Shop Articoli Sportivi

Nome: Marco

Cognome: Vignini

Matricola: 5444851

Data di consegna: 19/12/2014

CFU: 9

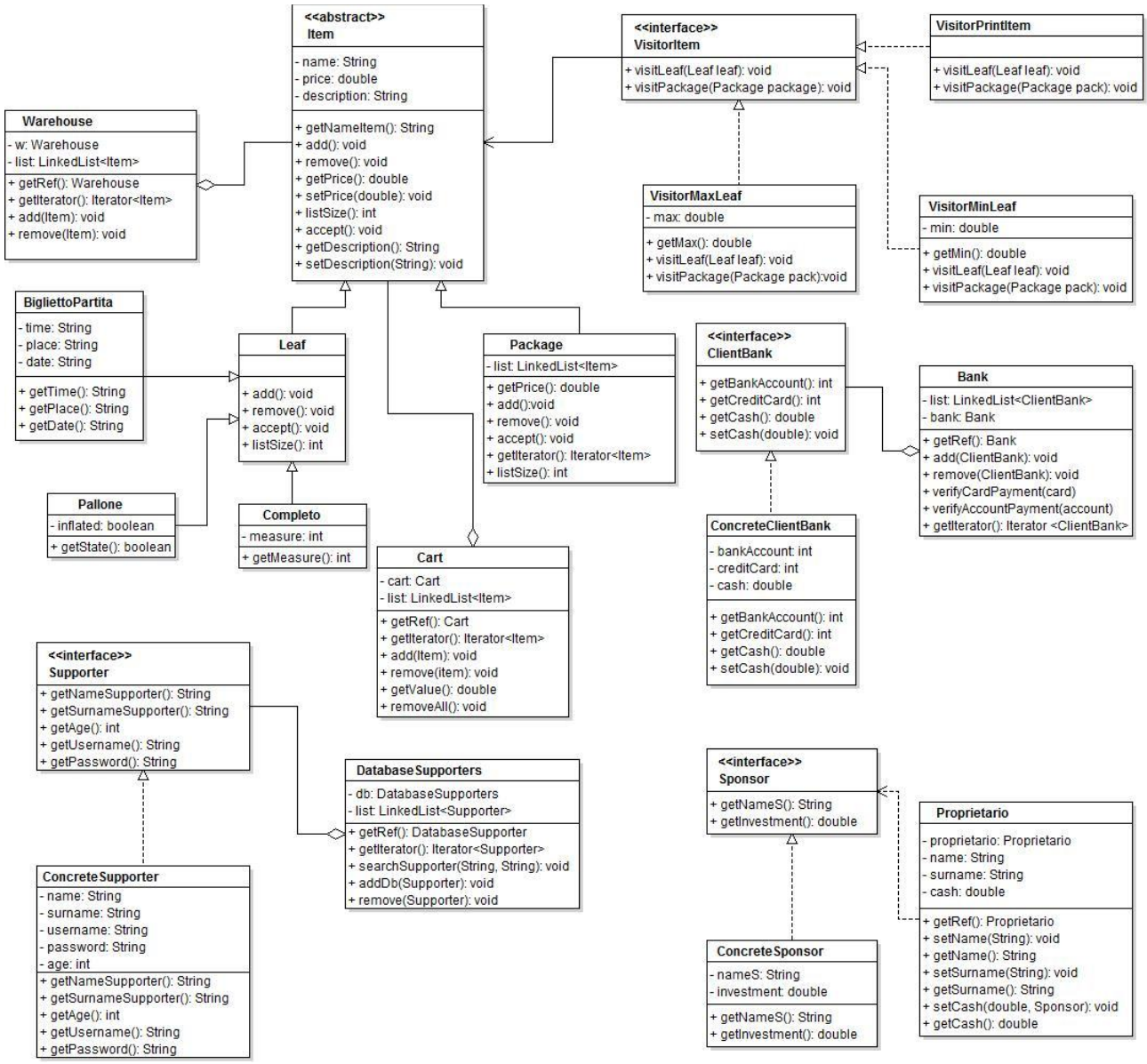
E-mail: marco.vignini@stud.unifi.it

A.a 2014-2015

Specifiche

Ho realizzato un negozio online che vende articoli sportivi (item) di un club calcistico, tali articoli posso essere singoli (leaf) o pacchetti (package). In particolare si occupa di vendere: completi dei giocatori, palloni e biglietti per le partite… Il prezzo degli articoli singoli è già definito mentre quello per i pacchetti è ottenuto dalla somma di tutti gli articoli che compongono tale pacchetto. Vengono fornite delle statistiche dei vari articoli singoli: nomi di tali articoli, prezzo minimo e massimo. Il negozio è stato ereditato dall'attuale proprietario dal padre quindi non prevede di mettersi in società con nessuno per una questione affettiva(proprietario unico). Possono inoltre entrare degli sponsor che mettono un investimento nelle casse del proprietario. Il tifoso (registrato nel database del club) che vuole comprare dal catalogo avrà a disposizione la gli articoli conservati nel magazzino e potrà farsi un carrello della spesa. Successivamente potrà scegliere la modalità di pagamento, ovvero carta di credito o account della banca…

Diagramma Uml



Design/Implementazione - Articoli (Composite)

Da un punto di vista di progettativo - implementativo si è utilizzato il design pattern Composite. Si ha una classe astratta chiamata Item la quale fattorizza gli elementi in comune agli articoli singoli e al pacchetto, infatti sia che gli articoli singoli (Leaf) che i pacchetti (Package) hanno un metodo getNameItem() e getPrice() che non differisce a livello implementativo tra Leaf e Package, setPrice() invece serve per tenere aggiornato il prezzo dei package (che è frutto della somma dei prezzi). Si sfrutta la proprietà ricorsiva del composite stesso, in Package è stata fatta una LinkedList di Item utilizzando i Generics (non c'è distinzione tra gli articoli nel component del Composite). Quando viene calcolato il prezzo del Package si usa per l'iterazione il pattern Iterator il quale garantisce estensibilità del codice permettendo a classi esterne di poter accedere a tali oggetti… La classe Leaf è frutto del continuo refactoring del codice che ha raggruppato le operazioni comuni di tutti gli articoli singoli, ovvero: add(), remove(), listSize() e accept(VisitorItem v). Poi ovviamente tali classi hanno le loro operazioni specifiche. I metodi add(), remove(), listSize() hanno il lancio dell'eccezione per il caso in cui vengano richiamati su una leaf (non avrebbero senso).

Design/Implementazione - Statistiche (Visitor)

Il design pattern Visitor è stato utilizzato per visitare e stampare alcune statiche interessanti riguardanti più nello specifico le Leaf ma anche un po' i Package, infatti grazie al metodo accept(VisitorItem v) implementato sia nelle Leaf che nei Package, VisitorPrintItem stampa i nomi delle Leaf e dei Package stessi (si usa anche qui l'iterator), VisitorMaxLeaf si occupa di restituire il massimo tra le Leaf, VisitorMinLeaf si occupa di restituire il minimo tra le Leaf.

Design /Implementazione - Proprietario del negozio (Singleton) - Sponsor

Il proprietario ha le sue caratteristiche ed è una classe singleton (un solo esemplare di Proprietario viste le specifiche) quindi con costruttore privato e metodi e variabili statiche in modo da poter accedere alla classe stessa con Proprietario.getRef()… Il proprietario riceve personalmente i finanziamenti dagli sponsor che vanno nelle sue tasche. La rappresentazione dello sponsor è stata fatta rispettando il principio di DIP (Dependency Inversion Principle), infatti la classe Singleton riceve Sponsor e non concrete Sponsor(la parte dei servizi offerti è separata da quella di implementazione).

Design/Implementazione – Carrello (Singleton)

Il carrello (Linkedlist<Item>) ovviamente offre possibilità di aggiunta e di rimozione degli articoli da esso, fa la somma del prezzo di tutti gli articoli e permette la cancellazione totale in una volta sola di tutti gli articoli che erano stati precedentemente inseriti. Esso è una classe Singleton, ovvero si può avere una sola istanza di questa classe che può essere richiamata all'esterno grazie al GetRef().

Design/Implementazione – DatabaseDeiTifosi(Singleton) – Tifosi(classe Supporter)

DatabaseSupporters è una classe Singleton che raccoglie i dati dei Supporters e li fa registrare, i Supporters registrati a tale database potranno procedere oltre la pagina di login(si vede nella GUI). Nella classe DatabaseSupporters oltre alla funzionalità più classiche come l'aggiunta del Supporter e la sua rimozione c'è un metodo che lo ricerca nel database stesso al momento del login, ovvero searchSupporter(String username, String password) il quale verifica se le credenziali inserite appartengono ad un qualche Supporter registrato nel database. La classe Supporter oltre che alle informazioni di base e alle credenziali

necessarie offre anche un'informazione sull'età del tifoso… Un domani qualcuno potrebbe decidere di applicare degli sconti basati sull'età del tifoso stesso, magari dinamicamente con un pattern Decorator.

Design/ Implementazione – Banca(Singleton) – ClienteDellaBanca(classe ClientBank)

La classe Bank è Singleton dato che vogliamo avere un solo esemplare della banca. Questa classe si occupa oltre che di aggiungere/rimuovere ClientBank anche di aggiornare i conti dei clienti con due metodi diversi i quali tengono in considerazione quale metodo di pagamento sia stato scelto. ClientBank rappresenta tutte le informazioni del cliente della banca, come i numeri di conto e quello di carta.

Interfaccia grafica (Template)

A livello grafico(impiego di java swing) è stato rappresentato il login del tifoso che è registrato al database del club calcistico, poi una volta entrato può costruirsi il proprio carrello(e gestirlo con aggiunte o rimozioni) scegliendo da tutto ciò che è in magazzino (classe Warehouse). Successivamente decide con cosa pagare ed è qui che interviene il pattern Template il quale risponde all'esigenza di lasciare una parte fissa nel frame e poi di fare la differenziazione nel metodo di pagamento perché a seconda della scelta viene aggiornato o l'account della banca o i soldi della carta di credito col metodo different(). Tale pattern rispetta il principio di Hollywood: "Non chiamarci, vi chiamiamo noi", ovvero è la superclasse che chiama le classi figlie e non viceversa.

Uso della grafica

Iniziare dalla classe Start. Per prima cosa apparirà una finestra che richiederà l'autenticazione del tifoso nel database, una volta fatto il login si va ad una finestra che richiederà se si vuol fare una nuova spesa o se si vuol uscire dal programma. Se facciamo una nuova spesa si aprirà la finestra nella quale a sinistra si avrà a disposizione l'elenco degli articoli disponibili. Con un click del mouse sull'articolo interessato compariranno il nome, il prezzo e la descrizione dell'articolo stesso, mentre con un doppio click avverrà l'inserimento nel carrello. Il carrello è visualizzabile con un click sull'icona in alto a destra, da lì si potranno togliere articoli da esso grazie a "delete" ed a "delete All". Una volta deciso come vogliano che sia composto il nostro carrello, possiamo premere il tasto "fine" che si trova sotto all'elenco degli articoli disponibili. A questo punto si arriva alla scelta di pagamento come indicato nelle specifiche e quindi si seleziona il metodo che preferiamo, dopodiché dovranno essere inserite le credenziali (numero di carta o numero di conto a seconda della scelta) e se premiamo su "Conferma" effettueremo l'acquisto e il nostro conto/ saldo della carta verrà aggiornato di conseguenza.

Junit Testing

Il codice è stato testato grazie a Junit, sono stati utilizzati principalmente i metodi: AssertEquals() che serve a confrontare due interi o due stringhe, AssertTrue() per verificare che una condizione booleana fosse vera, AssertNull() e AssertNotNull() che verificano se l'oggetto specificato esiste oppure no.

Metodo di sviluppo del progetto

Ho raccolto l'idea di questo progetto alla fine della lezione sul pattern Composite dove è stato proposto il negozio online come prova di progettazione, poi durante il susseguirsi delle lezioni il progetto è stato arricchito di idee, quindi poi sono stati aggiunti per esigenza altri pattern, è stato fatto continuo refactoring del codice e si è tenuto conto dei principi della programmazione agile; come: "Don't talk to Strangers", Dependency Inversion Principle(DIP) e Single- Responsability Principle(SRP).