Pag 1

1. falsa, vero ,falso
2. falso, falso, vero
3. falso. vero, falso

Pag2

1. vero, falso, vero

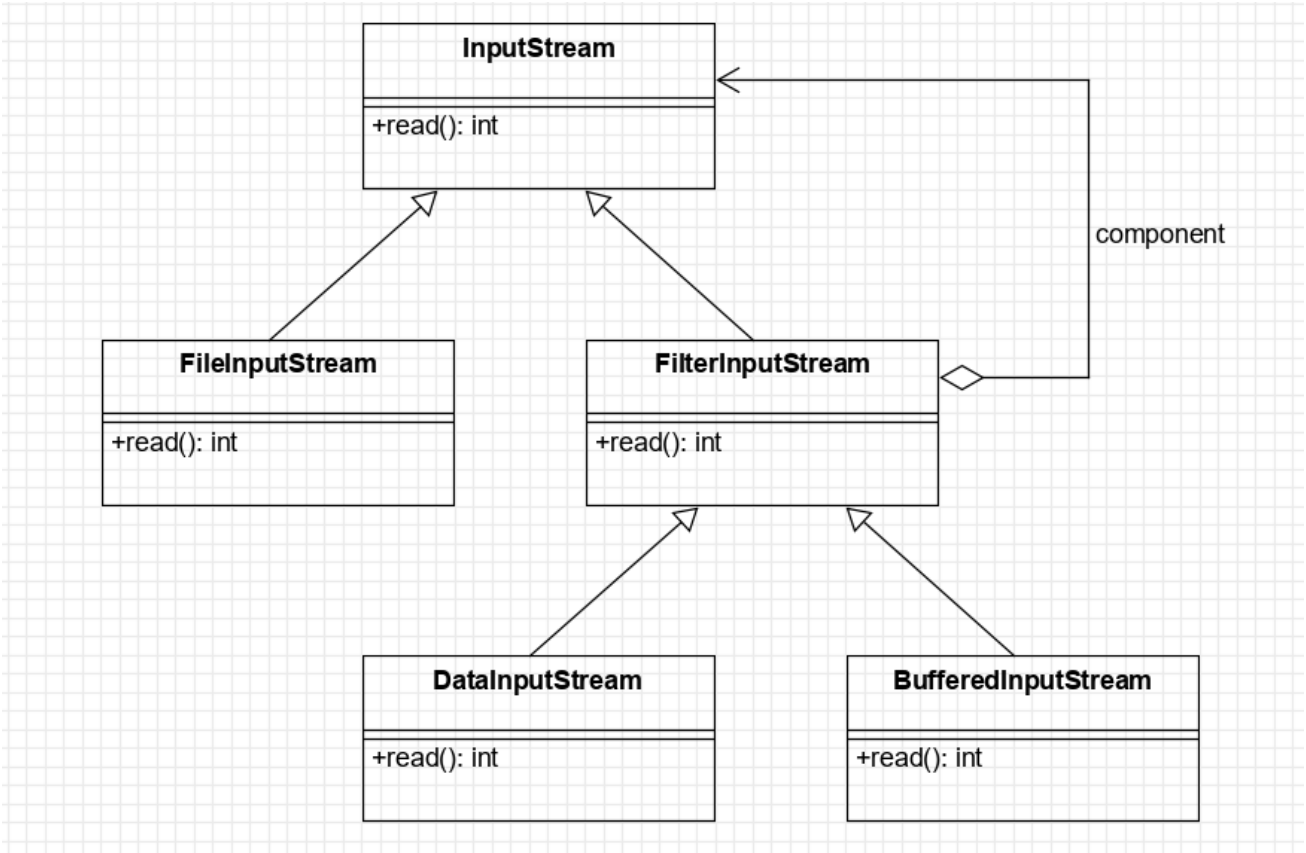
Pag3

1. falso,vero,falso,falso,vero
2. 5%,50%-70%,100%
3. vero, falso,vero,falso,falso
4. Falso,falso,vero,vero,falso,falso

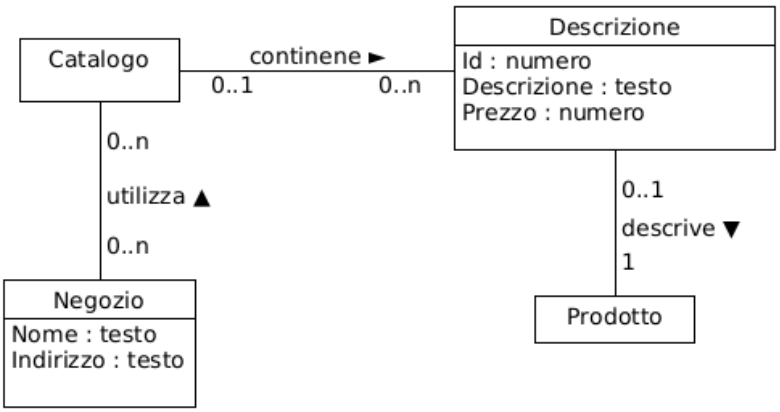
Pag4

1. falso,falso,vero,vero
2. stessa della domanda 1 pag 3 (falso,falso,vero,vero,falso)
3. creator, controller,low coupling, Information Expert, high cohesion
4. stessa della domanda 4 pag 3 (falso,vero,falso,falso,vero,falso)

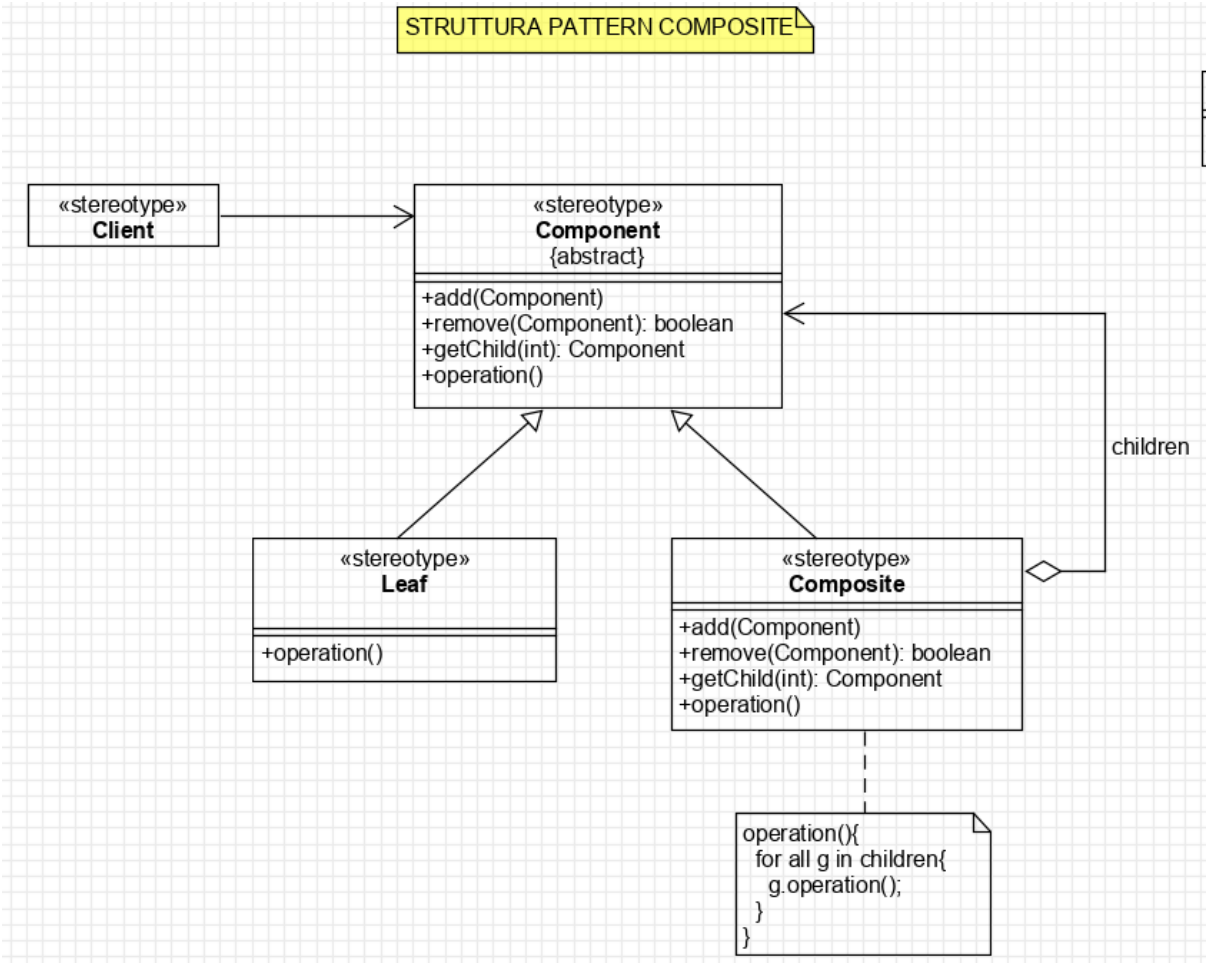
Pag5

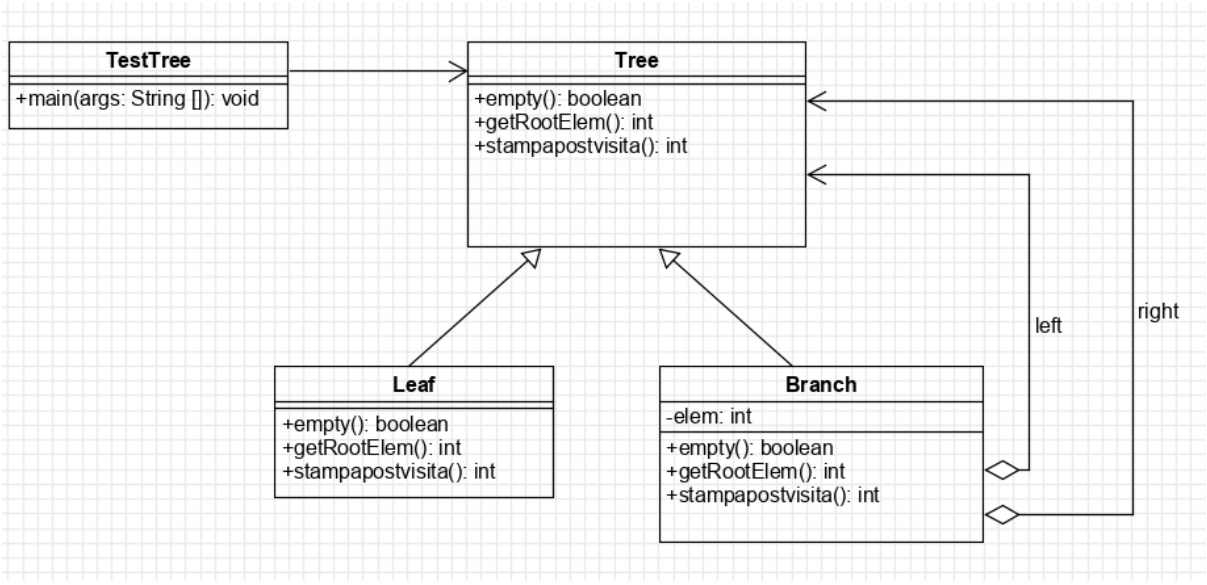


Pag6


Catalogo
continene ►
0..1
0..n
Descrizione
Id : numero
Descrizione : testo
Prezzo : numero
0..n
utilizza ▲
0..n
0..1
descrive ▼
1
Negozio
Nome : testo
Indirizzo : testo
Prodotto

Pag7


STRUTTURA PATTERN COMPOSITE
«stereotype»
Client
«stereotype»
Component
{abstract}
+add(Component)
+remove(Component): boolean
+getChild(int): Component
+operation()
children
«stereotype»
Leaf
+operation()
«stereotype»
Composite
+add(Component)
+remove(Component): boolean
+getChild(int): Component
+operation()
operation(){
for all g in children{
g.operation();
}
}



Pag8

Le post-condizioni descrivono i cambiamenti nello stato degli oggetti del modello di dominio. I cambiamenti dello stato del modello di dominio comprendono gli oggetti creati, i collegamenti formati o rotti, e gli attributi modificati.

Le pre-condizioni descrivono le ipotesi significative sullo stato del sistema prima dell'esecuzione dell'operazione.

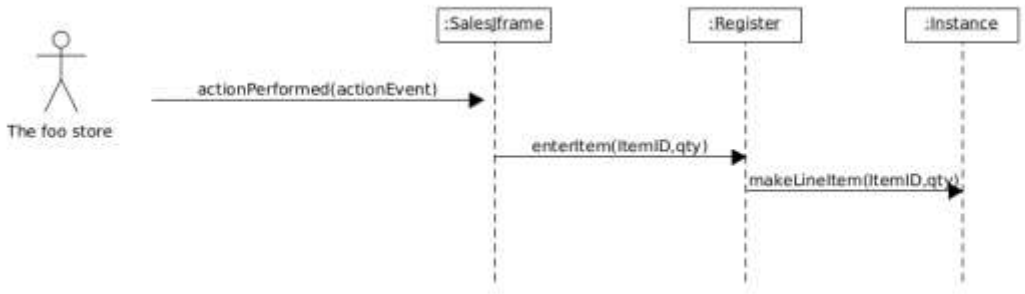
Operazione: modificaTitolo(nuovoTitolo: testo)

Riferimenti: casi d'uso: Gestire Menù

Pre-condizioni: è in corso la definizione di un Menu *m*

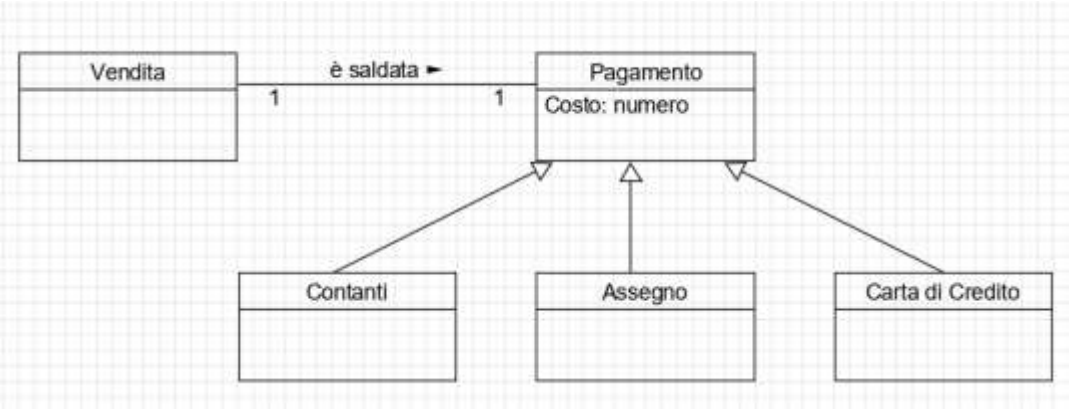
Post-condizioni: *m.titolo* = nuovoTitolo

pag9

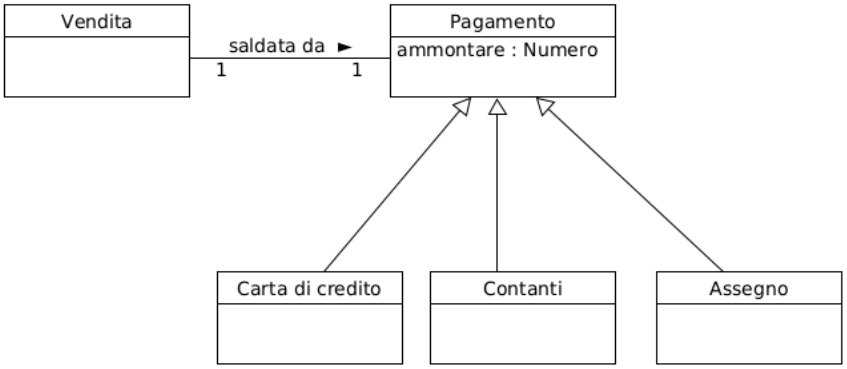


la classe SalesJFrame appartiene allo strati UI, è infatti l'observer (o listener) della UI.
Le altre due classi sono parte dello strato di dominio.
Il controller grasp è Register (il controller grasp non è il controller MVC, che invece è salesJFrame)

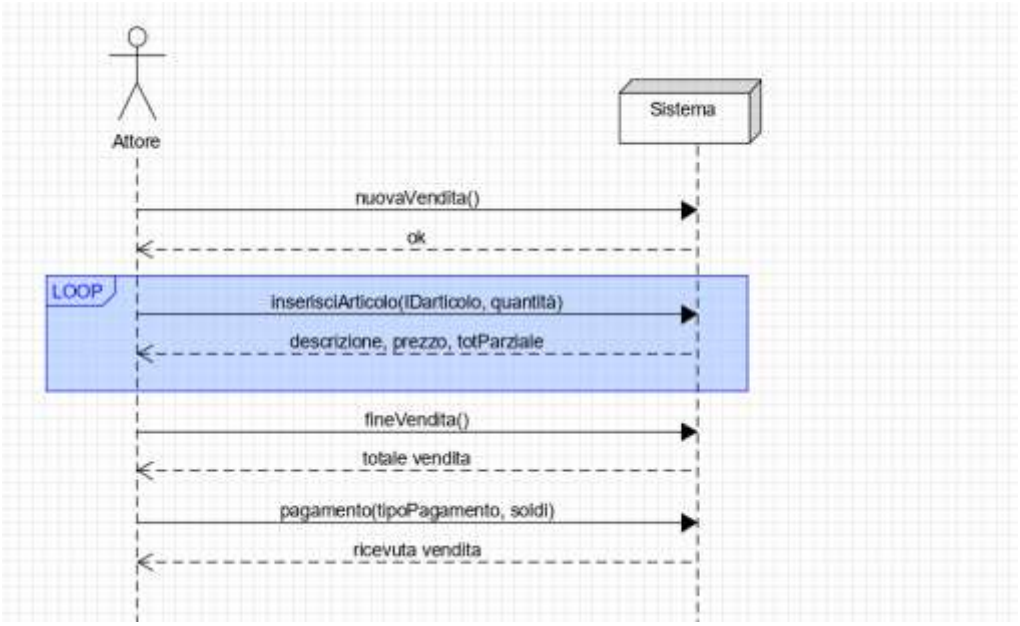
Pag10



pag 11
Modello di Dominio:



SSD:

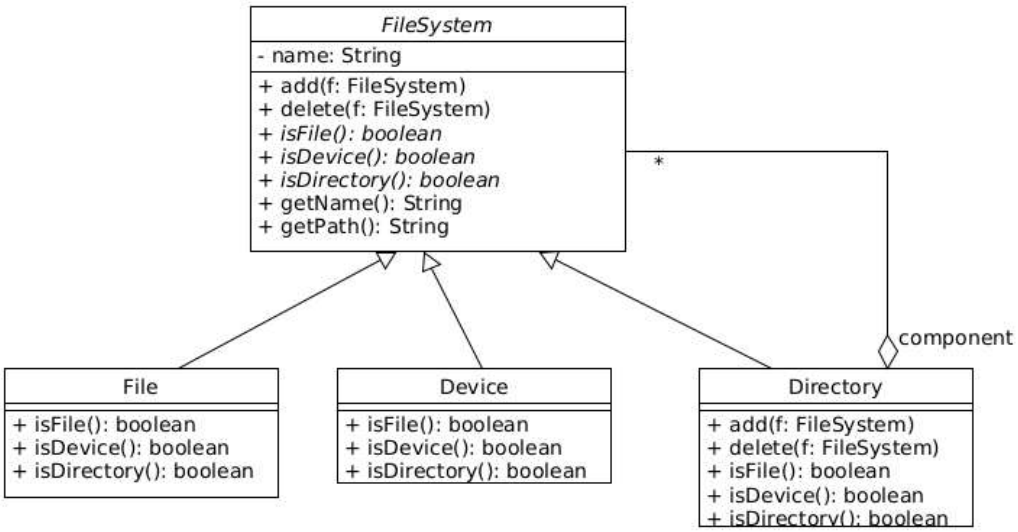


pag 12

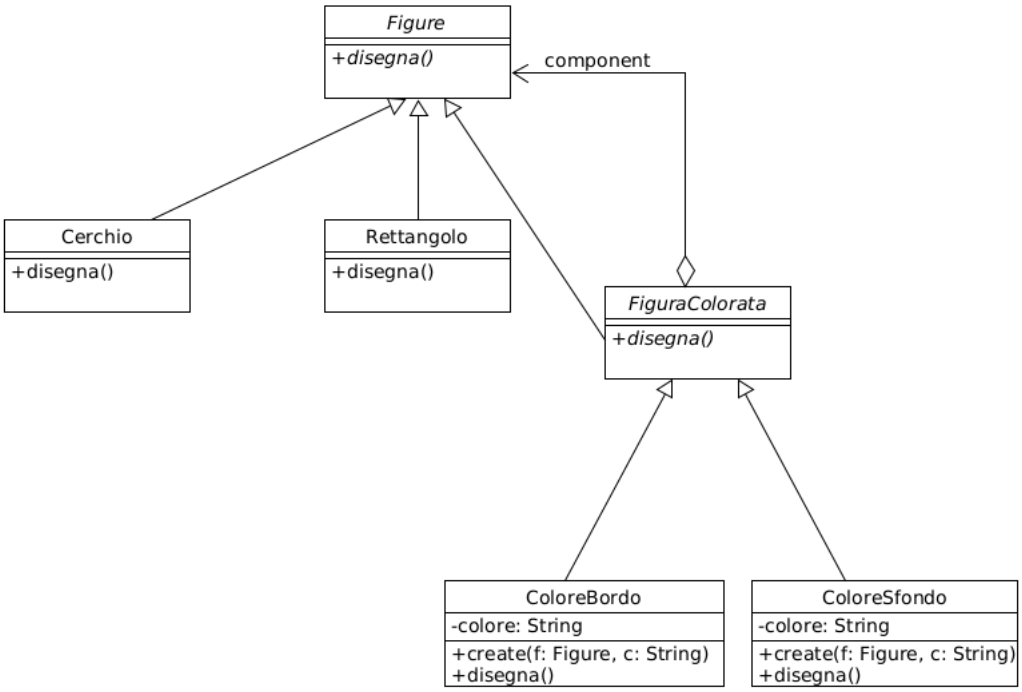
pag 13

1. falso, falso. vero, vero, vero
2. falso, vero, vero, vero

pag14
il pattern è composite



pag 15
pattern decorator

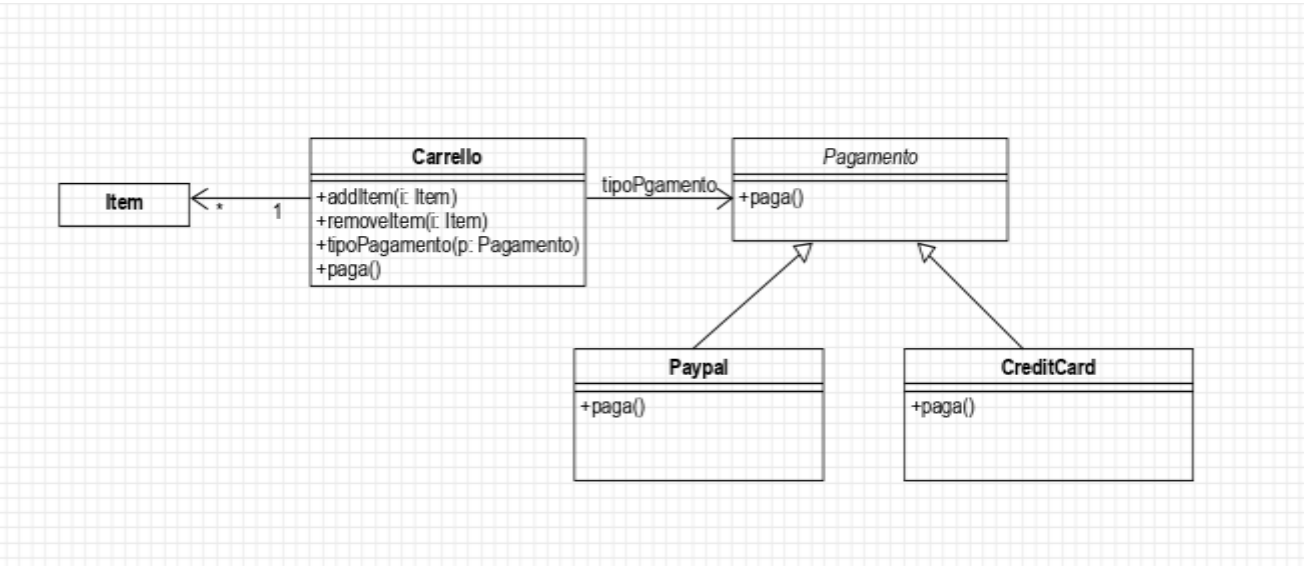


pag 16

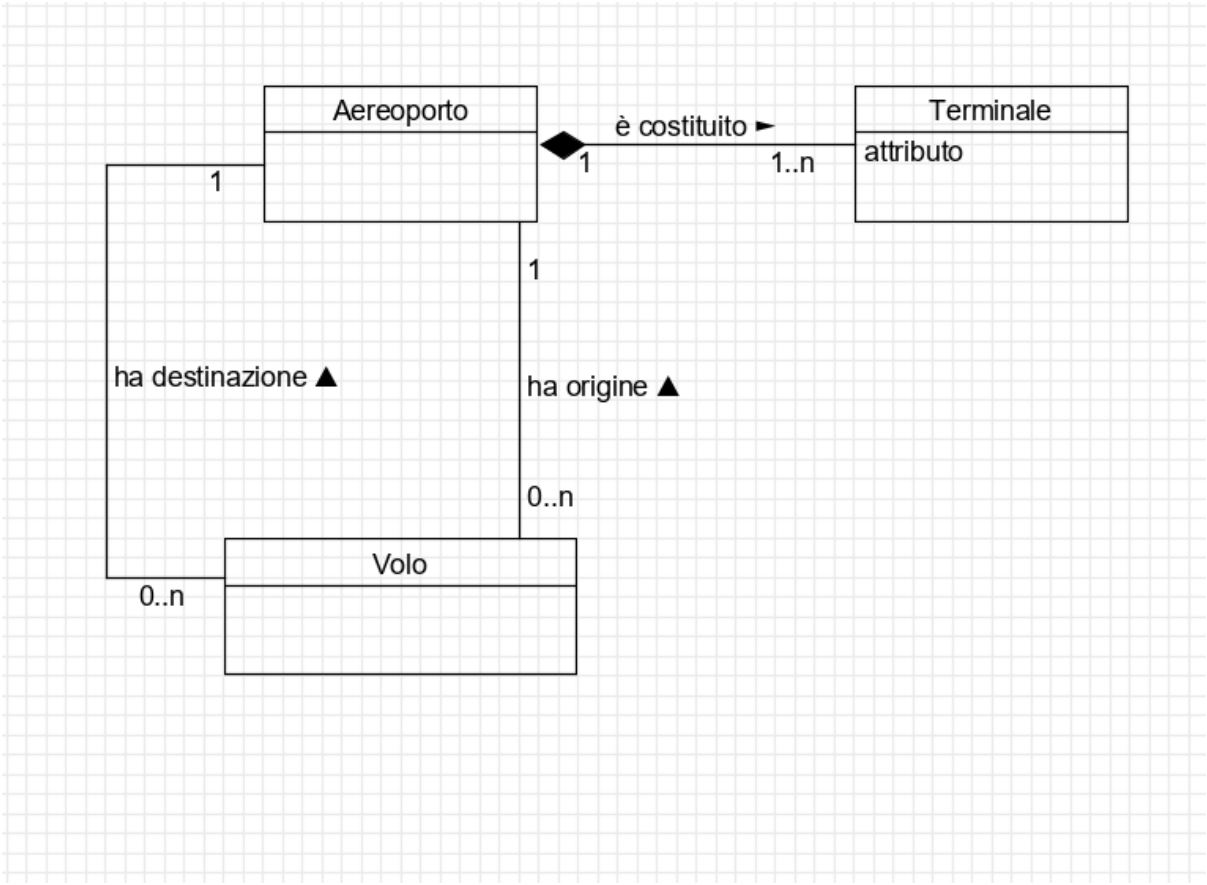
1. vero, falso, falso, vero, falso
2. vero, falso, falso,vero, falso
3. vero, falso (what??), falso, falso

pag 17

Pattern Strategy



pag18


Aereoporto
è costituito
Terminale
attributo
1
1
1..n
1
ha destinazione
ha origine
0..n
Volo
0..n

- Dire quali sono le tecniche di black box testing.

E' il testing funzionale e richiede l'analisi degli output generati dal sistema in risposta ad input definiti sulla base della sola conoscenza dei requisiti del sistema. Vengono effettuati i seguenti test: testing basato sui requisiti, delle partizioni, sulle tabelle di decisione su grafi causa-effetto.

- A cosa serve la complessità ciclomatica e in cosa consiste?

Detta anche complessità condizionale, è utilizzata per misurare la complessità di un programma. E' calcolata utilizzando il grafo di controllo di flusso del programma, i nodi del grafo corrispondono a gruppi indivisibili di istruzioni mentre gli archi connettono due nodi se il secondo gruppo di istruzione può essere eseguito immediatamente dopo il primo. Può essere applicata a singole funzioni, moduli, metodi o classi di un programma.

- Spiegare cos'è e che ruolo ha il validation testing

I test di validazione garantiscono che il prodotto soddisfi le esigente del cliente, può essere definito per dimostrare che il prodotto soddisfa il suo uso previsto quando viene distribuito. Consiste in dei test tra cui unitario, d'integrazione, di sistema e di accettazione dell'utente.

- Data la funzione per calcolare il voto di sas utilizzare le classi di equivalenza per scrivere dei test per verificare la correttezza
- Panoramica generale sul riuso del software, approcci che supportano il riuso del software, framework applicativi nel dettaglio e linee di prodotti nel dettaglio

Usiamo un architettura a strati per ridurre l'accoppiamento e le dipendenze, in uno strato la responsabilità degli oggetti devono essere fortemente correlate, l'uno all'altro, e non devono essere mischiate con le responsabilità degli altri strati (alta coesione). Usiamo una progettazione modulare secondo cui il software deve essere decomposto in un insieme di elementi software (moduli) coesi e debolmente accoppiati. I principali pattern in GRASP sono High Cohesion e Low Coupling.

- Programmazione estrema dei metodi agili

I metodi per lo sviluppo agile di solito applicano lo sviluppo iterativo ed evolutivo, questo introduce una risposta rapida e flessibilità ai cambiamenti (interazioni brevi, raffinamento evolutivo dei piani, dei requisiti e del progetto). Il valore della modellazione agile è quello di migliorare la comprensione, anzichè quello di documentare delle specifiche affidabili.

- Cos'è la progettazione del software, come si sostiene, come si utilizza, dov'è possibile utilizzare i prototipi e in che ambito? Fare esempi.

Un sistema software ben progettato è facile da comprendere, da mantenere e da estendere. Inoltre le scelte fatte consentono delle buone opportunit`a di riusare i suoi componenti software in applicazioni future.

- Partendo dal modello a spirale parlare della prototipazione
- Cos'è White Box Test, indicare anche alcuni nomi/modi.

White Box Testing è una tecnica di test del software in cui la struttura interna, il design e la codifica del software vengono testati per verificare il flusso di input-output e per migliorare il design, l'usabilità e la sicurezza, il codice da testare è visibile ai tester. Alcune tecniche sono: copertura delle dichiarazioni: ogni dichiarazione nel codice viene testata almeno una volta; branch coverage: controlla ogni possibile percorso di un'applicazione software; statement e branch generalmente coprono il 80/90% del codice.

Copertura della dichiarazione; Copertura decisionale; Copertura delle filiali; Copertura delle condizioni; Copertura di più condizioni; Copertura della macchina a stati finiti; Copertura del percorso; Test del flusso di controllo; Test del flusso di dati

- UML, Pattern e Up. Testing?

UML è un linguaggio visuale per la specifica, la costruzione e la documentazione degli elaborati di un sistema software.
Il pattern è una disposizione utilizzata per descrivere un disegno, modello, schema, struttura ripetitiva e viene utilizzato per indicare la ripetizione di una determinata sequenza all'interno di un insieme di dati grezzi.
Testing serve per testare la correttezza di quanto implementato, esistono diversi tipi di test: test unitari (verificano piccole parti del sistema), test di integrazione (verifica la comunicazione tra specifiche parti), test end-to-end (verifica il collegamento complessivo tra tutti gli elementi), test di accettazione (verifica il funzionamento complessivo del sistema, considerando il punto di vista dell'utente)

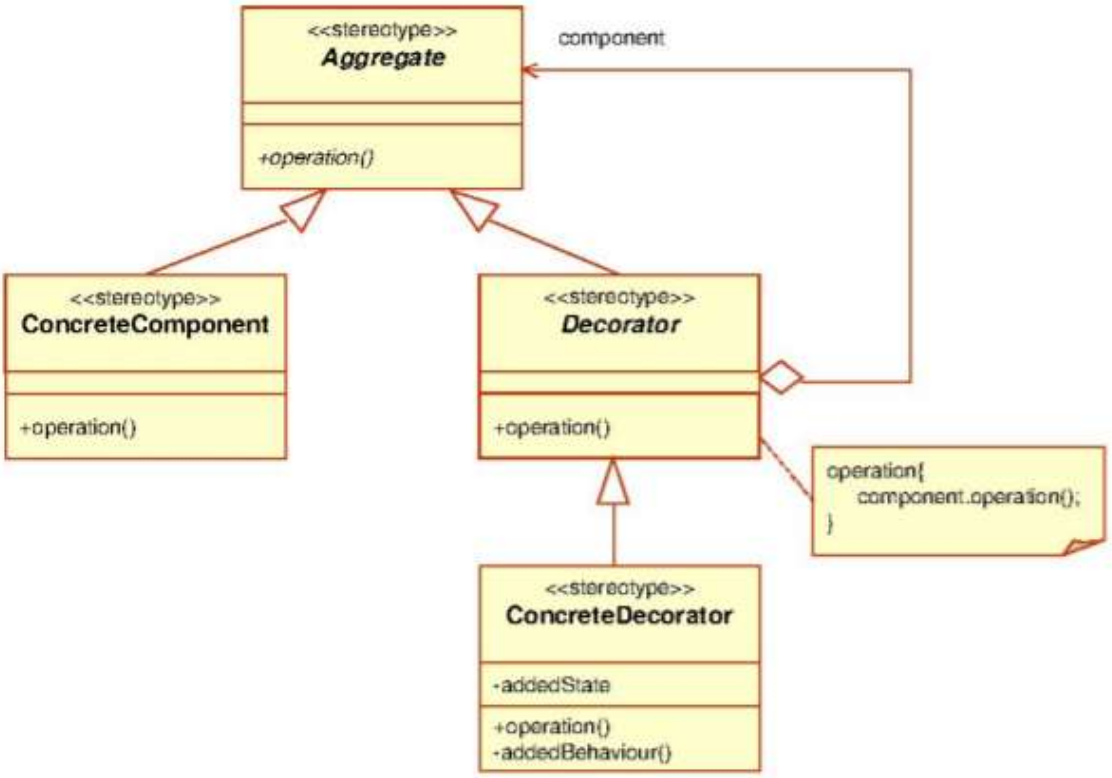
- Quando si fa il System Sequence Diagram (SSD)?
  - quando si comincia a scrivere il codice
  - all'inizio dell'analisi dei requisiti → **OK**
  - dopo i casi d'uso dettagliati → **OK**
  - non esiste questo diagramma in UP
- Cos'è il modello di dominio?
  - è una rappresentazione implementativa del sistema
  - è un altro nome per indicare il modello di progetto
  - è una rappresentazione concettuale del sistema → **OK**
  - serve a rappresentare i casi d'uso in modo grafico
- Quante iterazioni ha la fasi di ideazione?
  - quante ne servono
  - normalmente una → **OK**
  - dipende dalla percentuale dei casi d'uso che si intende implementare
  - non ha iterazioni
- Qual è il principale svantaggio della strategia top-down per l'integration testing, che la strategia bottom-up invece non presenta?
  - poichè combina subito insieme tutte le unità, rende difficile capire la causa dei problemi rilevati → **OK**
  - non può essere applicata in caso di progettazione orientata agli oggetti
  - richiede di scrivere molti stub → **OK**
  - testa per ultime le componenti più complesse e quindi più soggette ad errori, ritardandone la scoperta
- Quando si parla di regression testing o test di regressione?
  - quando si effettua dei test frequenti e non esaustivi relativi alle funzionalità principali, per decidere se il prodotto è sufficientemente stabile per essere testato da un team specializzato
  - quando si testano i componenti o le unità in ordine inverso rispetto a quando sono stati realizzati
  - quando si scrivono i test prima di avere implementato il codice
  - Quando, a seguito della correzione di un bug o dell'integrazione di un nuovo componente, si vanno a ripetere i test già effettuati per verificare che le modifiche apportate non abbiano introdotto dei problemi nuovi o non rilevanti → **OK**

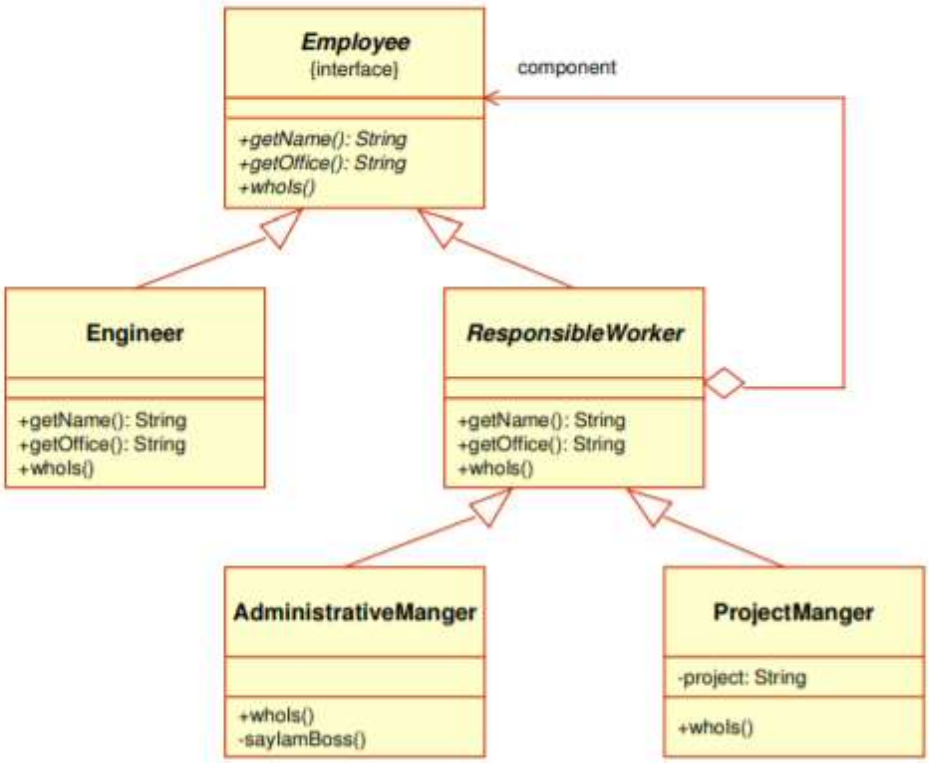
- Quale delle seguenti non è una tecnica di black-box testing?
  - testing casuale
  - calcolo dei cammini indipendenti
  - boundary value analysis o analisi dei valori limite
  - array ortogonali → **OK**
- Quale delle seguenti affermazioni sul validation testing è falsa?
  - il validation testing verifica che il software realizzato corrisponda alle specifiche iniziali/requisiti
  - alcune fasi del validation testing vengono eseguite direttamente dall'utente finale
  - documenti quali i Casi d'Uso dettagliati o i contratti delle Operazioni costituiscono un input rilevante per il validation testing → **OK**
  - Il validation testing può essere effettuato solo quando il software è in stato integralmente sviluppato
- Esporre sinteticamente come si calcola la complessità ciclomatica di una porzione di codice, cosa esprime, e in che modo questo è rilevante per il testing.

La complessità ciclomatica è calcolata utilizzando il grafo di controllo di flusso del programma, i nodi corrispondono a gruppi indivisibili di istruzioni e gli archi connettono due nodi se il secondo nodo di istruzioni viene eseguito dopo il primo. Per il testing è utile per testare singoli gruppi di istruzioni indivisibili.

- Nel testing cosa si intende per stub?
  - una tecnica di black box testing nella quale si utilizza una versione fittizia dell'unità da testare per farle calcolare i risultati attesi e confrontarli con quelli effettivamente ottenuti
  - un approccio all'integration testing
  - il codice che invoca i metodi di un dato modulo dell'unità per testarlo
  - una versione fittizia di un modulo/unità con la stessa interfaccia ma con comportamento predefinito, che consente il test in isolamento di altri moduli adesso dipendenti → **OK**
- Quale de seguenti NON è una delle tante forme di riuso dell'ingegneria del software?
  - framework
  - design pattern
  - pattern matching → **OK**
  - librerie
- Quali dei seguenti ruoli non è previsto dal metodo SCRUM di gestione dei progetti Agili?
  - Scrum Master
  - Project manager
  - Developer
  - Tester → **OK**
- Indicare le fasi V-Shaped.
- La differenza tra riuso COTS e sistemi applicativi integrati (Non dovremmo aver fatto questa roba quest'anno)
- Dire quale NON è un pattern GoF.
  - Abstract Factory
  - Proxy
  - Factory → **OK**
  - Composite

- Disegnare il pattern Decorator e scrivere il codice delle sue classi caratterizzanti, inclusi i costruttori. (Basta scrivere del codice schematico, che contenga però gli elementi essenziali)





- Quando si fanno gli SSD?

Si creano gli SSD una volta creato il Glossario e i Casi d'Uso principali per rappresentare gli eventi generati dagli attori esterni al sistema, il loro ordine e gli eventi inter-sistema.

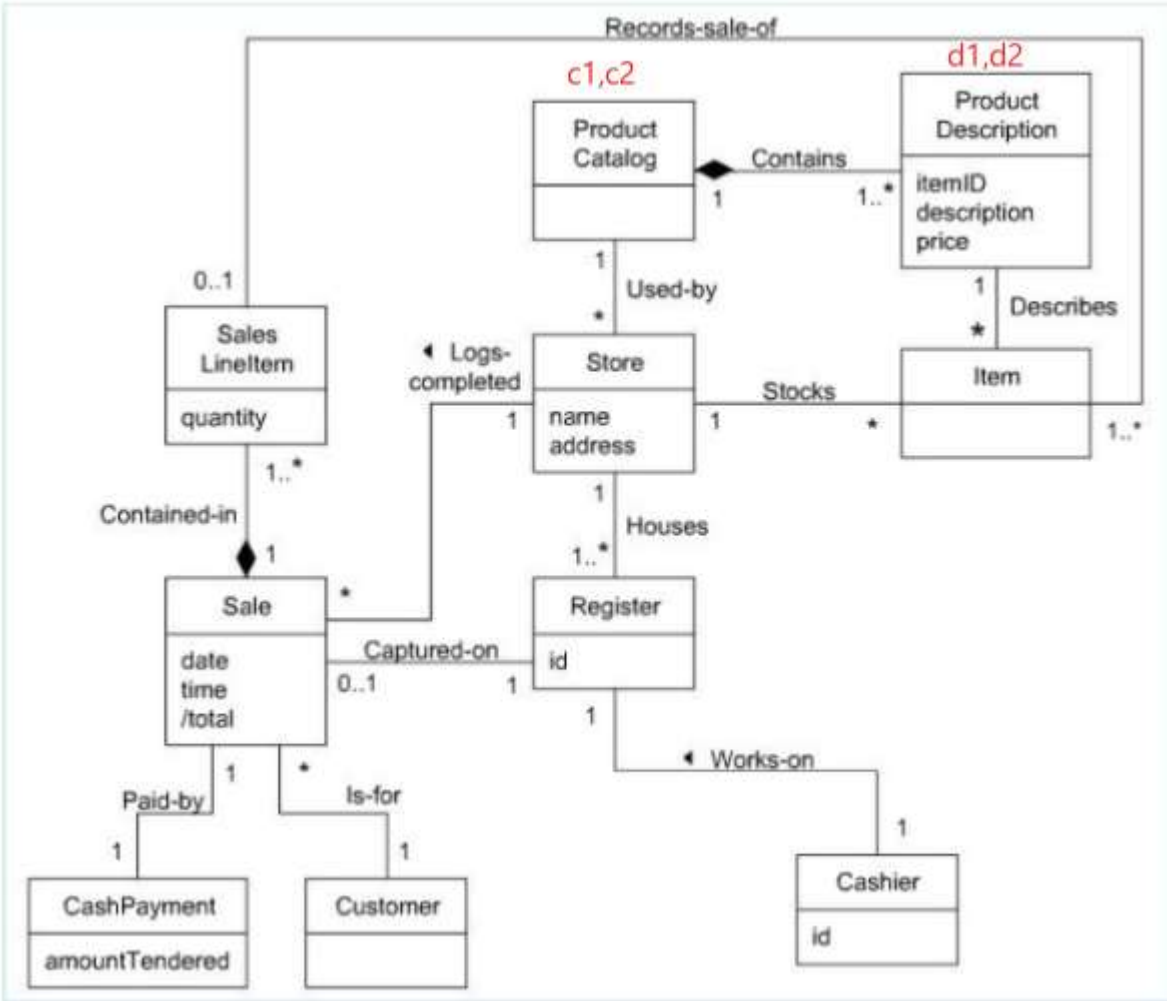
- Differenza tra modelli Plan-Driven e Agile?

La differenza principale è l'accettazione del cambiamento, nei metodi plan-driven le funzionalità del prodotto vengono congelate dall'inizio dello sviluppo impedendo il cambiamento secondo le richieste successive del cliente o del mercato, mentre le metodologie Agile hanno la capacità di cambiare i requisiti del prodotto in qualsiasi momento dello sviluppo.

- Vero o Falso:
  - Il refactoring è una pratica promossa dal metodo iterativo e agile XP → **VERO**
  - Il refactoring prevede lo sviluppo guidato dai test, ovvero uno sviluppo preceduto dai test → **FALSO**
  - Il refactoring è un metodo strutturato e disciplinato per scrivere o ristrutturare del codice esistente → **VERO**
- Quali delle seguenti affermazioni sono vere?
  - I requisiti non funzionali sono le proprietà del sistema nel suo complesso → **VERO**
  - I requisiti non funzionali sono i requisiti comportamentali → **FALSO**
  - I requisiti funzionali sono le proprietà del sistema nel suo complesso → **FALSO**
  - I requisiti funzionali sono requisiti comportamentali → **VERO**
- Quali delle seguenti affermazioni sono vere?
  - I casi d'uso sono una collezione di scenari correlati → **VERO**
  - Gli scenari sono una collezione di casi d'uso correlati → **FALSO**
  - Un caso d'uso è una collezione di storie nell'uso del sistema → **VERO**
  - Un caso d'uso è una collezione di SSD → **FALSO**
  - I casi d'uso si rappresentano mediante gli SSD → **FALSO**
  - Un caso d'uso è una descrizione testuale di scenari → **VERO**
  - Uno scenario esprime un obiettivo specifico raggiungibile da un attore attraverso sequenza di azioni ed interazioni con un sistema → **VERO**
- Quali delle seguenti affermazioni sono vere?
  - La disciplina dei requisiti è il processo per scoprire cosa deve essere costruito → **VERO**
  - La disciplina dei requisiti deve orientare lo sviluppo verso il sistema corretto→ **VERO**
  - La disciplina dei requisiti è completata durante la fase dell'elaborazione → **VERO**
  - La disciplina dei requisiti è completata durante la fase dell'ideazione → **FALSO**
  - La disciplina dei requisiti è iniziata durante l'elaborazione → **FALSO**
- La programmazione di qualità-produzione per un sottoinsieme dei requisiti si inizia dopo che l'analisi di tutti i requisiti sia stata completata ? Vero o **Falso**
- Quali di questi elaborati sono iniziati durante l'ideazione?
  - Scenario di sviluppo → **OK**
  - SSD
  - Modello di progetto

- Modello dei casi d'uso → **OK**
- DCD
- Glossario → **OK**
- Visione e studio economico → **OK**

- Si consideri il seguente Modello di Dominio, quali delle seguenti affermazioni è vera? Si supponga Product Catalog = {c1,c2} e Product Description = {d1,d2}



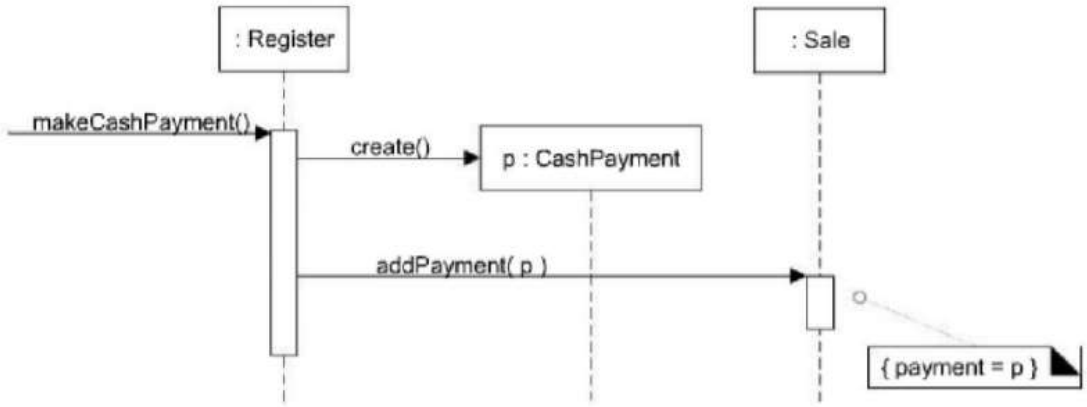
- Contains può essere {(c1,d1),(c1,d2),(c2,d2)} → **NO**
- Contains può essere {(c1,d1),(c2,d1),(c1,d2)} → **NO**
- Contains può essere {(c1,d1),(c2,d2)} → **OK**
- Contains può essere {(c1,d1),(c1,d2)} → **NO**

a C possono essere associati da 1 a più elementi e a D può essere associato un solo elemento. NB: essendo la cardinalità minima 1 devono esserci tutti gli elementi sia di C che di D ({(c1, d1), (c1, d2)} non va bene perchè manca c2

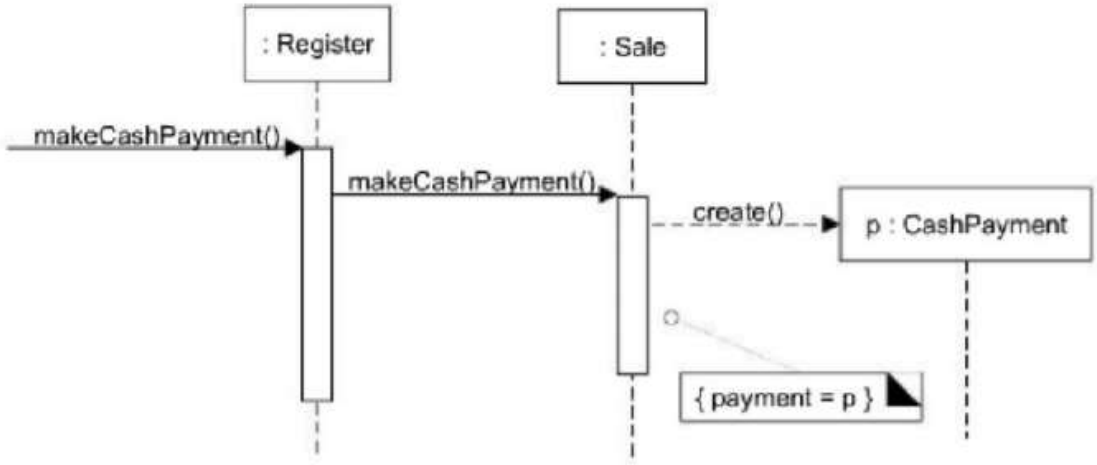
- Le post-condizioni descrivono i cambiamenti nello stato degli oggetti del modello di dominio. → **VERO**
- Le post-condizioni descrivono i cambiamenti nello stato degli oggetti del modello di PROGETTO. I cambiamenti dello stato del modello di progetto comprendono gli oggetti creati, i collegamenti formati o rotti, e gli attributi modificati? **Falso**

NB: IL MODELLO DI PROGETTO (DCD) E' DIVERSO DAL MODELLO DI DOMINIO!

- Quali delle seguenti affermazioni sulle associazioni utilizzate nel modello di dominio sono vere?
    - E' per sua natura unidirezionale → **FALSO**
    - La direzione di lettura è specifica di visibilità → **FALSO**
    - Rappresenta un valore logico degli oggetti di una classe → **FALSO**
    - Rappresenta una relazione significativa tra classi → **VERO**
    - Rappresenta un'insieme di n-tuple di oggetti delle classi → **VERO**
    - La direzione di lettura va sempre specificata → **FALSO**
- Durante la fase di elaborazione:
    - Viene scoperta e stabilizzata la maggior parte dei requisiti → **VERO**
    - Non vengono effettuati test al codice sviluppato → **FALSO**
    - I rischi maggiori sono attenuati o rientrano → **VERO**
    - Non si sviluppa codice → **FALSO**
    - Si realizza uno studio economico per stabilire l'ordine di grandezza del progetto e dei costi → **FALSO**
    - Viene programmato il nucleo, rischioso, dell'architettura → **VERO**
    - Si analizzano circa il 10% dei casi d'uso in dettaglio → **FALSO**
    - Si scrivono circa il 10% dei casi d'uso tra i più critici in formato dettagliato utilizzando template appositi (caso d'uso dettagliato e strutturato) → **FALSO**
    - I requisiti e le iterazioni sono organizzate in base alle richieste dell'utente finale → **FALSO**
    - Si effettua attività di programmazione di qualità-produzione e test. → **VERO**
    - I requisiti e le iterazioni sono organizzate in base al rischio, coperture e criticità → **VERO**
    - Vengono realizzati prototipi "usa e getta" per attenuare i rischi maggiori → **FALSO**
- Ricevere un parametro di tipo A per un metodo di una classe B rappresenta una relazione di dipendenza da B a A mentre estendere una classe A per una classe B non rappresenta una relazione di dipendenza da B a A. Vero o **Falso**?
- Quanti e quali "accoppiamenti" mostra il diagramma di interazione (A) e il diagramma di interazione (B)? Spiegare perchè. Quale dei due diagrammi è preferibile?

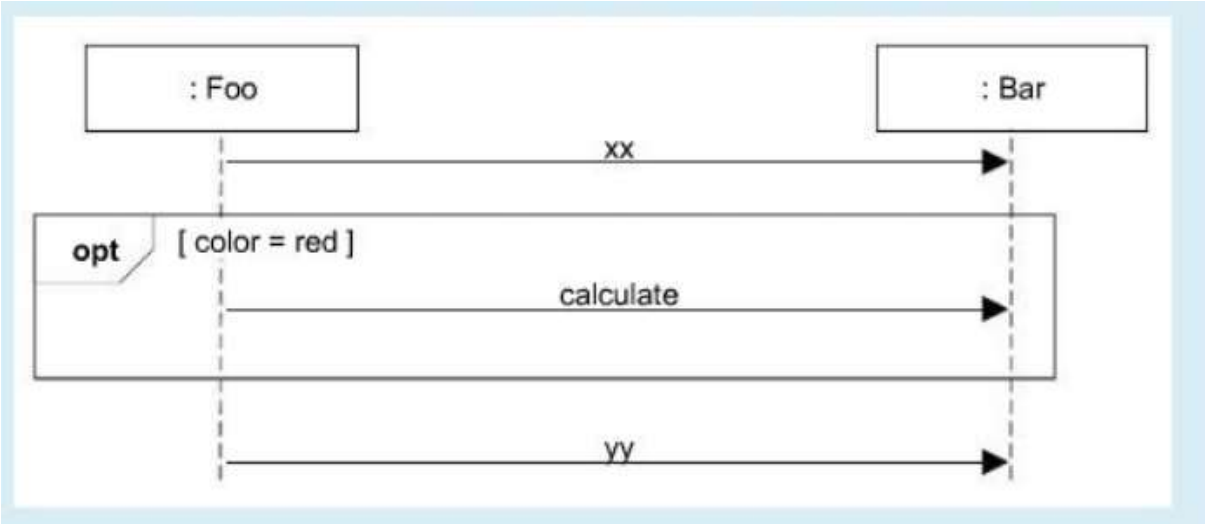


In questo caso esiste l'accoppiamento Register-CashPayment e Register-Sale quindi sono presenti due accoppiamenti con Register, perchè Register si assume la responsabilità di ricevere l'operazione di sistema makeCashPayment() e la responsabilità di soddisfarla.

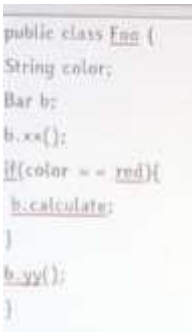


In questo caso esiste l'accoppiamento Register-Sale e Sale-CashPayment. Questo schema va preferito perchè non aumenta l'accoppiamento ma mantiene un accoppiamento complessivo più basso. Inoltre ha anche un'alta coesione in quanto le responsabilità sono equamente distribuite.

- In UML per responsabilità si intende:
  - la specifica di un metodo associato ad una classe Java → **FALSO**
  - la specifica di una variabile di istanza o di un metodo associato ad una classe Java → **FALSO**
  - la specifica di una variabile di istanza di una classe Java → **FALSO**
  - un contratto o un obbligo di un classificatore → **VERO**
- Scrivere il codice Java corrispondente al seguente diagramma di sequenza:



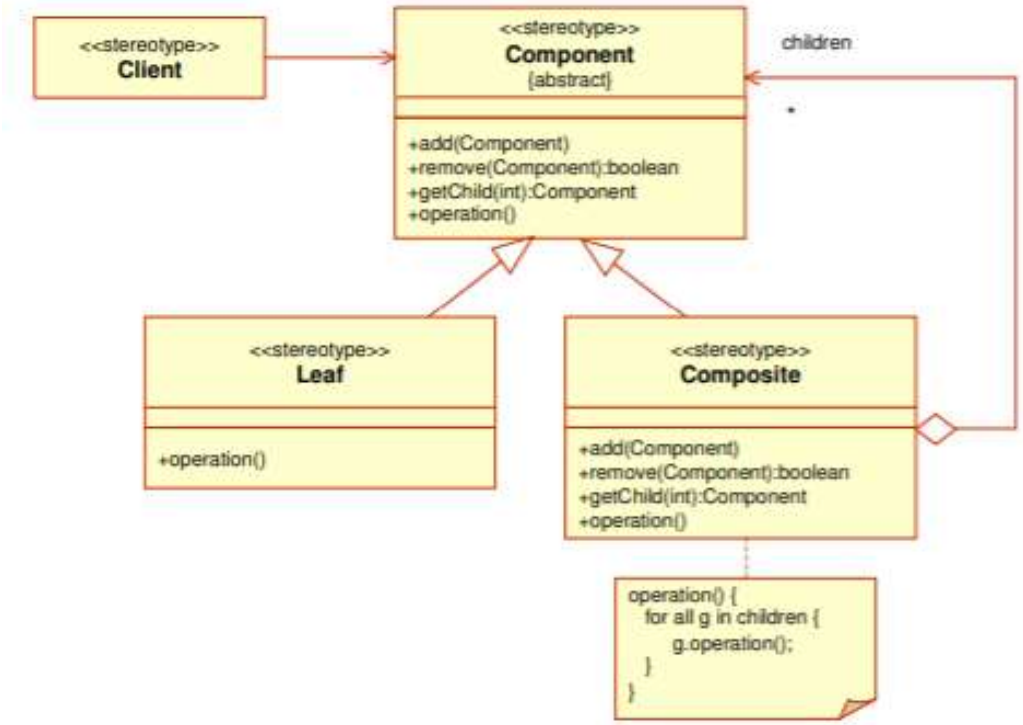

```
public class Foo{
  String color;
  Bar b;
  b.xx();
  if color == red {
    b.calculate;
  }
  b.yy();
}
```

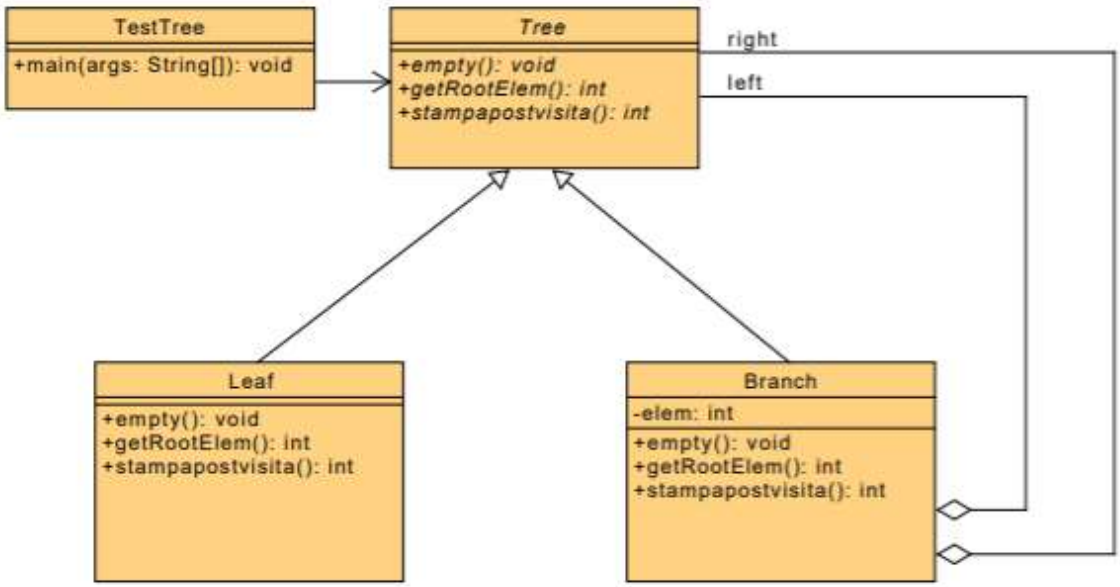


- Quale pattern tratta il problema “Come ridurre l’impatto dei cambiamenti? Come sostenere una dipendenza bassa, un impatto dei cambiamenti basso e una maggiore opportunità di riuso?”
    - Abstract Factory
    - Information Expert
    - Low Coupling → **OK**
    - Controller
    - Singleton
- Dire a quale pattern GoF la seguente implementazione della struttura dati albero binario si conforma. Disegnare la struttura del pattern e il diagramma delle classi della soluzione applicata alla struttura dati albero binario.

```
1  public abstract class Tree {
2    public abstract boolean empty();
3    public abstract int getRootElem();
4    public abstract void stampapostvisita();
5  }
6
7  public class Leaf extends Tree {
8      public Leaf() { }
9      public boolean empty() {
10         return true;
11     }
12     public int getRootElem() {
```

Il pattern è Composite





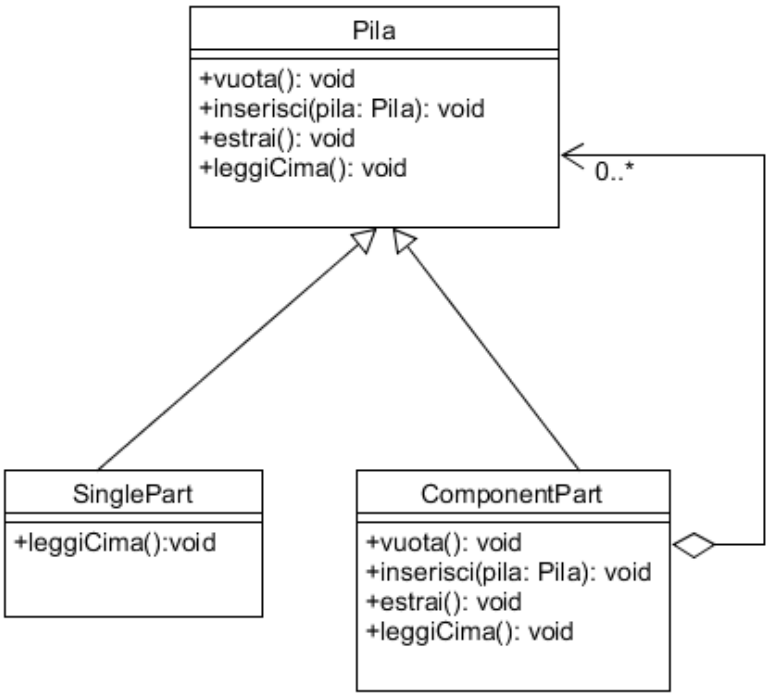
- Si consideri il seguente frammento di codice in Java:

```
public class TestPila {
    public static void main(String[ ] args) {
        Pila p = new PilaVuota();
        System.out.println(p.vuota());
        p = p.inserisci(2);
        System.out.println(p.vuota());
        System.out.println(p.leggiCima());
        p = p.inserisci(3);
        System.out.println(p.leggiCima());
        p = p.estrai();
        System.out.println(p.leggiCima());
        p = p.estrai();
        System.out.println(p.leggiCima());
    }
}
```

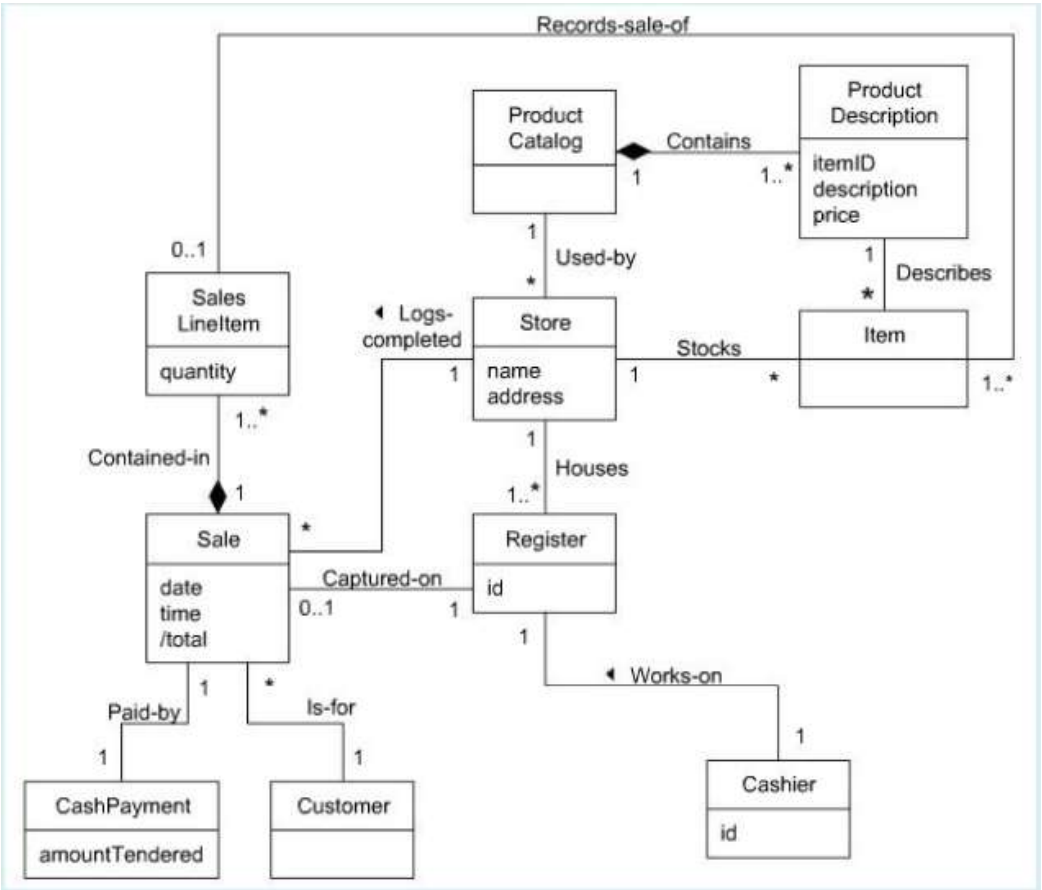
La cui esecuzione restituisce il seguente risultato a console:

```
true
false
2
3
2
Exceptioninthread "main" java.lang.AssertionError
    at PilaVuota.leggiCima(PilaVuota.java: 18)
    at TestPila.main(TestPila.java: 15)
```

Presentare un modello di progetto (diagramma UML delle classi, si può usare UMLetino) che utilizzando il pattern composite realizzi il comportamento descritto.



- Si consideri il seguente Modello di Dominio, dire se le seguenti affermazioni sono vere o false. Si supponga: SalesLineItem = {s1,s2} Item = {i1,i2,i3,i4}



    - Records-sale-of può essere: {(s1,i1),(s1,i2),(s2,i1)} → **FALSO**
    - Records-sale-of può essere: {(s1,i1),(s2,i1)} → **FALSO**
    - Records-sale-of può essere: {(s1,i1),(s2,i2)} → **VERO (NB: non è obbligatorio che ci siano tutte le istanze di Items perchè SalesLineItem ha cardinalità minima 0)**
    - Records-sale-of può essere: {(s1,i1),(s1,i2)} → **FALSO (NB: è invece obbligatorio che ci siano tutte le istanze di SalesLineItem)**
    - Records-sale-of può essere: {(s1,i1),(s1,i2),(s2,i3),(s2,i4)} → **VERO**

NB: quando la cardinalità minima è 0, è comunque obbligatorio che ci siano TUTTE le istanze della classe con cardinalità minima 0. Non è invece obbligatorio che ci siano TUTTE le istanze dell'altra classe (l'altra classe può non avere tutti gli elementi perchè associati a NESSUNA istanza della classe con cardinalità minima 0).
Invece quando la cardinalità minima è 1 per entrambe le classi è OBBLIGATORIO che ci siano TUTTE le istanze di ENTRAMBE le classi.

- Associare le corrette percentuali
    - In percentuale sul totale dei requisiti funzionali di un progetto quanti sono quelli che dovrebbero essere identificati entro la conclusione della fase di elaborazione? → **QUASI IL 100%**
    - In percentuale sul totale di casi d'uso complessivi di un progetto, quanti sono quelli che dovrebbero essere realizzati entro la conclusione della fase di ideazione? → **5%**
    - In percentuale sul totale dei requisiti funzionali di un progetto, quanti sono quelli che dovrebbero essere identificati entro la conclusione della fase di ideazione? → **TRA IL 50% E IL 70%**
- Dire se le seguenti affermazioni sono vere o false
    - La narrativa di un caso d'uso viene espressa a livello delle intenzioni dell'utente e delle responsabilità del sistema. → **VERO**
    - La narrativa di un caso d'uso viene espressa a livello delle azioni concrete dell'utente e delle responsabilità del sistema. → **FALSO**
    - Un caso d'uso è sempre completato (implementato) in una sola iterazione → **FALSO**
- Quale pattern GRASP è corretto utilizzare per i seguenti problemi?
    - Chi crea un oggetto A? Ovvero chi deve essere responsabile della creazione di una nuova istanza della classe? **PATTERN CREATOR**
    - Qual è il primo oggetto oltre lo strato UI che riceve e coordina un'operazione di sistema?**PATTERN CONTROLLER**
    - Come mantenere gli oggetti focalizzati, comprensibili e gestibili e, come effetto collaterale, sostenere Low Coupling? **Pattern High Cohesion**
    - Qual è il principio di base, generale per l'assegnazione di responsabilità agli oggetto? **Information Expert (o Expert)**
    - Come ridurre l'impatto dei cambiamenti? Come sostenere una dipendenza bassa, un impatto dei cambiamenti basso e una maggiore opportunità di riuso? **Pattern Low Coupling**
- Quali delle seguenti affermazioni relative agli SSD sono vere e quali sono false?
    - I contratti costituiscono un input per gli SSD delle operazioni e per la progettazione degli oggetti → **FALSE**
    - Sono espressi attraverso i diagrammi di comunicazione di UML → **FALSE**
    - Mostrano l'ordine degli eventi generati dagli attori esterni al sistema → **TRUE**

    - Mostrano gli eventi generati dagli attori esterni al sistema → **TRUE**
    - Un evento di sistema è un evento interno al sistema → **FALSE**
- Quali delle seguenti affermazioni sul modello di dominio sono vere e quali false?
    - Rappresentazione visuale delle classi Java → **FALSO**
    - Insieme di diagrammi di classi UML che includono associazioni tra classi concettuali → **VERO**
    - Rappresentazione grafica degli oggetti software → **FALSO**
    - Insieme di diagrammi di classi UML che includono le responsabilità di fare → **FALSO**
    - Rappresentazione visuale delle classi concettuali → **VERO**
    - Insieme di diagrammi di classi UML che includono associazioni tra classi software → **FALSO**
- Quali delle seguenti affermazioni sono vere e quali false?
    - I casi d'uso mettono in risalto gli obiettivi degli utenti → **VERO**
    - I casi d'uso sono utilizzati per la scoperta e la definizione dei requisiti non funzionali → **FALSO**
    - I casi d'uso sono utilizzati per la scoperta e la definizione dei requisiti funzionali → **VERO**
    - I casi d'uso mettono in risalto gli obiettivi del sistema → **FALSO**
    - I casi d'uso sono utilizzati solo nelle prime iterazioni di sviluppo → **FALSO**
- Si consideri il seguente frammento di codice in Java:

```
import java.io.*;
public classProvaFile {
   public static void main(String arg[ ]) throws Exception {
      InputStream is = new FileInputStream("DatiNumerici.txt");
      FilterInputStream bis = new BufferedInputStream(is);
      FilterInputStream dis = new DataInputStream(bis);
      int i;
      char c;
      try {
         while ((i = dis.read())!= -1) {
            c = (char) i;
            System.out.print(c);
         }
      } catch (Exception e) {
         e.printStackTrace();
      } finally {
         if (dis != null) dis.close();
      }
   }
}
```

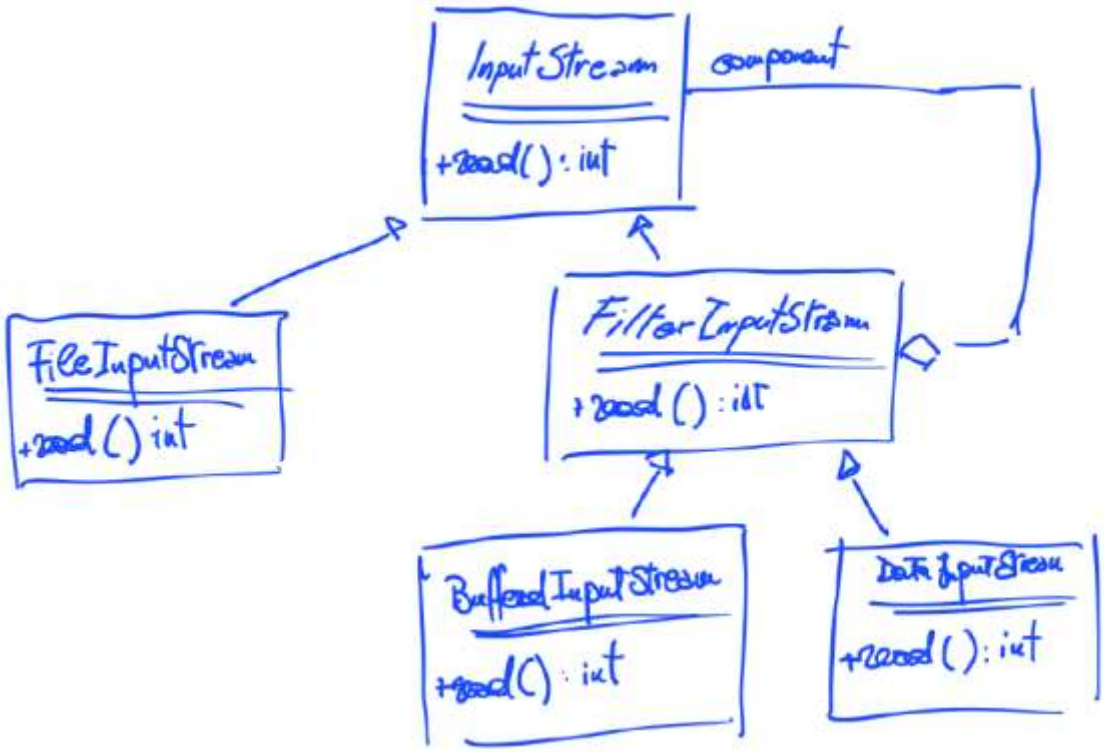
Il file DatiNumerici.txt è il seguente:

```
9
13
7
```

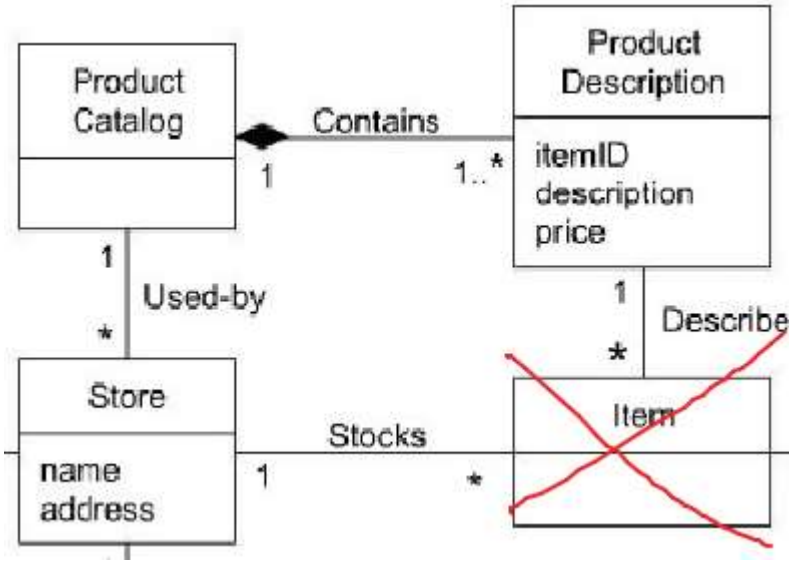
l'secuzione del main di ProvaFile restituisce il seguente risultato a console:

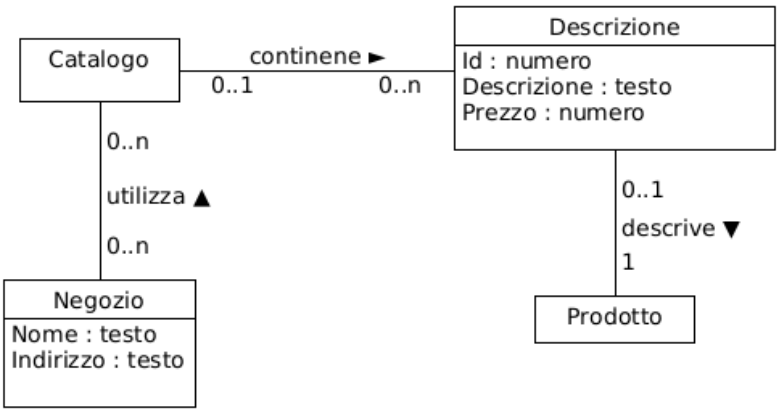
```
9
13
7
```

Presentare un modello di progetto (diagramma UML delle classi) che utilizzando il pattern decorator realizzi il comportamento descritto.

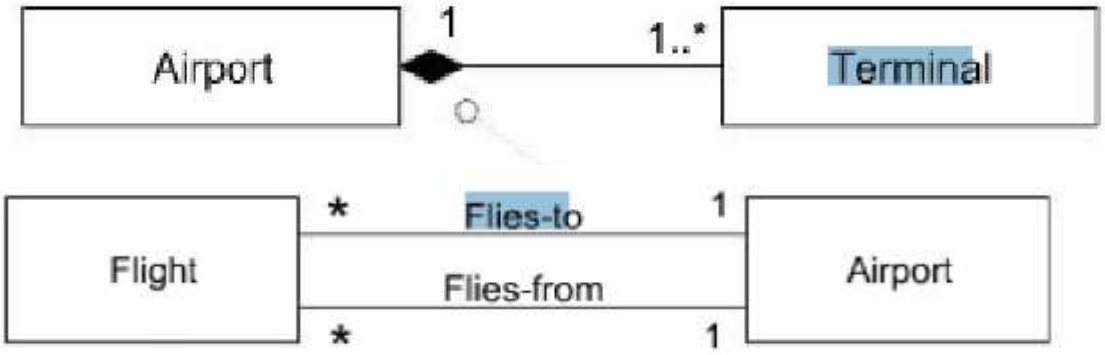


- “Un catalogo di prodotti è costituito da più descrizioni di prodotto. Ogni descrizione di prodotto è costituita da un identificatore, una descrizione e un prezzo. Ogni catalogo di prodotti è utilizzato da alcuni negozi, identificati da un nome e un indirizzo.” Disegnare il modello di dominio (utilizzando UML).





- Un aeroporto è costituito da uno o più terminali. Un volo ha origine da un solo aeroporto e ha destinazione in un solo aeroporto. Lo stesso aeroporto può essere originario di più voli e destinazioni di più voli. Disegnare il modello di dominio utilizzando UML.



- Dire a quale pattern GoF la seguente implementazione della struttura dati albero binario si conforma. Disegnare la struttura del pattern e il diagramma delle classi della soluzione applicata alla struttura dati albero binario.

```
public abstract class Tree {
   public abstract boolean empty();
   public abstract i n t getRootElem();
   public abstract void stampapostvisita();
}
public class Leaf extends Tree {
   public Leaf() { }
   public boolean empty() {
      return true ;
   }
   public int getRootElem() {
      assert false; return 0;
   }
   public void stampapostvisita() { }
}
public class Branch extends Tree {
   private int elem;
   private Tree left;
   private Tree right;
   public Branch(int elem, Tree left, Tree right) {
      this.elem = elem;
      this.left = left;
      this.right = right;
   }
   public boolean empty() {
      return false;
   }
   public int getRootElem() {
      return elem;
   }
   public void stampapostvisita() {
      right.stampapostvisita();
      left.stampapostvisita();
      System.out.print(this.getRootElem() + " " );
   }
}


public classTestTree {
   public static void main(String[ ] args) {
      ...
      System.out.println("Stampo albero postvisita");
      t.stampapostvisita();
      System.out.println("");
   }
}
```

Il pattern GoF utilizzato è il pattern Composite.
Struttura del pattern:

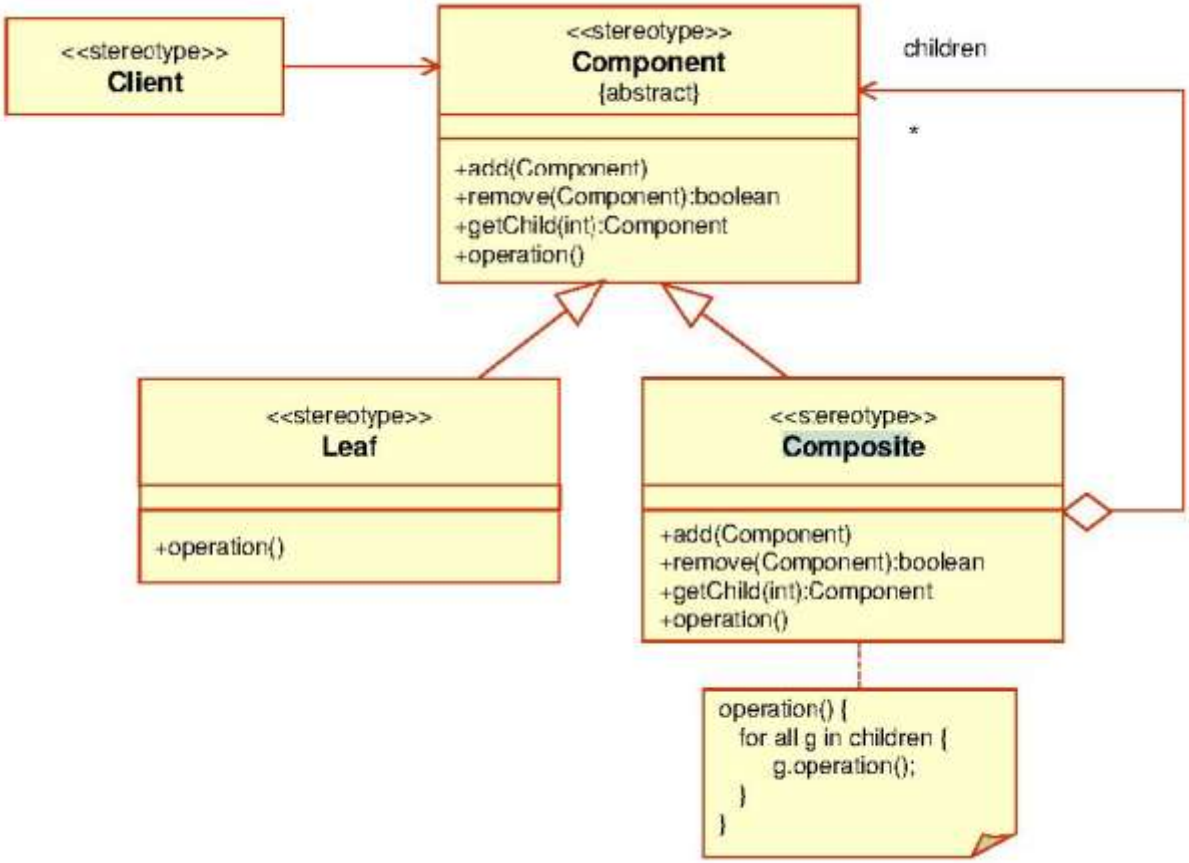
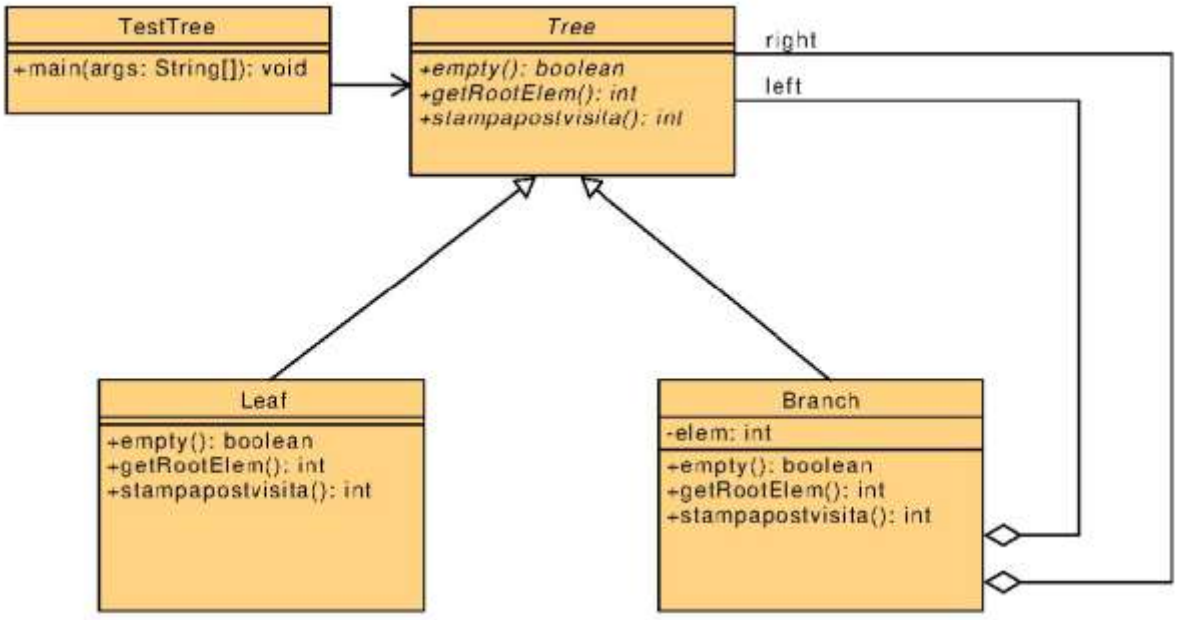


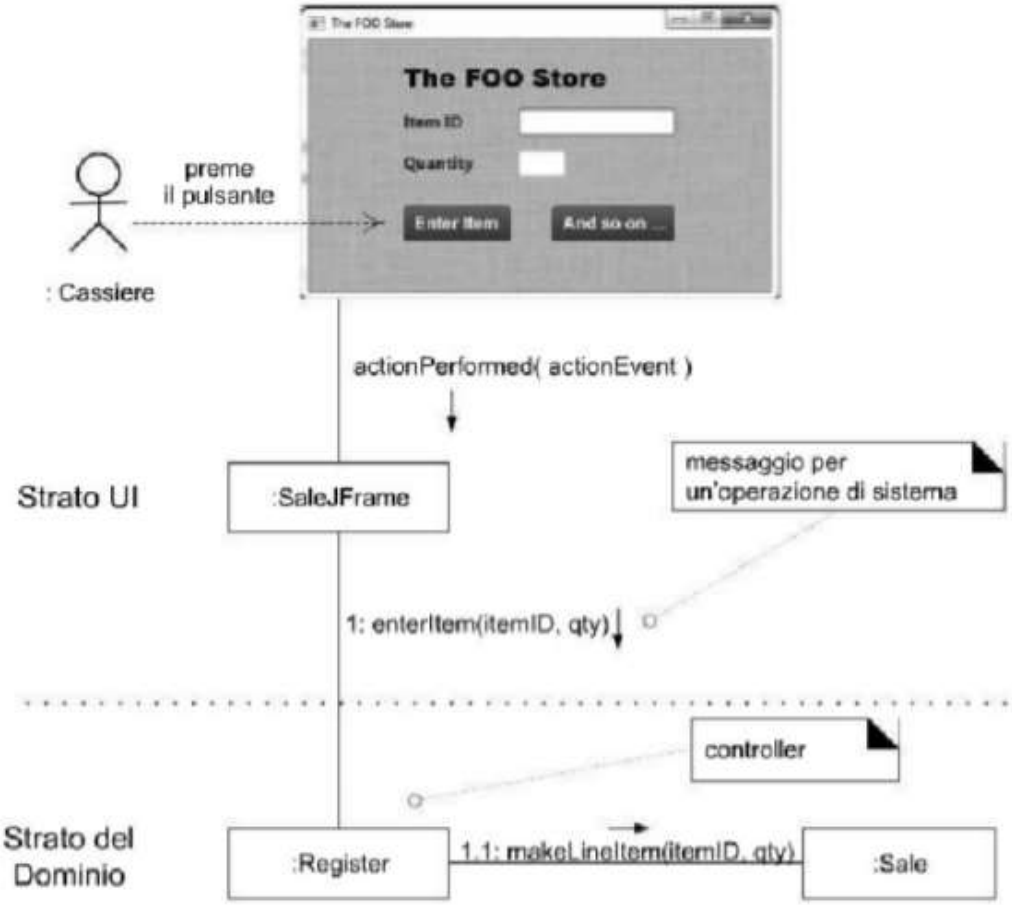
Diagramma delle classi:



- Definire in modo preciso la pre-condizione e la post-condizione di un'operazione di sistema. Fare un esempio di operazione con le sue pre e post condizioni.

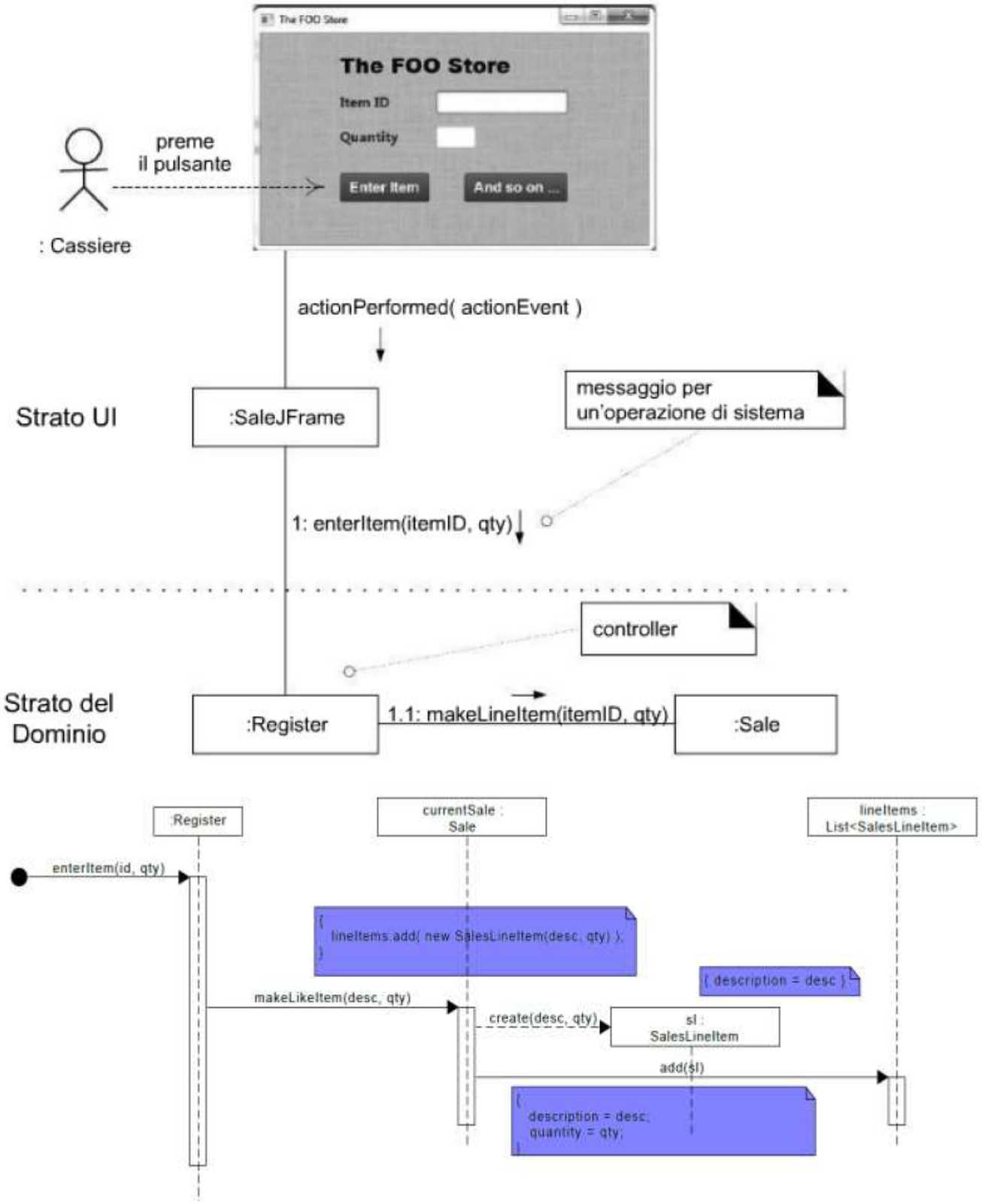
Una pre-condizione definisce lo stato del sistema e degli oggetti del Modello di Dominio prima di eseguire l'operazione (si tratta di ipotesi non banali). Deve essere

sempre vera prima di iniziare il caso d'uso. La post condizione descrive il cambiamento negli oggetti del Modello di Dominio dopo l'esecuzione con successo del caso d'uso dove il cambiamento prevede aggiunta/rimozione di collegamenti, attributi modificati o oggetti creati.

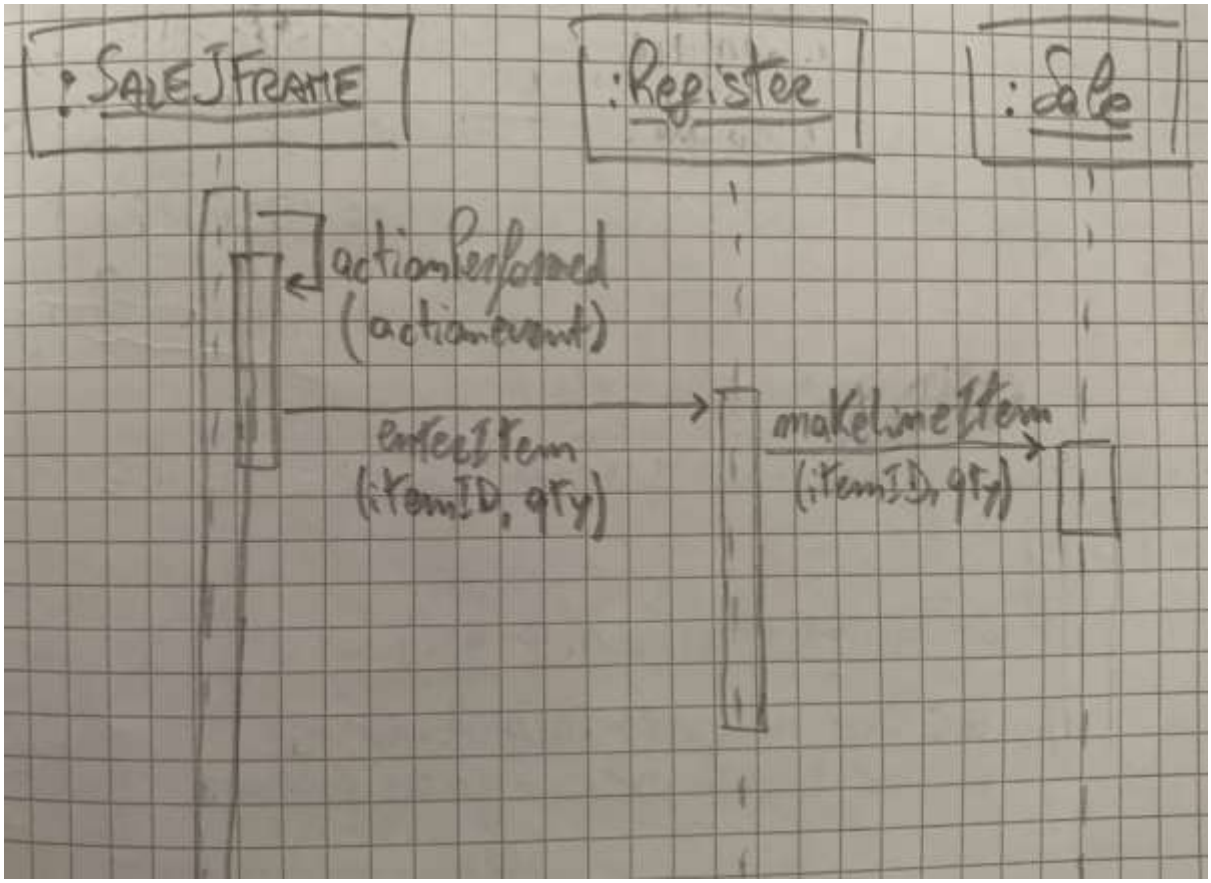


- Si consideri l'applicazione NextGen vista a lezione e presentata sul libro di testo. Si supponga che abbia una finestra (realizzata mediante la classe SaleJFrame) che visualizza le informazioni sulla vendita e che cattura le operazioni del cassiere, la classe SaleJFrame è inoltre ascoltatore degli eventi generati dalla pressione dei bottoni della finestra (cioè implement il metodo actionListener). Sono inoltre presenti le classi Register, un'astrazione dell'unità fisica, e la classe Sale che rappresenta la vendita in corso.
    - Si completi il diagramma di sequenza sottostante (si assuma la figura della finestra rappresentativa dell'IU senza preoccuparsi dei suoi dettagli ma come fosse un'entità unica) inserendo i messaggi actionPerformed(actionEvent), enterItem(itemID, qty) e makeLineItem(itemID, qty). Si noti che il messaggio enterItem(itemID, qty) è anche il nome del messaggio individuato dall'operazione di sistema che si sta realizzando.

Nelle slide è così:



Riadattato così:

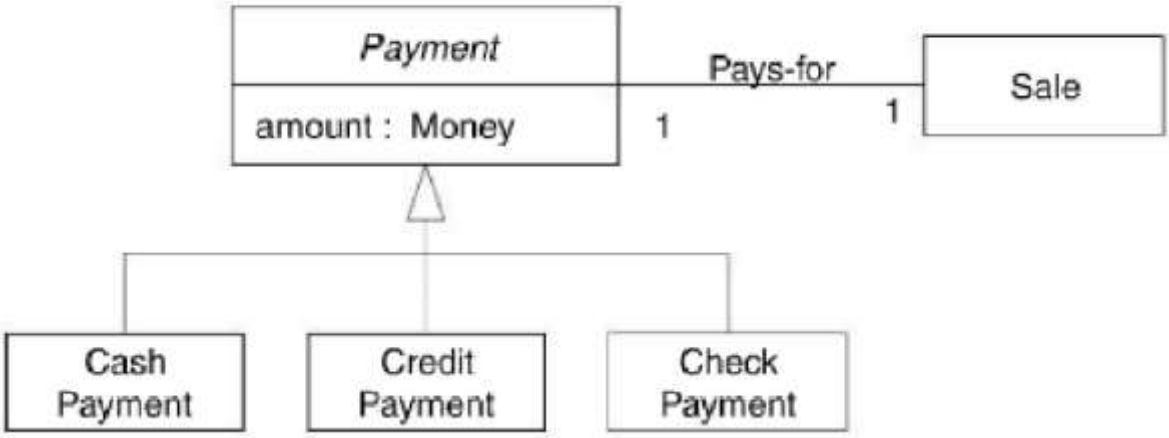


- Si indichi quali classi appartengono allo strato UI e quali allo strato di dominio.

La classe SaleJFrame appartiene allo stato UI e le classi Register e Sale allo stato di Dominio

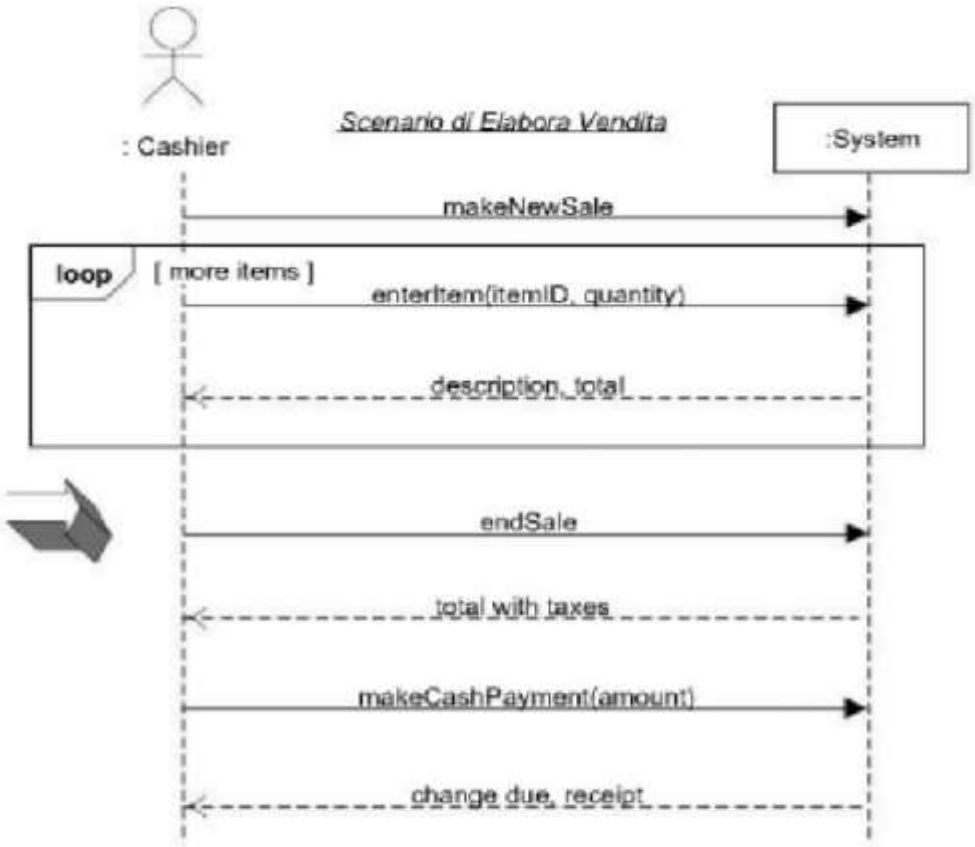
- Si indichi la classe che rappresenta il controllore GRASP. **REGISTER**

- Rappresentare mediante un diagramma delle classi (modello di dominio) le seguenti affermazioni:
  - una vendità è saldata da un pagamento;
  - un pagamento è necessariamente associato ad una vendita;
  - una vendità è necessariamente saldata;
  - un pagamento per contanti, con assegno o con carta di credito, è un pagamento;
  - un pagamento è caratterizzato da un ammontare di denaro.



- Si consideri il seguente scenario di base di Elabora Vendita:
    1. Il cliente arriva alla cassa POS con gli articoli e/o i servizi da acquistare
    2. Il cassiere inizia una nuova vendita
    3. Il cassiere inserisce il codice identificativo di un articolo
    4. Il Sistema registra la riga di vendita per l'articolo e mostra una descrizione dell'articolo, il suo prezzo e il totale parziale
    5. Il cassiere ripete i passi 3-4 fino a che non indica che ha terminato.
    6. Il sistema mostra il totale.
    7. Il Cassiere riferisce il totale al Cliente, e richiede il pagamento.
    8. Il cliente paga (in contati) e il sistema gestisce il pagamento.
    9. Il sistema registra la vendita completa.
    10. Il sistema genera la ricevuta.
    11. Il cliente va via con la ricevuta e gli articoli acquistati.

    Si disegni lo SSD corrispondente



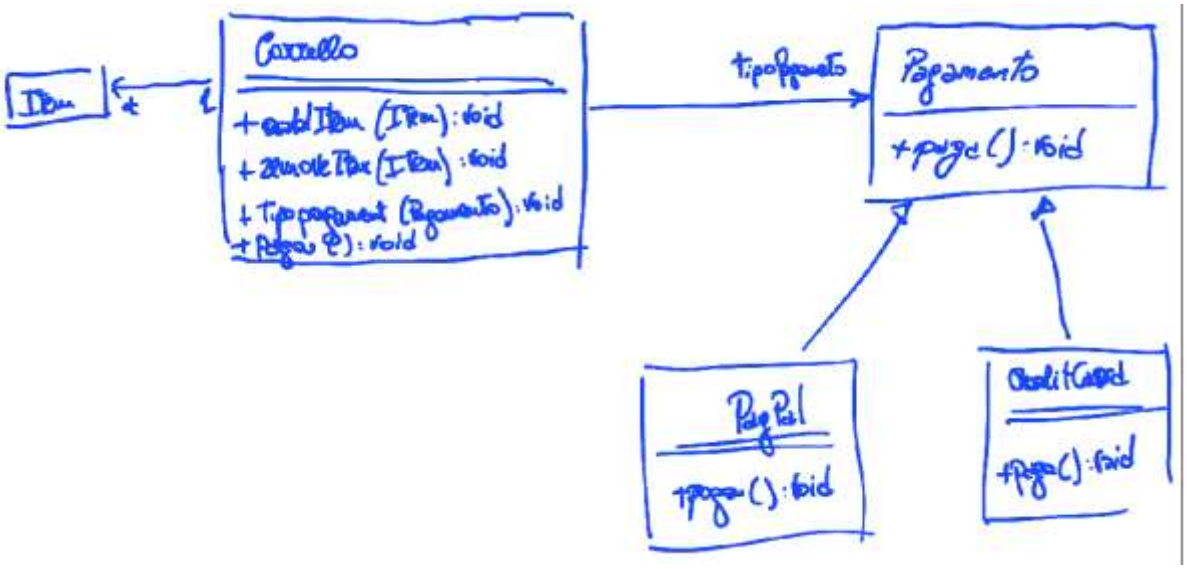
- Dire quali delle seguenti affermazioni sono vere e quali false:
  - La responsabilità è un'astrazione di ciò che fa o rappresenta un oggetto o un componente software → **VERO**
  - Le responsabilità sono assegnate ai concetti definiti nel modello di dominio → **VERO**
  - In UML per responsabilità si intende un contratto o un obbligo di un classificatore → **VERO**
  - Nel Responsibility-Drive Development gli oggetti software sono considerati come dotati di responsabilità → **VERO**
- Quali delle seguenti affermazioni sui contratti sono vere e quali false
  - il principale input per la redazione dei contratti sono le operazioni di sistema e il modello di dominio → **VERO**
  - Usano pre-condizioni e post-condizioni per descrivere nel dettaglio i cambiamenti agli oggetti in un modello di progetto (software) → **FALSO**
  - I contratti servono come input per il modello di dominio. → **FALSO**
  - Sono considerati parte del Modello dei Casi d'Uso, poichè forniscono maggiori dettagli dell'analisi sull'effetto delle operazioni di sistema → **FALSO**
  - Le pre-condizioni descrivono i cambiamenti di stato degli oggetti nel modello di dominio dopo il completamento dell'operazione → **FALSO**
- Quali delle seguenti affermazioni sui casi d'uso sono vere e quali false
  - I casi d'uso servono a catturare i requisiti funzionali → **VERO**
  - nei processi a cascata, l'analisi e la progettazione si basano sulla realizzazione di casi d'uso → **FALSO**

    - sono una collezione di soli scenari di successo che descrivono un attore che usa il sistema per raggiungere un obiettivo specifico. → **FALSO**
    - i casi d'uso sono descrizioni testuali di scenari di uso interessanti del sistema software che si deve realizzare → **VERO**
    - all'inizio del progetto vengono definiti in dettaglio tutti i requisiti derivanti dai casi d'uso → **FALSO**
- Dire se le seguenti affermazioni su UP sono vere o false
    - la fase di Ideazione viene prima della fase di Elaborazione. → **VERO**
    - la fase di ideazione viene dopo la fase di Costruzione o viceversa? **PRIMA**
    - La fase di ideazione è suddivisa in iterazioni? → **FALSO**
    - La fase di ideazione è una visione raffinata, implementazione iterativa del nucleo dell'architettura, risoluzione dei rischi maggiori, identificazione della maggior parte dei requisiti e della portata. → **FALSO (è elaborazione)**
- Si consideri il seguente codice:

```
public class TestAmmazzon {
    public static void main(String[] args) {

        Carrello cart1 = new Carrello();

        Item crema = new Item("Crema da barba", 6);
        Item rasoio = new Item("Rasoio elettrico", 75);
        Item dopobarba = new Item("Dopobarba", 12);

        cart1.addItem(crema);
        cart1.addItem(rasoio);
        cart1.addItem(dopobarba);

        cart.removeItem(crema);

        Pagamento pagapaypal = new PayPal("matteo@baldoni.it", "passwd");
        cart.tipopagamento(pagapaypal);

        cart.paga();

        Carrello cart2 = new Carrello();

        Item altracrema = new Item("Crema da barba", 6);
        Item altrorasoio = new Item("Rasoio a lama", 9);
        Item altrodopobarba = new Item("Crema lenitiva", 10);

        cart2.addItem(altracrema);
        cart2.addItem(altrorasoio);
        cart2.addItem(altrodopobarba);

        Pagamento cc = new CrediCard("Matteo Baldoni", "1234123412341234", "789", "12/21");
        cart2.tipopagamento(cc);

        cart.paga();

        }

    }
}
```

  Questo utilizza un insieme di classi che realizzano un carrello per un sito e-commerce utilizzando un noto pattern GoF. Si dica quale patter si tratta e disegnare il diagramma UML delle classi coinvolte.

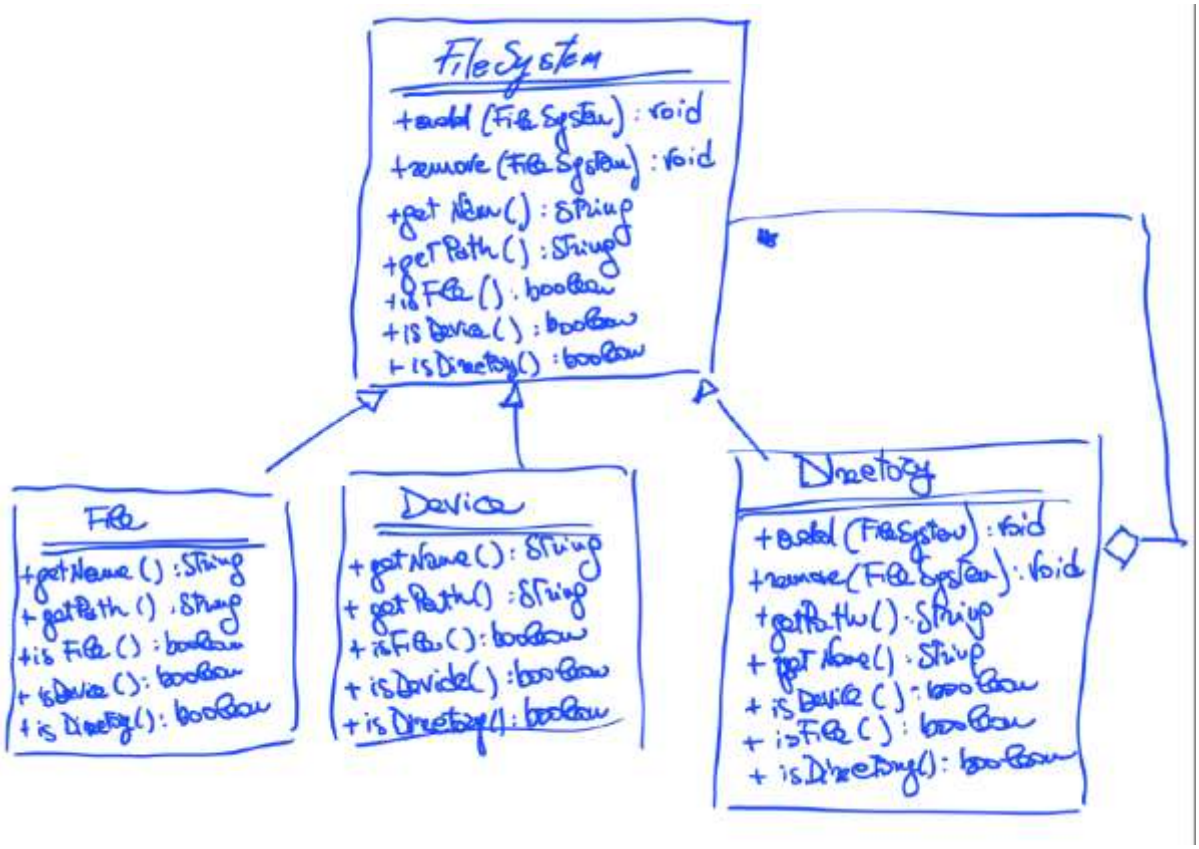
Il pattern è Strategy



- Si consideri la seguente codice:

```
public class TestFileSystem {
    public static void main(String[] args) {

        FileSystem root = new Directory("/");
        FileSystem devices = new Directory("/dev");
        FileSystem std_out = new Device("/dev/console");
        FileSystem home = new Directory("/home");
        FileSystem baldoni = new Directory("/home/baldoni");
        FileSystem compitoA = new File("/home/baldoni/compitoA.pdf");
        FileSystem compitoB = new File("/home/baldoni/compitoB.pdf");

        root.add(devices);
        root.add(home);

        devices.add(std_out);
        home.add(baldoni);

        baldoni.add(compitoA);
        baldoni.add(compitoB);
        baldoni.delete(compitoA);

        System.out.println("Path: " + compitoB.getName() + ": " + compitoB.getPath());

        System.out.println(std_out.getName() + " is a " + std_out.isFile());
        System.out.println(std_out.getName() + " is a " + std_out.isDevice());
        System.out.println(std_out.getName() + " is a " + std_out.isDirectory());
        System.out.println(compitoB.getName() + " is a " + compitoB.isFile());
        System.out.println(compitoB.getName() + " is a " + compitoB.isDevice());
        System.out.println(baldoni.getName() + " is a " + baldoni.isDirectory());
```

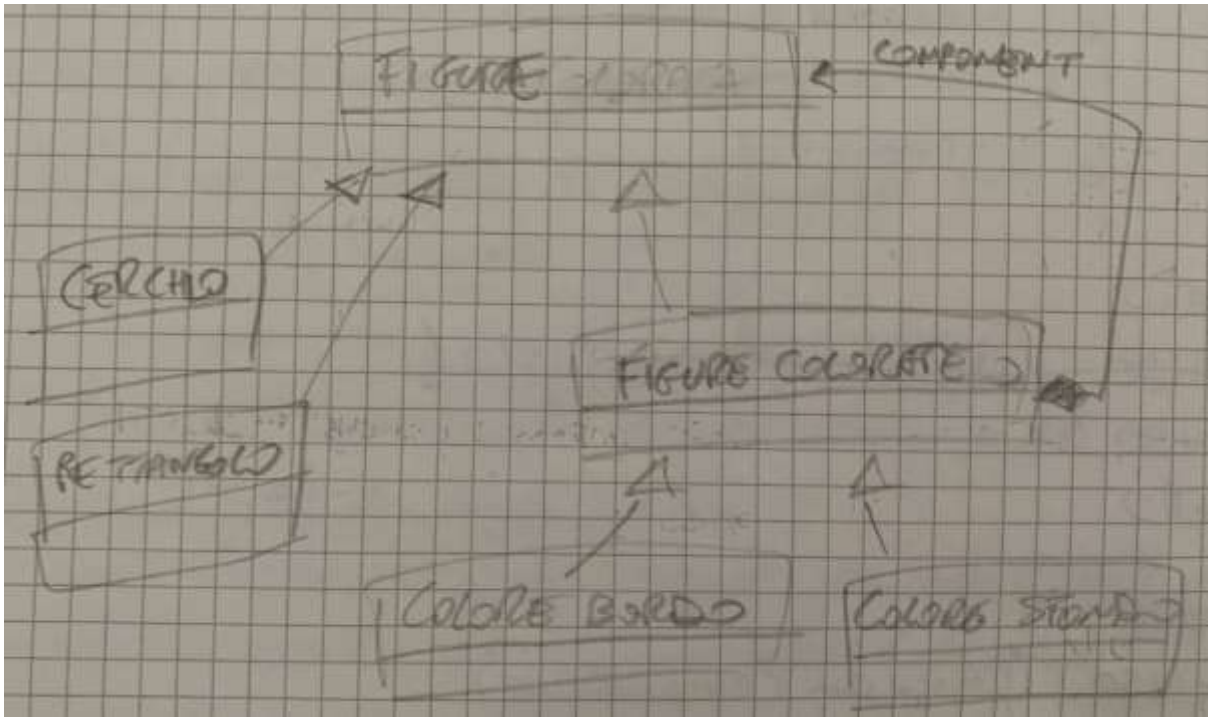
Il pattern è Composite

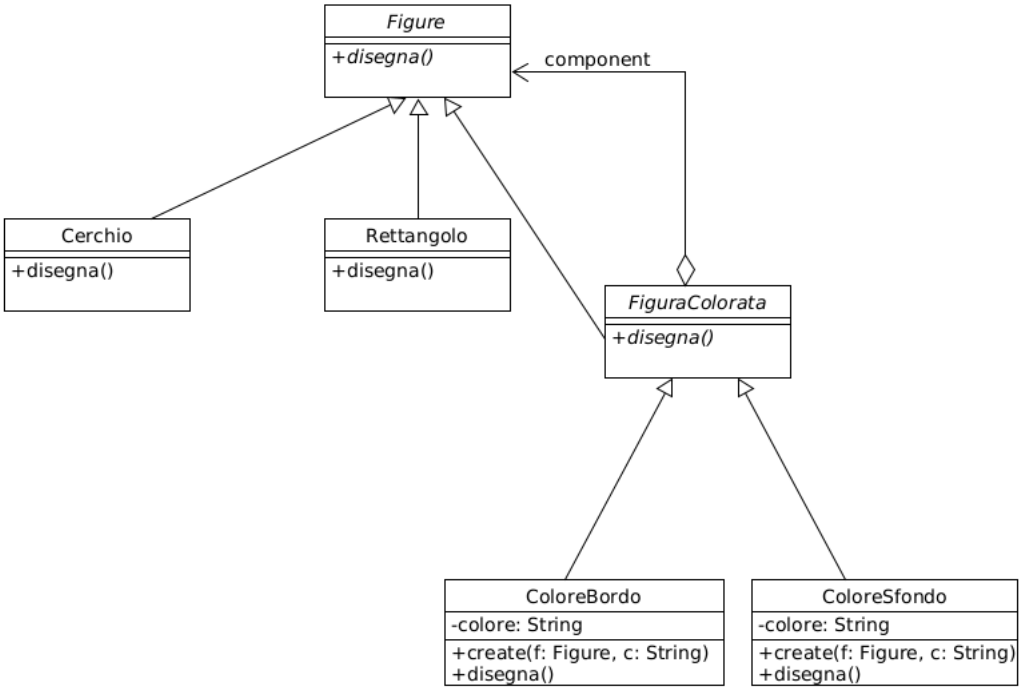



```
public class TestPaint {
    public static void main(String[ ] args) {
        Figure[ ] figure = new Figure[5];

        figure[0] = new Cerchio();
        figure[1] = new Rettangolo();

        FiguraColorata cerchioBordoRosso = new ColoreBordo(new Cerchio(), "rosso");
        FiguraColorata cerchioRosso = new ColoreSfondo(cerchioBordoRosso, "rosso");

        figure[2] = cerchioRosso;

        FiguraColorata cerchioSfondoBlu = new ColoreBordo(new Cerchio(), "blu");
        FiguraColorata cerchioBlu = new ColoreSfondo(cerchioSfondoBlu, "blu");

        figure[3] = cerchioBlu;

        FiguraColorata rettangoloSfondoBlu = new ColoreSfondo(new Rettangolo(), "blu");

        figure[4] = rettangoloSfondoBlu;
        for (int i = 0; i < 5; i++) {
            figura[i].disegna();
        }
    }
}
```

Questo utilizza un insieme di classi che realizzano un programma di disegno utilizzando un noto pattern GoF. Si dica di quale pattern si tratta e disegnare il diagramma UML delle classi coinvolte.

Il pattern è Decorator (non ne sono sicuro)





- Dire se le seguenti affermazioni sono vere o false
    - Le pre-condizioni sono ipotesi significative dello stato del sistema o degli oggetti del modello di progetto prima dell'esecuzione dell'operazione a cui è associata → **VERO**
    - La post-condizione descrive i cambiamenti di stato degli oggetti nel Modello di Dominio dopo il completamento dell'operazione → **VERO**

    - Le post-condizioni descrivono i cambiamenti nello stato degli oggetti del modello di progetto. I cambiamenti dello stato del modello di progetto comprendono gli oggetti creati, i collegamenti formati o rotti e gli attributi modificati → **FALSO**
- Dire se le seguenti affermazioni sono vere o false
    - Una sottoclasse è fortemente accoppiata alla sua superclasse → **VERO**
    - Ricevere un parametro di tipo A per un metodo di una classe B rappresenta una relazione di dipendenza da B a A mentre estendere una classe A per una classe B non rappresenta una relazione di dipendenza da B a A → **FALSO**
    - E' possibile che due classi siano collegate da più di una associazione → **VERO**