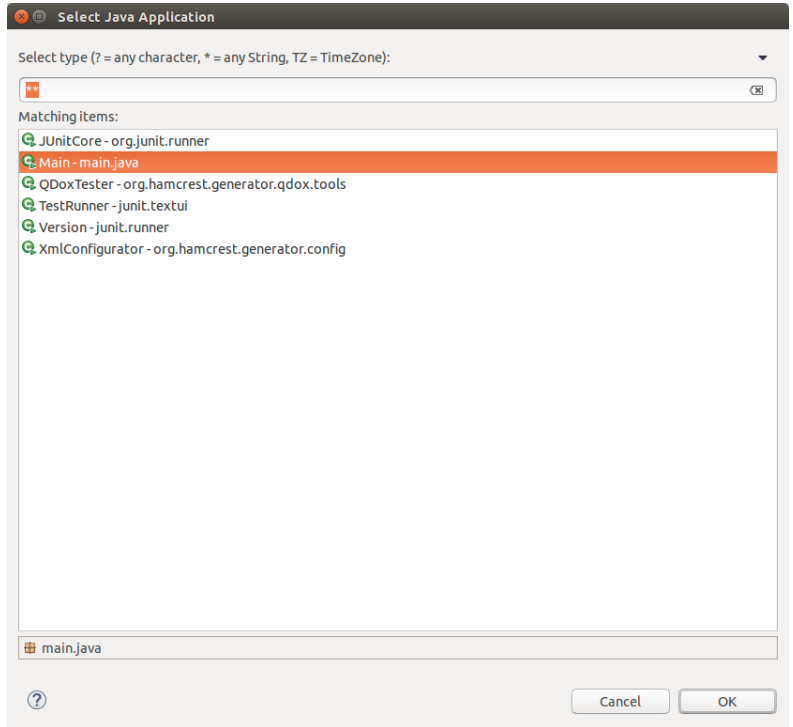
Construction à la main.

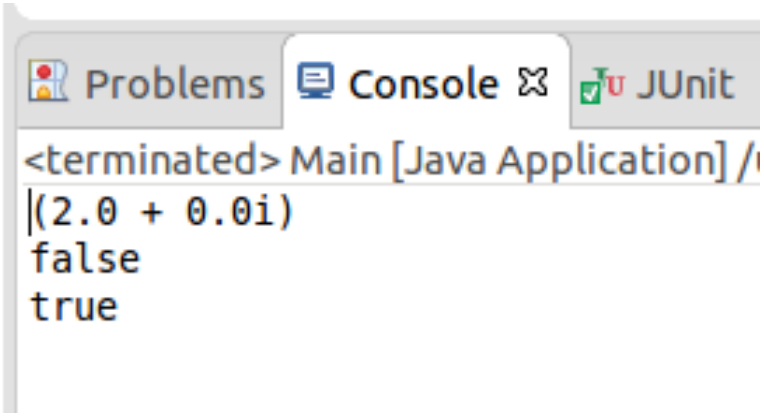
- Lancer ces commandes plusieurs fois par jour est laborieux : scripting.
- La gestion des dépendances complexes devient vite un cauchemar.
- La portabilité est problématique.

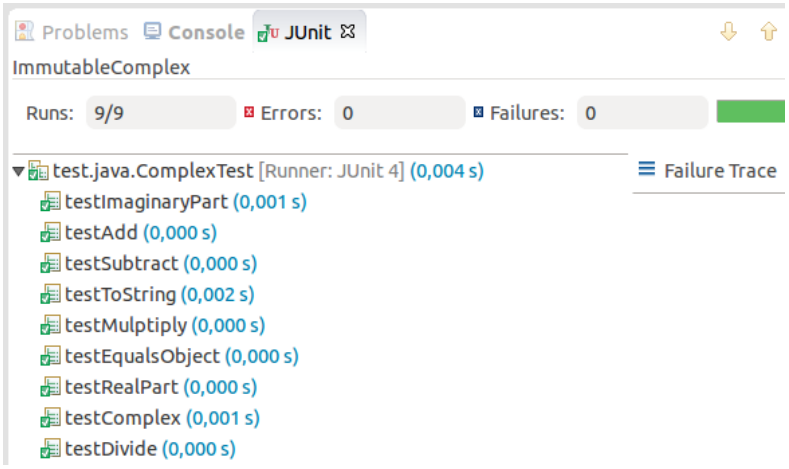
Construction avec Eclipse.

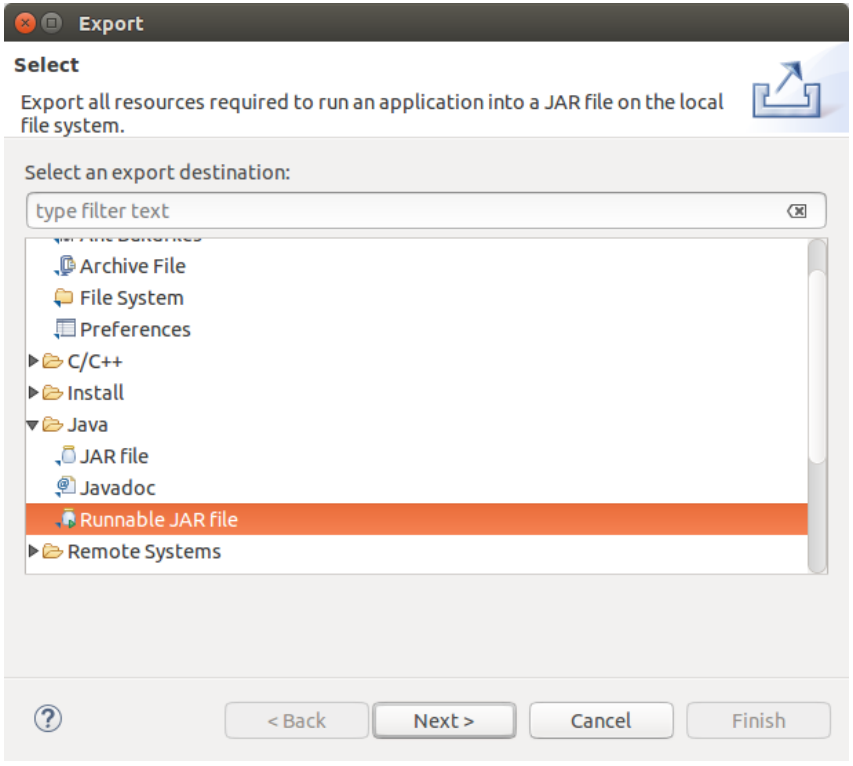
- *File->New->Java Project*
- *Browse...* : sélectionner le répertoire du projet.
- *Finish*
- Les deux bibliothèques sont reconnues automatiquement et ajoutées au *build path*.
- *Project->Build Project*
- *Run->Run As->Java Application* ; sélectionner java.Main.
- *Run->Run As->JUnit Test*
- *File->Export->java->Runnable JAR file*


Java - Eclipse
Quick Access
C/C++
Team Synchronizing
Git
Debug
PyDev
Java
Remote System Explorer
Package Explorer
Project Explorer
New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: ImmutableComplex
Use default location
Location:
Browse...
JRE
Use an execution environment JRE:
JavaSE-1.7
Use a project specific JRE:
java-7-openjdk-amd64
Use default JRE (currently 'java-7-openjdk-amd64')
Configure JREs...
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
Working sets:
Select...
The wizard will automatically configure the JRE and the project layout based on the existing source.
< Back
Next >
Cancel
Finish


Select Java Application
Select type (? = any character, * = any String, TZ = TimeZone):
Matching items:
JUnitCore - org.junit.runner
Main - main.java
QDoxTester - org.hamcrest.generator.qdox.tools
TestRunner - junit.textui
Version - junit.runner
XmlConfigurator - org.hamcrest.generator.config
main.java
Cancel
OK


Problems
Console
JUnit
<terminated> Main [Java Application] /
(2.0 + 0.0i)
false
true


Problems
Console
JUnit
ImmutableComplex
Runs: 9/9
Errors: 0
Failures: 0
Failure Trace
test.java.ComplexTest [Runner: JUnit 4] (0,004 s)
testImaginaryPart (0,001 s)
testAdd (0,000 s)
testSubtract (0,000 s)
testToString (0,002 s)
testMulptiply (0,000 s)
testEqualsObject (0,000 s)
testRealPart (0,000 s)
testComplex (0,001 s)
testDivide (0,000 s)


Export
Select
Export all resources required to run an application into a JAR file on the local file system.
Select an export destination:
type filter text
Archive File
File System
Preferences
C/C++
Install
Java
JAR file
Javadoc
Runnable JAR file
Remote Systems
< Back
Next >
Cancel
Finish

Construction avec Eclipse.

- Tous les développeurs actuels et futurs du projet doivent utiliser Eclipse.
- Attention aux surprises avec les changements de version.
- Problème du déploiement sur les plate-formes d'intégration continue.

Construction avec CMake.

```
ImmutableComplex
├── lib
│   ├── hamcrest-all-1.3.jar
│   └── junit-4.11.jar
├── src
│   ├── main
│   │   ├── java
│   │   │   ├── Complex.java
│   │   │   └── Main.java
│   │   └── CMakeLists.txt
│   └── test
│       ├── java
│       │   └── ComplexTest.java
│       └── CMakeLists.txt
└── CMakeLists.txt
```

Construction avec CMake.

Configurer

```
$ mkdir build && cd build
$ cmake .. -DJAR_INSTALL_DIR=../bin -L
-- Found Java: /usr/bin/java (found version "1.7.0.101")
-- Configuring done
-- Generating done
-- Build files have been written to: [...]/ImmutableComplex/build
-- Cache values
CMAKE_INSTALL_PREFIX:PATH=/usr/local
CMAKE_Java_ARCHIVE:FILEPATH=/usr/bin/jar
CMAKE_Java_RUNTIME:FILEPATH=/usr/bin/java
JAR_INSTALL_DIR:PATH=[...]/ImmutableComplex/bin
```

Construction avec CMake.

Compiler/exécuter les tests

```
$ make
$ make test
Running tests...
Test project [...]/ImmutableComplex/build
    Start 1: test_complex
1/1 Test 1: test_complex .....................   Passed    0.13 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   0.13 sec
```

Construction avec CMake.

Installer et lancer le jar exécutable

```
$ make install
$ cd ../bin
$ java -jar ImmutableComplex.jar
(2.0 + 0.0i)
false
true
```

Générer la javadoc

La doc est générée dans `bin/ImmutableComplex_doc` lors de l'installation.

Construction avec CMake.

- CMake n'a pas été conçu pour construire du code java.
- Il faudrait tester le passage à l'échelle sur des applications client/serveur.

Gradle

Réclame

- A very flexible general purpose build tool like Ant.
- Switchable, build-by-convention frameworks a la Maven. But we never lock you in !
- Very powerful support for multi-project builds.
- Very powerful dependency management (based on Apache Ivy).
- Full support for your existing Maven or Ivy repository infrastructure.
- Support for transitive dependency management without the need for remote repositories or pom.xml and ivy.xml files.
- Ant tasks and builds as first class citizens.
- Groovy build scripts.
- A rich domain model for describing your build.

Gradle

Réclame

- A very flexible general purpose build tool like Ant.
- Switchable, build-by-convention frameworks a la Maven. But we never lock you in !
- Very powerful support for multi-project builds.
- Very powerful dependency management (based on Apache Ivy).
- Full support for your existing Maven or Ivy repository infrastructure.
- Support for transitive dependency management without the need for remote repositories or pom.xml and ivy.xml files.
- Ant tasks and builds as first class citizens.
- Groovy build scripts.
- A rich domain model for describing your build.

CMake ? ? ?

Gradle

Concrètement

- Développé depuis 2008 par Gradle Inc. / Open Source.
- Création d'un fichier `build.gradle` à la racine du projet (un fichier par projet).
- Description des différentes tâches de construction à l'aide d'un DSL [1] basé sur *Groovy*.
- Possibilité d'utiliser et/ou de redéfinir des tâches prédéfinies dans des plugins :
  - Support pour les langages gérées par la JVM : *java, groovy, scala, antlr*.
  - Support pour d'autres langages : *assembler, c, cpp, objective-c/cpp*...
  - Support pour l'exécution, l'empaquetage et la distribution : *assemble, distribution*...
  - Support pour le développement : *eclipse, sonar, visual-studio, wrapper*...
  - ...

1. Domain Specific Language

Écrire des tâches

Dans le fichier `build.gradle`

```
task hello {
    doLast {
        println 'Hello world!'
    }
}
```

Appel

```
$ gradle hello
:hello
Hello world!

BUILD SUCCESSFUL

Total time: 0.648 secs
$ gradle -q hello
Hello world!
```


Écrire des tâches

- Un script gradle définit, pour un ou plusieurs projets, un ensemble de tâches.
- Les tâches peuvent être vues comme une pile d'actions.
- Une action s'écrit à l'aide de *closures* groovy :

```
{ [closureParameters -> ] statements }
```

- L'ajout d'actions à une tâche se fait à l'aide des méthodes `doFirst` et `doLast`.

Écrire des tâches

- La syntaxe est flexible et simplifiable (sic).

```
task hello1 {
    doLast {
        println 'hello1'
    }
}

hello1.doFirst{println 'before hello1'}

task(hello2) << {println "hello2"}
hello2 << {println 'more hello2'}
// Attention
// {println 'before hello2'} >> hello2
// ne marche pas...

task('hello3') << {println "hello3"}
tasks.create(name: 'hello4') << {println "hello4"}
```

Écrire des tâches

- Les tâches peuvent être organisées entre elles à l'aide de liens de dépendance.

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

```
$ gradle -q taskX
taskY
taskX
```

Écrire des tâches

- Les tâches peuvent être typées.

```
task copyDocs(type: Copy) {
    from 'src/main/doc'
    into 'build/target/doc'
    include '**/*.txt', '**/*.xml'
    exclude '**/*.properties
}
```

```
task makePretty(type: Delete) {
  delete 'uglyFolder', 'uglyFile'
  followSymlinks = true
}
```

...

Création du script de notre projet

- Initialisation à la racine du projet avec la commande `gradle init`
- Deux fichiers pré remplis sont créés : *build.graddle* et *settings.graddle*.
- D'autres fichiers sont ajoutés pour "wrapper" *gradle* et pouvoir déployer sur des systèmes n'ayant pas *gradle* installé.
- Pour avoir la liste des tâches disponibles à ce stade : `gradle :tasks`

```
------------------------------------------------------------
All tasks runnable from root project
------------------------------------------------------------

Build Setup tasks
-----------------
init - Initializes a new Gradle build. [incubating]
wrapper - Generates Gradle wrapper files. [incubating]

Help tasks
----------
components - Displays the components produced by root project 'ImmutableComplex'. [incubating]
dependencies - Displays all dependencies declared in root project 'ImmutableComplex'.
dependencyInsight - Displays the insight into a specific dependency in root project 'ImmutableComplex'.
help - Displays a help message.
projects - Displays the sub-projects of root project 'ImmutableComplex'.
properties - Displays the properties of root project 'ImmutableComplex'.
tasks - Displays the tasks runnable from root project 'ImmutableComplex'.
```

Ajout du plugin 'java'

Dans *build.graddle*

```
/*
 * This build file was auto generated by running the Gradle 'init' task
 * by 'arrivault' at '02/06/16 18:33' with Gradle 2.3
 *
 * This generated file contains a commented-out sample Java project to
     get you started.
 * For more details take a look at the Java Quickstart chapter in the
     Gradle
 * user guide available at http://gradle.org/docs/2.3/userguide/
     tutorial_java_projects.html
 */


// Apply the java plugin to add support for Java
apply plugin: 'java'
```

Ajout du plugin 'java'

gradle :tasks

```
Build tasks
-----------
assemble - Assembles the outputs of this project.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
classes - Assembles classes 'main'.
clean - Deletes the build directory.
jar - Assembles a jar archive containing the main classes.
testClasses - Assembles classes 'test'.

Documentation tasks
-------------------
javadoc - Generates Javadoc API documentation for the main source code.

Verification tasks
------------------
check - Runs all checks.
test - Runs the unit tests.
```

Ajout du plugin 'java'

gradle build

```
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:jar UP-TO-DATE
:assemble UP-TO-DATE
:compileTestJava
[..]/ImmutableComplex/src/test/java/ComplexTest.java:6: error: package org.junit does not exist
import org.junit.Assert;
                ^
...
  symbol:   variable Assert
  location: class ComplexTest
31 errors
:compileTestJava FAILED

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':compileTestJava'.
> Compilation failed; see the compiler error output for details.

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log output.

BUILD FAILED

Total time: 0.68 secs
```

Il faut préciser la dépendance à JUnit.

Ajout des dépendances

Dépendances locales

```
// Apply the java plugin to add support for Java
apply plugin: 'java'

repositories {
        flatDir {dirs 'lib'}
}
dependencies {
        compile "hamcrest:hamcrest-all:1.3" // group:name:version
        testCompile "junit:junit:4.11"
}
```

Ajout des dépendances

Utilisation du dépôt Maven

```
// Apply the java plugin to add support for Java
apply plugin: 'java'

repositories {
        jcenter()
}
dependencies {
        testCompile name: 'junit-4.11'
        // hamcrest dependency will be solved by gradle
}
```

- Les dépendances récursives sont gérées automatiquement.

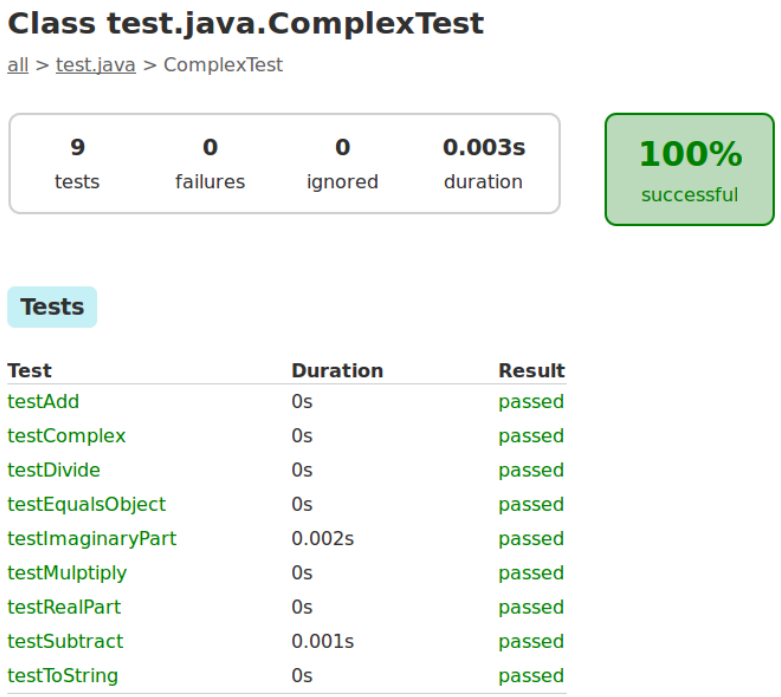
Ajout des dépendances

Utilisation du dépôt Maven

```
$ gradle build
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:jar UP-TO-DATE
:assemble UP-TO-DATE
:compileTestJava
Download https://jcenter.bintray.com/junit/junit/4.11/junit-4.11.pom
Download https://jcenter.bintray.com/org/hamcrest/hamcrest-core/1.3/hamcrest-core-1.3.pom
Download https://jcenter.bintray.com/org/hamcrest/hamcrest-parent/1.3/hamcrest-parent-1.3.pom
Download https://jcenter.bintray.com/junit/junit/4.11/junit-4.11.jar
Download https://jcenter.bintray.com/org/hamcrest/hamcrest-core/1.3/hamcrest-core-1.3.jar
:compileTestJava UP-TO-DATE
:processTestResources UP-TO-DATE
:testClasses UP-TO-DATE
:test UP-TO-DATE
:check UP-TO-DATE
:build UP-TO-DATE

BUILD SUCCESSFUL

Total time: 4.139 secs
```

Exécution des tests

- Les tests sont lancés par défaut par la tâche `build`.
- On peut les lancer explicitement avec la tâche `test`.
- Les résultats des tests sont disponibles au format xml ou html dans le répertoire *"build"*.



Class test.java.ComplexTest

all > test.java > ComplexTest

9	0	0	0.003s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
testAdd	0s	passed
testComplex	0s	passed
testDivide	0s	passed
testEqualsObject	0s	passed
testImaginaryPart	0.002s	passed
testMulptiply	0s	passed
testRealPart	0s	passed
testSubtract	0.001s	passed
testToString	0s	passed

Génération de la documentation java

- La documentation java est générée par la tâche `javadoc`.
- Elle se trouve dans le répertoire *"build/docs"*.

Ajout du plugin 'application'

Dans *build.graddle*

```
// Apply the java plugin to add support for Java
apply plugin: 'java'
apply plugin: 'application'

applicationName = "ImmutableComplex"
mainClassName = "main.java.Main"
```

gradle :tasks

```
Application tasks
-----------------
installApp - Installs the project as a JVM application along with libs and OS specific scripts.
run - Runs this project as a JVM application

Distribution tasks
------------------
assembleMainDist - Assembles the main distributions
distTar - Bundles the project as a distribution.
distZip - Bundles the project as a distribution.
installDist - Installs the project as a distribution as-is.
```

Exécution du Main

```
$ gradle run
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:run
(2.0 + 0.0i)
false
true

BUILD SUCCESSFUL

Total time: 2.077 secs
```

Distribuer, installer

- L'utilisation de la tâche `installDist` du pluggin 'application' créé un répertoire *"build/install"* contenant le projet compilé (en jar) ainsi que les exécutables s'il y en a.
- Gradle définit des tâches de publication sous forme de dépôts Maven. Aujourd'hui il faut utiliser la tâche `upload` en spécifiant le chemin du dépôt. Mais le plugin 'maven-publish' devrait la remplacer progressivement.
- Pour installer le projet localement on peut utiliser une tâche de copie et une propriété du projet définissant le chemin d'installation.

Distribuer, installer

```
// Local path for Maven depository
uploadArchives {
    repositories {
       flatDir {
           dirs 'repos'
       }
    }
}
// Local installation task
//'installDir' is property that should be defined with : "gradle
     installLocally -PinstallDir="/install/path""
task installLocally(type: Copy){
    from installDist
    into project.hasProperty('installDir') ? file(project.getProperty('
         installDir')) : file('install')
}

installLocally.onlyIf {
        project.hasProperty('installDir')
}
```

Distribuer, installer

```
$ gradle installLocally -P installDir="/mypath/install"
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:jar UP-TO-DATE
:startScripts UP-TO-DATE
:installDist UP-TO-DATE
:installLocally UP-TO-DATE

BUILD SUCCESSFUL

Total time: 2.464 secs
```

Construction avec Gradle.

- Souplesse et adaptabilité grâce au DSL.
- Le saut sémantique et programmatique peut être élevé quand on vient de C++/CMake.
- Défaut de ses qualités : la sémantique est fort peu lisible...