Testo Esercizio

Un negozio di musica vende anche libri e riviste musicali. Si intende automatizzare l'intero processo, dall'approvvigionamento alla vendita. Si analizzino i requisiti e se ne rappresentino i risultati in UML identificando le principali gerarchie "IS-A".
Si rappresenti mediante un diagramma di sequenza la procedura di verifica della disponibilità ed eventuale riordino di un articolo mancante.

Sommario

- Parte 1(vista esterna, aspetti use case driven) – rappresentazione dei requisiti in UML;
- Parte 2 (vista logica, aspetti strutturali) – rappresentazione delle gerarchie e delle classi principali del sistema in UML;
- Parte 3 (vista logica, aspetti dinamici) – rappresentazione di uno scenario d'utilizzo del sistema mediante diagrammi di sequenza.

Note.
La soluzione proposta è solo una possibile soluzione. In generale, non esiste la "soluzione corretta" o il "modello migliore". I modelli si costruiscono esattamente come avviene per qualsiasi artefatto umano. Molti dettagli possono essere soggettivi e dipendono fondamentalmente dalla capacità di astrazione, dallo spirito di osservazione e dalla creatività dell'autore. Questa proposta di soluzione rappresenta solo una linea guida: essa costituisce un invito a sperimentare, ad essere creativi, ma allo stesso tempo ad essere coerenti. Gli aspetti fondamentali di un modello UML sono i seguenti:

- Individuare i concetti essenziali del sistema e rappresentarli opportunamente;
- Caratterizzare ciascun concetto con il livello di dettaglio utile per risolvere il problema;
- Mantenere la coerenza tra le diverse viste (o prospettive) del modello. Se mostriamo un'interazione tra due oggetti in un diagramma di sequenza, per esempio, il metodo descritto nel diagramma deve essere stato prima definito nella classe, con gli opportuni argomenti (se ce ne sono).

Un modello è ragionevole quando contiene queste tre caratteristiche.

Svolgimento

Parte 1

Definiamo innanzitutto gli attori principali del sistema. Due attori ovvi sono il fornitore e il negoziante. Assumiamo che l'utente interagisca direttamente con il negoziante (e non, ad esempio, con uno terminale elettronico automatico). Sotto questa ipotesi l'utente non è un attore del sistema. Identifichiamo ora i requisiti. I due requisiti principali richiesti dall'esercizio sono la verifica della disponibilità di in prodotto in magazzino (caso d'uso *VerificaDispProdotto*) e l'effettuazione di un ordine al fornitore qualora la disponibilità non sia sufficiente (caso d'uso *EffettuaOrdineFornitore*). L'effettuazione di un ordine può essere scomposta nelle seguenti attività:

- Creazione di un ordine al fornitore (inizialmente vuoto);
- Aggiunta del (o dei) prodotto(i) da ordinare;
- Invio dell'ordine al fornitore.

Poiché ogni volta che un ordine viene effettuato sono necessarie tutte queste tre attività, utilizziamo la relazione «include» per legare il caso d'uso *EffettuaOrdineFornitore* con i casi d'uso *CreaOrdineFornitore*, *AggiungiProdottoInOrdine* e *InviaOrdineFornitore*.

Questi requisiti sono i requisiti indispensabili richiesti dall'esercizio. Possiamo tuttavia immaginare altri requisiti. Pensando alle operazioni che si possono compiere sul terminale del magazzino, oltre alla verifica della disponibilità di un prodotto è ragionevole prevedere anche l'eventuale aggiornamento dell'inventario in seguito ad un nuovo ordine pervenuto dal fornitore. In questo caso può essere utile generalizzare queste attività creando un caso d'uso generico *EffettuaOperazioniSulTerminaleMagazzino* che viene poi specializzato nei casi d'uso *VerificaDispProdotto* e *AggiornamentoInventarioMagazzino*. La Figura 1 illustra i casi d'uso identificati.

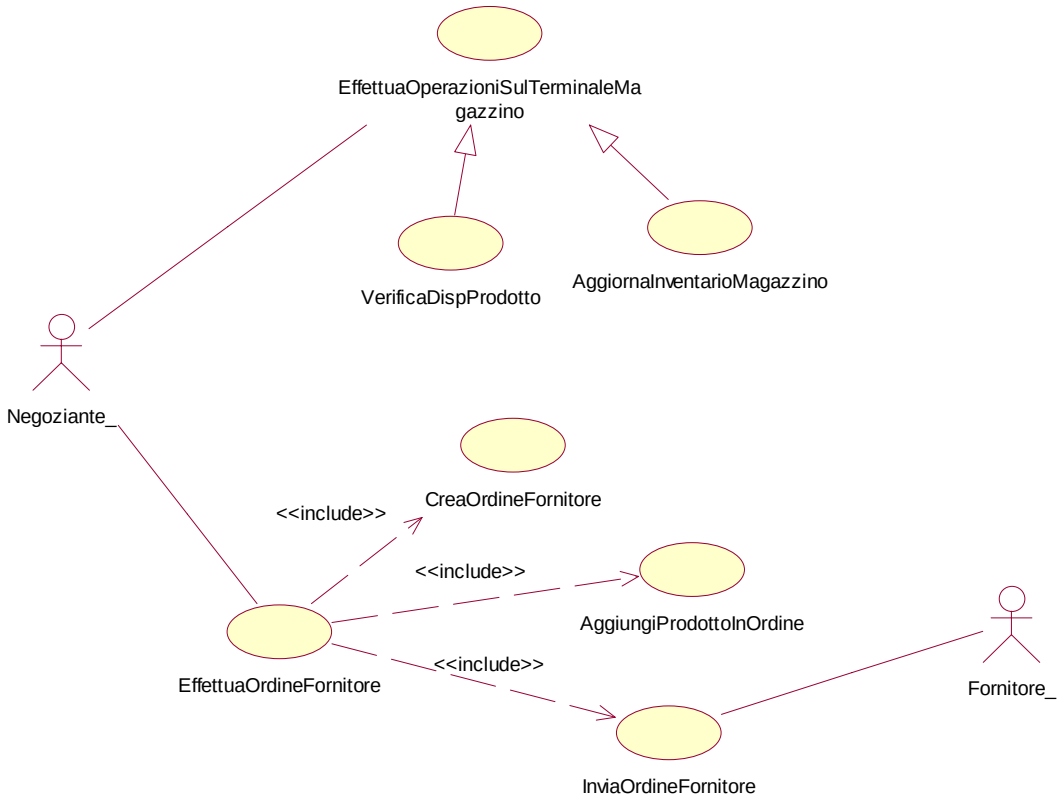

Figura 1 - Diagramma di alcuni casi d'uso per l'esercizio proposto

Parte 2

Il primo passo per costruire dei diagrammi di classe consiste nell'identificare le principali classi che costituiscono il sistema. Dall'analisi linguistica del testo identifichiamo subito il concetto generale di prodotto (un buon nome alternativo è articolo). Il negozio in particolare deve vendere i seguenti tipi di prodotto:

- Riviste musicali;
- Libri;
- Articoli musicali tipici di un negozio di musica quali, ad esempio, cd e musicassette.

Ognuna di queste entità deve essere rappresentata mediante una classe derivata nella gerarchia *Prodotto*. Supponiamo che per automatizzare il negozio si descriva ogni prodotto mediante un codice e mediante una descrizione, utile per eventuali operazioni di ricerca da un terminale elettronico. Ad ogni prodotto, inoltre, è associato un prezzo. Le classi derivate sono definite in modo del tutto analogo, osservando le caratteristiche più tipiche di ciascuna di essa nel mondo reale.

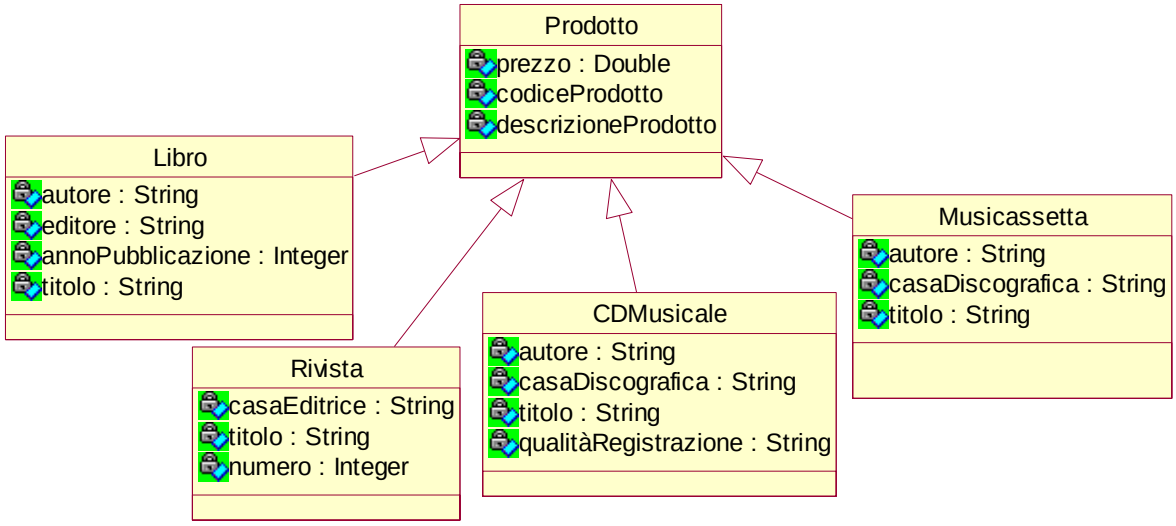

Figura 2 - Gerarchia Prodotto

Osservando la gerarchia di Figura 2 si potrebbe obiettare che l'attributo titolo, essendo dichiarato in tutte le classi derivate, potrebbe essere definito direttamente nella classe base. Si tratta di una considerazione corretta se il negozio vende esclusivamente i prodotti descritti dal diagramma. Se però ci sono altri prodotti in vendita per i quali il concetto di titolo non ha senso (strumenti musicali?), allora è più opportuno lasciare la sua definizione a livello di classi derivate. Questa scelta, inoltre, rende la gerarchia *Prodotto* estendibile: se in futuro il negoziante decidesse di vendere degli strumenti musicali, non dovremmo modificare *Prodotto* per togliere l'attributo titolo.

Un altro concetto molto importante nell'automazione del negozio è quello di ordine. Possiamo pensare ad un ordine come ad un documento che contiene uno o più "item", ciascuno delle quali descrive la richiesta d'ordine di un particolare prodotto e la relativa quantità da ordinare. In questo modo riusciamo ad esprimere quantità differenti per prodotti diversi in un solo ordine (come accade nella realtà). Chiamiamo questi item con il termine di *ProdottoInOrdine* (un altro nome per questa entità potrebbe essere *ItemOrdine*). La Figura 3 illustra le relazioni esistenti tra un ordine e un generico prodotto. Notiamo anche come un ordine sia fisicamente composto da uno o più prodotti in ordine (uso della composizione), mentre ciascuno di questi, a sua volta, sia riferito ad un prodotto (uso dell'aggregazione, ma poteva essere corretta anche un'associazione «*ProdottoInOrdine* –

rappresenta – *Prodotto*» poteva essere, dove “rappresenta” descrive il ruolo di *ProdottoInOrdine* rispetto a *Prodotto*).

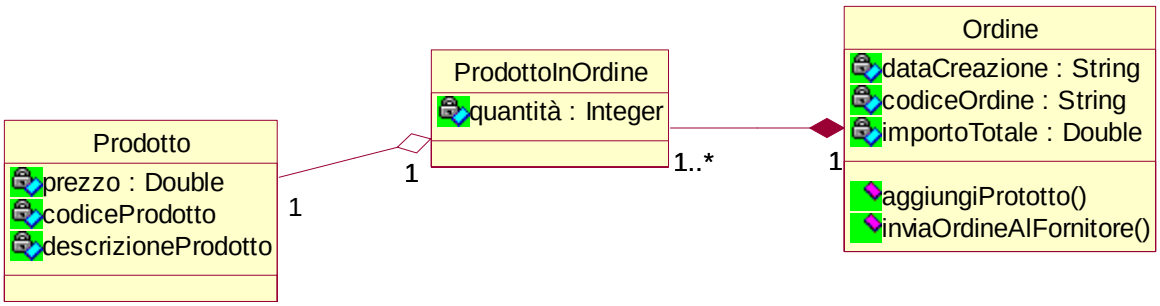

Figura 3 – Le relazioni tra un ordine e un generico prodotto

Definiti i prodotti e gli ordini, possiamo immaginare che il negozio sia costituito da due tipi di terminali elettronici: un gestore degli ordini e un terminale per il magazzino. Il terminale magazzino ha la responsabilità di verificare la disponibilità di un prodotto nel magazzino per la vendita all’utente finale. Il gestore ordini, invece, gestisce la creazione di un ordine da inviare al fornitore qualora il prodotto richiesto dall’utente finale non sia disponibile in magazzino. In Figura 4 descriviamo l’entità negozio composta dai due tipi di terminali e l’entità negoziante che lavora presso il negozio (utilizziamo un’associazione per descrivere il ruolo del negoziante)

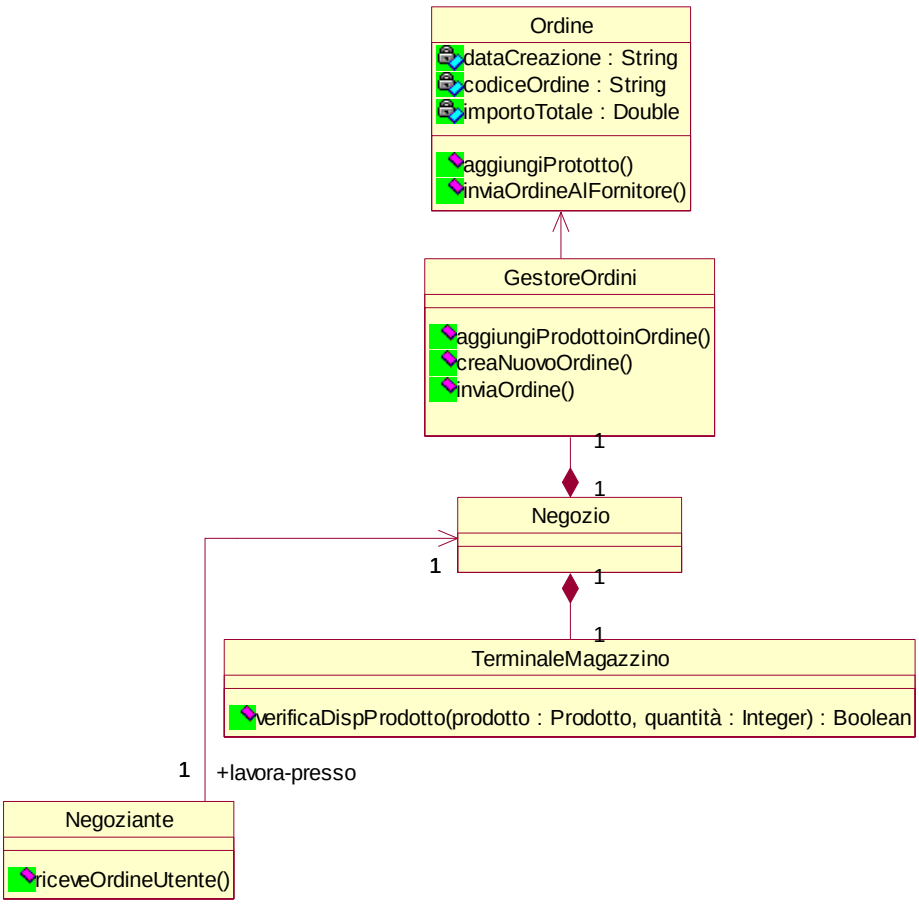

Figura 4 - La gerarchia Figura

Il fatto che un negozio sia composto da due tipi diversi di terminale potrebbe far supporre l'utilizzo dell'ereditarietà, individuando il concetto generale di terminale, specializzato in terminale magazzino e terminale gestione ordini. In questo caso ho preferito la composizione per i seguenti due motivi:

- Un negozio può essere visto come *composto fisicamente* da alcuni terminali;
- Non esiste un'interfaccia comune tra terminale magazzino e gestore ordini;
- Non è evidente alcuna possibilità di utilizzare in modo polimorfico i due tipi di terminali, come invece faremmo per una figura geometrica che ha un'interfaccia comune (il metodo disegna) che può essere sfruttata indipendentemente dal tipo particolare di figura alla quale si fa riferimento.

Identifichiamo ora altre gerarchie. Se pensiamo ad un negozio che deve effettuare degli ordini, una possibilità ragionevole è che esistano diversi fornitori. In questo caso definiamo un fornitore per ogni tipo diverso di prodotto venduto. Avremo quindi la gerarchia di Figura 5:

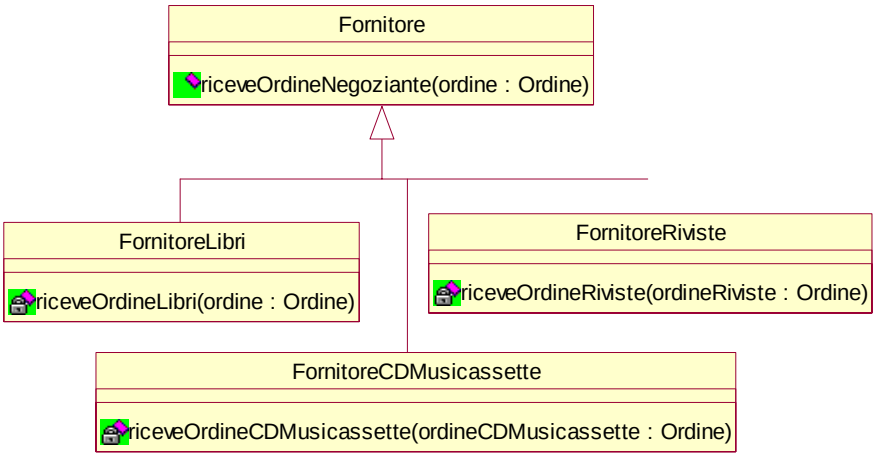

Figura 5 – La gerarchia dei fornitori

A questo punto sembra opportuno legare i concetti di ordine e di fornitore. Per ipotesi semplificativa, assumiamo che ciascun ordine sia associato ad un solo fornitore. Caratterizziamo questa relazione come un'associazione tra le due classi, come illustrato in Figura 6.

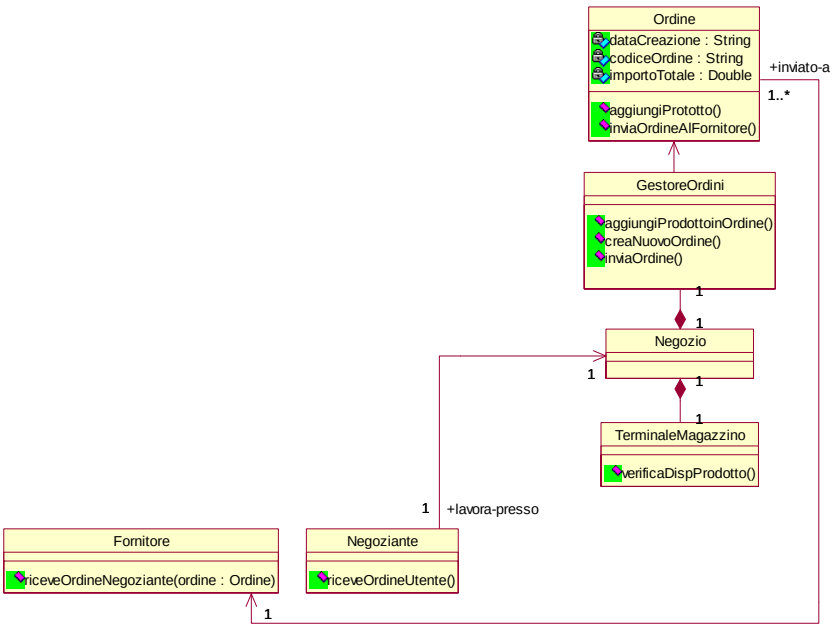

Figura 6 - La GUI costituita da un menu comandi e da una palette delle figure.

Arrivati a questo punto abbiamo identificato i concetti di prodotto, ordine e gestore ordine, negoziante e fornitori, terminale magazzino e gestore ordini. Per completare lo scenario di lavoro, inseriamo anche una classe *Cliente*. Immaginiamo che un cliente, oltre ad avere dei dati anagrafici, abbia anche un codice cliente. Analogamente, potremmo pensare che anche il fornitore abbia un codice simile. Se volessimo caratterizzare clienti, negozianti e fornitori come delle persone aventi un nome, un cognome, un'età eccetera, potremmo creare un'ulteriore gerarchia (Figura 7). In un programma reale difficilmente utilizzeremmo la gerarchia Persona (quale metodo comune potrei utilizzare in modo polimorfico tra clienti, fornitori e negozianti?). Come modello di analisi del dominio, tuttavia, può essere ancora accettabile, quantomeno per non duplicare i dati anagrafici.

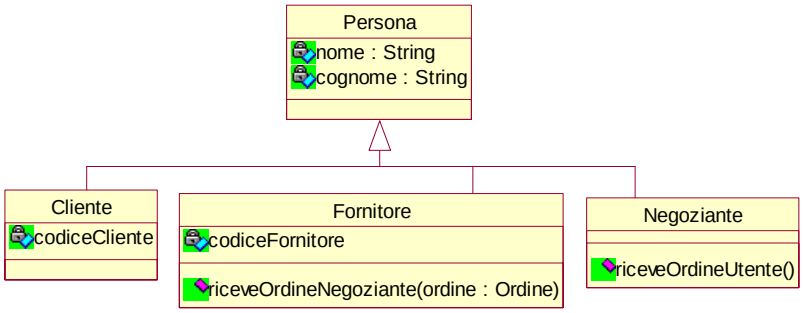

Figura 7 - La gerarchia Persona

Mettiamo tutti i concetti in un unico diagramma e otteniamo la Figura 8.

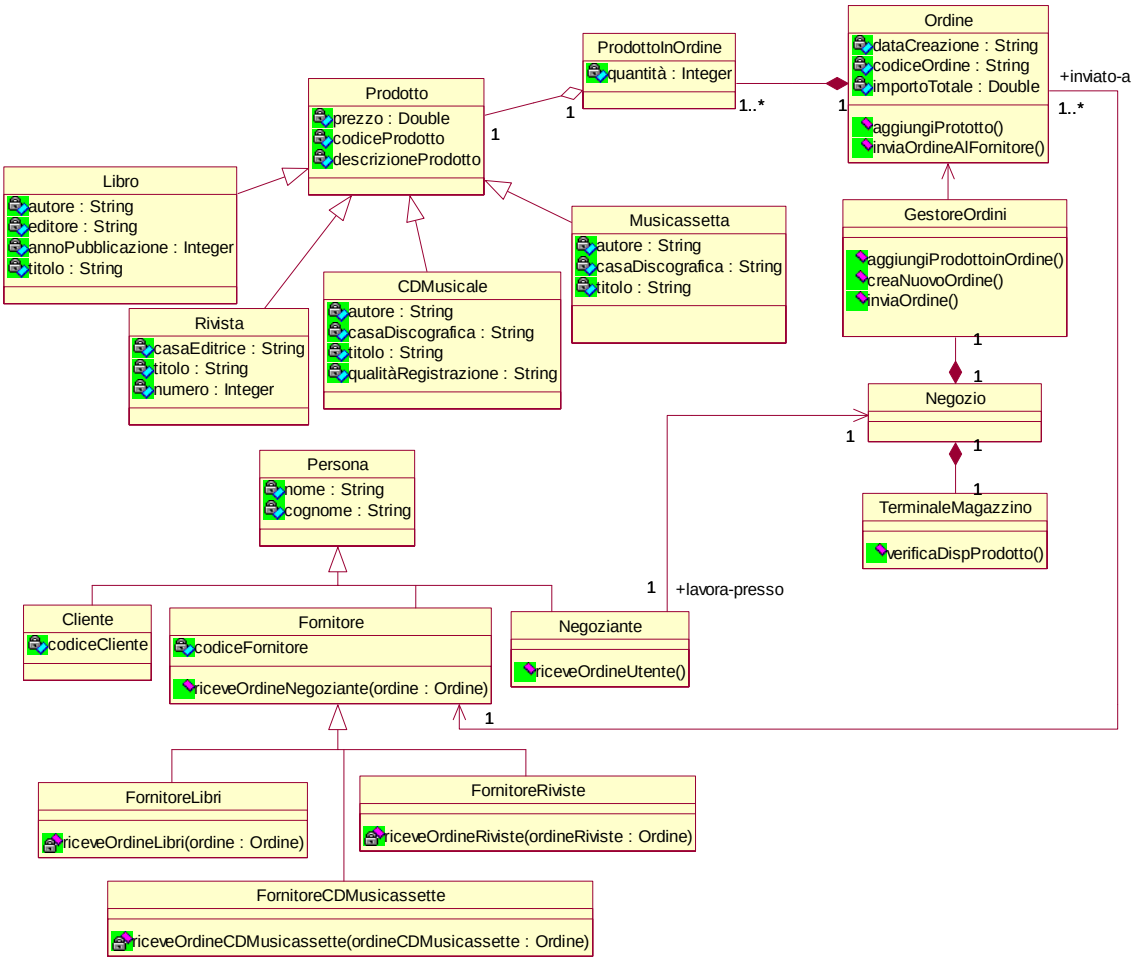

Figura 8 - Tutti gli elementi della soluzione proposta in un diagramma di classe

Parte 3

Realizziamo ora lo scenario richiesto dall'esercizio. Dobbiamo descrivere mediante un diagramma di sequenza la procedura di verifica della disponibilità di un prodotto (richiesto da un utente) e, assumendo che tale prodotto non sia disponibile, effettuare un ordine al fornitore. Come prima operazione, il negoziante riceve l'ordine di un utente che vuole comperare un libro in una certa quantità (è importante non confondere l'ordine dell'utente con l'ordine che il sistema deve compilare e spedire al fornitore). Appena ricevuto l'ordine dell'utente, il negoziante verifica la disponibilità del prodotto in magazzino, invocando il metodo *verificaDispProdotto()* della classe *TerminaleMagazzino*. Nel diagramma di Figura 9 evidenziamo che lo scenario descritto corrisponde al caso in cui non ci sia la disponibilità del prodotto (il metodo *verificaDispProdotto* ritorna false). A questo punto il negoziante utilizza il gestore ordini per creare un nuovo ordine e aggiungere il prodotto da ordinare presso il fornitore di libri.

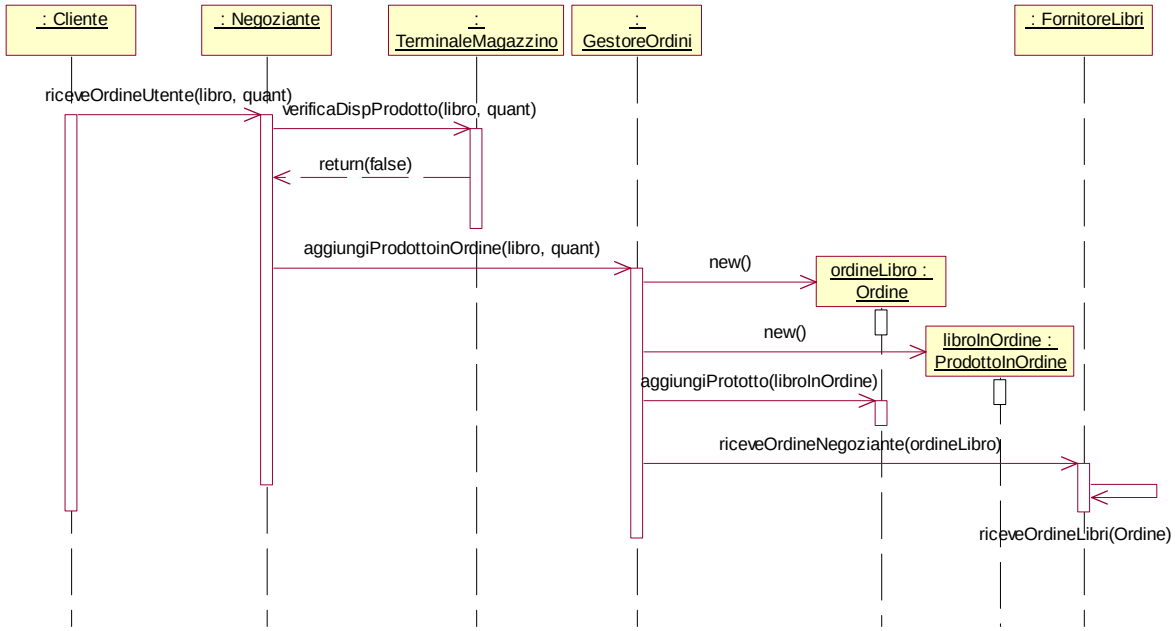

Figura 9 – Creazione di una nuova figura e sua colorazione.

Concludiamo con un'osservazione a proposito delle descrizioni dei metodi invocati dagli oggetti. In un diagramma di sequenza ogni freccia che collega due oggetti deve essere descritta mediante il nome del corrispondente metodo invocato. I nomi dei metodi, a loro volta, devono essere sempre definiti nelle classi degli oggetti coinvolti. Esistono due eccezioni a questa regola:

- La creazione di un nuovo oggetto;
- La restituzione di un oggetto (o un tipo di dato primitivo) come parametro di ritorno di un metodo.

Nel primo caso, non esistendo ancora l'oggetto da creare, non è possibile invocare un suo metodo. La convenzione è quella di descrivere la creazione mediante la chiamata al metodo *new()*, presupponendo che esso sia un metodo fornito dall'ambiente di esecuzione (libreria di sistema, sistema operativo, virtual machine, ect.). Il secondo caso è importante per mostrare quando un oggetto prende parte ad uno scenario e da chi e fornito. La convenzione è quella di descrivere questa situazione mediante lo statement *return(oggetto)*. Questi due casi non devono far supporre

che sulle frecce dei metodi ci si possano scrivere arbitrariamente dei dati (un diagramma di sequenza non è un data flow diagram) o, peggio, delle istruzioni di logica di controllo (if(condizione) else). Per descrivere gli algoritmi esistono diagrammi UML più indicati come, ad esempio, gli activity diagram.