Ingegneria del Software a.a. 2013-14
BOZZA DI SOLUZIONI Prova Scritta del 8 Luglio 2014

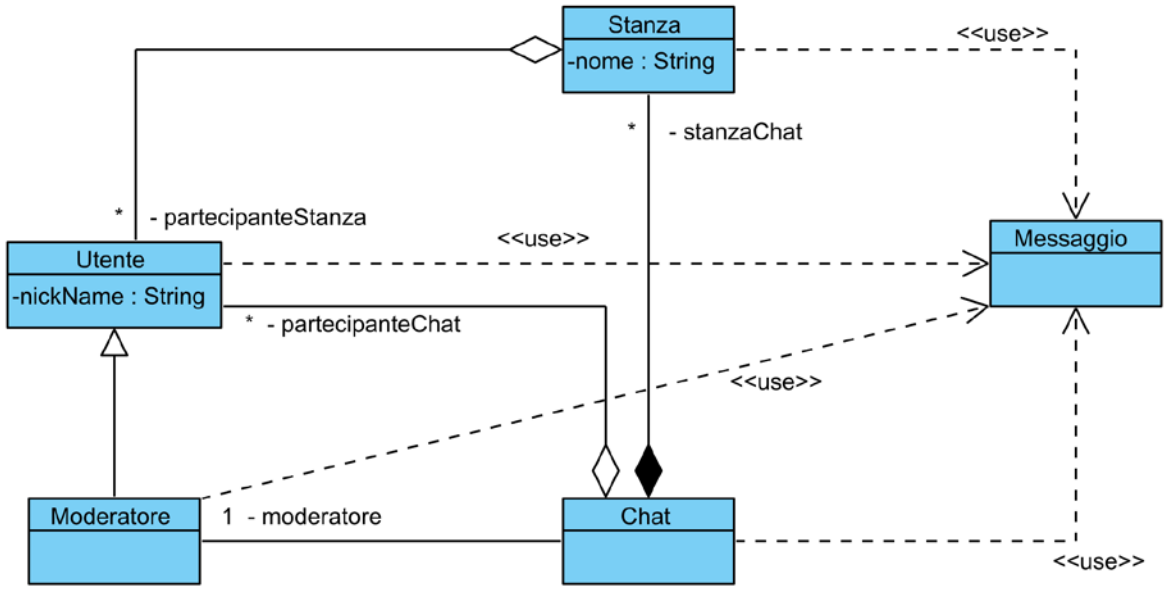
Esercizio 1

Vogliamo realizzare un sistema software per la gestione di una semplice chat (**EasyChat**) caratterizzata dai requisiti informali descritti nel seguito, che consente a diversi utenti lo scambio di messaggi in rete.

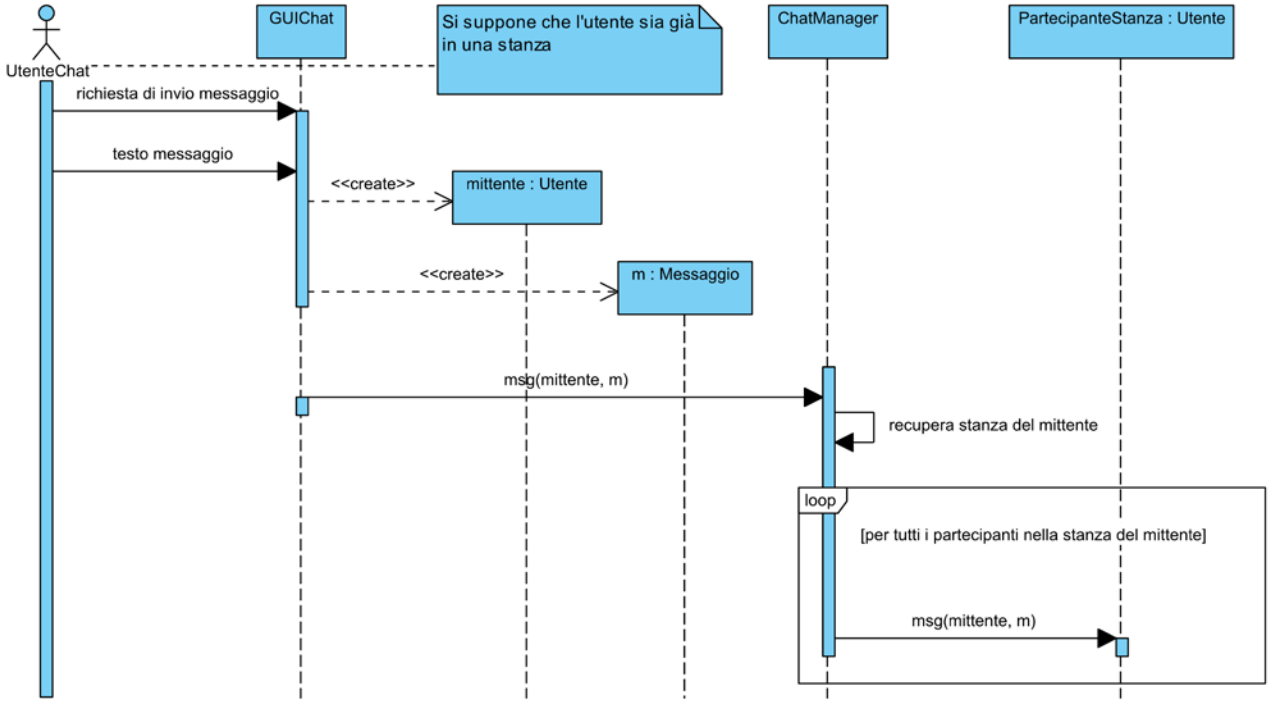
- In una chat esistono diverse stanze virtuali (dotate di nome) nelle quali un utente può entrare per scambiare messaggi con gli altri utenti presenti nella stessa stanza
- Un utente è identificato nel sistema software con un nick name
- I messaggi inviati da un utente sono trasmessi a tutti gli altri utenti presenti nella stanza
- E' comunque possibile inviare messaggi privati ad uno specifico utente presente nella stessa stanza. In questo caso il destinatario selezionato sarà l'unico a ricevere il messaggio
- Il software prevede la presenza di una tipologia di utente particolare (il *moderatore*), con l'autorità di inviare avvisi ad altri utenti, di espellere utenti e di creare o eliminare stanze della chat. In ogni istante nella chat può esistere un solo moderatore

a) Modellare con un Class diagram gli aspetti principali del dominio del problema sopra descritto (modello concettuale). Rappresentare nel class diagram gli attributi (descritti nei requisiti informali) ma non le operazioni
b) Modellare con un sequence diagram l'operazione *InvioMessaggio* da parte di un utente della chat verso tutti gli altri utenti presenti nella stessa stanza. Considerare come attori almeno le componenti GUIChat (presentation layer) e ChatManager (Logic layer)
c) Rappresentare mediante un component diagram UML una possibile architettura per **EasyChat.** Formalizzare e descrivere le interfacce fornite e richieste da ogni componente.

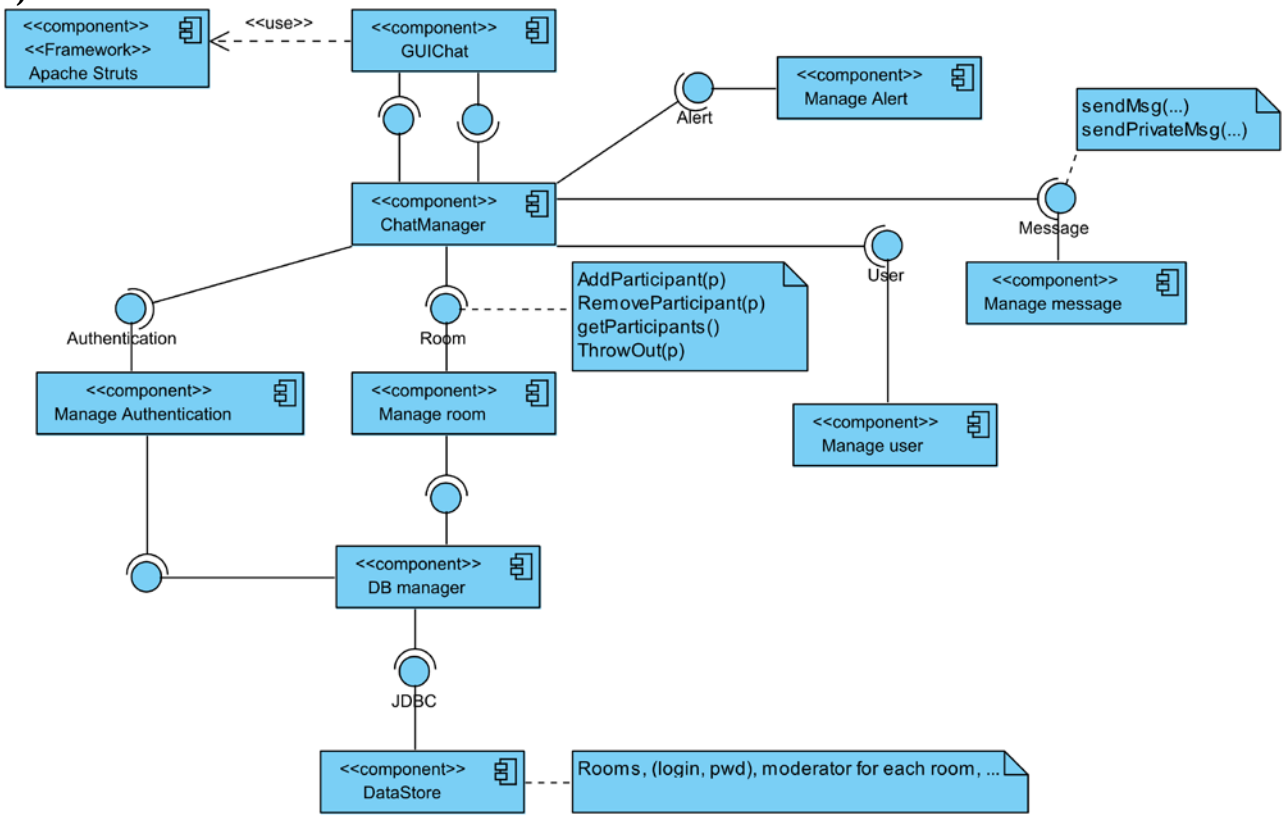
a)



b)



c)



Esercizio 2

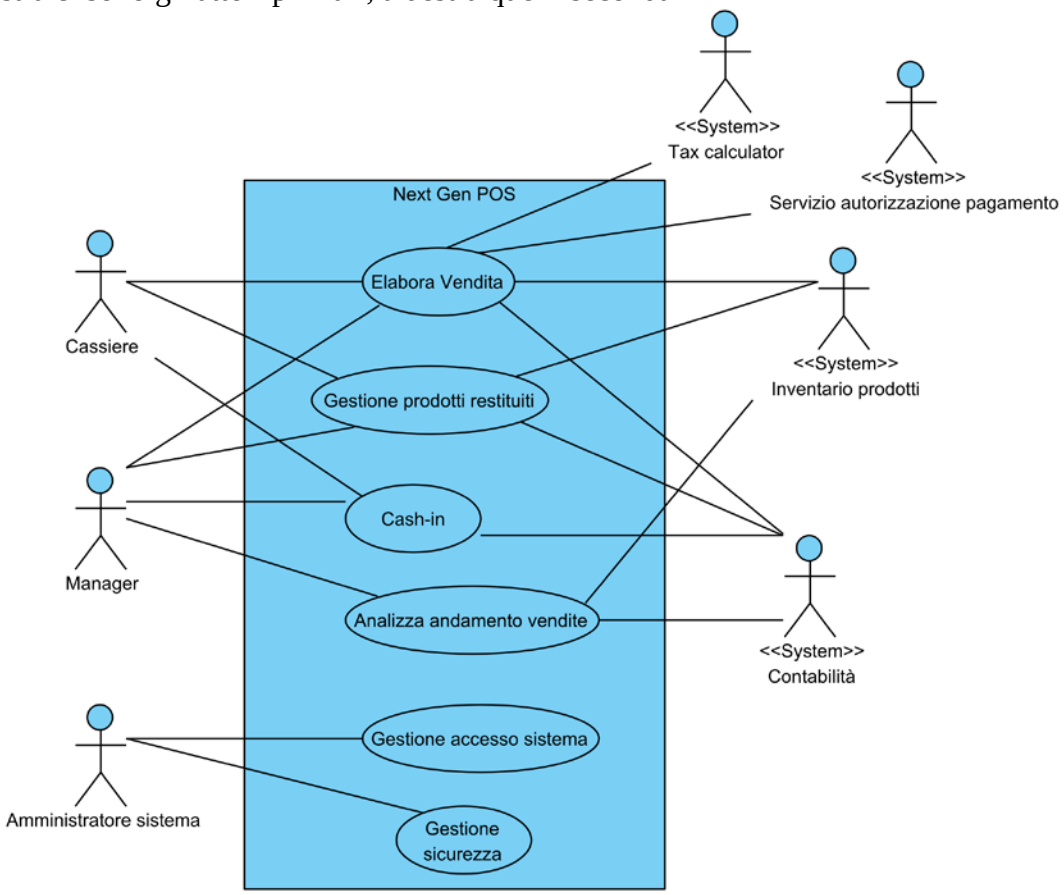
Vogliamo realizzare un sistema software per registrare le vendite e gestire i pagamenti in un supermercato chiamato **Next Gen POS** (Point Of Sale, punto di vendita).

I componenti hardware sono un computer ed un lettore di codici a barre. Next Gen POS si interfaccia ad altri sistemi prodotti da terzi: un sistema per l'inventario dei prodotti, un sistema per calcolare le tasse su ogni vendita, un sistema che gestisce la contabilità, un sistema di autorizzazione al pagamento che viene utilizzato nel caso in cui il cliente decide di pagare con bancomat o carte di credito (chiaramente è anche possibile pagare in contanti).

Il cassiere: (1) utilizza il sistema per registrare le vendite e gestire i pagamenti (**Elabora Vendita**), (2) utilizza il sistema per gestire i prodotti che sono per un qualche motivo rifiutati dal cliente e quindi restituiti (**Gestione prodotti restituiti**) (3) utilizza il sistema per effettuare il "**Cash-in**", ovvero la registrazione nel sistema dell'importo presente nel cassetto della cassa. Oltre al cassiere esistono anche la figura del manager che può eseguire le stesse mansioni del cassiere più analizzare l'andamento delle vendite e l'amministratore di sistema che mediante apposita GUI: (1) gestisce l'accesso al sistema per i cassieri fornendo le password e (2) gestisce la sicurezza.

a) Modellare con uno Use case diagram il sistema Next Gen POS, identificando gli attori primari e secondari
b) In quale use case tra quelli presenti nello Use case diagram potrebbe essere utile usare la relazione di *include* e perchè?
c) Descrivere, utilizzando la notazione vista a lezione, il caso d'uso **Gestione prodotti restituiti**
d) Modellare con un activity diagram il caso d'uso **Elabora vendita.** Utilizzare le swimlanes con i partecipanti: Cliente, Cassiere, sistema Next Gen POS e sistema di autorizzazione al pagamento.

a) A sinistra ci sono gli attori primari, a destra quelli secondari



b) Nello use case Elabora vendita si potrebbero includere tre (opiù) use case: pagamento con bancomat, pagamento con carta di credito e pagamento cash (altre tipologie di pagamento).

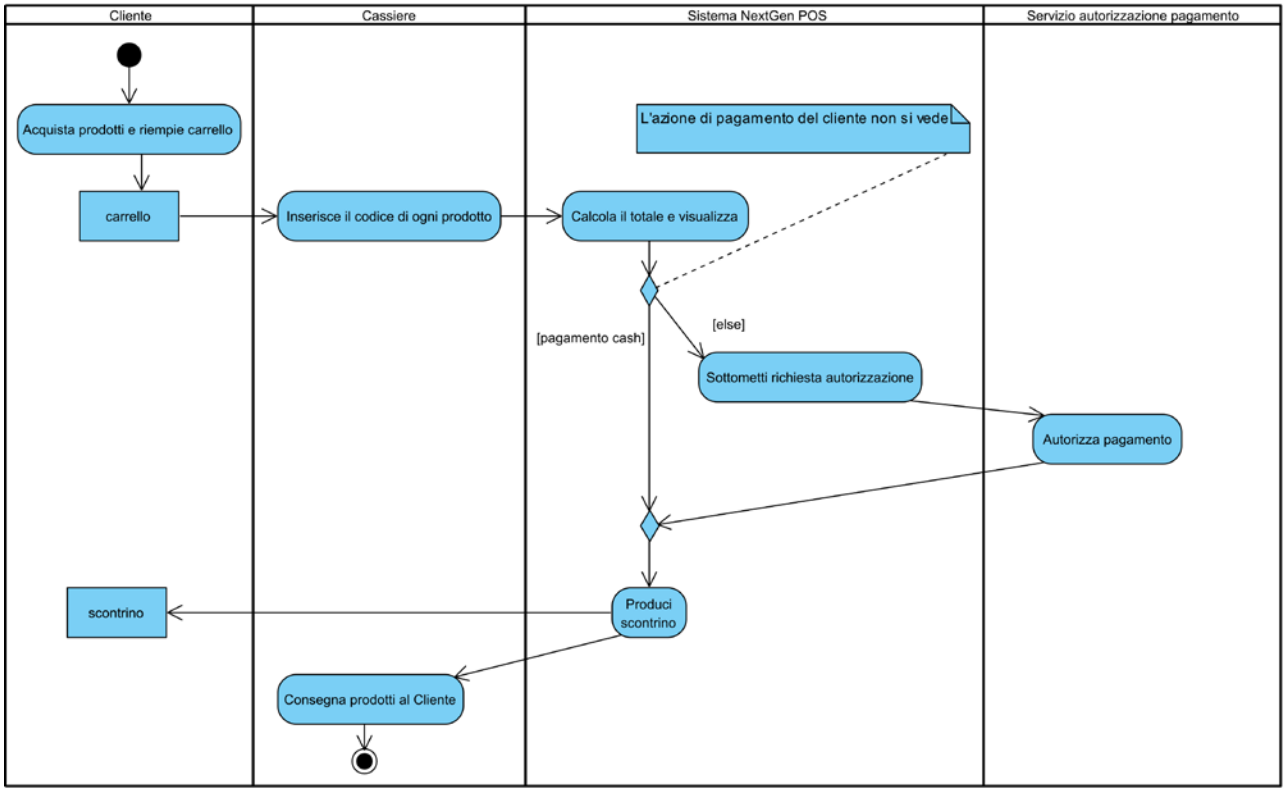
c) Si descrive lo use case ad alto livello

Caso d'uso: Gestione prodotti restituiti
Id: 6
Breve descrizione: il Cassiere gestisce la restituzione di uno o più prodotti da parte di un cliente
Livello: User-goal
Attori primari: Cassiere (seguendo le indicazioni del cliente)
Attori secondari: Inventario prodotti, Contabilità
Precondizioni: Il cliente ha acquistato uno o più prodotti nel supermercato e intende restituire alcuni di questi. Inoltre ha lo scontrino che testimonia l'acquisto
Sequenza degli eventi prinicpale:

1. Il cliente arriva alla cassa POS con i prodotti (uno o più) che intende restituire e lo scontrino che testimonia l'acquisto
2. Il cassiere inizia una nuova procedura di restituzione prodotti sul sistema Next Gen POS
3. Per ogni prodotto che il cliente intende restituire:
    a. Il cassiere inserisce il codice identificativo del prodotto
4. Il sistema mostra i prodotti inseriti (codice, nome, importo) e il totale da restituire al cliente e chiede conferma
5. Il cassiere conferma
6. Il sistema apre la cassa permettendo al cassiere di restituire l'ammontare, storna l'importo totale dal sistema di contabilità ed aggiorna l'inventario caricando i prodotti restituiti dal cliente
7. Il cassiere restituisce i soldi al cliente

Postcondizioni: L'ammontare dei prodotti restituiti è restituito al cliente. Invetario e conatbilità sono aggiornati
Sequenze eventi alternativa: ClienteCambiaIdeaproceduraAnnullata, CodiceInseritoNonValido, CassiereNonConferma, SistemacontabilitàNonFunzionante, SistemaInventarioNonFunzionante

d) Si modella un activity diagram semplificato (senza sequenza di eventi alternativa e senza mostrare tutti gli attori secondari)



Esercizio 3

Si supponga di voler sviluppare un'applicazione Java che realizzi un semplice carrello elettronico. Si supponga che l'applicazione debba appoggiarsi sui dati archiviati in un DB relazionale contenente le seguenti relazioni:

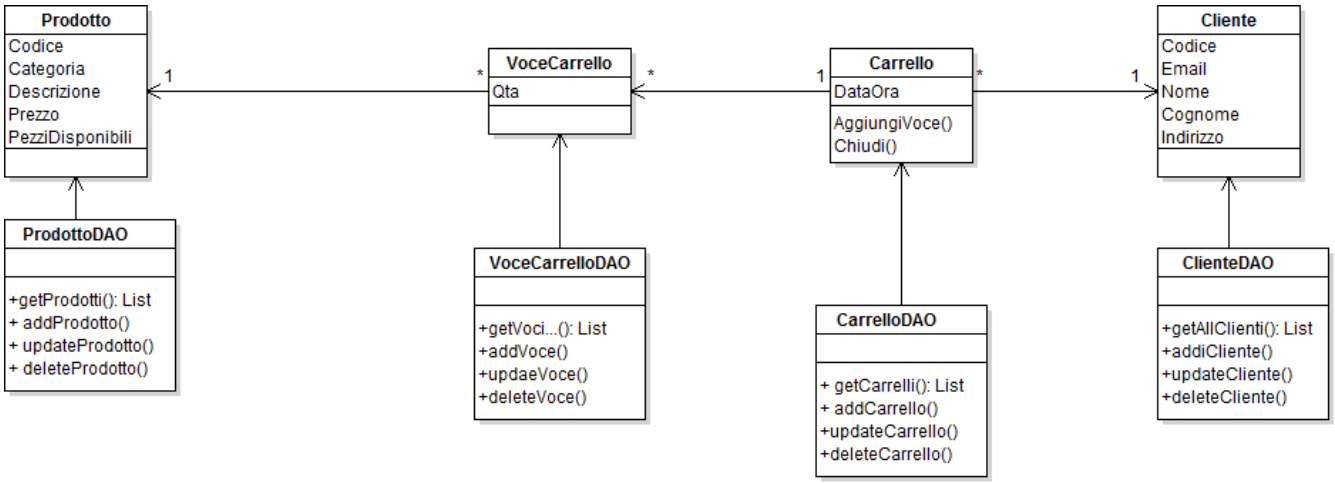
Catalogo(CodProdotto, Categoria, Descrizione, Prezzo, PezziDisponibili)
Cliente(CodCli, *Email*, Nome, Cognome, IndirizzoSpedizione)
Carrello(CodC, *DataOra*, *CodCli*^{Cliente}, ImportoTotale)
Contiene(CodC^{Carrello},CodProdotto^{Prodotto},Qta)

[Viene utilizzata la notazione vista nel corso di Basi di Dati: gli attributi sottolineati sono chiavi primarie, quelli in italico chiavi alternative, gli apici indicano per le chiavi esterne la relazione riferita]

a) Discutere quali degli approcci di gestione della persistenza visti risultano applicabili in questo contesto e quale risulta essere preferibile, motivando le affermazioni.

Approcci alla persistenza visti:
forza bruta, DAO, persistence framework (vedi slide per descrizioni e vantaggi/svantaggi)
In un contesto molto semplice come questo forse il preferibile è DAO: non serve "complessità" di persistence framework ma incapsula in strato separato la gestione della persistenza.

b) Supponendo ora di scegliere l'approccio DAO alla gestione della persistenza presentare (come struttura+segnatura classi Java o come class diagram) le classi corrispondenti alle DAO e alle classi di dominio.



I metodi delle classi DAO sono quelli che costruiscono i comandi SQL e li mandano in esecuzione sul DB utilizzando JDBC.

c) [opzionale] Supponendo invece di scegliere l'approccio alla gestione della persistenza basato su framework di persistenza (es. Hibernate) presentare un esempio di mapping, esplicitando dove e come viene utilizzato.

Es. di porzione di mapping Hibernate

```
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
 <hibernate-mapping package="…">
   <class name="Prodotto" table="Catalogo">
    <id name="Codice" column="CodProdotto">
    <generator class="increment"/>
   </id>
   <property name="Categoria" type="string" column="Categoria"/>
   <property name="Descrizione"/>
   <property name="Prezzo"/>
   <property name="PezziDisponibili"/>
 </class>
</hibernate-mapping>
```

d) Supporre infine di volere realizzare un'interfaccia grafica per la nostra applicazione che mantenga visualizzata in un'apposita area dello schermo i cinque prodotti più venduti nelle ultime 24 ore (aggiornati dinamicamente). Quale design pattern ritenete sia utile per implementare tale funzionalità e perchè?

Observer, permette di "osservare" i cambiamenti di stato dei prodotti/carrelli (pezzi venduti) e aggiornare di conseguenza l'area dello schermo.

e) Instanziare il design pattern che avete deciso al punto d) creando così un class diagram che sia un utile punto di partenza per l'implementazione della funzionalità.