Test apps en Android Studio

Alejandro Pérez Ruiz
Grupo de Ingeniería Software y Tiempo Real
perezruiza@unican.es

Pruebas unitarias

Las pruebas unitarias son métodos que prueban una unidad estructural de código. Para realizar este tipo de pruebas en nuestro entorno de desarrollo Android Studio haremos uso de JUnit para crear y ejecutar las pruebas.

Android Studio ofrece dos tipos de test:

- Tests locales (module-name/src/test/java/): Estos tests se ejecutan en la máquina local donde está instalado el entorno de desarrollo. Son útilizados para minimizar el tiempo de ejecución cuando los tests no tienen dependencia con el framework de Android.

- Tests instrumentados (module-name/src/androidTest/java): Estos tests se ejecutan sobre el telefono/tablet Android o el emulador.

Cuando se crea un nuevo proyecto para generar una nueva aplicación, Android Studio de manera automática crea los paquetes que contienen nuestros tests. En la figura 1 se ilustra cómo se visualizan.

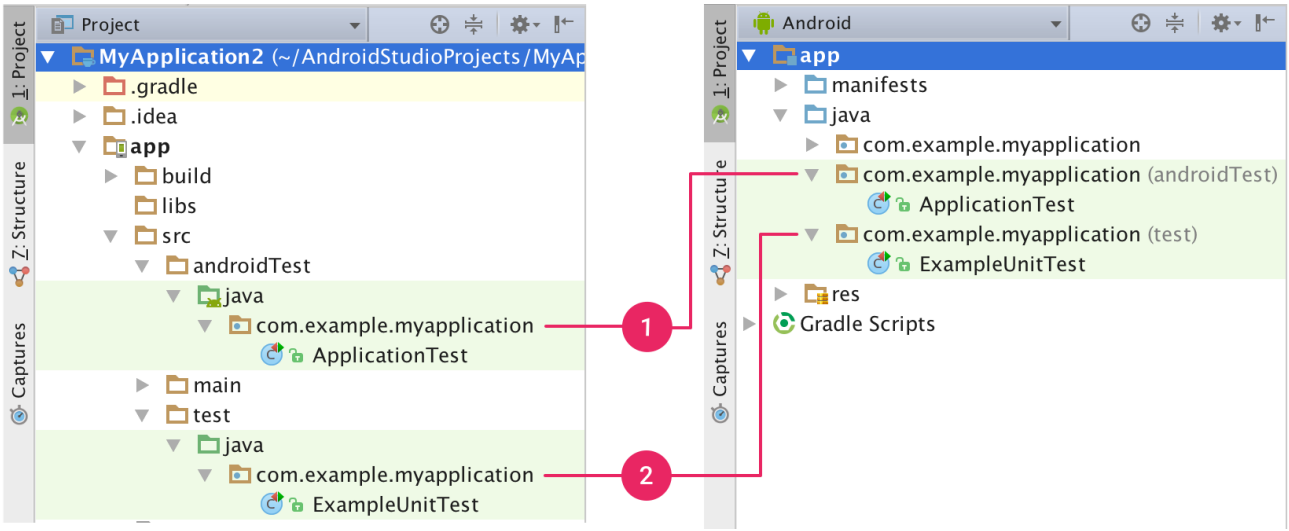


Figura 1. Tanto los tests instrumentados (1) como los locales (2) son visibles en la vista “Project” y en la vista “Android”.

Para crear un test unitario se puede realizar de manera muy sencilla para una clase o método concreto siguiendo los siguientes pasos:

1. Abrimos el fichero java que contiene el código que queremos probar.
2. Hacemos click sobre la clase o método (en el código fuente) que vayamos a probar y presionamos Ctrl+Shift+T
3. Nos aparecerá una opción denominada “Create New Test” y la seleccionamos.

4. En el diálogo (figura 2) que se nos muestra podemos seleccionar la versión de JUnit que utilizaremos y los métodos que deseamos probar.

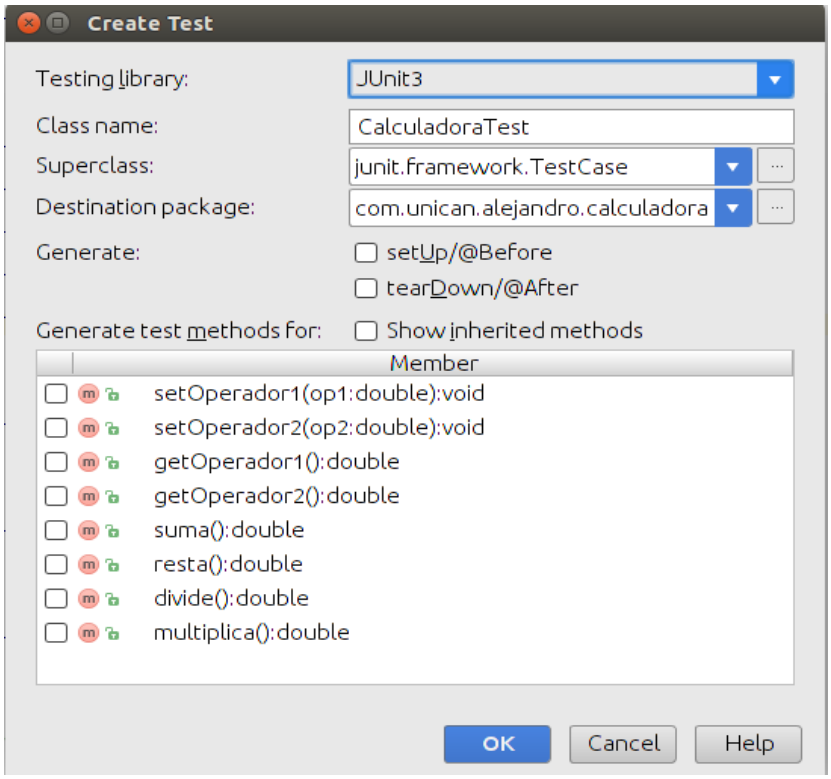


Figura 2. Diálogo que nos muestra Android Studio para generar tests unitarios

Una vez que hemos escrito los tests unitarios se deben ejecutar del siguiente modo:

1. Seleccionamos la clase de test que queremos ejecutar en la vista de “Project”.
2. Con el botón derecho del ratón nos aparecer un menú contextual donde debemos pulsar sobre “Run”.

Pruebas de integración

En nuestras prácticas vamos a realizar tests funcionales sobre la interfaz de usuario de nuestras aplicaciones. Dentro de Android Studio de forma “nativa” se nos incluyen dos frameworks para este cometido:

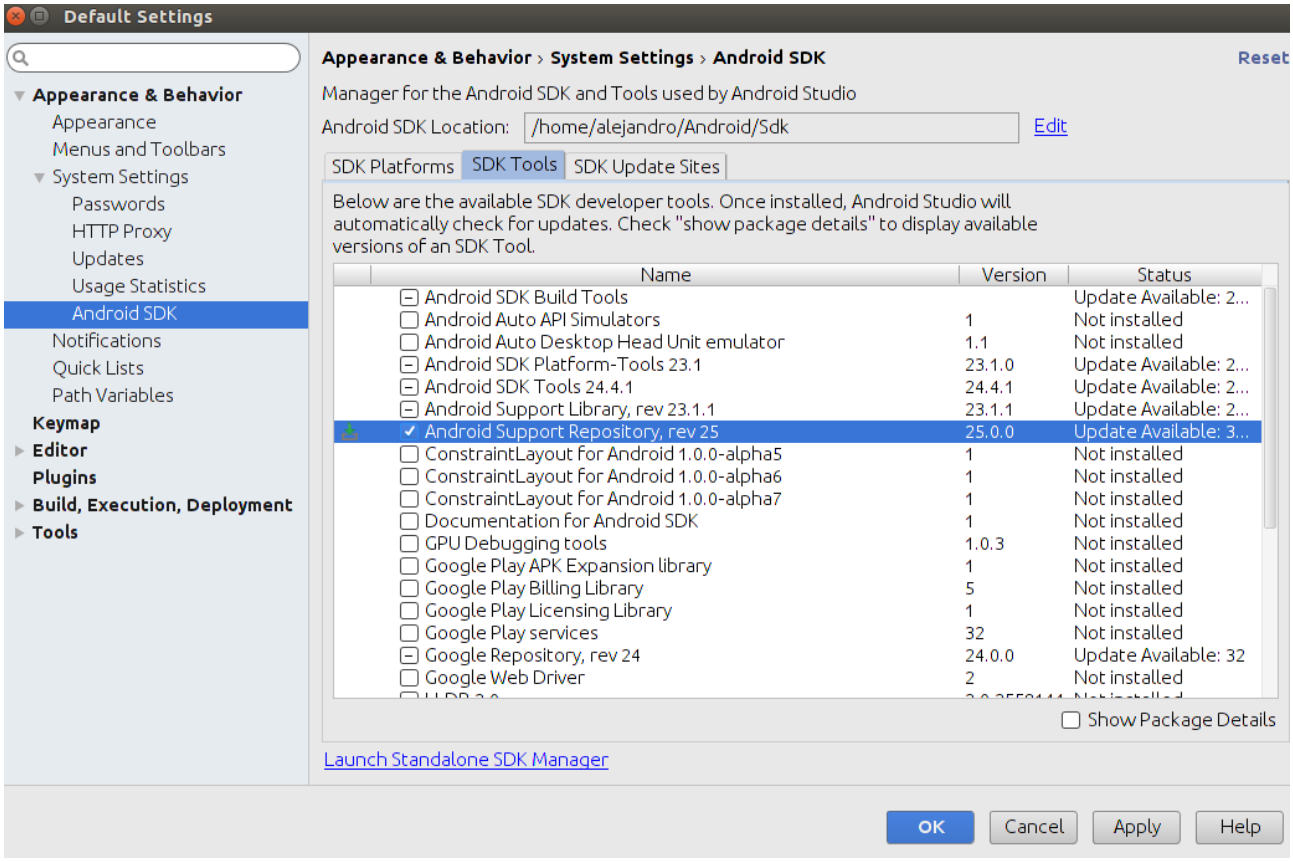
- **UI Automator:** proporciona un conjunto de APIs para construir test sobre la interfaz de usuario de la aplicación. Está pensado para escribir test de caja negra donde el código de las pruebas no se basa en los detalles de la implementación interna de la aplicación. Está orientado a realizar pruebas de integración con el sistema operativo, por ejemplo pulsado del botón de “back”, cambiar la rotación del dispositivo... Requiere de una versión de Android 4.3 (API level 18) o superior.

- **Espresso:** nos deja disponibles una serie de APIs para realizar tests de la interfaz de usuario siendo estos del tipo caja blanca donde el código de las pruebas utiliza los detalles de la implementación interna. Es soportado desde la versión de Android 2.2 (API level 8).

Para nuestros proyectos principalmente vamos a utilizar Espresso (https://google.github.io/android-

testing-support-library/docs/espresso/index.html)

Para instalar Espresso (en el caso de que no os apareciese) debemos seguir los siguientes pasos descritos a continuación:
1. Tools → Android → SDK Manager.
2. Seleccionar la pestaña SDK Tools, marcar Android Support Repository (puede que no salga listado en los ajustes de Android Studio, si ocurre esto se debe lanzar el SDK Manager y buscarlo allí) y pulsar OK para que comience la descarga y la instalación.



Para configurar Espresso en una aplicación debemos realizar lo sugiente (**por defecto esto ahora en los proyectos ya está añadido**):
1. Abrimos el fichero build.gradle de la app
2. Añadimos en el apartado de las dependencias las necesarias para Espresso.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.4.0'
        // dependencies for Espresso
        // Maybe, you should change the version of annotations if you get an
error
    androidTestCompile 'com.android.support:support-annotations:23.4.0'
    androidTestCompile 'com.android.support.test:runner:0.3'
    androidTestCompile 'com.android.support.test:rules:0.3'
    androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.1'
    // add this for intent mocking support
    androidTestCompile 'com.android.support.test.espresso:espresso-
intents:2.2.1'
    // add this for webview testing support
    androidTestCompile 'com.android.support.test.espresso:espresso-web:2.2.1'
```

```
}
```

3. En la sección defaultConfig del anterior fichero debemos añadir una nueva línea

```
defaultConfig {

    applicationId "es.unican.calculadora2016"
    minSdkVersion 15
    targetSdkVersion 23
    versionCode 1
    versionName "1.0"
        //It is necessary to run tests
    testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
}
```

Para ejecutar un test Espresso:

1. Pulsamos con el botón derecho sobre el fichero .java que contiene el código y seleccionamos Run.

Para la creación y ejecución de los tests con Espresso tenemos un tutorial bastante detallado en la siguiente url: http://www.vogella.com/tutorials/AndroidTestingEspresso/article.html
La forma más sencilla de crear los tests es utilizar la opción que aparece en Run→Record Espresso Test.
No obstante en ocasiones vamos a tener que codificar parte de los test nosotros mismos sin hacer uso de la herramienta gráfica.

Posibles problemas con Espresso

Solución a posible problema (*duplicate file exception*) con la ejecución de los tests de Espresso: http://stackoverflow.com/questions/35774744/using-contrib-and-web-espresso-imports-causes-duplicatefileexception/35774745

Utilización de mocks

Un mock es una clase que simula el comporrtamiento de otra clase más compleja, por ejemplo una clase que obtiene datos a través de internet o de una base de datos. En Android, los mocks nos permiten simular el contexto, escribir datos en sistesmas de ficheros, etc.
Encontramos una breve introducción a su uso en la siguiente url de la documentación oficial: https://developer.android.com/training/testing/unit-testing/local-unit-tests.html#mocking-dependencies