UML
Esercitazioni

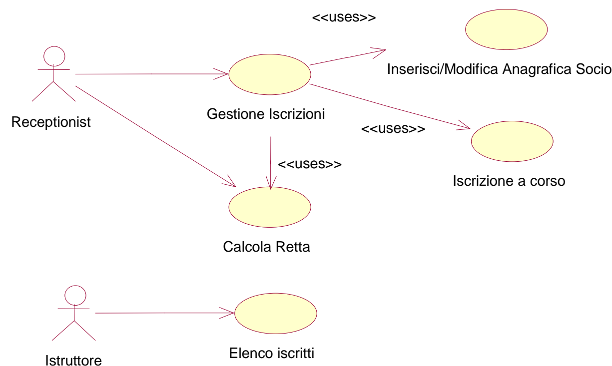
Circolo Sportivo

Descrizione informale del dominio del problema

- Si consideri un circolo sportivo in cui è possibile praticare nuoto, ginnastica aerobica, body building. Gli iscritti al circolo possono praticare un numero qualsiasi di queste discipline, ciascuna avente un costo mensile come di seguito specificato: nuoto €. 25, ginnastica aerobica € 40, body building € 50.
- Progettare una applicazione Java che consenta di memorizzare l'insieme degli iscritti al circolo e che registri per ogni iscritto il nome e le discipline che pratica.
- Si richiede:
- - che sia possibile, dato in input un nome, ottenere il costo della retta mensile (somma dei costi delle discipline praticate) se la persona corrispondente è iscritta al circolo, il messaggio "non è iscritto al circolo" in caso contrario.
- - che sia possibile stampare la lista di tutti gli iscritti ad una certa disciplina.

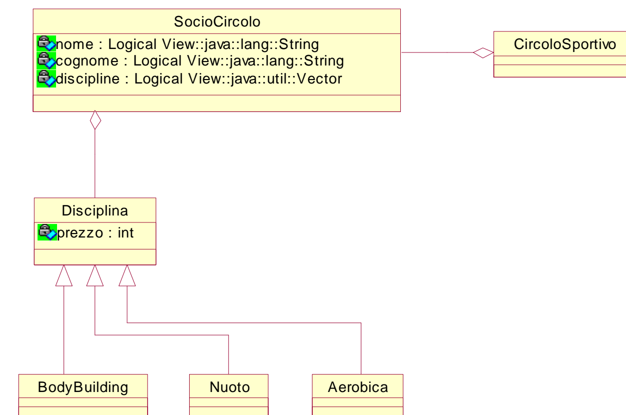
Specifica dei requisiti

use case diagram



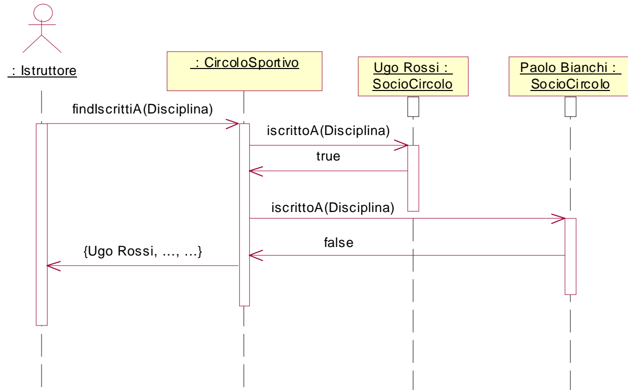
Specifica dei requisiti

class diagram



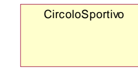
Specifica dei requisiti
sequence diagram

Descrizione dello use case *"Elenco Iscritti"* con un sequence diagram che descrive uno scenario in cui l'istruttore chiede la stampa dell'elenco iscritti.

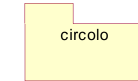


...verso l'implementazione....

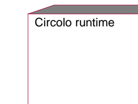
- L'architettura è centralizzata con solo componente



- Il codice è organizzato in un unico package

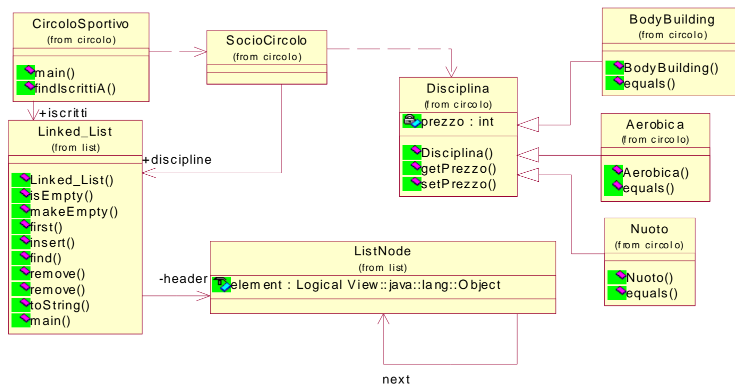


- A runtime c'è un solo processo in esecuzione su di un unico processore



Il circolo sportivo con la Linked_List

Dal diagramma delle classi ricavato dal reverse engeneering del codice Java effettuato con Rational Rose si evidenziano i dettagli dell'implementazione. Bisogna valutare di volta in volta se vale la pena di mostrare questi dettagli nel modello di progetto e se è necessario usare altri diagrammi (ad esempio sequence diagrams) per completare e/o arricchire la descrizione del sistema.

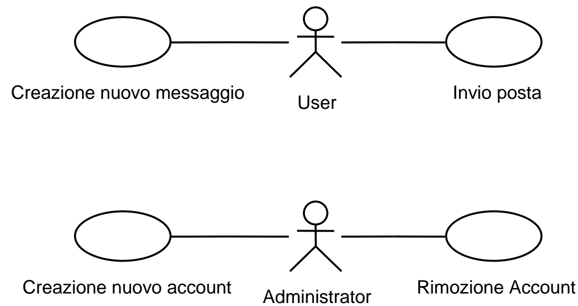


Mail system
Descrizione informale del dominio del problema

- In un sistema di posta elettronica ogni utente è caratterizzato dal suo username.
- Per ogni utente esistono tre mailbox (inbox, outbox, draft) che, per semplicità, consideriamo insiemi non ordinati di messaggi.
- Si specifichino le operazioni di creazione di una mail e di sua spedizione:
  - La prima operazione crea una nuova mail nella outbox dell'utente.
  - La seconda operazione preleva i messaggi in outbox e li deposita nelle inbox dei destinatari.

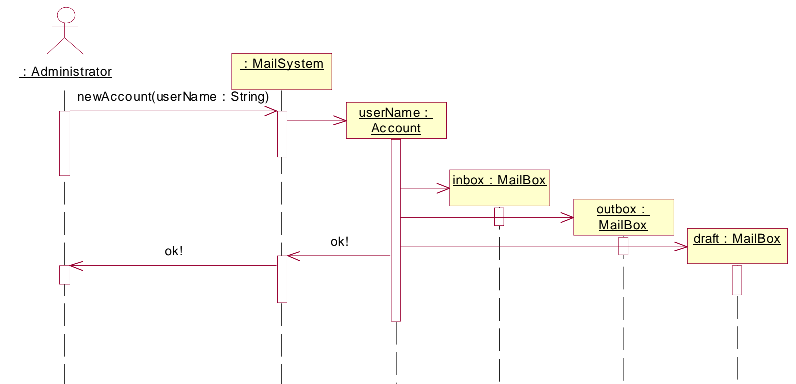
Specifica dei requisiti

use case diagram



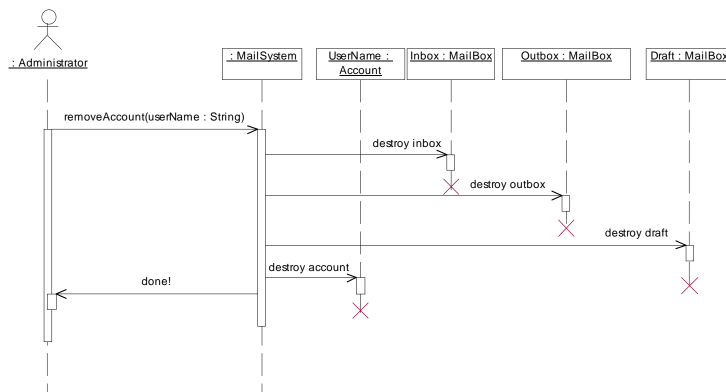
Use Case “creazione nuovo account”

descrizione di uno scenario con un sequence diagram



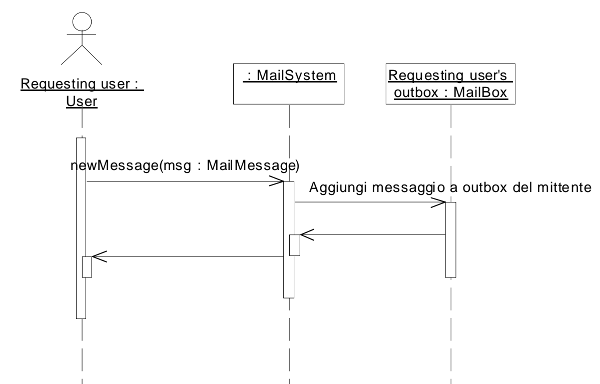
Use case “Rimozione account”

descrizione di uno scenario con un sequence diagram



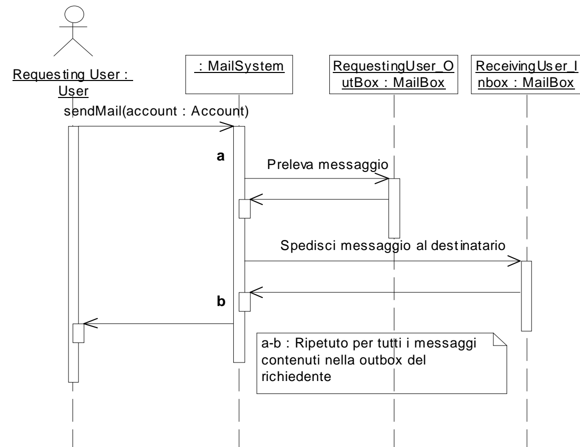
Use case “Creazione nuovo messaggio”

descrizione di uno scenario con un sequence diagram



USe case “Invio posta”

descrizione di uno scenario con un sequence diagram



Specifica dei requisiti

Diagramma delle classi

- Struttura statica delle *entità rilevanti* del problema
  - Classi
  - Proprietà delle classi
  - Relazioni fra classi
- Costruzione
  - Identificazione delle entità rilevanti (classi)
  - Identificazione delle relazioni fra classi
  - Identificazione degli attributi
  - Identificazione delle relazioni di generalizzazione/specializzazione
  - Eventuale suddivisione in moduli

Diagramma delle classi

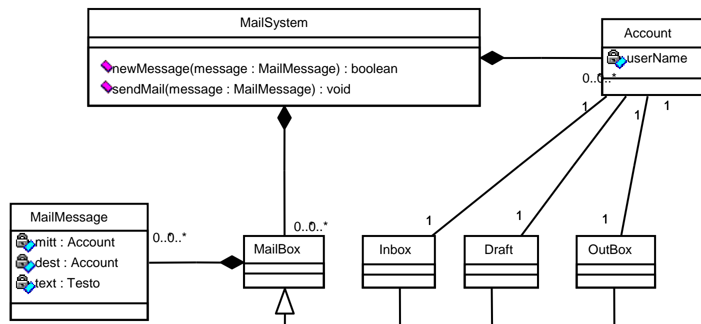
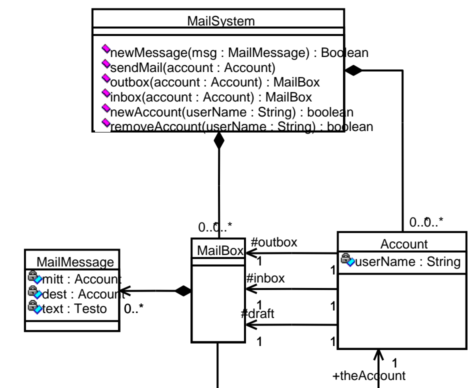


Diagramma delle classi alternativo

- Anche questo diagramma delle classi è conforme alla specifica testuale, nel senso che descrive la stessa situazione in cui ad un *Account* sono associate tre mailbox (un *inbox*, un *outbox* e un *draft*).
- Nel diagramma precedente *Inbox, Outbox e Draft* sono sottoclassi di MailBox
- Nel class diagram a lato *outbox, inbox e draft* sono i ruoli delle associazioni tra le classi Account e MailBox
- I significati sono leggermente diversi, bisogna fare attenzione a cosa si vuole comunicare



Un sistema informativo per una catena di negozi

Descrizione di massima delle esigenze

- Una catena di negozi decide di dotarsi di un sistema di registratori di cassa "intelligenti" (con una scheda a microprocessore "embedded" e virtual machine Java)
- Un registratore di cassa "intelligente" può essere "programmato" (Java) per inviare e richiedere dati ad un calcolatore a cui è connesso tramite LAN
  - Inviare dati sugli scontrini emessi
  - Ricevere dati su statistiche, rendiconti, … e mostrarli all'operatore
- La catena di negozi ha già un sistema informativo centrale ed un calcolatore periferico in ciascuno dei negozi, collegati fra di loro mediante una rete WAN
- Il sistema informativo centrale gestisce il magazzino degli articoli (listino)

Descrizione di massima delle esigenze

- L'utilizzo dei registratori di cassa intelligenti dovrebbe:
  - consentire un accesso "on-line" al listino prezzi da parte dell'operatore, per evitare laboriose operazioni di replicazione periodica (programmazione dei registratori di cassa)
  - permettere l'automazione della procedura di raccolta dati sugli scontrini emessi giornalmente (evitare di dover riportare i dati sugli scontrini emessi per ciascun registratore di cassa)
  - in prospettiva, consentire una gestione "on-line" della disponibilità degli articoli a magazzino, dei riordini, …
- Vincoli tecnologici/organizzativi:
  - Calcolatori centrali e periferici, applicazioni "legacy"
  - Rete WAN a banda limitata
  - Funzioni limitate dei registratori di cassa (descritte nel seguito)
  - Semplicità d'uso del sistema

Requisiti di massima (informali)

- In una catena di negozi ci sono dei registratori di cassa (POS) intelligenti, collegati ad un calcolatore dislocato in ogni negozio.
- Il calcolatore presente in ogni negozio è sempre in funzione e sempre collegato al calcolatore centrale (per altre applicazioni)
- Ogni POS è collegato con un dispositivo per la stampa degli scontrini, con un lettore di codici a barre e con una tastiera
- Il dispositivo per la lettura dei codici a barre invia il codice letto (stringa di caratteri) al POS ogni volta che viene utilizzato

Requisiti di massima (informali)

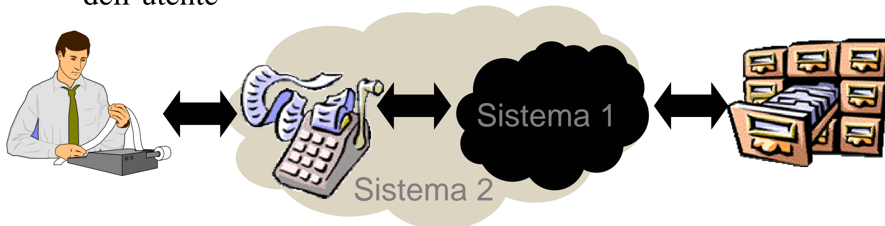
- La tastiera del POS ha 6 tasti con cui si possono attivare diverse funzioni:
  - *login* / *logout* / spegnimento del POS (3 tasti)
  - chiusura di uno scontrino (ed invio del totale al calcolatore del negozio, se previsto)
  - due tasti "programmabili" (possono essere usati per attivare funzioni inizialmente non previste), che si intendono usare per:
    - stampa del totale cumulativo di tutti gli scontrini emessi dal POS
    - stampa del totale cumulativo di tutti gli scontrini emessi da tutti i POS del negozio

Specifica dei requisiti

- Identificazione dei "confini" del sistema
- Identificazione degli "attori"
- Identificazione dei "casi d'uso"
- Descrizione di ogni caso d'uso attraverso l'identificazione di uno o più "scenari d'uso"
- Identificazione e descrizione delle "entità" rilevanti

Specifica dei requisiti: confini del sistema

- Confini del sistema:
  - Applicazione "legacy" per la gestione del magazzino
  - "Interfaccia" programmativa del registratore di cassa intelligente?
  - "Interfaccia utente" (ovvero, periferiche tastiera/stampante) del registratore di cassa?
- Decisione: descriveremo il "Sistema 2"
- Motivazione: si mostra il funzionamento dal punto di vista dell'utente



Specifica dei requisiti: identificazione degli attori

- Attori = "entità" che interagiscono, attivamente, con il sistema
- Per il "Sistema 1"
  - Registratore di cassa
  - Applicazione legacy magazzino
- Per il "Sistema 2"
  - Operatore del registratore di cassa
  - Applicazione legacy magazzino
- Nella nostra specifica: operatore e magazzino
- Nota: gli attori possono anche essere applicazioni!

Specifica dei requisiti: identificazione scenari d'uso

- Ogni caso d'uso rappresenta una funzionalità offerta dal sistema dal punto di vista degli attori
- Identificazione dei casi d'uso:
  - Accensione del POS e login
  - Emissione di uno scontrino
  - Stampa dei rendiconti
    - totale vendite del POS
    - totale vendite del negozio
  - Logout dell'operatore e spegnimento
- Identificazione di "scenari d'uso" per ✂:
  - Registrazione della vendita di un singolo articolo
  - Chiusura dello scontrino, calcolo e stampa del totale

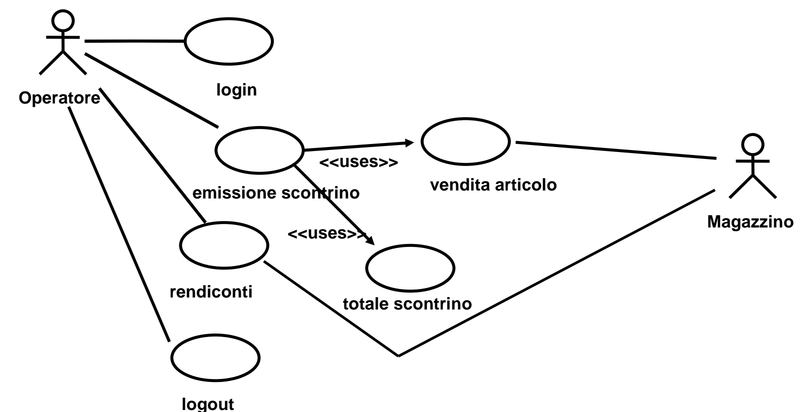
Descrizione informale dei casi d'uso

- Caso d'uso 1:
  - Il POS viene acceso dall'operatore
  - Il POS si collega al calcolatore periferico ed invia il proprio identificativo (login)
- Caso d'uso 2:
  - Scenario 2.1:
    - Un cliente si presenta alla cassa e l'operatore utilizza il lettore di codici a barre per rilevare il codice di ogni articolo acquistato
    - Il POS riceve il codice dell'articolo dal lettore e lo invia al calcolatore del negozio, che a sua volta lo invia al calcolatore centrale, il quale restituisce il prezzo dell'articolo, la percentuale dell'IVA e la descrizione dell'articolo

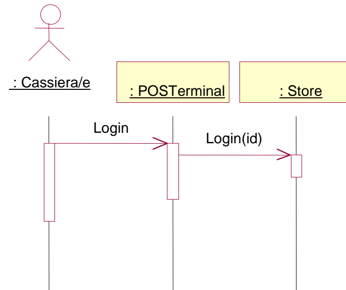
Descrizione informale dei casi d'uso

- Scenario 2.1 (continua):
  - Il calcolatore del negozio calcola il prezzo IVA inclusa dell'articolo e lo restituisce al POS
  - Il POS provvede a stampare una riga dello scontrino con la descrizione dell'articolo ed il prezzo IVA inclusa
- Scenario 2.2:
  - Al termine delle operazioni relative ad un cliente, la/il cassiera/e preme il tasto per la chiusura dello scontrino: il POS stampa il totale complessivo degli articoli acquistati
- Caso d'uso 3:
  - La/il cassiera/e richiede la stampa dei totali cumulativi di tutti i POS del negozio

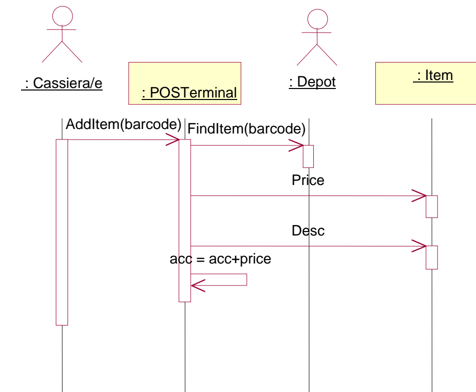
Casi d'uso: use case diagram



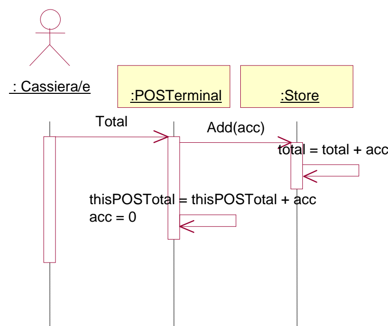
Descrizione dettagliata Scenario 1: login



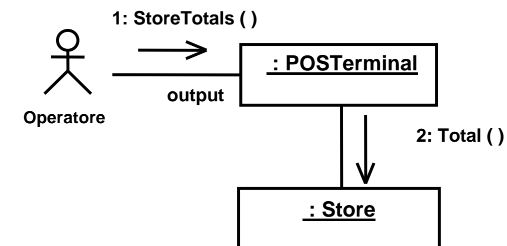
Descrizione dettagliata Scenario 2.1: codice articolo



Descrizione dettagliata Scenario 2.2: chiusura scontrino

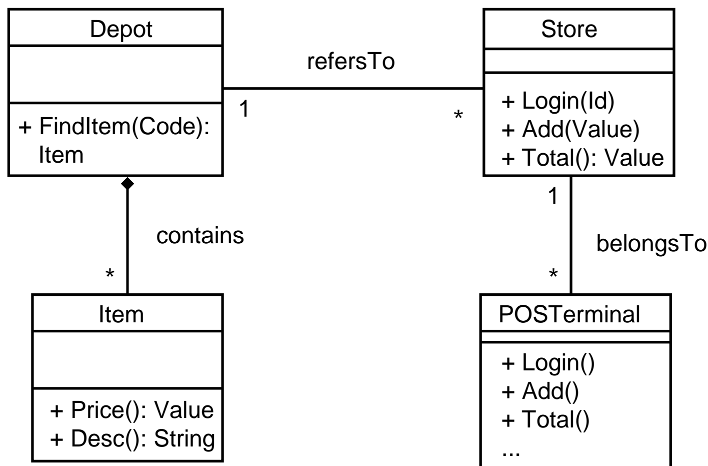


Descrizione dettagliata Scenario 3: totali negozio



- Questo è un “collaboration diagram” in UML. Fornisce le stesse informazioni di un sequence diagram

Entità rilevanti: diagramma delle classi



Descrizione informale delle classi

- Classe *Depot*
  - Classe che rappresenta il magazzino centrale (unico) della catena, che mantiene l'elenco degli articoli
  - Ha un metodo *FindItem* che effettua la ricerca di un articolo
- Classe *Item*
  - Rappresenta gli articoli venduti dalla catena (una istanza per ogni tipo di articolo)
  - Ha i metodi *Desc* e *Price*, ed altri che trascuriamo

...segue: Descrizione informale delle classi

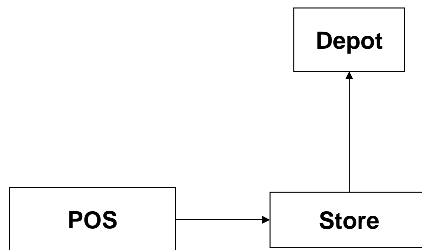
- Classe *Store*
  - Classe che rappresenta i negozi periferici della catena (un'istanza per ogni negozio)
  - Mantiene un elenco dei terminali collegati ed è in grado di sommare il totale delle vendite di tutti i POS
  - Ha un metodo *Login* per il collegamento iniziale dei terminali POS, ed i metodi *Add* e *Total* per memorizzare il totale di uno scontrino e restituire la somma degli scontrini

...segue: Descrizione informale delle classi

- Classe *POSTerminal*
  - Classe che rappresenta i terminali POS di ciascun negozio
  - Ha un metodo per ciascuna delle operazioni richiamabili dall'operatore del POS:
    - Login
    - Logout
    - AddItem
    - Total
    - POSTotal
    - StoreTotal

Functional View (class diagram semplificato)

- L'applicazione è scomposta funzionalmente in tre componenti o moduli. Gli "Item" fanno parte modulo Depot.
- E' un class diagram in cui si descrive il sistema ad un livello di astrazione diverso

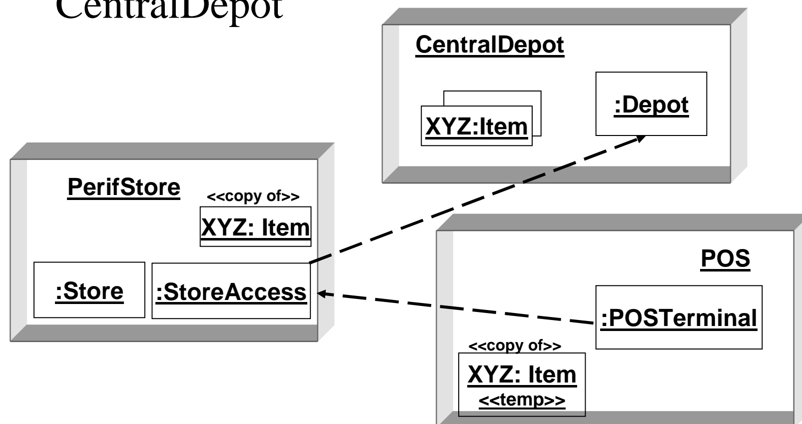


Definizione dell'architettura software

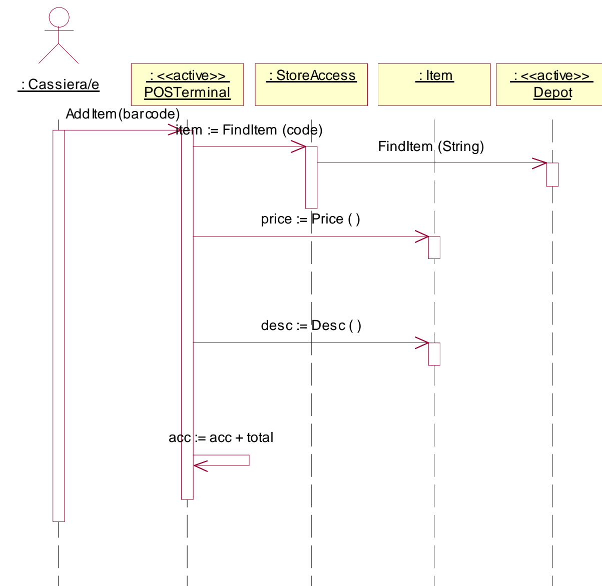
- Vincoli:
  - Architettura hardware (a disposizione o realizzabile)
  - Applicazioni esistenti con cui interfacciarsi
  - Caratteristiche delle reti (prestazioni)
- Alternative:
  - Tutta la logica nel client, il server è utilizzato come archivio (fat client)
  - Client/server con logica distribuita (two tier)
  - Architettura con intermediario (three-tier)
- Parametri di confronto:
  - Manutenibilità, evolvibilità
  - Prestazioni
  - ...

Runtime view (deployment diagram)

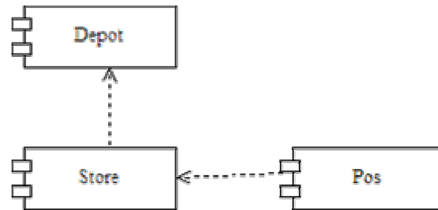
- StoreAccess: "proxy" fra POS e CentralDepot



Architettura 3-tier: Scenario 2.1



Unità di deployment (component diagram)

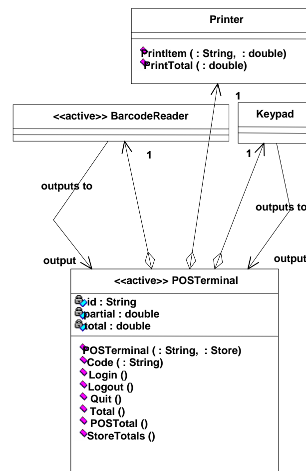


Progetto di dettaglio
Alcune decisioni di progetto

- Problema: interfacciamento con registratore di cassa
- Soluzione adottata:
  - implementare nel sistema software un “modello virtuale” di POS che sia una “copia” di quello reale (composto da una classe tastiera, una classe printer, …)
  - incapsulare nelle diverse classi la logica di comunicazione con il POS reale
- Vantaggi:
  - evolvibilità
  - testing: sostituendo alle classi che gestiscono la comunicazione dei “driver” che simulano il comportamento del POS possiamo testare separatamente la logica del sistema e la comunicazione col registratore di cassa

Progetto di dettaglio
Alcune decisioni di progetto

- Problema: gestire l’interazione fra le classi che costituiscono il modello del registratore di cassa
  - POSTerminal istanzia BarcodeReader e Keypad
  - BarcodeReader e Keypad devono invocare metodi di POSTerminal
- Soluzione adottata: approccio di tipo “observer” (semplificato)
  - soggetti: BarcodeReader e Keypad
  - osservatore: POSTerminal

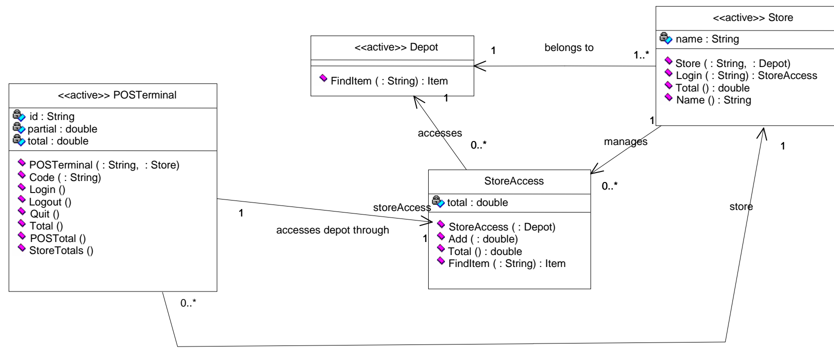


Progetto di dettaglio
Alcune decisioni di progetto

- Vantaggi della soluzione con “observer”:
  - i soggetti (BarcodeReader e Keypad) sono indipendenti da POSTerminal
  - è più semplice implementare un comportamento “reattivo”: POSTerminal non deve continuamente interrogare BarcodeReader e Keypad

Progetto di dettaglio
Alcune decisioni di progetto

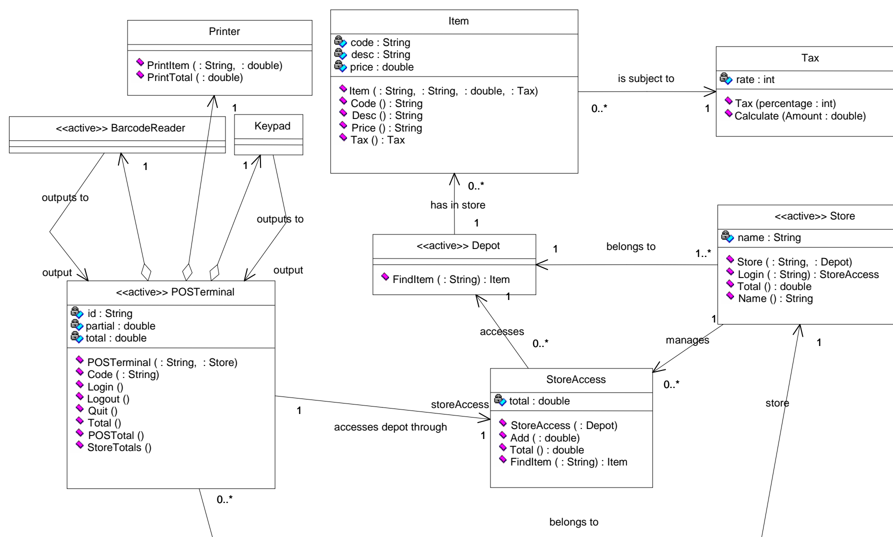
- Problema: interazione POSTerminal-Store-Depot
- Soluzione (già adottata a livello di architettura): StoreAccess come “proxy”



Progetto di dettaglio
Alcune decisioni di progetto

- Store = Factory, StoreAccess = Product, POSTerminal = Client
- Client crea Product attraverso un’interfaccia di Factory:
  - POSTerminal richiama il metodo Login di Store, provoca la creazione di un istanza di StoreAccess che viene restituita come risultato
- Pool di StoreAccess: Store usa un’unica istanza di StoreAccess per ciascun POSTerminal (sempre la stessa)

Class diagram progetto di dettaglio



Le classi dell’applicazione

- Classe *Depot*
  - Classe a cui appartiene l’oggetto corrispondente al magazzino centrale (unico) della catena, che mantiene l’elenco degli articoli
  - Ha un metodo pubblico (*FindItem*) che effettua la ricerca di un articolo
  - La classe è *<<active>>* e la sua (unica) istanza risiede su di un calcolatore diverso da quello su cui risiedono gli oggetti che invocano i suoi metodi (dovrà essere possibile accedere ai suoi metodi mediante chiamate “remote”)

Le classi dell’applicazione

- Classe *Store*
  - Classe a cui appartengono gli oggetti corrispondenti ai negozi periferici della catena (un’istanza per ogni negozio)
  - Implementa il metodo *Login* per il collegamento iniziale dei terminali POS
  - Mantiene un elenco dei terminali collegati ed è in grado di sommare il totale delle vendite di tutti i POS
  - All’atto del *login* restituisce all’oggetto chiamante un riferimento ad un oggetto di classe *StoreAccess*, che funge da *proxy* verso il magazzino centrale (*Depot*)
  - La classe è <<*active*>> e alcuni metodi sono invocati da oggetti remoti

Le classi dell’applicazione

- Classe *StoreAccess*
  - Esiste un oggetto *StoreAccess* per ogni terminale che ha effettuato il *login* presso un oggetto *Store*
  - L’oggetto viene riutilizzato in *login* seguenti (non viene distrutto quando il terminale si scollega)
  - L’oggetto *StoreAccess* mantiene il totale delle vendite del terminale per tutta la “vita” dell’oggetto *Store*
  - Esporta un metodo *FindItem* per la ricerca degli articoli, che implementa chiamando il metodo corrispondente dell’oggetto Depot

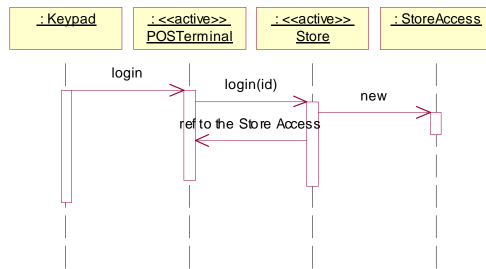
Le classi dell’applicazione

- Classe *BarcodeReader*
  - Simula il funzionamento di un lettore di codice a barre (legge il codice da tastiera ed invia il codice letto all’oggetto *POSTerminal* richiamandone il metodo *Code*)
- Classe *Printer*
  - Simula il funzionamento di una stampante ed esporta due metodi *PrintItem* e *PrintTotal*
- Classe *Keypad*
  - Simula la tastiera del terminale (implementato come insieme di 6 pulsanti su una finestra)
  - In corrispondenza della pressione di ciascun pulsante richiama un metodo dell’oggetto *POSTerminal*

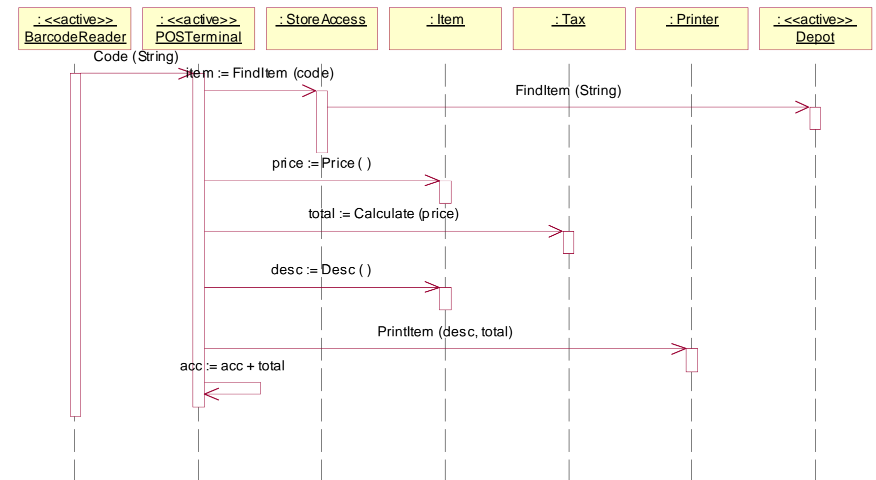
Le classi dell’applicazione

- Classe *POSTerminal*
  - Classe i cui oggetti corrispondono ai terminali POS di ciascun negozio
  - Metodo *Code*: richiamato all’atto della ricezione di un codice dal lettore di codici a barre, a sua volta richiama il metodo *FindItem* dell’oggetto *StoreAccess*, calcola il totale e stampa una riga dello scontrino
  - Metodo *Login*: richiamato dopo la pressione del tasto *Login* della tastiera, a sua volta richiama il metodo *Login* dell’oggetto *Store* e riceve in risposta un riferimento all’oggetto *StoreAccess*
  - Ecc. ecc. ecc.

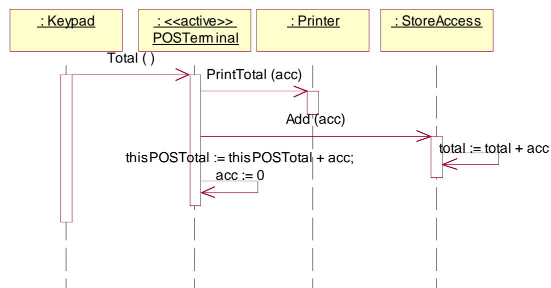
Seq. diagr. 1: login



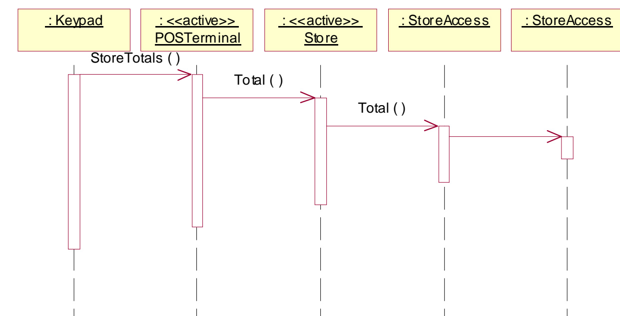
Seq. diagr. 2: codice articolo



Seq. diagr. 3: chiusura scontrino



Seq. diagr. 4: totali negozio



Costruzione di un'applicazione Java RMI

- Dichiarare le interfacce remote
- Implementare le classi remote
- Identificare le classi di parametri non remoti (passati per copia)
- Realizzare i client dei metodi remoti
- Realizzare i server
- Generare *stub* e *skeleton* e compilare

Dichiarare le interfacce "remote"

- L'accesso ai metodi di oggetti remoti deve avvenire attraverso interfacce remote
- Nell'esempio, i client accedono ai metodi della classe *Depot* attraverso l'interfaccia *DepotInterface*

```
import java.rmi.*;

public interface DepotInterface extends Remote
{
 public Item FindItem (String code)
   throws ItemNotFoundException,RemoteException;
}
```

Implementare le classi "remote"

- Nell'esempio, la classe *Depot* ha un metodo (*FindItem*) richiamabile in remoto

```
import java.rmi.*;
import java.rmi.server.*;
public class Depot extends UnicastRemoteObject
  implements DepotInterface {
...
 public Item FindItem (String code)
   throws ItemNotFoundException, RemoteException
   {
...
```

Diagramma delle classi