Report Lego Collector

Esame Advanced Programming Techniques (6 CFU)
CdL Magistrale in Ingegneria Informatica

https://github.com/AngeloDamante/lego-collector

Angelo D'Amante
7051006

Introduzione

Il progetto *lego collector* è stato sviluppato per l'esame di Advanced Programming Techniques tenuto dal Professor Lorenzo Bettini presso l'università degli studi di Firenze.

Con questo progetto, si vuole mostrare la propria competenza con la tecnica TDD e con le varie tecnologie mostrate durante il corso.

Applicazione Lego Collector

Il progetto consiste nel fornire all'utente finale un applicativo (molto semplice e basilare) per tenere traccia dei mattoncini Lego e dei kit in proprio possesso.

Gli obiettivi di questo applicativo si possono riassumere come segue:

- memorizzare mattoncini Lego;
- memorizzare Kit Lego;
- mostrare quali mattoncini appartengono a quali Kit;
- cercare un Lego tra quelli memorizzati che abbia una determinata caratteristica;
- aggiornare i dati di un Kit già memorizzato.

Come Avviare l'applicazione

I requisiti fondamentali per avviare l'applicazione sono Java 11, Maven e Docker. Una volta scaricata la repository, procediamo con il creare il *fatjar* dell'applicazione, successivamente avviare il container docker e infine avviare l'applicazione. I comandi risultano i seguenti,

```
mvn -f lego-collector/pom.xml -Dtests.ut.skip=true -Dtests.it.e2e.skip=true package
docker run -p 27017:27017 --rm mongo:4.4.3
java -jar lego-collector-1.0-SNAPSHOT-jar-with-dependencies.jar
```

È possibile inoltre, eseguire tutti i test di questo progetto.

```
# senza IT e E2E
mvn -f lego-collector/pom.xml -Dtests.it.e2e.skip=true clean verify

# senza UT
mvn -f lego-collector/pom.xml -Dtests.ut.skip=true clean verify

# tutti i test
mvn -f lego-collector/pom.xml clean verify
```

Tecnologie Utilizzate

È stata adottata la metodologia di sviluppo **TDD** (Test Driven Development). Brevemente elenchiamo le tecnologie utilizzate:

- **Git e Github**: è stato usato per mantenere il VCS del progetto.

- **Eclipse IDE**: versatile ed in particolare molto utile per la sua Integrazione con Maven attraverso plugin e file pom.xml.

- **Coveralls**: Servizio online per valutare il code coverage. È stato inserito un badge nel README del progetto.

- **Jacoco**: Servizio integrato per valutare il code coverage.
  ```
  mvn -f lego-collector/pom.xml clean verify -Pjacoco
  ```

- **SonarCloud**: Servizio online per il code analysis. Anche in questo caso, sono stati inseriti diversi badge.

- **Maven**: permette la gestione automatica delle dipendenze per semplificare il processo di sviluppo e garantire la coerenza delle librerie utilizzate utilizzando il file pom. In particolare notiamo:
  - L'utilizzo di *org.testcontainers* per effettuare i test di integrazione con docker.
  - Il plugin *assembly* per generare il fatjar.
  - il plugin *surefire* per gestire il report dei test eseguiti.

- **CI Github Actions**: È stato configurato il file *maven.yml* per poter per effettuare il build ad ogni push e pull request con Java 11. Inoltre, vengono generati i report opportuni.

- **PIT**: Per il mutation testing è stato configurato un profilo sul file *.pom* che genera i mutanti con regole di default. Sono state escluse le entità, l'interfaccia e il main.
  ```
  mvn -f lego-collector/pom.xml clean verify -Ppit
  ```

- **Swing**: Questo progetto vanta un'interfaccia (seppur minimale) per gestire i lego e i kit. Sviluppata con Swing.

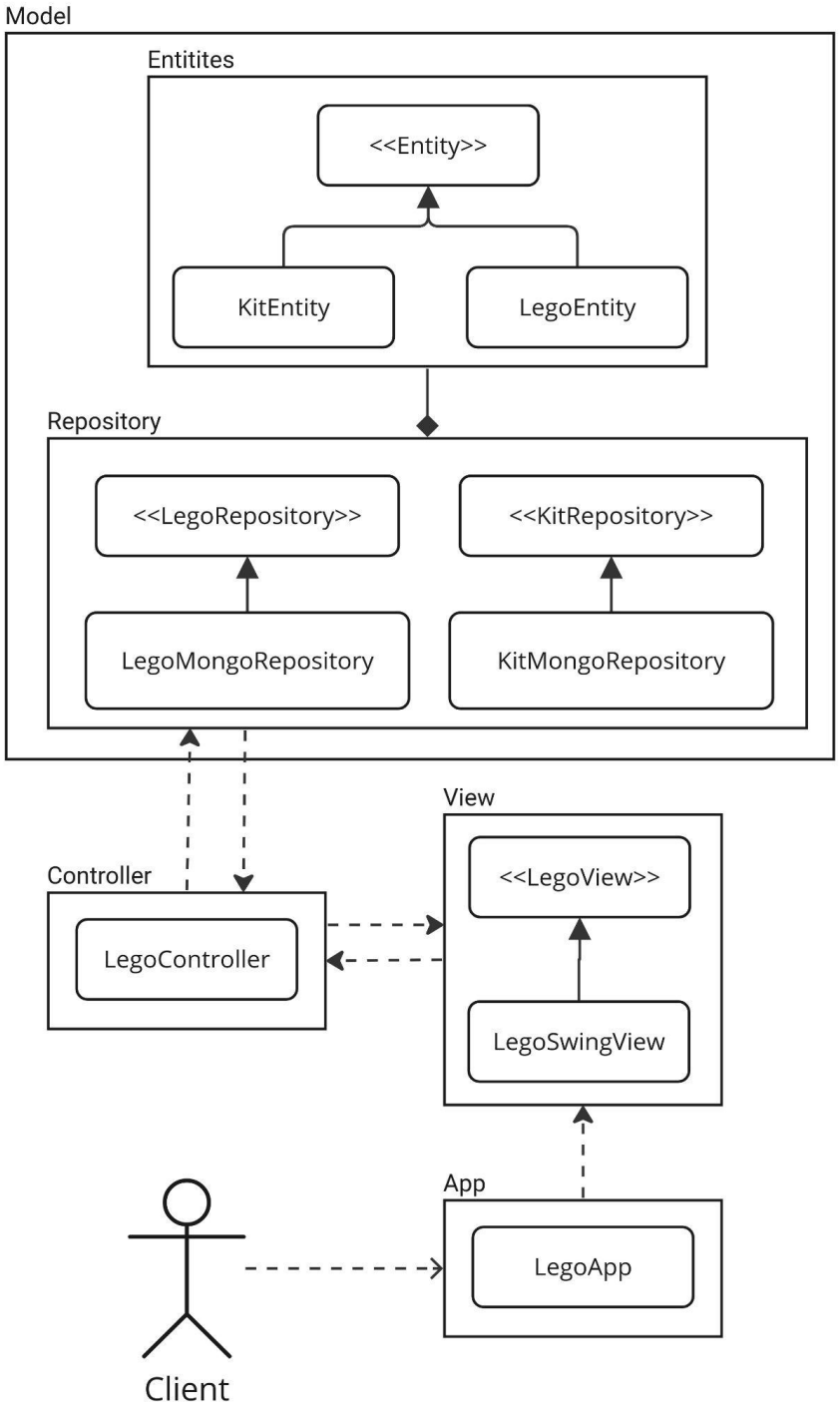
- **Picocli**: È inoltre possibile inserire delle opzioni per il fatjar da riga di comando.

```
--mongo-host
--mongo-port
--db-name
--db-collection-kits-name
--db-collection-legos-name
```

- **Mockito**: Per separare i vari componenti e testarli singolarmente.
- **Docker**: Questo progetto si appoggia ad un server mongoDB, lanciato da un container.

Architettura del Progetto

È stato scelto di implementare il pattern **MVC** (Model View Controller) con il pattern Repository come **DAL** (Data Access Layer) per implementare la logica di accesso al Database.



Layout delle Directory

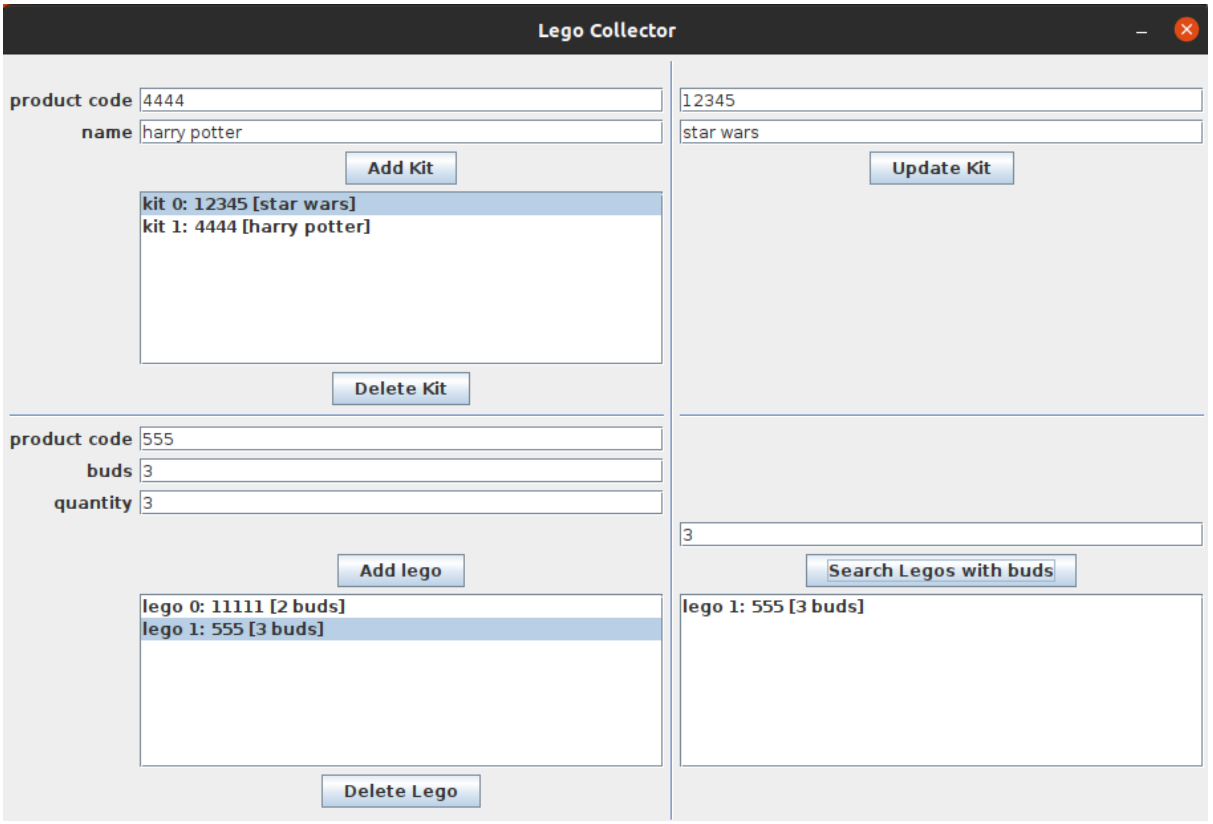
```
├── lego-collector
│   ├── pom.xml
│   └── src
│       ├── e2e
│       │   └── java/com/angelodamante
│       │       └── app
│       │           └── LegoAppE2E.java
│       ├── it
│       │   └── java/com/angelodamante
│       │       ├── model
│       │       │   └── repository
│       │       │       ├── KitMongoRepositoryTestcontainersIT.java
│       │       │       └── LegoMongoRepositoryTestcontainersIT.java
│       │       └── view
│       │           └── LegoSwingViewIT.java
│       ├── main
│       │   ├── java/com/angelodamante
│       │   │   ├── app
│       │   │   │   └── LegoApp.java
│       │   │   ├── controller
│       │   │   │   └── LegoController.java
│       │   │   ├── model
│       │   │   │   ├── entities
│       │   │   │   │   ├── Entity.java
│       │   │   │   │   ├── KitEntity.java
│       │   │   │   │   └── LegoEntity.java
│       │   │   │   └── repository
│       │   │   │       ├── KitMongoRepository.java
│       │   │   │       ├── KitRepository.java
│       │   │   │       ├── LegoMongoRepository.java
│       │   │   │       └── LegoRepository.java
│       │   │   └── view
│       │   │       ├── LegoSwingView.java
│       │   │       └── LegoView.java
│       │   └── resources
│       │       └── log4j2.xml
│       └── test
│           ├── java/com/angelodamante
│           │   ├── controller
│           │   │   └── LegoControllerTest.java
│           │   ├── model
│           │   │   └── repository
│           │   │       ├── KitMongoRepositoryTest.java
│           │   │       └── LegoMongoRepositoryTest.java
│           │   └── view
│           │       └── LegoSwingViewTest.java
│           └── resources
│               └── log4j2.xml
└── README.md
```

Package

Il progetto è stato suddiviso nel seguente modo:

- **app**: main dell'applicazione;
- **controller**: implementazione del controller;
- **model**:
    - entities: in questo caso lego e kit con i loro attributi;
    - repository: implementa la logica di accesso al database.
- **view**: dichiarazione dell'interfaccia e implementazione della GUI del progetto.

Operazioni Implementate



Gestione dei Kit

In alto a sinistra sono state implementate le funzioni di aggiunta ed eliminazione dei kit. I pulsanti vengono abilitati e disabilitati sotto opportune condizioni. L'id di ogni kit aggiunto è cumulativo, si è scelto di non dare all'utente l'opportunità di inserire due id uguali.

Gestione dei Lego

In basso a sinistra sono state implementate le funzioni di aggiunta e rimozione di mattoncini Lego. Per aggiungere un Lego occorre inserire tutti campi e selezionare il kit di appartenenza. I buds, sono i tondini sopra i mattoncini. Molti mattoncini, non presentano alcun bud, quindi lo 0 è permesso. Anche in questo caso, l'id è cumulativo e non è responsabilità dell'utente quella di inserirlo.

Aggiornamento Kit

È stata fornita all'utente la possibilità di modificare i kit memorizzati. Infatti è possibile cambiare il codice prodotto e il nome. Chiaramente, viene abilitata questa funzione solo se un kit è selezionato.

Visualizzazione Lego

Notiamo che selezionando un kit, la lista dei lego viene aggiornata istantaneamente, questo perchè viene fornita una vista di tutti i lego appartenenti al kit selezionato.

Ricerca Mattoncini

Una funzionalità molto utile per un collezionista che magari ha bisogno di sapere se ha o meno un certo mattoncino è quella di ricercare i mattoncini secondo i loro buds. Così è possibile sapere quali mattoncini si incastrano con quello desiderato.

Sviluppo

Seguendo il paradigma di programmazione TDD, ogni nuova feature è stata implementata seguendo una sorta di “catena” che consiste nei seguenti passi:

- Testing e Sviluppo nel repository (kit e lego) effettuando, quando serve il mocking del controller.
- Testing e Sviluppo del controller, effettuando, quando occorre, mocking per repository e interfaccia.
- Testing e Sviluppo delle funzionalità dell’interfaccia effettuando il mocking del controller.

Mutators

I mutanti generati, durante lo sviluppo sono i seguenti:

```
- Mutators
================================================================================
> org.pitest.mutationtest.engine.gregor.mutators.EmptyObjectReturnValsMutator
>> Generated 6 Killed 6 (100%)
> KILLED 6 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.VoidMethodCallMutator
>> Generated 20 Killed 20 (100%)
> KILLED 20 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.NullReturnValsMutator
>> Generated 6 Killed 6 (100%)
> KILLED 6 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.MathMutator
>> Generated 2 Killed 2 (100%)
> KILLED 2 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.NegateConditionalsMutator
>> Generated 6 Killed 6 (100%)
> KILLED 6 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
--------------------------------------------------------------------------------
================================================================================
>> Generated 40 mutations Killed 40 (100%)
>> Ran 69 tests (1.73 tests per mutation)
================================================================================
```

Test Eseguiti

Sono stati generati un totale di **89 test** tra test di unità, integration test e E2E test. Il seguente log, viene inserito per completezza.

```
-------------------------------> UT
[INFO] Running com.angelodamante.model.repository.KitMongoRepositoryTest
INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.807 s
[INFO] Running com.angelodamante.model.repository.LegoMongoRepositoryTest
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.137 s
[INFO] Running com.angelodamante.controller.LegoControllerTest
[INFO] Tests run: 13, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.277 s
[INFO] Running com.angelodamante.view.LegoSwingViewTest
[INFO] Tests run: 31, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 19.491 s
[INFO]
[INFO] Results:
[INFO] Tests run: 60, Failures: 0, Errors: 0, Skipped: 0

-------------------------------> IT
[INFO] Running com.angelodamante.model.repository.KitMongoRepositoryTestcontainersIT
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.356 s
[INFO] Running com.angelodamante.model.repository.LegoMongoRepositoryTestcontainersIT
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.807 s
[INFO] Running com.angelodamante.view.LegoSwingViewIT
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 11.514 s
[INFO]
[INFO] Results:
[INFO] Tests run: 18, Failures: 0, Errors: 0, Skipped: 0

-------------------------------> E2E
[INFO] Running com.angelodamante.app.LegoAppE2E
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 18.887 s
[INFO]
[INFO] Results:
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0
```

Testi e Riferimenti

Per questo progetto, è stato seguito il testo di riferimento del corso *Test-Driven Development, Build Automation, Continuous Integration with Java, Eclipse and friends (2021)*.

Il lavoro svolto si trova al link https://github.com/AngeloDamante/lego-collector.