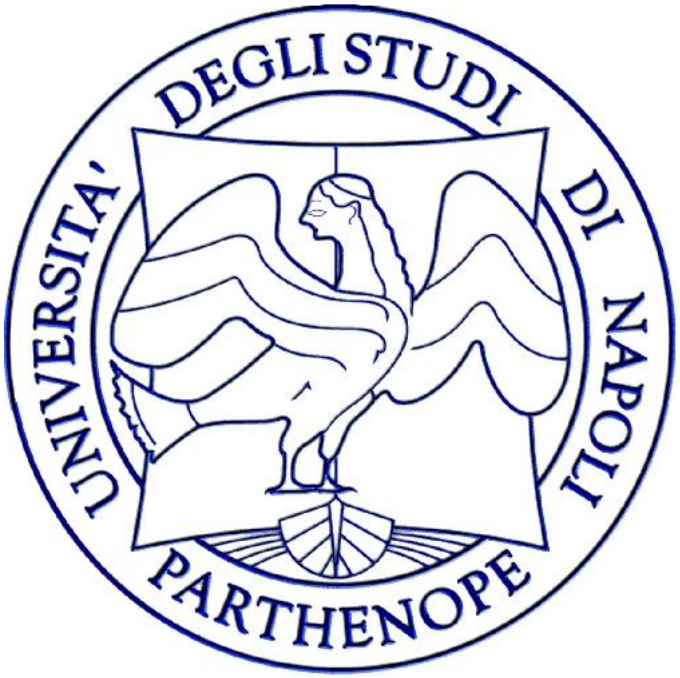
Università degli Studi di Napoli "Parthenope"
Dipartimento di Scienze e Tecnologie
Corso di laurea Triennale in Informatica



Programmazione 3 e Laboratorio di Programmazione 3

Distributore bevande

Docente
Prof. Angelo Ciaramella

Candidato
Vittorio Fones 0124/1384

Anno Accademico 2018-2019

Indice

Elenco delle figure

Capitolo 1

Introduzione

Il progetto in esame **è un simulatore di un distributore automatico di bevande**, abbreviato D.V.M.[1]. L'applicativo prevede due utilizzi, uno da **utilizzatore** e il secondo come **super utente**. Per praticitá l'utente non ha bisogno di effettuare un accesso tramite username e password, sebbene sia identificabile come vedremo. Si definisco di seguito gli attori e i loro task come da traccia.
L'utente può:

- scegliere, prelevare e pagare una bevanda. Il pagamento può avvenire secondo le modalità: contanti (5,10,20,50 centesimi, 1 e 2 euro), chiavetta ricaricabile o carta di credito;
- ricaricare una chiavetta inserendo contanti (5,10,20,50 euro).

L'amministratore può:

- periodicamente aggiungere bevande alla scorta. Il sistema controlla automaticamente se la bevanda è sotto scorta (minore di 1 litro);
- definire il prezzo per ogni tipo di bevanda;
- fare un report sui consumi mensili delle diverse tipologie di bevande;
- aggiungere una nuova tipologia di bevanda partendo da quelle già esistenti (e.g., thè con limone);

[1] Drink Vending Machine. In inglese vending machine è il distributore automatico

Capitolo 2

Scelte progettuali

2.1 Tecnologie usate

Il progetto è stato sviluppato come una web application, utilizzando le note tecnologie Servlet e JSP eseguite nel web server Apache Tomcat. Per le JSP[1], ovvero le pagine dinamiche in formato HTML, sono state sviluppate usando la recente notazione con tag JSTL che rende la lettura e interpretazione (da parte del programmatore) più semplice. I dati sono memorizzati in un database relazionale, usando PostgreSQL come RDBMS. L'interfaccia con la quale si comunica alla base di dati è offerta dal noto framework Open Source Hibernate, per tanto il cambiamento della sorgente dati è immediato, basterà modificare il file di configurazione .xml[2].

[1] JavaServer Pages

[2] Scelta arbitraria tra la configurazione del file .xml o la codifica nella classe atta alla configurazione

2.2 Architettura e logica di funzionamento

Lo sviluppo delle web application è per lo più basato seguendo un pattern noto come MVC (Model - view - controller). Il compito di questo pattern è di dividere la presentazione, ovvero l'interfaccia utente, e la sua logica di come manipola i dati e li presenta all'utilizzatore dell'applicativo.

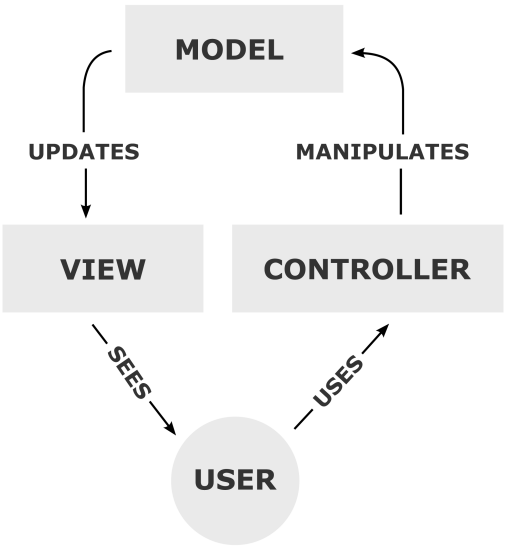


Figura 2.1: Diagramma di interazione del pattern MVC.

Il componente **controller** è dato dalle classi che estendono le Servlet: prendono i dati forniti dall'utente e contribuiscono al normale flusso dell'applicazione, interagendo con i dati. Il **model**, è composto dai **data model**, identificati più semplicemente come **Java Bean** (classi con attributi privati e metodi Getter e Setter) e dalle **actions** (modificatori dei data model). Nel caso di questo progetto le actions e i controller sono le servlet che scambiandosi i parametri delle chiamate Http usando i metodi Get e Post, aggiornano le **views**: i file JSP che utilizzando EL[3].

[3]Expression Language

Capitolo 3

Conclusioni

3.1 Database

Il database mostrato di seguito è usato per conservare le informazioni di accesso degli amministratori, i prodotti da vendere, gli acquisti delle bibite e le chiavi ricaricabili. Quest'ultime sono le uniche che contengono informazioni sugli utenti, poichè a seguito di assunzioni si è pensato che gli utenti vengano registrati dal DBA, così come gli amministratori. A seguito di ciò nello sviluppo non è stato preso in considerazione una pagina per la registrazione e un metodo nella classe Dao per l'eliminazioni di tuple.

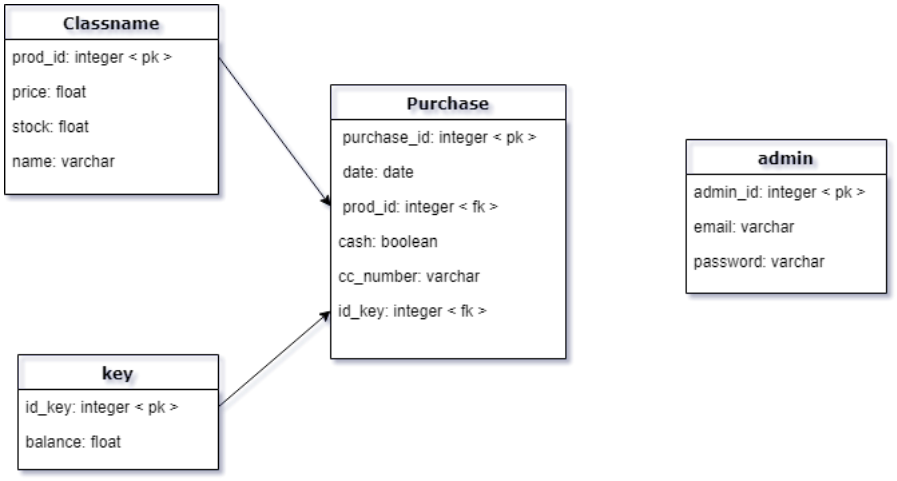


Figura 3.1: Database relazionale.

Capitolo 4

Diagramma delle classi

Di seguito vengono illustrati i **diagrammi delle classi** che fanno uso di specifici pattern. Per quanto riguarda le servlet, poiché queste hanno tutte le stesse **signature** si è deciso di mostrare solamente quelle che mostrano le informazioni alle views, piuttosto che elencare tutti i controller delle **rotte**[1].

4.1 Servlet come controller

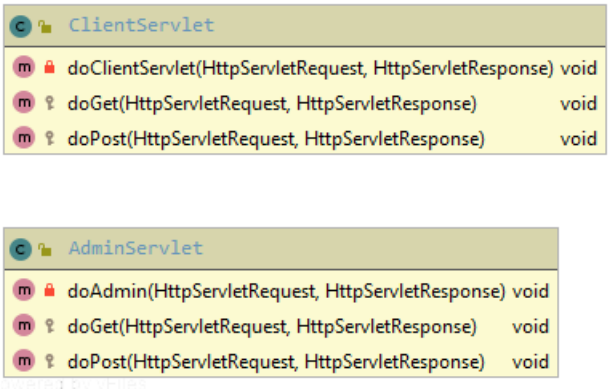


Figura 4.1: Servlet principali per le informazioni da passare alle views.

[1]In ambito web le rotte non sono altro che le risorse della url

4.2 Template Method e Singleton

Metodi per le operazioni del database creati con il template method e singleton. L'unica classe usata per interrogare il database sarà **GenericDao** che è una classe templata.

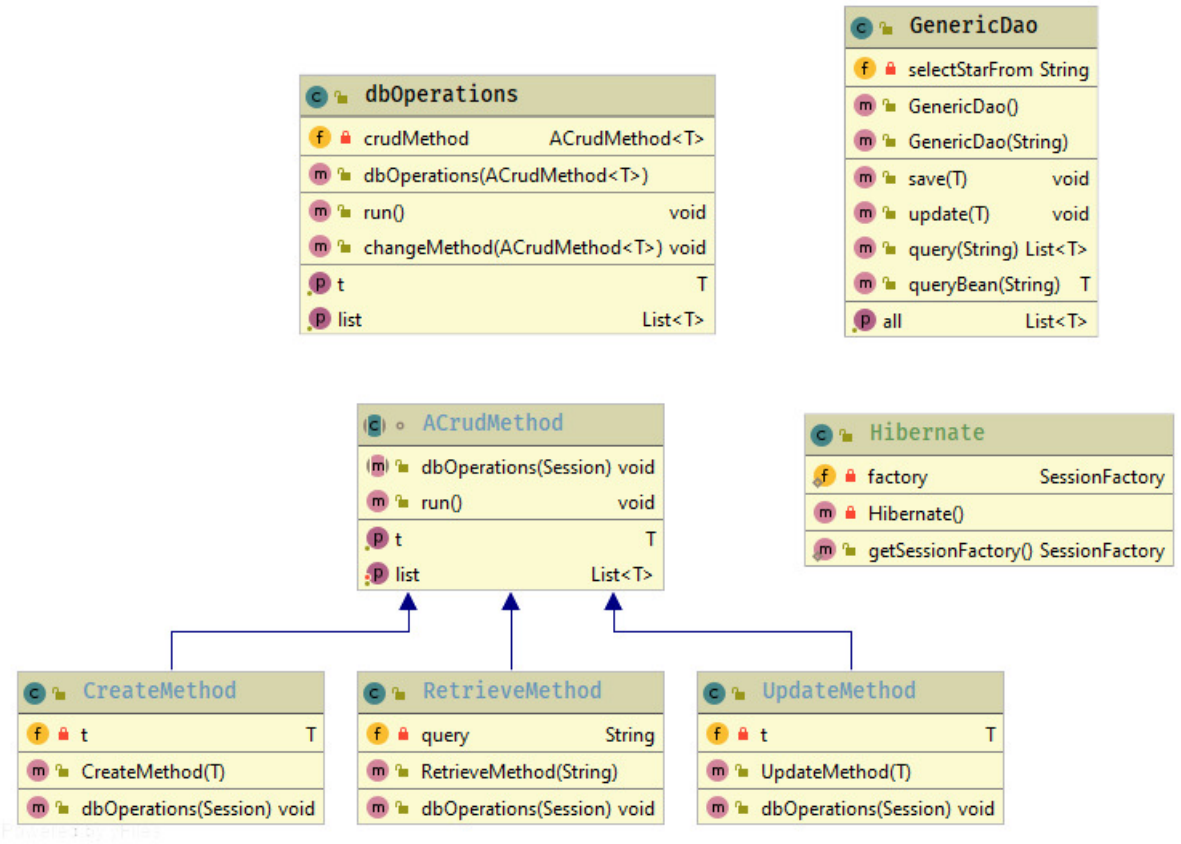


Figura 4.2: Diagramma dei metodi templati.

4.3 Java Bean

Principali Java Bean usate per "mappare" le entità del database

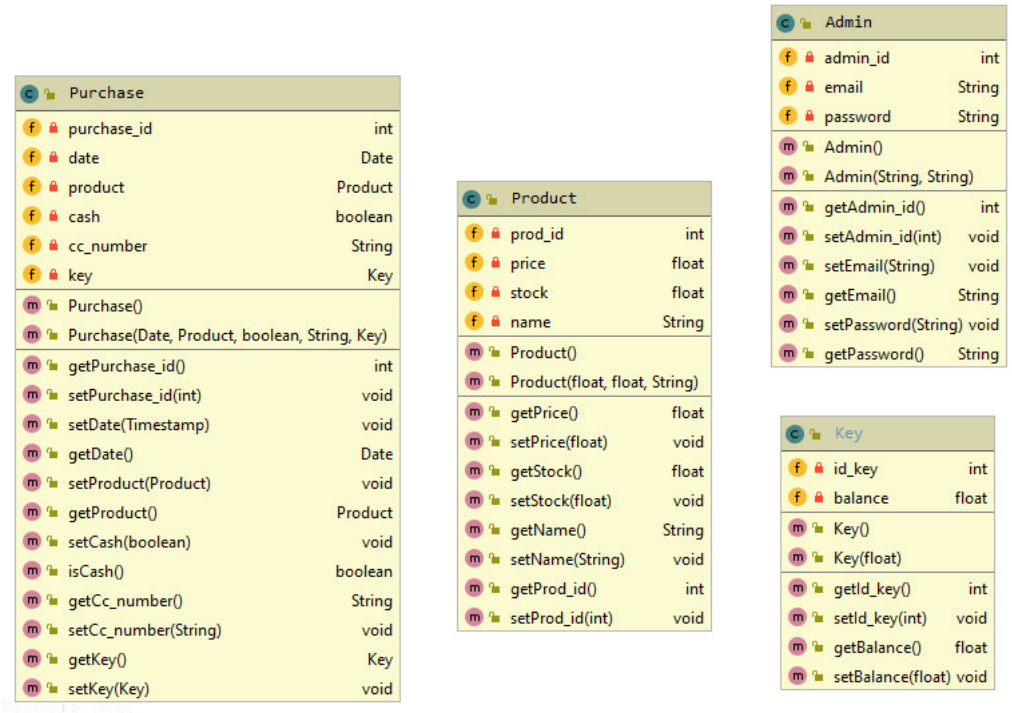


Figura 4.3: Java bean.

4.4 Chain of Responsibility

Controllo dei metodi di pagamento tramite catena di responsabilità.

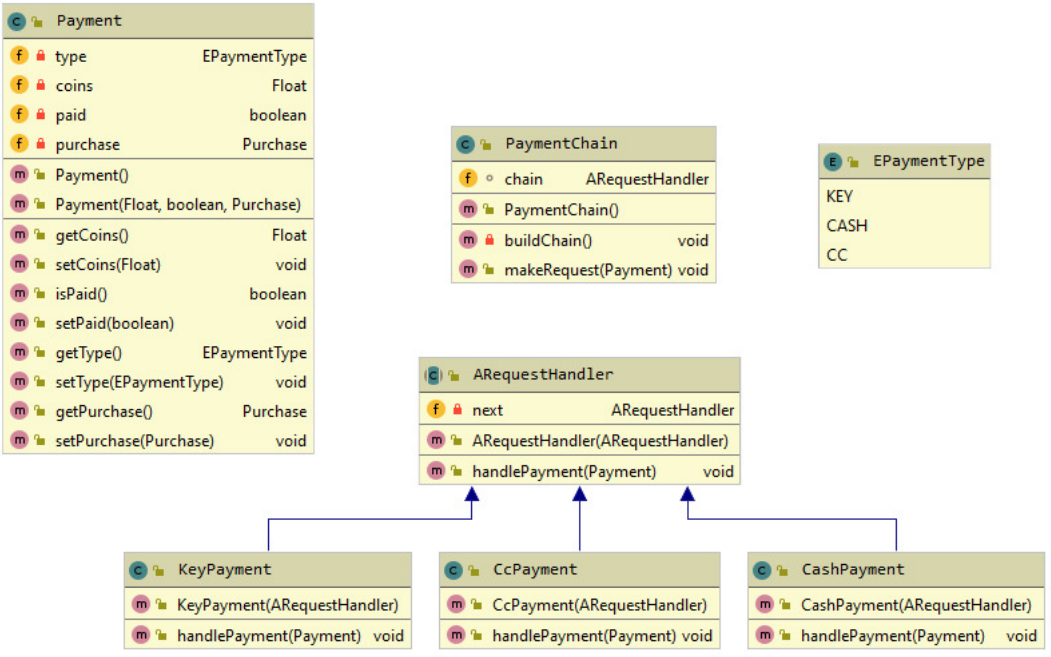


Figura 4.4: Diagramma della Chain of Responsibility.

Bibliografia

[1] Apache Tomcat
https://tomcat.apache.org/

[2] Hibernate middleware
https://hibernate.org/

[3] Wikipedia
https://wikipedia.org

[4] Google
https://www.google.com

[5] Stackoverflow
https://stackoverflow.com