Laurea Triennale in informatica-Università di Salerno
Corso di *Ingegneria del Software*- Prof.ssa F.Ferrucci

ODD
OBJECT DESIGN DOCUMENT

NewDM

Riferimento	
Versione	1.1
Data	21/01/2021
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Cirillo Franco Cirillo Luigi Fusco Ciro Aiello Vincenzo
Approvato da	

Revision History

Data	Versione	Cambiamenti	Autori
21/01/2021	1.0	Introduzione	Tutti
22/01/2021	1.1	Package, interfacce e class diagram	Ciro Fusco
03/02/2021	1.2	Revisione	Tutti

Sommario

1. Introduzione

1.1 Object design trade-offs

Nella fase dell'Object Design sorgono diversi compromessi di progettazione ed è necessario stabilire quali punti rispettare e quali rendere opzionali. Per quanto riguarda la realizzazione del sistema sono stati individuati i seguenti trade-off:

Buy vs Build: la necessità di sviluppare un'applicazione apre alla possibilità di attingere da una vastissima collezione di framework e librerie utili alla realizzazione del prodotto. La volontà di non voler reinventare nuovamente la ruota ed i tempi stringenti per la consegna fanno quindi valutare l'adozione di componenti off-the-shelf per il core dell'applicazione. La linea guida, in questo caso, è quindi quella di riutilizzare il più possibile le soluzioni offerte da terzi, scegliendo con attenzione, però, quali tra queste possano fare al caso nostro. (Scegliamo il buy)

Costi vs Estensibilità: Il sistema deve permettere l'estensibilità così da dare la possibilità al cliente di richiedere lo sviluppo di nuove funzionalità, a discapito dei costi in termini di ore lavoro. Questa scelta è dettata dalla previsione che il prodotto software in oggetto avrà successive release. (Scegliamo l'estensibilità)

Comprensibilità vs Tempo: Uno degli obiettivi dell'implementazione sarà quello di scrivere del codice che rispetti lo standard proposto da Google per la programmazione nel linguaggio Java, oltre all'uso di commenti sui metodi. Ciò favorisce anche la comprensibilità, agevolando il processo di mantenimento e di modifica del progetto anche per futuri sviluppatori che non hanno lavorato dall'inizio al progetto stesso. Questo vantaggio, tuttavia, comporta un incremento del tempo per lo sviluppo e la realizzazione dell'intero sistema, che però è ripagato da una maggiore manutenibilità e chiarezza dei contenuti implementativi. (Scegliamo Comprensibilità)

Tempo di risposta vs Affidabilità: Il sistema sarà implementato in modo tale da preferire l'affidabilità al tempo di risposta, garantendo un controllo più accurato dei dati in input a discapito del tempo di risposta del sistema. (Scegliamo Affidabilità)

1.1.1 Componenti off-the-shelf

Al fine di evitare che i tempi di sviluppo del software vengano allungati più del necessario, si è deciso di utilizzare diverse componenti già pronti.
Per la gestione persistente dei dati si è deciso di ricorrere a MySQL come DBMS poiché gli sviluppatori hanno già esperienza nell'utilizzo di questo software.
Per scambiare informazioni ed eseguire azioni tra il database e il software si utilizzerà JDBC che è un componente il cui utilizzo ed efficienza sul mercato è ampiamente documentato.
Per la creazione delle interfacce verrà utilizzato il software "SceneBuilder" il quale ci aiuterà nella creazione delle componenti grafiche in JavaFX.

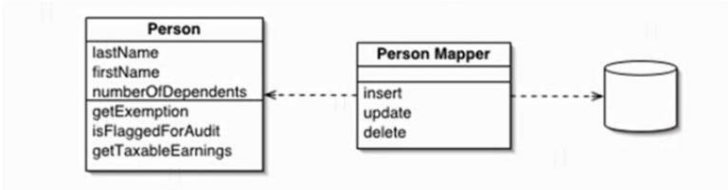
Tutte gli strumenti selezionati sono gratuiti ed open source, scelta in linea con i requisiti di costo.

1.1.2 Design Patterns

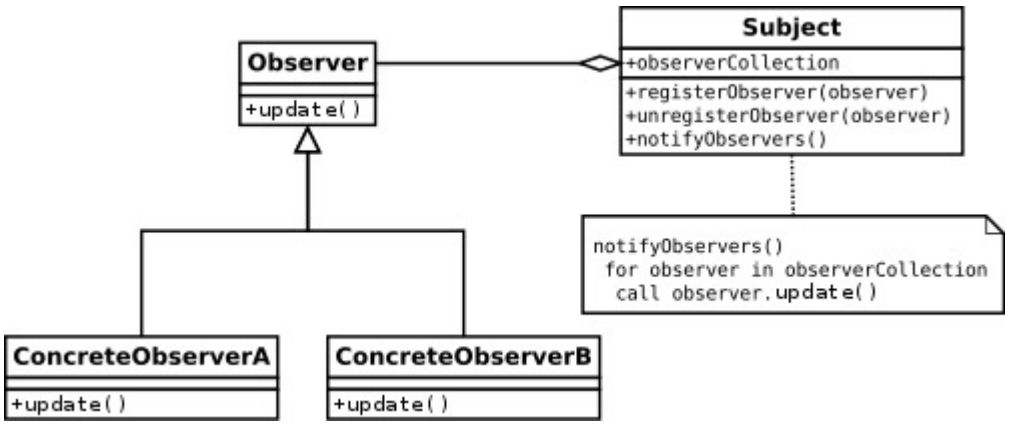
Per velocizzare lo sviluppo del sistema abbiamo utilizzato alcuni design patterns che risolvono problemi tipici anche del nostro progetto.

La scelta è ricaduta su tre Design Pattern, specificamente: Data Mapper, Singleton e Observer.

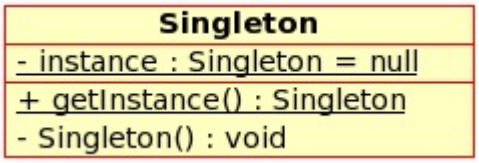
Il Data Mapper è un pattern che si pone l'obbiettivo di separare gli oggetti che rappresentano l'informazione persistente dalla logica relativa alla mappatura di questa informazione sul database sottostante. Ciò permette agli oggetti che risiedono in memoria di essere completamente disaccoppiati dal livello di persistenza (indipendentemente da quale esso sia, database relazionale, file o altro), favorendo la manutenibilità del sistema e lasciando la possibilità di variare, in futuri aggiornamenti, il tipo di persistenza con modifiche minimali al codice.



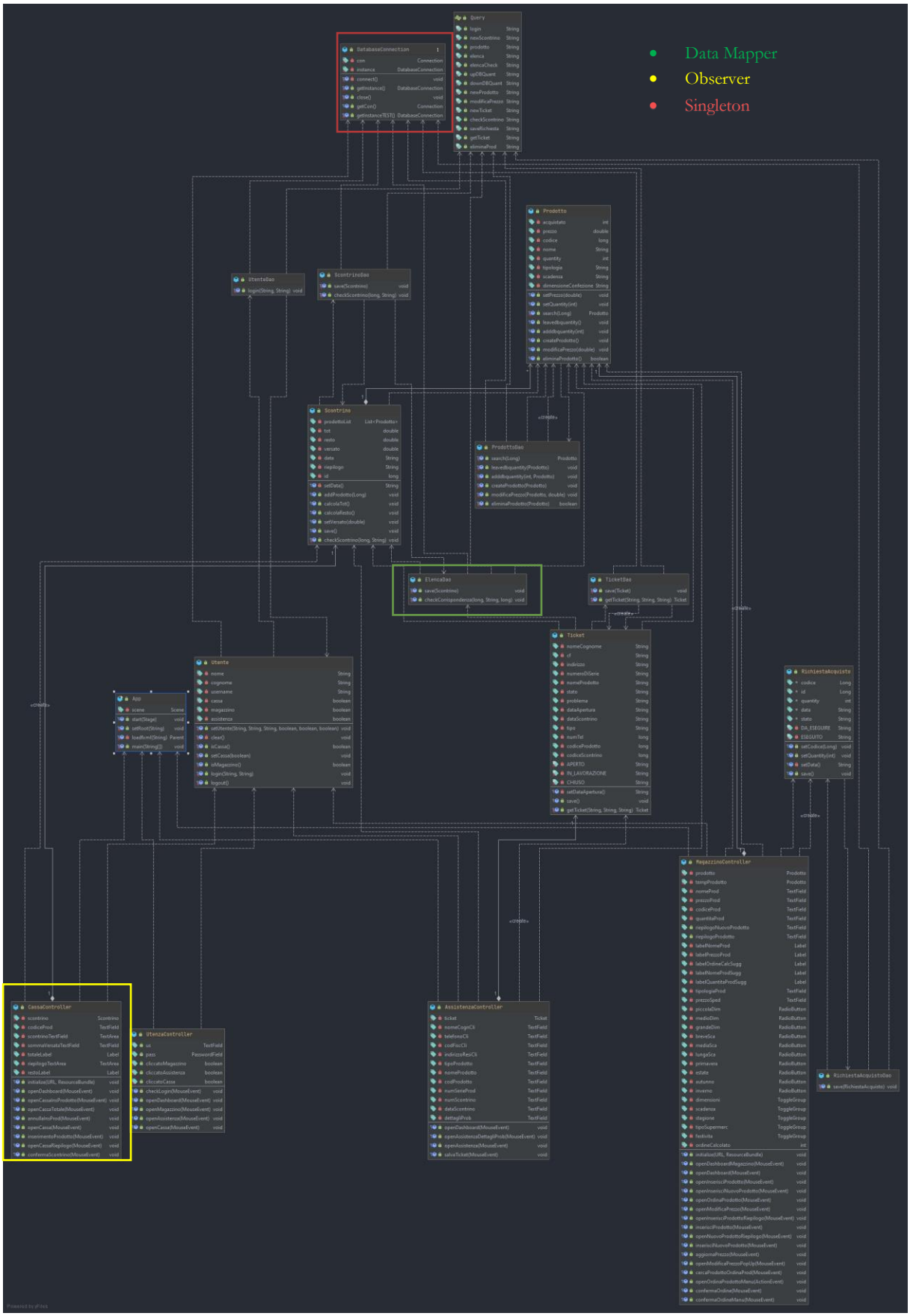
Poiché nel nostro software verranno utilizzati bottoni ed eventi, si utilizzerà il pattern Observer il quale ci permetterà di avere del codice che alla ricezione di un evento cambi lo stato di tutti gli oggetti interessati e invochi i metodi necessari al proseguo delle attività. Ogni componente che si interfaccia con la GUI necessita di un Observer per permettere la navigazione ed avviare l'esecuzione di routine prestabilite.



Per ultimo, verrà utilizzato il design pattern Singleton che ha lo scopo di garantire che venga creata una sola istanza di una componente. Verrà utilizzato all'interno del codice, per componenti come: la connessione al database, la cui multipla esistenza per un singolo Client sarebbe solo uno spreco di risorse per il server che occupa inutilmente una connessione.



Per ogni Pattern definito sopra, nell'immagine di seguito viene evidenziata una classe esempio che lo implementa.


Data Mapper
Observer
Singleton

1.2 Linea guida per la documentazione dell'interfaccia

È richiesto agli sviluppatori di seguire le seguenti linee guida al fine di essere consistenti nell'intero progetto e facilitare la comprensione delle funzionalità di ogni componente. Le linee da seguire fanno riferimento allo standard Google Java.

Nomenclatura delle componenti:

- Nomi delle classi
  - Ogni classe deve avere nome in CamelCase;
  - Ogni classe deve avere nome singolare;
  - Ogni classe che modella un'entità deve avere per nome un sostantivo che possa associarla alla corrispondente entità di dominio;
  - Ogni classe che realizza un form deve avere nome composto dal sostantivo che descrive il form seguito dal suffisso "Form";
  - Ogni classe che esegue la logica applicativa del form deve avere nome composto dal sostantivo che descrive l'azione seguito dal suffisso "Controller";
  - Ogni classe che esegue la logica di business interfacciandosi con il database deve avere nome composto dal sostantivo che descrive l'operazione e dal suffisso "DAO".
- Nomi dei metodi
  - Ogni metodo deve avere nome in camelCase (lower);
  - Ogni metodo deve segnalare un errore lanciando un'eccezione.
- Nomi delle variabili
  - Ogni variabile deve avere nome in camelCase (lower).
- Nomi delle eccezioni
  - Ogni eccezione deve avere nome esplicativo del problema segnalato.

Quando si codificano classi e interfacce Java, si devono usare le seguenti regole di formattazione

- Non inserire spazi tra il nome del metodo e la parentesi tonda "("che apre la lista dei parametri
- La parentesi graffa aperta "{"si trova alla fine della stessa linea dell'istruzione di dichiarazione
- La parentesi graffa chiusa "}" inizia su una nuova riga vuota allo stesso livello di indentazione del nome della classe o dell'interfaccia

Nel caso di istruzioni singole ogni linea dovrà contenere un'istruzione. Nel caso di istruzioni più complesse e articolate dovranno essere seguite le seguenti indicazioni

- Le istruzioni racchiuse all'interno di un blocco (esempio: for), devono essere indentate di un'unità all'interno dell'istruzione composta
- La parentesi di apertura del blocco deve trovarsi alla fine della riga dell'istruzione composta
- La parentesi di chiusura del blocco deve trovarsi allo stesso livello di indentazione dell'istruzione composta
- Le istruzioni composte formate da un'unica istruzione devono essere racchiuse da parentesi

Il Database SQL dovrà rispettare le seguenti regole:

- Nomi delle tabelle devono seguire le seguenti regole:

  - Devono essere costituiti da sole lettere maiuscole;
  - Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

- I nomi dei campi devono seguire le seguenti regole:
  - Devono essere costituiti da sole lettere maiuscole;
  - Se il nome è costituito da più parole, è previsto l'uso di underscore (_);
  - Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

Link Google Java: https://google.github.io/styleguide/javaguide.html#s4.6-whitespace

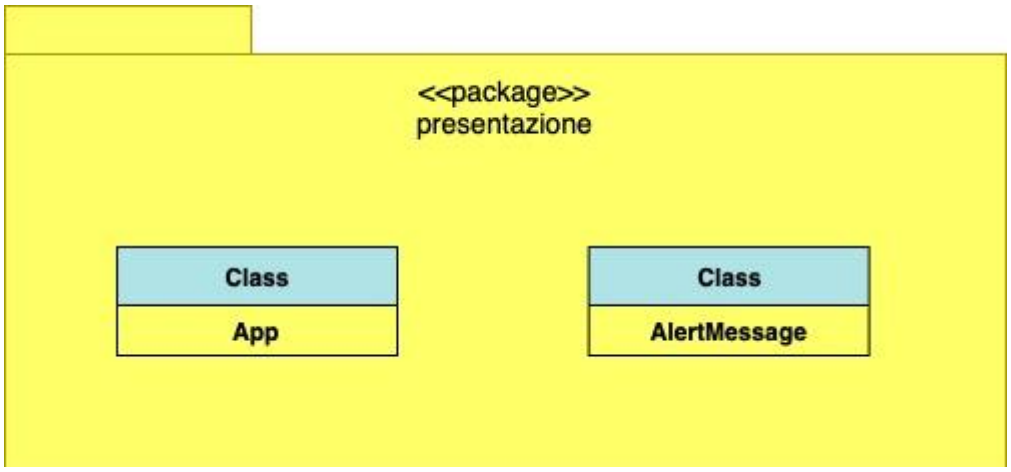
2. Packages

In questa sezione verrà presentata quella che è la divisione in sottosistemi e l'organizzazione del codice in files.

2.1 Divisione in pacchetti

Il nostro sistema si basa sull'architettura 3-layer così come descritto nel System Design Document ed è suddiviso nei layer: Presentazione, Business e Persistenza. Ognuno di questi layer avrà un proprio pacchetto e conterrà al suo interno i propri sottosistemi.

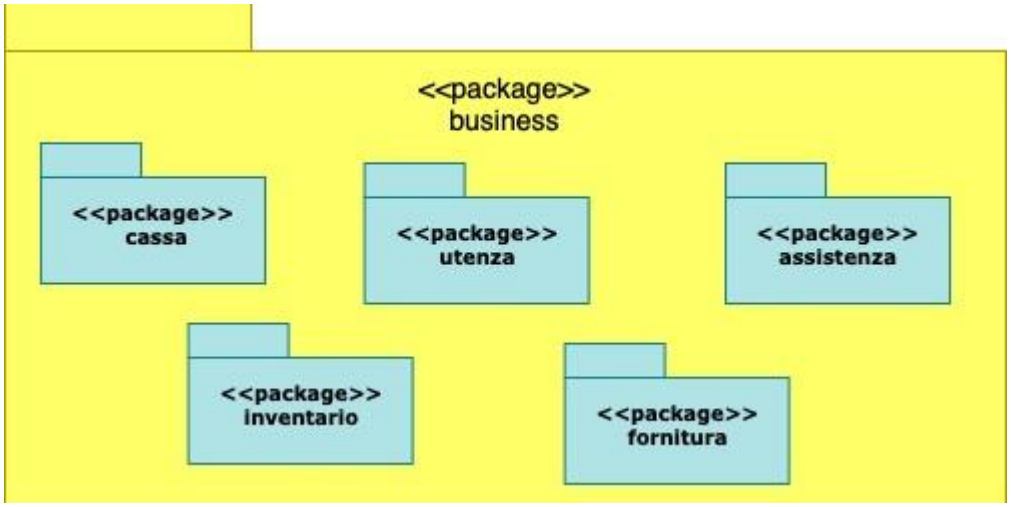
Questa divisione nasce dalla volontà di riunire nello stesso pacchetto le classi che sono legate da una forte dipendenza da un punto di vista semantico aumentando così la coesione dei sottosistemi.



Il pacchetto *presentazione* contiene le classi che hanno il compito di creare le grafiche del Software. In particolare, c'è una classe che effettua la creazione delle grafiche convertendo una versione xml-like delle interfacce, in maniera simile all'html in un browser.

Di seguito è riportato lo schema di decomposizione del pacchetto *presentazione* e delle sue dipendenze.

Nome pacchetto	presentazione
Descrizione	Contiene le classi necessarie alla creazione delle interfacce
Dipendenze	N\A



Le classi contenenti gli oggetti utili alla logica di business e gli Observer che gestiscono le interazioni con l'utente sono contenute nel pacchetto *business*.

Di seguito è riportato lo schema di decomposizione del pacchetto *business* e le sue dipendenze.

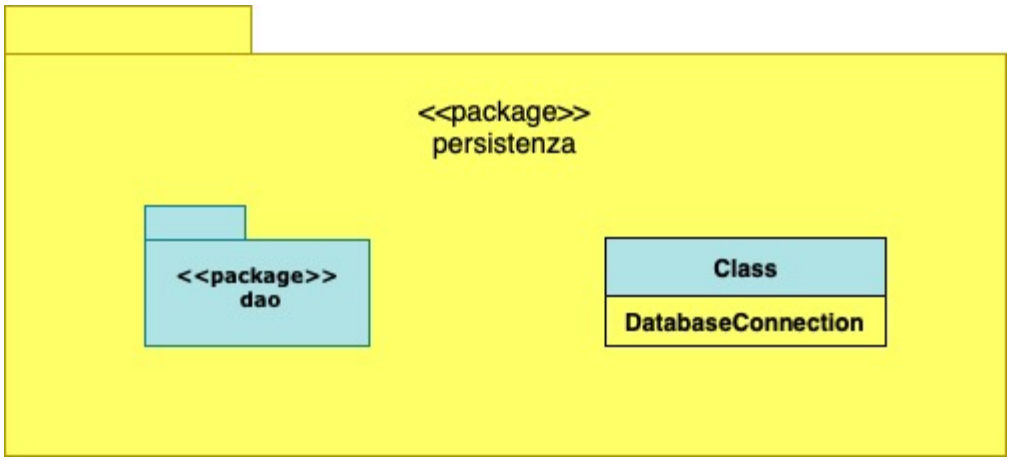
Nome pacchetto	business.utenza
Descrizione	Contiene le classi necessarie alla gestione dell'utente e dei suoi eventi eventi
Dipendenze	persistenza

Nome pacchetto	business.inventario
Descrizione	Contiene le classi necessarie alla gestione delle informazioni dei prodotti e dei suoi eventi eventi
Dipendenze	persistenza

Nome pacchetto	business.fornitura
Descrizione	Contiene le classi necessarie alla gestione delle richieste di fornitura ed eventi
Dipendenze	persistenza

Nome pacchetto	business.assistenza
Descrizione	Contiene le classi necessarie alla gestione delle richieste di assistenza e dei suoi eventi eventi
Dipendenze	persistenza, inventario, cassa

Nome pacchetto	business.cassa
Descrizione	Contiene le classi necessarie alla gestione degli scontrini e dei suoi eventi
Dipendenze	persistenza, inventario



Le classi contenenti i metodi utili alla gestione della persistenza sono contenute nel pacchetto *persistenza*.

Di seguito è riportato lo schema di decomposizione del pacchetto *persistenza* e le sue dipendenze.

Nome pacchetto	persistenza
Descrizione	Contiene le classi necessari alla gestione della connessione con il database
Dipendenze	N\A

Nome pacchetto	persistenza.dao
Descrizione	Contiene le classi necessari alla gestione dei dati persistenti
Dipendenze	persistenza

2.2 Organizzazione del codice in file

Come imposto da Java, ogni classe del sistema sarà collocata nel relativo file. Ognuno di essi sarà quindi collocato nella cartella dedicata (individuata in base al nome del pacchetto).

I nomi dei pacchetti saranno mappati nel rispettivo percorso src/main/java/… ad eccezione del pacchetto contenente la definizione delle interfacce utente che saranno collocate nella directory src/main/java/resources/presentazione.

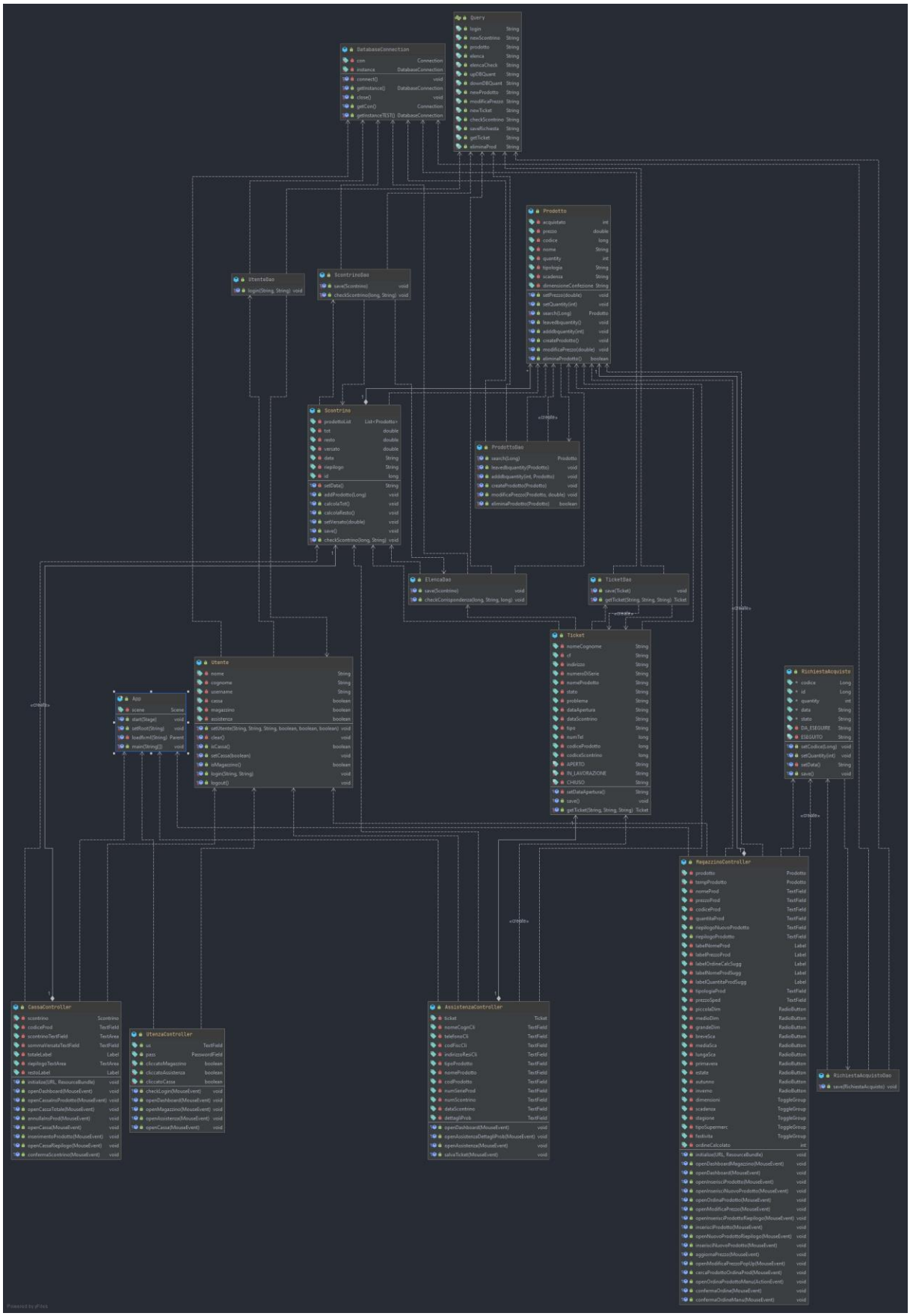
3. Interfacce delle classi

È possibile reperire la documentazione relativa all'interfaccia pubblica delle varie classi nei file Javadoc allegati presenti in /JAVA/javadoc/.

Per una migliore navigazione si consiglia di accedere alla documentazione tramite il file index.html.

4. Class Diagram

Una versione in formato .png del seguente Class Diagram è riportato in \NewDM\DocumentazioneODD\classDiagram


Query
DatabaseConnection
Prodotto
UtenteDao
ScontrinoDao
Scontrino
ProdottoDao
ElencoDao
TicketDao
Ticket
Utente
App
RichiestaAcquisto
CassaController
UtenzaController
AssistenzaController
MagazzinoController
RichiestaAcquistoDao

5. Glossario

Componenti off-the-shelf: prodotti software sviluppati da terzi e riutilizzabili.

Javadoc: Documentazione generata automaticamente a partire dai commenti scritti nelle classi Java.