

# Tests unitaires et tests d'IHM sur un projet Android utilisant Maven



par Maxime Ghignet ([Site Web](#))

Date de publication : 25 janvier 2012

Dernière mise à jour :

Cet article a pour but de montrer comment créer un projet de test sur un projet Android utilisant Maven. On peut lire tout et son contraire sur la toile concernant la configuration à mettre en place.

Partant du principe que le projet fonctionne et a besoin d'être testé de manière automatique, ce tutoriel est destiné aux personnes ayant un niveau avancé sur Android.

I - Introduction.....	3
II - Créer un projet de test.....	3
III - "Maveniser" le projet !.....	4
IV - Créer une classe de test.....	8
V - Remerciements.....	10
VI - Références.....	10

## I - Introduction

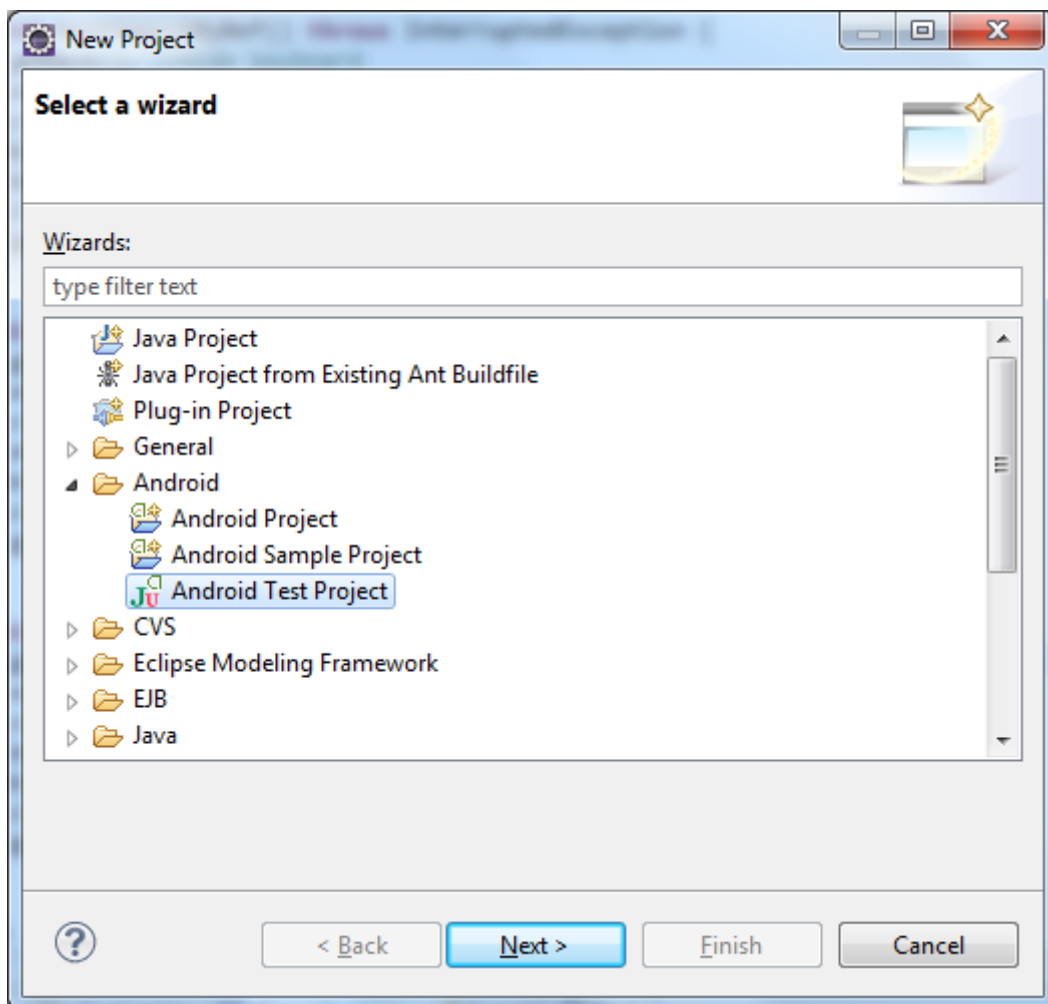
Dans ce tutoriel nous utiliserons l'EDI recommandé pour Android : Eclipse.

Supposons que vous ayez déjà un projet Android qui compile. Nous l'appellerons ici AdeoAndroidApp.

Nous utiliserons aussi **Maven** pour gérer les dépendances (disponible en installant "Android Configurator for M2E" sur l'Eclipse Marketplace) et le déploiement, ce qui rend la manipulation plus particulière que ce qui est indiqué sur la **documentation Android**.

## II - Créer un projet de test

- Nous le nommerons *AdeoAndroidAppTest*.
- Choisir "Create new project in Workspace" (c'est un nouveau projet, on ne réutilise pas les sources du projet à tester).
- À l'étape suivante, choisir le projet cible. Dans notre cas, *AdeoAndroidApp*.
- Choisir ensuite la version du SDK utilisée dans le projet cible, puis valider pour quitter l'assistant.



*On crée un projet de type Android Test project.*

Nous avons maintenant un projet de test créé, qui cible notre projet à tester. En ouvrant le *AndroidManifest.xml* du projet, on observe :

### AndroidManifest.xml

```

<instrumentation
    android:name="android.test.InstrumentationTestRunner"
    android:targetPackage="com.adeo.android.app" />

```

Preuve que le projet cible bien le bon package.

### III - "Maveniser" le projet !

Il faut pour cela créer un fichier pom.xml (POM = Project Object Model) à la racine du projet.

Globalement, on peut reprendre le pom.xml du projet cible, en changeant l'artifactId du projet (AdeoAndroidAppTest) et même chose pour le nom.

Dans les dépendances, nous allons tout enlever et en renseigner trois :

- le projet cible. Maven gère la transitivité donc nul besoin de redéclarer les dépendances du projet à tester ;
- le package *android-test* qui va nous permettre d'utiliser JUnit ;
- le package *robotium-solo* qui nous permettra d'utiliser **Robotium**, l'outil de test d'IHM.

Enfin, les plugins Maven (Maven Android Plugin et Compiler Plugin) et autres sont déjà renseignés dans la partie des plugins si vous avez copié/collé le *pom.xml* du projet cible, nul besoin d'y toucher.

Cela donne ce genre de fichier :

```
pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.adeo.android</groupId>
  <artifactId>AdeoAndroidAppTest</artifactId>
  <version>1.1.2-SNAPSHOT</version>
  <packaging>apk</packaging>
  <name>AdeoAndroidAppTest</name>

  <dependencies>

    <dependency>
      <groupId>com.google.android</groupId>
      <artifactId>android-test</artifactId>
      <version>2.3.1</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>com.adeo.android</groupId>
      <artifactId>AdeoAndroidApp</artifactId>
      <scope>provided</scope>
      <type>jar</type>
      <version>1.1.2-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>com.jayway.android.robotium</groupId>
      <artifactId>robotium-solo</artifactId>
      <version>3.0</version>
    </dependency>

  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>com.jayway.maven.plugins.android.generation2</groupId>
        <artifactId>maven-android-plugin</artifactId>
        <version>2.8.3</version>
        <configuration>
          <androidManifestFile>${project.basedir}/AndroidManifest.xml</androidManifestFile>
          <assetsDirectory>${project.basedir}/assets</assetsDirectory>
          <resourceDirectory>${project.basedir}/res</resourceDirectory>
          <sourceDirectory>${project.basedir}/src</sourceDirectory>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

pom.xml

```
<nativeLibrariesDirectory>${project.basedir}/src/main/native</nativeLibrariesDirectory>
<deleteConflictingFiles>true</deleteConflictingFiles>
<undeployBeforeDeploy>true</undeployBeforeDeploy>
</configuration>
<extensions>true</extensions>
</plugin>

<plugin>
<artifactId>maven-compiler-plugin</artifactId>
<version>2.3.2</version>
<configuration>
<source>1.6</source>
<target>1.6</target>
</configuration>
</plugin>
</plugins>
</build>
<properties>
</properties>
</project>
```

À noter que les deux dépendances *android-test* et *AdeoAndroidApp* (notre projet cible) ont un scope *provided*.

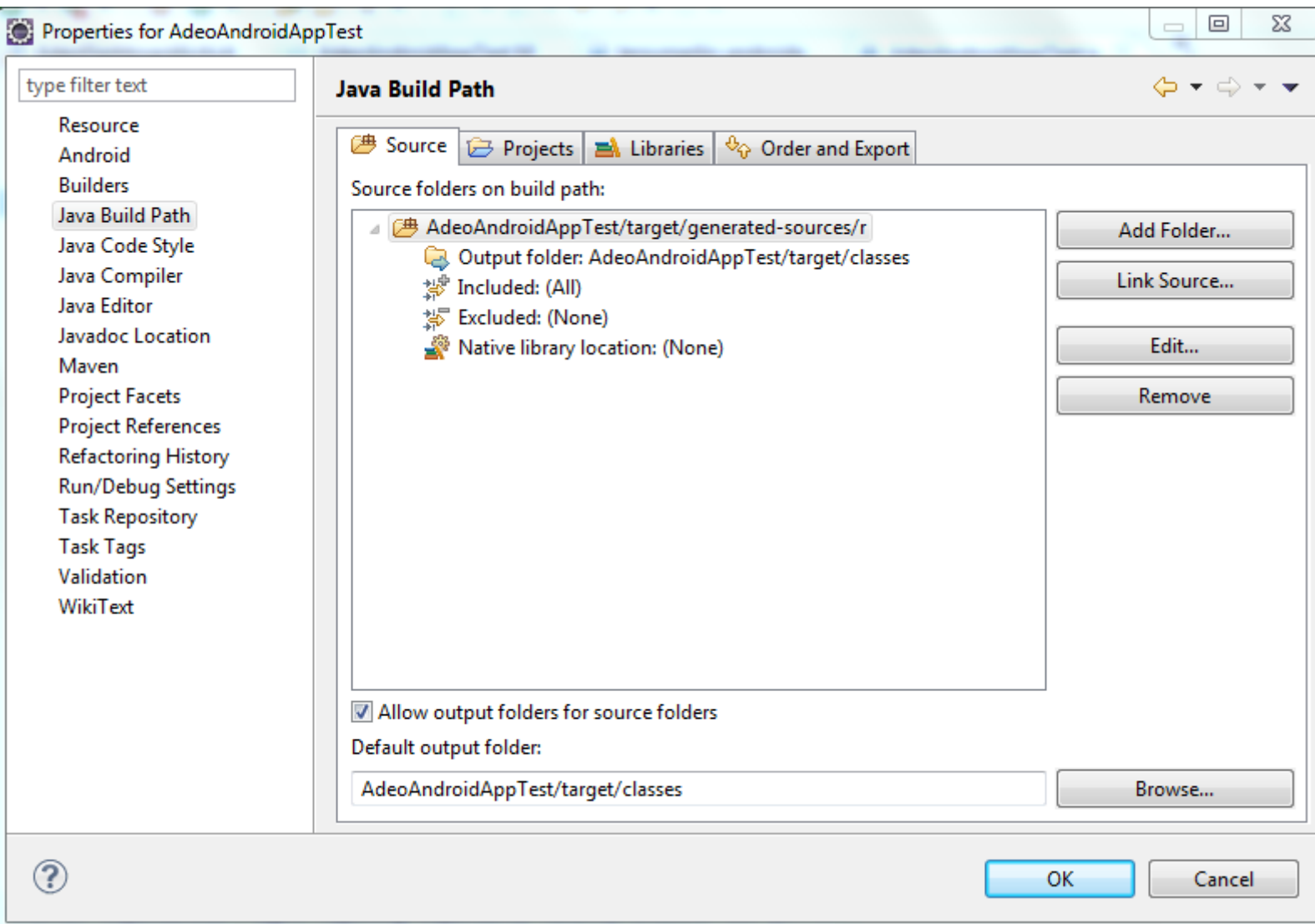
De plus, on a inclus *AdeoAndroidApp* en type **jar** et non **apk**.

Après l'enregistrement du *pom.xml*, le projet n'est pas encore complètement "mavenisé". On va activer la gestion des dépendances par un **clic droit sur le projet de test => Maven => Enable Dependency Management**.

Cela a pour effet d'ajouter plusieurs containers, dont *Maven Dependencies*. En l'ouvrant, on observe les dépendances du projet cible (car Maven gère la transitivité) et celles que l'on a ajoutées : le projet cible lui-même, JUnit, Robotium...

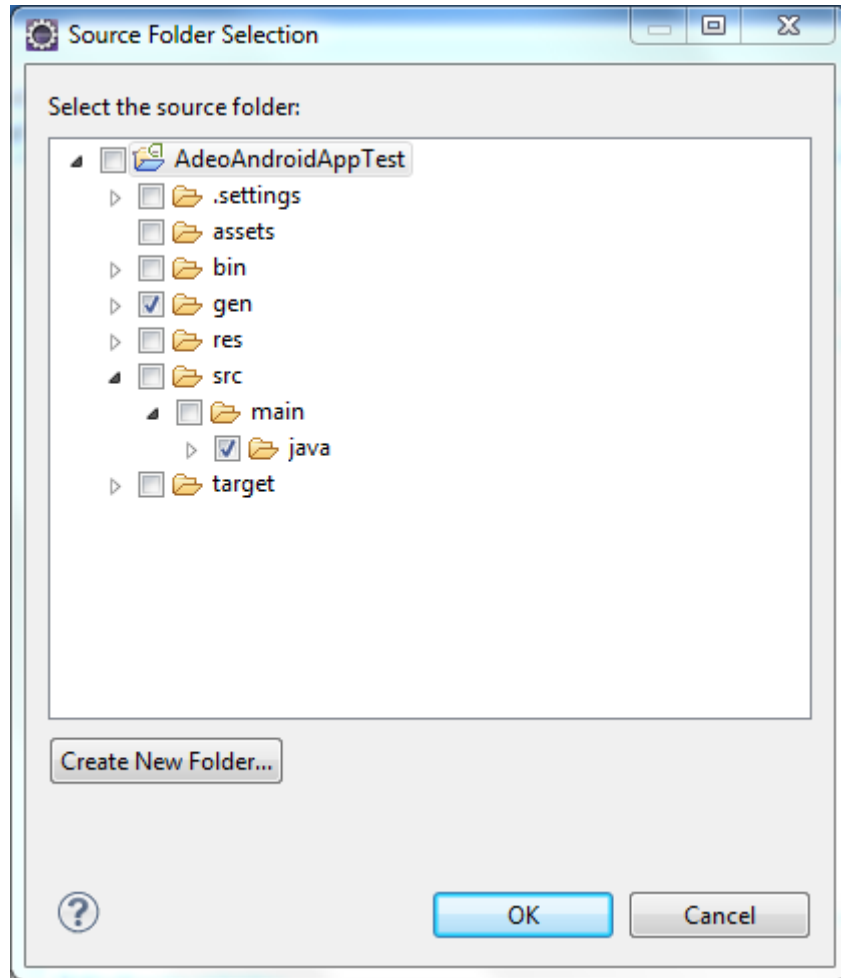
De plus, le dossier *src* est devenu visible dans l'explorateur de projet Eclipse. Pour faire fonctionner les tests, j'ai réorganisé ce dossier en plaçant les dossiers du package (*com.adeo.android.app*) dans *src/main/java/* et non plus dans *src/*, ce qui donne : *src/main/java/com/adeo/android/app/*. Le dossier *test/* peut être supprimé. Nous placerons nos classes de test dans le dossier *src/main/java/com/adeo/android/app/*.

Aussi, nous allons éditer les chemins de build de Java afin de faire correspondre notre arborescence : **clic droit sur le projet de test => sélectionner Java Build Path** puis supprimer les chemins déjà présents.



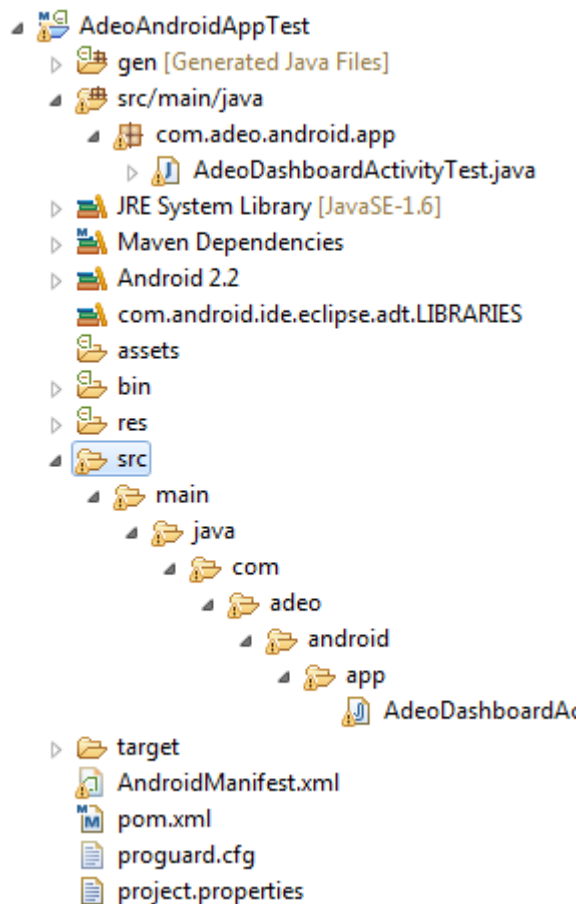
Fenêtre de chemin(s) de build

On va ajouter les dossiers *gen/* et *src/main/java*, comme sur la capture d'écran suivante :



*On sélectionne les bons dossiers de build*

On obtient ainsi une arborescence de ce type :



Arborescence du projet de test

## IV - Créer une classe de test

Nous allons créer une classe nous permettant de tester une Activity (placée dans le répertoire `src/main/java/com/adeo/android/app/`).

- Elle hérite de la classe `ActivityInstrumentationTestCase2`.
- Elle contient forcément une méthode `setUp()` décrite ci-dessous.
- Elle peut contenir une méthode `testPreconditions()` qui sera lancée une seule fois avant la série de tests.
- Elle contient les méthodes de test.

Voici son code :

```
AdeoDashBoardActivityTest.java
package com.adeo.android.app;

import com.adeo.android.app.activity.AdeoDashboardActivity;

import android.app.Instrumentation;
import android.test.ActivityInstrumentationTestCase2;
import android.widget.TextView;

public class AdeoDashboardActivityTest extends ActivityInstrumentationTestCase2 {

    private AdeoDashboardActivity mAdeoDashboardActivity;
    private Instrumentation mInstrumentation;
    private TextView mTextView;

    private static final String TARGET_PACKAGE = "com.adeo.android.app";

    @SuppressWarnings("unchecked")
```

**AdeoDashBoardActivityTest.java**

```

public AdeoDashboardActivityTest () {
    super(TARGET_PACKAGE, AdeoDashboardActivity.class);
}

@Override
protected void setUp() throws Exception {
    super.setUp();

    setActivityInitialTouchMode(false);

    mAdeoDashboardActivity = (AdeoDashboardActivity) getActivity();
    mInstrumentation = getInstrumentation();

    mTextView = (TextView) mAdeoDashboardActivity.findViewById(com.adeo.android.app.R.id.my_textview);
}

public void testPreconditions() {
    //Indiquez ici vos préconditions. Cette méthode est exécutée une seule fois, avant l'exécution du premier test.
    //Par exemple :
    assertNotNull(mTextView);
}

public void testText() {
    String resourceString = "Ok";
    assertEquals(resourceString, (String)mTextView.getText());
}
}
    
```

Jusqu'ici, notre classe de test n'effectue que des tests basiques sur les états des composants de l'Activity et non des tests d'IHM. La dépendance *robotium-solo* que nous avons ajoutée dans le *pom.xml* va nous servir maintenant. Ainsi, on instancie un objet *Solo* dans notre méthode *setUp()*. Cet objet est un attribut de la classe de test, afin de pouvoir l'utiliser dans toutes les méthodes. Il suffit ensuite d'utiliser les méthodes de l'objet *Solo* pour effectuer les tests.

Le code de la classe utilisant Robotium serait alors :

**AdeoDashBoardActivityTest.java**

```

package com.adeo.android.app;

import com.adeo.android.app.activity.AdeoDashboardActivity;
import com.jayway.android.robotium.solo.Solo;

import android.app.Instrumentation;
import android.test.ActivityInstrumentationTestCase2;
import android.widget.TextView;

public class AdeoDashboardActivityTest extends ActivityInstrumentationTestCase2 {

    private AdeoDashboardActivity mAdeoDashboardActivity;
    private Instrumentation mInstrumentation;
    private Solo solo;
    private TextView mTextView;

    private static final String TARGET_PACKAGE = "com.adeo.android.app";

    @SuppressWarnings("unchecked")
    public AdeoDashboardActivityTest () {
        super(TARGET_PACKAGE, AdeoDashboardActivity.class);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();

        setActivityInitialTouchMode(false);

        mAdeoDashboardActivity = (AdeoDashboardActivity) getActivity();
    }
}
    
```

### AdeoDashBoardActivityTest.java

```
mInstrumentation = getInstrumentation();

mTextView = (TextView) mAdeoDashboardActivity.findViewById(com.adeo.android.app.R.id.my_textview);

solo = new Solo(mInstrumentation, mAdeoDashboardActivity);

}

public void testPreconditions() {
    //Indiquez ici vos préconditions. Cette méthode est exécutée une seule fois, avant l'exécution du premier test.
    //Par exemple :
    assertNotNull(mTextView);
}

public void testText() {
    String resourceString = "Ok";
    assertEquals(resourceString, (String)mTextView.getText());
}

public void testButtonClick() {
    //On simule un clic sur le bouton 0, par exemple
    solo.clickOnButton(0);
}

@Override
public void tearDown() throws Exception {
    solo.finishOpenedActivities();
}
}
```

En lançant le build, vous pourrez regarder votre application Android s'animer et faire ce que vous lui avez demandé.

Voilà pour les tests unitaires et d'IHM sur un projet Android "mavenisé" !

## V - Remerciements

Je tiens à remercier **Nicolas Inchauspé** qui m'a aidé à acquérir les connaissances nécessaires à la réalisation de ce tutoriel.

Un merci aussi à **Mahefasoa** et **Claude LELOUP** pour la relecture et à **Feanorin** qui m'a aidé dans la mise en place de ce tutoriel sur le site.

## VI - Références

- [Introduction au SDK Android](#)
- [Importer un projet Maven dans Eclipse en 5 minutes](#)
- [3T : les Tests en Trois Temps](#)
- [Les Tests en Trois Temps \(3T\) en pratique](#)
- [À la découverte de JUnit](#)