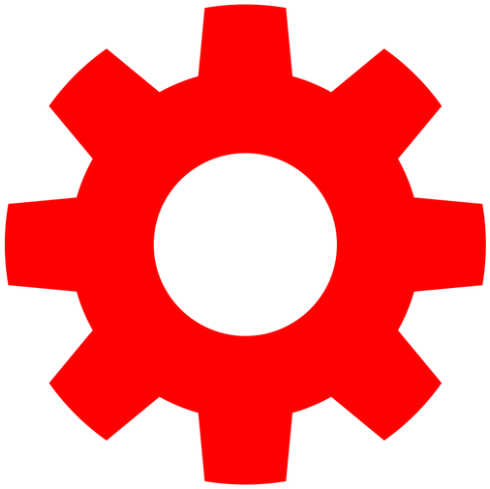
Università degli Studi di Salerno
Corso di Ingegneria del Software

SHOP-AUTORICAMBI
ObjectDesign Document - ODD
Versione 2.0



INDICE

1. Introduzione

1.1 Object design trade-off

1.1.1 Modularità contro efficienza

La modularità definita nel progetto SHOP-AUTORICAMBI si scontra con l'efficienza nell'elaborazione in lato server. La modularità facilita la creazione e la manutenzione del programma (simile al principio del divide et impera), inoltre ci garantisce l'utilizzo del codice in altri progetto/applicazione. Allo stesso tempo riduce l'efficienza dei tempi di risposta dei moduli che si occupano di determinati servizi.

1.1.2 Sicurezza contro efficienza

Nel nostro sistema i clienti vengono gestiti attraverso le sessioni ed un controllo del livello di utenza. Ciò comporta una diminuzione dell'efficienza ma tali controlli sono necessari per rispettare i requisiti iniziali del sistema. All'interno di ogni pagina utilizziamo delle precondizioni per gestire il controllo degli utenti, per evitare che i rischi di sicurezza, qualora l'utente malizioso digiti dal proprio browser il percorso esatto della chiamata al controller. Tali controlli sono un buon compromesso a discapito della poca efficienza persa per ogni chiamata ed aggiungono robustezza al sistema.

1.1.3 Portabilità contro efficienza

La portabilità del sistema SHOP-AUTORICAMBI è garantita dalla scelta del linguaggio di programmazione Java. Lo svantaggio dato da questa scelta è nella perdita di efficienza introdotta dal meccanismo della macchina virtuale Java. Tale compromesso è accettabile per i numerosi supporti forniti dal linguaggio Java.

1.2 Interface Documentation Guidelines

Gli sviluppatori dovranno seguire alcune linee guida per la scrittura del codice.

1.2.1 File Java

Ogni file sorgente deve contenere una sola classe o interfaccia pubblica. Ogni file deve contenere nel seguente ordine:

- Commenti per una migliore comprensione
- Dichiarazione del package
- Sezione import
- Dichiarazione di interfaccia o classe:
  - Attributi pubblici
  - Attributi privati
  - Attributi protetti
  - Costruttori
  - Altri metodi
- Classi interne

È previsto l'utilizzo di commenti JavaDoc.

1.2.2 Naming

L'utilizzo di convenzioni sui nomi rendono il programma più leggibile e comprensibile da tutti i membri del team. In particolare secondo il modello del codice programmato, è auspicabile che tutti siano in grado di intervenire su una qualsiasi linea di codice.

Classi e interfacce

I nomi delle classi sono nomi (composti anche da più parole) la cui iniziale è in maiuscolo. Ogni parola che compone un nome ha l'iniziale in maiuscolo.

I nomi delle classe devono essere semplici e descrittivi. Evitare l'uso di acronimi e abbreviazioni per i nomi delle classi.

Nel caso una o più classi incarnino design patterns noti è consigliato l'utilizzo di suffissi (inglesi) che richiamano lo specifico componente del design pattern (esempio: DatabaseAdapter, GiocatoreFactory, …).

E' consigliato l'uso della lingua italiana per i nomi, fatta eccezione per nomi inglesi di uso comune (esempio: TestingClass, …).

Metodi

I metodi devono essere verbi (composti anche da più parole) con iniziale minuscola.

Costanti

In accordo con le convenzioni suggerite dalla Sun, i nomi di constanti vengono indicati da nomi con tutte le parole in maiuscolo. Le parole vengono separate da underscore "_".

Ad esempio:

```
staticfinalint MAX_LENGHT = 24;
```

1.2.3 Uso dei commenti

E' permesso l'utilizzo di due tipi di commenti:
Commenti Javadoc (aree di testo compresa tra il simbolo /** e */)
Commenti in stile C (righe delimitate da //)
L'utilizzo dei commenti Javadoc è suggerito prima della dichiarazione di:
classi e interfacce
costruttori
metodi di almeno 3 righe di codice
variabili di classe
Ogni commento, compreso tra il simbolo /** e */, deve specificare le funzionalità e le specifiche del codice, senza esplicitare dettagli legati all'implementazione, in maniera tale da rendere leggibile tale documentazione anche a sviluppatori che non posseggono l'implementazione.

I commenti di Javadoc consentono la generazione automatica della documentazione del codice, attraverso l'utilizzo di appositi tools.

Il commenti stile C, ovvero le linee di codice precedute da //, sono utilizzati all'interno dei metodi, al fine di descrivere in maniera concisa e sintetica branch, cicli, condizioni o altri passi del codice.

1.2.4 Altre regole di stile

E' importante che vengano seguite anche ulteriori "regole di stile", al fine di produrre codice chiaro, leggibile e privo di errori.
Tra queste "regole di stile" elenchiamo le seguenti:

- I nomi di package, classi e metodi devono essere nomi descrittivi, facilmente pronunciabili e di uso comune
- Evitare l'utilizzo di abbreviazioni di parole
- Utilizzare, dove possibile, nomi largamente in uso nella comunità informatica (ie: i nomi dei design patterns)
- Preferire nomi con senso positivo a quelli con senso negativo
- Omogeneità dei nomi all'interno dell'applicazione

- Ottimizzazioni del codice non devono comunque inficiare la leggibilità dello stesso. Se si è costretti a sviluppare codice poco leggibile, perché le estreme prestazioni sono indispensabili è necessario documentarlo adeguatamente.
- Evitare la scrittura di righe di codice più lunghe di 80 caratteri e di file con più di 2000 righe
- È consigliato, per l'indentazione, l'utilizzo di spazi al posto dei "tab". Questo rende il codice ugualmente leggibile su tutti gli editor (alcuni editor convertono in automatico le tabulazioni in 4/6 spazi)
- È consigliato l'utilizzo di nomi in italiano. Tuttavia è consigliato l'utilizzo di termini inglesi laddove si tratta di uso comune o nel caso, molto comune, di termini comunemente usati nella loro versione inglese. E' di fondamentale importanza l'utilizzo di un dizionario dei nomi unico per tutto il progetto, che tutti i programmatori saranno tenuti a seguire.
- È consigliato l'utilizzo di nomi inglesi anche nel caso si adoperino termini della libreria standard di Java (ie: OptimizedList anziché ListaOttimizzata)
- Si consiglia l'utilizzo di parti standard dei nomi in casi come:
- Classi astratte, suffisso Abstract- (ie: AbstractProdotto)
- Design patterns (ie: se si usa l'MVC utilizzare ListModel)
- Accezioni terminanti per Exception (ie: UtenteNonTrovatoException)
- Altre situazioni analoghe
- I nomi delle interfacce segue le regole standard dei nomi. E' sconsigliato usare il prefisso o suffisso "Interface"
- È consigliato l'utilizzo di suffissi "standard" come "get", "set", "is" o "has" in inglese
- È possibile scrivere dichiarazioni di metodi e classi in due righe, se eccessivamente lunghi
- Evitare la notazione ungherese. La notazione ungherese, che prevede l'utilizzo di prefissi per descrivere il tipo di dato, non dovrebbe essere utilizzata. La motivazione è semplice: la notazione ungherese va bene per linguaggi che hanno tipi semplici, e dove è possibile creare un vocabolario di prefissi limitato. In linguaggi OOP i tipi primitivi hanno un uso più limitato, mentre sono gli oggetti a farla da padrone.

- Dichiarare le variabili ad inizio blocco, sia questo un metodo o una classe, in modo da raccogliere in un unico punto tutte le dichiarazioni.
- Utilizzare la dichiarazione per definire una sola variabile – evitando più dichiarazioni sulla stessa riga
- L'inizializzazione delle variabili deve essere eseguita in fase di dichiarazione, impostando un valore di default o il risultato di un metodo. Se proprio ciò non è possibile, in quanto il valore da impostare è il risultato di una elaborazione compiuta nel metodo stesso, inizializzare la variabile appena prima del suo utilizzo
- Allineare la dichiarazione delle variabili per renderle più leggibili, strutturandole in blocchi omogenei per contesto (e non per tipo di dato)
- Nel caso di algoritmi troppo complessi, eseguire un refactoring per separarlo in diversi sotto-metodi più semplici.
- I cicli devono seguire le seguenti regole:
- Per le variabili, utilizzare l'area di visibilità più stretta possibile, dichiarando le variabili appena prima del loro utilizzo.
- Per le chiamate a metodo non utilizzare spazi dopo il nome del metodo.

1.3Definizioni, acronimi e abbreviazioni

ACID	Atomicità, Consistenza, Isolamento e Durabilità
DBMS	Database Management System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JDBC	Java Database Connectivity
JSP	Java Server Page
MVC	Model View Controller
RAD	Requirements Analysis Document
RAD	Documento di Analisi dei Requisiti
SDD	System Design Document
SQL	Structured Query Language

1.4Riferimenti

- RAD SHOP-AUTORICAMBI documento analisi dei requisiti
- SDD SHOP-AUTORICAMBI documento di system design
- dispense dei corsi seguiti in precedenza

1.5Overview

Nelle sezioni successive sarà descritta l'architettura del sistema e le sue componenti principali. Saranno esposte le tipologie di utenza ed i comportamenti del sistema previsti per ogni tipologia, nonché le funzionalità delle componenti invocate.

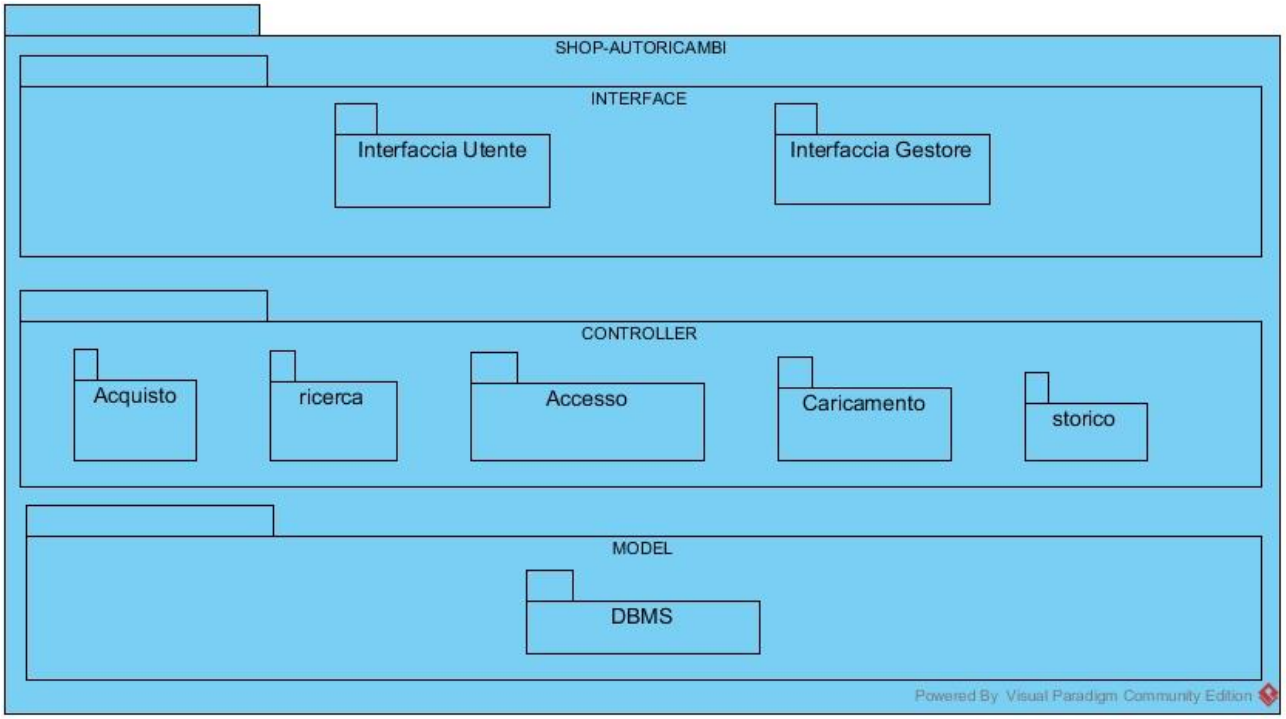
Saranno inoltre descritti i requisiti minimi per la macchina che ospiterà il sistema e le politiche di sicurezza adottate dal sistema.

2.Packages

Come possiamo notare dal documento SDD SHOP-AUTORICAMBI le componenti base che costituiscono il sistema sono raccolte in moduli a loro volta raccolti in livelli. I tre livelli rappresentano la suddivisione dettata dal modello di

architettura preso in considerazione per il sistema SHOP-AUTORICAMBI “MVC” (Model View Controller). Ciascun livello rappresenta un package contenente le componenti relative alle funzioni associate al livello.

- PACKAGE SOURCE
    - MODEL
        - Acquisto_Manager
        - Prodotto_Manager
        - Carrello_Manager
        - Client_Manager
        - Login_Manager
        - Storicocliente_Manager
        - Storicogestore_Manager
    - VIEW
        - Prodotti_Boundary
        - Carrello_Boundary
        - RegistrazioneClient_Boundary
        - Login_Boundary
        - Home_Boundary
        - AreaUtente_Boundary
        - Storicocliente_Boundary
        - Storicogestore_Boundary
    - CONTROLLER
        - Acquisto_Control
        - AmministratoreProdotti_Control
        - Carrello_Control
        - Registrazione_Control
        - LogOut_Control
        - Storicocliente_Control
        - Storicogestore_control

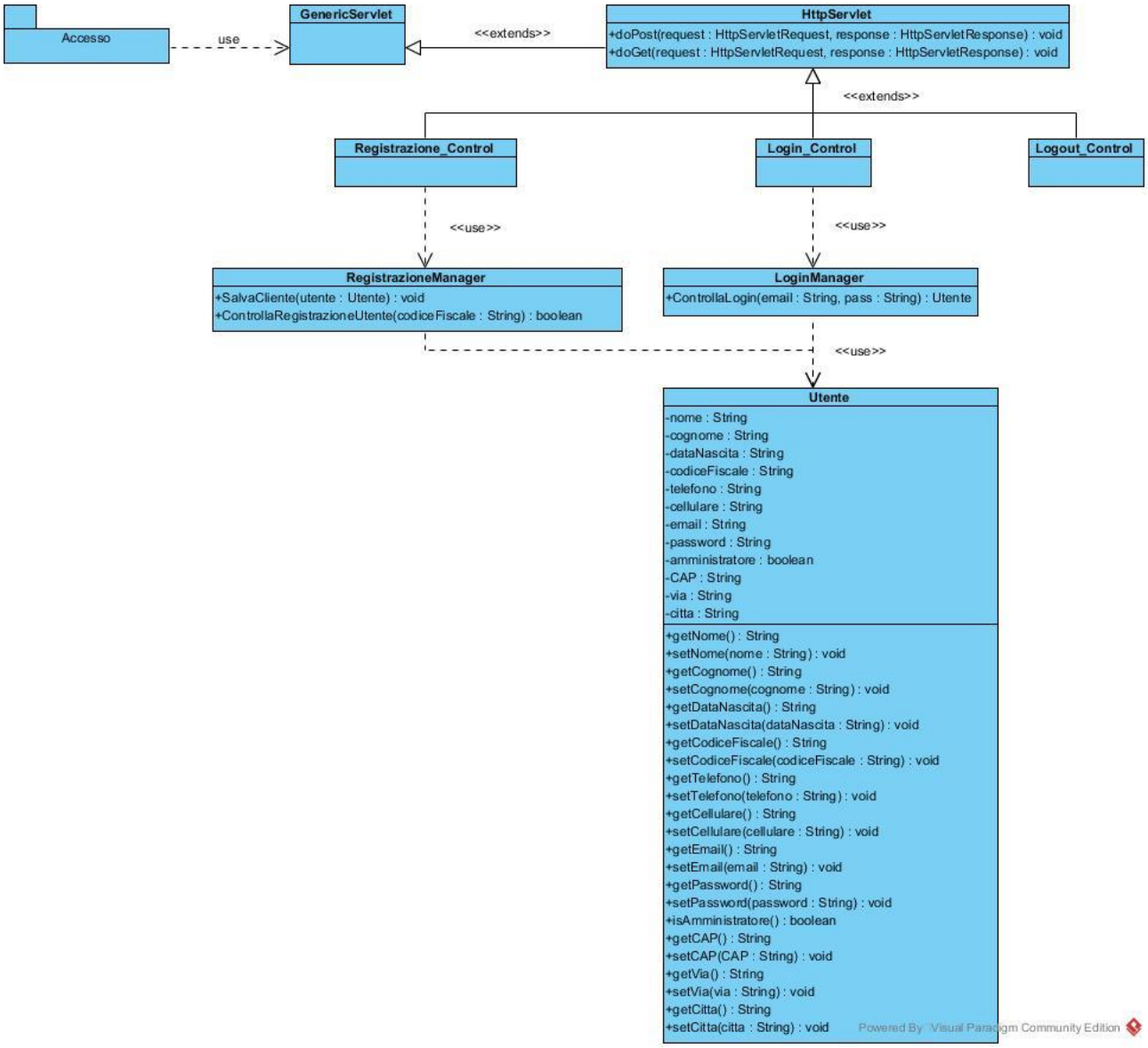

SHOP-AUTORICAMBI
INTERFACE
Interfaccia Utente
Interfaccia Gestore
CONTROLLER
Acquisto
ricerca
Accesso
Caricamento
storico
MODEL
DBMS
Powered By Visual Paradigm Community Edition

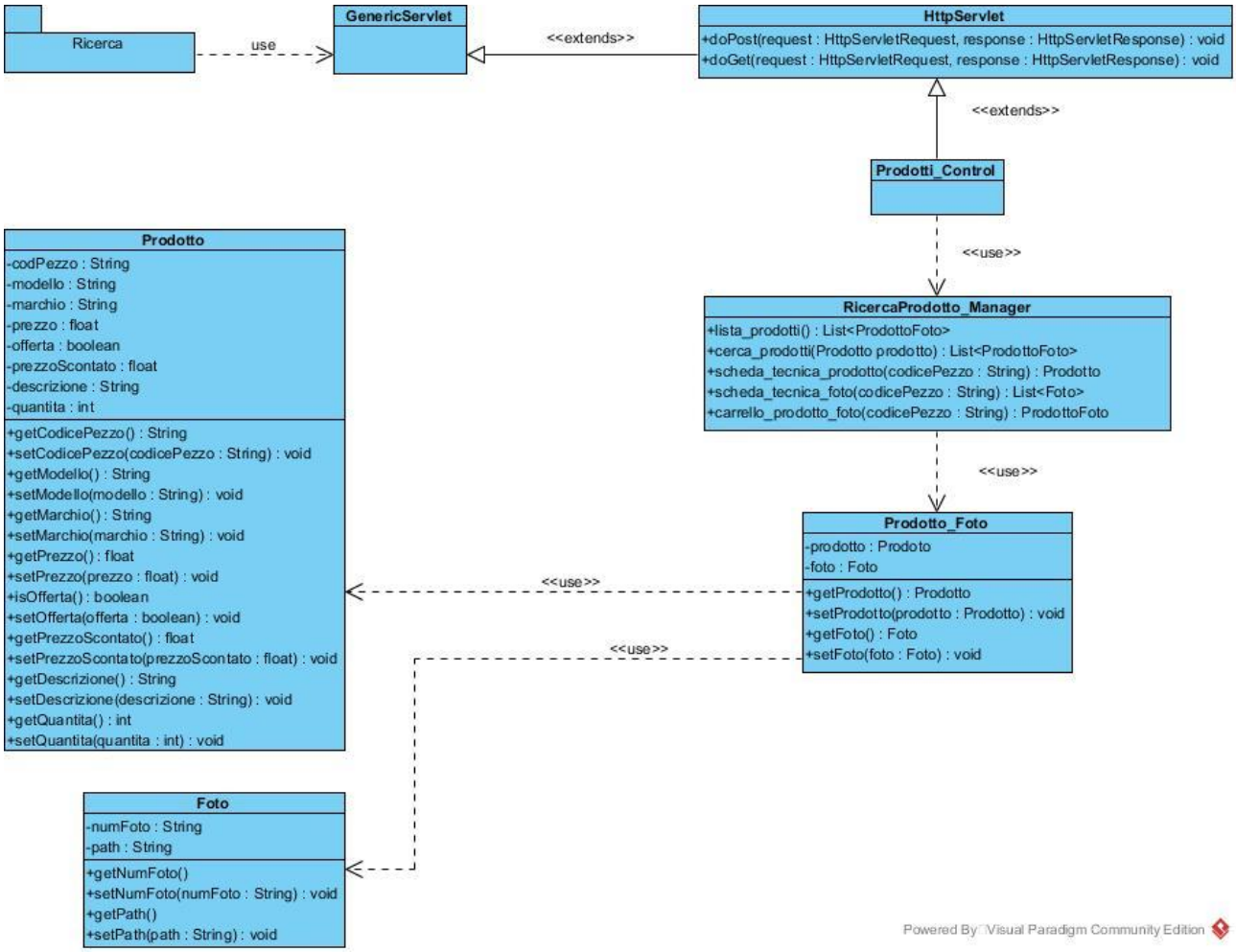
3. Interfaccedelle classi

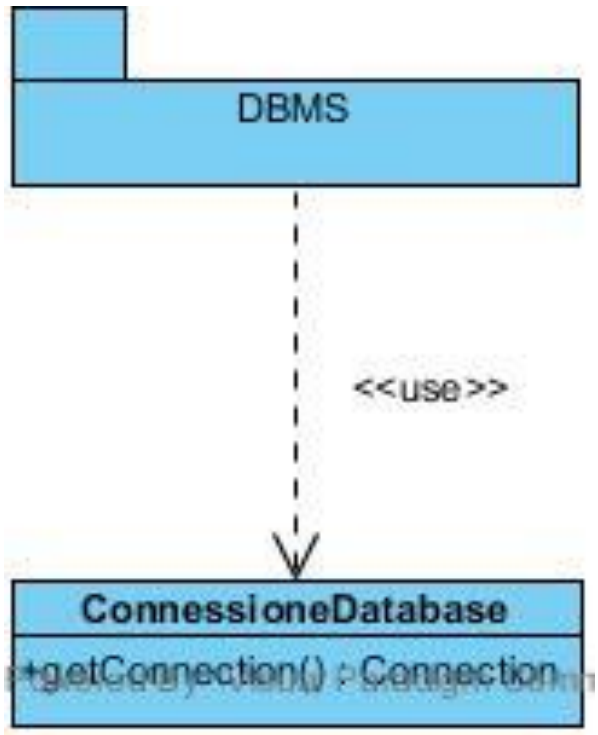
Si procede all'analisi dettagliata delle piccole classi implementate nel sistema.

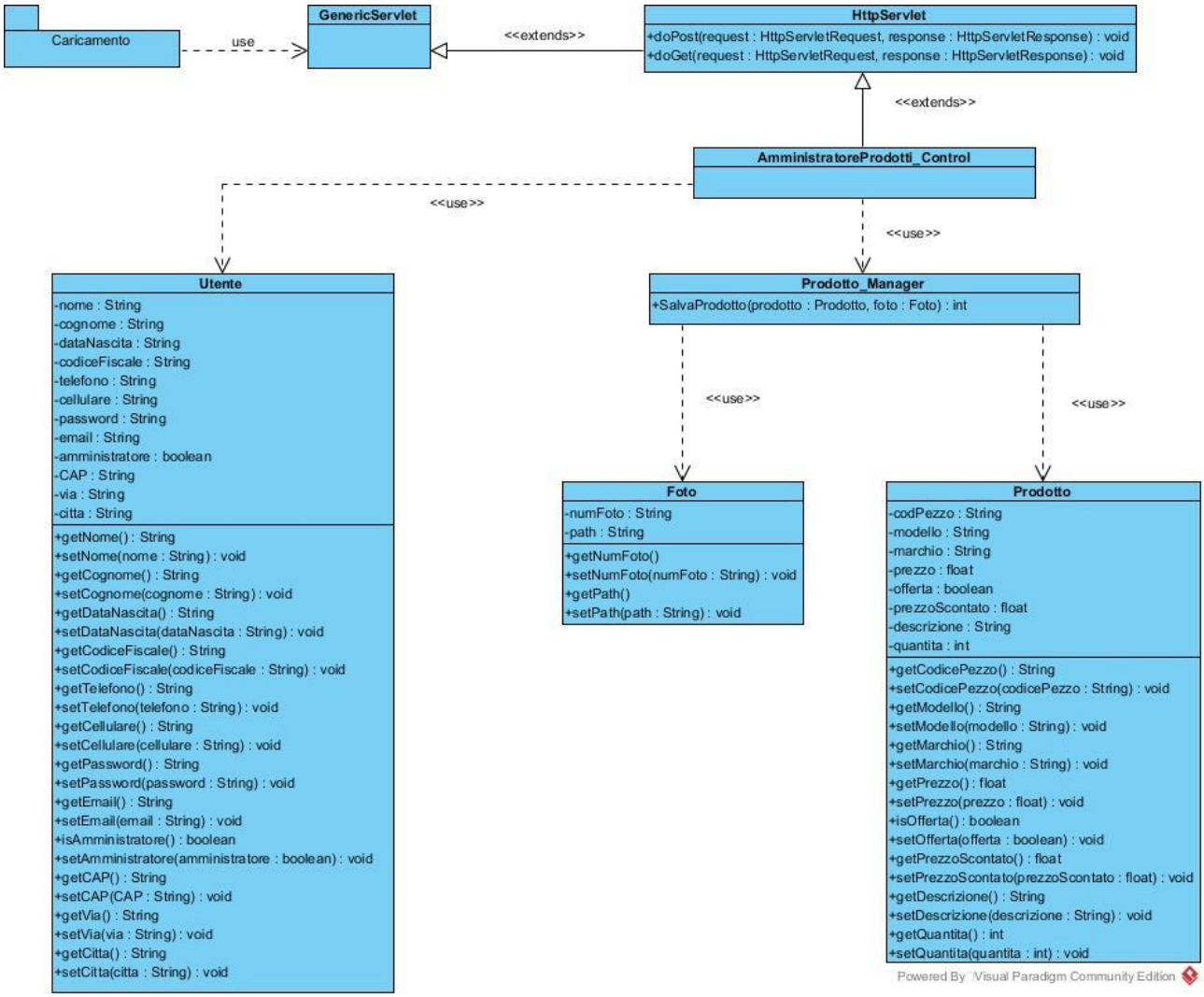
L'analisi serve ad evidenziare le interfacce di interazione utilizzate nella progettazione del software.

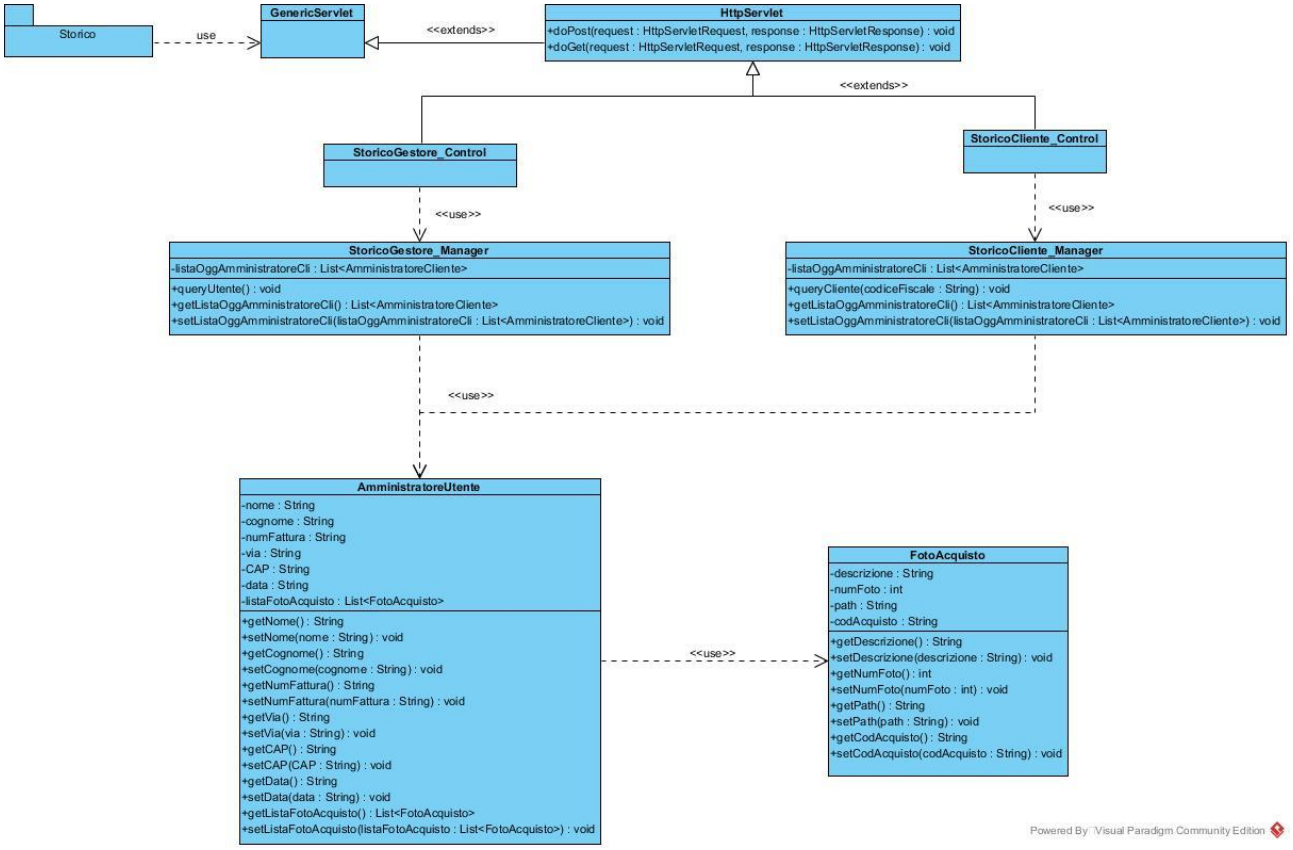
3.1 ClassDiagram

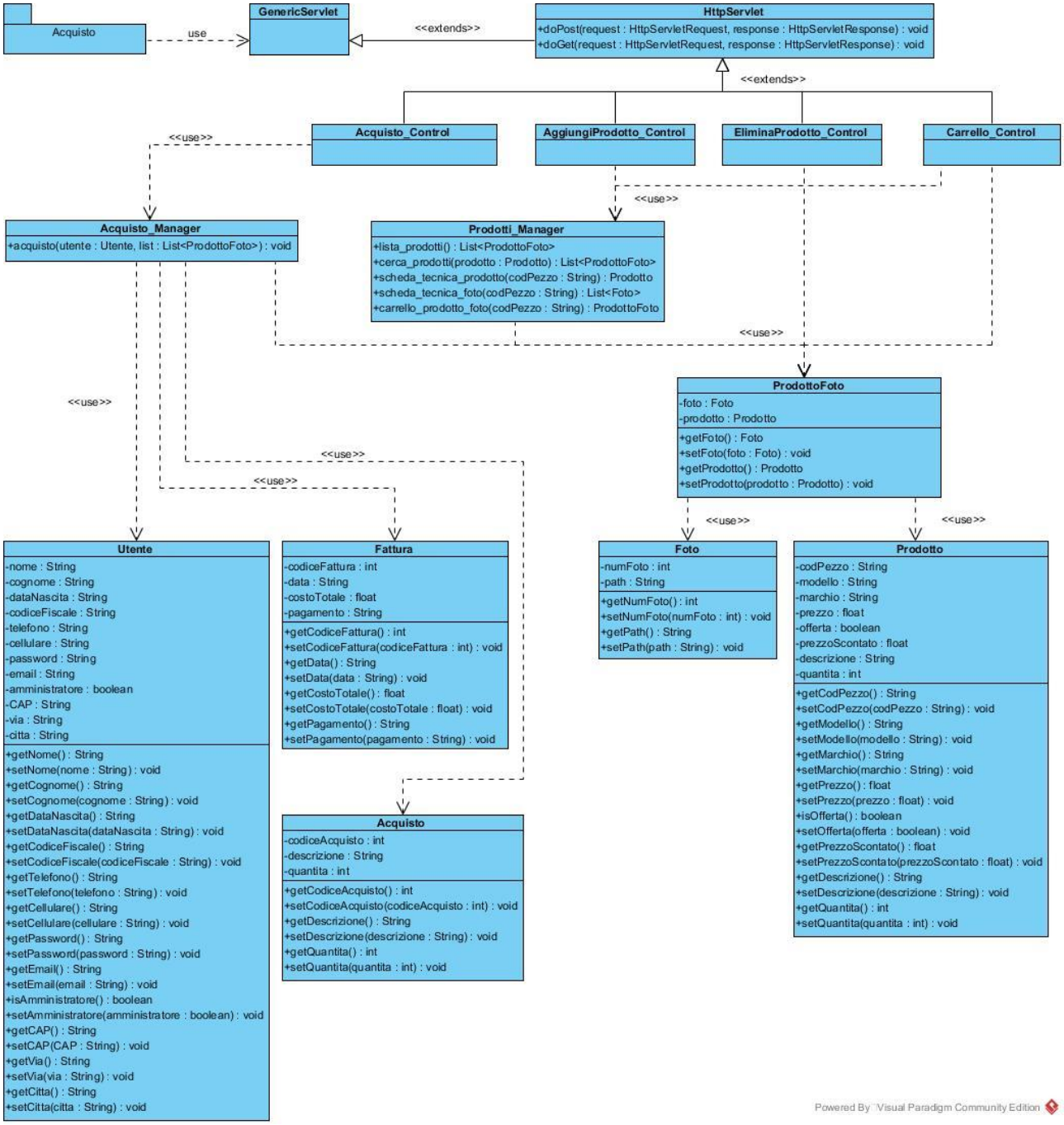



Ricerca
use
GenericServlet
<<extends>>
HttpServlet
+doPost(request : HttpServletRequest, response : HttpServletResponse) : void
+doGet(request : HttpServletRequest, response : HttpServletResponse) : void
<<extends>>
Prodotti_Control
<<use>>
RicercaProdotto_Manager
+lista_prodotti() : List<ProdottoFoto>
+cerca_prodotti(Prodotto prodotto) : List<ProdottoFoto>
+scheda_tecnica_prodotto(codicePezzo : String) : Prodotto
+scheda_tecnica_foto(codicePezzo : String) : List<Foto>
+carrello_prodotto_foto(codicePezzo : String) : ProdottoFoto
<<use>>
Prodotto_Foto
-prodotto : Prodoto
-foto : Foto
+getProdotto() : Prodotto
+setProdotto(prodotto : Prodotto) : void
+getFoto() : Foto
+setFoto(foto : Foto) : void
<<use>>
<<use>>
Prodotto
-codPezzo : String
-modello : String
-marchio : String
-prezzo : float
-offerta : boolean
-prezzoScontato : float
-descrizione : String
-quantita : int
+getCodicePezzo() : String
+setCodicePezzo(codicePezzo : String) : void
+getModello() : String
+setModello(modello : String) : void
+getMarchio() : String
+setMarchio(marchio : String) : void
+getPrezzo() : float
+setPrezzo(prezzo : float) : void
+isOfferta() : boolean
+setOfferta(offerta : boolean) : void
+getPrezzoScontato() : float
+setPrezzoScontato(prezzoScontato : float) : void
+getDescrizione() : String
+setDescrizione(descrizione : String) : void
+getQuantita() : int
+setQuantita(quantita : int) : void
Foto
-numFoto : String
-path : String
+getNumFoto()
+setNumFoto(numFoto : String) : void
+getPath()
+setPath(path : String) : void
Powered By Visual Paradigm Community Edition


DBMS
<<use>>
ConnessioneDatabase
+getConnection() : Connection


Caricamento
use
GenericServlet
<<extends>>
HttpServlet
+doPost(request : HttpServletRequest, response : HttpServletResponse) : void
+doGet(request : HttpServletRequest, response : HttpServletResponse) : void
<<extends>>
AmministratoreProdotti_Control
<<use>>
<<use>>
Utente
-nome : String
-cognome : String
-dataNascita : String
-codiceFiscale : String
-telefono : String
-cellulare : String
-password : String
-email : String
-amministratore : boolean
-CAP : String
-via : String
-citta : String
+getNome() : String
+setNome(nome : String) : void
+getCognome() : String
+setCognome(cognome : String) : void
+getDataNascita() : String
+setDataNascita(dataNascita : String) : void
+getCodiceFiscale() : String
+setCodiceFiscale(codiceFiscale : String) : void
+getTelefono() : String
+setTelefono(telefono : String) : void
+getCellulare() : String
+setCellulare(cellulare : String) : void
+getPassword() : String
+setPassword(password : String) : void
+getEmail() : String
+setEmail(email : String) : void
+isAmministratore() : boolean
+setAmministratore(amministratore : boolean) : void
+getCAP() : String
+setCAP(CAP : String) : void
+getVia() : String
+setVia(via : String) : void
+getCitta() : String
+setCitta(citta : String) : void
Prodotto_Manager
+SalvaProdotto(prodotto : Prodotto, foto : Foto) : int
<<use>>
<<use>>
Foto
-numFoto : String
-path : String
+getNumFoto()
+setNumFoto(numFoto : String) : void
+getPath()
+setPath(path : String) : void
Prodotto
-codPezzo : String
-modello : String
-marchio : String
-prezzo : float
-offerta : boolean
-prezzoScontato : float
-descrizione : String
-quantita : int
+getCodicePezzo() : String
+setCodicePezzo(codicePezzo : String) : void
+getModello() : String
+setModello(modello : String) : void
+getMarchio() : String
+setMarchio(marchio : String) : void
+getPrezzo() : float
+setPrezzo(prezzo : float) : void
+isOfferta() : boolean
+setOfferta(offerta : boolean) : void
+getPrezzoScontato() : float
+setPrezzoScontato(prezzoScontato : float) : void
+getDescrizione() : String
+setDescrizione(descrizione : String) : void
+getQuantita() : int
+setQuantita(quantita : int) : void
Powered By Visual Paradigm Community Edition

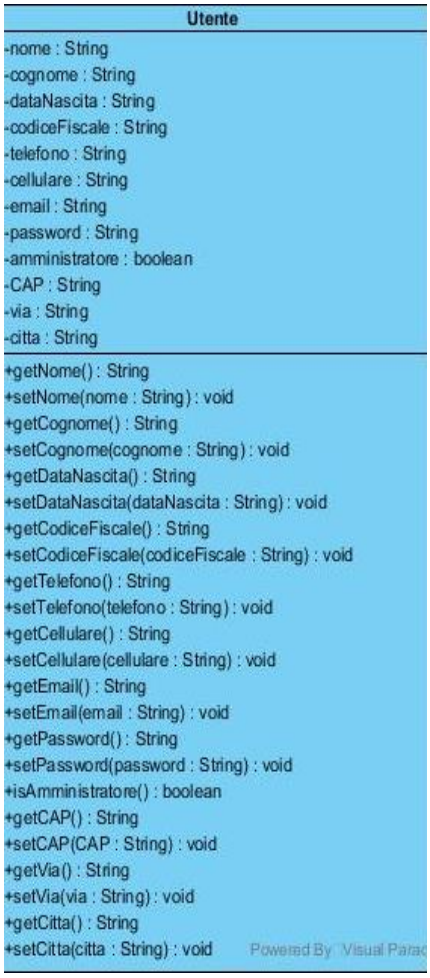

Storico
use
GenericServlet
<<extends>>
HttpServlet
+doPost(request : HttpServletRequest, response : HttpServletResponse) : void
+doGet(request : HttpServletRequest, response : HttpServletResponse) : void
<<extends>>
StoricoGestore_Control
StoricoCliente_Control
<<use>>
<<use>>
StoricoGestore_Manager
-listaOggAmministratoreCli : List<AmministratoreCliente>
+queryUtente() : void
+getListaOggAmministratoreCli() : List<AmministratoreCliente>
+setListaOggAmministratoreCli(listaOggAmministratoreCli : List<AmministratoreCliente>) : void
StoricoCliente_Manager
-listaOggAmministratoreCli : List<AmministratoreCliente>
+queryCliente(codiceFiscale : String) : void
+getListaOggAmministratoreCli() : List<AmministratoreCliente>
+setListaOggAmministratoreCli(listaOggAmministratoreCli : List<AmministratoreCliente>) : void
<<use>>
AmministratoreUtente
-nome : String
-cognome : String
-numFattura : String
-via : String
-CAP : String
-data : String
-listaFotoAcquisto : List<FotoAcquisto>
+getNome() : String
+setNome(nome : String) : void
+getCognome() : String
+setCognome(cognome : String) : void
+getNumFattura() : String
+setNumFattura(numFattura : String) : void
+getVia() : String
+setVia(via : String) : void
+getCAP() : String
+setCAP(CAP : String) : void
+getData() : String
+setData(data : String) : void
+getListaFotoAcquisto() : List<FotoAcquisto>
+setListaFotoAcquisto(listaFotoAcquisto : List<FotoAcquisto>) : void
<<use>>
FotoAcquisto
-descrizione : String
-numFoto : int
-path : String
-codAcquisto : String
+getDescrizione() : String
+setDescrizione(descrizione : String) : void
+getNumFoto() : int
+setNumFoto(numFoto : int) : void
+getPath() : String
+setPath(path : String) : void
+getCodAcquisto() : String
+setCodAcquisto(codAcquisto : String) : void
Powered By Visual Paradigm Community Edition


Acquisto
use
GenericServlet
<<extends>>
HttpServlet
+doPost(request : HttpServletRequest, response : HttpServletResponse) : void
+doGet(request : HttpServletRequest, response : HttpServletResponse) : void
<<extends>>
Acquisto_Control
AggiungiProdotto_Control
EliminaProdotto_Control
Carrello_Control
<<use>>
<<use>>
Acquisto_Manager
+acquisto(utente : Utente, list : List<ProdottoFoto>) : void
Prodotti_Manager
+lista_prodotti() : List<ProdottoFoto>
+cerca_prodotti(prodotto : Prodotto) : List<ProdottoFoto>
+scheda_tecnica_prodotto(codPezzo : String) : Prodotto
+scheda_tecnica_foto(codPezzo : String) : List<Foto>
+carrello_prodotto_foto(codPezzo : String) : ProdottoFoto
<<use>>
<<use>>
ProdottoFoto
-foto : Foto
-prodotto : Prodotto
+getFoto() : Foto
+setFoto(foto : Foto) : void
+getProdotto() : Prodotto
+setProdotto(prodotto : Prodotto) : void
<<use>>
<<use>>
<<use>>
<<use>>
Utente
-nome : String
-cognome : String
-dataNascita : String
-codiceFiscale : String
-telefono : String
-cellulare : String
-password : String
-email : String
-amministratore : boolean
-CAP : String
-via : String
-citta : String
+getNome() : String
+setNome(nome : String) : void
+getCognome() : String
+setCognome(cognome : String) : void
+getDataNascita() : String
+setDataNascita(dataNascita : String) : void
+getCodiceFiscale() : String
+setCodiceFiscale(codiceFiscale : String) : void
+getTelefono() : String
+setTelefono(telefono : String) : void
+getCellulare() : String
+setCellulare(cellulare : String) : void
+getPassword() : String
+setPassword(password : String) : void
+getEmail() : String
+setEmail(email : String) : void
+isAmministratore() : boolean
+setAmministratore(amministratore : boolean) : void
+getCAP() : String
+setCAP(CAP : String) : void
+getVia() : String
+setVia(via : String) : void
+getCitta() : String
+setCitta(citta : String) : void
Fattura
-codiceFattura : int
-data : String
-costoTotale : float
-pagamento : String
+getCodiceFattura() : int
+setCodiceFattura(codiceFattura : int) : void
+getData() : String
+setData(data : String) : void
+getCostoTotale() : float
+setCostoTotale(costoTotale : float) : void
+getPagamento() : String
+setPagamento(pagamento : String) : void
Acquisto
-codiceAcquisto : int
-descrizione : String
-quantita : int
+getCodiceAcquisto() : int
+setCodiceAcquisto(codiceAcquisto : int) : void
+getDescrizione() : String
+setDescrizione(descrizione : String) : void
+getQuantita() : int
+setQuantita(quantita : int) : void
Foto
-numFoto : int
-path : String
+getNumFoto() : int
+setNumFoto(numFoto : int) : void
+getPath() : String
+setPath(path : String) : void
Prodotto
-codPezzo : String
-modello : String
-marchio : String
-prezzo : float
-offerta : boolean
-prezzoScontato : float
-descrizione : String
-quantita : int
+getCodPezzo() : String
+setCodPezzo(codPezzo : String) : void
+getModello() : String
+setModello(modello : String) : void
+getMarchio() : String
+setMarchio(marchio : String) : void
+getPrezzo() : float
+setPrezzo(prezzo : float) : void
+isOfferta() : boolean
+setOfferta(offerta : boolean) : void
+getPrezzoScontato() : float
+setPrezzoScontato(prezzoScontato : float) : void
+getDescrizione() : String
+setDescrizione(descrizione : String) : void
+getQuantita() : int
+setQuantita(quantita : int) : void
Powered By Visual Paradigm Community Edition

3.2 Descrizione delle classi

3.2.1 UTENTE

La classe contiene le informazioni relative ad un generico utente.

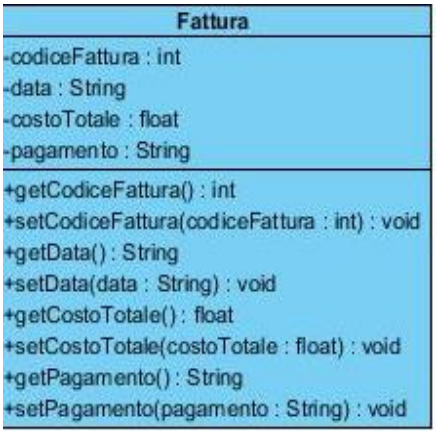


- Private nome as string
  Nome dell'utente
- Private cognome as string
  Cognome dell'utente
- Private dataNascita as string
  Data di nascita dell'utente
- Private codiceFiscale as string
  Codice fiscale dell'utente
- Private telefono as string
  Telefono dell'utente
- Private cellulare as string
  Cellulare dell'utente
- Private email as string
  Email dell'utente
- Private password as string
  Password dell'utente
- Private amministratore as boolean
  Segna se l'utente registrato è un amministratore
- Private CAP as string
  Codice di avviamento postale dell'utente
- Private via as string
  Nome della strada dove è ubicato l'utente
- Private citta as string
  Nome della città dove è ubicato l'utente

Sono inoltre presenti tutti i metodi di lettura e scrittura (set e get) per gli attributi private della classe.

3.2.2. FATTURA

La classe fattura contiene le fatture per il singolo cliente.



- Private codiceFattura as int
  Indica il codice della fattura
- Private data as string
  Indica la data della fattura
- Private costoTotale as float
  Indica il costo totale riportato nella fattura
- Private pagamento as string
  Indica il metodo di pagamento della somma

Sono inoltre presenti tutti i metodi di lettura e scrittura (set e get) per gli attributi private della classe.

3.2.3 ACQUISTO

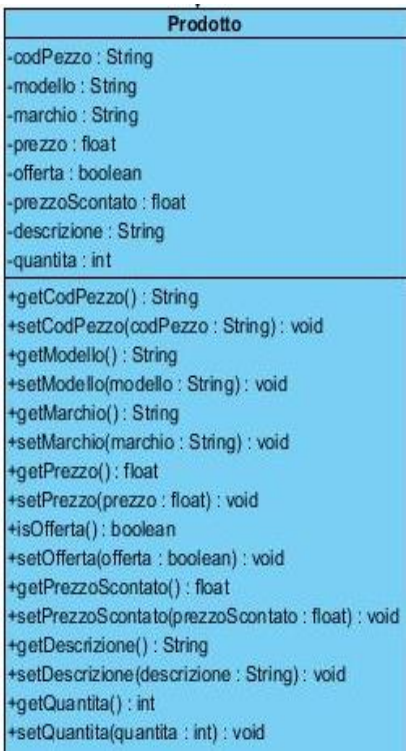
La classe acquisto contiene i prodotti acquistati dal cliente

- Private codiceAcquisto as int
  Indica il codice dell'acquisto
- Private descrizione as string
  Contiene la descrizione dell'acquisto
- Private quantita as int
  Contiene il numero di prodotti comprati

Sono inoltre presenti tutti i metodi di lettura e scrittura (set e get) per gli attributi private della classe.

3.2.4 PRODOTTO

Contiene i prodotti del sito Shop-Autoricambi.

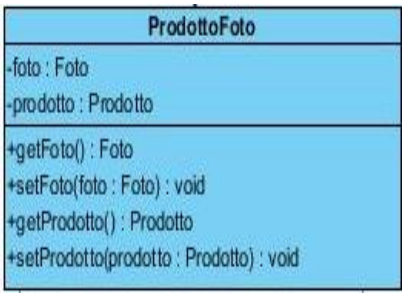


- Private codPezzo as string
  Contiene il codice del prodotto
- Private modello as string
  Contiene il modello del prodotto
- Private marchio as string
  Contiene il marchio del prodotto
- Private prezzo as float
  Contiene il prezzo del prodotto
- Private offerta as boolean
  È vero se il prodotto è in offerta
- Private prezzoScontato as float
  Contiene un eventuale prezzo scontato del prodotto
- Private descrizione as string
  Contiene la descrizione del prodotto
- Private quantita as int
  Contiene la quantità del prodotto che è disponibile

Sono inoltre presenti tutti i metodi di lettura e scrittura (set e get) per gli attributi private della classe.

3.2.5. PRODOTTO FOTO

Contiene la foto del prodotto

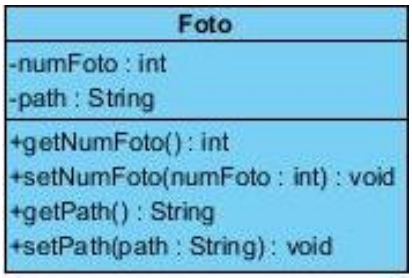


- Private foto as Foto
  Contiene l'oggetto foto associata al prodotto
- Private prodotto as Prodotto
  Contiene l'oggetto prodotto associato alla foto

Sono inoltre presenti tutti i metodi di lettura e scrittura (set e get) per gli attributi private della classe.

3.2.6 FOTO

Contiene la foto del prodotto.

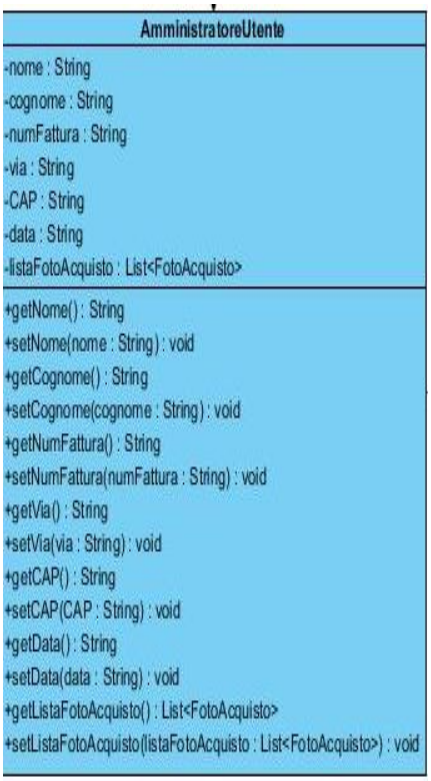


- Private numFoto as int
  Contiene l'id della foto
- Private path as string
  Contiene il path della foto

Sono inoltre presenti tutti i metodi di lettura e scrittura (set e get) per gli attributi private della classe.

3.2.7 AMMINISTRATORE UTENTE

Contiene l'amministratore del sistema.



- Private nome as string
  Contiene il nome dell'amministratore
- Private cognome as string
  Contiene il cognome dell'amministratore
- Private numFattura as String
  Contiene il numero della fattura
- Private via as String
  Contiene la via dove è ubicato l'amministratore
- Private CAP as String
  Contiene il Codice di avviamento postale
- Private data as String
  Contiene la data.
- Private listafotoacquisto as lista
  Contiene la lista dell'oggetto FotoAcquisto.

Sono inoltre presenti tutti i metodi di lettura e scrittura (set e get) per gli attributi private della classe.

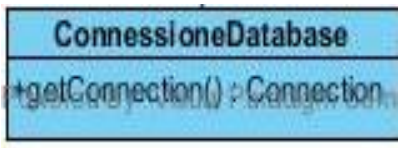
3.2.8 FOTOACQUISTO

contiene la foto dell'acquisto.

- Private descrizione as string
  Contiene la descrizione
- Private numfoto as int
  Contiene l'id della foto
- Private path as string
  Contiene il path della foto
- Private codAcquisto as string
  Contiene il codice dell'acquisto

3.2.9 DATABASE

Viene utilizzato per la connessione al database.



- Public getConnection() as Connection
  Serve per accedere al database.

Sono inoltre presenti tutti i metodi di lettura e scrittura (set e get) per gli attributi private della classe.

4. GLOSSARIO

Termini	Descrizione
ODD	Object Design Document
SDD	System Design Document
DB	Database management system