apt-subscriptions README

Lorenzo Maffucci

2025-12-19

TL;DR

Build:

```
mvn clean verify -Pjacoco,mutation-testing
```

Build nel virtual framebuffer:

```
xvfb-run -a --server-args="-screen 0 1366x768x24" \
  mvn clean verify -Pjacoco,mutation-testing
```

Esecuzione manuale del programma con alcuni dati campione:

```
./setup.sh
java -cp target/apt-subscriptions-1.0-SNAPSHOT-jar-with-\
  dependencies.jar org.blefuscu.apt.subscriptions.\
  SubscriptionsSwingApp
./teardown.sh
```

---

1. Specifiche

Voglio leggere da un database le informazioni relative agli ordini di acquisto di copie o abbonamenti a una rivista, filtrare e formattare i risultati, e quindi esportarli in un file in cui mantengo solo alcuni campi a cui sono interessato.

Voglio visualizzare un elenco sintetico degli ordini presenti e i dettagli di un particolare ordine selezionato. Voglio fare operazioni di lettura, aggiornamento e cancellazione sul database degli ordini.

Una funzione che mi interessa particolarmente è poter specificare un intervallo di date ed estrarre gli ordini conclusi in quel periodo.

1.1 Ambiente locale

Con il gestore di pacchetti del S.O. (Arch Linux) installo:

- OpenJDK 8.472.u08-1
- Eclipse 2025-12

- Maven 3.9.11
- Git 2.52.0
- Docker Engine 29.1.3

Sul sistema sono già presenti le versioni 17 e 24 di OpenJDK; imposto la 8 come default:

```
# archlinux-java set java-8-openjdk
```

Tramite il Marketplace di Eclipse installo alcuni plugin:

- Pitclipse 2.2.0
- Docker Tooling 5.18.1
- SonarQube for IDE 11.22
- WindowBuilder 1.22.0

Imposto il progetto Maven:

```
mvn archetype:generate \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-quickstart \
-DarchetypeVersion=1.1 \
-DgroupId=org.blefuscu \
-DartifactId=apt-subscriptions \
-DinteractiveMode=false
```

Dopodiché lo importo in Eclipse secondo la procedura descritta nella sezione 7.3.2 del libro.

Inizializzo il repository nella cartella `apt-subscriptions` appena creata:

```
git init
git config user.name "Lorenzo Maffucci"
git config user.email "lorenzo.maffucci@edu.unifi.it"
```

Quindi lo importo in Eclipse (Window > Show View > Other... > Git > Git Repositories) e lo connetto al repository remoto su GitHub:

```
git remote add origin git@github.com:mfflnz/apt-subscriptions.git
```

---

2. Tecniche e framework utilizzati

2.1 TDD Inizialmente faccio uno sketch del Model e scrivo le interfacce del Repository e delle View (File > New > Interface). Quindi comincio ad applicare i principi del TDD per costruire il Controller. Riporto di seguito un riepilogo dei primi passi dello sviluppo (per poi diradare i dettagli).

Faccio un *mock* del Repository e della View; quindi nel primo test verifico che il metodo `requestOrders()`, invocato senza parametri, riporti sulla View la lista di tutti gli ordini presenti. Per passare dalla fase *red* alla fase *green*, con il

content assist di Eclipse comincio a definire i primi tratti del Controller: i campi `listView` e `orderRepository` e il metodo `requestOrders()`.

Col test successivo scrivo una versione del metodo `requestOrders` che accetta due parametri di tipo `LocalDate` e verifico che ci siano interazioni coi *mock* del Repository e della View.

Proseguo con alcuni test su `requestOrders(LocalDate fromDate, LocalDate toDate)` che permetteranno di gestire i casi di errore sulle date verificando con AssertJ che si sollevino eccezioni accompagnate da messaggi pertinenti:

- `fromDate` non è specificato;
- `toDate` non è specificato;
- `fromDate` è successivo a `toDate`.

Verifico che il codice del Controller sia raggiungibile e che i mutanti siano eliminati.

Proseguo con l'implementazione di altre funzioni del Controller:

- recupero dei dettagli di un ordine;
- eliminazione di un ordine;
- modifica di un ordine;
- formattazione di una lista di ordini in cui i campi vengono trattati secondo i seguenti criteri (laddove non vi sono indicazioni, il contenuto del campo non viene manipolato):
  - `orderId`
  - `orderDate`: "2025-08-05 11:11:01" -> "05/08/2025"
  - `paidDate`: come sopra, se presente, altrimenti campo vuoto
  - `orderTotal`
  - `orderNetTotal`
  - `paymentMethodTitle`
  - `shippingFirstName` se presente, altrimenti `shippingFirstName` <- `billingFirstName`
  - `shippingLastName` se presente, altrimenti `shippingLastName` <- `billingLastName`
  - `shippingAddress1`: come sopra (medesimo criterio)
  - `shippingPostcode`: come sopra
  - `shippingCity`: come sopra
  - `shippingState`: come sopra
  - `customerEmail`
  - `billingPhone`
  - `shippingItems`: togliere la stringa "items:" in testa e togliere a partire dal primo carattere "|" fino a fine della stringa
  - `firstIssue`
  - `lastIssue`
  - `customerNote`
- esportazione della lista formattata nel file .csv:

  - qui si pone il problema di organizzare degli unit test su un metodo che scrive sul filesystem. Per farlo, dichiaro un'interfaccia *wrapper*, `ExportController`, con i metodi necessari (controllo presenza del file, scrittura del file, cancellazione del file) che invocherò come *mock* in questa fase e successivamente implementerò richiamando metodi delle API `java.nio`.
- comunicazione con le View

Proseguo con l'implementazione del Repository. Ho già aggiunto al POM le dipendenze per il Mongo Java Driver e per l'accesso ai log. Aggiungo la dipendenza per MongoDB Java Server (database in-memory). Al lancio del primo unit test ottengo un errore di inizializzazione:

```
java.lang.UnsupportedClassVersionError:
de/bwaldvogel/mongo/MongoBackend has been compiled by a more
recent version of the Java Runtime (class file version 61.0),
this version of the Java Runtime only recognizes class file
versions up to 52.0
```

Abbasso la versione del mongo-java-server in modo da farlo funzionare anche con Java 8 (52.0). Aggiungo `toString`, `equals` e `hashCode` al codice della classe Order per far funzionare i confronti con la mappatura dei documenti estratti dal database (v. `findAll()` nel Repository).

Concludo l'implementazione del Repository e verifico in locale che tutto funzioni:

```
mvn clean test
mvn clean verify -Pjacoco,mutation-testing
```

Controllo che il mutation test da Eclipse vada a buon fine e vedo che sopravvivono solo 4 mutanti nel Model.

Osservo i report di SonarQube e faccio un po' di pulizia. Nel Repository estraggo costanti laddove ho indicato con una stringa i riferimenti ai campi dei documenti (`ORDER_ID`, `ORDER_DATE`). Nel test del Repository SonarQube suggerisce, ad esempio, di limitare a uno i possibili metodi che possono sollevare un'eccezione nella lambda di `testFindByDateRangeWhenDateRangeIsIncorrect()`.

Nel frattempo modifico il workflow per Windows di GitHub Actions in modo che il trigger avvenga solo manualmente[1] (`on: workflow_dispatch`).

Comincio a implementare la View preparando i primi unit test. Aggiorno nel POM la dipendenza di AssertJ Core sostituendola con AssertJ Swing.

Tengo sotto controllo la code coverage. A buon punto della scrittura degli unit test raggiungo il 100% del codice del Controller e del Repository, il 99.4% della View (non è raggiunto il metodo `main()` della `DashboardSwingView`).

[1] https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows#workflow_dispatch

2.2 Mutation Testing Nelle fasi iniziali della scrittura dei test provo a vedere come si comporta il codice:

```
mvn clean test org.pitest:pitest-maven:mutationCoverage
```

(Aggiusto Pitclipse in modo da escludere il Model: al momento di eseguire i mutation test su `SubscriptionsController` innanzitutto prendo nota del *fully qualified name* di `Order.java`, quindi faccio clic col tasto destro su `SubscriptionsControllerTest.java` nel Package Explorer, quindi Run As > Run Configurations... e modifico la Run Configuration sotto PIT Mutation Test > SubscriptionsControllerTest (1) inserendo nel campo "Excluded classes" delle Preferences: `*Test, org.blefuscu.apt.subscriptions.model.Order`.

Soluzione più pratica: nella finestra `Preferences` (Ctrl+3 > Preferences) nelle impostazioni di Pitest inserisco direttamente `*Test, org.blefuscu.apt.subscriptions.model.*` nel campo Excluded classes.)

Configuro il plugin di Pitest in modo da includere i mutator del gruppo STRONGER.

Sorge un problema al momento di dover far girare Pitclipse sui test delle View, che prevedono l'uso di interfacce grafiche. Facendo riferimento ad alcune issue (qui[2], a cui fa riferimento questa PR[3]; qui[4], e qui[5]), modifico la Run Configuration in modo da inserire l'opzione `-Djava.awt.headless=false` (Run Configurations... > PIT Mutation Test > OrderSwingViewTest (1) > Arguments > VM Arguments > -Djava.awt.headless=false).

Per mantenere il plugin `pitest-maven` compatibile con Java 8, mantengo la versione cui si fa riferimento nel libro (1.5.2), poiché le successive richiedono almeno Java 11. Per ora mantengo nel POM l'opzione esplicita `-Djava.awt.headless=false` ed evito di attivare il mutation testing nel workflow per macOS.

Escludo dal Mutation Testing il codice relativo ai test sulle View, che sono generate in massima parte dal WindowBuilder e soggette alla risultanza di decine di mutanti superstiti, ad esempio:

```
...
1. removed call to org/blefuscu/apt/subscriptions/\
view/DashboardSwingView::setTitle -> SURVIVED
1. removed call to org/blefuscu/apt/subscriptions/\
view/DashboardSwingView::setBounds -> SURVIVED
...
```

2.3 Integration Tests Aggiungo al POM le dipendenze di Mongo Java API e di Logback, e aggiungo i plugin Build Helper (per gestire la cartella degli

[2] https://github.com/hcoles/pitest/issues/581
[3] https://github.com/hcoles/pitest/pull/601
[4] https://github.com/hcoles/pitest/issues/881
[5] https://github.com/hcoles/pitest/issues/1033

IT) e Failsafe (per eseguire gli IT). Faccio una bozza dell'implementazione di `OrderRepository`.

Sposto fuori dal profilo `docker` l'attivazione del plugin Docker.

Dopo aver scritto il `SubscriptionsController` e le View, mi concentro su uno dei pezzi mancanti: l'`ExportController`, che si occupa di scrivere su disco e di cancellare dal disco i file esportati a partire dalla `ListView`. Considero i relativi test come degli Integration Test.

Gli Integration Test sono quindi svolti a partire dai due controller (Export e Subscriptions). Eseguiti da Eclipse, dopo aver lanciato i container Docker di MongoDB (`./setup.sh`), danno buon esito. Provo con Maven da linea di comando ma ottengo errore:

```
[ERROR] Failed to execute goal io.fabric8:docker-maven-plugin:\
0.45.1:start (docker-start) on project apt-subscriptions: \
Execution docker-start of goal io.fabric8:docker-maven-plugin:\
0.45.1:start failed.: NullPointerException -> [Help 1]
org.apache.maven.lifecycle.LifecycleExecutionException: Failed to \
execute goal io.fabric8:docker-maven-plugin:0.45.1:start \
(docker-start) on project apt-subscriptions: Execution \
docker-start of goal io.fabric8:docker-maven-plugin:0.45.1:start \
failed.
```

Aggiorno la versione di `docker-maven-plugin` (da 0.45.1 a 0.48.0) e riprovo: funziona.

Scrivo una versione alternativa degli IT che fa uso della libreria Testcontainers, in modo da automatizzare la gestione dei container al lancio dei test da Eclipse. Aggiungo al POM un profilo che esclude l'esecuzione di queste versioni dei test.

2.4 E2E Test Una volta scritti i primi test E2E provo con: `mvn clean verify -Pjacoco,mutation-testing`, ma ottengo due warning da JaCoCo:

```
[WARNING] Rule violated for package org.blefuscu.apt.\
subscriptions.view: lines covered ratio is 0.99, but expected \
minimum is 1.00
[WARNING] Rule violated for package org.blefuscu.apt.\
subscriptions: lines covered ratio is 0.92, but expected minimum \
is 1.00
```

Porto al 100% la code coverage della View e aggiungo tra gli `<excludes>` di JaCoCo la classe `SubscriptionsSwingApp` (l'unica porzione di codice non raggiunta dai test è il ramo `catch` che lancia l'eccezione nella lambda della classe `call()`).

Con le impostazioni di default di `xvfb-run`, al momento di fare il test di `SubscriptionsSwingApp`, ottengo una failure dovuta al fatto che gli elementi

dell'interfaccia si trovano al di fuori dell'area del framebuffer virtuale (`-screen 0 640x480x8`). Riprovo con:

```
xvfb-run -a --server-args="-screen 0 1366x768x24" \
mvn clean verify
```

2.5 Continuous Integration con GitHub Actions Ancora nelle fasi iniziali della scrittura, Faccio un primo workflow su GitHub Actions coi plugin di JaCoCo e Pitest. Il repository è connesso a Coveralls; faccio un test in locale:

```
mvn coveralls:report -DrepoToken=<TOKEN>
```

Inserisco il token di Coveralls tra i *secrets* del repository e aggiorno di conseguenza il workflow finora sperimentato (`linux.yaml`).

A una prima build, il Quality Gate di SonarQube lamenta un eccesso di codice duplicato nel Model: lo escludo dall'analisi (nell'interfaccia web di SonarQube: Administration > General Settings > Analysis Scope > Coverage Exclusions e Duplication Exclusions).

Modifico il POM spostando in due profili ad hoc l'attivazione dei plugin di JaCoCo e Pitest. I *goal* che generano i report sono legati alla fase *verify*, per cui il comando diventa:

```
mvn clean verify -Pjacoco,mutation-testing
```

Verifico in locale la configurazione di SonarQube con GitHub Actions (finora era impostata la Automatic Analysis). La build va a buon fine ma, probabilmente per via della formattazione degli `additional-maven-args`, non esegue i profili e i goal richiesti per la versione 17 di Java:

```
Warning:  The requested profile "mutation-testing \
coveralls:report org.sonarsource.scanner.maven:\
sonar-maven-plugin:sonar -Dsonar.projectKey=\
mfflnz_apt-subscriptions" could not be activated \
because it does not exist.
```

(Infatti anche l'ultimo computo di Coveralls non ha ricevuto i dati dell'ultimo commit.)

Imposto Docker su GitHub Actions. Nel POM configuro il `docker-maven-plugin` in modo da attivarlo con il profilo `docker`. (Faccio un test in locale con `mvn verify -Pdocker`.) Attivo due container di MongoDB, uno dei quali con funzione di server, l'altro per importare da shell (`mongoimport`) alcuni documenti-campione su cui fare i test.

Installo il wrapper `xvfb-run`:

```
sudo pacman -S xorg-server-xvfb
```

e provo a lanciare Maven in locale:

```
xvfb-run mvn clean test
```

Uno dei test per la `OrderView` ha una failure che in apparenza sembra legata al timeout per lanciare il Controller:

```
[ERROR] org.blefuscu.apt.subscriptions.view.OrderSwingViewTest.\
testDeleteButtonShouldDelegateToSubscriptionsController -- \
Time elapsed: 26.06 s <<< FAILURE!
Wanted but not invoked:
subscriptionsController.deleteOrder(425);
-> at     org.blefuscu.apt.subscriptions.view.OrderSwingViewTest.\
testDeleteButtonShouldDelegateToSubscriptionsController\
(OrderSwingViewTest.java:188)
Actually, there were zero interactions with this mock.
```

I test *flaky* sono quelli che prevedono il clic su un pulsante individuato da una JButtonFixture in AssertJ-Swing:

```
window.button(JButtonMatcher.withText("Update")).click();
```

Osservo che raggiungo una stabilità migliore utilizzando il metodo `doClick()` di JButton (che pure prevede una forzatura rispetto alla lettura diretta del campo `btnUpdate`):

```
orderSwingView.getBtnUpdate().doClick();
```

Passo l'opzione `--auto-display` a `xvfb-run` e verifico:

```
xvfb-run --auto-display mvn clean verify
```

2.6 Code Quality e SonarQube È noto che le assertion di AssertJ Swing non vengono trattate da SonarQube (v. sez. 15.2.1 del libro) e vengono segnalate come *issue* bloccanti. Ad esempio:

```
window.button(JButtonMatcher.withText("Update")).requireEnabled();
```

A questa assertion aggiungo per completezza (a costo di ridondanza nel codice dei test) le analoghe assertion di AssertJ supportate da SonarQube, ad esempio:

```
assertTrue(window.button(JButtonMatcher.withText("Update"))\
.isEnabled());
```

Riduco laddove possibile la Cognitive Complexity indicata altrove da SonarQube.

Resta da rifattorizzare l'algoritmo che decide i parametri di ricerca della `SearchView` (implementato nel listener del pulsante "Search"), che allo stato attuale concorre a incrementare fino a 20 la Cognitive Complexity del costruttore. Il *technical debt* indicato è stimato in 10 minuti. Osservo che la complessità totale è data anche da:

- (+1) nel listener legato al campo `fromDateTextBox`;
- (+1) nel listener legato al campo `toDateTextBox`;

- (+2) nel metodo privato `removeWhitespacesOnly`, di supporto al listener legato al pulsante "Search";
- (+3) nel metodo privato `checkIfDatesAre...AndPerformSearch`, di supporto al listener legato al pulsante "Search";

Restano dunque 13 punti di cui è direttamente responsabile il codice del listener del pulsante "Search", che implementa un algoritmo di questo tipo:

- se entrambi i campi `fromDate` e `toDate` sono vuoti, restituisci tutti gli ordini presenti nella collection;
- se il solo campo `fromDate` è vuoto, restituisci tutti gli ordini dal più vecchio fino a `toDate`;
- se il solo campo `toDate` è vuoto, restituisci tutti gli ordini da `fromDate` alla data attuale.

Considerando anche che ulteriore complessità cognitiva è introdotta dai due gestori delle eccezioni che possono verificarsi al momento del *parse* delle due date, ritengo che l'implementazione sia sufficientemente comprensibile e decido di mantenerla così com'è.

SonarQube prescrive anche di rifattorizzare tre unit test della `SearchView` in modo da ottenere un solo test con parametri, ma preferisco lasciare i tre codici separati, ancorché ridondanti.

Altrettanto per quanto riguarda l'indicazione di rinominare le classi dei test E2E in modo da seguire la convenzione di Java rispetto alla nomenclatura: nell'interfaccia web di SonarQube indico come falso positivo la richiesta di rinominare le classi, appuntando la motivazione di un uso convenzionale del suffisso "E2E" per individuare facilmente i test.

---

3. Scelte di design e di implementazione

Per l'accesso al database seguo il pattern Repository (descritto in [Eva03] nel riferimento bibliografico del testo). Per la creazione degli oggetti mi rifaccio al pattern Builder ([GHJV95]).

(Il termine "Order" = "ordine" in questo contesto è da intendersi nel senso di ordine di acquisto di un prodotto.)

Schema del Model–view–presenter:

- **model**
  - Order
  - FormattedOrder
    - ordini opportunamente formattati con i soli dati rilevanti
- **repository**
  - OrderRepository
    - operazioni di ricerca, aggiornamento ed eliminazione

- **view**
  - DashboardView
    - SearchView
      - ricerca degli ordini per intervallo di date
    - ListView
      - elenco sintetico degli ordini
      - esportazione degli ordini
    - OrderView
      - vista dettagliata dell'ordine selezionato
      - aggiornamento dell'ordine selezionato
      - eliminazione dell'ordine selezionato
    - MessageView
      - un campo che mostra messaggi informativi o di errore
- **controller**
  - SubscriptionsController
    - operazioni sugli ordini (-> database)
  - ExportController
    - operazioni sugli export (-> filesystem)

3.1 UI La View è strutturata in quattro interfacce, come indicato nello schema MVC: `SearchView`, `ListView`, `OrderView` e `MessageView`, racchiuse in una `DashboardView`:

3.1.1 SearchView

- Due campi, "From" e "To", in cui è possibile specificare un intervallo di date entro cui effettuare la ricerca;
- un pulsante "Search"; se i due campi sono entrambi vuoti, la ricerca restituisce tutti gli ordini presenti nel database; se i campi sono specificati, la ricerca restituisce gli ordini dell'intervallo richiesto. I riferimenti agli ordini vengono visualizzati nella `ListView`;
- il pulsante "Search" è inizialmente attivo; finché uno o entrambi i campi non sono correttamente formattati (`YYYY-MM-DD`), si disattiva;
- se il solo campo "From" è compilato, il campo "To" viene impostato automaticamente alla data corrente;
- se il solo campo "To" è compilato, il campo "From" viene impostato automaticamente alla data del 1970-01-01;
- se le date specificate non sono ordinate in modo coerente, verrà visualizzato un opportuno messaggio di errore nel campo dedicato.

3.1.2 ListView

- Una lista in cui vengono riportati sinteticamente gli ordini individuati con la "Search";
- selezionando uno degli ordini, nella `OrderView` vengono visualizzati i corrispondenti dettagli;

- un pulsante “Export” che mostra un selettore di file tramite cui scegliere il percorso del file `.csv` che conterrà i campi rilevanti degli ordini al momento presenti nella lista; se la lista è vuota, il pulsante è disattivato.

3.1.3 OrderView

- Una maschera con form editabili corrispondenti ai campi di interesse specificati nel model `FormattedOrder` (id, data dell'ordine, data del pagamento, etc.);
- un pulsante “Update” che richiama l'opportuno metodo del Controller (eventualmente aggiornando la corrispondente vista sintetica della `ListView`) per aggiornare nel database il documento corrispondente all'ordine presente con i campi al momento specificati;
- un pulsante “Delete” che richiama l'opportuno metodo del Controller (cancellando contestualmente l'ordine presente dalla `ListView`) per rimuovere dal database il documento corrispondente all'ordine presente;
- le operazioni di “Update” e “Delete” mostrano eventuali messaggi rilevanti (di informazione o di errore) nel campo a essi dedicato.

3.1.4 MessageView

- Una maschera non editabile in cui compaiono, laddove rilevanti, messaggi informativi o di errore.

3.1.5 DashboardView

- Include le quattro view precedenti.

N.B.: Per testare le quattro view interne (Search, List, Order e Message), che implementano dei JPanel e non dei JFrame, uso la classe `Containers` di AssertJ-Swing[6]).

(Per evitare instabilità nei test (v. ad esempio gli unit test per la `ListSwingView` che si occupano del comportamento di pulsanti e finestre di dialogo) faccio uso del framework Awaitility e introduco un timeout di 5 secondi di attesa della pressione dei pulsanti.)

---

4. Sviluppo e test di alcune porzioni interessanti del codice

4.1 `SubscriptionsController` Il metodo `formatOrder` contiene il codice più rilevante rispetto all'uso principale che è previsto per l'applicazione, ovvero la semplificazione del formato dei documenti del database e la conseguente traduzione di oggetti di classe `Order` in oggetti di classe `FormattedOrder`. Man mano che le porzioni di codice relative alla formattazione dei vari campi venivano

[6] https://assertj-swing.readthedocs.io/en/latest/assertj-swing-advanced/#support-for-platform-specific-features

testate e scritte, la rifattorizzazione ha portato all'estrazione di 8 metodi privati dedicati alla formattazione di altrettanti campi.

Il campo "Shipping Items" degli ordini contiene, per design, un certo numero di caratteri speciali ("{", "}", "|") intorno a una parte della stringa a cui non sono interessato. Per escludere questa porzione nella formattazione dei dati, ricorro al raffronto con un'espressione regolare (`".[|{}]"`, cioè tutte le stringhe in cui occorre almeno una volta uno dei tre caratteri) che grazie all'analisi di SonarQube apprendo essere un *security hotspot* (come riferito qui[7]). Come soluzione semplice aggiungo alla regex un limite massimo alla lunghezza della stringa.

4.2 `ExportController` In una prima stesura, la logica che si occupava dell'esportazione su file della lista di ordini era prevista all'interno del `SubscriptionsController`. È stata la stessa tecnica del TDD a suggerire di separare le funzioni di raccordo con il database da quelle di gestione della scrittura su disco, conducendo a tempo debito - ossia al momento un cui si rendeva necessaria la scrittura di un file su disco - alla dichiarazione del *mock* della nuova classe `ExportController` e, solo successivamente, alla sua implementazione.

Fatta eccezione per alcune sporadiche licenze accordate nell'ottica di semplificare la comprensione dei test, ho cercato di mantenere l'ordine cronologico degli unit test man mano che venivano scritti, in modo da poter ricostruire agevolmente la logica della scrittura delle varie parti.

4.3 `OrderMongoRepository` Analogamente a quanto riportato nel `SubscriptionsController`, anche nel caso di questa implementazione del Repository una sezione del codice particolarmente delicata (ancorché accompagnata da poca logica) riguarda la conversione da documenti Bson a oggetti di classe `Order` e viceversa. Vista la relativamente grande quantità di coppie *field-value* nei documenti del nostro caso d'uso, per individuare i (facili) refusi è stato cruciale l'apporto del controllo della code coverage e del mutation testing.

4.4 View e UI La maggior parte degli unit test sulle UI riguarda l'attivazione o la disattivazione dei pulsanti in base ai contenuti di alcuni campi agganciati ai *listener*. Nelle View il tentativo è stato quello di ridurre il più possibile l'insorgenza di casi d'eccezione dovuti a valori critici negli input: ad esempio, la validazione dei contenuti in forma di data da inserire nei campi "From" e "To" nella `SearchSwingView`, prima ancora di essere delegata al Controller per le opportune verifiche e l'eventuale interrogazione del database, è sottoposta a un primo controllo sintattico, in modo da escludere stringhe diverse dal pattern "4 cifre, trattino, 2 cifre, trattino, 2 cifre" individuato dall'espressione

[7] https://rules.sonarsource.com/java/type/Security%20Hotspot/RSPEC-5852/

`"^\\d{4}-\\d{2}-\\d{2}$"`. (Ovviamente si potrebbe implementare una versione della Search View che utilizzi altri campi di input, come un selettore di date, e che conduca a una diversa manipolazione del formato dei dati, che per questo esempio manteniamo tra i due standard di `LocalDate` e di stringhe di pattern "yyyy-MM-dd".)

---

5. Problemi e soluzioni

5.1 Aggiustamenti con macOS Nelle fasi iniziali provo a impostare i workflow anche per macOS e Windows per verificare se per il momento tutto va a buon fine. Scopro che il JDK 8 non è disponibile per il runner `macos-13` e lo sostituisco con il JDK 11. Nella build per Java 17, allo step "Install Docker", vedo intanto che:

```
Installing QEMU
/opt/homebrew/bin/brew install qemu
Error: The operation was canceled.
```

Ripeto il job con messaggi di debug e vedo che:

```
Starting lima instance
Error: The process '/opt/homebrew/bin/limactl' \
failed with exit code 1
```

Provo a usare il runner `macos-latest` (che al momento equivale a `macos-15`) con la versione 4 della `setup-docker-action` (senza specifiche sulle versioni minori, come da esempio sul README[8] della action) e la variabile d'ambiente per Lima (di nuovo come da esempio, con specifiche sul numero di CPU e sulla quantità di memoria da emulare). Di nuovo ottengo lo stesso errore, e altrettanto succede con il workflow minimale riportato nella documentazione della action. Almeno due i punti problematici di questa configurazione: il fatto che `macos-latest` gira su architettura ARM (non supportata), e la versione 9.1.0 dell'emulatore QEMU (pare che vada tutto a buon fine con la versione 9.0.2). Faccio riferimento a questa issue[9] e a questa[10]. Faccio un tentativo e sembra funzionare con questa configurazione:

- runner `macos-13`
- QEMU 9.0.2
- `setup-docker-action@v3`

Provo a integrare questo espediente nel precedente workflow e ottengo questo errore:

```
Error:  Failed to execute goal io.fabric8:docker-maven-plugin:\
0.45.1:start (docker-start) on project apt-subscriptions: \
```

[8] https://github.com/docker/setup-docker-action
[9] https://github.com/docker/actions-toolkit/issues/455
[10] https://github.com/docker/setup-docker-action/issues/108

```
Execution docker-start of goal io.fabric8:docker-maven-plugin:\
0.45.1:start failed: No <dockerHost> given, no DOCKER_HOST \
environment variable, no read/writable '/var/run/docker.sock' \
or '//./pipe/docker_engine' and no external provider like Docker \
machine configured -> [Help 1]
```

Aggiungo la chiave `set-host` alla configurazione di `setup-docker-action`, che avevo dimenticato di specificare, e la imposto a `true`.

Con il commit 68cbfdb[11] la build su macOS si interrompe con l'errore di prima relativo a QEMU. Provo a modificare il workflow. Il runner `macos-latest` gira su architettura ARM, quindi non è supportato[12] da `setup-docker-actions`. Come non detto: mantengo `macos-13` nel workflow e riprovo con la soluzione[13] applicata finora. La build con Java 11 ha un buon esito.

Il Quality Gate di SonarQube segnala un Security Hotspot nel workflow, e prescrive di specificare il SHA del commit della `setup-docker-action` anziché il tag. Procedo di conseguenza.

Dal 4 dicembre 2025 GitHub Actions non mette più a disposizione[14] il runner `macos-13`, l'ultima versione con cui era possibile installare Docker con il procedimento sopra descritto. In alternativa uso la action Setup Docker on macOS[15] con `macos-15-intel` specificando nel workflow la variabile d'ambiente `DOCKER_HOST` (che punta al socket aperto da Colima, di cui fa uso la action).

5.2 Test manuale dell'applicazione Nella presente cartella si trovano due script, `setup.sh` e `teardown.sh`, utili per lanciare e interrompere i container di MongDB che servono all'applicazione.

In particolare, nel `setup.sh` sono sintetizzati i seguenti passaggi:

- Creo una rete per mettere in comunicazione i container che useremo:

  docker network create apt-network

- Lancio il container di MongoDB:

  docker run -d –name apt-mongo –network apt-network
  –publish 27017:27017 –rm mongo

- Nella cartella `src/main/resources` ho copiato un file `.csv` con alcuni dati di esempio da caricare su MongoDB (intestazione e informazioni sugli ordini) e lo importo con il tool `mongoimport`:

  docker run –network apt-network
  –volume "$PWD"/src/main/resources:/resources

[11] https://github.com/mfflnz/apt-subscriptions/commit/68cbfdb6d7e47e241ebb85acd180fae6f0933934
[12] https://github.com/docker/setup-docker-action?tab=readme-ov-file#about
[13] https://github.com/docker/setup-docker-action/issues/108#issuecomment-2393657360
[14] https://github.com/actions/runner-images/issues/13046
[15] https://github.com/marketplace/actions/setup-docker-on-macos

–rm mongo:latest mongoimport –host apt-mongo
–db=‘subscriptions’ –collection=‘orders’
–headerline –file=resources/sample-orders.csv –type=csv

Col comando sopra ottengo:

```
2025-12-16...    connected to: mongodb://apt-mongo/
2025-12-16...    5 document(s) imported successfully. \
0 document(s) failed to import.
```

- Nella shell di MongoDB faccio una verifica sui contenuti appena importati:

  docker run -it –network apt-network
  –rm mongo:latest mongosh –host apt-mongo

  test> use subscriptions subscriptions> db.orders.countDocuments() 5

5.3 Altre osservazioni A buon punto dello sviluppo, clono il repository su un'altra macchina e provo la build del codice da zero. Apro un nuovo branch del repository per gli opportuni aggiustamenti.

Una questione notevole riguarda l'incompatibilità della versione 5 di MongoDB con la vecchia CPU del laptop su cui tento la build, pertanto abbasso la versione alla 4.4.3.

Eseguendo `mvn clean verify` osservo che molti dei test svolti da AssertJ-Swing falliscono per qualche problema che sembra legato alla disposizione della tastiera (ad esempio, al posto del carattere `@` viene inserito il carattere `"`). (Noto che questo non si verifica né eseguendo i test tramite `xvfb-run` né testando a mano l'applicazione.)

In rete le uniche indicazioni sufficientemente autorevoli al riguardo si concentrano in alcune issue nel repository di FEST/AssertJ-Swing (ad esempio questa[16]). Provo a impostare l'opzione `-Dassertj.swing.keyboard.locale=it` (come argomento della JVM da linea di comando o inserendola nel file `.mvn/jvm.config`) ma ancora non ottengo il risultato sperato. Provo anche a inserire il testo utilizzando il metodo di AssertJ-Swing `pressAndReleaseKeys(int KeyEvent)` anziché `enterText(String text)`, e a inserire il carattere `@` utilizzando direttamente il codice esadecimale `\u0040`, ma di nuovo senza sucesso (ottengo in un caso il carattere `"` e nell'altro il carattere `q`). Osservo che l'errore nell'inserimento dei caratteri riguarda il metodo `enterText()` (che simula la digitazione da tastiera) ma non il metodo `setText()` della `TextInputFixture`, che imposta direttamente una stringa nel campo di testo e che inserisce i caratteri corretti. Valuto altre possibili soluzioni che prevedono l'uso di risorse dalla classe `InputContext`[17] e che ritengo troppo costose, e decido quindi di emendare il codice dei test sensibili aggiungendo un controllo ulteriore sui campi da validare: al loro interno inserisco (con `setText()`) il prefisso che contiene i caratteri non

[16] https://github.com/assertj/assertj-swing/issues/199
[17] https://docs.oracle.com/javase/8/docs/api/java/awt/im/InputContext.html

correttamente rilevati (ad esempio: “customer@”), e il suffisso con il metodo `enterText()`, che è necessario per attivare il listener collegato al campo da validare. Ritengo questa soluzione senz'altro non ottimale ma accettabile in questo contesto.