Object Design Document

Riferimento	
Versione	0.1
Data	15/10/2021
Destinatario	Gravino Carmine
Presentato da	S.Cataldo, P.Guidotti, C.Fiumarella, C.Orrigo
Approvato da	

Team Members

Nome e Cognome	Matricola
Onelia Sara Cataldo	0512105504
Paolo Guidotti	0512105261
Chiara Orrigo	0512105090
Camilla Fiumarella	0512105213

Cronologia revisioni

Data	Versione	Descrizione	Autori
10/12/2021	0.1	Definizione punti salienti	[Tutti]
12/12/2021	0.2	Introduzione	Sara Cataldo, Paolo Guidotti
15/12/2021	0.3	Package	Camilla Fiumarella, Chiara Orrigo
15/12/2021	0.4	Class Interfaces	[Tutti]
16/01/2021	0.5	Class Diagram	Camilla Fiumarella, Paolo Guidotti, Chiara Orrigo
17/01/2022	0.6	Design Pattern, Glossario	Sara Cataldo, Paolo Guidotti, Camilla Fiumarella
1/02/2022	0.7	Revisione totale	[Tutti]

Indice

1.Introduzione

Autoshop offre agli eventuali clienti di semplificare l'acquisto di pezzi di ricambio o eventualmente di richiedere un preventivo di autovetture comodamente da casa.

Nella prima sezione del documento verranno descritti i trade-off e le linee guida per la fase di implementazione, la documentazione e la convenzione sui formati.

1.1 Object design goals

Riusabilità: Il sistema Autoshop si basa sulla riusabilità, attraverso l'utilizzo di Design Pattern per risolvere problemi comuni e per proteggere le classi da futuri cambiamenti, selezionate componenti utili per strutture dati e servizi di base ed ereditarietà.

Robustezza : Il sistema deve essere robusto, deve reagire in maniera corretta a situazioni impreviste attraverso il controllo degli errori e la gestione delle eccezioni, ovvero deve essere capace di sopravvivere ad input non validi immessi dall'utente.

Incapsulamento : Il sistema è in grado di garantire la segretezza sui dettagli implementativi della classi mediante l'utilizzo delle interfacce, rendendo possibile l'utilizzo di funzionalità offerte dai diversi componenti o layer sottoforma di blackbox.

1.2 Object trade-offs

● **Funzionalità vs. Usabilità**: Il sistema consente di prediligere l'usabilità a discapito delle funzionalità previste nella fase di Analisi, in quanto risulta essere di maggiore importanza fornire un sistema user friendly a discapito di operazioni superficiali. Si eviterà di arricchire eccessivamente il numero e la complessità delle funzionalità disponibili, in modo da consentire anche a utenti meno pratici nell'uso della tecnologia di comprendere il funzionamento del sistema e padroneggiare completamente l'uso.

● **Tolleranza agli errori vs. Consistenza dei dati:** Il sistema dovrà fornire una maggiore tolleranza agli errori a discapito della consitenza dei dati. Nel caso in cui il sistema o parte di esso dovesse trovarsi in uno stato di errore, si preferirà interrompere il funzionamento.

● **Usabilità vs. Sicurezza:** entrambi saranno obiettivi cruciali del processo di design, ma sarà opportuno che la semplificazione dell'interfaccia utente non comporti tagli sui vari controlli.

1.3 Components Off The Shelf (COST)

Il Sistema utilizzerà i seguenti componenti off the shelf :

- Bootstrap, un framework per aiutare lo sviluppo delle interfacce grafiche che utilizza HTML, CSS e JS.

1.4 Linee guida per la documentazione dell'interfaccia

Il sistema è multi-utente (può accedervi chiunque, sia un semplice cliente o un dipendente). Al cliente, il sistema nasconde la logica delle operazioni, fornendogli solamente la consultazione del catalogo dei prodotti e la possibilità di acquistarli. Il dipendente invece deve avere accesso all'interfaccia magazzino e gestione dipendenti. Il dipendente riceve i risultati delle richieste effettuate dal sistema, senza preoccuparsi delle operazioni e di come queste sono state effettuate. Avvenuta l'operazione di vendita di uno o più prodotti il sistema decrementa dal DataBase totale il numero delle quantità vendute del prodotto acquistato (aggiorna quindi il DataBase totale).

Per rendere il codice più estensibile e manutenibile, prima dell'implementazione della logica del sistema è opportuno stabilire delle convenzioni da seguire nel corso dell'implementazione. È stato scelto di utilizzare Checkstyle per verificare se il codice è conforme alle regole di codifica.

Commenti
Si farà uso di commenti per aggiungere informazioni accessorie che facilitino la comprensione del codice, eventualmente giustificando decisioni particolari o determinate scelte algoritmiche. Il lavoro di documentazione sarà coadiuvato dall'utilizzo di Javadoc, con l'ausilio della funzione di autogenerazione di Eclipse (Generate Javadoc).

Dichiarazioni
Le dichiarazioni delle variabili dovranno essere posizionate all'inizio del blocco di codice in cui le si utilizza. Dichiarare le variabili solo al momento del loro primo uso può infatti ostacolare la comprensione del codice.

Indentazione
L'indentazione dovrà essere effettuata con TAB e, a prescindere dal linguaggio usato per la produzione del codice, ogni istruzione dovrà essere opportunamente indentata.
Es.

```
<html>
        <html>
        </head>

        <body>
        </body>
</html>
```

Parentesi
Si riporterà il blocco di istruzioni che seguono un IF, un FOR un WHILE tra parentesi graffe, anche nel caso vi fosse una sola istruzione.

Script Javascript
Nel caso vi fosse la necessità di corredare le pagine con degli script in linguaggio Javascript per ampliare le funzionalità, tali script saranno collocati in file separati per agevolare la documentazione e il riuso del codice. Funzioni e oggetti Javascript dovranno essere preceduti da un commento in stile Javadoc che li descriva e li documenti.

1.5 Definizioni, acronimi e abbreviazioni

Acronimi	Descrizione
HTML	acronimo di Hyper Text Markup Language, un linguaggio usato per definire la struttura di una pagina web;
CSS	acronimo di “Cascading Style Sheets”, un linguaggio di usato per definire lo stile e la formattazione di una pagina web;
JS	acronimo di JavaScript, un linguaggio di scripting utilizzato nelle pagine web per fornire dinamicità e logica a queste ultime;

1.6 Riferimenti

- Bruegge, Dutoit, Object-Oriented Software Engineering.
- Di seguito una lista di riferimenti ad altri documenti utili durante la lettura:
  1 Statement Of Work;
  2 Requirements Analysis Document;
  3 System Design Document;
  4 Object Design Document;
  5 Test Plan;
  6 Manuale di installazione;
  7 Manuale utente;
  8 Matrice di tracciabilità;

2 .Packages

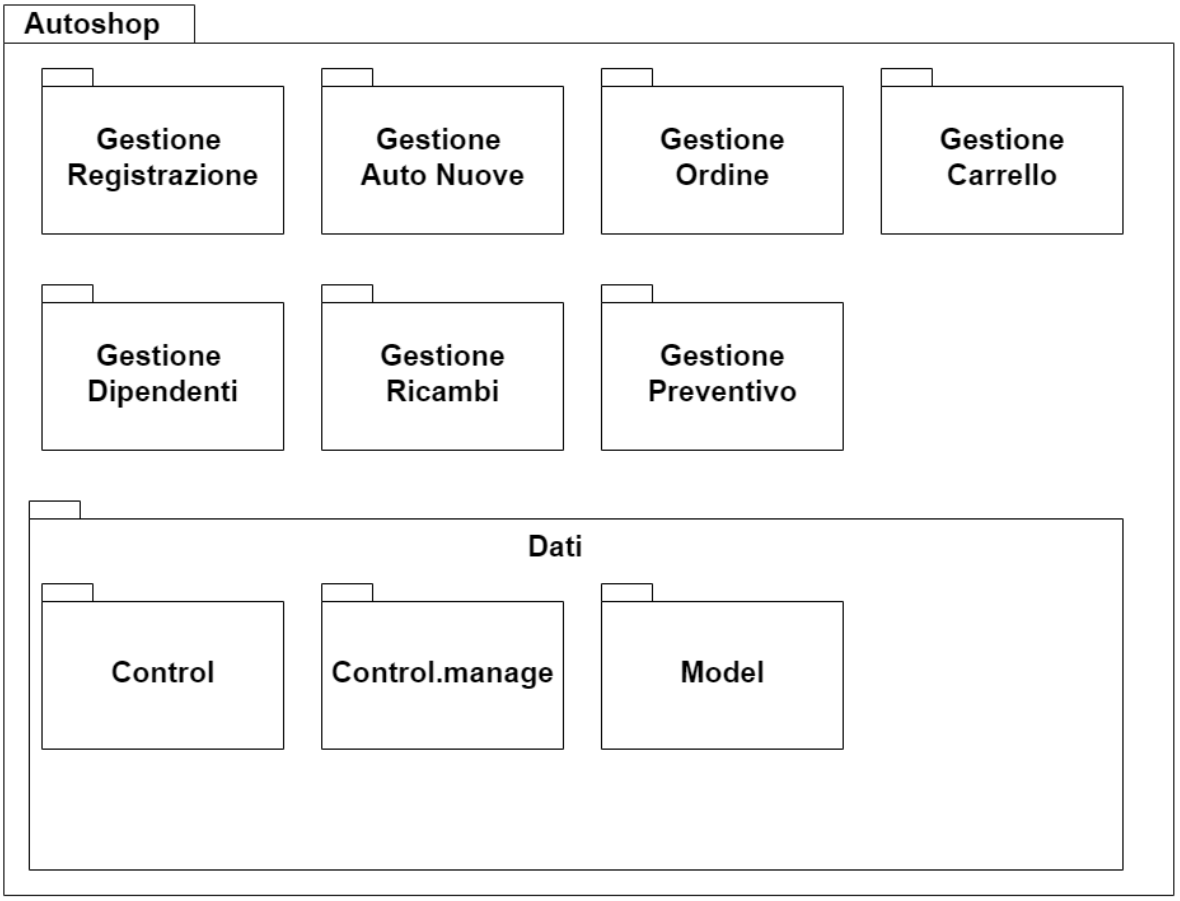
In questa sezione viene mostrata la suddivisione del sistema in package, in base a quanto definito nel documento di System Design. Tale suddivisione è motivata dalle scelte architetturali prese precedentemente.

- src, contiene tutti i file sorgente
  - control, contiene tutte le servlet.
  - control.interfaccia, contiene le interfacce pubbliche
  - control.manage, contiene le classi che permettono di gestire la connessione con il db.
  - model, sono contenute le classi Java.
  - test, contiene le classi Java per l'implementazione del testing.

- WebContent, contiene i file relativi alle componenti Interface
  - css, contiene i fogli di stile CSS
  - js contiene i fogli javascript
  - WEB-INF, contiene i file .jsp

- db, contiene tutti i dati del database

3. Class Interface



Di seguito saranno presentate le interfacce di alcuni package.

Javadoc di Autoshop

Per motivi di leggibilità abbiamo optato di creare un sito, hostato tramite GitHub pages, contenente la JavaDoc di Autoshop. In tale maniera, chiunque può consultare la documentazione aggiornata dell'intero sistema.

Di seguito, il link al sito in questione: **https://github.com/kern3lpanic98/Autoshop.git**

3.1.1 Gestione Carrello

Nome classe	modelCarrello
Descrizione	Questa classe permette di gestire le operazioni relative al carrello
Metodi	+removeArticleById(int id, String userClient):void +svuotaCarrello(String userClient):void +doSave(Carrello carrello):boolean +doRetrieveByQnt(String userClient, int id):int +doRetrieveByUserClient(String userClient):ArrayList<Carrello>
Invariante di classe	

Nome metodo	+removeArticleById(int id, String userClient)
Descrizione	Questo metodo consente di rimuovere un articolo dal carrello.
Precondizioni	context : CarrelloService : : removeArticleById(id, userClient) pre: ricambi_auto.exist(id) = = true
Post-condizioni	context : CarrelloService : : removeArticleById(id, userClient) post: ricambi_auto.exist(id) = = false and ricambi_auto.size() == @pre.ricambi_auto.size()-1

Nome metodo	+svuotaCarrello(String userClient)
Descrizione	Questo metodo consente di svuotare il carrello.
Precondizioni	/
Post-condizioni	context : CarrelloService : : svuotaCarrello(userClient) post: carUtente.size == 0

Nome metodo	+doSave(Carrello carrello)
Descrizione	Questo metodo consente di effettuare l'update del carrello.
Precondizioni	/
Post-condizioni	context : CarrelloService: :doSave(Carrello carrello) post: model.doSave(car) = = true

Nome metodo	+doRetrieveByQnt(String userClient, int id)
Descrizione	Questo metodo consente di recuperare la quantità di un articolo scelta dal cliente.
Precondizioni	context CarrelloService: :doRetriveByQnt(userClient,id) post: qnt=ricambi.setId(id).getQuantità()>0 and userClient!=null
Post-condizioni	context : CarrelloService: :doRetriveByQnt(userClient,id) post: qnt=ricambi.setId(id).getQuantità()>0

Nome metodo	+doRetrieveByUserClient(String userClient)
Descrizione	Questo metodo consente di recuperare tramite lo user del cliente il relativo carrello.
Precondizioni	context CarrelloService : : doRetrieveByUserClient(String userClient) pre:rs.getString("userClient")!=null
Post-condizioni	context : CarrelloService : : doRetrieveByUserClient(String userClient) post: ArrayList<Carrello> id_ricambi!=null

3.1.2 Gestione Dipendente

Nome classe	modelDip
Descrizione	Questa classe permette di gestire le operazioni relative ai dipendenti.
Metodi	+doSave(Dipendente d):boolean +doRetrieveAll():Collection<Dipendente> +doRetriveByUser(String user):boolean +remove(String user):void
Invariante di classe	/

Nome metodo	+doSave(Dipendente d)
Descrizione	Questo metodo permette di effettuare l'update dei dati del dipendente.
Precondizioni	/
Post-condizioni	context : DipendenteService: :doSave(Dipendente d) post: model.doSave(dip) = = true

Nome metodo	+doRetrieveAll()
Descrizione	Questo metodo consente di recuperare l'elenco dei dipendenti.
Precondizioni	/
Post-condizioni	context : DipendentiService : : doRetriveAll () post : ArrayList<Dipendente> n != null

Nome metodo	+doRetriveByUser(String user)
Descrizione	Questo metodo consente di verificare che esiste il dipendente.
Precondizioni	context : DipendentiService : : doRetriveByUser(user) pre : user != null
Post-condizioni	/

Nome metodo	+remove(String user)
Descrizione	Questo metodo permette di rimuovere un dipendente.
Precondizioni	context : DipendenteService : : remove (user) pre : user != null
Post-condizioni	/

3.1.3 Gestione Auto Nuove

Nome classe	modelloAutoNuove
Descrizione	Questa classe permette di gestire le operazioni relative alle auto nuove.
Metodi	+doRetrieveByKey(int id):Automobile +doRetrieveAll():Collection<Automobile> +doRetrieveByFilter(String marca, String modello):Collection<Automobile>
Invariante di classe	

Nome metodo	+doRetrieveByKey(int id)
Descrizione	Questo metodo consente di recuperare i dettagli di un'automobile corrispondente all'id.
Precondizioni	context : AutoNuoveService : : doRetriveByKey(id) pre : id > 0
Post-condizioni	contex : AutoNuoveService : : doRetriveByKey(id) post : nuove.isInstance(Automobile) == true

Nome metodo	+doRetrieveAll()
Descrizione	Questo metodo consente di visualizzare tutte le automobili.
Precondizioni	/
Post-condizioni	context : AutoNuoveService : : doRetriveAll() post : ArrayList<Automobile> n != null

Nome metodo	+doRetrieveByFilter(String marca, String modello)
Descrizione	Questo metodo permette di recuperare tutte le automobili corrispondenti ad un determinato modello e marca.
Precondizioni	context : AutoNuoveService : : doRetriveByFilter (marca, modello) pre : marca != null and modello != null marca == null and modello !=null marca != null and modello ==null
Post-condizioni	context : AutoNuoveService : : doRetriveByFilter (marca, modello) post : ArrayList<Automobile> n != null

3.1.4 Gestione Preventivo

Nome classe	modelloPreventivo
Descrizione	Questa classe permette di gestire le operazioni relative ai preventivi.
Metodi	+doSave(Preventivo p):boolean
Invariante di classe	

Nome metodo	+doSave(Preventivo p)
Descrizione	Questo metodo permette di effettuare l'update del preventivo.
Precondizioni	/
Post-condizioni	context : PreventivoService: :doSave(Preventivo p) post: model.doSave(dip) = = true

3.1.5 Gestione Ordine

Nome classe	modelOrdine
Descrizione	Questa classe permette di gestire le operazioni relative agli ordini.
Metodi	+doSave(ArrayList<Carrello> c, ArrayList<Integer> id_r, ArrayList<Integer> qnt):boolean
Invariante di classe	

Nome metodo	+doSave(ArrayList<Carrello> c, ArrayList<Integer> id_r, ArrayList<Integer> qnt)
Descrizione	Questo metodo permette di effettuare l'update dell'ordine.
Precondizioni	/
Post-condizioni	context : OrdineService: :doSave(c, id_r, qnt) post: c.exist(carello)==true id_r.exist(int) == true qnt.exist(int) == true

3.1.6 Gestione Registrazione

Nome classe	modelRegister
Descrizione	Questa classe permette di gestire le operazioni relative alla registrazione.
Metodi	+doSave(Client client):boolean +doRetriveByUser(String user):boolean +getIndirizzo(String userClient):ArrayList<String>
Invariante di classe	

Nome metodo	+doSave(Client client)
Descrizione	Questo metodo permette di effettuare l'update dell'utente.
Precondizioni	/
Post-condizioni	context : RegistrazioneService: :doSave(Client client) post: model.doSave(client) = = true

Nome metodo	+doRetriveByUser(String user)
Descrizione	Questo metodo consente di recuperare un utente esistente.
Precondizioni	context : RegistrazioneService : :doRetriveByUser(user) pre : user != null
Post-condizioni	/

metodo	+getIndirizzo(String userClient)
Descrizione	Questo metodo dato uno userClient permette di visualizzare il relativo indirizzo.
Precondizioni	context : RegistrazioneService : : getIndirizzo (userClient) pre : userClient != null
Post-condizioni	context : RegistrazioneService : : getIndirizzo (userClient) post : ArrayList <String> indirizzo != null

3.1.7 Gestione Ricambi

Nome classe	modelRicambi
Descrizione	Questa classe permette di gestire le operazioni relative alla registrazione.
Metodi	+doSave(Client client):Collection<RicambiAuto> +doRetrieveByKey(int id):RicambiAuto +doRetrieveAllByArray(ArrayList<Carrello> idr):ArrayList<RicambiAuto> +updateQnt(ArrayList<Integer> id,ArrayList<Integer> qnt):boolean +addQnt(int id,int qnt):boolean +removeRicambi(int id):boolean +addRicambio(RicambiAuto r):boolean
Invariante di classe	

Nome metodo	+doSave(Client client)
Descrizione	Questo metodo permette di effettuare l'update del ricambio.
Precondizioni	/
Post-condizioni	context : RicambiService: :doSave(Client client) post: model.doSave(client) = = true

Nome metodo	+doRetrieveByKey(int id)
Descrizione	Questo metodo consente di recuperare dettagli di un ricambio corrispondente all'id.
Precondizioni	context : RicambioService : : doRetriveByKey(id) pre : id > 0
Post-condizioni	contex : RicambioService : : doRetriveByKey(id) post : ricambio.isInstance(RicambiAuto) == true

Nome metodo	+doRetrieveAllByArray(ArrayList<Carrello> idr)
Descrizione	Questo metodo consente di recuperare i ricambi contenuti nel carrello.
Precondizioni	context : RicambiService : : doRetriveAllByArray(idr) pre : ArrayList<Carrello>idr != null
Post-condizioni	context : RicambiService : : doRetriveAllByArray(idr) post : ArrayList<RicambiAuto> ricambi != null

Nome metodo	+updateQnt(ArrayList<Integer> id,ArrayList<Integer> qnt)
Descrizione	Questo metodo consente di aggiornare la quantità dei ricambi.
Precondizioni	context : RicambiService : : updateQnt(id, qnt) pre : ArrayList <Integer> id != null and ArrayList <Integer> qnt != null
Post-condizioni	context : RicambiService : : updateQnt(id, qnt) post : id.size() != @pre.id.size() and qnt.size() != @pre.qnt.size()

Nome metodo	+addQnt(int id,int qnt)
Descrizione	Questo metodo consente di aggiungere una determinata quantità di un corrispettivo pezzo di ricambio.
Precondizioni	context : RicambiService : : addQnt(id, qnt) pre : id >0 and qnt >0
Post-condizioni	context : RicambiService : : addQnt(id, qnt) post : qnt > @pre.qnt

Nome metodo	+removeRicambi(int id)
Descrizione	Questo metodo permette di rimuovere un pezzo di ricambio.
Precondizioni	context : RicambiService : : removeRicambi(id) pre : id >0
Post-condizioni	/

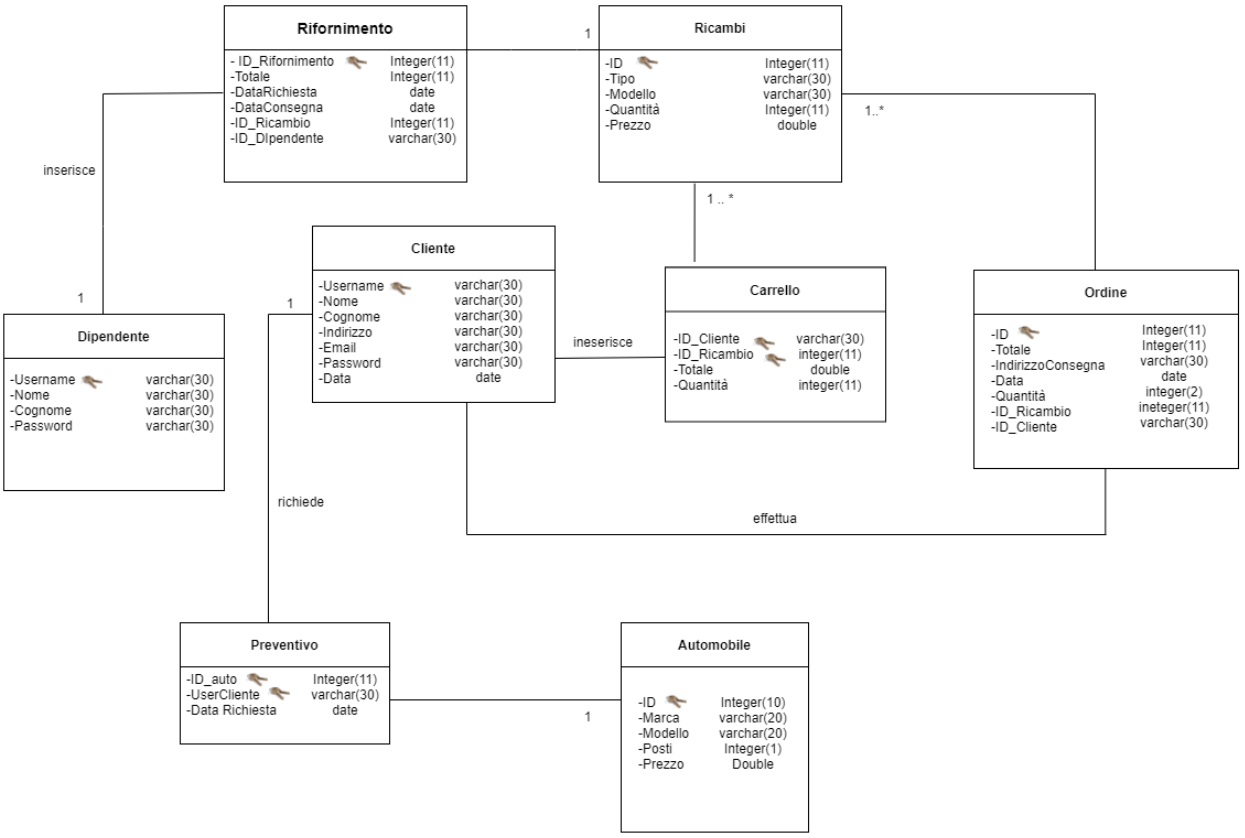
Nome metodo	+addRicambio(RicambiAuto r)
Descrizione	Questo metodo permette di aggiungere un nuovo pezzo di ricambio.
Precondizioni	context : RicambiService : : addRicambio (RicambiAuto r) pre : r.exists() == false
Post-condizioni	/

3.1.8 Gestione Rifornimento

Nome classe	modelRifornimeto
Descrizione	Questa classe permette di gestire le operazioni relative ai rifornimenti.
Metodi	+doSave(Rifornimento r):boolean
Invariante di classe	

Nome metodo	+doSave(Client client)
Descrizione	Questo metodo permette di effettuare l'update del rifornimento.
Precondizioni	/
Post-condizioni	context : RicambiService: :doSave(Client client) post: model.doSave(client) = = true

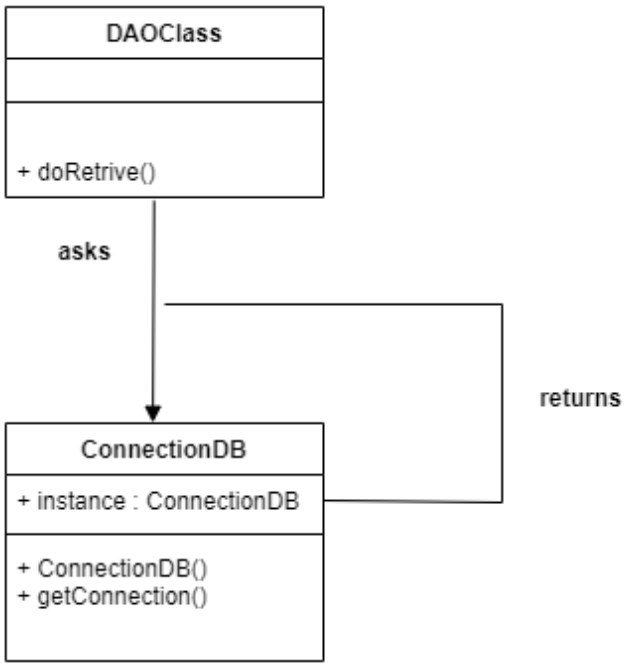
4 .Class Diagram



5. Design Pattern

Nel suddetto paragrafo andremo a specificare i design patterns utilizzati nello sviluppo di Autoshop. Per il pattern, quindi:
- Andrà esplicitata una breve introduzione teorica;
- Il relativo problema che doveva risolvere Autoshop;
- Una breve descrizione di come abbiamo risolto il problema in AutoShop ;
- Un grafico della struttura delle classi che implementano il pattern;

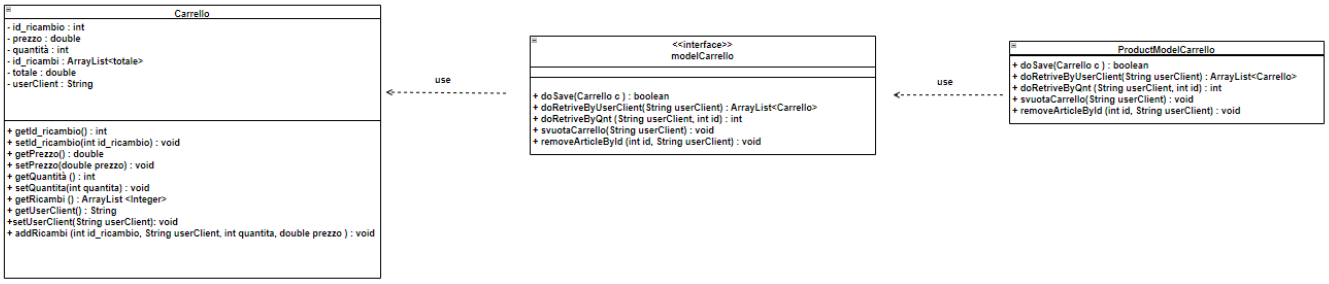
Singleton Pattern
Il singleton è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza. L'implementazione più semplice di questo pattern prevede che la classe fornisca un metodo "getter" statico che restituisce l'istanza della classe (sempre la stessa), creandola preventivamente alla prima chiamata del metodo, e memorizzando il riferimento in un attributo privato anch'esso statico. AutoShop richiede continui accessi a un database, quindi ha bisogno di poter interagire con la stessa istanza del database stesso.



DAOClass nel codice corrisponde al package control.manage

DAO

Un DAO (Data Access Object) è un pattern che offre un'interfaccia astratta per alcuni tipi di database. Mappando le chiamate dell'applicazione allo stato persistente, il DAO fornisce alcune operazioni specifiche sui dati senza esporre i dettagli del database. AutoShop è una web application che presenta un database, quindi ha bisogno di poter interagire con esso in modo rapido e sicuro con query. Per questo motivo abbiamo usato varie interfacce DAO all'interno del nostro sistema.



6. Glossario

Sigla/Termine	Definizione
Package	Raggruppamento di classi ed interfacce.
Javascript	Linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione, in siti web e applicazioni web
Checkstyle	Checkstyle è uno strumento di sviluppo per aiutare i programmatori a scrivere codice Java che aderisce a uno standard di codifica. Automatizza il processo di controllo del codice Java per risparmiare agli umani questo compito noioso (ma importante). Ciò lo rende ideale per i progetti che desiderano applicare uno standard di codifica.
Package Diagram	I package diagram sono diagrammi strutturali utilizzati per mostrare l'organizzazione e la disposizione dei vari elementi del modello sotto forma di pacchetti.