Università Politecnica delle Marche – Facoltà di Ingegneria

INGEGNERIA INFORMATICA E DELL’AUTOMAZIONE

**APPLICAZIONE PER LA GESTIONE DI UN MAGAZZINO DI UN SUPERMERCATO**

**A.A. 2016 - 2017**

Identificativo: **OOP1617Gruppo06**

Iezzi Valerio (matr. 1078021)

Chelli Giacomo (matr.)

Genovesi Riccardo (matr.)

Luciani Mattia (matr.)

**SOMMARIO**

1. Introduzione 3
2. Analisi dei requisiti 4
   1. Diagramma dei casi d’uso 4
3. Analisi orientata ai dati 5
   1. Diagramma delle classi 6
   2. Diagramma delle sequenze 7
4. Progettazione del database 12
5. Struttura del progetto 14
   1. Struttura generale 14
   2. Contenuto della cartella “src” 15
6. IL design pattern MVC 16
   1. Struttura 16
   2. Implementazione 17
   3. View 18

6.3.1 Utilizzo di Swing 19

6.4 Model 24

6.5 Controller 27

6.5.1 Eventi e listener 27

1. Strumenti di programmazione utilizzati 33
   1. Ereditarietà
   2. Classi astratte e interfacce
   3. Polimorfismo
   4. Eccezioni
   5. Notazioni finali
2. **INTRODUZIONE**

Il progetto è stato sviluppato nell’ottica di fornire aiuto agli amministratori di supermercati, in particolare ad uno specifico amministratore del supermercato Coop, nella gestione del proprio magazzino e dei suoi dipendenti ed inoltre fornire al cliente del supermercato la possibilità di cercare il prodotto in vendita attraverso diversi filtri, visualizzare le offerte in corso e visionare la scaffalatura del prodotto stesso.

L’applicazione permette agli amministratori di inserire i propri prodotti nel database associato. Successivamente sarà possibile cercare i prodotti inseriti, utilizzando diversi filtri, oppure procedere alla modifica degli stessi. Inoltre sarà possibile inserire e modificare i dipendenti che forniscono servizio al supermercato associando loro il rispettivo turno lavorativo.

Il lettore viene invitato a consultare i manuali di installazione e utente, forniti a parte, al fine di:

* Il manuale di installazione permette di configurare la propria macchina sia per la consultazione e modifica del codice, che per la semplice esecuzione dell’applicazione;
* Il manuale utente mostra quelle che sono le funzionalità dell’applicazione e le possibili schermate in cui l’utilizzatore può incappare.

1. **ANALISI DEI REQUISITI**

Dopo aver studiato a fondo il dominio della nostra applicazione, abbiamo effettuato l’analisi dei requisiti utilizzando i diagrammi UML e in particolare il diagramma dei casi d’uso riportato in seguito.

* 1. Diagramma dei casi d’uso

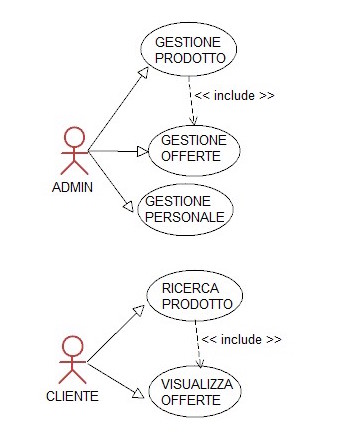


Fig. 1 – Diagramma dei casi d’uso

Come si evince dalla Fig. 1 i principali casi d’uso sono 5:

* Gestione prodotto: L’amministratore una volta che ha effettuato il login può inserire, modificare ed eliminare un prodotto;
* Gestione offerte: L’amministratore può inserire un prodotto in offerta indicandone il prezzo, e la data di inizio e fine offerta;
* Gestione del personale: L’amministratore può inserire un suo dipendente indicando i suoi dati anagrafici e la sua data di assunzione e licenziamento. E’ possibile inoltre associare ad ogni dipendete il proprio turno lavorativo;
* Ricerca prodotto: Permette all’utente di effettuare l’accesso all’applicazione come ospite e cercare il prodotto in database utilizzando diversi filtri (nome, settore) e visionare quindi la rispettiva scaffalatura.
* Visualizza offerte: Permette all’utente di visualizzare le offerte in ordine ascendente di data fine offerta.

1. **ANALISI ORIENTATA AI DATI**

Dal diagramma dei d’uso, passando attraverso numerosi raffinamenti successivi, si è arrivati alla definizione prima delle classi necessarie (Fig.2) e in seguito al diagramma delle stesse (Fig.3).

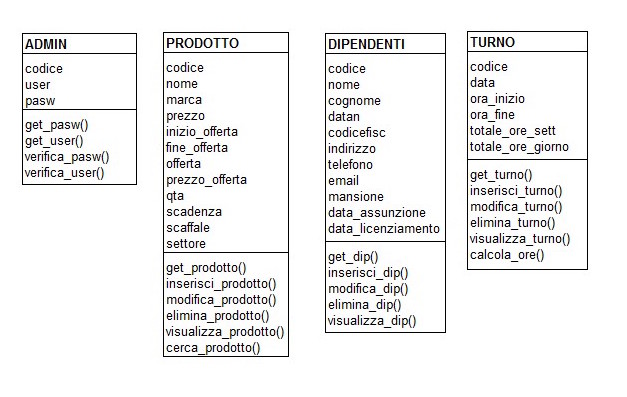


Fig. 2 – Classi

Le classi identificate sono:

* Admin:
* Prodotto:
* Dipendenti:
* Turno:
  1. Diagramma delle classi

Le classi sopra descritte sono collegate tra di loro e formano così il seguente diagramma:

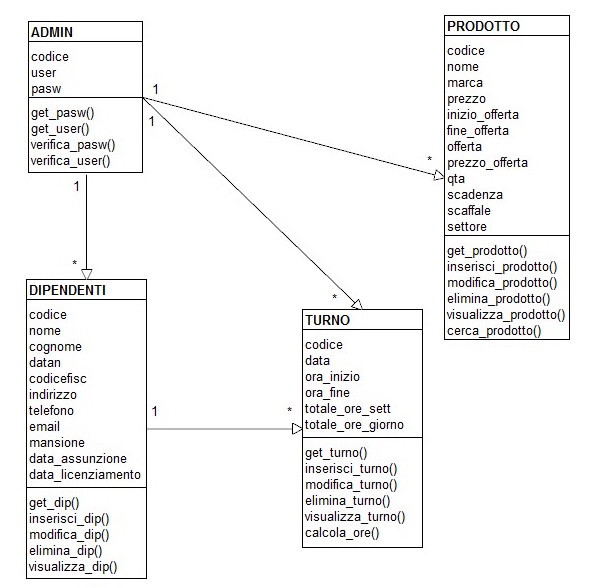


Fig. 3 – Diagramma delle classi

* 1. Diagramma delle sequenze

Al fine di comprende meglio il funzionamento dell’applicazione di seguito sono riportati i diagrammi delle sequenze relative all’amministratore e al cliente in funzione delle azioni che essi svolgono.

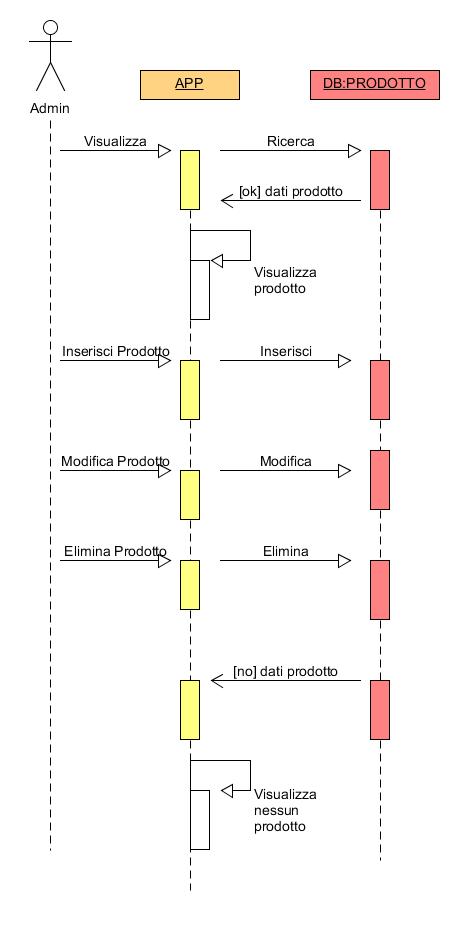


Fig. 4 – Diagramma delle sequenze Admin/Prodotto

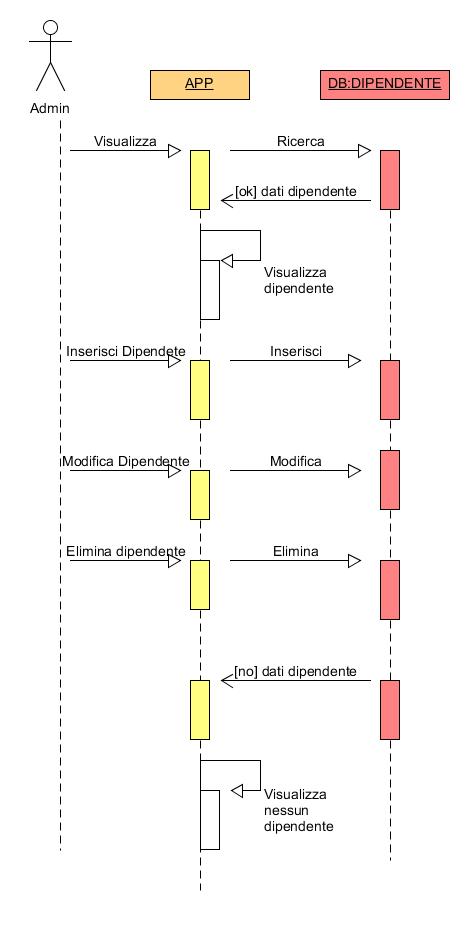


Fig. 5 Diagramma delle sequenze Admin/Dipendente

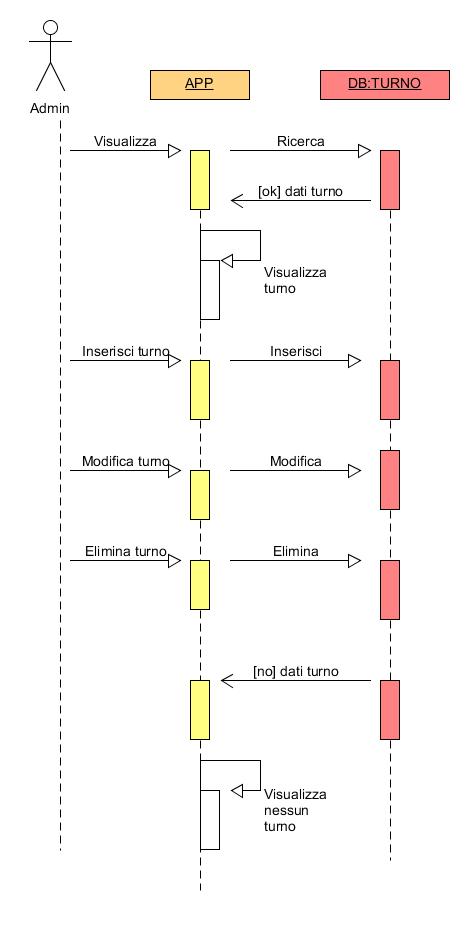


Fig. 6 – Diagramma delle sequenze Admin/Turno

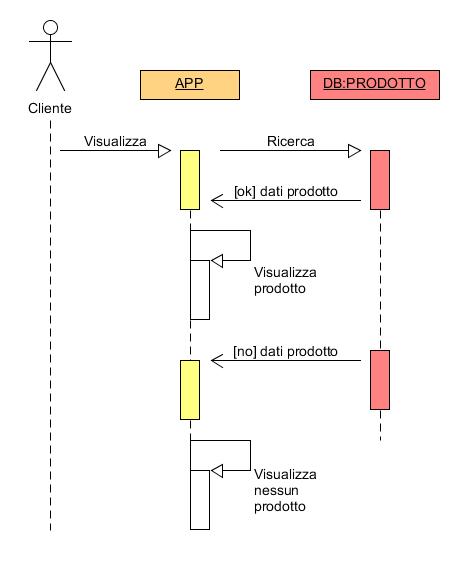


Fig. 7 – Diagramma delle sequenze Cliente/Prodotto

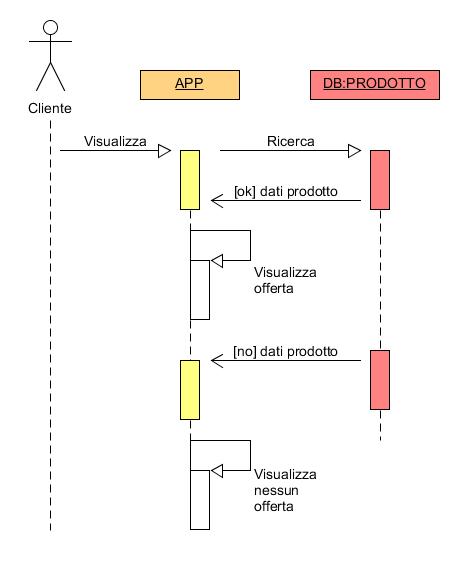


Fig. 8 – Diagramma delle sequenze Cliente/Prodotto in offerta

1. **PROGGETTAZIONE DATABASE**

Per l’applicazione è stato utilizzato un database di tipo relazionale, dove cioè i dati vengono raccolti in tabelle. L’ DBMS ( DataBase Management System) utilizzato è MySQL.

Di seguito è mostrato lo scherma E/R:

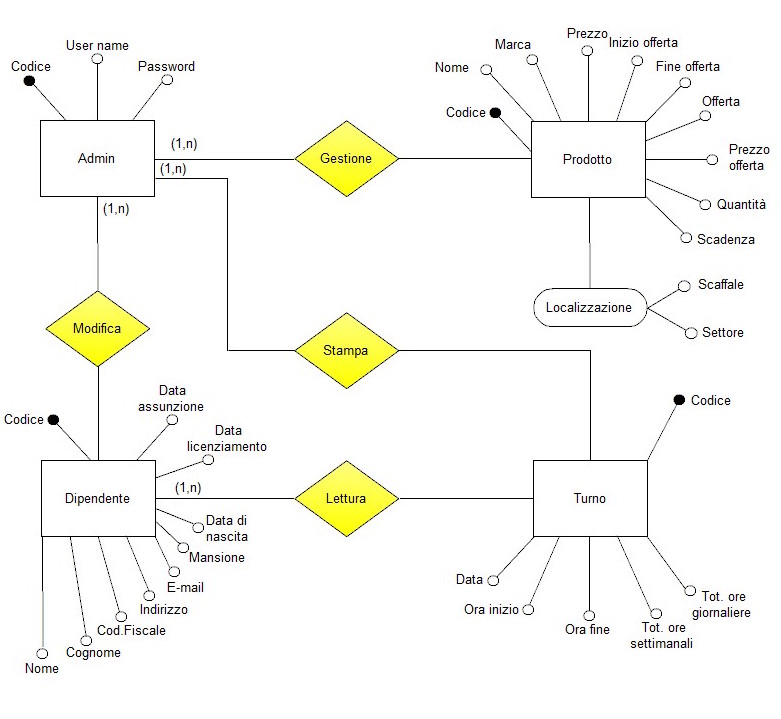


Fig. 9 – Schema E/R

Come si evince della Fig. 9, la base di dati si compone di 4 tabelle e le relative associazioni tra di loro.

La tabella *Admin* con i suoi attributi ci permette di memorizzare gli amministratori.

NB: Nella nostra applicazione l’amministratore è uno solo ed i suoi attributi sono già memorizzati e non sarà quindi necessario inserirlo/registrarsi. Attraverso l’username e la password forniti si potrà accedere alla schermata principale dell’applicazione.

Per conoscere le modalità di accesso, vi riportiamo alla consultazione del manuale utente.

La tabella *Prodotto* consente all’amministratore, una volta effettuato l’accesso, di andare a inserire e quindi memorizzare i propri prodotti con tutti gli attributi sopra descritti. In particolare si offre la possibilità di memorizzare il prodotto in offerta, con il prezzo offerta e la data di inizio e fine offerta. Degno di nota è anche la possibilità di poter memorizzare la locazione del prodotto intesa come scaffalatura a settore.

Inoltre sarà possibile modificare o eliminare i prodotti da questa tabella.

La tabella *Dipendete* consente di andare a memorizzare tutti i dati anagrafici dei dipendenti e la relativa data di assunzione e licenziamento e la rispettiva mansione che esso ricompre all’interno del supermercato.

Nella tabella *Turno* sarà possibile memorizzare i turni scelti dall’amministratore, ovvero i turni possono essere personalizzati a seconda delle diverse esigenze. Sarà possibile scegliere un nome del turno e successivamente, per ogni giorno della settimana, sarà possibile selezione mattina, pomeriggio o riposo. Per mattina è inteso il turno dalle 08:00 alle 13:00, per pomeriggio dalle 15:00 alle 20:00. Andando ad inserire il turno personalizzato verranno quindi calcolate le ore settimanali relative a quel turno, che poi sarà possibile associare ai dipendenti.

1. **STRUTTURA DEL PROGETTO**
   1. Struttura generale

All’interno della cartella del progetto OOP1617 vi sono 4 sottocartelle (Fig. 10 di seguito).

La cartella *img* contiene tutte le immagini utilizzate e richiamate dall’applicazione, come ad esempio la mappa del supermercato che indica la scaffalatura o le immagini che vogliamo attribuire ai prodotti.

La cartella *Resources* contiene le librerie che andremo in fase di configurazione ad importare. In particolare esse sono: rs2xml.jar e mysql-connector-java-5.1.40-bin.jar.

La prima è una libreria che può essere usata per fare il set di risultati di una query e fornirlo in ingresso per il modello di tabella.

La seconda è una libreria con la quale MySQL fornisce la connettività per le applicazioni client sviluppate nel linguaggio di programmazione Java con MySQL Connector / J ,un driver che implementa l'API Java Database Connectivity (JDBC).

La cartella *doc* contiene il JavaDoc esportato alla fine del progetto. Ed infine la cartella *src* è il cuore del progetto.

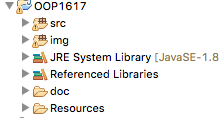


Fig. 10 – Struttura del progetto

* 1. Contenuto della cartella “src”

Veniamo ora al cuore del programma: tutto il codice sorgente si trova all’interno di src. Il progetto viene strutturato sul pattern MVC di cui andremo ad approfondire nel capitolo successivo. Il questo paragrafo andremo ad esaminare più attentamente gli altri package di cui è comporta la cartella src.

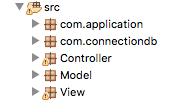


Fig. 11 – Contenuto della cartella “src”

Come mostrato in Fig. 11 la cartella src è comporta da 5 package:

* com.application: contiente la classe necessaria per l’avvio dell’applicazione.
* com.connectiondb: contiene la classe necessaria per stabilire la connessione al database (nel manuale di installazione, verrà spiegato come configurarlo).
* Controller: contiene le classi controller dell’applicazione.
* Model: contiene le classi model dell’applicazione.
* View: contiene le classi view dell’applicazione.

1. **IL DESIGN PATTERN MVC**

Il Model-View-Controller è un design pattern, cioè un’architettura progettuale che descrive un problema, una soluzione e le relative conseguenze. L’obbiettivo principale di un pattern è rendere la soluzione indipendente rispetto alla sua implementazione in modo tale da rendere il codice riutilizzabile e facilmente modificabile.

Sostanzialmente la “filosofia” dell’MVC è quella di progettare un software in modo da dividere l’interfaccia utente (GUI) dalla parte che rappresenta i dati e le regole che governano le modalità con cui questi vengono acceduti e dalla parte che gestisce le richieste dell’utente e le azioni che devono essere intraprese per modificare i dati stessi.

In generale l’applicazione deve separare i componenti software che implementano il modello delle funzionalità di business (Model), dai componenti che implementano la logica di presentazione (View) e di controllo che utilizzano tali funzionalità (Controller).

* 1. Struttura

Analizziamo nello specifico gli elementi fondamentali che compongono la struttura progettuale MVC.

* Model o Modello: Rappresenta l’elemento responsabile della gestione dei dati. Legge e memorizza i dati usati dall’applicazione. Per lo sviluppo del Model si utilizzano delle tipiche tecniche di progettazione le quali consentono di ottenere componenti software che rappresentano al meglio i concetti importanti del mondo reale.
* View o Vista: Responsabile della presentazione dei dati gestiti dal model in un formato specifico. La logica di visualizzazione dei dati viene gestita solo e solamente dalla View, ciò implica che questa deve fondamentalmente gestire la costruzione dell’interfaccia grafica (GUI) che rappresenta il mezzo mediante il quale gli utenti si interfacciano con il sistema. L’esecuzione dei processi richiesti dall’utente viene infine delegata al controller dopo la cattura degli input e la scelta delle eventuali schermate da presentare.
* Controller: Costituisce l’organo che permette al Model e alla View di lavorare insieme. Ha la responsabilità di trasformare le interazioni dell’utente con la View in azioni eseguite dal Model. Il controller non rappresenta un semplice “ponte” tra View e Model, ma gestendo gli input dell’utente, selezionando le schermate della View da presentare e richiamando i processi da eseguire sul Model implementa interamente la logica di controllo del software.
  1. Implementazione

La nostra applicazione è stata costruita sulla base dell’architettura progettuale MVC.

Diversi sono i package realizzati in cui sono raggruppate classi che hanno determinate caratteristiche e precisi compiti. Andremo in seguito ad analizzare la directory contenenti il package MVC.

Abbiamo già parlato dei package specifici nel capitolo precedente; andiamo ad analizzare la cartella src contenente il codice sorgente dell’applicazione. Notiamo che per semplicità i package gestiti con il pattern MVC sono stati inclusi direttamente nella directory src in modo tale da facilitare l’importazione delle classi dove questa è richiesta.

Analizziamo ora i singoli elementi fondamentali del MVC.

* 1. View

Abbiamo definito il componente View come l’organo responsabile della gestione della visualizzazione dei dati. La Vista consiste nella creazione di una interfaccia grafica con cui l’utente può interagire. Per questo motivo è costituita da una serie di schermate che verranno visualizzate all’occorrenza. La View è costituita da 5 classi che permettono all’utente di interagire e visualizzare le varie informazioni.

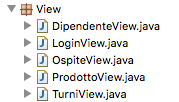


Fig. 12 – Package View

La classe LoginView prevede la creazione dell’interfaccia per permette all’utente di fare l’accesso all’applicazione o di accedere come utente ospite.

La classe OspiteView genera l’interfaccia che permette all’utente ospite di cercare i prodotti presenti nel supermercato e di visionare le caratteristiche e in particolare la scaffalatura e i prodotti in offerta.

La classe ProdottoView mi genera l’interfaccia iniziale una volta effettuato l’accesso come amministratore. Questa interfaccia ci permetterà di cercare i prodotti in magazzino, inserirli, modificarli o eliminarli attraverso i rispettivi pannelli. Visualizzerà anche la possibilità di accedere alle altre due classi (TurniView e DipendentiView) tramite due appositi bottoni.

La classe TurniView genera l’interfaccia con la quale l’amministratore del supermercato potrà inserire il turno lavorativo personalizzandolo per le proprie esigenze; modificare i turni già creati ed eliminarli. Inoltre sarà possibile visualizzare per ogni turno, quali dipendenti lo svolgono.

La classe DipendeteView possiede l’interfaccia dove sarà possibile oltre che a visualizzare i dipendenti per nome o per mansione, ci permetterà anche di inserire i dipendenti associandogli un turno, modificare i dati del dipendente o il turno lavorativo ed eliminare il dipendente.

6.3.1 Utilizzo di Swing

Componenti

Le classi usate per la gestione della GUI sono Swing e Event. Della seconda diremo soltanto che in ogni classe della View vi è un metodo creato per aggiungere ai componenti grafici degli ascoltatori.

La classe Swing contiene tutti i componenti necessari per la costruzione di una finestra (o un’interfaccia più in generale). Andiamo ad esaminarne alcuni usando come esempio la classe ProdottoView.

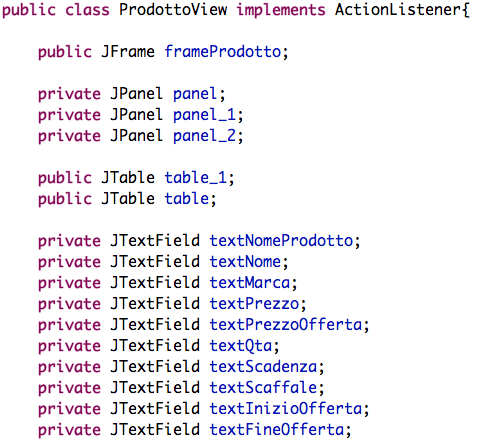


Fig. 13 – ProdottoView

La classe JPanel del package Swing permette di creare una “cornice” che viene poi impostata come “contenuta” nel frame e rende possibile l’aggiunta di altri componenti al suo interno.

Infatti il nostro frameProdotto è costituito da 3 JPanel: panel, panel\_1, panel\_2 rispettivamente Cerca Prodotto, Inserisci Prodotto e Modifica Prodotto. Ognuno di questi panel contiene al suo interno i componenti necessari. In Fig. 13 possiamo notare oggetti come che permettono l’interazione con l’utente come le tabelle o i campi di testo. La classe JTable ci permette di aggiungere al panel una tabella. La classe JTextField ci permette invece di inserire per esempio nel panel\_1, specifico per l’inserimento di un prodotto, i campi di testo necessari.

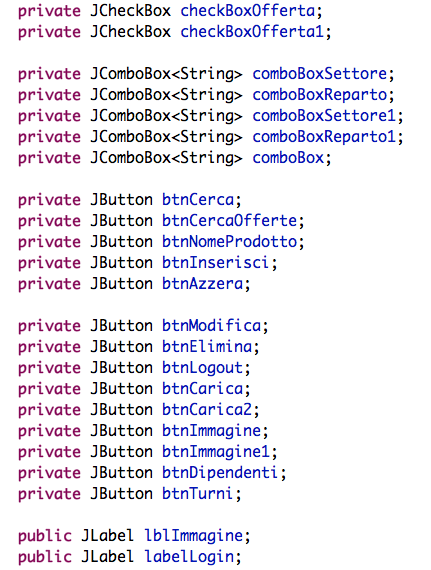


Fig. 14 – ProdottoView

Passiamo ora ai pulsanti. La classe JButton ci permette di aggiungere al panel dei “bottoni” ai quali verrà associata un’azione da compiere nel caso in cui l’utente li prema. JCheckBox permettere di aggiungere una casella di spunta, mentre JComboBox è la classe relativa al menù a tendina. Infine JLabel corrisponde alla classe che permette di aggiungere una semplice scritta.

Inizializzazione dei componenti

Per quanto riguarda l’inizializzazione dei componenti dell’interfaccia grafica andremo a visualizzare i metodi di alcuni componenti di esempio della classe ProdottoView.

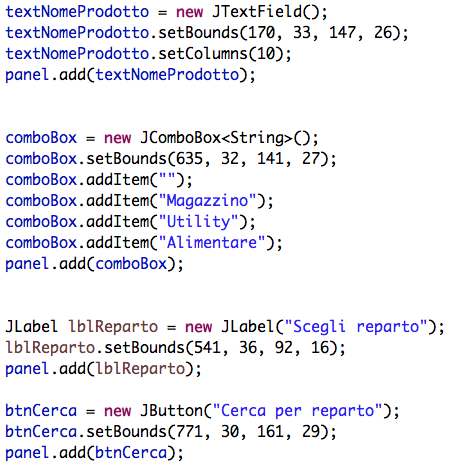


Fig. 15 – Inizializzazione dei componenti

Come mostrato in Fig. 15 il metodo setBounds (int x, int y, int width, int heigth) permette di settare la posizione e la grandezza dell’oggetto. Il metodo addItem per la comboBox permette di inserire un elemento nel menù a tendina. Il metodo add attribuito al panel permette di aggiungere l’oggetto passato come parametro al panel corrispondente.

Metodi presenti

All’interno delle classi appartenenti al package View sono presenti dei metodi che vengono principalmente richiamati dalle classi corrispondenti nei rispettivi controller.

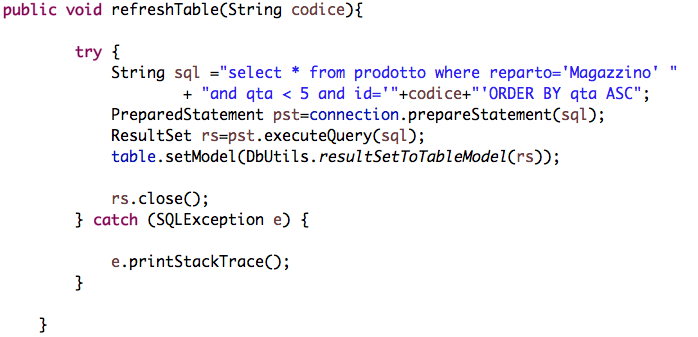


Fig. 16 – Metodo “mostraprodotti”

Il metodo refreshTable, presente nella classe ProdottoView, permette tramite connessione al db di eseguire una query che seleziona tutti i prodotti del reparto Magazzino e con quantità minore di cinque, ordinandoli per quantità ascendente. Il risultato della query viene poi visualizzato nella tabella tabel.

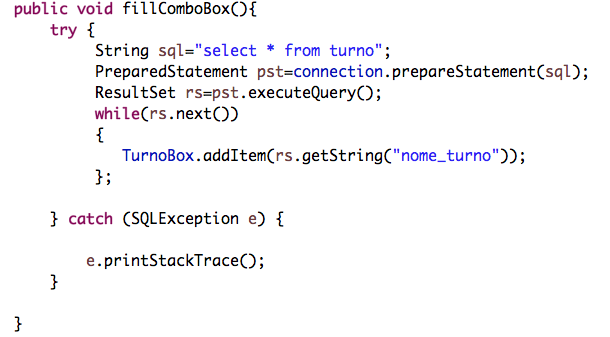


Fig. 17 – Metodo “fillComboBox”

Il metodo fillComboBox, presente sulla classe DipendeteView, ci permette tramite preventiva connessione al db, tramite una query di selezione tutti i campi dalla tabella turno; andando adesso a scorrere il risultato della query con il metodo next, eseguiamo il metodo addItem all’oggetto TurnoBox, aggiungendo cosi elementi al menù a tendina. Gli elementi, di volta in volta, saranno costituiti dalla stringa contenuta nel db nel campo nome\_turno.

Vi sono poi i metodi per l’aggiunta degli ascoltatori ai componenti di swing; questi ultimi prevedono come parametro un oggetto della classe ActionListener e lo aggiungono al pulsante in questione.

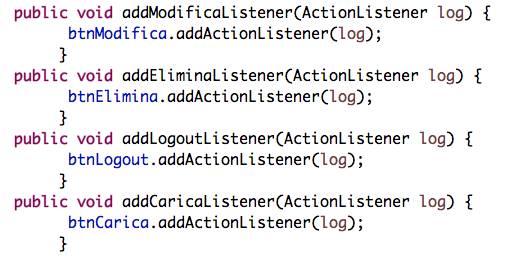


Fig. 18 – Listener

* 1. Model

Il Modello è la componente che si preoccupa della gestione e dell’organizzazione dei dati utili. Le classi definite nel package Model sono rispettivamente 5 come mostrato nella Fig. 19 sottostante.

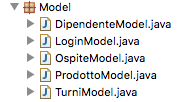


Fig. 19 – Model

Attributi e costruttori

Si analizzerà nel seguito la classe TurniModel.java a titolo esemplificativo: le altre implementazioni risultano analoghe.

Passiamo alla descrizione e analisi degli attributi e costruttori. (Fig. 20)

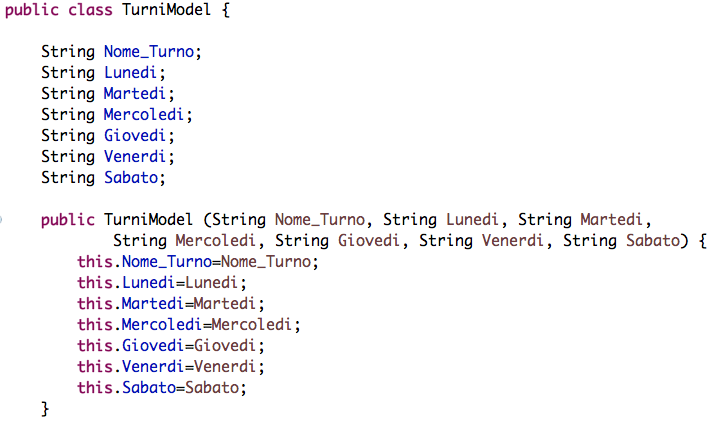


Fig. 20 – TurniModel

Gli attributi mostrati sono dichiarate String e rappresentano il nome che attribuiremo al turno, e per ogni giorno della settimana sarà poi possibile associare una stringa tra: Mattina, Pomeriggio e Riposo.

Il costruttore prende come parametri sette Stringhe e si occupa di restituire il valore contenuto dai paramenti che gli passiamo.

Metodi presenti

Oltre ai metodi get che verranno elencati in seguito, è presente un metodo proprio della classe TurniModel.java.

Il metodo Calcola\_ore() è un metodo pubblico che restituisce un intero, ovvero il numero totale di ore settimanali da cui è composto quel rispettivo turno.

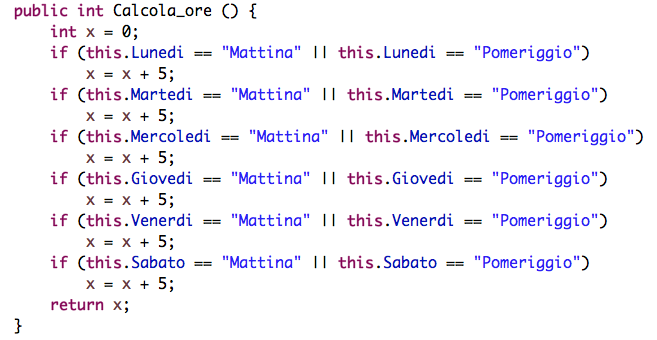


Fig. 21 – Calcola\_ore

Ogni turno che sia mattina o pomeriggio è composto da 5 ore. Si procede con una serie di if che vanno a verificare per ogni giorno della settimana quale turno è stato scelto. Se corrisponde a Mattina o Pomeriggio, il numero di ore viene incrementato di 5, altrimenti si procede al successivo giorno della settimana. Infine sarà possibile quindi sapere il numero ore settimanali di cui sarà composto quel turno.

Vediamo di seguito nella Fig. 22 i metodi get utilizzati in questa specifica classe TurniModel.java.

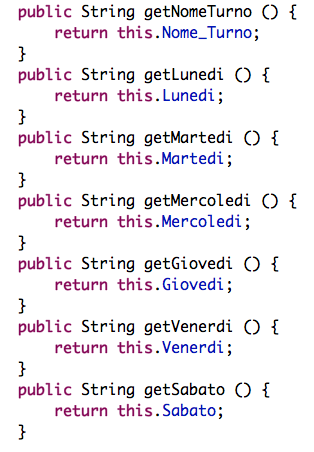


Fig. 22 – Metodi get

* 1. Controller

Il Controller è l’elemento che permette di unire View e Model trasformando gli input e l’interazione dell’utente con la GUI in operazioni da eseguire in quest’ultimo.

Le classi definite nel package Controller permettono di gestire i vari Eventi che si verificano nelle schermate della View, associando ad ogni azione un’operazione.

Un clic su un bottone o in una casella di testo viene gestito interamente dal controller; questo, in base all’opzione selezionata, si preoccupa di creare determinati oggetti e richiamare opportuni metodi che permettono di aggiornare il Model o visualizzare altre tipologie di schermate.

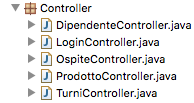


Fig. 23 – Controller

6.5.1 Eventi e Listener

Il controller è uno degli elementi essenziali dell’architettura progettuale MVC e viene definito come il componente che fa da tramite tra View e Model. La responsabilità fondamentale di questo organo è la gestione degli input e delle varie azioni che l’utente compio con l’interfaccia grafica, producendo un’adeguata risposta.

Il controller si preoccupa della gestione delle operazioni da eseguire in base al tipo di interazione da parte dell’utente con la GUI. Nel momento in cui l’utente conferma un determinato form, o aggiunge una particolare Attività è il controller che fa in modo di visualizzare delle “finestre aggiuntive” o di modificare i dati utili interfacciandosi con il Model. Una delle caratteristiche fondamentali e comuni di tutte le classi appartenenti alla sezione Controller è la gestione degli Eventi tramite particolari elementi Java, i Listener.

Con il termine Evento si indica un oggetto indicante un’azione, come la pressione di un bottone, la chiusura di una finestra, o qualunque altra cosa che ci si aspetta produca risposta. Un evento generato da un oggetto viene detto “scatenato” o “emesso”. Un oggetto che è in grado di generare un evento deve avere uno o più oggetti ascoltatori detti Listeners. Nel momento in cui un oggetto genera un evento, quest’ultimo viene immediatamente inviato all’oggetto ascoltatore che, attraverso opportuni metodi detti “gestori di eventi”, produce una risposta adeguata. Questo tipo di programmazione viene definita “ad eventi” e nella maggior parte dei casi l’ordine con cui saranno eseguite le varie azioni non è mai noto a priori ma dipende dall’utente.

Attributi e costruttori: ProdottoController

In seguito sarà analizzata una determinata classe del package controller (ProdottoController), questo poiché tutte le classi inerenti alla sezione Controller sono simili, ciò che cambia solo le operazioni che vengono eseguite nel momento in cui si verifica un evento. Per esempio nel momento in cui viene premuto il bottone “btnInserisciProdotto” viene memorizzato nel database un nuovo prodotto, mentre se viene premuto il bottone “btnModificaProdotto” va ad aggiornare il prodotto nel database con i campi modificati nella form apposita. L’evento gestito è lo stesso (bottone premuto) ma le operazioni che vengono eseguite sono diverse.

Tutte le classi del package Controller importano determinate librerie di Java che permettono l’uso di particolari strumenti come eventi, ascoltatori, action, componenti di Swing… inoltre è presente l’importazione di package inerenti al progetto che consentono di utilizzare oggetti del Model e della View (Fig. 24).

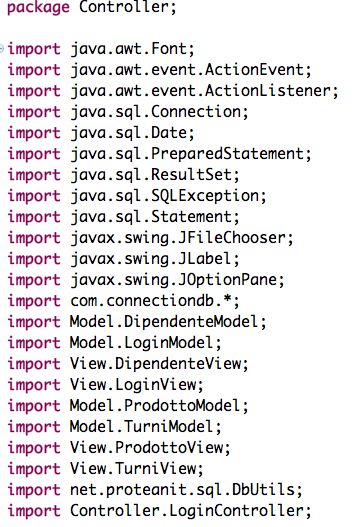


Fig. 24 – Importazione librerie

Passiamo alla descrizione degli attributi e del metodo costruttore.

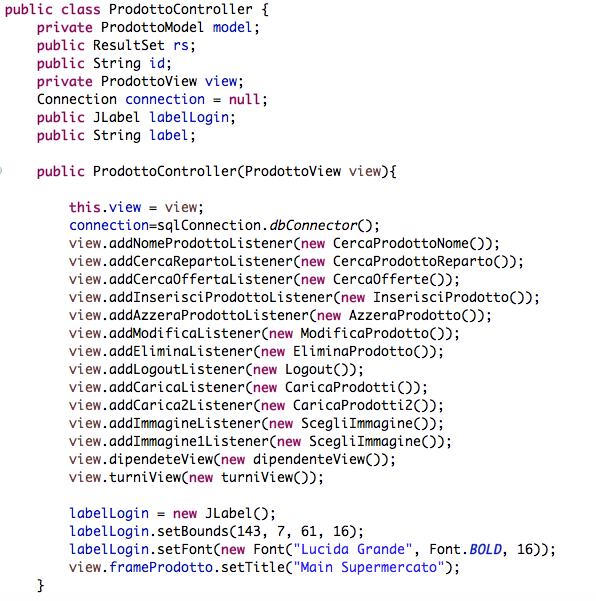


Fig. 25 – Attributi e metodo costruttore

Un aspetto comune a tutte le classi del package controller sono alcune caratteristiche degli attributi. Poiché l’oggetto di tipo ProdottoController gestisce gli eventi del corrispettivo oggetto del package View, viene definito un attributo di tipo ProdottoView settato nel costruttore passato come argomento.

Action-ProdottoController

Il metodo addNomeProdottoListener(ActionListener) viene richiamato tramite l’oggetto view di tipo ProdottoView passandogli come paramento un’istanza della classe CercaProdottoNome. Questo metodo, richiamato nel costruttore di ProdottoController, permette di associare un ascoltatore ad un particolare componente dell’oggetto view di tipo ProdottoView. In generale abbiamo un componente (un bottone in questo caso) del form di tipo ProdottoView a cui viene assegnato un ascoltatore, per questo motivo andiamo ad analizzare la classe interna CercaProdottoNome, la quale gestisce gli eventi che possono verificarsi (Fig. 26).

Tutte le classi del package Controller hanno classi e metodi differenti che permetto di gestire ulteriori eventi riferiti a particolari finestre dell’interfaccia grafica. Abbiamo deciso di descrivere la classe ProdottoController perché si tratta di una delle classi più complete e rappresentative riguardo il concetto di Controllo del paradigma MVC. Verranno esaminate solo alcune action, data la forte somiglianza delle stesse.

La classe CercaProdottoNome implementa l’interfaccia ActionListener. Ciò significa che la classe CercaProdottoNome deve ridefinire il metodo actionPerformed(ActionEvent). Questo è il gestore dell’evento “pressione del bottone” e definisce le azioni da eseguire nel momento in cui l’evento si verifica. In questo caso possiamo notare dall’implementazione del metodo che l’azione da eseguire al momento in cui si preme il rispettivo bottone è quella di recuperare il nome del prodotto dal model, successivamente eseguire la query che seleziona tutti i prodotti aventi quel nome (appartenenti all’utente loggato, andando a fare un controllo sull’id) ordinando il risultato per prezzo.

Il risultato della query viene visualizzato sull’oggetto table della view corrispondente.

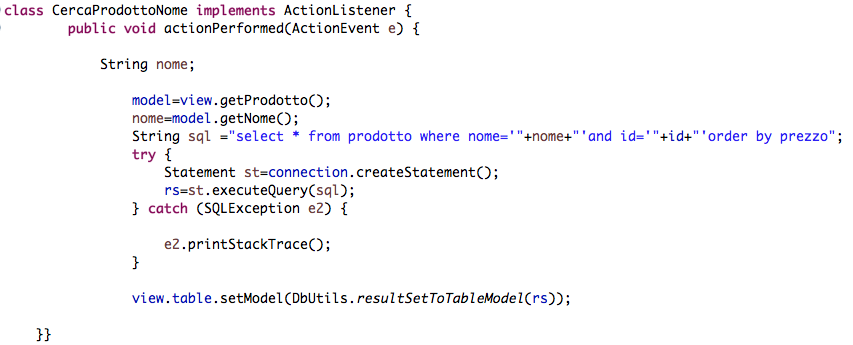


Fig. 26 – CercaProdottoNome

Passiamo adesso alla descrizione di una classe che consente di gestire l’evento “pressione di un bottone”. Il metodo implementato permette di eliminare il prodotto selezionato sulla table, dal database.

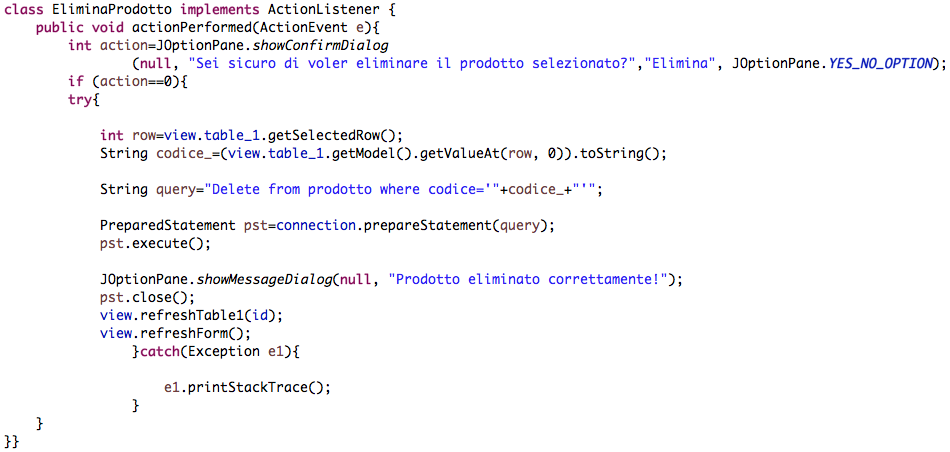


Fig. 27 – EliminaProdotto

La classe EliminaProdotto, come nella descrizione della classe precedente, anche questa implementa l’ActionListener. Questa volta però il metodo “gestore dell’evento” compie un’azione differente. Inizialmente verifico se la risposta alla domanda “Sei sicuro di di voler eliminare il prodotto selezionato?” sia positiva. Se positiva il valore action sarà uguale a 0. Verifico con un if che action sia zero e procedo. Nella variabile int row viene salvato il numero della riga selezionata sull’oggetto table\_1. Poi nella variabile String codice\_ viene salvata la stringa contenuta nella riga selezionata di colonna zero, ovvero contenente il codice identificativo del prodotto. A questo punto viene eseguita una query per l’eliminazione del prodotto, dal database nella tabella prodotto, avente quel codice identificativo sopra ricavato.

Se l’operazione è andata a buon fine verrà visualizzato il messaggio Prodotto eliminato correttamente, e si procederà a riaggiornare la tabella table\_1 con il metodo visto nel capitolo 6.3.1.

1. **STRUMENTI DI PROGRAMMAZIONE UTILIZZATI**