Testing

Gabriele Chignoli

Luglio 2025

Contents

1 Introduzione

Nel seguente documento vengono mostrate le varie attività di test eseguito sul software Pantrymanager. Si intende documentare per ogni componente testata il nome e la posizione del componente, la data di esecuzione, il tipo e l'obiettivo del test ed i risultati ottenuti.

2 Test Unitari

2.1 Costruttore classe `Product` - 25/07/2025

Obiettivo Viene testato il costruttore di oggetti *Product*

```
public Product(String name, float weight, int qty,
    float calories, LocalDate expiration_date) {
        ...
}
```

Input Sono stati utilizzati 5 valori differenti per testare il nome del prodotto, 5 per il peso, e solamente 1 per gli altri attributi. Sono stati inclusi anche valori che non hanno senso per quello che rappresentano, ma verranno vincolati più avanti e ad un livello superiore della struttura, verso l'interfaccia utente.

Aspettative e risultati Il test era molto semplice e testava una parte molto piccola di codice che difficilmente risulta problematica. L'obiettivo era prendere confidenza con JUnit, per poi provare a seguire in futuro anche un approccio guidato dei test.

I risultati del test vengono caricati attraverso un file xml esportato direttamente da Eclipse.

Directory Componente

```
../src/main/java/controller/Product.java
```

Directory Test

```
../src/test/java/controller/ProductTest.java
```

2.2 `ProductManager` - 26/08/2025

Obiettivo Viene testata la classe `ProductManager` che effettua le operazioni sul DataBase, in particolare i seguenti metodi:

```
public static int saveProduct(Product to_save) {
    return run(OperationMode.SAVE, to_save);
}

public static int deleteProduct(String name) {
    ...
    return run(OperationMode.DELETE, to_delete);
}

public static int modifyProduct(Product p) {
```

```
    return run(OperationMode.MODIFY, p);
}

public static List<Product> getProducts() {
        ...
    return entity_manager.createQuery(criteria).getResultList();
}
```

Input Sono stati utilizzati diversi prodotti, per i quali viene verificato che le operazioni abbiano correttamente salvato, eliminato o modificato nel database `PANTRY.mv.db`.

Aspettative e risultati Il test voleva verificare principalmente una corretta interazione con il database, per la quale si considera il test superato; tuttavia ha anche evidenziato come gli input vengano semplicemente salvati senza essere controllati. In particolare, a parte il nome che non può essere omesso e che deve essere unico, gli altri parametri numerici possono assumere valori negativi o comunque non voluti se si agisce a questo livello.

Passando per l'interfaccia grafica questo problema viene evitato non permettendo all'utente di inserire certi valori; rimane da decidere se introdurre un ulteriore verifica a questo stadio.

Directory Componente

../src/main/java/controller/ProductManager.java

Directory Test

../src/test/java/controller/ProductManagerTest.java

2.3 `MainWindow` - 28/08/2025

Obiettivo Viene testata la classe `MainWindow` che gestisce l'interfaccia grafica. Purtroppo la classe non risulta facilmente testabile: per i metodi getter l'operazione è immediata, mentre per verificare che l'interfaccia grafica si comporti correttamente si è stati in grado solo di verificare che venisse instanziata la classe e fossero ritornati i valori attesi.

Input Non sono stati utilizzati input ma si è verificato che la classe istanziasse o non istanziasse (ritornasse null) per i metodi getter. Si consiglia la visione dei file su *Github*.

Aspettative e risultati Il test dell'interfaccia rimane un'attività da approfondire, poiché, pur essendo stato verificato durante lo sviluppo che gli input fossero salvati e modificati correttamente utilizzando l'interazione e verifica manuale dello sviluppatore, sarebbe più opportuno automatizzare il processo.

Una delle possibili soluzione sarebbe predisporre la classe `MainWindow` di metodi che simulino l'input dell'utente modificando i campi senza passare per l'interfaccia, ma agendo sui componenti direttamente. Poi attraverso una classe esterna eseguire i metodi e verificarne l'output.

Directory Componente

../src/main/java/view/MainWindow.java

Directory Test

../src/test/java/view/MainWindow.java

2.4 Test di Sistema

Risultati di tutti i test eseguiti insieme.

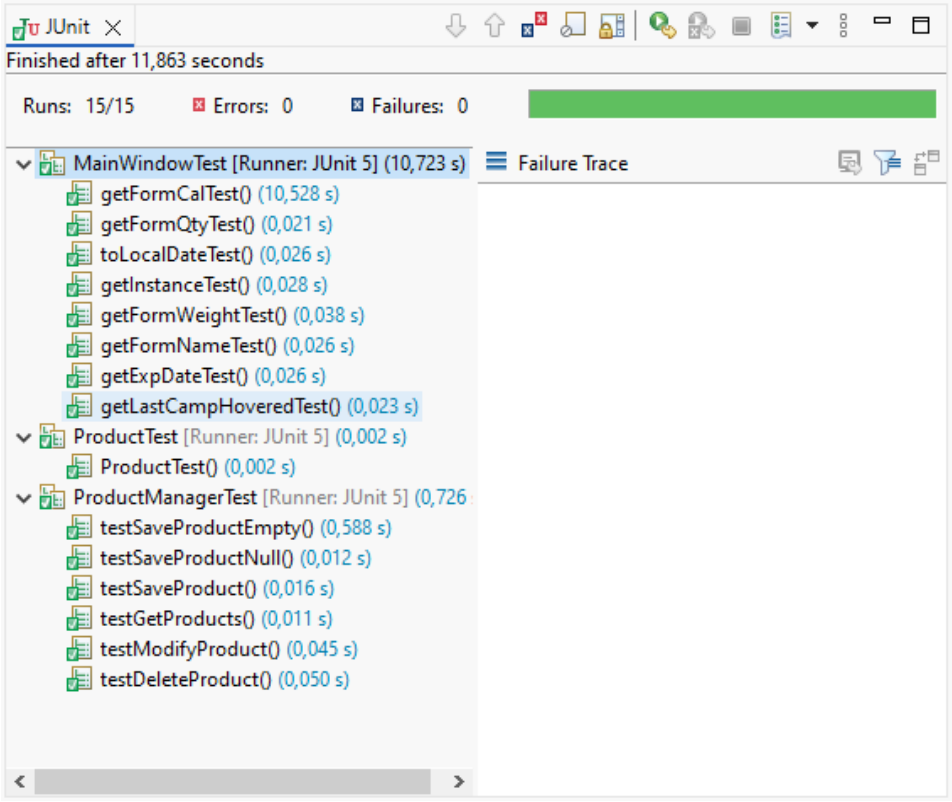

Figure 1: System Test results

2.5 Test di Copertura

Viene mostrato quanto codice i test coprono, ovvero quante righe di codice eseguono (raggiungono) e testano.

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
pantrymanager	66,2 %	1.368	700	2.068
src/main/java	58,7 %	996	700	1.696
controller	14,0 %	44	270	314
ModifyProductAction.java	6,7 %	8	111	119
SaveProductAction.java	10,1 %	8	71	79
DeleteProductAction.java	11,1 %	4	32	36
RemoveButtonAction.java	16,0 %	4	21	25
AddProductAction.java	17,4 %	4	19	23
NextPageAction.java	44,4 %	4	5	9
PrevPageAction.java	44,4 %	4	5	9
SearchMtxAction.java	57,1 %	4	3	7
ShowExpirationDateAction.java	57,1 %	4	3	7
model	52,1 %	264	243	507
DishManager.java	0,0 %	0	84	84
ExpirationDateManager.java	0,0 %	0	58	58
Dish.java	0,0 %	0	38	38
ProductManager.java	82,3 %	167	36	203
Product.java	61,4 %	43	27	70
OperationMode.java	100,0 %	54	0	54
view	84,3 %	688	128	816
MatrixRenderer.java	58,1 %	61	44	105
MainWindow.java	93,9 %	627	41	668
SwipePage.java	0,0 %	0	24	24
Launcher.java	0,0 %	0	19	19
database	0,0 %	0	59	59
DatabaseInit.java	0,0 %	0	59	59
src/test/java	100,0 %	372	0	372

Figure 2: Test Coverage

Considerazioni

- Come si può vedere non si è riusciti a testare le azioni del `controller`, legate alla interfaccia utente.
- Le classi legate a `dish` non sono utilizzate nella versione rilasciata e quindi non sono state testate.
- `MainWindow` ha una buona copertura ma non si ritiene comunque di aver testato al meglio la classe.