Università degli Studi di Salerno

Corso di Ingegneria del Software

e-Wine

Object Design Document

ODD

Anno 2017/2018

INDICE

1. INTRODUZIONE

1.1 Object design trade-off

1.1.1 Comprensibilità vs costi

Si preferisce aggiungere costi per la documentazione al fine di rendere il codice comprensibile anche alle persone non coinvolte al progetto o le persone coinvolte che non hanno lavorato a quella parte. Commenti diffusi nel codice facilitano la comprensione, di conseguenza, migliorare la comprensibilità, agevola il mantenimento ed il processo di modifica, a discapito però dei costi di mantenimento del sistema.

1.1.2 Portabilità vs Efficienza

La portabilità del sistema e-Wine è garantita dalla scelta del linguaggio di programmazione java. Lo svantaggio dato è nella perdita di efficienza introdotta dal meccanismo della Java Virtual Machine. Tale compromesso è accettabile per i numerosi supporti forniti dal linguaggio java.

1.1.3 Modularità vs Efficienza

La modularità definita nel progetto e-Wine si scontra con l'efficienza nella elaborazione in lato server. La modularità facilita la creazione e la manutenzione del programma, in maniera simile al principio del Divide et Impera, inoltre ci garantisce l'utilizzo del codice in altre applicazioni. Allo stesso tempo riduce l'efficienza dei tempi di risposta dei moduli che si occupano di determinati servizi.

1.2 Interface Document Guidelines

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

• Dichiarazioni del package

• Sezione import

• Dichiarazione di interfaccia o classe:

o Attributi pubblici

o Attributi privati

o Attributi protetti

o Costruttori

o Altri metodi

• Classi interne

1.2.2 Naming

L'utilizzo di convenzioni sui nomi rende il programma più leggibile e comprensibile da parte dei membri del team. In particolare secondo il modello del codice programmato è auspicabile che tutti siano in grado di intervenire su una qualsiasi linea del 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 il nome ha l'iniziale in maiuscolo. I nomi delle classi devono essere semplici e descrittivi evitando l'uso di acronimi e abbreviazioni. Nel caso una o più classi si riferiscano a design patterns noti, è consigliato l'utilizzo di suffissi inglesi che richiamano lo specifico componente del design pattern. È consigliato l'uso della lingua italiana per i nomi, fatta eccezione per nomi inglesi per uso comune (es: testing class).

Metodi

I metodi devono essere verbi, composti anche da più parole con l'iniziali minuscole.

Costanti

I nomi di costanti vengono scritti con tutte le parole in maiuscole. Nel caso di più parole queste vengono separate da un underscore(_).

1.2.3 Uso dei Commenti

E' permesso l'utilizzo di due tipi di commenti:

- Commenti javadoc (aree di testo comprese tra il simbolo /** e */)

- Commenti in stile C (righe delimitate da // se il commento è scritto su un'unica riga), (righe delimitate da /* se il commento è scritto su più righe).

L'utilizzo dei commenti di javadoc è suggerito prima della dichiarazione di: classi e interfacce, costruttori, variabili di classi e metodi. Ogni commento deve specificare le funzionalità e le specifiche del codice, senza esplicitare dettagli legati all'implementazione. In maniera tale da rendere leggibile tale documentazione anche da sviluppatori che non posseggono l'implementazione. I commenti javadoc

consentono la generazione automatica della documentazione del codice attraverso l'utilizzo di tools.

I commenti stile C sono utilizzati all'interno dei metodi per descrivere in maniera sintetica cicli, condizioni o altri passi del codice.

1.2.4 Altre regole di stile

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

• I nomi di package, classi e metodi devono essere nomi descrittivi e di uso comune.

• Evitare l'uso di abbreviazioni di parole.

• Utilizzare dove possibile nomi largamente in uso nella comunità informatica.

• 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 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 l'utilizzo di spazi al posto dei tab.

• È consigliato l'utilizzo dei nomi in italiano, tuttavia e consigliato l'utilizzo di termini in inglese laddove si tratta di uso comune o nel caso di termini comunemente usati nella loro versione inglese. È 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 dei nomi inglesi nel caso si adoperino termini della libreria standard di java.

• Si consiglia l'utilizzo di parti standard dei nomi in casi come:

o Classi astratte, suffisso Abstract (es: AbstractProdotto)

o Design pattern

o Accezioni terminanti per Exception (es: UtenteNonTrovatoException)

• I nomi dell'interfaccia seguono le regole standard, è sconsigliato usare il suffisso o prefisso interface.

• È consigliato l'utilizzo di suffisso standard come get e set in inglese.

• È possibile scrivere dichiarazioni di metodi o classi in più righe se sono eccessivamente lunghe.

• Dichiarare le variabili ad inizio blocco sia per un metodo che per 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 devono essere eseguite in fase di dichiarazione, impostando un valore di default o il risultato di un metodo. Se ciò non è possibile, inizializzare la variabile appena prima del suo utilizzo.

• Allineare la dichiarazione delle variabili per renderle più leggibili.

• Nel caso di algoritmi troppo complessi eseguire un re-factoring per separarlo in diversi sottometodi più semplici.

1.3 Definizioni, acronimi, abbreviazioni

DBMS: Database Management System

Off-the-Shelf: Servizi esterni di cui viene fatto utilizzo da terzi.

Frame work: software di supporto allo sviluppo web.

HTML: linguaggio di markup per pagini web.

CSS: linguaggio usato per definire lo stile delle pagini web.

JavaScript: linguaggio di script orientato agli oggetti e agli eventi. Comunemente utilizzato nella programmazione web lato client per la creazione in siti web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati dall'utente sulla pagina web.

RAD: Documento di Analisi dei Requisiti.

SDD: System Design Document.

SQL: Structured Query Language.

1.4 Riferimenti

➢ RAD_e-Wine: Documento di Analisi dei Requisiti.

➢ SDD_e-Wine: Documento di System Design.

➢ Libro di testo: Object Oriented Software Engineering.

➢ Slides: materiale didattico

1.5 Overview

Nelle sezioni successive sarà descritta l'architettura del sistema e le sue componenti principali. Saranno esposte le tipologie di utenza e i comportamenti 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

Le componenti base che costituiscono il sistema sono raccolti in moduli a loro volta raccolti in livelli. I 3 livelli rappresentano la suddivisione dettata dal modello di architettura preso in considerazione per il sistema e-Wine "MVC" Model View Controller.

Ciascun livello rappresenta un package contenente le componenti relative alle funzioni associate al livello.

- Package source

MODEL

- Carrello_Manager
- Utente_Manager
- Fattura_Manger
- Prodotto_Manager

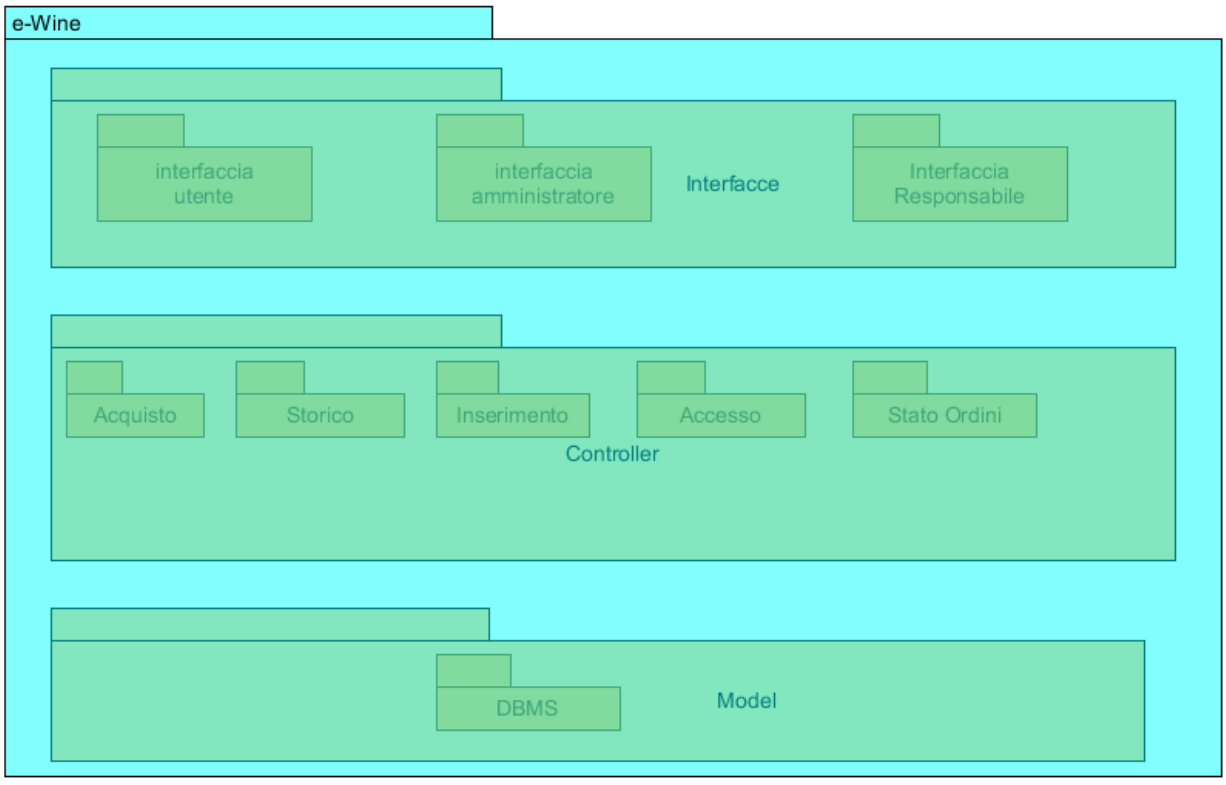
VIEW

- Carrello_Boundary
- Checkout_Boundary
- CheckOutData_Boundary
- CheckOutCredit_Boundary
- Login_Boundary
- RegistrazioneUtente_Boundary

- Store_Boundary
- UserDashBoard_Boundary
- Prodotti_Boundary

CONTROLLER

- CheckOut_Controller
- Carrello_Controller
- RegistrazioneUtente_Controller
- Logout_Controller
- Login_Controller
- Storico_Controller
- AggiornaStato_Controller

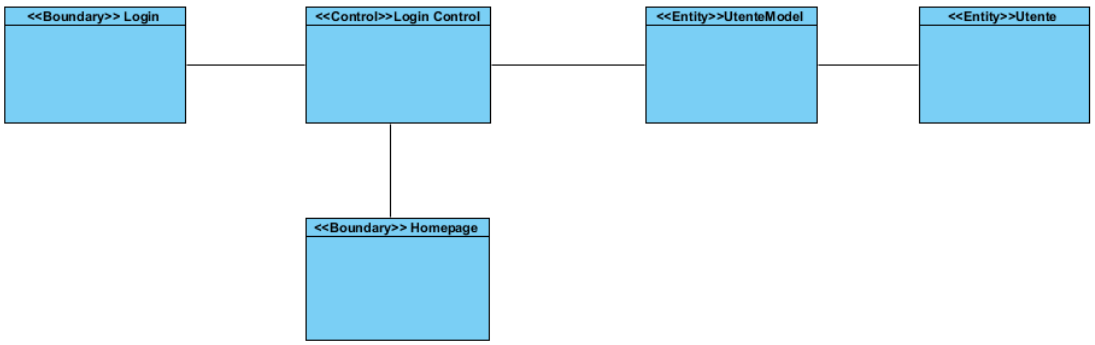


3 INTERFACCE DELLE CLASSI

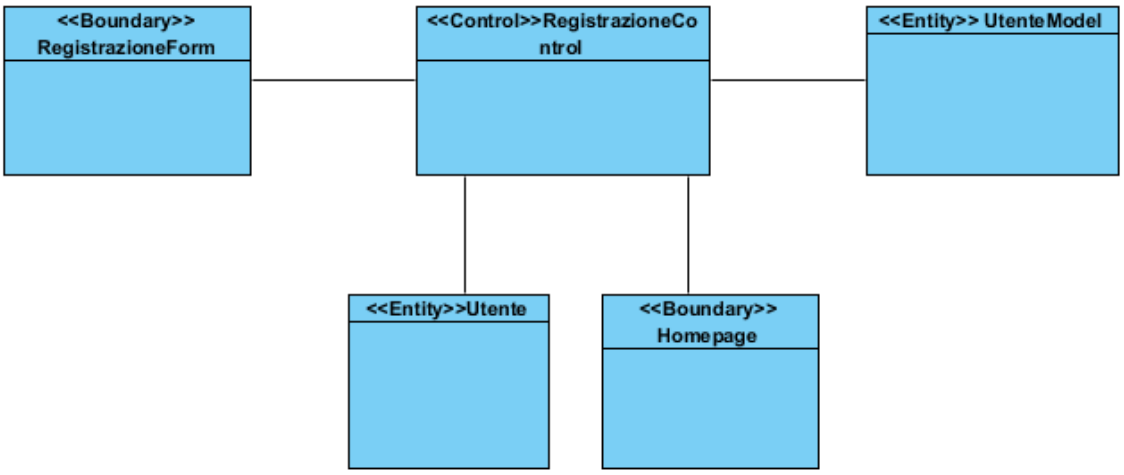
Si procede all'analisi dettagliata delle classi implementate nel sistema. L'analisi serve ad evidenziare le interfacce di interazione utilizzate nella progettazione del software.

3.1 Class Diagram

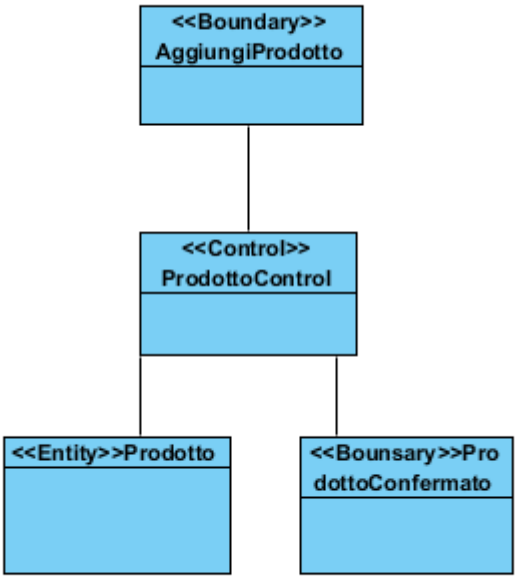
3.1.1 *Accesso*



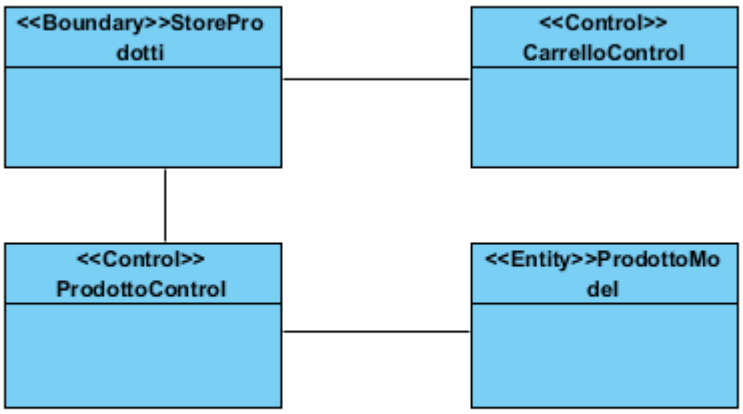
3.1.2 Registrazione



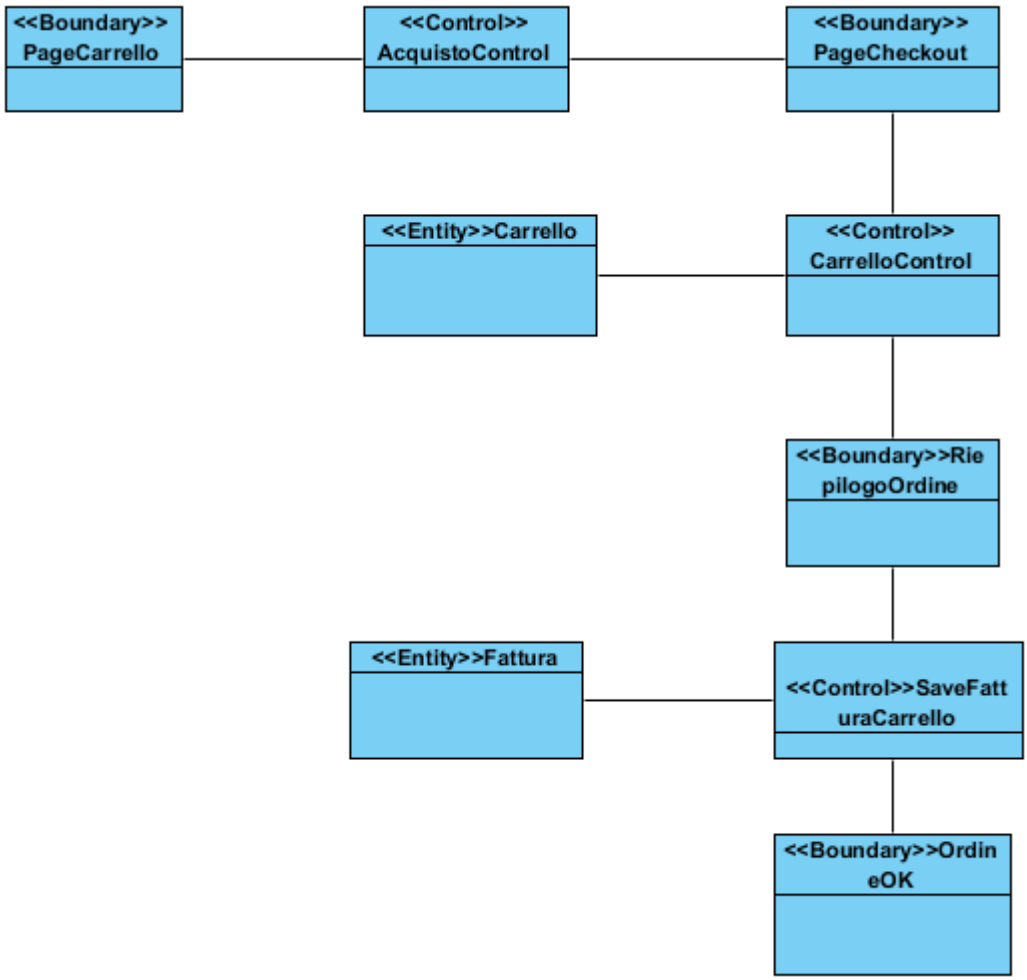
3.1.3 Inserimento Prodotto



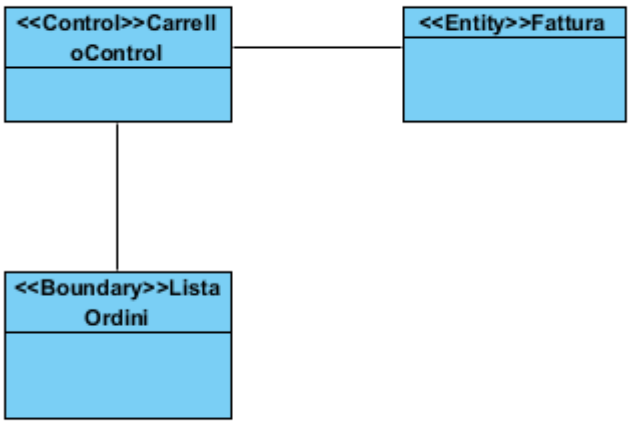
3.1.4 Aggiungi Prodotto Carrello



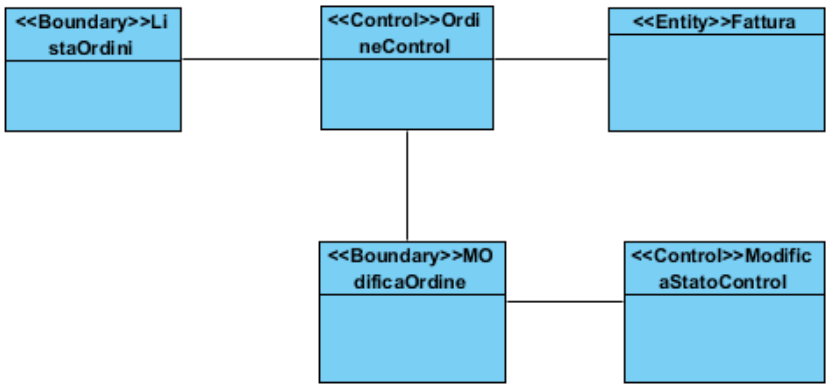
3.1.5 Acquisto Prodotto



3.1.6 Visualizza Ordini

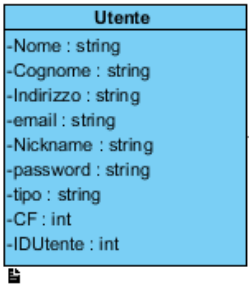


3.1.7 Modifica Stato Ordine

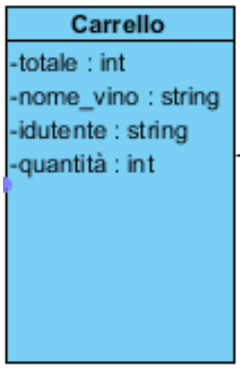


3.2 Descrizione delle Classi

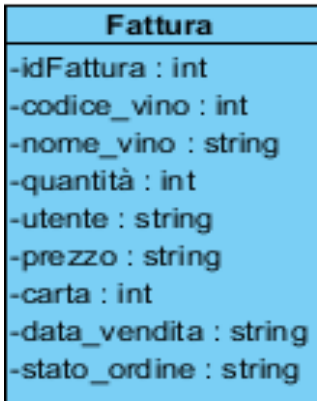
3.2.1 Utente



3.2.2 Carrello



3.2.3 Fattura



3.2.4 Prodotto