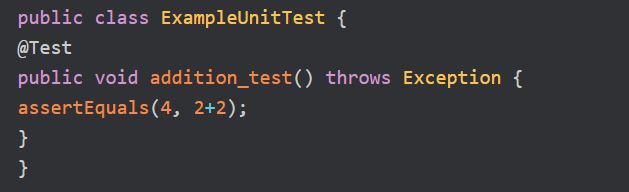
Testing con JUnit

La classe predisposta per i test unitari sarà del tipo:



Si tratta di una classe java, il cui unico metodo ha due caratteristiche particolari:

* L’annotazione @Test, per indicare che quel metodo rappresenta un test.
* Il metodo *assertEquals.* In questo esempio, il test consiste nel confrontare due valori, e si considera passato se essi risultano uguali. In generale il primo parametro è il valore che ci si aspetta di ottenere, mentre il secondo generalmente invoca un metodo il cui risultato dovrebbe essere uguale al primo.

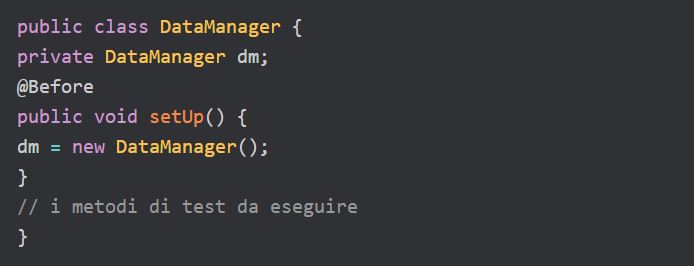
I colori associati ai singoli test hanno un significato particolare:

* Verde: il test ha avuto successo;
* Giallo: l’esecuzione non ha riscontrato problemi ma il test non ha avuto successo;
* Rosso: durante il test è fallita l’esecuzione del codice per esempio per via di una eccezione non gestita.

**Utilizzare Hamcrest**

Hamcrest è una libreria che permette di aumentare notevolmente la leggibilità del codice di collaudo.

Quando vengono implementati metodi che utilizzano algoritmi in cui si richiede la sua effettiva correttezza in tutti i casi di esecuzione allora Hamcrest ci viene in aiuto.

Hamcrest introduce l’annotazione *@Before* che introduce un metodo da eseguire prima di ogni test: il suo scopo principale è quello di permettere una partenza pulita del test successivo. 

Analogamente esiste l’annotazione *@After* per poter effettuare operazioni di pulizia e di chiusura di eventuali collegamenti a risorse esterne.

Vediamo un esempio:



Per il confronto utilizziamo il metodo *assertThat* che permette di costituire confronti più liberi tra volere atteso e quello reale.

Completiamo il confronto sfruttando i metodi *is* e *equalTo* offerti da Hamcrest.

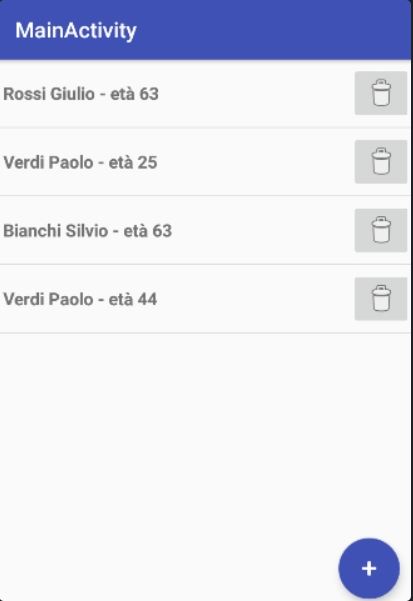
Il secondo specifica che siamo alla ricerca di una uguaglianza mentre il primo contribuisce per lo più alla leggibilità.



Il metodo *hasSize* torna utile in questi casi per confrontare la dimensione di liste/array etc…

**Il test delle interfacce utente**

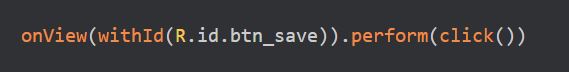
La figura seguente mostra l’Activity, che include una *ListView* con alcuni elementi inseriti in fase di inizializzazione.

La funzionalità che vogliamo sottoporre a test consiste nella cancellazione di un elemento gestito dall’Adapter.

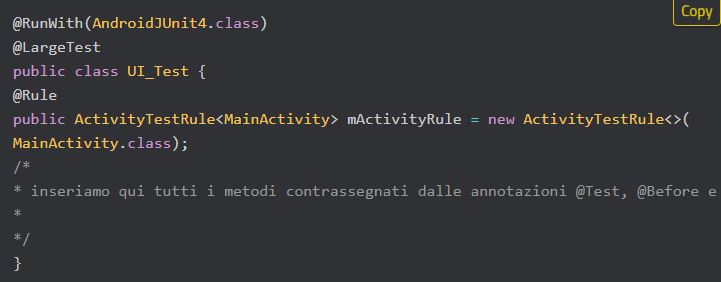
I test prodotti con Espresso risultano perfettamente fluidi per questo tipo di test.

Ogni operazione sarà distribuita in tre fasi:

* Invocazione del componente visuale sul quale simulare l’interazione. Faremo ciò con il metodo *onView()*, mentre useremo il metodo *onData()* per gli elementi di un Adapter.
* Effetueremo l’azione tramite il metodo *perform()*:
* Effettueremo controlli tramite *ViewAssertions* per verificare gli effetti dell’operazione eseguita.



La struttura della classe dei test sarà in stile JUnit.

Per poter avere a disposizione una Activity prima di ogni test, sfrutteremo la classe *ActivityTestRule* che verrà istanziata per poi essere distrutta.

Prendendo da esempio la cancellazione di una persona dalla listview precedente creiamo il test:



Noteremo come questo tipo di test sia una simulazione di ciò che un ipotetico utente potrebbe fare.